LAB DISTRIBUTED DATA ANALYTICS I
TUTORIAL IV

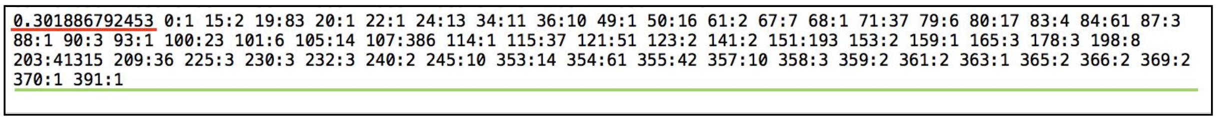
Juan Fernando Espinosa Reinoso
303158
Group 1: Monday Tutorial

The performance of the Linear Regression model has to be done in both **Dynamic features of Virusshare dataset**(**DFVD**) **& KDD Cup 1998.**

Note: For this tutorial I will present results for both tasks 1 and 2 for each dataset.

--- DFVD DATASET ---

The DFVD dataset folder contains several inner documents in which each of them contains a sparse representation of the virus with the respective classification as a number.



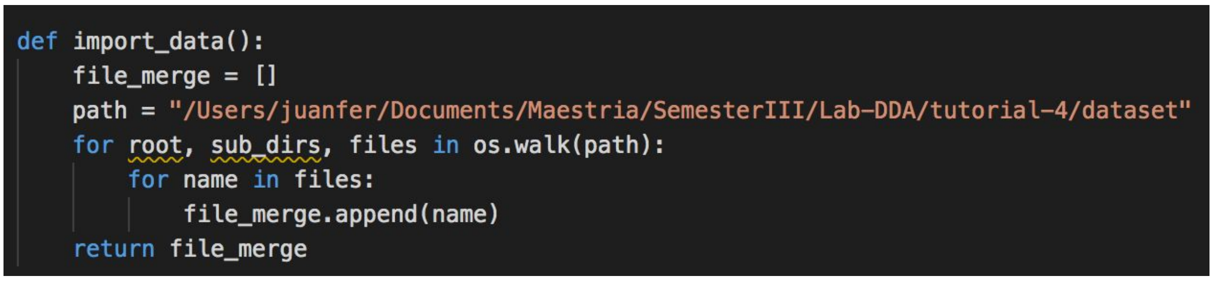
```
0.301886792453 0:1 15:2 19:83 20:1 22:1 24:13 34:11 36:10 49:1 50:16 61:2 67:7 68:1 71:37 79:6 80:17 83:4 84:61 87:3
88:1 90:3 93:1 100:23 101:6 105:14 107:386 114:1 115:37 121:51 123:2 141:2 151:193 153:2 159:1 165:3 178:3 198:8
203:41315 209:36 225:3 230:3 232:3 240:2 245:10 353:14 354:61 355:42 357:10 358:3 359:2 361:2 363:1 365:2 366:2 369:2
370:1 391:1
```

🟥 Y
🟩 X values

--- DATA PREPROCESSING ---

1. Import data function

The import the data by browsing in the OS and iterate through the root path to get the files and append them in a list of documents.

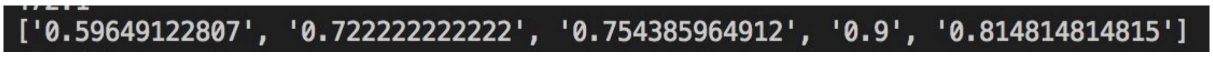


```python
def import_data():
    file_merge = []
    path = "/Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-4/dataset"
    for root, sub_dirs, files in os.walk(path):
        for name in files:
            file_merge.append(name)
    return file_merge
```

Secondly, as each virus has several features (ie: 15:2, 36:10) and one value (ie: 0.301886) I encountered it necessary to create a sparse matrix from which the regression will be applied.

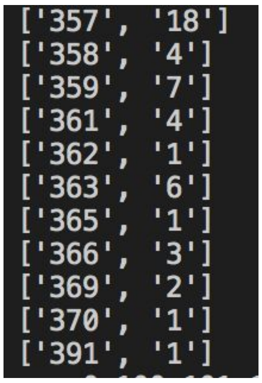
2. Sparse Matrix Function

Initialization of two empty arrays for X and Y takes place. Iterate through all the documents. Initialization of an empty dictionary. Then, splitting each datapoint inside a document and take the value in position 0 as Y.
Example:



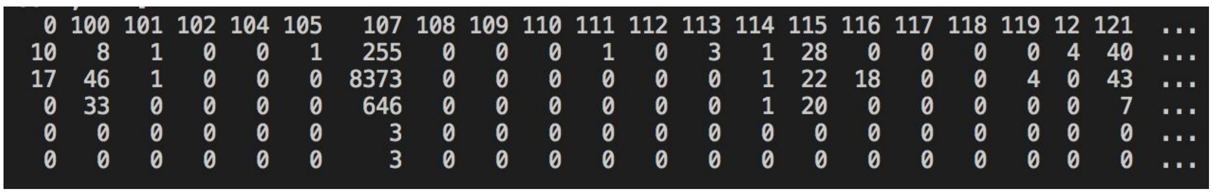
```
['0.59649122807', '0.722222222222', '0.754385964912', '0.9', '0.814814814815']
```

Next, adding to the empty dictionary each key and value point after splitting the values from position 1 to the end by “:”. The results are as follow:



```
['357', '18']
['358', '4']
['359', '7']
['361', '4']
['362', '1']
['363', '6']
['365', '1']
['366', '3']
['369', '2']
['370', '1']
['391', '1']
```

Finally, add the list that group all the dictionaries of viruses inside each document and transform it into a sparse matrix.



```
  0 100 101 102 104 105   107 108 109 110 111 112 113 114 115 116 117 118 119 12 121  ...
 10   8   1   0   0   1   255   0   0   0   1   0   3   1  28   0   0   0   0  4  40  ...
 17  46   1   0   0   0  8373   0   0   0   0   0   0   1  22  18   0   0   4  0  43  ...
  0  33   0   0   0   0   646   0   0   0   0   0   0   1  20   0   0   0   0  0   7  ...
  0   0   0   0   0   0     3   0   0   0   0   0   0   0   0   0   0   0   0  0   0  ...
  0   0   0   0   0   0     3   0   0   0   0   0   0   0   0   0   0   0   0  0   0  ...
```

The next step is to normalize the data, split the data into Train/Test, and finally initialize MPI.

--- MPI initialization ---

For the purpose of this model, the dataset will be splitted in equal parts and send to all workers by using collaborative communication. The weights, which are going to be updated in each worker, are going to be broadcasted to all the workers, gathered them back, average them and repeat the process.

Worker 0

- Worker 0 first imports the files and append in a list. Iterate through it, split all the types of virus inside a file and create a sparse matrix.
- Randomly initialize the Betas

MPI world

- Broadcast the Betas to all workers.
- Split data among workers
- Initialize a for loops.
    - Calculate the predicted y_hat for all workers.
    - Update the values of betas
    - By using comm.reduce get the sum of all the betas across workers.

worker 0

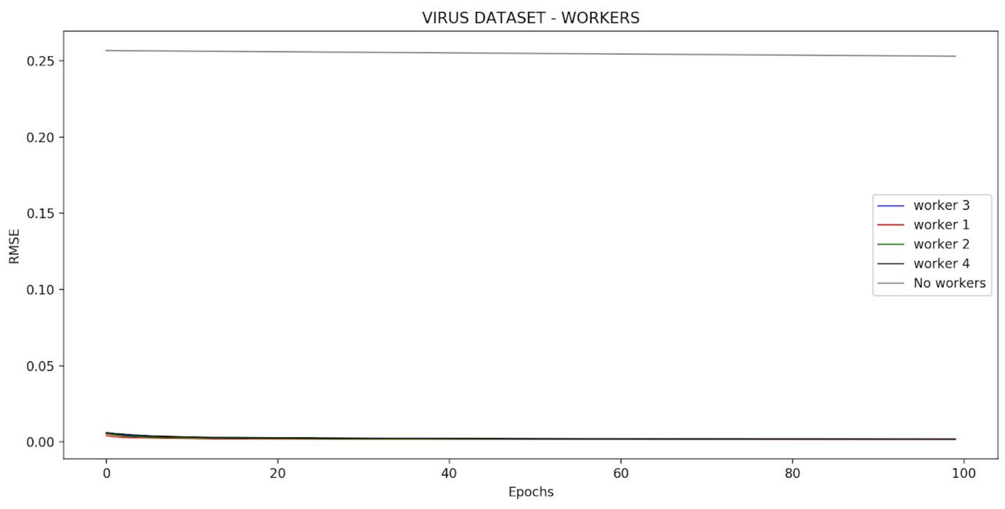
- Collects the betas and divide by number of workers to get the mean of the weights.
- Calculate the loss for the training set using the collected betas.
- Calculate y_hat with test set and the loss function.
- Append the values in an array and save them as csv for future plotting.

--- RESULTS ---

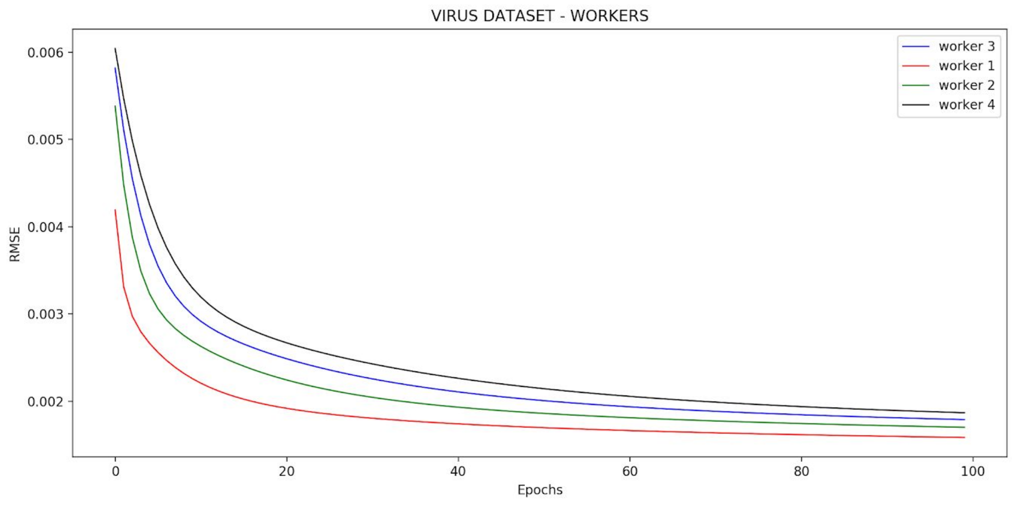
Table 1: results workers per iterations/runnings.

2 Runs / WORKERS	**Sequential**	**1**	**2**	**3**	**4**
100, LR = 0.00001	13.618560 075759888	17.078553	16.55127	18.026999	19.850308
100, LR = 0.00001	14.234598	14.280427	16.292938	17.874845	19.355867

Plot test convergence - different workers and sequential model

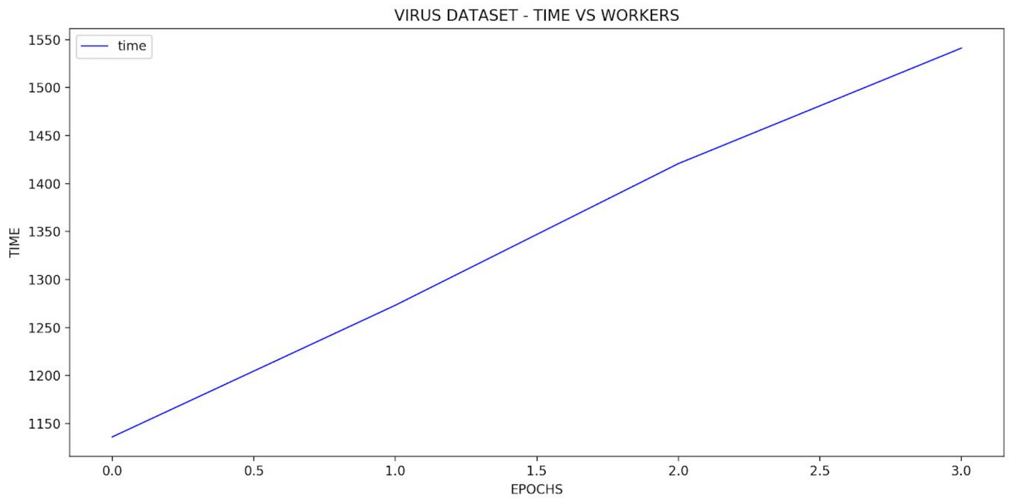


There is a huge difference in convergence between the MPI solution vs the sequential one. The Sequential model does not reach convergence with the LR used and the iterations. (See gray line in 0.25 RMSE)



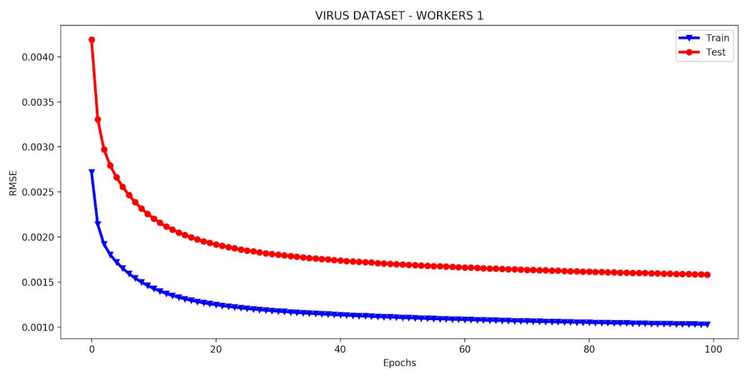
In table 1 and in the plots we can see that running with one worker gets better results than with 2, except in the first test running. In the plot worker 1 reaches convergence faster than the others. It is a rare event because 2 workers are known to get better improvement than only 1. Nonetheless, since my laptop only has 2 cores, workers 3 and 4 have a lower performance than the beginning two.

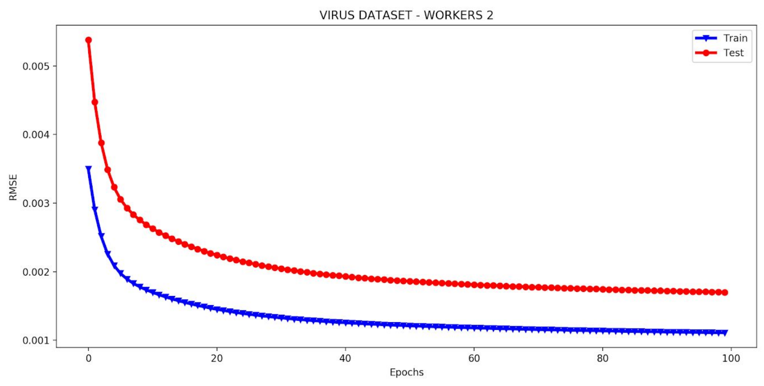
Time vs Epochs

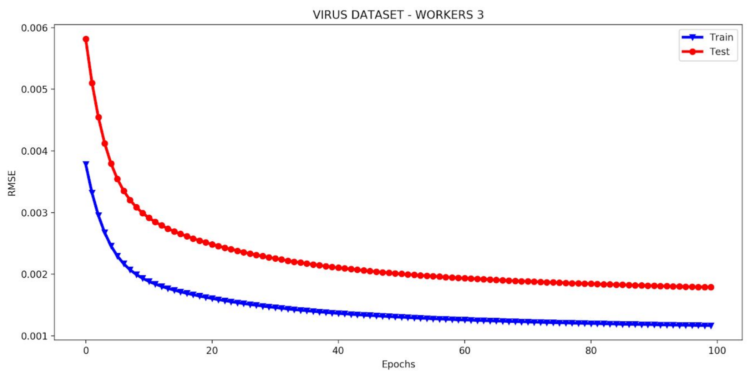


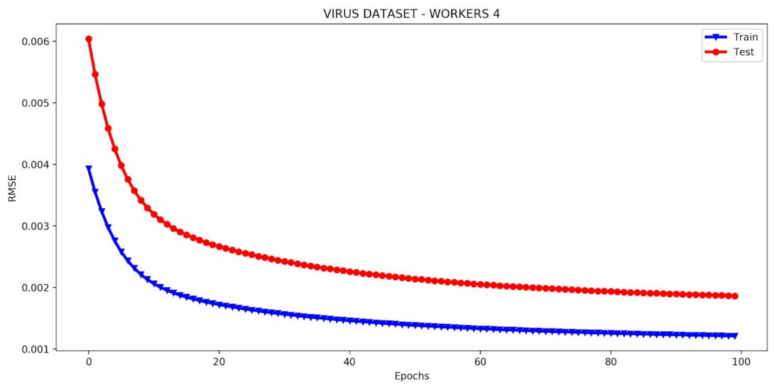
As presented in the table 1 the time increases in each worker. As we can see after worker 2 the increase is not as prominent considering a specific run. Those values vary due to the randomness of the training split.

Convergence of the algorithm with different workers (Train / Test)






VIRUS DATASET - WORKERS 3
RMSE
Epochs
Train
Test
0.006
0.005
0.004
0.003
0.002
0.001
0
20
40
60
80
100


VIRUS DATASET - WORKERS 4
RMSE
Epochs
Train
Test
0.006
0.005
0.004
0.003
0.002
0.001
0
20
40
60
80
100

--- KDD1998 DATASET ---

The DFVD dataset folder after downloading is a “txt” file where it first has the columns and then a sparse representation of the values.

--- DATA PREPROCESSING ---

The data preprocessing requires to read the file with the pandas csv read option which returned a dataframe of data. I selected the most influential columns to work with (Pearson correlation) and finally, apply get_dummies to transform object, strings texts into numerical values.

--- MPI initialization ---

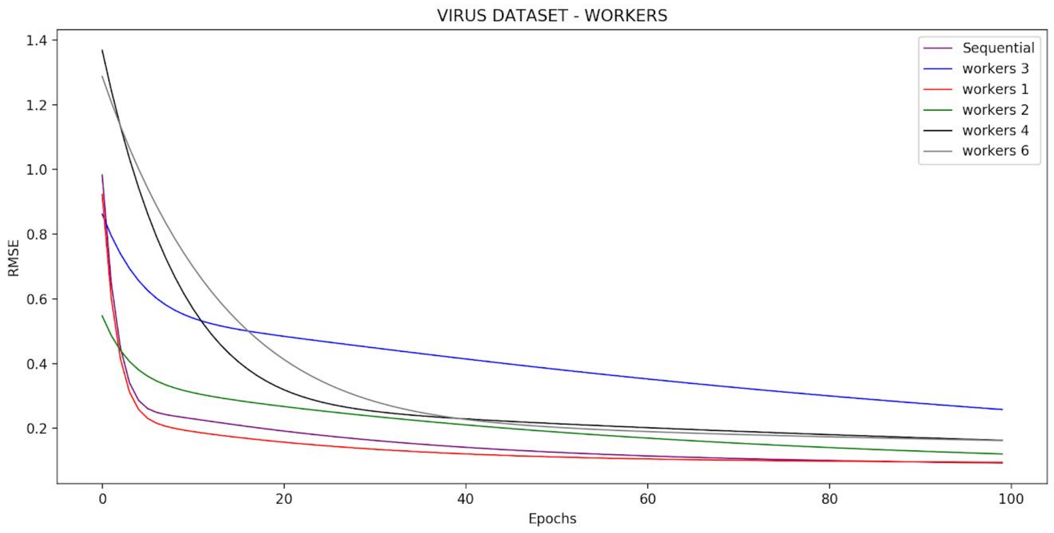
The model works exactly the same as explained in the first exercise by using collaborative communication. Therefore, I will not dive in in the explanation but show the results for the model.

--- RESULTS ---

Table 2: results workers per iterations/runnings.

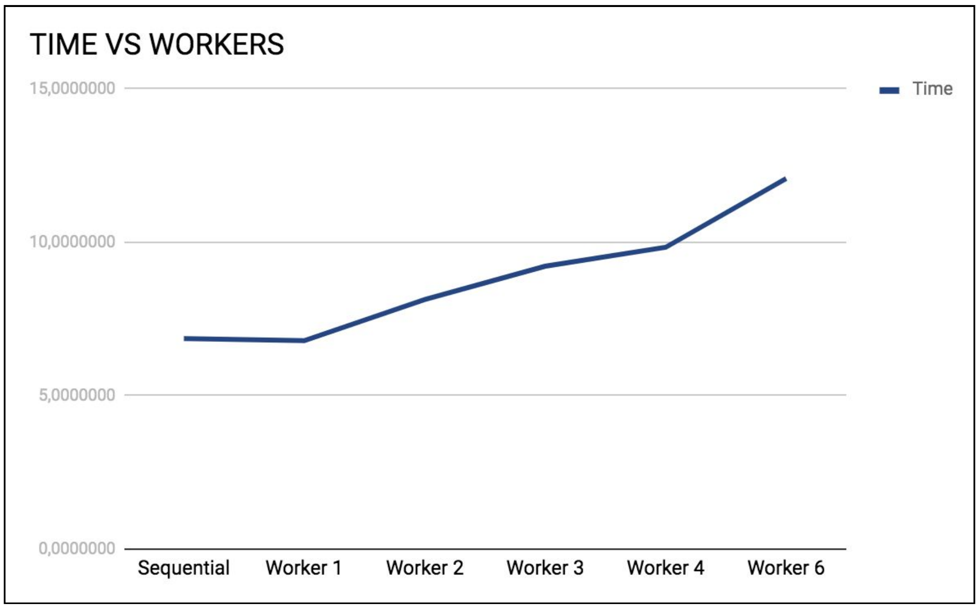
2 Runs / WORKERS	**Sequential**	**1**	**2**	**3**	**4**	**6**
iter = 100, LR = 0.0000000001	6.86221075	6.791913	8.136818	9.220147	9.836011	12.079241

Plot test convergence - different workers and sequential model



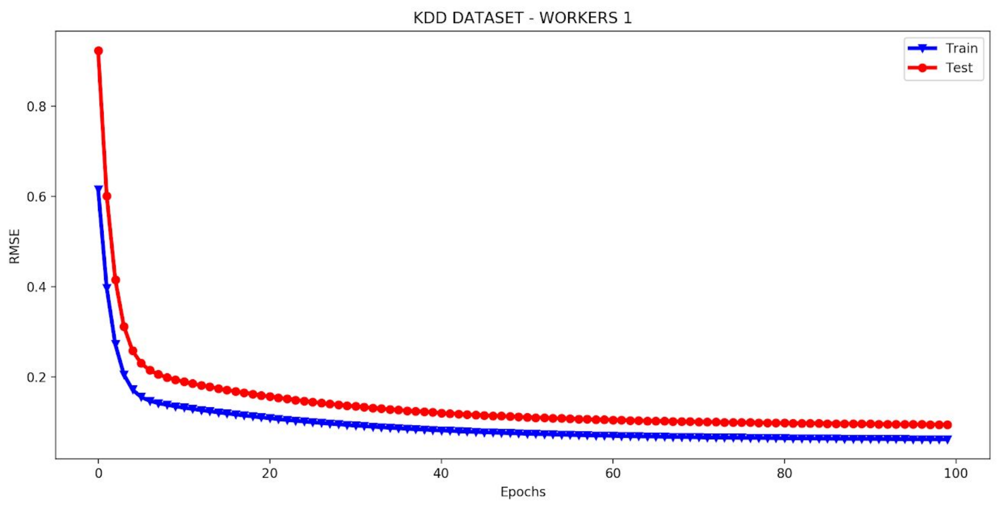
As we can appreciate in the above graph the sequential model has a better performance in comparison to a model with 2 or more workers. Moreover, worker 3 performs worst in comparison to the other workers. Workers 4 and 6 got to the same convergence point but to workers 6 it takes more iterations to converge.

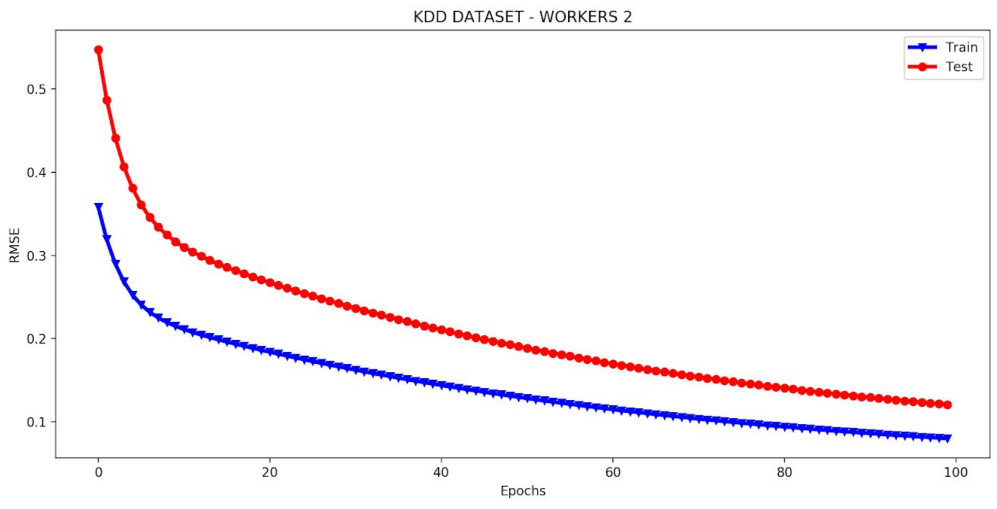
Time vs Epochs

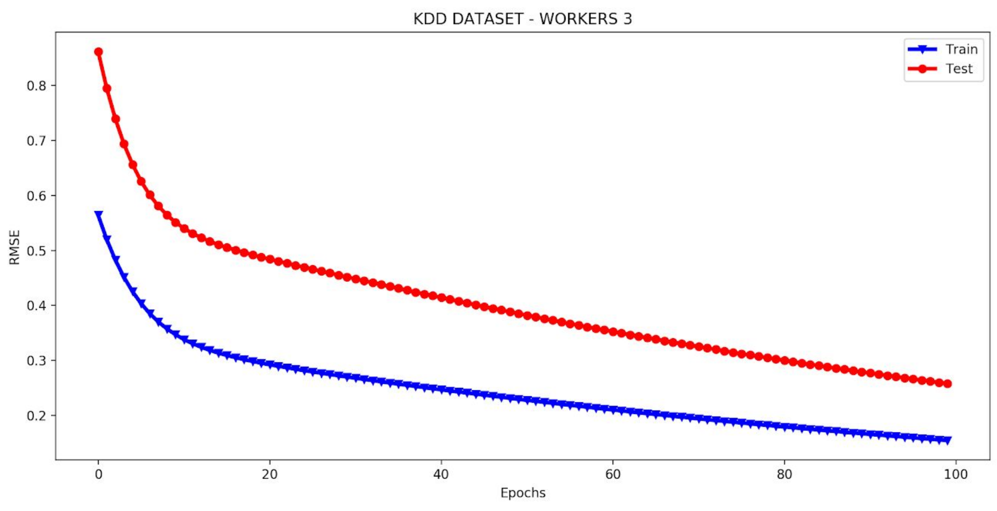


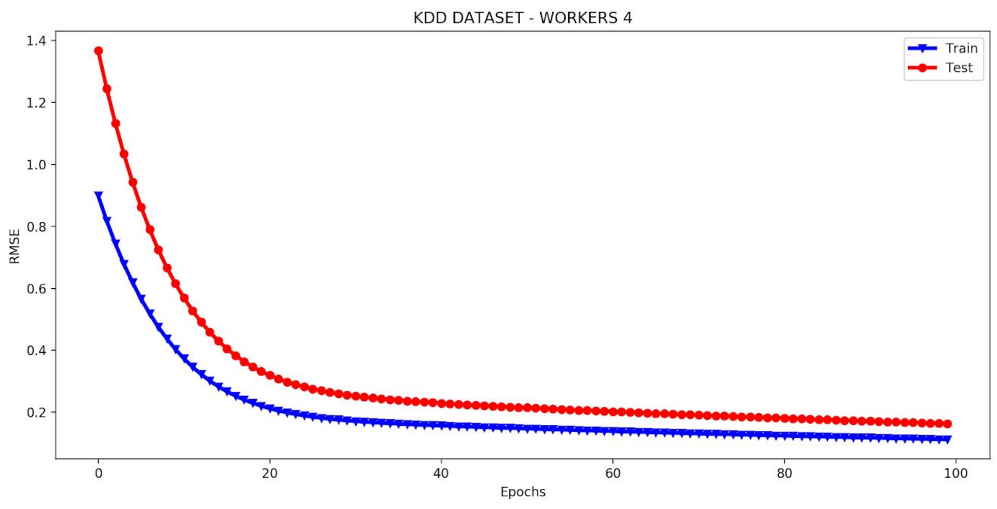
Similar to the behavior experienced in the first exercise, the sequential and the algorithm with 1 worker got the minimum time of execution. Then, it is increasing the time as we increase the number of workers.

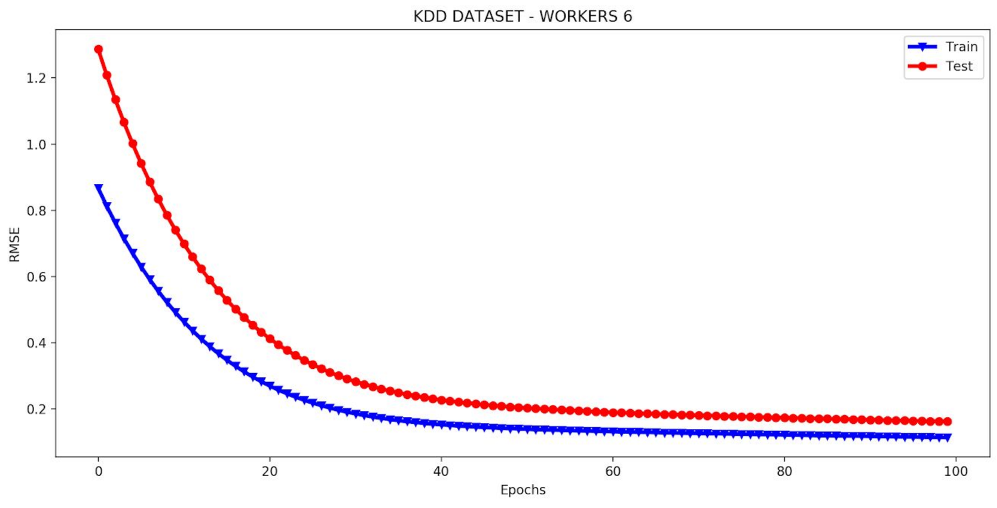
Convergence of the algorithm with different workers (Train / Test)






KDD DATASET - WORKERS 3
Train
Test
RMSE
Epochs
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100


KDD DATASET - WORKERS 4
Train
Test
RMSE
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100


KDD DATASET - WORKERS 6
Train
Test
RMSE
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100

References:

[1]
https://necromuralist.github.io/neural_networks/posts/normalizing-with-numpy/#:~:text=Normalize%20Rows,See%20the%20numpy%20documentation.