Logistic Algorithm Report

The Algorithm

The weight are initialized with very small random numbers instead of zero. Through trial and error I found this to be a faster way to reach convergence. The sigmoid, function will take the linear classifier "`np.dot(X, weights.T)`" as a parameter. It will fit the output between 1 and 0 The cross- entropy/ cost / Ein is calculated: "`-np.sum(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))/ N + (lambDa / (2 * N)) * np.sum(weights**2)`". To this calculator as seen in the code the regularization part is added, where lambda is the regularization parameter(it defines the strength) and the sum of the squares of weights that will penalize large weights to avoid overfitting. At this point the main part of the algorithm is implement the gradient-Ein. The gradient descent will find the weights that minimize the regularized logistic loss in order to reach an optimized generalization. It is comprised of two derivatives. The first one, "`np.dot(X.T, (y_pred - y))`" is the derivative of the logistic loss with respect to the weights and determines how much increase or decrease the logistic loss will have. The second, "`(lambDa/N) * weights`" is the derivative of the regularization term, mentioned above, and will prevent the overfitting. The gradient multiplied with learning rate (step) will update in negative the weight vector, " `weights -= learning_rate * gradient_Ein`". The goal here is to minimize the logistic loss. Finally the logistic loss is calculated, mentioned above, and by plotting it will help visalize if the gradient descent is working well. Finally the weight is the value that is returned in the algorithm. After the training part the Eval is calculated based on the average accuracy, then this value will be subtracted from 1, "`Eval = 1 - np.mean(prediction == y)`". The rest of the code is just implementation of this algorithm and cross validation.

The Experiment

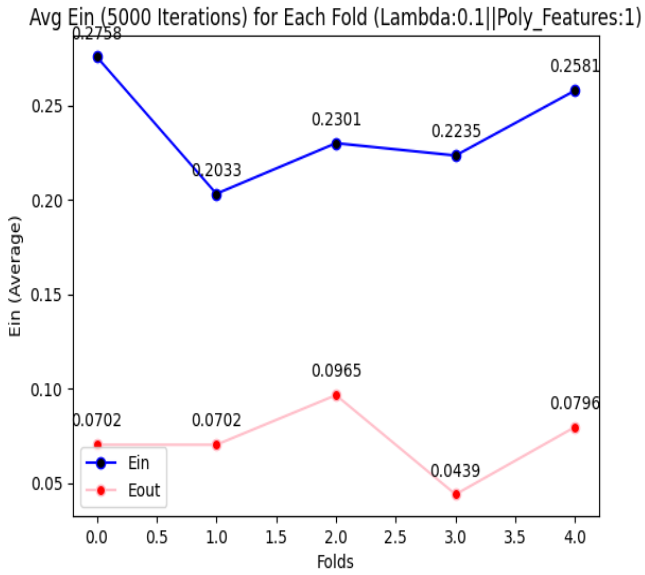
As required the dataset used for this experiment is the Breast Cancer Data Set from sklearn. The set is used tro train and test my algorithm. The hyperparameter, nr of iteration is set to 1000. I noticed by prior trials that this is a good nr of iteration for the final setup since a provides a good picture of the logistic loss tendency. I apply 1000 iteration for each lambda which are allocated in an array of five elements. To these settings the polynomial feature increase is added. The feature's polynomial space is increased as required at 2nd 3rd and 5th degree. The last setting regards the k-foldind in order to perform a cross validation and determine the general performance.

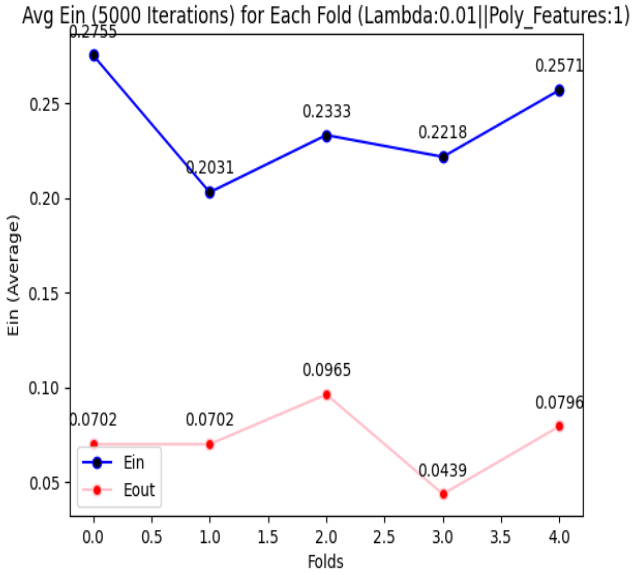
Results & Observations

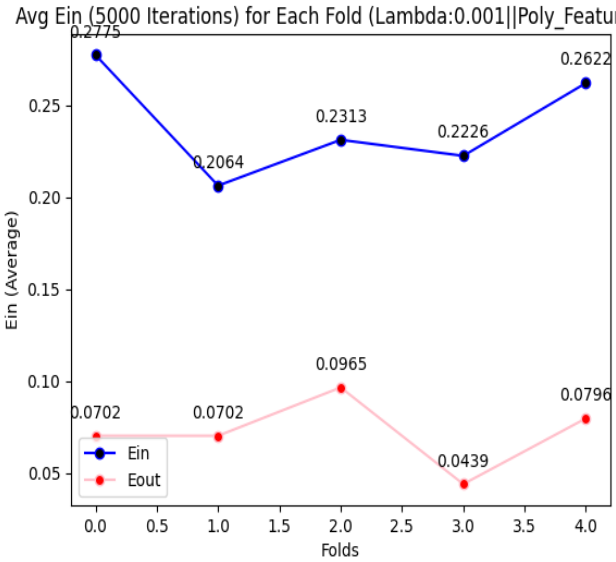
The first observation regards feature polynomial transformation. Whereas the pc would handle polynomials features up to 5 without hesitation, it was not able to run when the polynomial was raised to 10. According to the error messages on the terminal, features with that polynomial would require approximately 3TB of memory. It makes me think of all logistical and financial implications that working with big data can introduce. Another observation is the fact that by applying PCA(feature reduction) the processing it will get faster but the end result was not encouraging therefore I opted for not including it in my final implementation since the program would run anyway.

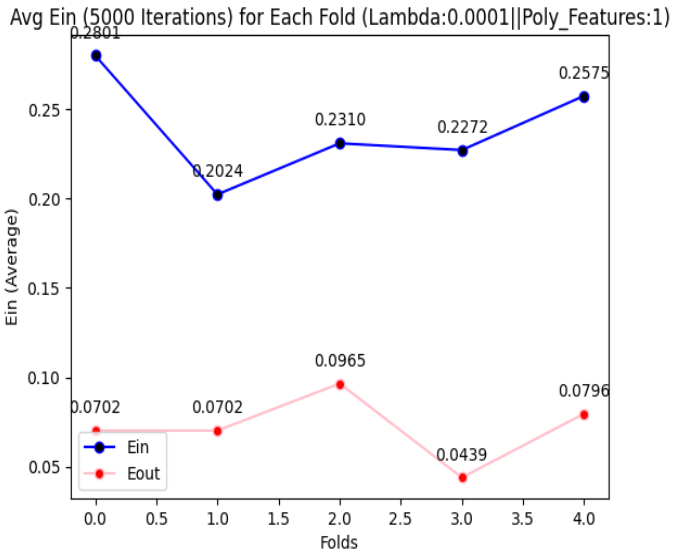
Feature Polynomial = 1

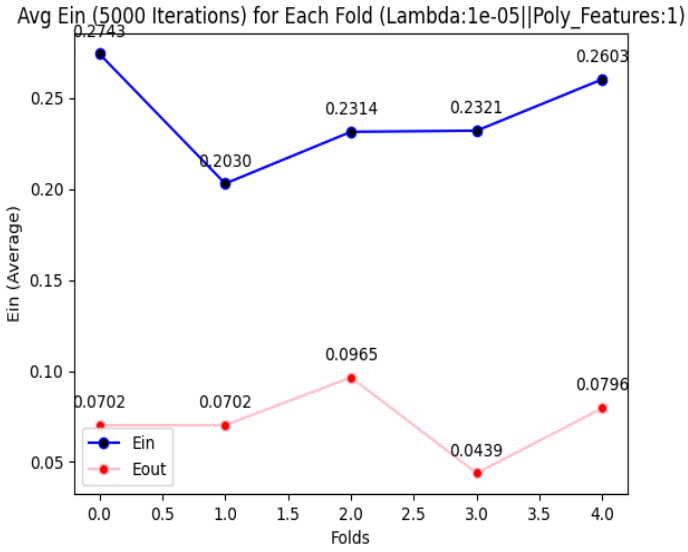
At this magnitude the algorithme performs optimally. This is the best performance among all other magnitudes. Ein and Eval are very small and also they are positined close to each other. Overall the best performance with this magnitude is achieved when lambda = 0.0001.





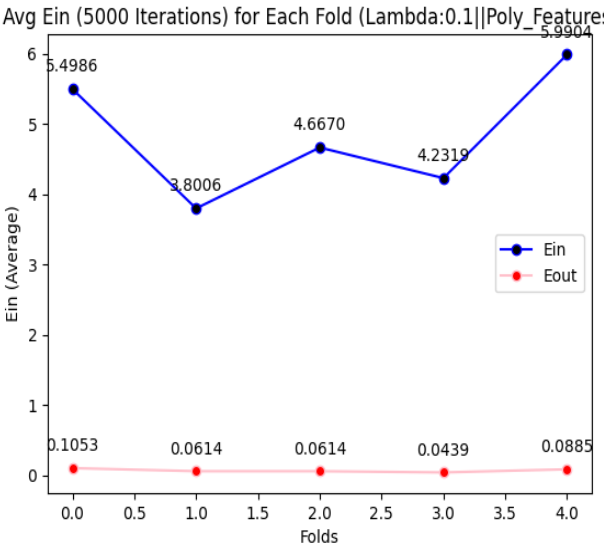

Avg Ein (5000 Iterations) for Each Fold (Lambda:0.001||Poly_Features:1)
Ein (Average)
Folds
Ein
Eout
0.2775
0.2064
0.2313
0.2226
0.2622
0.0702
0.0702
0.0965
0.0439
0.0796

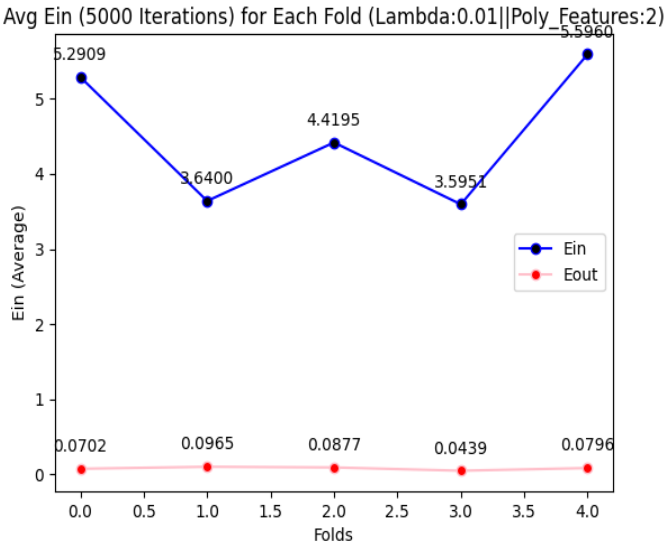

Avg Ein (5000 Iterations) for Each Fold (Lambda:0.0001||Poly_Features:1)
Ein (Average)
Folds
Ein
Eout
0.2801
0.2024
0.2310
0.2272
0.2575
0.0702
0.0702
0.0965
0.0439
0.0796

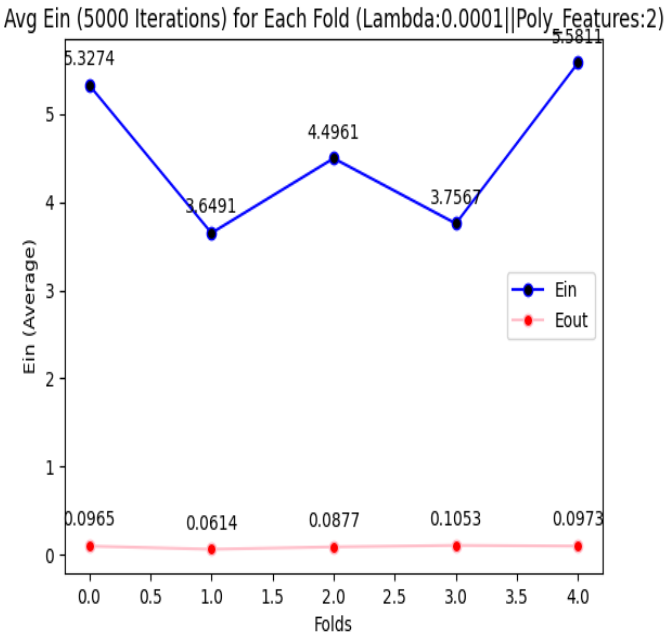

Avg Ein (5000 Iterations) for Each Fold (Lambda:1e-05||Poly_Features:1)
Ein (Average)
Folds
Ein
Eout
0.2743
0.2030
0.2314
0.2321
0.2603
0.0702
0.0702
0.0965
0.0439
0.0796

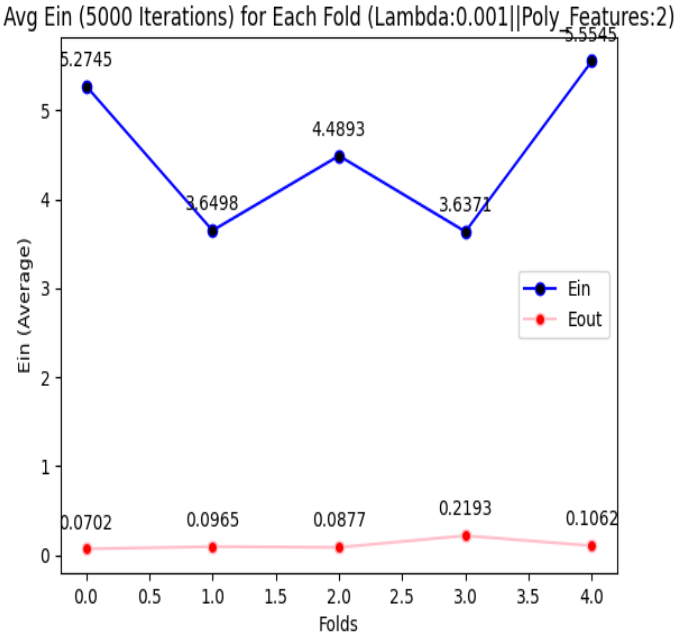
Feature Polynomial = 2

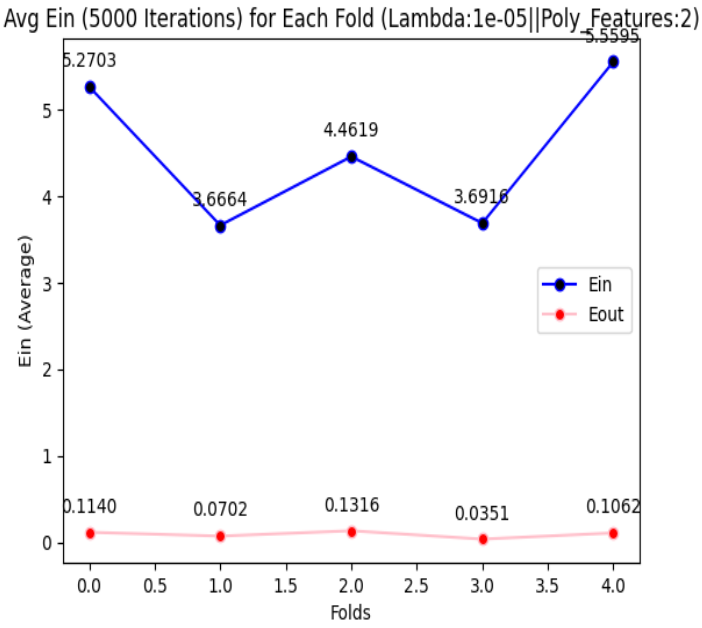
In this case the performance is still good but it is noticeable a slight distancing between Ein and Eval. Seemingly the lambda = 0.01 is the best choice even though by very small margins.





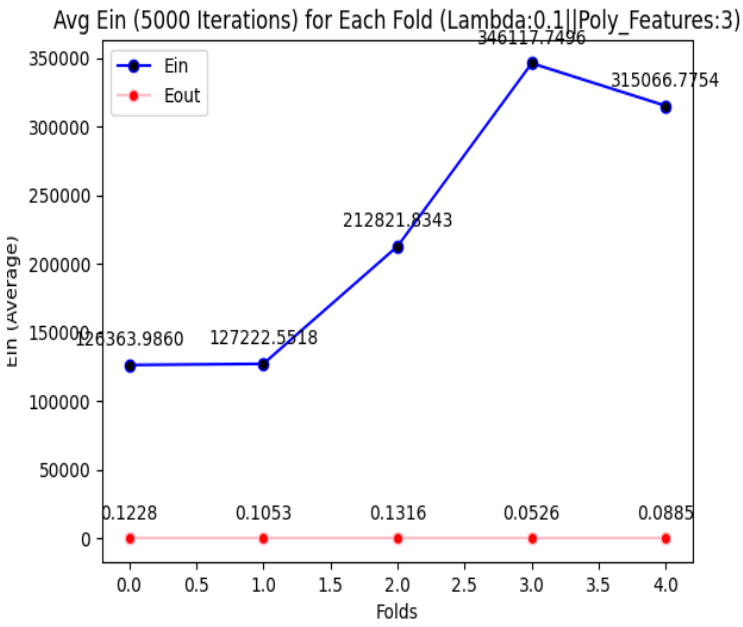


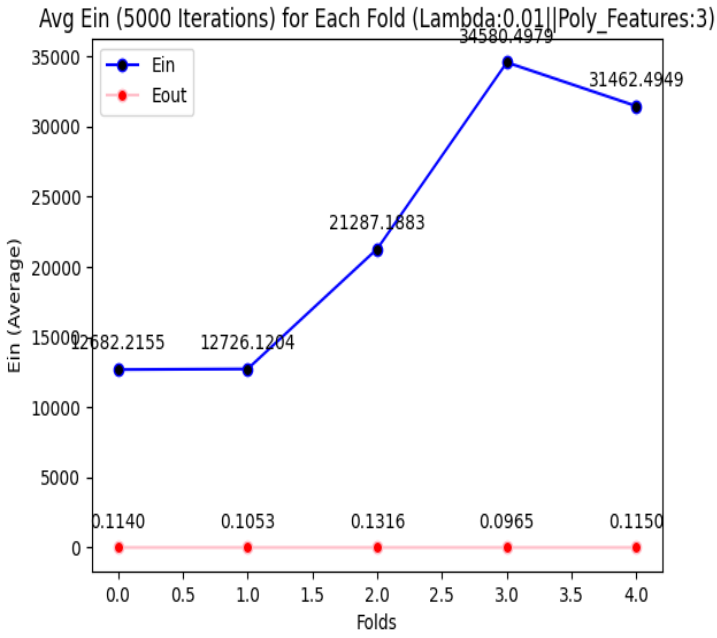


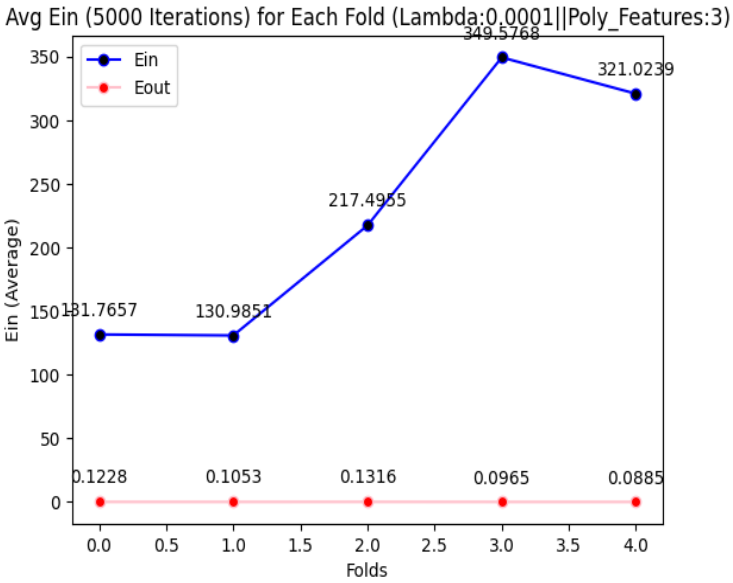


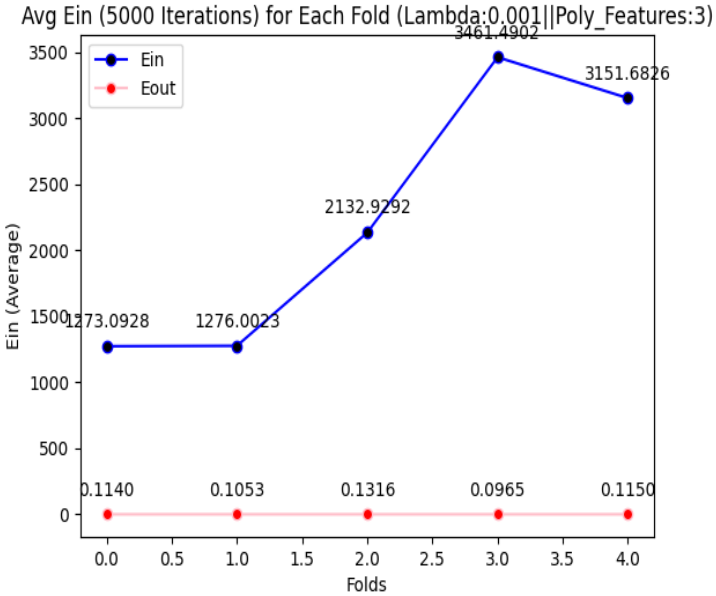
Feature Polynomial = 3

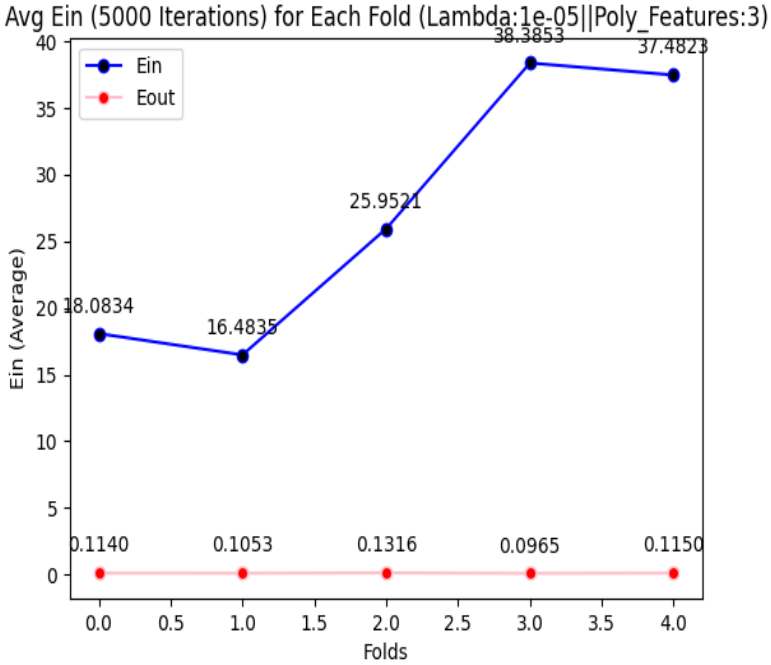
In this case the better performance is achieved when lambda get smaller in fact when lambda = 0.00001 it scores the best performance and the worst when lambda= 0.1





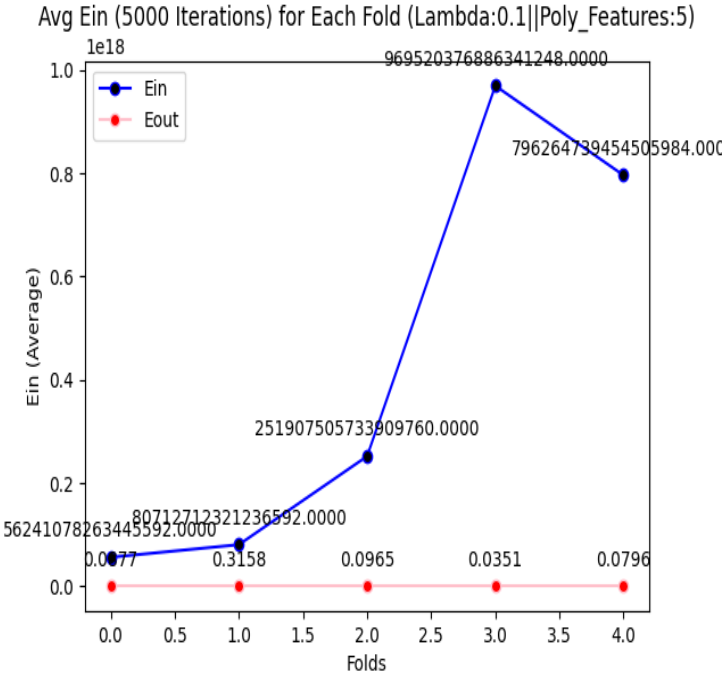

Avg Ein (5000 Iterations) for Each Fold (Lambda:0.0001||Poly_Features:3)
Ein
Eout
Ein (Average)
Folds
131.7657
130.9851
217.4955
349.5768
321.0239
0.1228
0.1053
0.1316
0.0965
0.0885

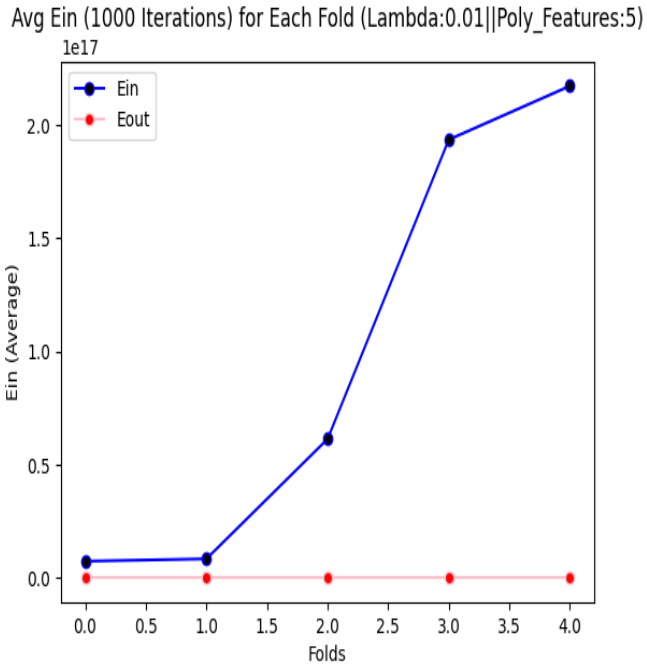

Avg Ein (5000 Iterations) for Each Fold (Lambda:0.001||Poly_Features:3)
Ein
Eout
Ein (Average)
Folds
1273.0928
1276.0023
2132.9292
3461.4902
3151.6826
0.1140
0.1053
0.1316
0.0965
0.1150

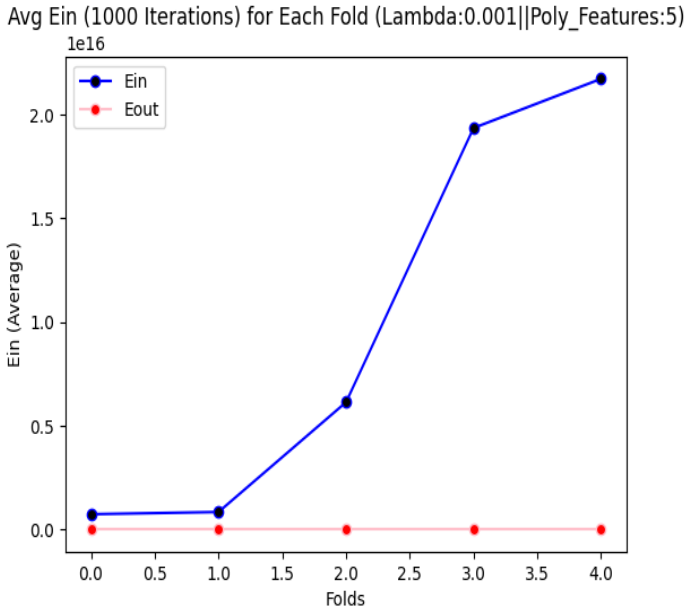

Avg Ein (5000 Iterations) for Each Fold (Lambda:1e-05||Poly_Features:3)
Ein
Eout
Ein (Average)
Folds
18.0834
16.4835
25.9521
38.3853
37.4823
0.1140
0.1053
0.1316
0.0965
0.1150

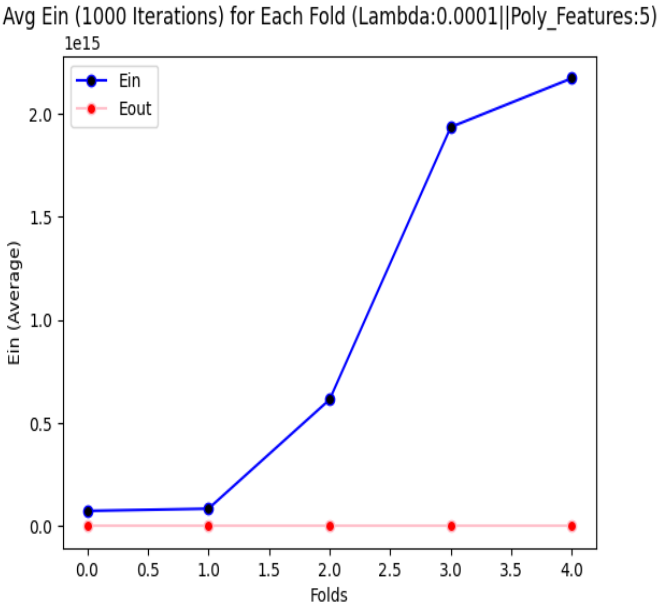
Feature Polynomial = 5

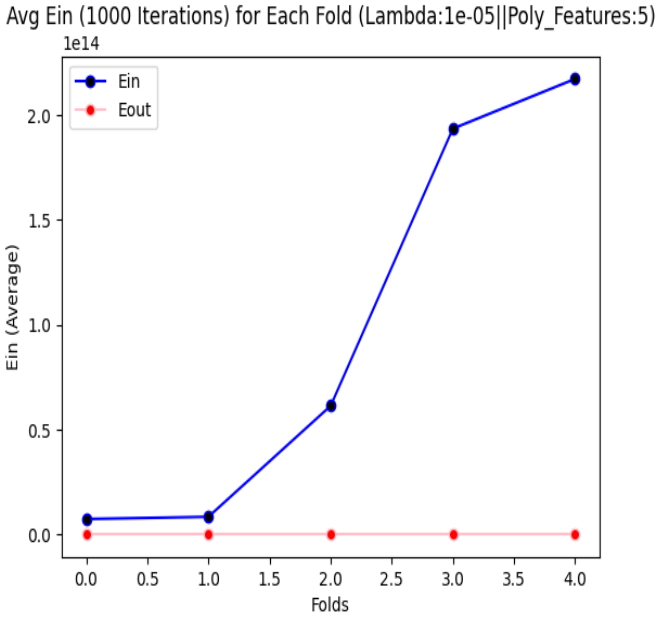
With this magnitude the performance is the worst in terms of distance between Ein and Eval. Ein values are skyrocketing. The performance gets better while labda gets smaller.











Conclusion

In my personal experience this was a challenging experiment since it involved dealing with several parameter/ settings that have an effect on the general performance. In this case was interesting to notice that increasing the magnitude of the feature polinomial was caunter productive as the performance drastically decreased. This may be explained that more complexity was created hence there was overfitting happening. Interestingly by applying feature reduction (PCA) would not bring any relevant improvement. I am not sure if understand the reason for it. However by decreasing lambda at higher magnitude it had a positive effect while improving performance

even if not in an incisive manner. Also I during this experiment I kept nr of iteration and and the learning rate at fixed value since it would have complicated more the experiment. However, I noticed that the more iterations the better the model performs. Another thing that I noticed is that unlike the Pocket Algorithm experiment Eval stays constantly at low levels showing that it stil generalizes well. I am convinced that in this case there is no need to increase the polinomial space of the features.