CSE 574: Introduction to Machine Learning
Fall 2017

Project 2: Learning to Rank Using Linear Regression

Instructor: Dr. Sargur N. Srihari

Prasad Salvi
Person #: 50207353

Veerappan Saravanan
Person #:50247314

Sarah Mullin
Person #: 34508498

Contents

1 Model and Project Synopsis

For this project, we would like to solve the Learning to Rank (LeToR) problem such that we take a categorical target variable (t=[0,1,2]) and map our input vector x, features used to decipher the ranking, to a real valued scalar target y(x,w) using linear regression. For two datasets, one real data (LeTOR) and one synthetic data, we will predict the weights, error (cost function), using both a closed form solution and stochastic gradient descent.

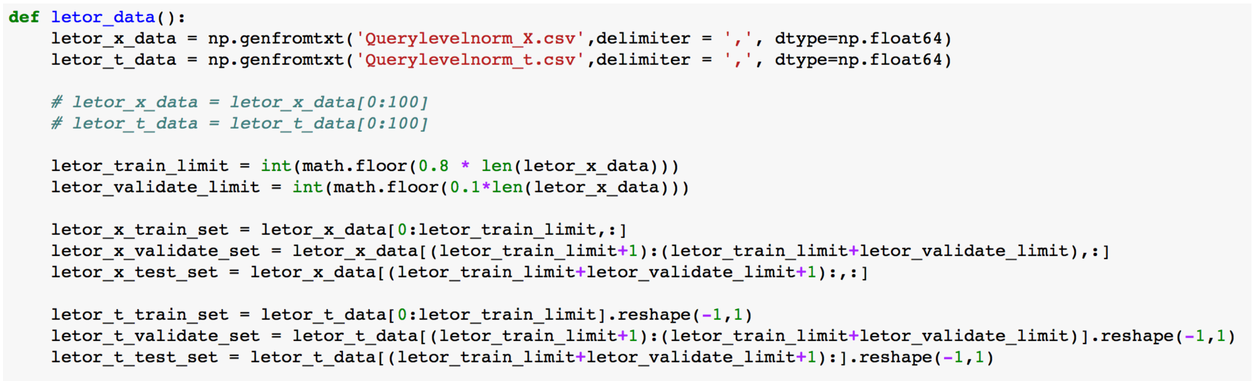
1.1 Model

Our linear regression model has the form $y(x, w) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$ where
$\boldsymbol{w} = [\, w_0, w_1, w_2, \dots, w_M]$ is a weight vector from the training samples and where
$\boldsymbol{\phi} = [\, \phi_0, \phi_1,\ \phi_2, \dots,\ \phi_M]$ such that we have chosen the optimal number of basis functions to be 9 with the assumption that the basis function $\phi_0 \equiv 1$ for the bias term.

1.2 Data Partition

For the LeTOR data there are 46 features (46 columns for each data point x_i) and a total sample size of 69623. Data was split into training, validation, and testing sets such that the training set was 80% (n_{train}=55,698) and the validation ($n_{validation}$=6,961) and testing (n_{test}=6,962) sets were each 10%.

LeTOR Data Set



```
def letor_data():
    letor_x_data = np.genfromtxt('Querylevelnorm_X.csv',delimiter = ',', dtype=np.float64)
    letor_t_data = np.genfromtxt('Querylevelnorm_t.csv',delimiter = ',', dtype=np.float64)

    # letor_x_data = letor_x_data[0:100]
    # letor_t_data = letor_t_data[0:100]

    letor_train_limit = int(math.floor(0.8 * len(letor_x_data)))
    letor_validate_limit = int(math.floor(0.1*len(letor_x_data)))

    letor_x_train_set = letor_x_data[0:letor_train_limit,:]
    letor_x_validate_set = letor_x_data[(letor_train_limit+1):(letor_train_limit+letor_validate_limit),:]
    letor_x_test_set = letor_x_data[(letor_train_limit+letor_validate_limit+1):,:]

    letor_t_train_set = letor_t_data[0:letor_train_limit].reshape(-1,1)
    letor_t_validate_set = letor_t_data[(letor_train_limit+1):(letor_train_limit+letor_validate_limit)].reshape(-1,1)
    letor_t_test_set = letor_t_data[(letor_train_limit+letor_validate_limit+1):].reshape(-1,1)
```

Synthetic Data Set

For the synthetic data partition, there are 10 features (10 columns for each data point x_i) and a total sample size of 20,000. Data was split into training, validation, and testing sets such that the training set was 80% (n_{train}=16,000) and the validation ($n_{validation}$=2000) and testing (n_{test}=2000) sets were each 10%.

```python
def syn_data():
    dataset_input= pd.read_csv('input.csv',delimiter = ',', header=None)
    dataset_output = pd.read_csv('output.csv',delimiter = ',', header=None)

    syn_x_data = dataset_input.as_matrix()
    syn_t_data = dataset_output.as_matrix()

    # syn_x_data = syn_x_data[0:100]
    # syn_t_data = syn_t_data[0:100]

    syn_train_limit = int(math.floor(0.8 * len(syn_x_data)))
    syn_validate_limit = int(math.floor(0.1*len(syn_x_data)))

    syn_x_train_set = syn_x_data[0:syn_train_limit,:]
    syn_x_validate_set = syn_x_data[(syn_train_limit+1):(syn_train_limit+syn_validate_limit),:]
    syn_x_test_set = syn_x_data[(syn_train_limit+syn_validate_limit+1):,:]

    syn_t_train_set = syn_t_data[0:syn_train_limit].reshape(-1,1)
    syn_t_validate_set = syn_t_data[(syn_train_limit+1):(syn_train_limit+syn_validate_limit)].reshape(-1,1)
    syn_t_test_set = syn_t_data[(syn_train_limit+syn_validate_limit+1):].reshape(-1,1)
```

1.3 Basis Functions and creation of the design matrix $\boldsymbol{\phi}$

Gaussian radial basis functions such that $\phi_j(x) = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_j})^{\boldsymbol{T}}\boldsymbol{\Sigma}_{\boldsymbol{j}}^{-1}(\boldsymbol{x}-\boldsymbol{\mu_j})\right.$ were used. The $\boldsymbol{\mu_j}$ were found by the method of k-means clustering. Since we have a large amount of features in the LeTOR dataset (46), using all in sample datapoint may lead to overfitting. Therefore, we need to choose M-1 (such that M=bias+radial basis functions) representatives such that M-1<46 that can interpolate. To choose the best possible M-1 centers, we used the scikit learn function KMeans to cluster the training data into M-1 centers.

```python
def calculateSigmaInverse(data):
    temp = []
    row,col=data.shape
    for i in range(0,col):
        variance = np.var(data[:,i])
        temp.append(variance)
    dig = np.diag(temp)
    sigmainv = np.linalg.pinv(dig)
    return sigmainv
```

```python
def calculateMean(M,data):
        sigma_inverse = []
        kmeans = KMeans(n_clusters=M,random_state=0)
        kmeans.fit(data)
        centroid = kmeans.cluster_centers_
        labels=kmeans.labels_
        return centroid
```

```python
def calculateMeanAndVariance(M,data):
        sigma_inverse = []
        kmeans = KMeans(n_clusters=M,random_state=0)
        kmeans.fit(data)
        centroid = kmeans.cluster_centers_
        labels=kmeans.labels_
        sigma_inverse = []
        for i in range(0, M):
            sigma_inverse.append(np.eye(data.shape[1]))
        return centroid,np.asarray(sigma_inverse)
```

```python
def compute_design_matrix(X, centers, spreads):
    # use broadcast
    basis_func_outputs = np.exp(
    np. sum(np.matmul(X - centers, spreads) * (X - centers), axis=2) / (-2) ).T
    # insert ones to the 1st col
    return np.insert(basis_func_outputs, 0, 1, axis=1)
```

In our final model, we chose we chose $\boldsymbol{\Sigma}_j^{-1}$ such that for each cluster created in the k-means clustering we calculated the variance and took the inverse of the diagonal matrix. In addition, we also tried using the identity matrix such that each variance for a column on the diagonal (and inverse variance) is equal to 1.

The design matrix, using the basis function calculated by k-means clustering, had this form:

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

Such that N=number of samples in the training, validation, and test sets, respectively.

1.4 Closed Form Solution of the weights, **w,** $E_{RMS,}$ and E(w)

Using the design matrix and the basis function found above, the weights, **w** were computed with the weight decay regularization term and followed this formula:

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

Such that when λ=0 the regularization term drops out of the equation and w is calculated in this way:

$$\mathbf{w}_{\mathrm{ML}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \mathbf{t}$$

Error was calculated such that:

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

and models were evaluated using:

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N_{\mathrm{V}}}$$

The results can be found in Section 2.1.1 and 2.2.1 for the closed form solution. An excerpt of our code is as follows to calculate the closed form solution, $E(w)$ and E_{RMS}:

```python
def closed_form_sol(L2_lambda, design_matrix, output_data):
    return np.linalg.solve(L2_lambda * np.identity(design_matrix.shape[1]) + np.matmul(design_matrix.T, design_matrix),
```

```python
def calculateErrorWithReg(phi,w,t,lamb):
    w_t = np.transpose(w)
    sum = 0
    e_d=0
    row,col=phi.shape
    for i in range(0,row):
        temp = np.dot(w_t,np.transpose(phi[i,:]))
        squr_term = np.square(np.subtract(t[i],temp))
        sum = sum + (squr_term)
    e_d = sum/2
    #e_w = lamb*(np.dot(w_t,w))/2
    #error=e_d+e_w
    return e_d
```

```python
def calculateRMSError(error, shape):
    e_rms = np.round(np.sqrt((2*error)/shape),5)
    return e_rms
```

1.5 Stochastic Gradient Descent Solution

Our stochastic gradient descent solution used early stopping and a mini-batch algorithm, such that the batch sized equaled the total sample size (batch stochastic descent). Since there was not a significant cost in computation time, this was chosen to ensure convergence.

The results can be found in Section 2.1.2 and 2.2.2 for the gradient descent solution. An excerpt of our code is as follows to calculate the closed form solution, $E(w)$ and E_{RMS}:

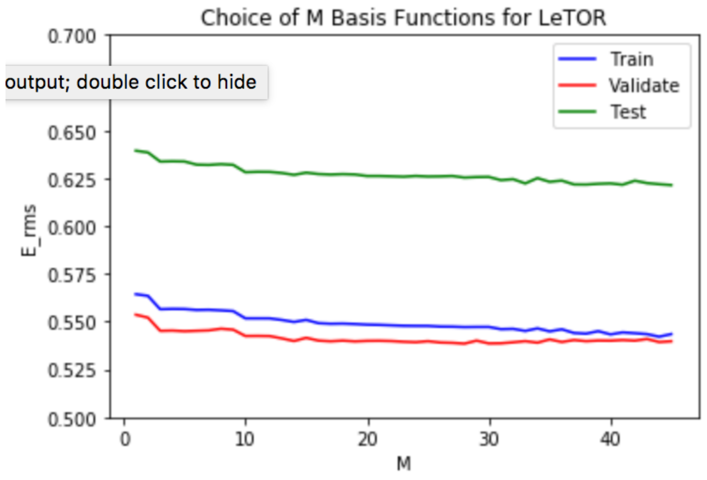
```python
def SGD_sol_early(learning_rate, minibatch_size,num_epochs, L2_lambda,train_design_matrix,train_output_data,validate_de
    N, _ = train_design_matrix.shape
    prev = 999;
    counter = 0
    # You can try different mini-batch size size
    # Using minibatch_size = N is equivalent to standard gradient descent
    # Using minibatch_size = 1 is equivalent to stochastic gradient descent
    # In this case, minibatch_size = N is better
    weights = np.zeros([1, M+1])
    # The more epochs the higher training accuracy. When set to 1000000,
    # weights will be very close to closed_form_weights. But this is unnecessary
    for epoch in range(num_epochs):
        #print("ITERATIONS", epoch)
        size_N = int(N / minibatch_size)
        for i in range(size_N):
            lower_bound = i * minibatch_size
            upper_bound = min((i+1)*minibatch_size, N)
            Phi = train_design_matrix[lower_bound : upper_bound, :]
            t = train_output_data[lower_bound : upper_bound, :]
            E_D = np.matmul((np.matmul(Phi, weights.T)-t).T,Phi )
            E = (E_D + L2_lambda * weights) / minibatch_size
            weights = weights - learning_rate * E
        error=calculateErrorWithReg(validate_design_matrix,np.transpose(weights),validate_output_data,L2_lambda)
        error_rms=calculateRMSError(error,validate_design_matrix.shape[0])
        #print ("Previous ", prev)
        #print ("Current ", error_rms)
        if(prev < error_rms):
            counter = counter + 1;
            #print('--------------------------------------------------')
            #print('counter', counter)
            if(p == counter):
                print("Convergence Achieved At Iteration= ", epoch)
                print("***** Validation Set Gradient Descent *****")
                print("Error RMS",prev)
                return weights.flatten()
        if(prev>error_rms):
            counter = 0
            prev = error_rms[0]
        #print ("Error ",np.linalg.norm(E))
        #print ("Weights ",weights)
    return weights.flatten()
```

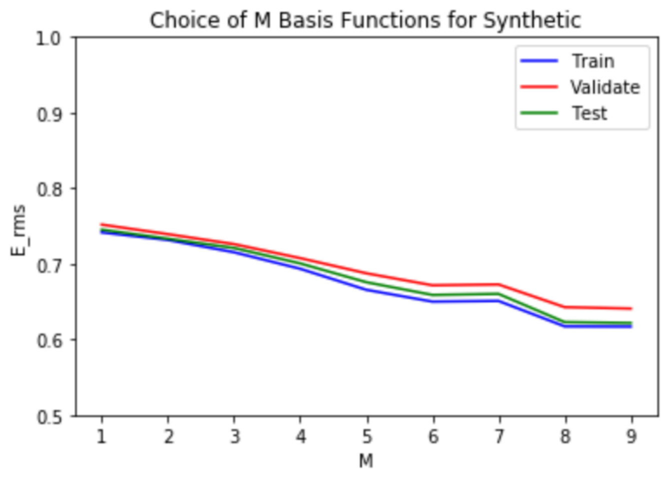
1.6 Hyper Parameter Selection: Choice of M

A mentioned above, we would like to choose M<46 such that the E_{RMS} is the smallest. A parameter space was defined such that M is in the interval [1, 46] for LeTOR data and [1,10] for synthetic data. The values of M by E_{RMS} are found in the plot below. The E_{RMS} values were recorded for each M and the minimum E_{RMS} was selected. A small constant λ=.1 was chosen.

For LeTOR, The optimal value of E_{RMS} was when M=45 (E_{RMS_train}=0.543, $E_{RMS\_validation}$=0.540) From the plot, you can see that as we increase the capacity, M-1, of the model, E_{RMS} training decreases gradually. However, increasing capacity can be computationally expensive and negates the use of radial basis functions for data-matrix reduction purposes. In addition, increasing M does not decrease E_{RMS} significantly. Therefore, in order to limit computation time and complexity, M=10 (plus 1 for bias basis function) was chosen since it decrease significantly after that for the LeTOR dataset. (E_{RMS_train}=0.551, $E_{RMS\_validation}$=0.542)

For synthetic data, the optimal value of E_{RMS} was when M=9 (E_{RMS_train}=0.617, $E_{RMS\_validation}$=0.641) From the plot, you can see that as we increase the capacity, M-1, of the model, E_{RMS} training decreases gradually other than at M=7. In addition, increasing to M=9 does not decrease E_{RMS} significantly. So, for data-matrix reduction purposes and since there is the increase at 7, we shall choose M=6+1 (plus 1 for bias basis function) for the synthetic dataset (E_{RMS_train}=0.649, $E_{RMS\_validation}$=0.622)



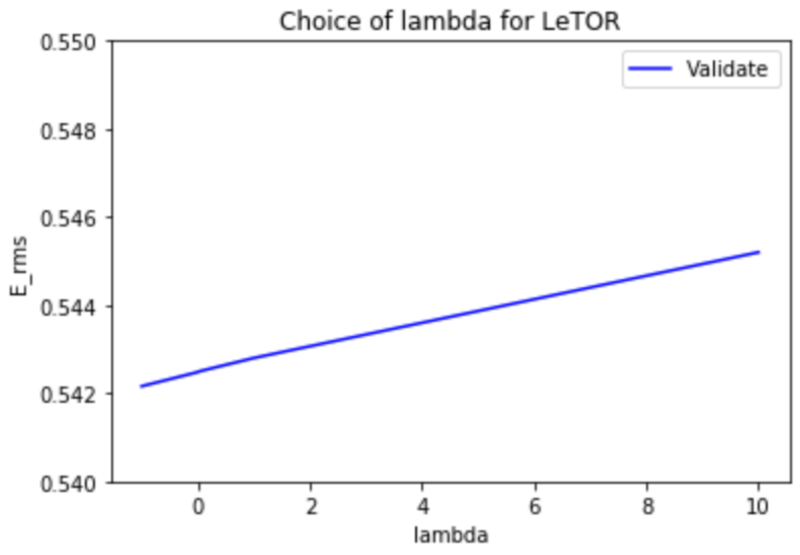


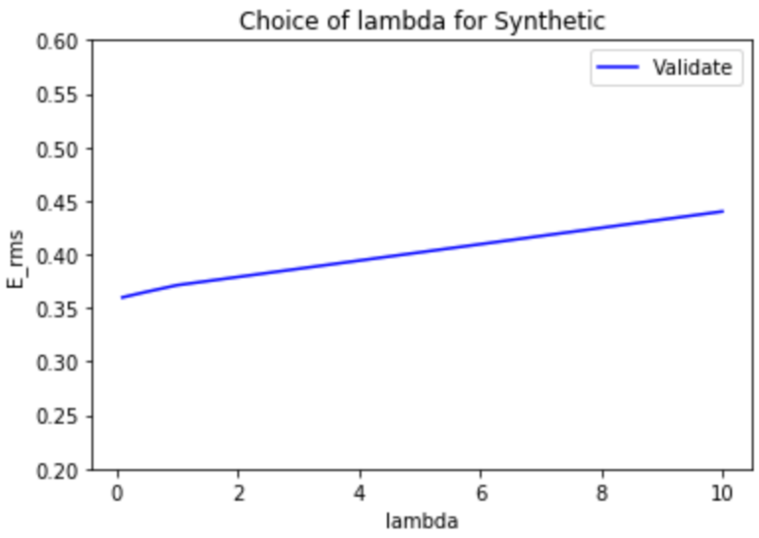
1.7 Hyperparameter Selection: Tuning λ using Validation Set

1.7.1 Closed Form Solution

Another method to control for overfitting is by using a regularization term. Here, we have chosen to use weight decay such that $E(w) = \frac{1}{2}w^T w$ and a coefficient λ. A parameter space was defined such that $\lambda=$ [0.1,0.3,0.5,1,10] for both LeTOR datasets and Synthetic data and evaluated using the cost $E_{rms\_validation}$

For the LeTOR dataset, M=10+1(for bias) number of basis functions, $E_{rms_validation}$is lowest for $\lambda = .1$ ($E_{rms\_validation}$=0.54248). Therefore, we will take $\lambda = .1$. For the synthetic dataset, M=6+1(for bias) number of basis functions, $E_{rms_validation}$is lowest for $\lambda = .1$ ($E_{rms\_validation}$=0.54248). Therefore, we will take $\lambda = .1$.





Plotting just the $E_w(w) = \frac{1}{2} w^T w$ term against λ term, we get out smallest $\lambda = .1$ also.

1.7.2 Stochastic Gradient Descent

For Stochastic Gradient Descent, Lambda (λ), in order to avoid over fitting of training and validating data, we used the same parameter space of λ values defined above in error function calculation. And as per our result, we observed good fit, meaning validation error was low, but slightly higher than the training for lambda=0.1.

1.8 Hyperparameter Selection: Tuning η for Gradient Descent using Validation Set

We find the optimal value of learning rate (η) = 0.2. below this value after calculating ERMS using Early stop we could not achieve convergence which in turn gave higher ERMS value.

1.9 Hyperparameter Selection: Tuning p, patience for Gradient Descent using Validation Set

Patience (P) is used as early stop criteria to find minima in the gradient descent function. We varied the value of P in [10,20,50,100,1000], but we got all the corresponding E_{RMS} values similar and the minima didn't change much. So, we used P=20 as our optimal value for calculations.

2 Results

2.1 LeTOR Model Results

2.1.1 Closed Form Solution:

Taking M=10+1 basis functions with $\phi_0 \equiv 1$ of the design matrix for the bias term and $\lambda = 0.1$ for our regularization term.

Our weights for the model are as follows:

$$\boldsymbol{w} = [\mathbf{0.257}, \mathbf{-0.184}, \mathbf{0.12}, \mathbf{0.014}, \mathbf{0.014}, \mathbf{0.542}, \mathbf{0.045}, \mathbf{0.165}, \mathbf{-0.783}, \mathbf{0.291}, \mathbf{0.332}]$$

with some of the basis functions providing negative weights indicating a decrease in target rank when all other basis functions are held constant.

Evaluating our solution using the test error, $E_{RMS\_testing}(w) = 0.63$, showing a small difference (0.08) between $E_{RMS\_testing}(w)$ and $E_{RMS\_training}(w)$.

$E_{training}(w) = 8{,}569.9$ with $E_{RMS\_training}(w) = 0.555$

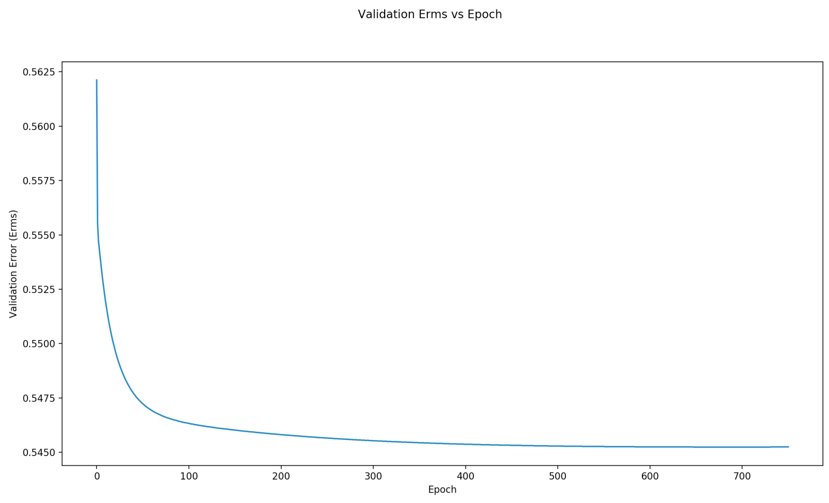
$E_{validation}(w) = 1{,}035.7$ with $E_{RMS\_validation}(w) = 0.546$

$E_{test}(w) = 1{,}380.9$ with $E_{RMS\_testing}(w) = 0.63$

2.1.2 Stochastic Gradient Descent Solution:

Taking M=10+1 basis functions with $\phi_0 \equiv 1$ of the design matrix for the bias term, $\lambda = 0.1$ for our regularization term, $\eta = 0.5$ and patience=20 and converged in 750 iterations.

Below is a plot of epochs versus $E_{RMS\_validation}(w)$. As we can see our stochastic gradient descent algorithm decreases quickly, especially in the first few iterations, and converges after 750 iterations. Since we have taken our mini-batch size equal to the sample size and have calculated a batch gradient descent algorithm, we have converged to a minimum.



Our weights for the stochastic gradient descent model where as follows:

$$\boldsymbol{w} = [\mathbf{0.268}, \mathbf{-0.004}, \mathbf{0.13}, \mathbf{-0.059}, \mathbf{0.038}, \mathbf{0.421}, \mathbf{0.053}, \mathbf{0.142}, \mathbf{-0.631}, \mathbf{0.225}, \mathbf{0.268}]$$

with some of the basis functions providing negative weights indicating a decrease in target rank when all other basis functions are held constant. It is interesting to note that w_3 changes in coefficient sign when comparing the closed form solution to the stochastic gradient descent solution. The coefficients all have similar magnitude.

Evaluating our solution using the test error, $E_{RMS\_testing}(w) = 0.63$, showing a small difference (0.085) between $E_{RMS\_testing}(w)$ and $E_{RMS\_validation}(w)$, which is similar to the closed form solution. Although error is slightly high and a better model may exist, the model fits the test data well and can be generalized.

$E_{RMS\_validation}(w) = 0.545$

$E_{RMS\_testing}(w) = 0.63$

2.2 Synthetic Data Results

2.2.1 Closed Form Solution

Taking M=6+1 basis functions with $\phi_0 \equiv 1$ of the design matrix for the bias term and $\lambda = 0.1$ for our regularization term.

Our weights for the model are as follows:

$$\boldsymbol{w} = [\mathbf{-1.44}, \mathbf{-0.719}, \mathbf{4.017}, \mathbf{0.452}, \mathbf{0.238}, \mathbf{1.775}, \mathbf{-1.829}]$$

with some of the basis functions providing negative weights indicating a decrease in target rank when all other basis functions are held constant.

Evaluating our solution using the test error, $E_{RMS\_testing}(w) = 0.63$, showing a small difference (0.006) between $E_{RMS\_testing}(w)$ and $E_{RMS\_training}(w)$.

$E_{training}(w) = 3325.45$ with $E_{RMS\_training}(w) = 0.6447$

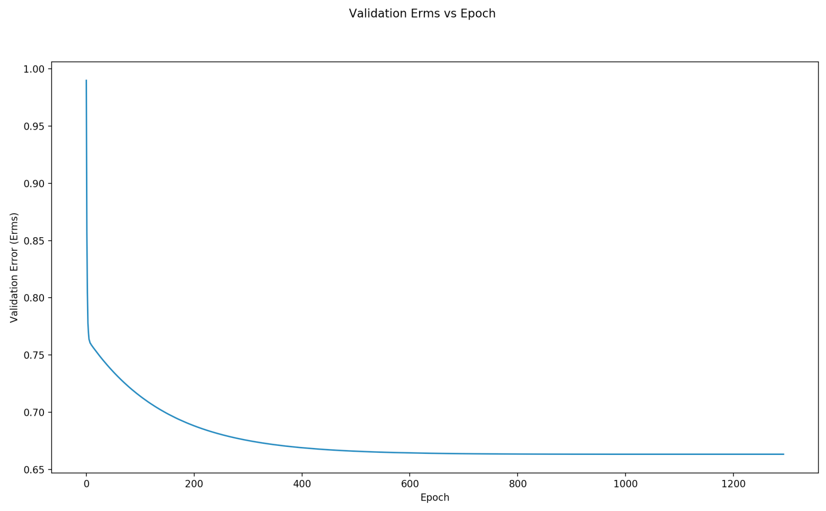
$E_{validation}(w) = 439.719$ with $E_{RMS\_validation}(w) = 0.663$

$E_{test}(w) = 423.232$ with $E_{RMS\_testing}(w) = 0.651$

2.2.2 Stochastic Gradient Descent Solution:

Taking M=6+1 basis functions with $\phi_0 \equiv 1$ of the design matrix for the bias term, $\lambda = 0.1$ for our regularization term, $\eta = 0.5$ and patience=20 and converged in 1,293 iterations.

Below is a plot of epochs versus $E_{RMS\_validation}(w)$. As we can see our stochastic gradient descent algorithm decreases quickly in the first few iterations and slows down in comparison to the plot for LeTOR gradient descent, hence why the number of iterations is larger. Since we have taken our mini-batch size equal to the sample size and have calculated a batch gradient descent algorithm, we have converged to a minimum.



Our weights for the stochastic gradient descent model where as follows:

$$\boldsymbol{w} = [\mathbf{-1.434, -0.694, 3.968, 0.44, 0.237, 1.756, -1.78}]$$

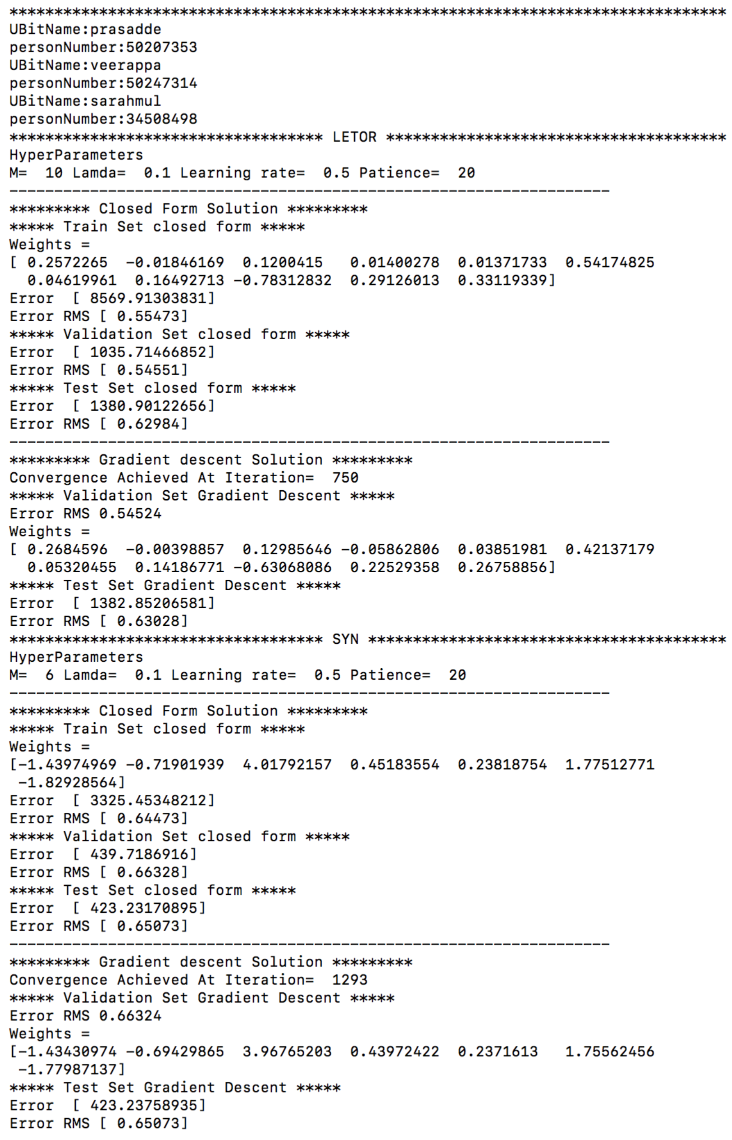
with some of the basis functions providing negative weights indicating a decrease in target rank when all other basis functions are held constant. The coefficients all have similar magnitude when comparing the closed form solution to the synthetic data results. In addition, all of the signs for the individual weights are the same.

Evaluating our solution using the test error, $E_{RMS\_testing}(w) = 0.651$ and $E_{testing}(w) = 423.238$ showing a small difference (0.12) between $E_{RMS\_testing}(w)$ and $E_{RMS\_validation}(w)$, which is similar to the closed form solution. Here, E_{RMS} validation error is slightly lower than E_{RMS} testing error. Although error is slightly high and a better model may exist, the model fits the test data well and can be generalized.

$$E_{RMS\_validation}(w) = 0.663$$

$$E_{RMS\_testing}(w) = 0.651$$

3 Final Output



```
**********************************************************************************
UBitName:prasadde
personNumber:50207353
UBitName:veerappa
personNumber:50247314
UBitName:sarahmul
personNumber:34508498
************************************ LETOR **************************************
HyperParameters
M=  10 Lamda=  0.1 Learning rate=  0.5 Patience=  20
-------------------------------------------------------------------
********* Closed Form Solution *********
***** Train Set closed form *****
Weights =
[ 0.2572265  -0.01846169  0.1200415   0.01400278  0.01371733  0.54174825
  0.04619961  0.16492713 -0.78312832  0.29126013  0.33119339]
Error  [ 8569.91303831]
Error RMS [ 0.55473]
***** Validation Set closed form *****
Error  [ 1035.71466852]
Error RMS [ 0.54551]
***** Test Set closed form *****
Error  [ 1380.90122656]
Error RMS [ 0.62984]
-------------------------------------------------------------------
********* Gradient descent Solution *********
Convergence Achieved At Iteration=  750
***** Validation Set Gradient Descent *****
Error RMS 0.54524
Weights =
[ 0.2684596  -0.00398857  0.12985646 -0.05862806  0.03851981  0.42137179
  0.05320455  0.14186771 -0.63068086  0.22529358  0.26758856]
***** Test Set Gradient Descent *****
Error  [ 1382.85206581]
Error RMS [ 0.63028]
************************************ SYN ****************************************
HyperParameters
M=  6 Lamda=  0.1 Learning rate=  0.5 Patience=  20
-------------------------------------------------------------------
********* Closed Form Solution *********
***** Train Set closed form *****
Weights =
[-1.43974969 -0.71901939  4.01792157  0.45183554  0.23818754  1.77512771
 -1.82928564]
Error  [ 3325.45348212]
Error RMS [ 0.64473]
***** Validation Set closed form *****
Error  [ 439.7186916]
Error RMS [ 0.66328]
***** Test Set closed form *****
Error  [ 423.23170895]
Error RMS [ 0.65073]
-------------------------------------------------------------------
********* Gradient descent Solution *********
Convergence Achieved At Iteration=  1293
***** Validation Set Gradient Descent *****
Error RMS 0.66324
Weights =
[-1.43430974 -0.69429865  3.96765203  0.43972422  0.2371613   1.75562456
 -1.77987137]
***** Test Set Gradient Descent *****
Error  [ 423.23758935]
Error RMS [ 0.65073]
```