Pocket Algorithm Report

Goal:

Implement the Pocket Algorithm by initializing the weight vector using 2 different methods: first data point and linear regression.

In General:

In general, the Pocket algorithm is a binary classification algorithm that iteratively updates its hypothesis to minimize classification errors. It does this by adopting a "pocket" where the “best” weights are stored each time (iteration). For each iteration during the training phase, the in-sample error (Ein) will be updated by a smaller error until we reach the “best” or the smallest error. To achieve this we first map to +1|-1 the linear combination of the feature matrix “X” and the weight vector. This is called binary classification. The classification (prediction) may contain mistakes in classification hence by randomly picking a misclassified example and through weights+=X[random_index] * y[random_index] we aim to push the classification sign towards the real sign (y[random_index]). If y[random_index] is +1, through the multiplication the sign of X[random_index is changed and pushed to increase the weight components and vice versa. The mistake will count as part of our in-sample error and in theory or from what I observed so far in my experiments the error will get smaller. If the in-sample error cannot improve it means that we have stored already the best weights in our pocket and will use them for our testing.

Method 1 - First data point :

The weights vector is initialized using the first data point taken from our training set. The first advantage here is that we do not start our weights from 0 hence will take shorter to find optimal weights hence fewer errors. Second, if the first data point is a good representation of its class these initial weights will be very close to our final weights will help converge way faster.

Method 2 - Linear Regression:

By initializing our weights using linear regression we aim to start with an optimal weight vector. The linear regression dot products the pseudoinverse of the matrix X with the target y (class labels). This dot product finds the weights that minimize the squared errors between our prediction and true labels. Doing so will help the algorithm to converge faster (in theory).

Observations:
(see graphs below)

- **Case 1 - Decent Amount of Iterations (1200)**

In this first run I present the following diagrams as part f my observations. Fig # 1 / 3 and 2/ 3 are the average performance for each method in all 5 random splits by the number of iterations. It means that I take the average values for all splits and represent its performance in the determined iteration. In this specific case the trining is conducted for 1200 iterations and the training-testing split ratio is .8-.2. As shown in the graphs it seems that Ein for both methods is almost similar. Both methods provide a good training in this case. However what I found surprising is that Eout in Method 1 outperform the Eout in Method 2. The performance of Method 1indicates that generilizes unseen data way better than Method 2. This might be due to overfitting. (I hope I am not abusing with terminology). Changing the traing-testing split to .7-.3. does not affect the general behaiviour.

- **Case 2 - A lot of Iterations(10000)**

When the training is run for 10000 iteration the biggest change noticeable is that the Ein gets lower for both methods. The gap Eout-Ein in Method 1 stays the same however this gap increases in Method 2 since there are no improvement when yielding the Eout. In general I would think of running less iteration even given the fact that Ein decreases but there is no major improvement in the gap Ein-Eout and there is an added cost of computation.

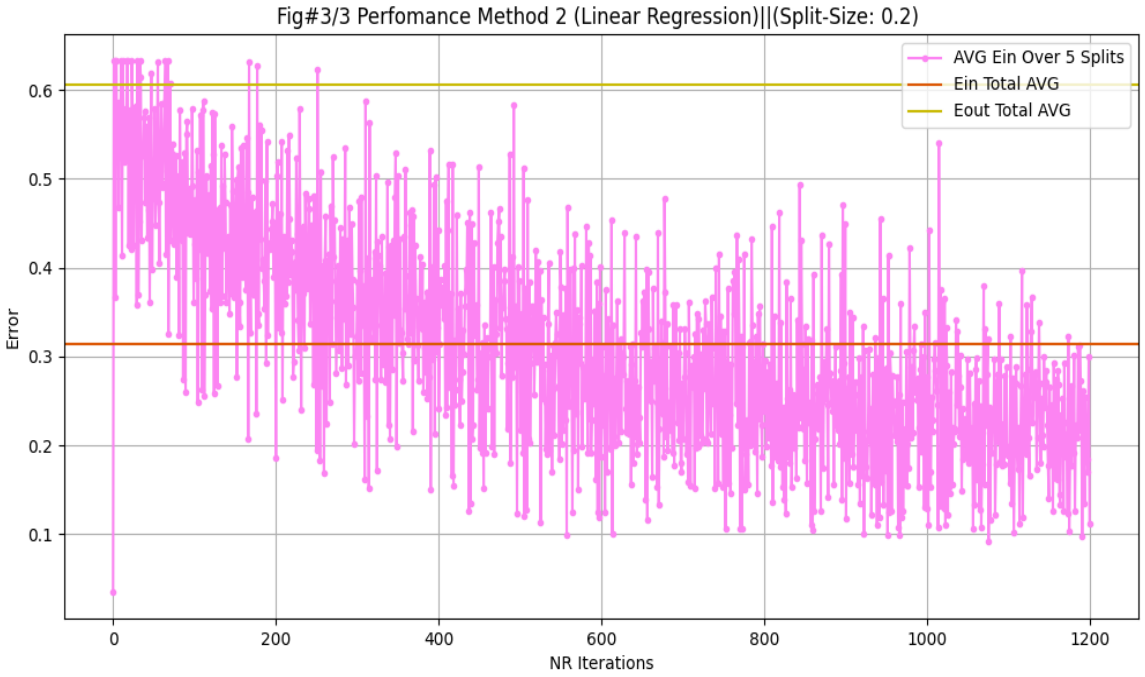
- **Case 3- Few Iterations(10000)**

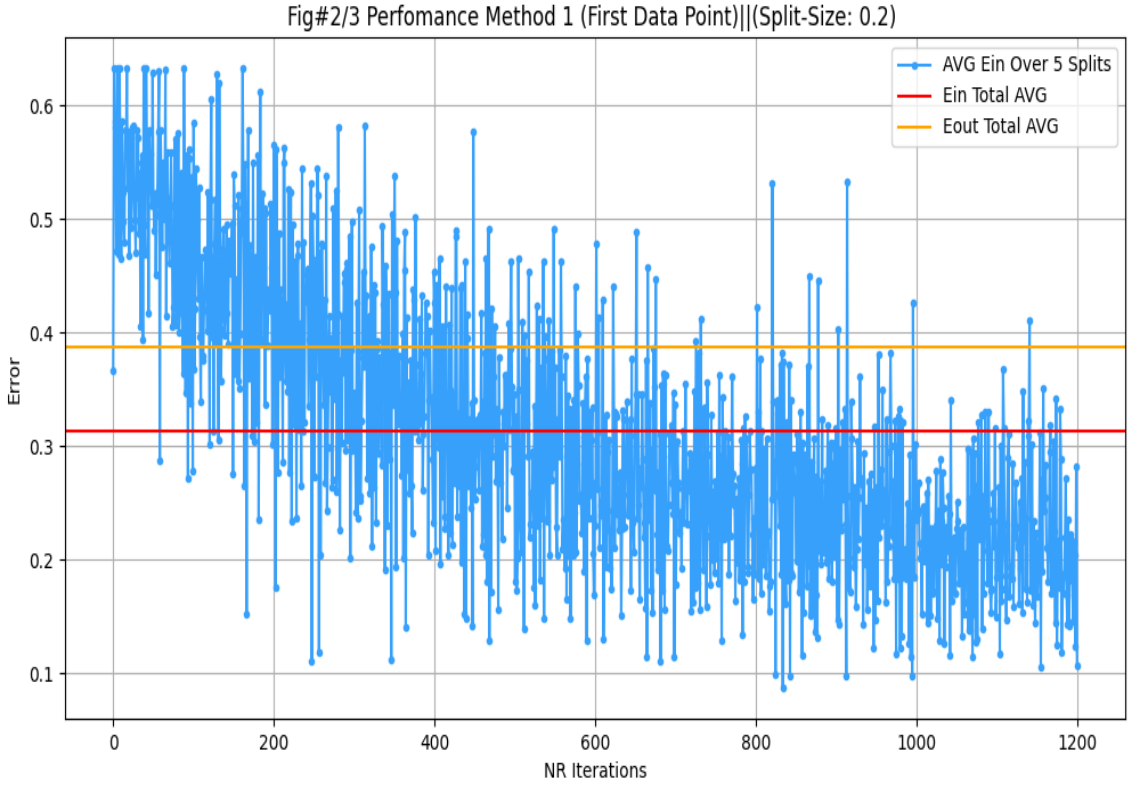
A big “improvement” in the gap Ein-Eout is registered when the algorithm is run for 300 iterations. In general the Ein gets worse but in Method 1 the Eout and Ein get significantly close. In Method 2 the gap is improved as well. Both algorithms improved on their generalization of unseen data. An interesting fact is that when the split is changed from .7-.3 to .8-.2 the gap is almost the same for both method.

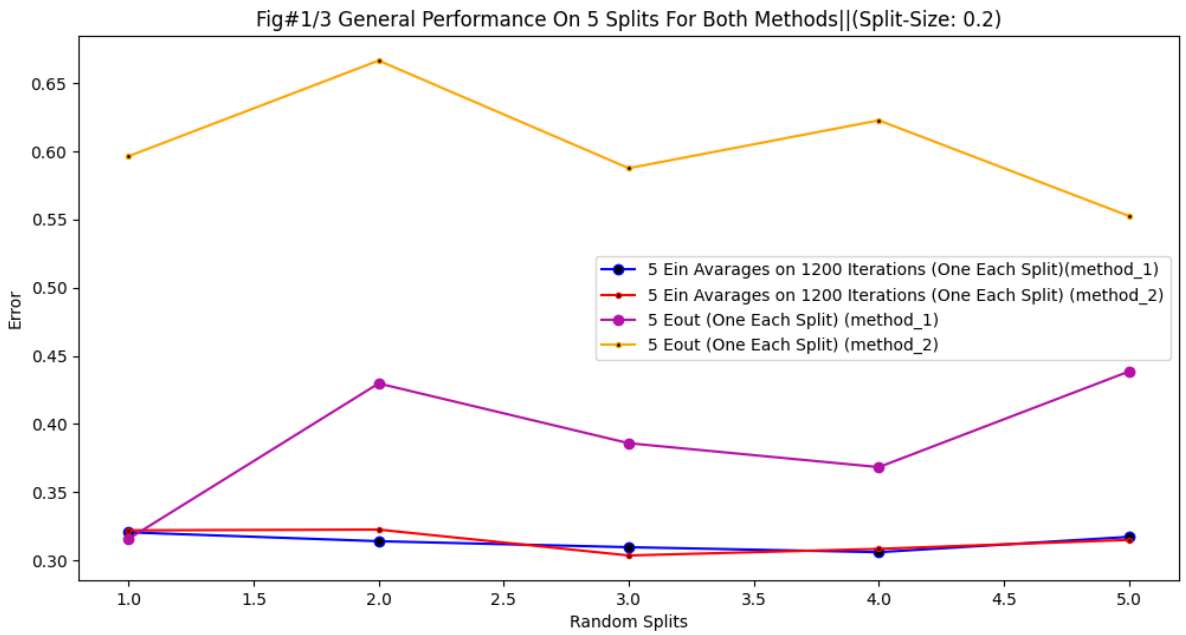
In Conclusion:

Given my observation i believe that in this particular case and this given dataset Method 1 (First Data Point) proved to be a more efficient implementation of the Pocket Algorithm. It generelizes better the unseen data. Also, even though an bigger amount of iterations will result in lower Ein it does not result in better generalization. The optimal settings in my case was at about 300 iteration with a .7-.3split(Method 1) or .8-.2 (Both Methods). This proves that testing on the engineer side is crucial in dealing with different cases and datasets.

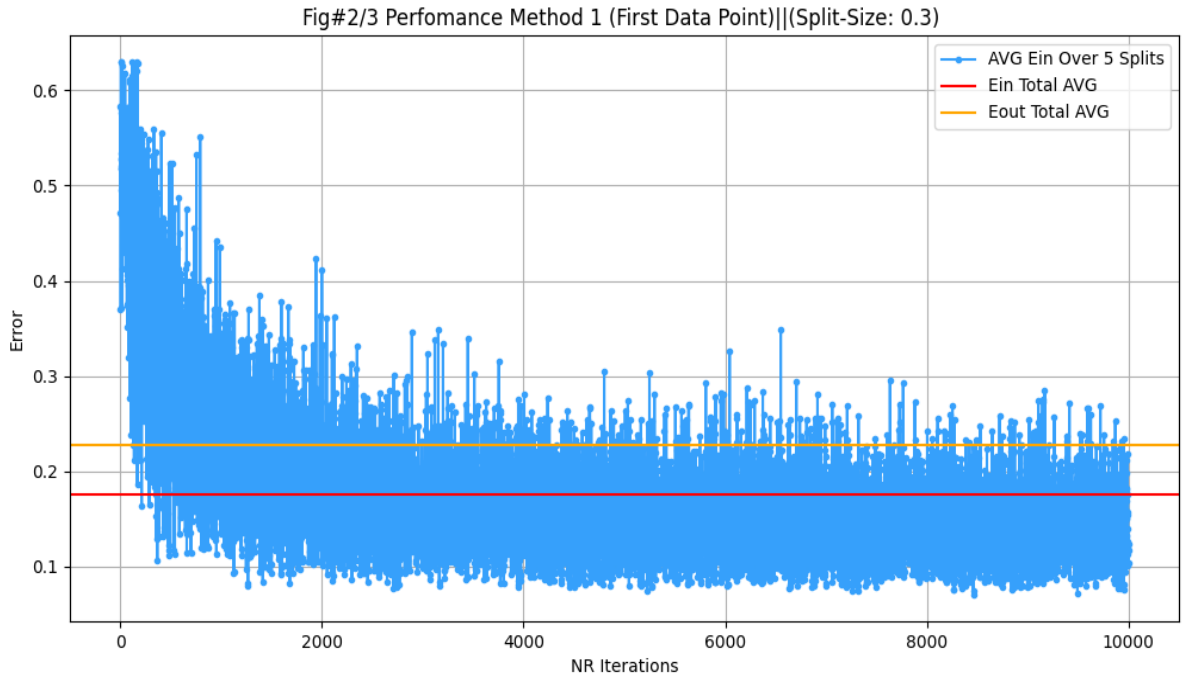
- **Case 1**

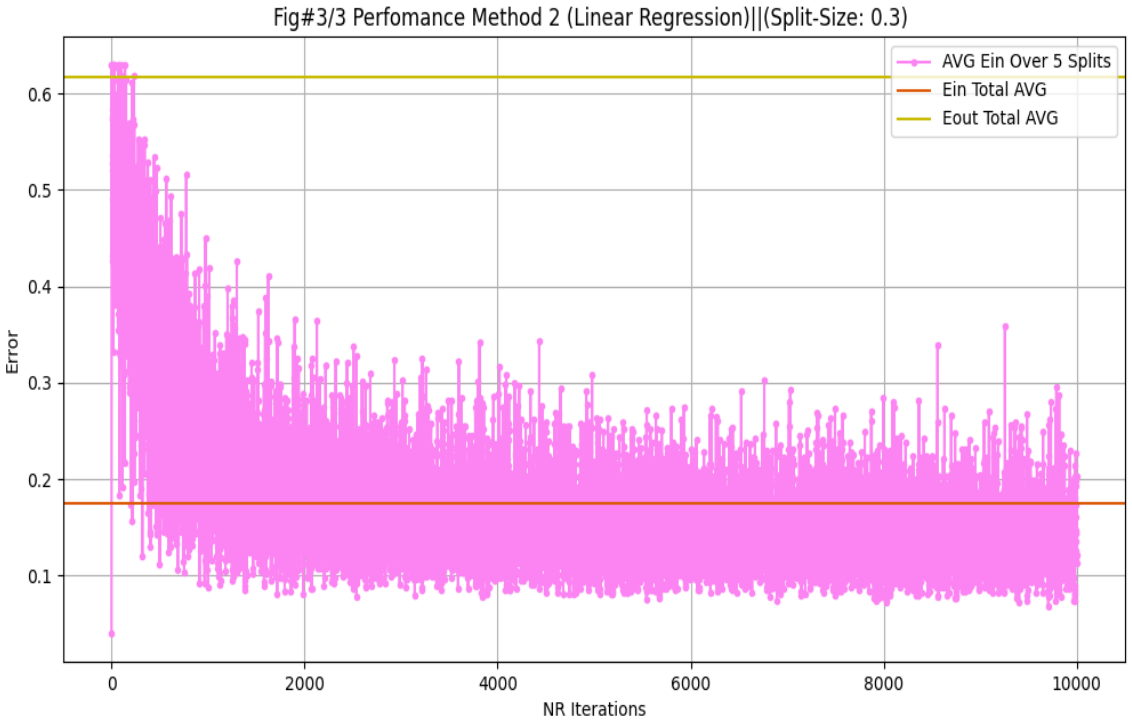

Fig#3/3 Perfomance Method 2 (Linear Regression)||(Split-Size: 0.2)
AVG Ein Over 5 Splits
Ein Total AVG
Eout Total AVG
Error
NR Iterations
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000
1200

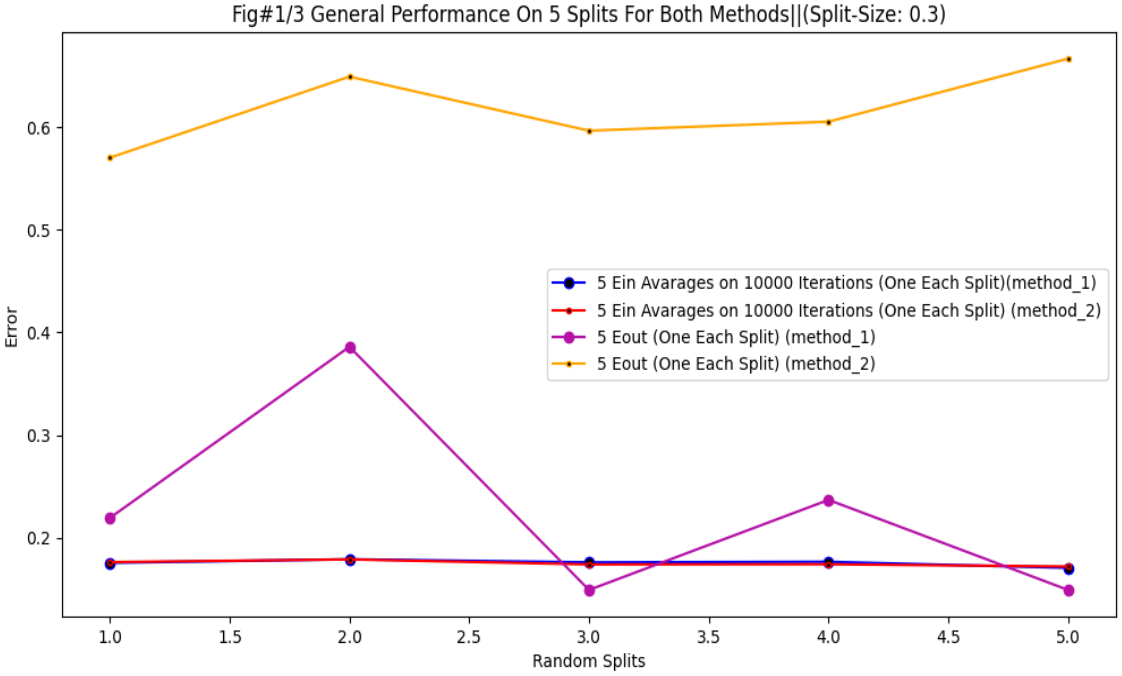

Fig#2/3 Perfomance Method 1 (First Data Point)||(Split-Size: 0.2)
AVG Ein Over 5 Splits
Ein Total AVG
Eout Total AVG
Error
NR Iterations
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000
1200


Fig#1/3 General Performance On 5 Splits For Both Methods||(Split-Size: 0.2)
Error
Random Splits
5 Ein Avarages on 1200 Iterations (One Each Split)(method_1)
5 Ein Avarages on 1200 Iterations (One Each Split) (method_2)
5 Eout (One Each Split) (method_1)
5 Eout (One Each Split) (method_2)

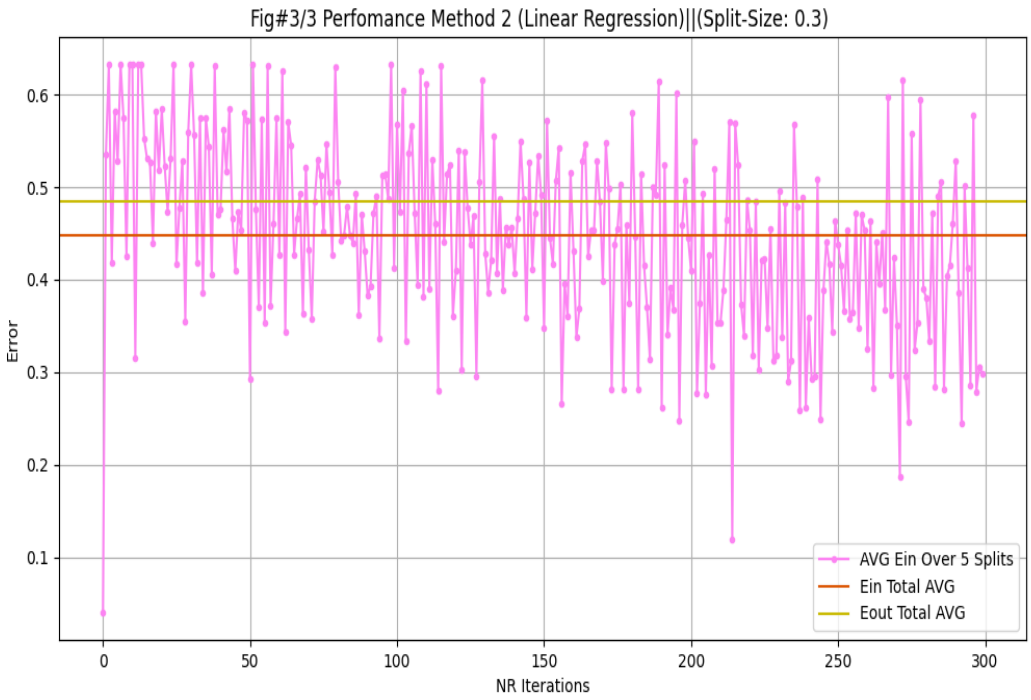
- **Case 2**

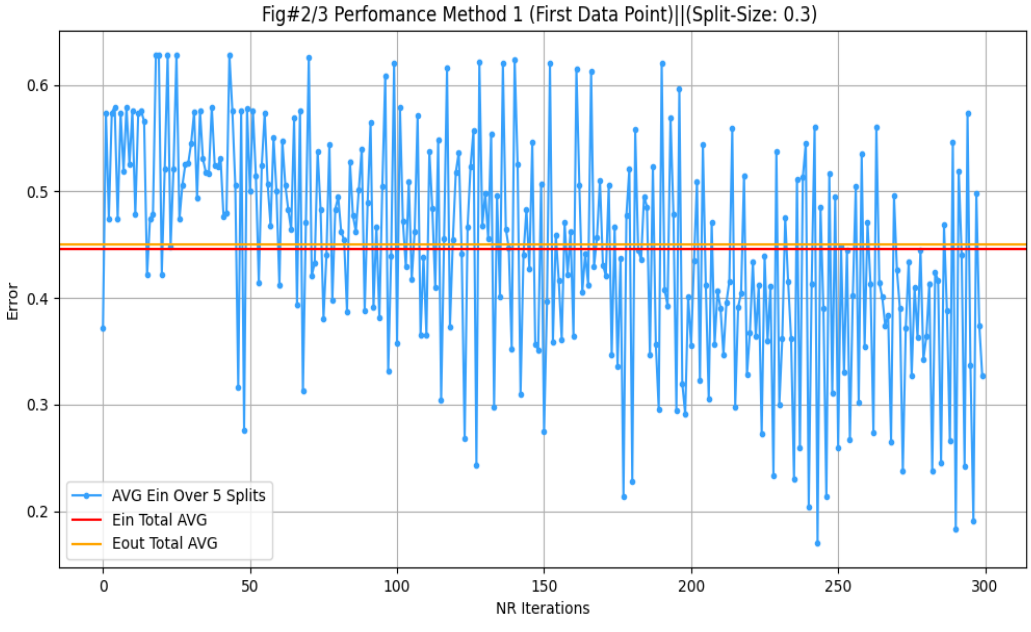

Fig#2/3 Perfomance Method 1 (First Data Point)||(Split-Size: 0.3)
Error
NR Iterations
AVG Ein Over 5 Splits
Ein Total AVG
Eout Total AVG

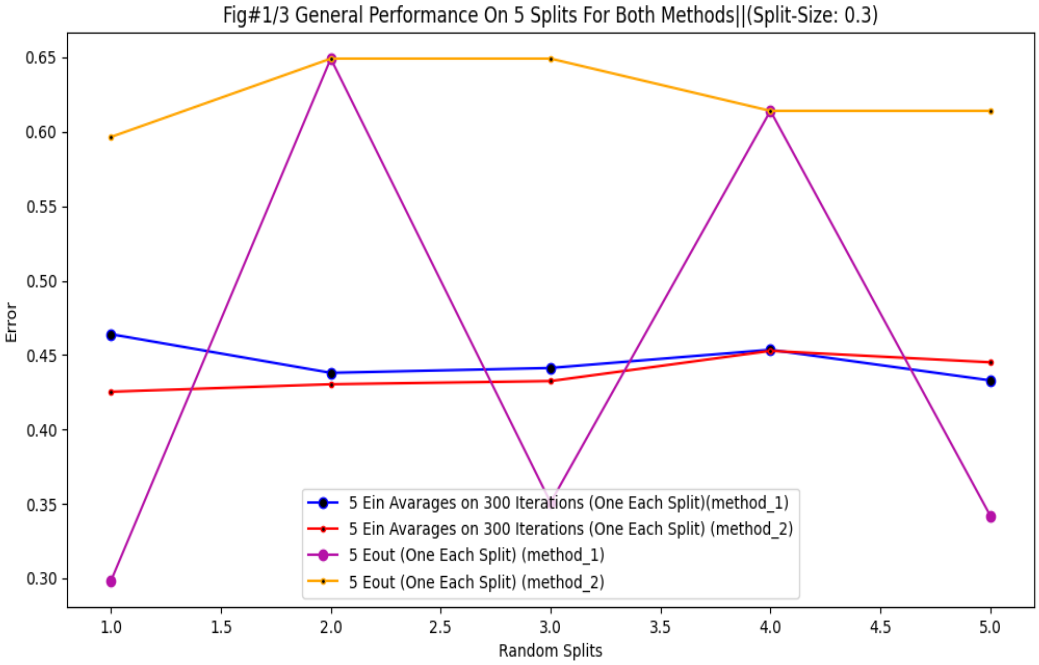

Fig#3/3 Perfomance Method 2 (Linear Regression)||(Split-Size: 0.3)
AVG Ein Over 5 Splits
Ein Total AVG
Eout Total AVG
Error
NR Iterations
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

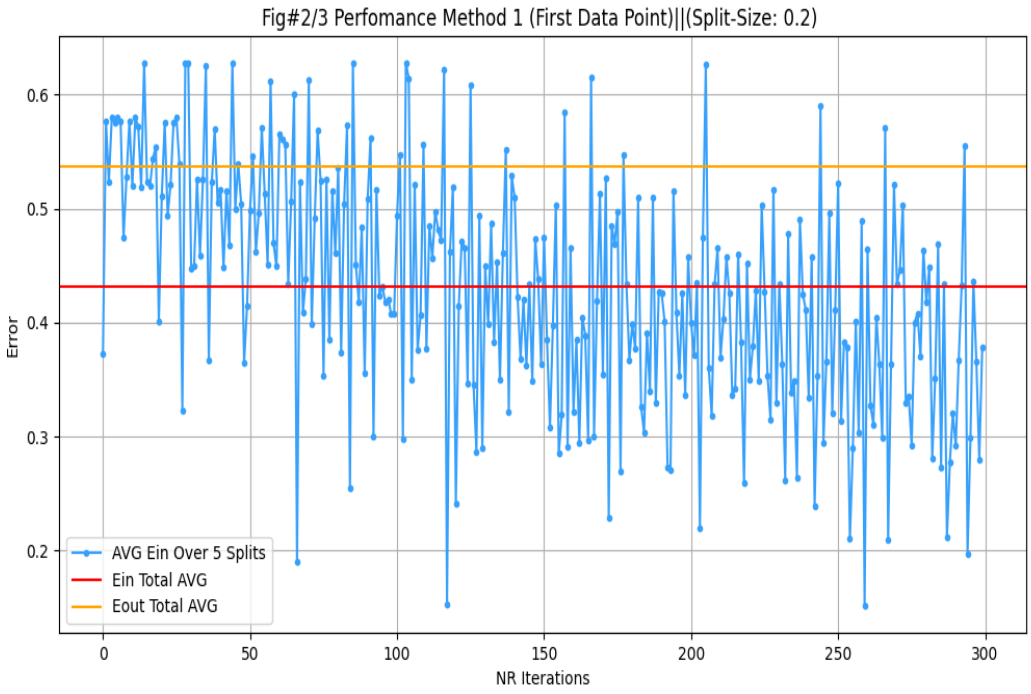

Fig#1/3 General Performance On 5 Splits For Both Methods||(Split-Size: 0.3)
5 Ein Avarages on 10000 Iterations (One Each Split)(method_1)
5 Ein Avarages on 10000 Iterations (One Each Split) (method_2)
5 Eout (One Each Split) (method_1)
5 Eout (One Each Split) (method_2)
Error
Random Splits
0.2
0.3
0.4
0.5
0.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

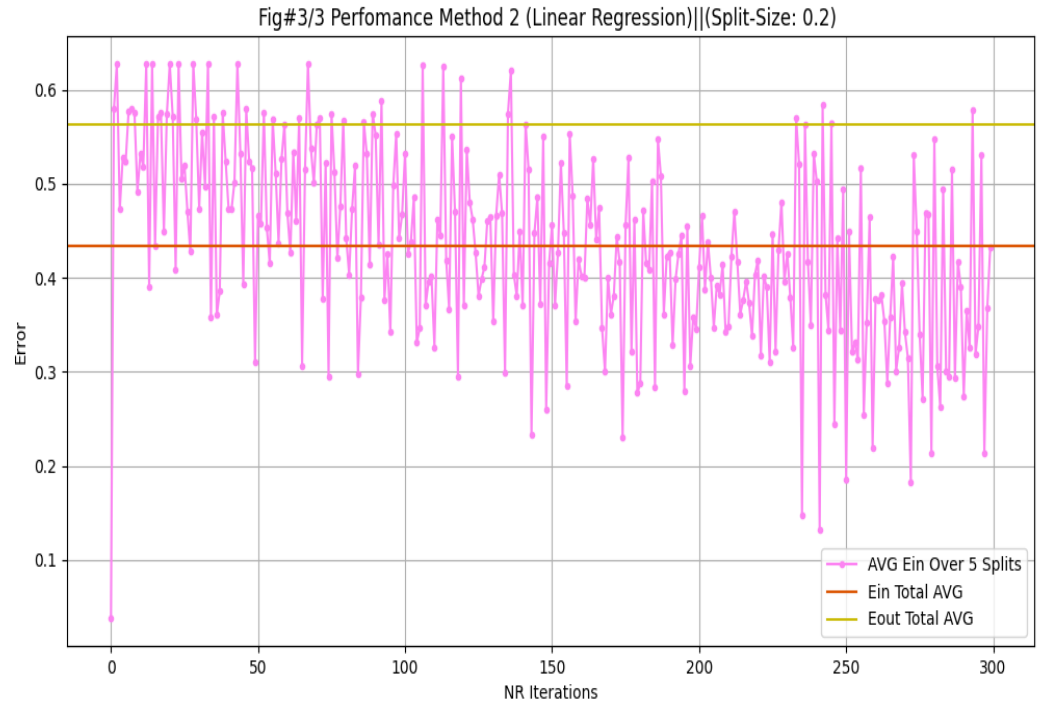
- **Case 3 (0.3 and .0.2 splits)**

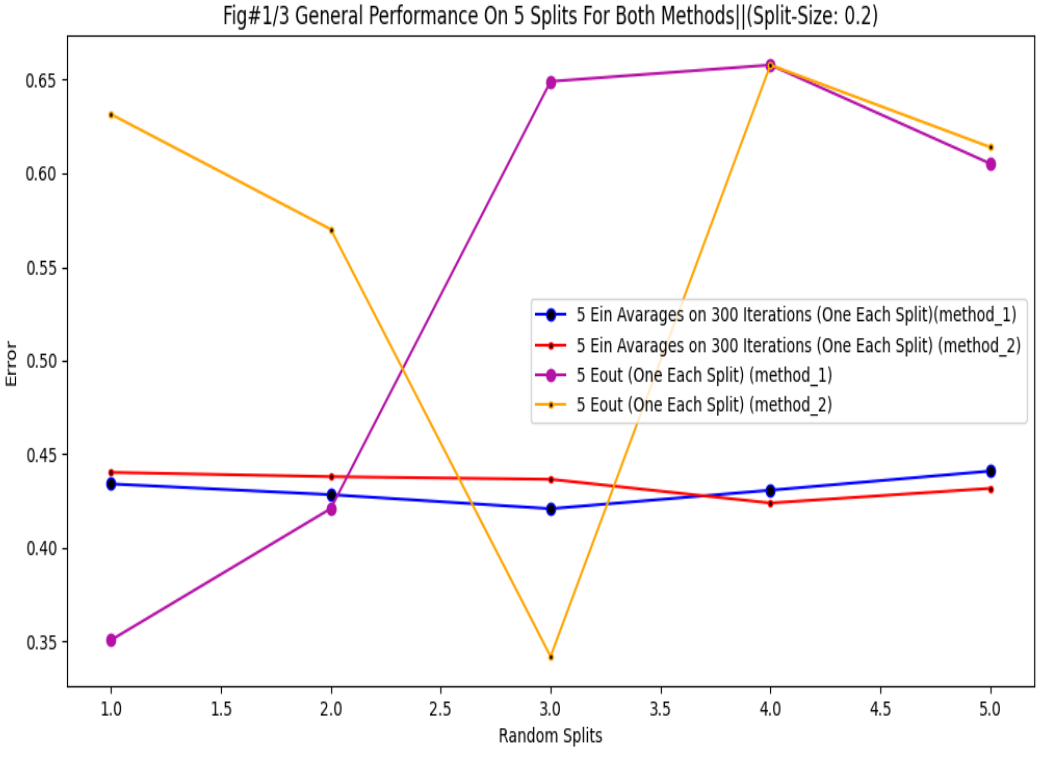





Fig#1/3 General Performance On 5 Splits For Both Methods||(Split-Size: 0.3)
Error
Random Splits
5 Ein Avarages on 300 Iterations (One Each Split)(method_1)
5 Ein Avarages on 300 Iterations (One Each Split) (method_2)
5 Eout (One Each Split) (method_1)
5 Eout (One Each Split) (method_2)


Fig#2/3 Perfomance Method 1 (First Data Point)||(Split-Size: 0.2)
Error
NR Iterations
AVG Ein Over 5 Splits
Ein Total AVG
Eout Total AVG


Fig#3/3 Perfomance Method 2 (Linear Regression)||(Split-Size: 0.2)
Error
NR Iterations
AVG Ein Over 5 Splits
Ein Total AVG
Eout Total AVG


Fig#1/3 General Performance On 5 Splits For Both Methods||(Split-Size: 0.2)
Error
Random Splits
5 Ein Avarages on 300 Iterations (One Each Split)(method_1)
5 Ein Avarages on 300 Iterations (One Each Split) (method_2)
5 Eout (One Each Split) (method_1)
5 Eout (One Each Split) (method_2)