

Q1) a) Let $X_1, X_2, X_3, \dots, X_n$ be i.i.d Gaussian random variables with mean $= \theta$ variance σ^2

A Gaussian distribution is expressed as:

$$j(k|\mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}} \times e^{-\frac{(k-\mu)^2}{2\sigma^2}}$$

$$\theta_{\text{MAP}}(x) = \underset{\theta}{\operatorname{argmax}} (j(\theta|x)) = \underset{\theta}{\operatorname{argmax}} j(x|\theta) g(\theta)$$

θ is selected from a gaussian distribution
 $N(\mu, \sigma^2) = g(\theta)$

$$\theta_{\text{MAP}}(X_1, X_2, \dots, X_n) = \underset{\theta}{\operatorname{argmax}} j(X_1, X_2, \dots, X_n|\theta) g(\theta)$$

Since $X_1, X_2, X_3, \dots, X_n$ are i.i.d

$$= \underset{\theta}{\operatorname{argmax}} j(x_1|\theta) j(x_2|\theta) \times j(x_3|\theta) \dots j(x_n|\theta) g(\theta)$$

$$\Rightarrow j(x_i|\theta) = j(x_i|\theta, \sigma^2) = N(\theta, \sigma^2)$$

$$\theta_{\text{MAP}} = \underset{\theta}{\operatorname{argmax}} \quad \underbrace{f(x_1 | \theta \sigma^2)}_{f(x_1 | \theta \sigma^2)} \underbrace{f(x_2 | \theta \sigma^2)}_{f(x_2 | \theta \sigma^2)} \dots \underbrace{f(x_n | \theta \sigma^2)}_{f(x_n | \theta \sigma^2)} \times g(\theta) \quad - (1)$$

$$= \frac{1}{\sqrt{2\sigma_0^2\pi}} \times e^{-\frac{(x_1 - \theta)^2}{2\sigma_0^2}} \times \frac{1}{\sqrt{2\sigma_0^2\pi}} \times e^{-\frac{(x_2 - \theta)^2}{2\sigma_0^2}} \dots$$

$$\dots \frac{1}{\sqrt{2\sigma_0^2\pi}} \times e^{-\frac{(x_n - \theta)^2}{2\sigma_0^2}} \times \frac{1}{\sqrt{2\sigma_0^2\pi}} \times e^{-\frac{(\theta - \mu)^2}{2\sigma_0^2}}$$

$$= \frac{1}{(2\sigma_0^2\pi)^{n/2}} \prod_{i=1}^n e^{-\frac{(x_i - \theta)^2}{2\sigma_0^2}} \times \frac{1}{\sqrt{2\sigma_0^2\pi}} e^{-\frac{(\theta - \mu)^2}{2\sigma_0^2}}$$

$$\Rightarrow p' = \frac{1}{(2\sigma_0^2\pi)^{n/2}} \times \frac{1}{(2\sigma_0^2\pi)^{1/2}}$$

$$\Theta_{MAP} = p' \times \left(\prod_{i=1}^n e^{-\frac{(x_i - \Theta)^2}{2\sigma_0^2}} \right) \times e^{-\frac{(\Theta - \mu)^2}{2\sigma^2}}$$

Taking log on both sides

$$\log \Theta_{MAP} = \log p' + \sum_{i=1}^n -\frac{(x_i - \Theta)^2}{2\sigma_0^2} + \frac{-(\Theta - \mu)^2}{2\sigma^2}$$

Differentiating with respect to Θ

$$\hat{\Theta}_{MAP} = 0 + \nabla \sum_{i=1}^n -\frac{(x_i - \Theta)^2}{2\sigma_0^2} + \nabla \frac{-(\Theta - \mu)^2}{2\sigma^2}$$

$$= -\nabla \sum_{i=1}^n \frac{(x_i - \Theta)^2}{2\sigma_0^2} + \nabla \frac{-(\Theta - \mu)^2}{2\sigma^2}$$

$$= \sum_{i=1}^n \frac{x_i - \Theta}{\sigma_0^2} - \frac{(\Theta - \mu)}{\sigma^2}$$

$\hat{\Theta}_{MAP} = 0$ To find max/minimum value of Θ .

$$\sum_{i=1}^n \frac{x_i - \theta}{\sigma^2} = \frac{\theta - \mu}{\sigma^2}$$

$$\sum_{i=1}^n \frac{x_i}{\sigma^2} - \sum_{i=1}^n \frac{\theta}{\sigma^2} = \frac{\theta - \mu}{\sigma^2}$$

$$\sum_{i=1}^n \frac{x_i}{\sigma^2} + \frac{\mu}{\sigma^2} = \frac{n\theta}{\sigma^2} + \frac{\theta}{\sigma^2}$$

$$\hat{\theta}_{MAP}^* = \left(\left[\sum_{i=1}^n \frac{x_i}{\sigma^2} \right] + \frac{\mu}{\sigma^2} \right) / \left(\frac{n}{\sigma^2} + \frac{1}{\sigma^2} \right)$$

$$\hat{\theta}_{MAP} = \sum_{i=1}^n \frac{x_i - \theta}{\sigma^2} - \frac{(\theta - \mu)}{\sigma^2}$$

Differentiating $\hat{\theta}_{MAP}$ again to find whether θ^* is max or min for θ_{MAP}

$$\hat{\hat{\theta}}_{MAP} = \frac{\partial}{\partial \theta} \left(\sum_{i=1}^n \frac{x_i - \theta}{\sigma^2} - \frac{(\theta - \mu)}{\sigma^2} \right)$$

$$\hat{\theta}_{MAP} = \sum_{i=1}^n -\frac{1}{b_0^2} - \frac{1}{b^2}$$

$$= -\frac{n}{b_0^2} - \frac{1}{b^2}$$

Since $b_0^2 > 0$, $b^2 > 0$ and $n > 0$

$\hat{\theta}_{MAP} < 0$ (-ve) so θ_{MAP}^* is a max solution.

Q2) b) Laplace distribution

$$p(x) = \frac{1}{2b} e\left(\frac{-|x-\mu|}{b}\right)$$

Since $\mu = 0$ and $x = 0$

$$p(x) = \frac{1}{2b} e\left(\frac{-|x|}{b}\right)$$

$$p(0) = \frac{1}{2b} e^{-|0|/b} = g(0)$$

Using equation (1) from question ((1)a)

$$\Theta_{MAP} = \underset{\Theta}{\operatorname{argmax}} \prod (x_i | \Theta \sigma^2) \prod (x_n | \Theta \sigma^2) \dots \prod (x_n | \Theta \sigma^2) g(\Theta)$$

$$= \frac{1}{(2\sigma^2\pi)^{n/2}} \prod_{i=1}^n e^{-\frac{(x_i - \Theta)^2}{2\sigma^2}} \times \frac{1}{2b} \times e^{-|\Theta|/b}$$

$$p' = \frac{1}{(2\sigma^2\pi)^{n/2}} \times \frac{1}{2b}$$

$$= p' \times \prod_{i=1}^n e^{-\frac{(x_i - \Theta)^2}{2\sigma^2}} \times e^{-|\Theta|/b}$$

Taking log on both sides

$$\log \Theta_{MAP} = \log p' + \sum_{i=1}^n -\frac{(x_i - \Theta)^2}{2\sigma^2} + \frac{-|\Theta|}{b}$$

Differentiating with respect to Θ to find maximum solution for Θ_{MAP}^*

$$\hat{\Theta}_{MAP} = 0 + \sum_{i=1}^n \frac{x_i - 0}{6_0^2} + \nabla - \frac{|0|}{6}$$

$$= \sum_{i=1}^n \frac{x_i - 0}{6_0^2} - \frac{1}{6} \frac{|0|}{0} = 0$$

$\hat{\Theta}_{MAP}$ is set to 0 to find the max min solution for 0

$$\sum_{i=1}^n \frac{x_i}{6_0^2} - \frac{1}{6} \frac{|0|}{0} = \sum_{i=1}^n \frac{0}{6_0^2}$$

$$\Theta_{MAP}^* = \left(\sum_{i=1}^n \left(\frac{x_i}{6_0^2} \right) - \frac{1}{6} \frac{|0|}{0} \right) \frac{6_0^2}{n}$$

Θ_{MAP}^* is not a closed form solution thus we will need to use an iterative approach to obtain the solution.

Using Newton Raphson's Method to find an iterative approach to find the root of Θ_{MAP}^* where

$$\Theta_{MAP}^* = 0$$

$$x_{n+1} = x_n - \frac{J(x_0)}{J'(x_0)}$$

$$J'(x_0) = \nabla \hat{\Theta}_{MAP}^* = \hat{\Theta}_{MAP}$$

$$J(x_0) = \hat{\Theta}_{MAP}$$

$$\nabla \hat{\Theta}_{MAP}^* = \hat{\Theta}_{MAP}$$

→ Finding this

$$\hat{\Theta}_{MAP} = \nabla \sum_{i=1}^n \frac{(x_i - \Theta)^2}{6_0^2} - \nabla \frac{1}{6} \frac{|0|}{\Theta}$$

$$= - \sum_{i=1}^n \frac{1}{6_0^2} - \frac{1}{6} \frac{|0|}{\Theta^2} + \frac{1}{6} \frac{|0|}{\Theta^2}$$

$$= - \sum_{i=1}^n \frac{1}{6_0^2}$$

$\hat{\Theta}_{MAP}$ is ~~the~~ -ve so Θ_{MAP}^+ is a max solution.

$$\left[\frac{\nabla |0| \Theta^{-1}}{\nabla \Theta} = \frac{|0|}{\Theta^2} - \frac{|0|}{\Theta^2} = |0| \nabla \Theta^{-1} + \Theta^{-1} \nabla |0| \right]$$

So Newton Raphson Method

$$\theta_{MAP, n+1}^* = \theta_{MAP, n}^* - \frac{J'(\theta)}{J''(\theta)}$$

$$= \theta_{MAP, n}^* - \frac{1}{-\sum_{i=1}^n \frac{1}{\sigma_i^2}} \times \left(\sum_{i=1}^n \left(\frac{x_i - \theta}{\sigma_i^2} \right) - \frac{1}{\sigma} \frac{|\theta|}{\theta} \right)$$

$$= \theta_{MAP, n}^* - \frac{\sigma_0^2}{n} \left(\sum_{i=1}^n \left(\frac{x_i - \theta}{\sigma_0^2} \right) - \frac{1}{\sigma} \frac{|\theta|}{\theta} \right)$$

We may use a α parameter to control the convergence of root. $\alpha = \left(-m \frac{\sigma_0^2}{n} \right)$ (m is a constant)

$$\theta_{MAP, n+1}^* = \theta_{MAP, n}^* - \alpha \hat{\theta}_{MAP}$$

Note we previously calculated ~~$\hat{\theta}_{MAP}$~~ $\hat{\theta}_{MAP}$

$$\hat{\theta}_{MAP} = \left(\sum_{i=1}^n \left(\frac{x_i - \theta}{\sigma_i^2} \right) - \frac{1}{\sigma} \frac{|\theta|}{\theta} \right)$$

Q2)c) Multivariate Gaussian is given by

$$p_X(0, \Sigma) = N(0, \Sigma) \sim X_i =$$

$$\frac{1}{(2\pi)^d |\Sigma|} e^{-\frac{1}{2}(X_i - \mu)^T \Sigma^{-1} (X_i - \mu)}$$

$$\Sigma = I \quad |\Sigma| = |I| = 1 \quad \Sigma^{-1} = I^{-1} = I$$

Using equation (1) from question (C1)a)

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \frac{p(x_1 | \theta \Sigma^2) p(x_2 | \theta \Sigma^2) \dots}{p(x_n | \theta \Sigma^2) g(\theta)}$$

$$= \underset{\theta}{\operatorname{argmax}} \frac{p(x_1 | \theta \Sigma) p(x_2 | \theta \Sigma) \dots}{p(x_n | \theta \Sigma) g(\theta)}$$

$$= \frac{1}{\sqrt{(2\pi)^d} \times 1} \times e^{-\frac{1}{2}((x_1 - \theta)^T I (x_1 - \theta))}$$

$$\times \frac{1}{\sqrt{(2\pi)^d}} \times e^{-\frac{1}{2}((x_2 - \theta)^T I (x_2 - \theta))}$$

$$\frac{1}{\sqrt{(2\pi)^d}} \times e^{-\frac{1}{2}((x_n - \theta)^T \mathbf{I} (x_n - \theta))}$$

$$\times \left(\frac{1}{(2\pi)^d} |\sigma^2 \mathbf{I}| \right)^{\frac{1}{2}} \times e^{-\frac{1}{2}(\theta^T (\mathbf{I} \sigma^2)^{-1} \theta)}$$

$$\Rightarrow g(\theta) = \frac{1}{\sqrt{(2\pi)^d} |\sigma^2 \mathbf{I}|} \times e^{-\frac{1}{2}((\theta - 0)^T (\mathbf{I} \sigma^2)^{-1} (\theta - 0))}$$

Since $\mu = 0$ and $\Sigma = \sigma^2 \mathbf{I}$

$$\theta_{\text{MAP}} = \frac{1}{((2\pi)^d)^{n/2}} \times \frac{1}{\sqrt{(2\pi)^d} |\sigma^2 \mathbf{I}|} \times \prod_{i=1}^n e^{-\frac{1}{2}((x_i - \theta)^T \mathbf{I} (x_i - \theta))} \times e^{-\frac{1}{2}(\theta^T (\mathbf{I} \sigma^2)^{-1} \theta)}$$

Taking log on both sides; $p' = \frac{1}{((2\pi)^d)^{n/2}} \times \frac{1}{\sqrt{(2\pi)^d} |\sigma^2 \mathbf{I}|}$

$$\log \theta_{\text{MAP}} = \log p' + \sum_{i=1}^n -\frac{1}{2}((x_i - \theta)^T \mathbf{I} (x_i - \theta)) + -\frac{1}{2}(\theta^T (\mathbf{I} \sigma^2)^{-1} \theta)$$

Differentiating with respect to θ

$$\hat{\theta}_{\text{MAP}} = \theta + \nabla \sum_{i=1}^n -\frac{1}{2} \left((x_i - \theta)^T \mathbf{I} (x_i - \theta) \right) \\ + \nabla -\frac{1}{2} \left(\theta^T (\mathbf{I} \sigma^2)^{-1} \theta \right)$$

$$\Rightarrow x_i - \theta \in \mathbb{R}^{d \times 1} \Rightarrow (x_i - \theta)^T \in \mathbb{R}^{1 \times d} \\ \mathbf{I} \in \mathbb{R}^{d \times d} \quad \left((x_i - \theta)^T \times \mathbf{I} \right) \in \mathbb{R}^{1 \times d} \quad \text{not} \\ = (x_i - \theta)^T$$

$$\text{So } \mathbf{A} \mathbf{I} = \mathbf{A}$$

$$\Rightarrow (\mathbf{I} \sigma^2)^{-1} = (\sigma^2)^{-1} (\mathbf{I})^{-1}$$

σ^2 is a variance parameter on the diagonal hence it is a scalar.

$$= \frac{1}{\sigma^2} (\mathbf{I})^{-1}$$

$$\mathbf{I}^{-1} = \mathbf{I}$$

$$= \frac{1}{\sigma^2} \mathbf{I}$$

$$\hat{\theta}_{MAP} = \nabla \sum_{i=1}^n -\frac{1}{2} ((x_i - \theta)^T (x_i - \theta)) \\ + \nabla -\frac{(b^2)^4}{2} (\theta^T \mathbb{I} \theta)$$

$$= \nabla \sum_{i=1}^n -\frac{1}{2} ((x_i - \theta)^T (x_i - \theta)) \\ + \nabla -\frac{1}{2b^2} (\theta^T \theta)$$

$$= \sum_{i=1}^n -\frac{1}{2} \frac{\nabla (x_i - \theta)^T (x_i - \theta)}{\nabla \theta} - \sum_{i=1}^n \frac{1}{2} (x_i - \theta)^T \frac{\nabla (x_i - \theta)}{\nabla \theta} \\ + \nabla -\frac{1}{2b^2} (\theta^T \theta)$$

$$= \sum_{i=1}^n -\frac{1}{2} -(x_i - \theta)^T - \sum_{i=1}^n \frac{1}{2} -(x_i - \theta)^T \\ + -\frac{1}{2b^2} \frac{\nabla \theta^T}{\nabla \theta} \theta - \frac{1}{2b^2} \frac{\nabla \theta}{\nabla \theta}$$

$$= \sum_{i=1}^n (x_i - \theta)^T \neq -\frac{\theta^T}{\sigma^2}$$

To find the θ_{MAP}^* we put $\hat{\theta}_{\text{MAP}} = 0$

$$\sum_{i=1}^n (x_i - \theta)^T - \frac{\theta^T}{\sigma^2} = 0$$

$$\sum_{i=1}^n (x_i^T - \theta^T) = \frac{\theta^T}{\sigma^2}$$

$$\sum_{i=1}^n x_i^T = \sum_{i=1}^n \theta^T + \frac{\theta^T}{\sigma^2}$$

$$\sum_{i=1}^n x_i^T = n\theta^T + \frac{\theta^T}{\sigma^2}$$

$$\theta^T = \frac{\sum_{i=1}^n x_i^T}{\left(n + \frac{1}{\sigma^2}\right)}$$

$$\theta_{MAP}^* = \frac{\sum_{i=1}^n x_i}{\left(n + \frac{1}{\sigma^2}\right)}$$

To find if this θ_{MAP}^* is a max solution we
find $\hat{\hat{\theta}}_{MAP}$

$$\begin{aligned}\hat{\hat{\theta}}_{MAP} &= \nabla \left(\sum_{i=1}^n (x_i - \theta)^T - \frac{\theta^T}{\sigma^2} \right) \\ &= -n - \frac{1}{\sigma^2}\end{aligned}$$

Since $n > 0$ and $\sigma^2 > 0$

$\hat{\hat{\theta}}_{MAP}$ is -ve and hence θ_{MAP} is
maximum for θ_{MAP}^*

[Note: Question 2 Solutions are written in
Computer PDF format]

[This finishes Question 1 section]

[All code is made on Python 2.7.12 and developed on Anaconda Python Package Environment]

[Model Error is over the trained data] [Test Prediction Error is over test data]

Question 2) a

As we start to increase the number of features we get an error while computing optimal weights for linear regression over our CT Scan Data. This is because $(X^T X)$ becomes a singular matrix after we include too many features. And since we are trying to calculate our weights through a closed form solution inverse of $(X^T X)$ is not possible because it becomes a singular matrix. Generally, this happens after we include 70+ features out of 385.

There are 2 ways to deal with this. First is to find the generalized inverse rather than Inverse. Generalized Inverse can be found even if the matrix is singular. Though its result is not the same but is close enough to train our model on such a big dataset. The second approach is use an iterative approach. We could use Gradient Descent instead of using closed form solution.

Both approaches have been implemented

- Gradient Descent: BatchGradientDescentRegression: BatchGradientDescent in regressionalgorithms.py
- Generalized Inverse: FSLinearRegression: FSLinearRegression in regressionalgorithms.py

Question 2) b

The Code has been modified so that 2 Approaches of splitting data is allowed.

Approach 1) 10-Fold Split Data.

The Dataset contains 53500 samples. If Fold 2 is selected, then Test Data is from 5351 to 10750. Where if Fold is 1 the Test Data is from 1 to 5350. By intuition 10% is test data and the value of fold selects which subset of dataset should be selected as test data. The fold value is automatically calculated using iterations user provides over the dataset. The remaining 90% data is training data. Hence there are 10 different ways in which the dataset can be split into 90% training data and 10% test data.

Approach 2) 70%-30% Random Selection

This approach divides the dataset into 70% training data and 30% test data. The 70% is selected randomly from the dataset. While the remaining 30% is test data and is always randomly different from previous selection. Each sample only occurs once and will be either in trainset or test set. *I have used this approach throughout the remaining questions.* As this helps to generate randomness over the sampling and prevents us to select samples that may be dependent on each other. Eventually allows our model to train and predict better. Plus, we can run new combination of data on model every time that helps to give us a generalized average error.

As per my understanding I have interpreted mean as average of errors over same parameters and different dataset selections depending upon which approach user takes. Whereas standard error I understand it as standard deviation. I have reported these at the end when all algorithms are run all error for each algorithm are printed.

Qusetion 2) c

The code has been modified as follows basic pseudocode:

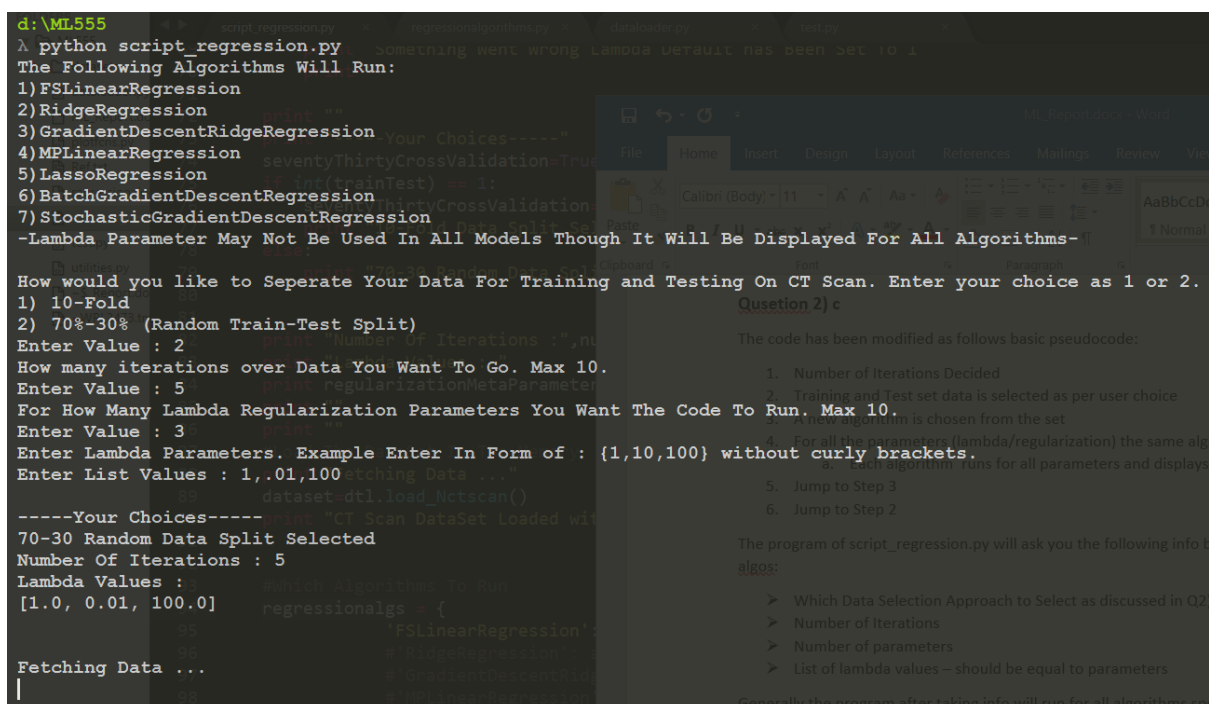
1. Number of Iterations Decided
2. Training and Test set data is selected as per user choice
3. A new algorithm is chosen from the set
4. For all the parameters (lambda/regularization) the same algorithm is iterated
 - a. Each algorithm runs for all parameters and displays its model error one by one
5. Jump to Step 3
6. Jump to Step 2

The program of script_regression.py will ask you the following info before it iterates over all the algos:

- Which Data Selection Approach to Select as discussed in Q2) b *Preferred is 70-30
- Number of Iterations
- Number of parameters
- List of lambda values – should be equal to parameters

Generally, the program after taking info will run for all algorithms specified in the dictionary regressionals. Each algorithm will run for all parameters. And If iterations are more than 1 then a new random dataset of train and test data is selected to run again all algorithms, until all iterations end. Some parameters like epsilon, alpha, epoch have been set manually for algorithms in regressionals and not asked by user. But once manually changed here will run for different values. You can also manually comment algorithms which you don't want to run.

A screenshot of how the startup of program looks and how to enter values



```
d:\ML555
λ python script_regression.py
The Following Algorithms Will Run:
1) FSLinearRegression
2) RidgeRegression
3) GradientDescentRidgeRegression
4) MPLinearRegression
5) LassoRegression
6) BatchGradientDescentRegression
7) StochasticGradientDescentRegression
-Lambda Parameter May Not Be Used In All Models Though It Will Be Displayed For All Algorithms-

How would you like to Seperate Your Data For Training and Testing On CT Scan. Enter your choice as 1 or 2.
1) 10-Fold
2) 70%-30% (Random Train-Test Split)
Enter Value : 2
How many iterations over Data You Want To Go. Max 10.
Enter Value : 5
For How Many Lambda Regularization Parameters You Want The Code To Run. Max 10.
Enter Value : 3
Enter Lambda Parameters. Example Enter In Form of : {1,10,100} without curly brackets.
Enter List Values : 1,.01,100
-----Your Choices-----
70-30 Random Data Split Selected
Number Of Iterations : 5
Lambda Values :
[1.0, 0.01, 100.0]
Fetching Data ...
```

You will need to press Enter many times in between to tell the program to proceed forward. This is done because there is lot of data and the user would like time to see and process data and then move forward.

Question 2) d

The result differs from the Q2) a but by a very small value. When the lambda is increased by a very large value only then the bias in the model increases and variance reduces which is noticeable. With increase in lambda the variance in model falls as shown by Model error which increases with lambda. The Prediction Error also increases this means we are way above the optimal lambda value that brings the minimum bias and variance in our model.

The screenshots of results are pasted here where FSLinearRegression is part a of the question, for better comparison.

RidgeRegression is implemented as RidgeRegression in regressionalgorithms.py

A Gradient Descent approach of Ridge Regression is also implemented which produces the same result: BatchGradientDescentRidgeRegressor in regressionalgorithms.py

```
----- Run 1 Parameter 1 for RidgeRegression -----
Running RidgeRegression on trainSet=37450 and testSet=16050 and lambdaParameter=0.01
Ridge Regression With 385 features and Samples : 37450 and lambda 0.01
Test Prediction Error : RidgeRegression: 0.0653688292018
Model Error : 0.042379835391

----- Run 1 Parameter 2 for RidgeRegression -----
Running RidgeRegression on trainSet=37450 and testSet=16050 and lambdaParameter=0.1
Ridge Regression With 385 features and Samples : 37450 and lambda 0.1
Test Prediction Error : RidgeRegression: 0.0653724596729
Model Error : 0.0423799157378

----- Run 1 Parameter 3 for RidgeRegression -----
Running RidgeRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1.0
Ridge Regression With 385 features and Samples : 37450 and lambda 1.0
Test Prediction Error : RidgeRegression: 0.065378929968
Model Error : 0.0424054574672
```

```
----- Run 1 Parameter 1 for FSLinearRegression -----
Running FSLinearRegression on trainSet=37450 and testSet=16050 and lambdaParameter=0.01
Test Prediction Error : FSLinearRegression: 0.0653685048843
Model Error : 0.0423798347731

----- Run 1 Parameter 2 for FSLinearRegression -----
Running FSLinearRegression on trainSet=37450 and testSet=16050 and lambdaParameter=0.1
Test Prediction Error : FSLinearRegression: 0.0653685048843
Model Error : 0.0423798347731

----- Run 1 Parameter 3 for FSLinearRegression -----
Running FSLinearRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1.0
Test Prediction Error : FSLinearRegression: 0.0653685048843
Model Error : 0.0423798347731
```

For FSLinearRegression lambda is not used for calculation. Since we entered 3 parameters it is run 3 times. And the result will be the same.

Question 2) e

The MPLinearRegression Error Is Reported Below:

Total 372 features were selected out of 385 and these features can be used to train the model where the remaining features can be removed. Model Error: .4308 | Prediction Error: .6627

Epsilon for the residual error/threshold was taken as: 0.1 (It's not fixed and can be changed manually from script_regression.py in regression)

```
Press Enter To Start The Algorithm Run for : MPLinearRegression
70-30 DataSet Function Selected To Fetch Data
Test Data Randomly Selected; 37450 is Train Data and 16050 is Test Data

----- Run 1 Parameter 1 for MPLinearRegression -----
Running MPLinearRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1.0
Tolerance Reached With 372 features
Weights Learned By MPLinearRegression on the above mentioned features.
Test Prediction Error : MPLinearRegression: 0.662705721916
Model Error : 0.430840779662
```

MPLinearRegression is implemented as MPLinearRegression in regressionalgorithms.py

Question 2) f

The Implementation of Lasso has been done using soft thresholding operator. This leads to convert weights of many unimportant features to 0. I have run Lasso for [100,500,1000,5000,10000]. My lasso implementation has the lambda divided by the samples number (37450). Hence a bigger value of lambda will be needed to remove unimportant features. The working only differs in choosing the magnitude of lambda and not how soft threshold works. [*]

```
----- Run 1 Parameter 1 for LassoRegression -----
Running LassoRegression on trainSet=37450 and testSet=16050 and lambdaParameter=100.0
Step 1 ) Old Cost : 2337.67290838 New Cost : 1053.54643087 Step Size : 0.03125 Lambda : 100.0
Step 101 ) Old Cost : 79.7638346263 New Cost : 79.4797613507 Step Size : 0.03125 Lambda : 100.0
Step 201 ) Old Cost : 76.070838701 New Cost : 76.0155075311 Step Size : 0.25 Lambda : 100.0
Step 301 ) Old Cost : 74.7473542042 New Cost : 74.6982599698 Step Size : 0.5 Lambda : 100.0
Step 401 ) Old Cost : 73.9642958949 New Cost : 73.8203839897 Step Size : 1 Lambda : 100.0
Weights Learned By LassoRegression on 385 features.
By Using Lasso with lambda 100.0 , 77 Features weight were reduced to 0.
Test Prediction Error : LassoRegression: 0.0679078767298
Model Error : 0.0439741615438

----- Run 1 Parameter 2 for LassoRegression -----
Running LassoRegression on trainSet=37450 and testSet=16050 and lambdaParameter=500.0
Step 1 ) Old Cost : 2499.00585357 New Cost : 1067.75977067 Step Size : 0.03125 Lambda : 500.0
Step 101 ) Old Cost : 99.6852433095 New Cost : 99.1390019678 Step Size : 0.0625 Lambda : 500.0
Step 201 ) Old Cost : 98.4176019614 New Cost : 98.1769627797 Step Size : 0.0625 Lambda : 500.0
Step 301 ) Old Cost : 88.5743362849 New Cost : 88.4759323811 Step Size : 0.0625 Lambda : 500.0
Step 401 ) Old Cost : 85.2221558852 New Cost : 85.0356725897 Step Size : 0.03125 Lambda : 500.0
Weights Learned By LassoRegression on 385 features.
By Using Lasso with lambda 500.0 , 77 Features weight were reduced to 0.
Test Prediction Error : LassoRegression: 0.071335164419
Model Error : 0.0462609214627

----- Run 1 Parameter 3 for LassoRegression -----
Running LassoRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1000.0
Step 1 ) Old Cost : 2230.20100424 New Cost : 1105.60031099 Step Size : 0.03125 Lambda : 1000.0
Step 101 ) Old Cost : 107.748057576 New Cost : 106.478389877 Step Size : 0.25 Lambda : 1000.0
Step 201 ) Old Cost : 104.122887195 New Cost : 103.135428342 Step Size : 0.0625 Lambda : 1000.0
Step 301 ) Old Cost : 96.3762813684 New Cost : 95.9833516102 Step Size : 0.125 Lambda : 1000.0
Step 401 ) Old Cost : 91.5201369081 New Cost : 91.4184359266 Step Size : 0.125 Lambda : 1000.0
Weights Learned By LassoRegression on 385 features.
By Using Lasso with lambda 1000.0 , 281 Features weight were reduced to 0.
Test Prediction Error : LassoRegression: 0.0764840952886
Model Error : 0.04964331636
```

```

----- Run 1 Parameter 4 for LassoRegression -----
Running LassoRegression on trainSet=37450 and testSet=16050 and lambdaParameter=5000.0
Step 1 ) Old Cost : 2467.41490348 New Cost : 1082.55067093 Step Size : 0.03125 Lambda : 5000.0
Step 101 ) Old Cost : 198.48624322 New Cost : 181.638725513 Step Size : 0.125 Lambda : 5000.0
Step 201 ) Old Cost : 197.69253298 New Cost : 183.159591409 Step Size : 0.125 Lambda : 5000.0
Step 301 ) Old Cost : 197.375418127 New Cost : 183.709292609 Step Size : 0.125 Lambda : 5000.0
Step 401 ) Old Cost : 197.241900268 New Cost : 183.775116355 Step Size : 0.125 Lambda : 5000.0
Weights Learned By LassoRegression on 385 features.
By Using Lasso with lambda 5000.0 , 351 Features weight were reduced to 0.
Test Prediction Error : LassoRegression: 0.108071485562
Model Error : 0.0703226689485

----- Run 1 Parameter 5 for LassoRegression -----
Running LassoRegression on trainSet=37450 and testSet=16050 and lambdaParameter=10000.0
Step 1 ) Old Cost : 2683.91254954 New Cost : 1060.68000117 Step Size : 0.03125 Lambda : 10000.0
Step 101 ) Old Cost : 292.680886529 New Cost : 250.088481908 Step Size : 0.03125 Lambda : 10000.0
Step 201 ) Old Cost : 319.940080639 New Cost : 284.12522719 Step Size : 0.125 Lambda : 10000.0
Step 301 ) Old Cost : 321.814951707 New Cost : 289.728618943 Step Size : 0.0625 Lambda : 10000.0
Step 401 ) Old Cost : 305.216058172 New Cost : 280.404379115 Step Size : 0.25 Lambda : 10000.0
Weights Learned By LassoRegression on 385 features.
By Using Lasso with lambda 10000.0 , 334 Features weight were reduced to 0.
Test Prediction Error : LassoRegression: 0.135595647278
Model Error : 0.0884472308472

```

The outputs of Lasso Regression are shown in above screenshots for all lambda.

I think lambda=100 is best as the model and prediction error does not shoot that much.

```

----- Run 1 Parameter 1 for LassoRegression -----
Running LassoRegression on trainSet=37450 and testSet=16050 and lambdaParameter=100.0
Step 1 ) Old Cost : 2337.67290838 New Cost : 1053.54643087 Step Size : 0.03125 Lambda : 100.0
Step 101 ) Old Cost : 79.7638346263 New Cost : 79.4797613507 Step Size : 0.03125 Lambda : 100.0
Step 201 ) Old Cost : 76.070838701 New Cost : 76.0155075311 Step Size : 0.25 Lambda : 100.0
Step 301 ) Old Cost : 74.7473542042 New Cost : 74.6982599698 Step Size : 0.5 Lambda : 100.0
Step 401 ) Old Cost : 73.9642958949 New Cost : 73.8203839897 Step Size : 1 Lambda : 100.0
Weights Learned By LassoRegression on 385 features.
By Using Lasso with lambda 100.0 , 77 Features weight were reduced to 0.
Test Prediction Error : LassoRegression: 0.0679078767298
Model Error : 0.0439741615438

```

Model error lambda=100 is 0.043

Prediction error lambda=100 is 0.067

With lambda=100 about 77 features weights were converted to 0. Which means that the data for this features can all together be removed as they do not play an important part in prediction.

With increase in lambda the number of features whose weight turns to 0 also increases. But the model and prediction error are seriously affected and become worse.

LassoRegression is implemented as LassoRegression in regressionalgorithms.py

Question 2) g

Stochastic Linear Regression is implemented by running the gradient of error function over 1 single sample and finding the error and updating the weights of all features on just one sample. And then again iterating over all samples and changing the sample on each iteration for which the weight gets updated. Updating of weights happen for all unique samples one at a time. Whereas alpha is reduced with each new sample. The process does not end here after we move over all samples. And learn weights. We again do the same process again for epoch iterations. Epoch has been set to 25 by default but can be manually changed in regressionalgs. This will include all features and the computation time will be small per epoch iteration. Generally preferred if the dataset is very large and you can include all features.

The Run for Stochastic Regression Is given Below:


```

Press Enter To Start The Algorithm Run for : StochasticGradientDescentRegression
----- Run 1 Parameter 1 for StochasticGradientDescentRegression -----
Running StochasticGradientDescentRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1.0
On Epoch : 1
Step 1 ) Old Cost : 336.384344795 Step Size : 0.01
Step 10001 ) Old Cost : 35.6313756028 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 1713.28026128 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 715.58070871 Step Size : 3.33322222593e-07
On Epoch : 2
Step 1 ) Old Cost : 257.988486747 Step Size : 0.01
Step 10001 ) Old Cost : 8.6489687415 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 1239.33906352 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 329.544114693 Step Size : 3.33322222593e-07
On Epoch : 3
Step 1 ) Old Cost : 35.2928581568 Step Size : 0.01
Step 10001 ) Old Cost : 46.7506490363 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 833.083202098 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 118.849687319 Step Size : 3.33322222593e-07
On Epoch : 4
Step 1 ) Old Cost : 2.93654877805 Step Size : 0.01
Step 10001 ) Old Cost : 73.1424808173 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 549.631042806 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 32.3608579981 Step Size : 3.33322222593e-07

```

```

On Epoch : 21
Step 1 ) Old Cost : 0.00088359997907 Step Size : 0.01
Step 10001 ) Old Cost : 116.135861773 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 1.61208391214 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 0.379599996041 Step Size : 3.33322222593e-07
On Epoch : 22
Step 1 ) Old Cost : 0.00323579347071 Step Size : 0.01
Step 10001 ) Old Cost : 116.106834992 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 2.48588029643 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 0.228886220454 Step Size : 3.33322222593e-07
On Epoch : 23
Step 1 ) Old Cost : 0.00623305384234 Step Size : 0.01
Step 10001 ) Old Cost : 115.948872452 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 3.40761317131 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 0.123845780947 Step Size : 3.33322222593e-07
On Epoch : 24
Step 1 ) Old Cost : 0.00935065481825 Step Size : 0.01
Step 10001 ) Old Cost : 115.675006693 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 4.33447463852 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 0.055691173969 Step Size : 3.33322222593e-07
On Epoch : 25
Step 1 ) Old Cost : 0.0122820457785 Step Size : 0.01
Step 10001 ) Old Cost : 115.298171257 Step Size : 9.99900009999e-07
Step 20001 ) Old Cost : 5.23674494516 Step Size : 4.9997500125e-07
Step 30001 ) Old Cost : 0.0168895952281 Step Size : 3.33322222593e-07
Weights Learned By StochasticGradientDescent on 385 features.
Test Prediction Error : StochasticGradientDescentRegression: 0.0866053383966
Model Error : 0.056919315077

```

Model Error: 0.056 | Prediction Error: 0.086

After 25 epochs the Model error is decent, comparably good with respect to Batch Gradient Descent.

```

Press Enter To Start The Algorithm Run for : BatchGradientDescentRegression

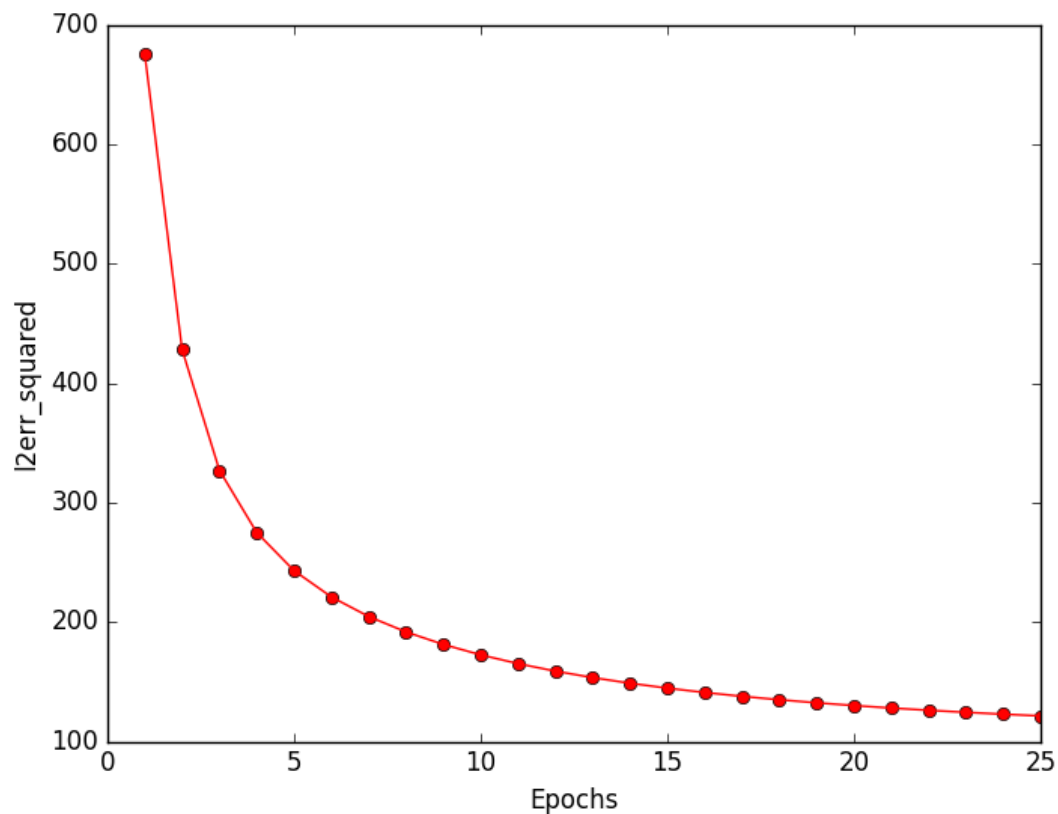
----- Run 1 Parameter 1 for BatchGradientDescentRegression -----
Running BatchGradientDescentRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1.0
Step 1 ) Old Cost : 2278.73788127 New Cost : 1066.52651176 Step Size : 0.03125
Step 101 ) Old Cost : 84.1319761713 New Cost : 83.87925101 Step Size : 0.03125
Step 201 ) Old Cost : 76.6849437434 New Cost : 76.5963492545 Step Size : 0.03125
Step 301 ) Old Cost : 73.8438174652 New Cost : 73.8429142983 Step Size : 0.25
Step 401 ) Old Cost : 72.3570828744 New Cost : 72.3568171064 Step Size : 0.0625
Weights Learned By BatchGradientDescent on 385 features.
Test Prediction Error : BatchGradientDescentRegression: 0.0667671117246
Model Error : 0.0436776494097 "Batch RunTime VS L2_SquaredError"

```

Stochastic Gradient Descent is implemented as StochasticGradientDescent in regressionalgorithms.py

Question 2) h

Error Vs Epoch graph:



Batch Gradient Descent is implemented as BatchGradientDescent in regressionalgorithms.py

Batch Gradient Descent:

Model Error: 0.043 | Prediction Error: 0.066

```

Press Enter To Start The Algorithm Run for : BatchGradientDescentRegression

----- Run 1 Parameter 1 for BatchGradientDescentRegression -----
Running BatchGradientDescentRegression on trainSet=37450 and testSet=16050 and lambdaParameter=1.0
Step 1 ) Old Cost : 2278.73788127 New Cost : 1066.52651176 Step Size : 0.03125
Step 101 ) Old Cost : 84.1319761713 New Cost : 83.87925101 Step Size : 0.03125
Step 201 ) Old Cost : 76.6849437434 New Cost : 76.5963492545 Step Size : 0.03125
Step 301 ) Old Cost : 73.8438174652 New Cost : 73.8429142983 Step Size : 0.25
Step 401 ) Old Cost : 72.3570828744 New Cost : 72.3568171064 Step Size : 0.0625
Weights Learned By BatchGradientDescent on 385 features.
Test Prediction Error : BatchGradientDescentRegression: 0.0667671117246
Model Error : 0.0436776494097 "Batch RunTime VS L2_SquaredError"

```

Error vs Runtime:

```

Stochastic RunTime VS L2_SquaredError

Stochastic Run Time:
[73.02300000190735]

Stochastic Error:
[128.17253442000774]

Batch RunTime VS L2_SquaredError

Batch Run Time:
[39.81599998474121]

Batch Error:
[70.884906492188094]

```

For a big epoch value in this case 25 the run time is greater compare to batch. Had the dataset been any bigger like 10,00,000+ samples, the Stochastic runtime would have been smaller compared to batch gradient descent.

Important points about Code:

- Epsilon value/Tolerance value for following algorithms can be changed manually by updating the dictionary `regressionals` in `script_regression.py`
 - ✓ `GradientDescentRidgeRegression`
 - ✓ `MPLinearRegression`
 - ✓ `LassoRegression`
 - ✓ `BatchGradientDescentRegression`
- Alpha value for following algorithms can be changed manually by updating the dictionary `regressionals` in `script_regression.py`
 - ✓ `GradientDescentRidgeRegression`
 - ✓ `LassoRegression`
 - ✓ `BatchGradientDescentRegression`
 - ✓ `StochasticGradientDescentRegression`
- You can comment any algo you don't want to run
- Answer 2) c) contains how you can run the code and what info the program asks you to run the algos.
- Original Code `splitdataset` function in `dataloader.py` was modified to accompany data split approach. No original code was deleted, only new conditions are added.
- Code will run similar to what was originally given to us
 - ✓ Call `[python script_regression.py]`

References

1. <http://www.numerical.rl.ac.uk/people/nimg/msc/lectures/uepart2.2.pdf>
2. <http://www.simonlucey.com/soft-thresholding/> [*]
3. <http://www.stat.washington.edu/courses/stat535/fall14/Handouts/l3slides-training.pdf>
4. https://en.wikipedia.org/wiki/Matrix_calculus
5. <https://www.math.ubc.ca/~ansteemath104/104newtonmethod.pdf>
6. https://en.wikipedia.org/wiki/Newton%27s_method
7. <http://www.jmlr.org/papers/volume10/zhang09a/zhang09a.pdf>
8. http://www.holehouse.org/mlclass/07_Regularization.html