LEARNING TO RANK USING LINEAR REGRESSION

CSE 574 - Introduction to Machine Learning (Fall 2017)

KETAN SHAH
ketansha@buffalo.edu
Person # - 50247131

PARITOSH WALVEKAR
pwalveka@buffalo.edu
Person # - 50248710

CONTENTS

Brief

Project 2 of the machine learning addresses the implementation of one of the basic learning algorithms in this domain called linear regression.

We focus mainly on two ways to implement linear regression namely the closed form solution and the stochastic gradient descent method. We were to implement two algorithms on two datasets; the LeToR dataset (published by Microsoft Asia) and a synthetically generated dataset. The synthetic dataset is generated using unknown functions and some Gaussian noise. The final aim of the project remains to show how the model designed using linear regression fits a standard and a synthetic dataset.

The data was divided into three sets used for training, validation and testing sections. Eighty percent of the data is used to train the model, ten percent of the data used for validation and remanning ten percent data is used for testing of the model.

While the first project focused on implementing the basic probability operations, this project aptly uses all those implementations in building a robust linear regression model.

This report in the further sections gives a detail about the implementation of the algorithms, tuning of the hyper parameters and the helps you visualize the output of the learning algorithm using various overlapping line graphs.

Hyper Parameters

What are Hyper Parameters?

Hyper parameters are defined as the parameters in the learning algorithm that cannot be learned from the training data. We have to arbitrarily start with a set of hyper parameters and tune them to get as low error as possible.

Hyper Parameters in this project

The hyper parameters in the implementation of the closed form solution are as follows:

Hyper Parameter Symbol	**Details**
k	Number of clusters in the data
M	Number of basis functions
∑	Spread of a specific cluster in clustered data
μ	Mean of a specific cluster in the clustered data
λ	Regularization Constant
η	Learning Rate (used in Gradient Descent)

Choosing the hyper parameters

While implementing the closed form solution or the stochastic gradient descent, we choose M basis functions to represent the linear relation between real valued target and the input. All the input row vectors are evaluated using the M basis functions and a design matrix is calculated. For N training sample and M basis functions, we get a design matrix of dimension N x (M+1) (we keep first column of the design matrix to be all 1 values).

The best value of k/M will be the one which gives minimum error when we run the trained model on the validation set. The details on the procedure to choose the hyper parameters are in the grid search section of the report.

Closed Form Solution

Implementation

Post the grid search, with the M and λ fixed, we calculate the design matrix. The design matrix is of the following form:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix}$$

In the above matrix, phi is the Gaussian basis function. Having calculated the design matrix, we can learn the parameters from the trainman data with the following formula:

$$\mathbf{w}^* = (\lambda \mathbf{I} + \mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{t}$$

In the above equation, t represents the vector of the target values for the input values. Using the design matrix, λ from the grid search and the t vector we learn the parameter vector from the input data.

Evaluating Performance

To evaluate the performance of the model with the parameters we learned, we compute the target values for the validation input dataset.

We now have two datasets for the output. The one predicted by our model and the perfect one which is already provided to us. The measure of the performance which we have used is the root mean squared error.

The comparison of the RMSE for the validation and test dataset indicates whether the model we learned is a perfect fit, under fit or over fit in general to the training data. If the RMSE for test is less than validation, the model is under fit. It the test RMSE is greater than the validation, the model is over fit. The model will be a perfect fit when the test RMSE is equal to the validation RMSE.

For our implementation of the computation following is RMSE (add for both dataset)

Dataset	**RMSE**	**Dataset**	**RMSE**
Letor Validation set	0.5529580082018908	Synthetic Validation set	0.7862039381581133
Letor Test Set	0.6409478374818667	Synthetic Test Set	0.788335985086342

Stochastic Gradient Descent

Implementation

Post the grid search, with the M and λ fixed, we calculate the design matrix. The design matrix is of the following form:

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix}$$

In the above matrix, phi is the Gaussian basis function. Having calculated the design matrix, the parameters are updated for a batch of input samples using the below formula:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

The update in parameters is calculated by subtracting the product of the learning rate and the gradient of the error function. Instead of updating parameters for every input sample, we use a slightly modified version of the algorithm called batch gradient descent.

We divide the input samples into batch, accumulate the gradient at every level in the batch and subtract that at once from the parameters to update the parameters. Using batch updates, ensure speedy run of the algorithm for large datasets. In the implementation, we can change the batch size and experience the time difference in the execution first hand.

Evaluating Performance

To evaluate the performance of the model with the parameters we learned, we compute the target values for the validation input dataset.

We now have two datasets for the output. The one predicted by our model and the perfect one which is already provided to us. The measure of the performance which we have used is the root mean squared error.

The comparison of the RMSE for the validation and test dataset indicates whether the model we learned is a perfect fit, under fit or over fit in general to the training data. If the

RMSE for test is less than validation, the model is under fit. It the test RMSE is greater than the validation, the model is over fit. The model will be a perfect fit when the test RMSE is equal to the validation RMSE.

For our implementation of the computation following is RMSE(add for both letor and synthetic dataset)

Dataset	**RMSE**	**Dataset**	**RMSE**
Letor Validation set	0.5538202376210921	Synthetic Validation set	0.79003501939716
Letor Test Set	0.6406205040244164	Synthetic Test Set	0.7871819462980018

Performing Grid Search

How does grid search work?

We performed the training algorithm on different combinations of M and λ. For each combination, the parameters of the model were different. For each model, root mean squared error was calculated for the validation dataset. The combination which yielded minimum RMSE was used in the main algorithm to train the model.

These parameters can be seen fixed in the initial part of the main.py file.

Innovative Part - Using public cloud

It is quite evident that the computing power required to perform the combinations in the above section is huge and it cannot be run on a single machine. We had always read about the compute power at disposal on the public cloud but never used it. This would have been no better opportunity than this to get started on taking advantage of that massive compute power.

We signed up for Google Cloud Platform and ran a virtual machine with 32 cores at disposal and the ran the grid search on it. It was blazing fast and we are more than interested to try out different algorithms on such a setup. This approach has opened up new avenues for us.

Grid Search Results

The below table lists the results obtained from the grid search:

Closed Form Solution:

	Letor Validation Dataset Error	Synthetic Validation Dataset Error
λ=0.1, M=5	0.555243178	0.785268621
λ=0.1, M=10	0.554263122	0.785462893
λ=0.1, M=15	**0.551492662**	0.785100407
λ=0.1, M=20	0.553817456	0.785996606
λ=0.1, M=25	0.552281771	0.785867411
λ=0.1, M=30	0.552626458	0.78547415

	Letor Validation Dataset Error	Synthetic Validation Dataset Error
λ=0.1, M=35	0.552897296	**0.7840498**
λ=0.1, M=40	0.555017476	0.786211901
λ=0.1, M=45	0.556390392	0.786249553

From the above table we see that fixing the hyper parameters at M=15 for LeToR dataset and M=35 for synthetic gives minimum RMSE on the validation set. The value λ=0.1 remains constant throughout.

Stochastic Gradient Descent:

	Letor Validation Dataset Error	Synthetic Validation Dataset Error
λ=0.1, M=5	0.553479825	0.788684977
λ=0.1, M=10	0.553650049	0.788790894
λ=0.1, M=15	0.553555654	0.787414455
λ=0.1, M=20	0.554415317	0.790037021
λ=0.1, M=25	0.553886443	**0.782520331**
λ=0.1, M=30	0.553947508	0.787252149
λ=0.1, M=35	**0.552806288**	0.78870966
λ=0.1, M=40	0.555018972	0.786211945
λ=0.1, M=45	0.556391610	0.786242590

From the above table we see that fixing the hyper parameters at M=35 for LeToR dataset and M=25 for synthetic gives minimum RMSE on the validation set. The value λ=0.1 remains constant throughout.

We can fine tune the hyper parameters to a large extent (large number of permutations with λ and M changing) implied we a huge amount compute power at out disposal.

Code Execution

Environment Details

Module/Platform/Library Name	Version
Python	3.6
Mac OS	10.13
Conda	4.3.25
Numpy	1.13.1
SKLearns	0.20
Pandas	0.20
Matplotlib	2.1.0

Module Details

1. regressionEssentials

This module implements the functions that are necessary for both the linear regression algorithms. The function included are:

Function Name	Description
calculateM	This function will cluster data into M clusters.
createDataClusters	This function divides the data according to the labels associated to the data.
covariancePerCluster	This function calculates the covariance matrix for the cloistered data.
calculateDesignMatrix	This function calculates the design matrix using the design matrix and the input data.
calculateRootMeanSquaredError	This function calculates the root mean squared error given two sets of values.

2. stochasticGradientDescent

This module implements the functions necessary for modeling using the stochastic gradient descent algorithm. The functions included are:

Function Name	Description
stochasticUpdate	This function calculates the parameter vector for each batch of the input and return the parameter vector with the minimum RMSE on the validation set.
learnParameterSGD	This function initializes a random parameter vector and calls stochasticUpdate to update the parameter vector.
computeTargetValuesSGD	This function calculates the output vector using the input data, parameter vector and the design matrix.
computeTargetSGD	This function computes the output vector with the input data and parameter vector.
calculateBatchDeltaE	This function computes the gradient change for a batch of input samples.

3. closedFormSolution

This module implements the functions necessary for modeling using the closed form solution. The functions included are:

Function Name	Description
computeClosedFormSolution	This function computes the parameters using the input data, output data and the design matrix of the input data.
computeTargetValuesClosedFormSolution	This function computes the output vector given the learned parameters and input data.

Function Name	Description
learnParametersClosdFormSolution	This function calls the various function in the interim to compete the design matrix of the input. Then calls the computeClosedFormSolution to model the parameters using the input data, output data and the design matrix.

What's included in the zip?

1. main.py
2. regressionEssentials.py
3. stochasticGradientDescent.py
4. closedFormSolution.py
5. main.pdf
6. synthtetic_input_data.csv
7. synthtetic_output_data.csv
8. letor_input_data.csv
9. letor_output_data.csv
10. proj2.pdf

Instructions to execute the code

1. Unzip the zip file.
2. Navigate into the root level of the folder created by unzipping the submitted file.
3. Run the below command.

```
python main.py
```

Output

Console Output

```
--------------------------------------
Training the model for the letor input data set with closed form
solution...
The parameters are -
[[ 0.31206134  0.73416108  0.28303536  2.97214632 -1.02400359
-1.60291557 -3.17577579 -0.3304074 2.35766272 1.57701284
-0.21304171 0.34271998 0.43161648 3.51316696 -1.72930192
-0.66884511]]
RMSE for letor validation dataset is - 0.5529580082018908
RMSE for letor test dataset is - 0.6409478374818667
Graphical representation of the output can be found in the file
titled letor-cfs-test-output-plot.png!
--------------------------------------
Training the model for the synthetic input data set with closed
form solution...
The parameters are -
[[ 0.99226184] [-0.07249242] [-0.01295248] [-0.04634675]
[-0.01876571] [ 0.01627097] [ 0.05294998] [ 0.01993638]
[-0.00158734] [-0.0374385 ] [-0.01831549] [-0.0950163 ]
[ 0.00362529] [-0.00319889] [ 0.04604267] [-0.05786594]
[ 0.02521117] [-0.08226291] [-0.04330843] [-0.06793114]
[-0.073603   ] [-0.03260478] [-0.02375276] [-0.11064687]
[ 0.00603568] [-0.0351356 ] [-0.0535085 ] [-0.06929624]
[-0.07093745] [-0.08446785] [-0.07457193] [-0.15084625]
[-0.09897064] [-0.01462209] [ 0.02281188] [-0.06852606]]
RMSE for synthetic validation dataset is - 0.7862039381581133
RMSE for synthetic test dataset is - 0.788335985086342
Graphical representation of the output can be found in the file
titled synthetic-cfs-test-output-plot.png!
--------------------------------------
Training the model for the letor input data set with stochastic
gradient descent...
The parameters are -
[[ 0.299901    0.28854823  0.10572841  0.08138887  0.35660398
0.39130823
```

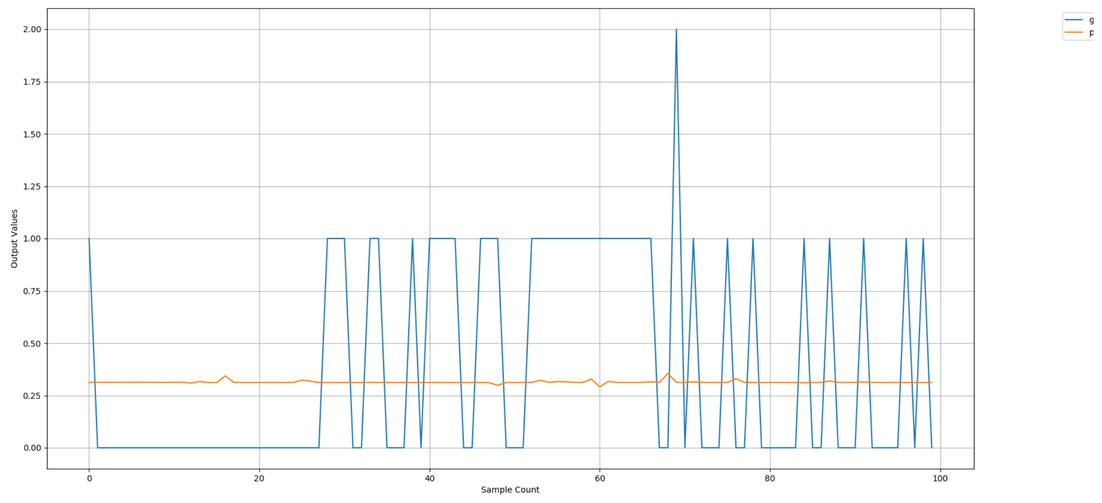
```
   0.12418379  0.07926888  0.45056234  0.53669161  0.22655775
0.39386735
   0.03915133  0.52730749  0.38163015  0.40949742  0.31716448
0.00827979
   0.45093188  0.50743927  0.1349985   0.1795435   0.40622115
0.41513093
   0.15125264 -0.04636527  0.54154459  0.2900554   0.22387876
0.10325244
   0.49672579  0.30097164  0.58395233  0.16524253  0.47251605
0.30243488]]
Graphical representation of the output can be found in the file
titled letor-sgd-test-output-plot.png!
--------------------------------------
Training the model for the synthetic input data set with
stochastic gradient descent...
The parameters are -
[[ 0.6963077   0.30496098  0.51205849  0.40288147  0.40170055
0.38879231
   0.41973608  0.40812637  0.35819148  0.30294847  0.3815828
0.34362793
   0.34265475  0.42264927  0.4178238   0.44889244  0.35321233
0.31636292
   0.41390786  0.39649827  0.45509515  0.38131868  0.36559253
0.45859818
   0.3099335   0.3425399 ]]
RMSE for synthetic validation dataset is - 0.79003501939716
RMSE for synthetic test dataset is - 0.7871819462980018
Graphical representation of the output can be found in the file
titled synthetic-sgd-test-output-plot.png!
--------------------------------------
```

Test Data Output vs Predicted Test Data Output

An overlapped graph of the given output for test data and the predicted output for test data helps better the visualize the performance of the model.

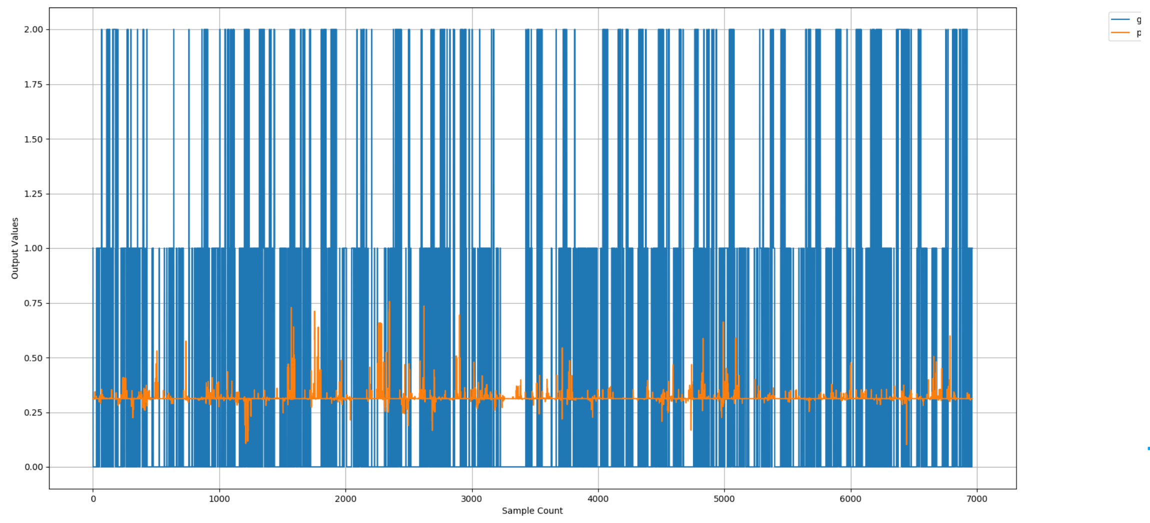
Closed Form Solution Plots

Plot for first hundred samples of the LeToR dataset:

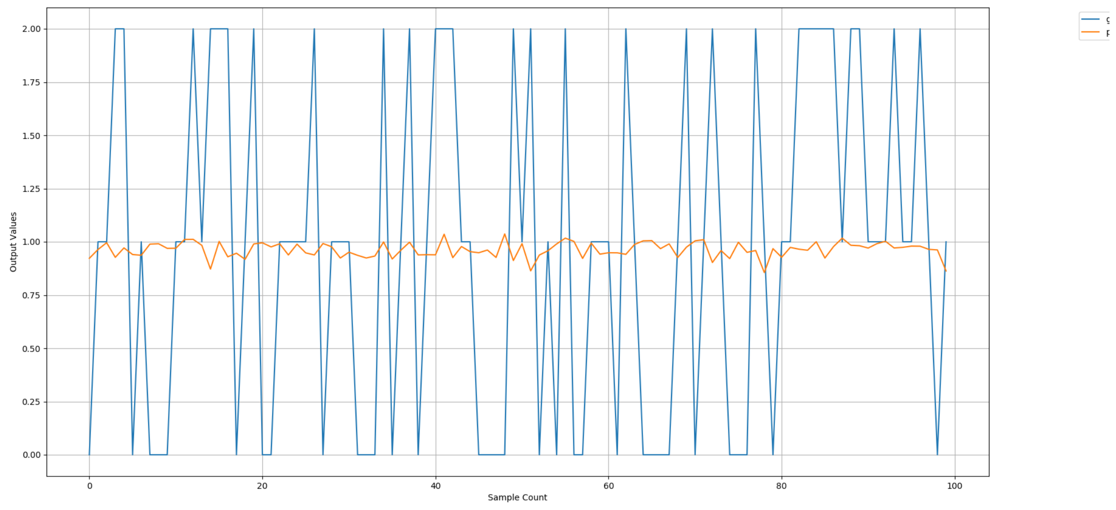


Legend : 🟧 Predicted Output 🟦 Given output

Plot for the complete LeToR dataset:



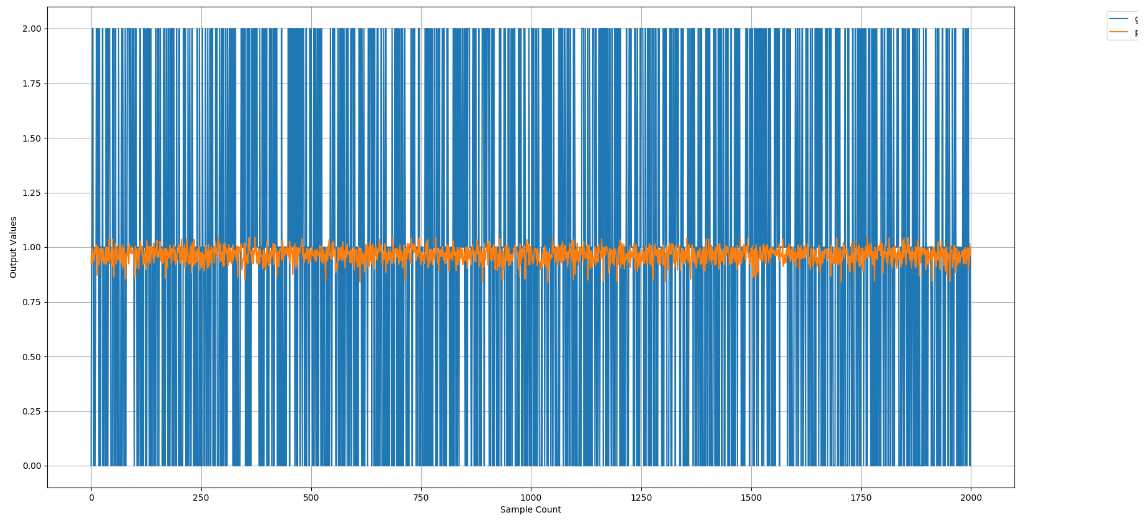
Legend : 🟧 Predicted Output 🟦 Given output



Plot for first hundred samples of the synthetic dataset:

Legend : 🟧 Predicted Output 🟦 Given output

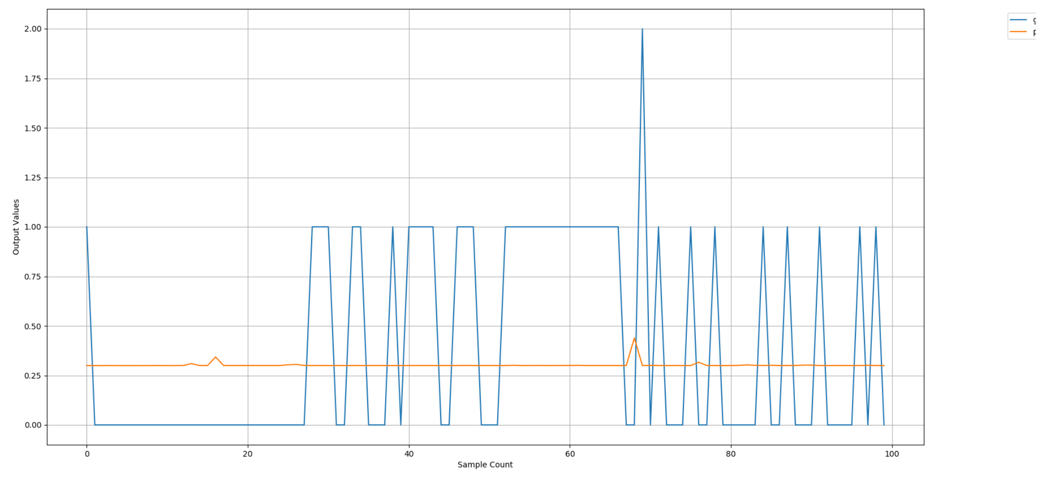
Plot for the complete synthetic dataset:



Legend : 🟧 Predicted Output 🟦 Given output

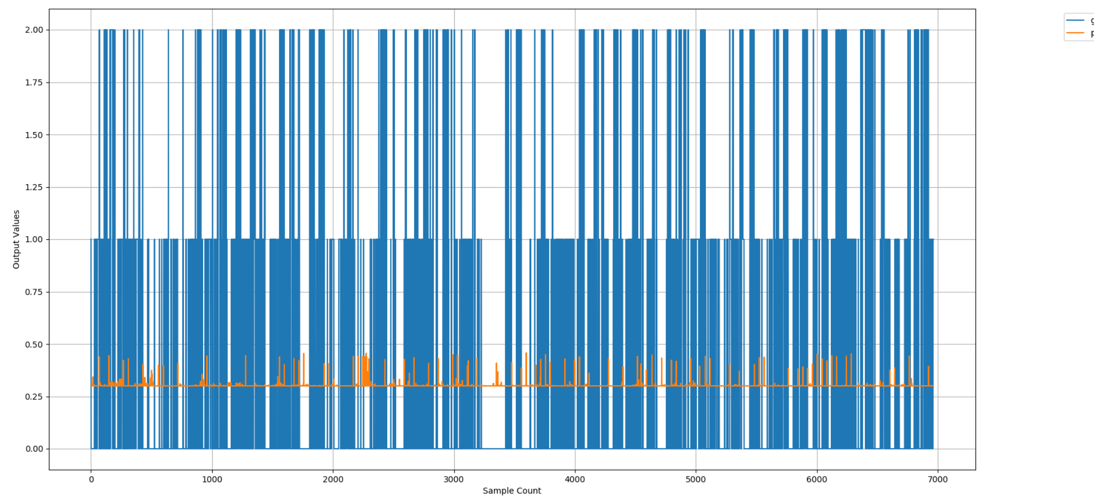
Stochastic Gradient Descent Plots

Plot for first hundred samples of the LeToR dataset:



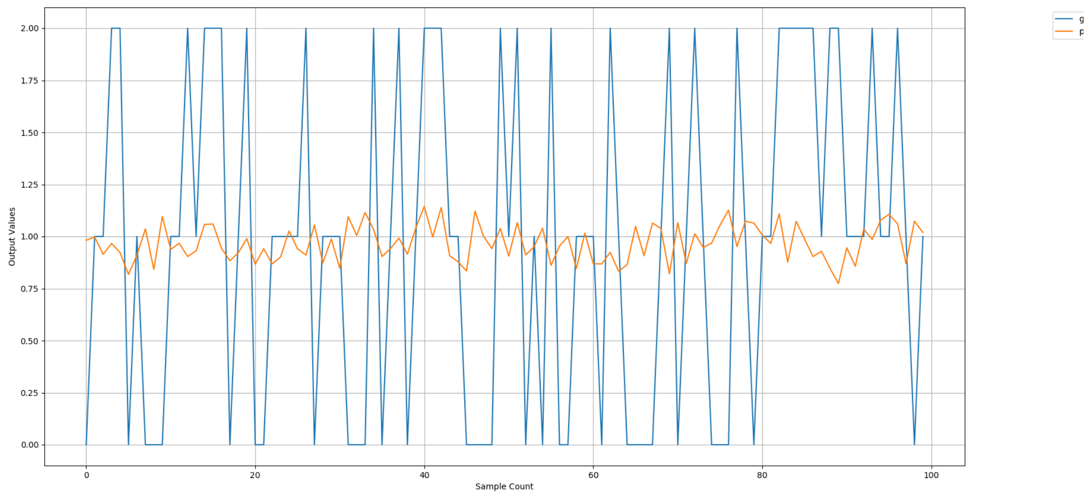
Legend : 🟧 Predicted Output 🟦 Given output

Plot for the complete LeToR dataset:



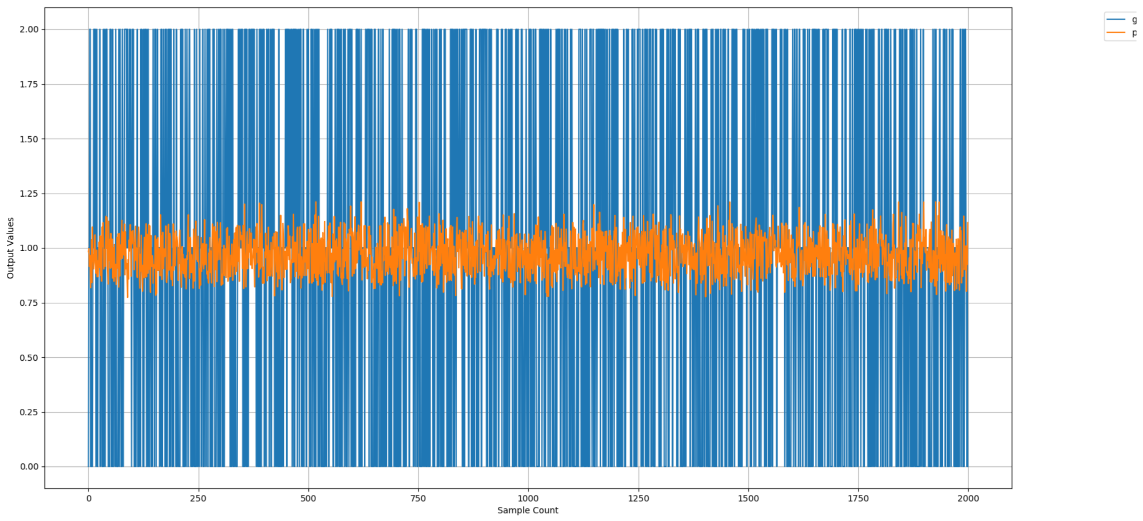
Legend : 🟧 Predicted Output 🟦 Given output

Plot for first hundred samples of the synthetic dataset:



Legend : 🟧 Predicted Output 🟦 Given output

Plot for the complete synthetic dataset:



Legend : 🟧 Predicted Output 🟦 Given output

Test Data RMSE vs Validation Data RMSE

Following table summarizes the RMSE for validation and test subsets for both the datasets:

Dataset	RMSE Error for the model	Algorithm Used
LeTor Validation Set	0.5529580082018908	Closed Form Solution
LeTor Test Set	0.6409478374818667	Closed Form Solution
LeTor Validation Set	0.5538202376210921	Stochastic Gradient Descent
LeTor Test Set	0.6406205040244164	Stochastic Gradient Descent
Synthetic Validation Set	0.7862039381581133	Closed Form Solution
Synthetic Test Set	0.788335985086342	Closed Form Solution
Synthetic Validation Set	0.79003501939716	Stochastic Gradient Descent
Synthetic Test Set	0.7871819462980018	Stochastic Gradient Descent

References

Library Functions

1. Numpy.genfromtxt
2. numpy.asmatrix
3. numpy.random.rand
4. numpy.matmul
5. numpy.add
6. numpy.subtract
7. numpy.multipy
8. sklearns.cluster.KMeans
9. sklearns.metrics.mean_squared_error