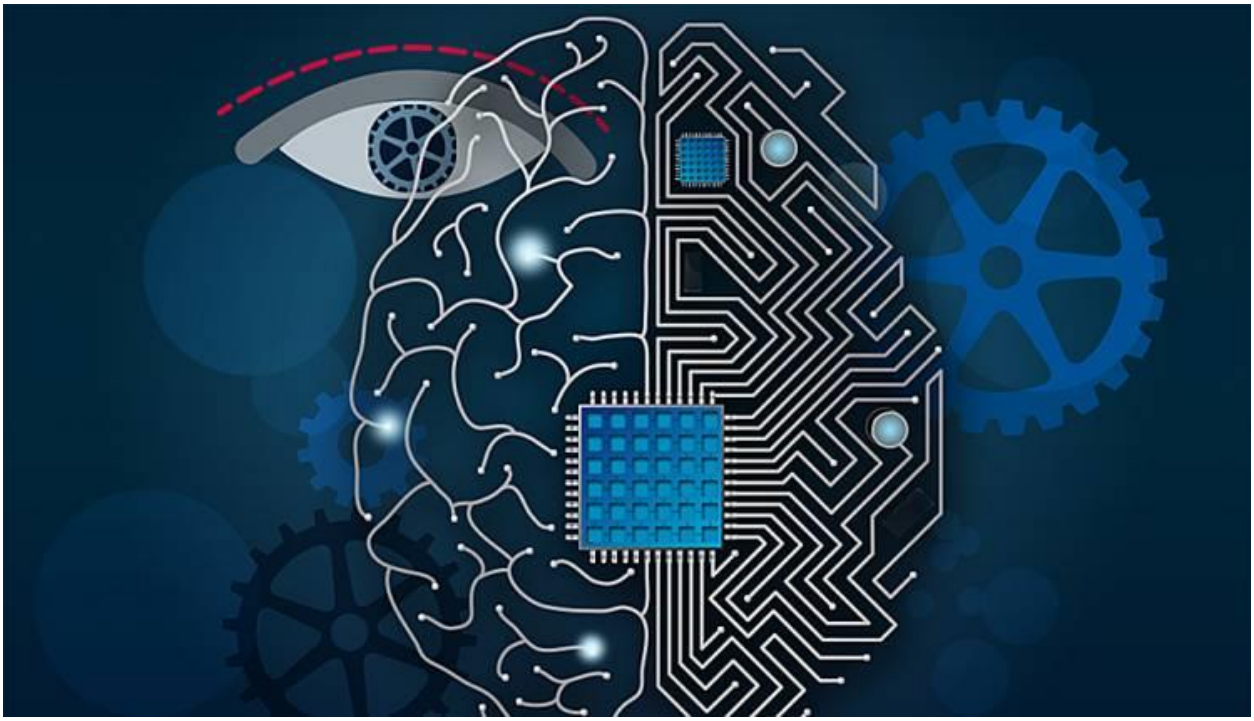
LETOR

Learning to Rank with Machine Learning



Devashish Bharadwaj

UB# 50096815

Contents

The Models

I have used three machine learning models for the given dataset. Here I will list them and discuss them in detail.

Closed-Form Solution

Basis function

The closed-form solution is the simplest type of regression model. For the closed form solution I have created a NxM size matrix. Where N stands for number of training samples and M stands for the number of Basis functions. We call this matrix the design matrix or ϕ. Here is the basis function I have used.

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu}_j)^2}{2s^2}\right)$$

The Gaussian basis function, gives one scalar value for the one row/document/observation. I have used lots of combinations for the values of the μj and s. I will list some of them.

1. μj = Vector of mean of all columns or features of documents. (size: 1xM)
   s = Vector of standard deviation of columns. (size: 1xM)

2. μj = Vector of mean of all the rows or documents. (size: 1xN)
   s = Vector of standard deviation of all rows. (size: 1xN)

3. μj = Vector of random numbers. (size: 1xM)
   s = Vector of random numbers. (size: 1xM)

I have ignored the means and standard deviation values which are zero. I have only taken meaning full values for μj and s. Through training and validation I have found that option 1 and 3 are the best. Which makes sense if you understand how the Basis function works. As μj is the amplitude/height of the Gaussian function and s is the width of Gaussian function takes. So if μj takes the mean of every known features we get one scalar value for that feature which is then marginalized by the standard deviation. To give the best result.

Random numbers also work well for the values of μj and s. The explanation I can think of is we get a variety of Basis functions as the value of M increases. The data always has some bias but if we use random numbers we get very distinct Basis functions and as the number of Basis functions increase the error becomes lesser and lesser.

Making the basis function from the row mean doesn't work well for me as we are combining different features and taking their mean.

Design Matrix

I have varied the value of **M from 2 to 20**. I first column of the Design matrix is constant and I have fixed the value to 1.

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

So as my Design matrix dimensions change I vary my calculations accordingly. As the number of Basis functions increase the complexity of the model increases. The design matrix is then used by the closed form solution to calculate the weight matrix. I also save the Design matrix to disk so I can use it later for other models. This is necessary specially in the case of random number Gaussian function. Also it makes the program run a lot faster.

Weight Vector

I have then calculated the weight vector from the Design matrix. Here is the formula for the weight vector.

$$\mathbf{w}_{ML} = (\lambda\mathbf{I} + \Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$$

I have made the Design matrix of the form of NxM and the value of t is the relevance labels of the training dataset. So we get our weight matrix which has the size of 1xM. As the weight vector is the weight assigned to the scalar values from the Gaussian functions. Lambda is there to stop over-fitting. More on lambda on the next page.

Error Function

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2$$

Next, I calculated the error of the predictions minus the actual value whole squared and then summed. Of course this is our least square error for the entire data set.

Lambda

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

The challenge is to make sure we don't over-fit our training dataset to the notion of lambda is introduced. The process is to vary the value of lambda over our calculated value of the weight vector to give us the best possible answer. Lambda provides protection for over-fitting by adding the squared and summed value of the weights to the real error.

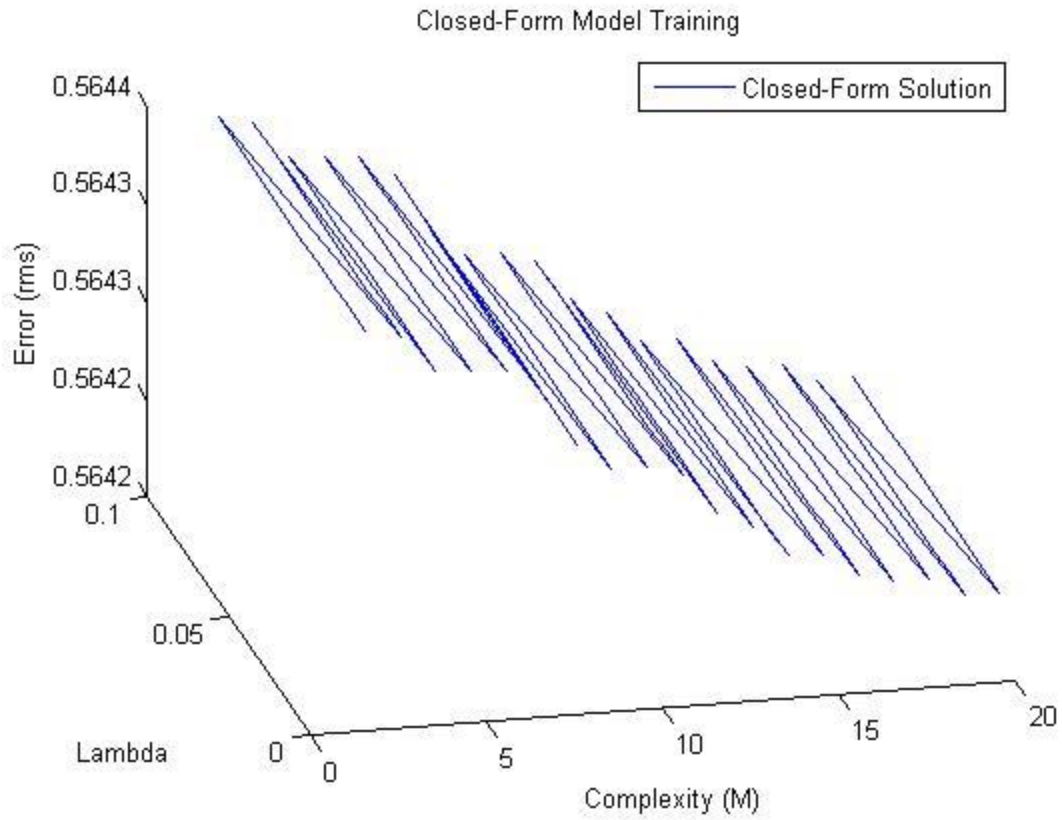
$$E_W(\mathbf{w}) = \frac{1}{2}\sum_{j=0}^{M-1}|w_j|^q$$

Thereby, making harder to over-fit as we increase the complexity of the model the lambda part of the equation increases and we can control the complexity of the model. The value of lambda should be small otherwise the weights might overplay their part in the final error calculation. I have varied the value for lambda in the project from **0.01 to 0.1.**

Error Root Mean Square

$$E_{RMS} = \sqrt{2E(\mathbf{w}*)/N_T}$$

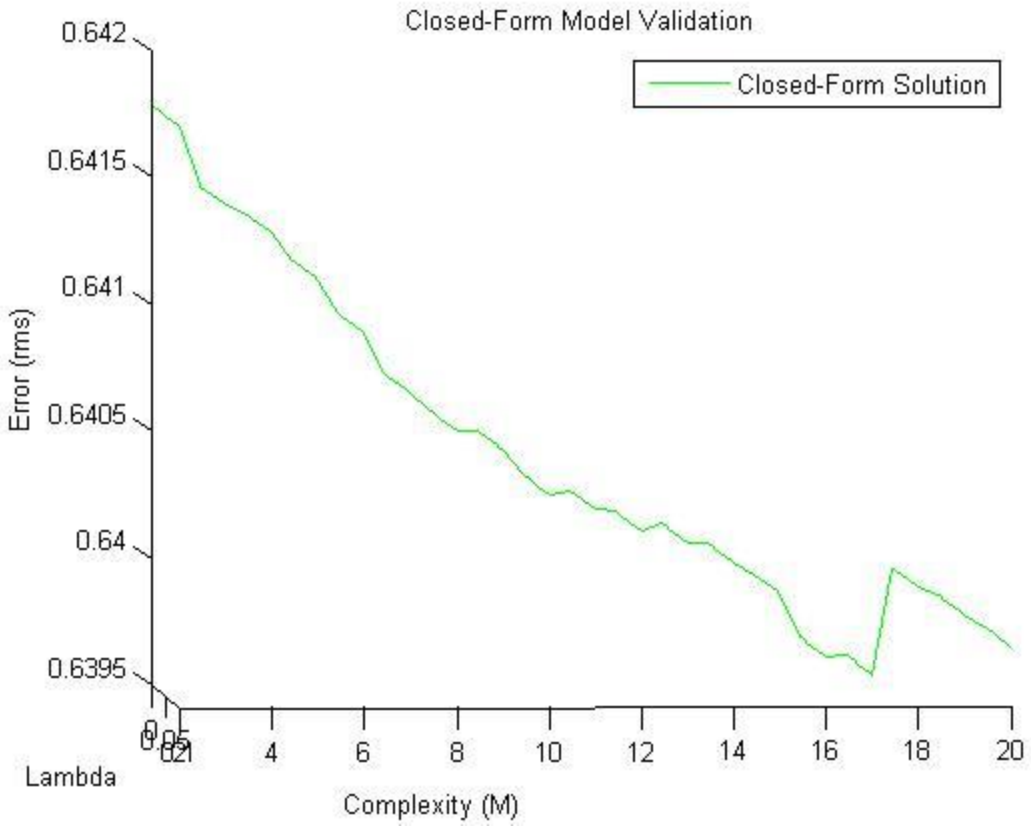
So, as to get the idea of the error per document/observation. We divide the final error by number of documents and get the Erms value. The value comes out between **0.54 to 0.56** for my closed-form model. The value is affected by the random numbers so it changes a slightly sometimes. But it does decrease as the complexity increases but only to a certain extent. After that the regularization causes the error to increase. Here is the graph for Erms X M X lambda.



We can see the error decreasing as the complexity increases. But after M=20 it doesn't decrease more. As lambda increases the error becomes slightly higher but not much. Error is minimum for minimum value of lambda. To check for complexity we run the values against the validation set.

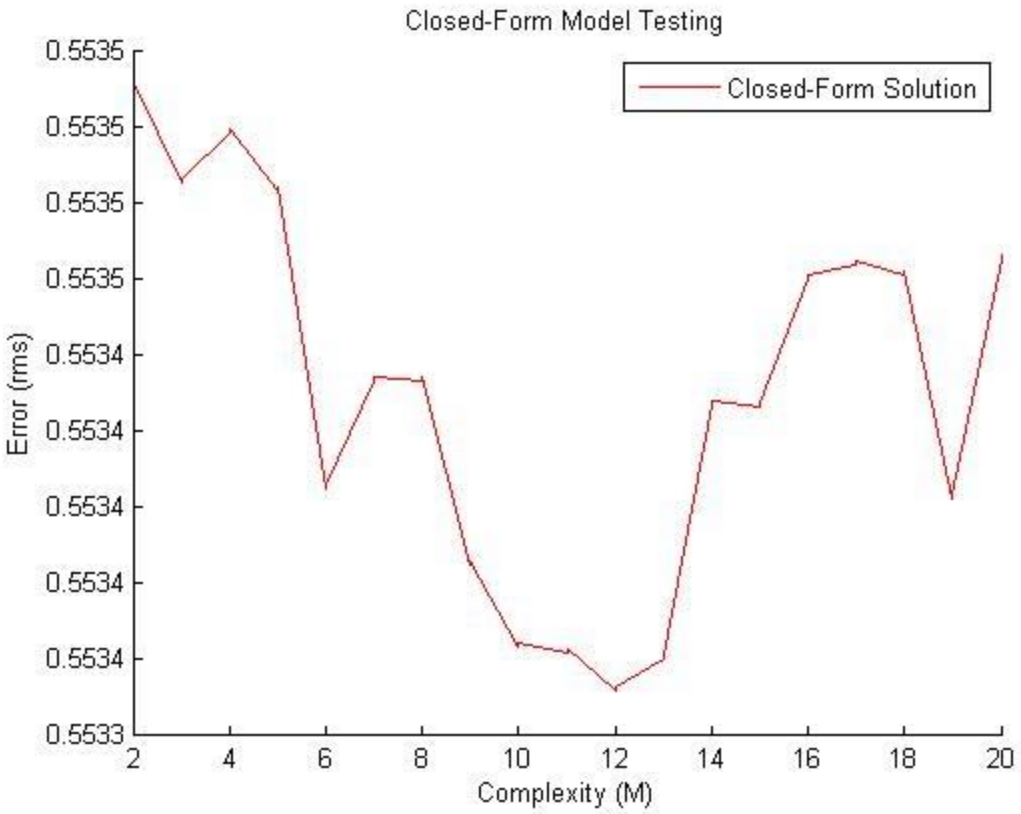
Validation

After the calculation of the weight vector we move to validation. From validation we can tell that the model complexity should be limited between **M = 14 to M = 17**. As the error begins to rise if the complexity is increased anymore. Thus regularization with lambda works correctly. After validating with the data set several times I see that we can fix the value of **M = 13** to give good result. If the value of M becomes more than that we find that the error increases.



Testing

So with all the parameters in place I go forward with testing. With the weight vector in place I passed the vector to the testing program and confirm the result. Error for training comes out to be around **Erms=0.55** and **M=13** and **lambda=0.01**.



I have done these tests may times to see that we get an average errors and values. Overall the model works and the error goes down with the complexity for a while but after certain complexity the error rate is unpredictable.

Stochastic Gradient Descent

The gradient descent algorithm takes a row from the dataset and calculates the weight vector according to that row in the dataset. Then that weight vector is applied to a subset of the rows to give some error and we go on a new row, new weights and new error till the error converges.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} + \eta(t_n - \mathbf{w}^{(\tau)T}\phi_n)\phi_n$$

Design Matrix

The design was calculated again in the same way, with varying complexity.

Weight Vector

What is different here is that the initial weight vector is of random values. It is used in the initial iteration to find the next weight vector and so on till convergence.

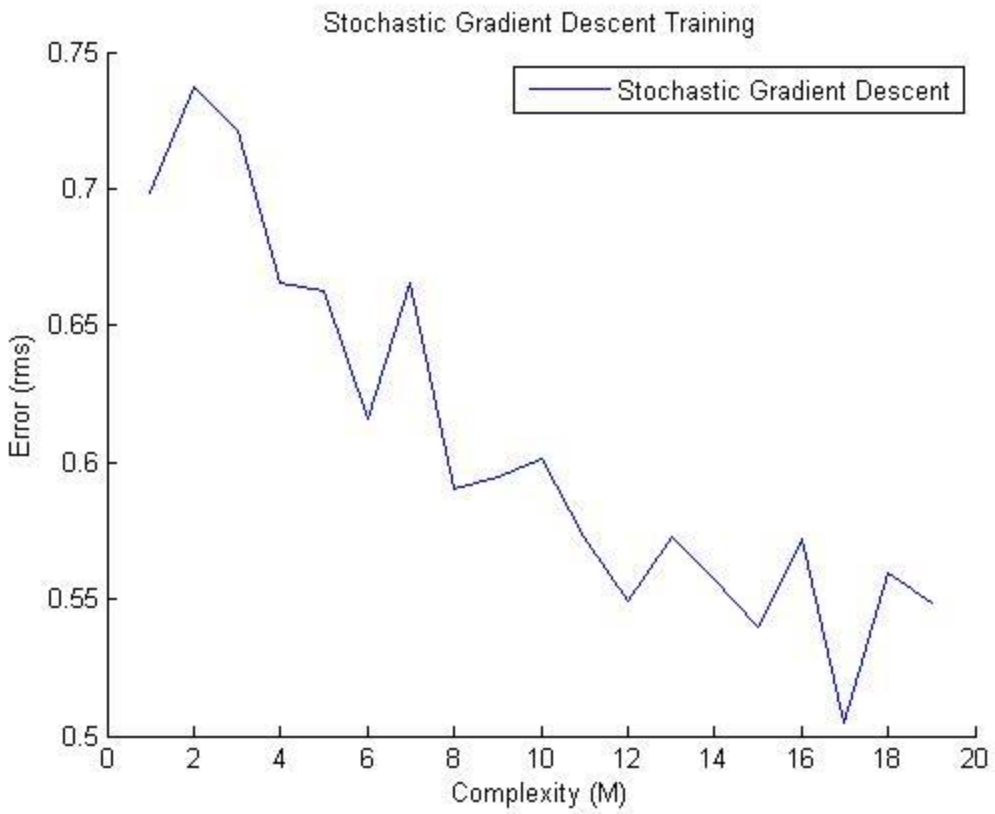
The Purpose of η

As we take steps towards convergence we do so in iterations. η helps measuring those steps. We take big steps we can overshoot our target and if our steps are too small it takes a long time for convergence.

The Loop to Convergence

So, how do I know if I have reached convergence, well if the error value doesn't change over the iteration and we stop there. Inside the loop to convergence I select a random row and then calculate the error over **20,000 rows**. Why 20,000? I started with 5,000 but as I increase the graph has a smoothing effect which wears if I increase the number of rows. Error is calculated with Erms formula and fluctuates wildly so this is the only parameter which helps with the problem.

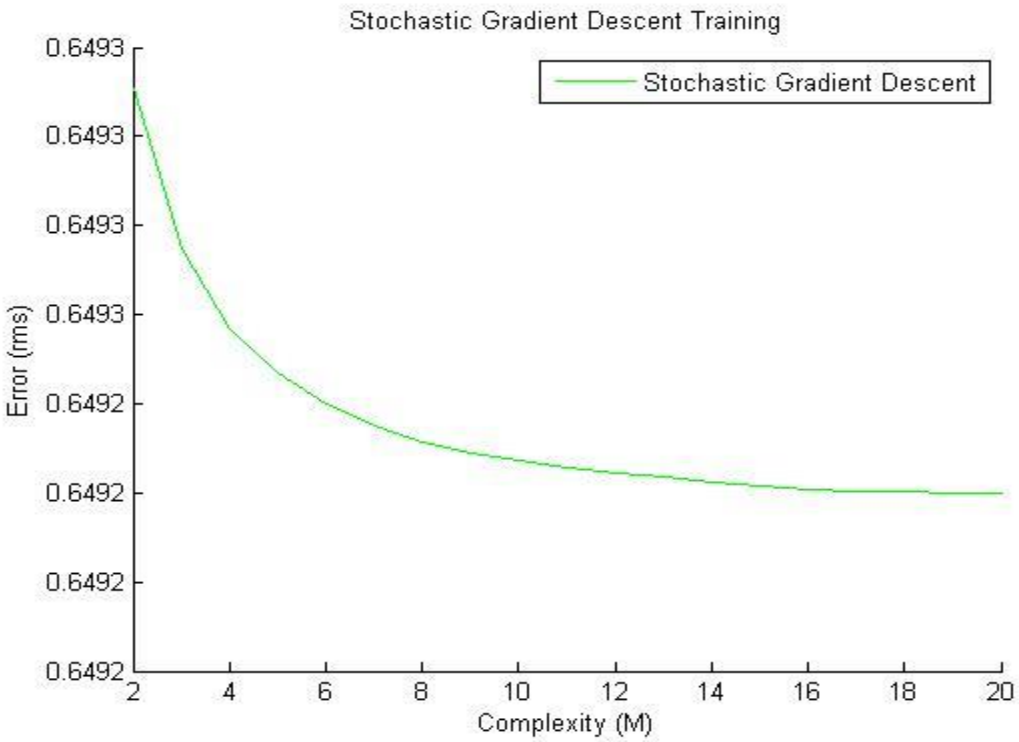
Here is the graph for the training set.



As we see there is a general trend of reducing the error with complexity but the Erms fluctuates wildly between consecutive values of M. Choosing the right number of rows to calculate the error on helps to smooth the graph.

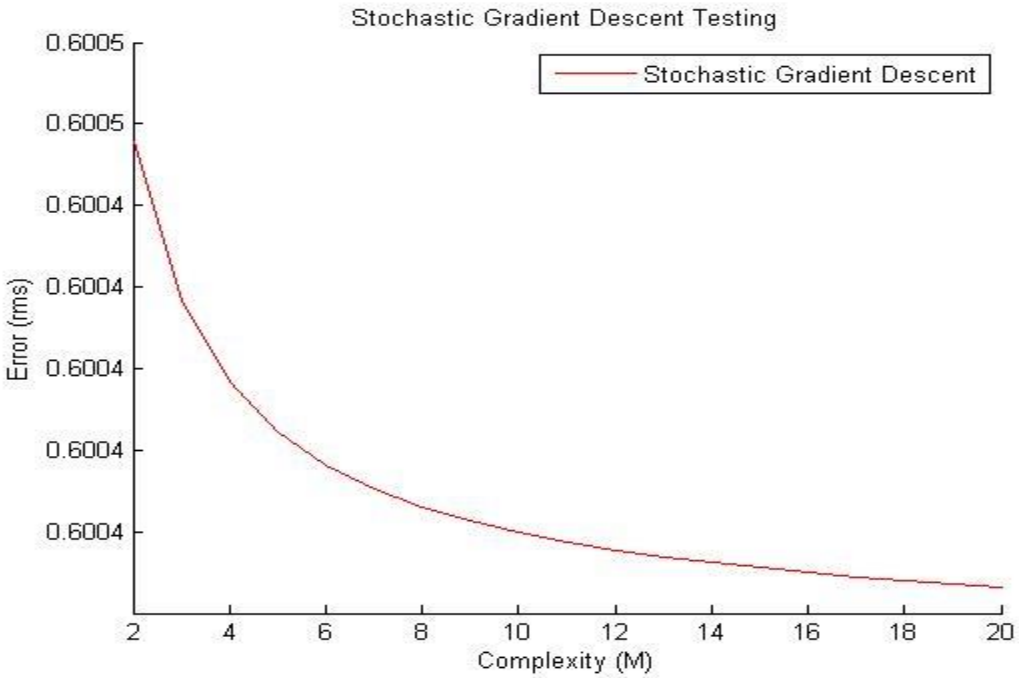
Validation

Validation gives a very smooth graph for an error set of 20,000 rows. In fact the error doesn't vary much at all. I would say that weight vector fits the average between the wide error range of training data set. The error range is from **0.6493 to 0.6492**. I haven't done any regularization as I wasn't sure if that is required in gradient descent after all we are trying to compute the weight vector using one row. Not sure how regularization would go by validation seems to pass the test.



Testing

Pretty much confirms validation and a lower error value.



Bayesian Linear Regression

The Bayesian method doesn't concern itself with finding out the weights for the problem. It tried to predict the target values given certain parameters. Here is the step by step process.

$$p(t|\mathbf{x}, \mathbf{t}, \alpha.\beta) = N(t|\mathbf{m}_N^T \phi(\mathbf{x}), \sigma_N^2(\mathbf{x}))$$

The α and β

All equations required by the Bayesian method require these two parameters. So we take random values for α and β and iterate over the equations which give new values of α and β and so on till we converge.

Hessian Matrix

$$\mathbf{A} = \alpha\mathbf{I} + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$$

The Hessian can be calculated from alpha and beta which in turn can be used to calculate E(Mn). The Hessian is also required to calculate the conjugate and final error value but is only useful with correct values of α and β.

From Mn to E(Mn)

$$\mathbf{m}_N = \beta\mathbf{A}^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}.$$

Another new variable introduced in the mix is the Mn which helps to calculate the error and also gives the new value of α. The following equation helps to calculate α from Mn.

$$\alpha = \frac{\gamma}{\mathbf{m}_N^{\mathrm{T}}\mathbf{m}_N}$$

A New Variable γ

We use γ to calculate α. But first we need to calculate γ. We do so with the following equation.

$$\gamma = \sum_i \frac{\lambda_i}{\alpha + \lambda_i}$$

To Lambda and Beyond

This is where the story of new variables ends. And we get lambda from old ones. Lambda is the eigen value vector of β and design matrix transpose multiplied by itself.

$$\left(\beta \mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}\right) \mathbf{u}_i = \lambda_i \mathbf{u}_i$$

Back to α and β

We calculate the new values of α and β and iterate over the loop again till the values stop changing. I have already shared the equation for α, here is the equation for β.

$$\frac{1}{\beta} = \frac{1}{N - \gamma} \sum_{n=1}^{N} \left\{ t_n - \mathbf{m}_N^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\}^2$$

The Random and Unexpected

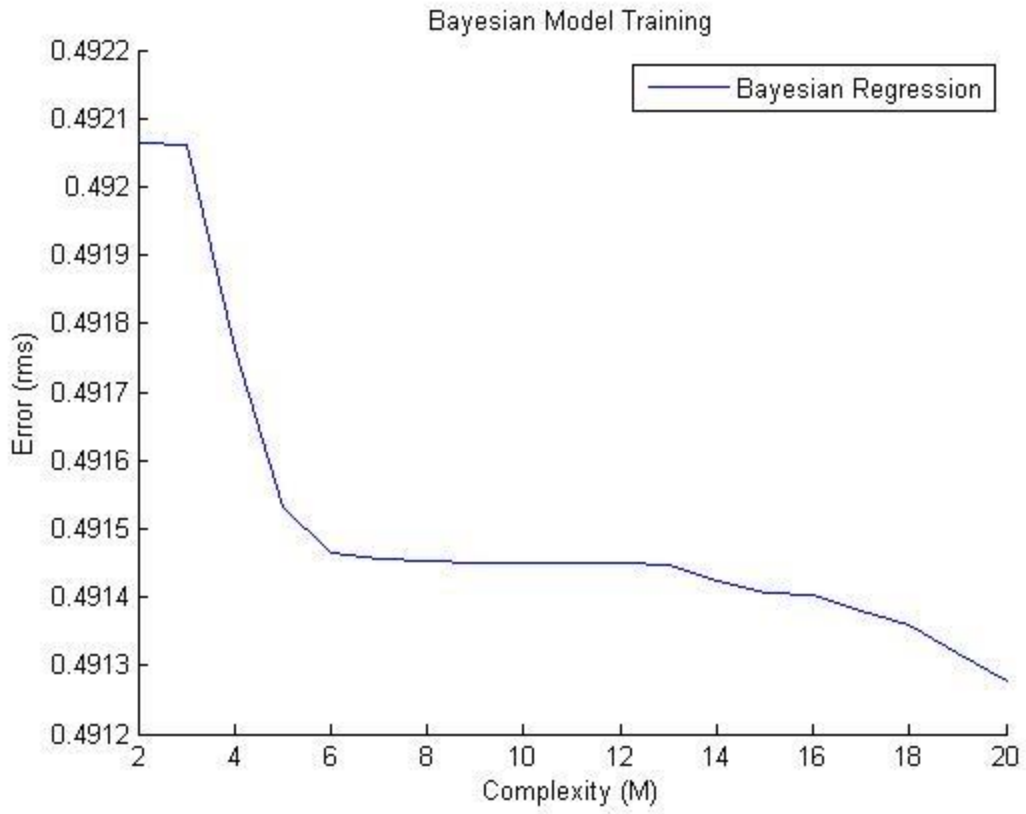
Unfortunately, the values of α and β don't converge but oscillate between two pairs of values. So, I added a random number to both of them at the end of iteration. This random jump help converge to the true values of α and β. And because I introduce a random number I compute the values of α and β 20 times and take their average.

The Result

To get the result I calculate the Hessian and Em values and E(Mn) values as per the formulas. And to make sure I also take the conjugate. I calculate the error by taking the Erms of the E(Mn) value.

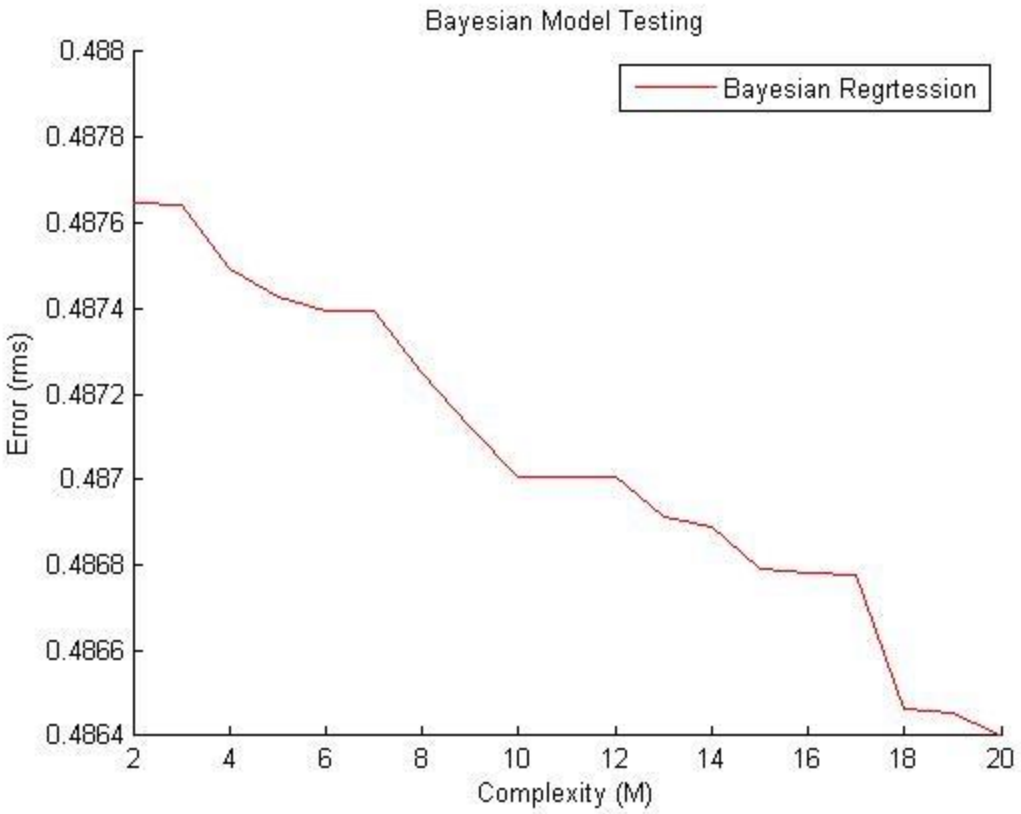
$$E(\mathbf{m}_N) = \frac{\beta}{2} \left\| \mathbf{t} - \mathbf{\Phi} \mathbf{m}_N \right\|^2 + \frac{\alpha}{2} \mathbf{m}_N^{\mathrm{T}} \mathbf{m}_N .$$

Here is the Error X Complexity graph.



Testing

As there is no validation required by Bayesian approach. I do the testing with alpha and beta values. The values vary marginally as I use random values to calculate alpha and beta. But graph curve remains the same and the error becomes less with complexity.



The Bayesian model gives the lowest error out of all three more in the performance section.

Performance

Closed-Form Solution

E(rms) = 0.54 to 0.56

Lambda(Λ) = 0.01

Complexity(M) = 13

Stochastic Gradient Descent

E(rms) = 0.58 to 0.61

Eta(η) = 0.000000032

Complexity(M) = 20

Bayesian Linear Regression

E(rms) = 0.49 to 0.52

Conjugacy = -8.1482e+03