CS F320

FOUNDATIONS OF DATA SCIENCE

Assignment - 1

Group members

Name	ID
T. Praneeth Bhargav	2020A7PS1299H
S.V.S.Rahul	2020A7PS0204H
Raghav Krishna Regalla	2020B3PS1382H

1A

1. Let the probability of a person liking the update be s. And (a+b) people are surveyed(data), out of which a liked and b disliked the update, then the likelihood of data is L(s) =P(a person liking and b person disliking the update)
   Since a single person liking/disliking the update is independent of the liking/disliking of the other person. We can rewrite the above equation as L(s) = P(a person liking) * P(a person liking) …. (a times) * P(a person disliking) * P(a person disliking) ….b times.
   Hence we have L(s) = (P(a person liking the update))^{a} * P(a person disliking the update) ^{b}.

   Let the probability of a person liking the update be s. Hence the probability of a person disliking the update becomes 1-s. Since they are mutually exclusive and exhaustive events.

   Hence the final equation becomes

   $$L(s) = (s)^{a} * (1-s)^{b}$$

   We now need to choose an 's' such that it maximizes the likelihood function value.
   We know that max L(s) = max log(L(s))
   $= \max(\log(s^{a} \times (1-s)^{b}))$
   $= \max(a \log s + b \log(1- s))$

   So, for the likelihood to be maximum L'(s) should be 0.

$$L'(s) \Rightarrow a/s - b/(1-s) = 0$$
$$\Rightarrow s = a / (a+b)$$

So this s maximizes the likelihood function value.
For the given problem, a and b varies depending on the result of the surveys, and this posterior probability becomes the prior probability for the next survey.

2. We are given the Prior distribution of the random variable s. And then, we are given data, using which we find the posterior probability distribution of s. This is done by Bayes theorem, which states, Posterior probability = Prior Probability x likelihood. Say if a+b people are surveyed, out of them liked the update. If alpha and beta are the original parameters of the beta distribution, the new parameters become alpha+a and beta+b. This can be proved mathematically.

The probability of ‘a’ customers liking the update and ‘b’ customers disliking the update.
$P(D|s) = s^{a} \times (1-s)^{b}$

Bayes theorem is stated as

$$P(s|D) = \frac{p(s) \times p(D|s)}{P(D)}$$

P(D) term is constant as it is integral with respect to all the possible values of s.
P(D) = k (constant)

We now have $P(s|D)\ \alpha\ p(s) \times p(D|s)$

$$p(s) = \frac{\gamma\,(\alpha + \beta) * s^{\alpha-1} * (1-s)^{\beta-1}}{\gamma(\alpha) * \gamma\,(\beta)}$$

$$P(s|D)\ \alpha\ \frac{s^{a} * (1-s)^{b} * \gamma\,(\alpha + \beta) * s^{\alpha-1} * (1-s)^{\beta-1}}{\gamma(\alpha) * \gamma\,(\beta)}$$

$$P(s|D)\ \alpha\ \frac{\gamma\,(\alpha + \beta) * s^{\alpha+a-1} * (1-s)^{\beta+b-1}}{\gamma(\alpha) * \gamma\,(\beta)}$$

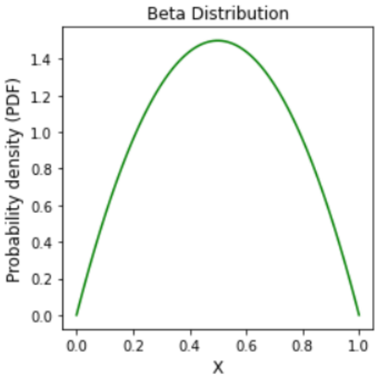
Ignoring the constants, this can be written as

$$P(s|D)\ \alpha\ s^{\alpha+a-1} * (1-s)^{\beta+b-1}$$

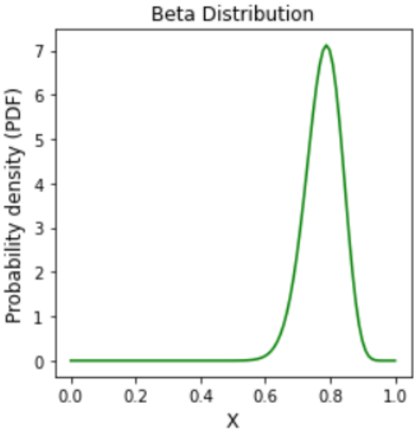
$$P\,(s|D) = \frac{s^{\alpha+a-1} * (1-s)^{\beta+b-1}}{\int s^{\alpha+a-1} * (1-s)^{\beta+b-1}\, ds}$$

After a few computations, we get P(s|D) = Beta distribution with parameters as α + a and β + b.
P(s|D) denotes the posterior probability after seeing some data with a people liking the update and b people disliking the update.
In this way we can find the posterior distribution given some data.

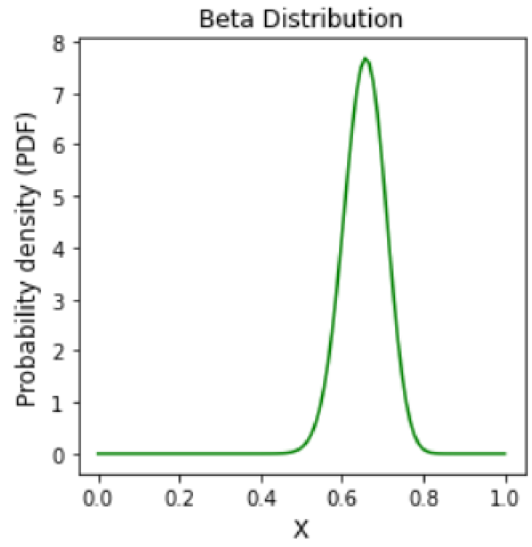
Prior distribution: Beta distribution with parameters alpha = 2 and beta=2



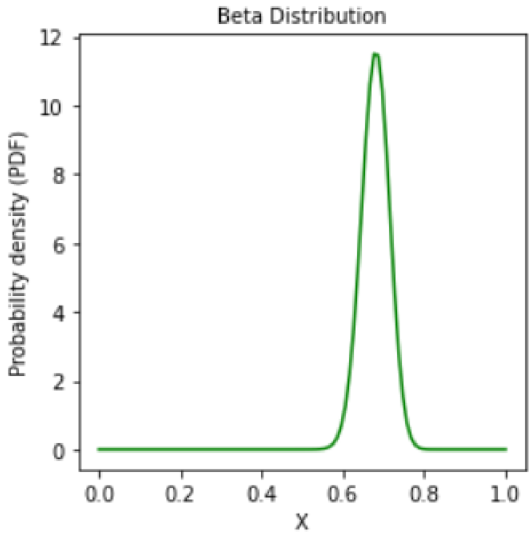
Survey 1: 40 out of 50 liked the update, hence a = 40 and b = 10.
Hence the parameters of the beta distribution change to alpha = 2 + a = 42 and beta = 2 + 10 = 12. This is obtained using Bayes theorem.



Survey 2: 17 out of 30 disliked the update. Hence a = 13 and b = 17. Hence the parameters of the Beta distribution become alpha = 42 +a = 55 and beta = 12 + b = 29.



Survey 3: 70 out of 100 people liked the update. Here a = 70 and b =30. Hence the parameters of the beta distribution become alpha = 55 + a = 125 and beta = 29 + b = 59.



1B

1. **Description of the Model:** The model is polynomial in nature, meaning the predicted output is continuous. Hence regression. It is tested with various degrees ranging from 0 to 9, and the one which gave the best predictions on not only the training set but also on the testing data is concluded as the best model.

   For degree 1, we assumed our model to be as follows
   $y(x) = w_0 + w_1 x_1 + w_2 x_2$
   First, we split the data in the ratio of 80:20 as training data and testing data, respectively after entirely shuffling the give dataset.
   Let y_n be the value of LC50 for the nth training example.
   ⇒ Without regularization, the error function is as below:
   $E(w0, w1, w2) = (1/2N) * \sum (yn - (w_0 + w_1x_{1n} + w_2x_{2n}))^2$
   where N denotes the no. of training data examples
   Error must be minimum for our model to be the best possible model, that is, the
   error function must be convex.
   To reach the minimum, we can use the gradient descent method or
   stochastic gradient descent method.

   Similarly, for degree d the model is assumed to be

   $y = w_0 x_1^0x_2^0 + w_1x_1^1x_2^0 + w_2x_1^1x_2^0 + \ldots\ldots\ w_dx_1^0x_2^d$

   For degree d there will be a total of (d+1) * (d+2) / 2 number of coefficients (weights). Accordingly the weight vector and the associated features ($x_1^ax_2^b$) are precomputed and used accordingly.

Algorithms: Gradient descent and stochastic gradient descent algorithms have been used.

Gradient descent:

For degree 1 it is shown below, for degree d it is similar to this

In the Gradient Descent method, we will reach the values of parameters w0, w1, and w2, which minimizes the Error using $E(w_{(k+1)}) < E(w_k)$ where
w denotes a vector of the parameters w0, w1, and w2,
k denotes kth iteration,$w_{(k+1)} = w_k - * (\nabla E)w = w(k)\ \eta$,
η denotes scaling parameter ($\eta > 0$)
and $(\nabla E) = [\ \partial E/\partial w0\ \partial E/\partial w1\ \partial E/\partial w2\ ]$

Here
$\partial E/\partial w0 = (1/2N) * 2 * \sum (y_n - (w_0 + w_1x_{1n} + w_2x_{2n})) * (-1)$
$\partial E/\partial w1 = (1/2N) * 2 * \sum (y_n - (w_0 + w_1x_{1n} + w_2x_{2n})) * (-x_{1n})$
$\partial E/\partial w2 = (1/2N) * 2 * \sum (y_n - (w_0 + w_1x_{1n} + w_2x_{2n})) * (-x_{2n})$

For degree d we have data matrix with m rows(training examples) and 55 columns (maximum of degree 9 it has 55 features) which represent multivariate features which are precomputed and the (d+1)*(d+2)/2 columns are taken for degree d. This data is then split into 80:20 ration for training and testing respectively.
Then the loss and its gradient is computed wrt the d weights and weight vector is updated accordingly. This runs for 1e5 iterations with learning rate of 0.01. All the losses after gradient descent of degree d are stored in an array J_train and the losses of testing dataset in array J_test.

Stochastic gradient descent:

In the Stochastic Gradient Descent method, we will reach the values of parameters w0, w1, and w2, which minimizes the Error using E(w(k+1)) < E(w(k)).

This method is precisely the same as the Gradient descent method except that we choose one random training example to perform the weight updates

(x_{1t}, $x_{2t,}$ y_t) among the 'N' given training

examples to build the model.

The equations used in this method slightly differ from the Gradient descent method as depicted below:

$E(w_0, w_1, w_2) = (1/2) * (y_n - (w_0 + w_1x_{1n} + w_2x_{2n}))^2$

$w_{(k+1)} = w_{(k)}- * (\nabla E)w = w(k)\ \eta$

$\partial E/\partial w0 = (1/2) * 2 * (y_t - (w_0 + w_1x_{1t} + w_2x_{2t})) * (-1)$

$\partial E/\partial w1 = (1/2) * 2 * (y_t - (w_0 + w_1x_{1t} + w_2x_{2t})) * (-x_{1t})$

$\partial E/\partial w2 = (1/2) * 2 * (y_t - (w_0 + w_1x_{1t} + w_2x_{2t})) * (-x_{2t})$

In the same gradient descent can be applied for various degree polynomials and get the best fit model.

For degree d we have data matrix with m rows(training examples) and 55 columns (maximum of degree 9 it has 55 features) which represent multivariate features which are precomputed and the (d+1)*(d+2)/2 columns are taken for degree d. From this data a random tuple is selected and the gradient is computed using it and the weights are updated accordingly.

Then the loss and its gradient is computed wrt the d weights and weight vector is updated accordingly. This runs for 1e5 iterations with learning rate of 0.01. All the losses after gradient descent of degree d are stored in an array J_train_stoc and the losses of testing dataset in array J_test_stoc.

How is regularization implemented: Regularization is implemented using the penalty made upon the norms of the weights. This is also done for various regularization parameters (lambdas) and the power of the regularization (q).
The updated cost function now is as follows,

$$J(\theta) = \underbrace{\sum_{i=1}^{n}(y_i - \theta_0 - \sum_{k=1}^{p} x_{ik}\theta_k)^2}_{\text{Sum of Squared Residuals}} + \lambda \underbrace{\sum_{k=1}^{p}\theta_k^2}_{\text{Squared Regularization Term}}$$

The weights are now updated according to the gradients with respect to this cost function, and hence the weights are going to be restricted, preventing the problem of overfitting.
With regularization, the error function is as below:
$E(w_0, w_1, w_2) = (1/2N) *[\ (y_n - (w_0 + w_1x_{1n} + w_2x_{2n}))^2\ +\ \lambda \sum (|w1|^q + |w2|^q)\ ]$

where λ denotes the regularization parameter which indicated how much penalization can be applied to weights to prevent the issue of overfitting.
Here, we can again apply the gradient descent and stochastic gradient descent
methods as described above using this modified error function.

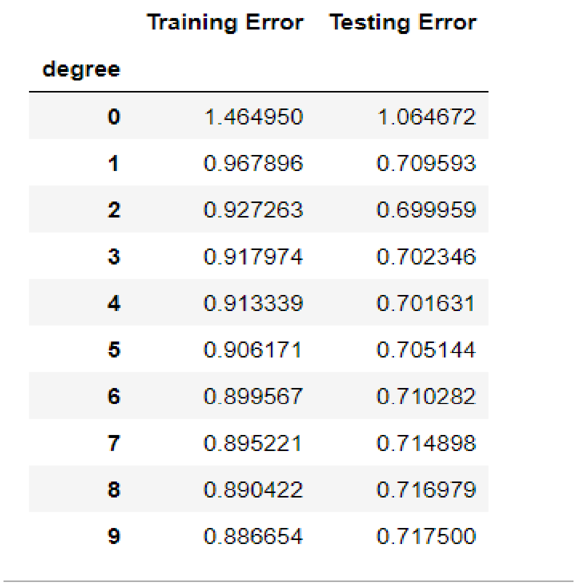
Batch gradient descent

We have done the gradient descent algorithm based on the q value.

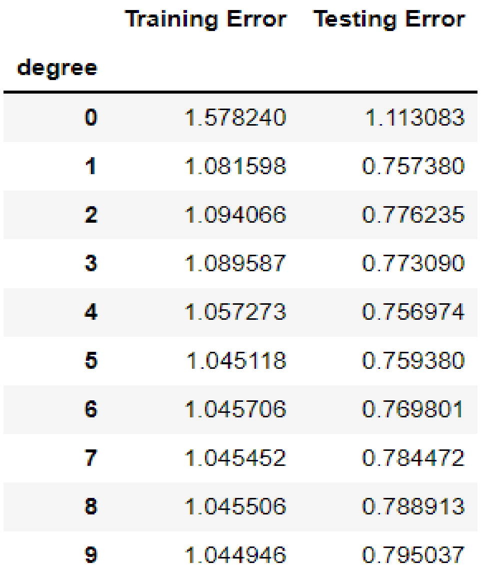
q = 0.5, dE/dw$_i$ =(1/n) * Σ (x * (y^ - y) + (1/4) * |w$_i$|$^{-1/2}$)
q = 1 ,   dE/dw$_i$ = (1/n) * Σ (x * (y^ - y) + 1) if w$_i$ >= 0
dE/dw$_i$ = (1/n) * Σ (x * (y^ - y)) if w$_i$ < 0
q = 2 ,   dE/dw = (1/n) * Σ (x * (y^ - y) + |w$_i$|)
q = 4,   dE/dw$_i$ = (1/n) * Σ (x * (y^ - y) + 4 * |w$_i$|^3)

Here, y^ is the predicted output for a particular tuple.

2.



degree	Training Error	Testing Error
0	1.464950	1.064672
1	0.967896	0.709593
2	0.927263	0.699959
3	0.917974	0.702346
4	0.913339	0.701631
5	0.906171	0.705144
6	0.899567	0.710282
7	0.895221	0.714898
8	0.890422	0.716979
9	0.886654	0.717500



degree	Training Error	Testing Error
0	1.578240	1.113083
1	1.081598	0.757380
2	1.094066	0.776235
3	1.089587	0.773090
4	1.057273	0.756974
5	1.045118	0.759380
6	1.045706	0.769801
7	1.045452	0.784472
8	1.045506	0.788913
9	1.044946	0.795037

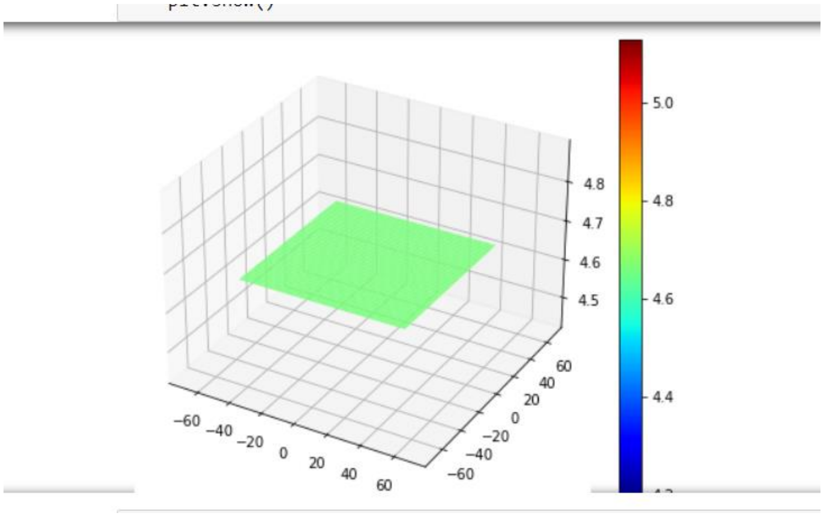
Batch gradient descent **Stochastic gradient descent**

Observations: As we can see, the model fits the data pretty well till degree 2 as the training and testing errors decrease steadily till degree 2. But after degree 2, the training error still decreases, but the testing error starts to rise, which means our model from degree 2 starts to overfit the training data, due to which it fails miserably with the unseen(testing) data. Hence polynomial of degree 2 can be thought of as the best model.

And when it comes to stochastic gradient descent, the errors keep on fluctuating as we take a single random data sample instead of the entire dataset to update the parameters

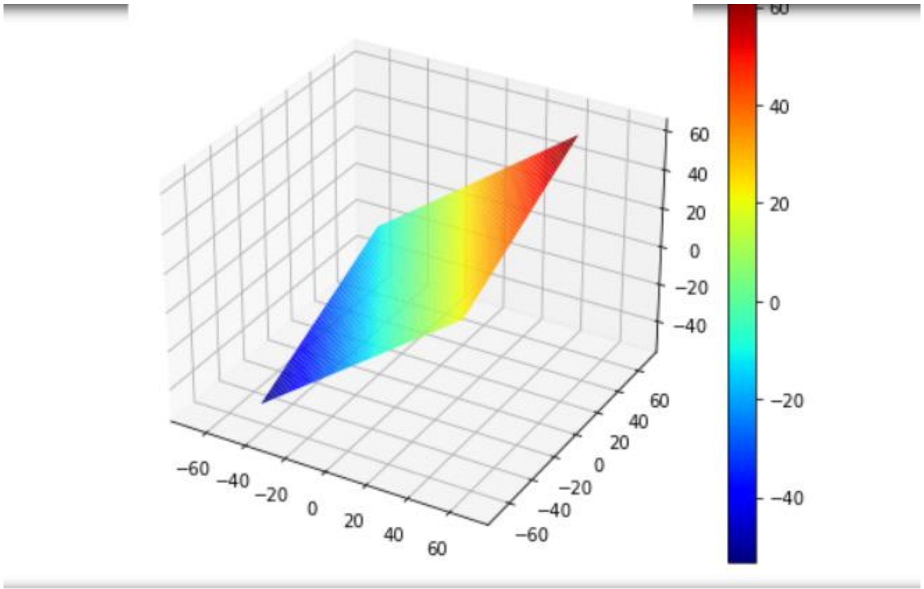
3.

Degree 0

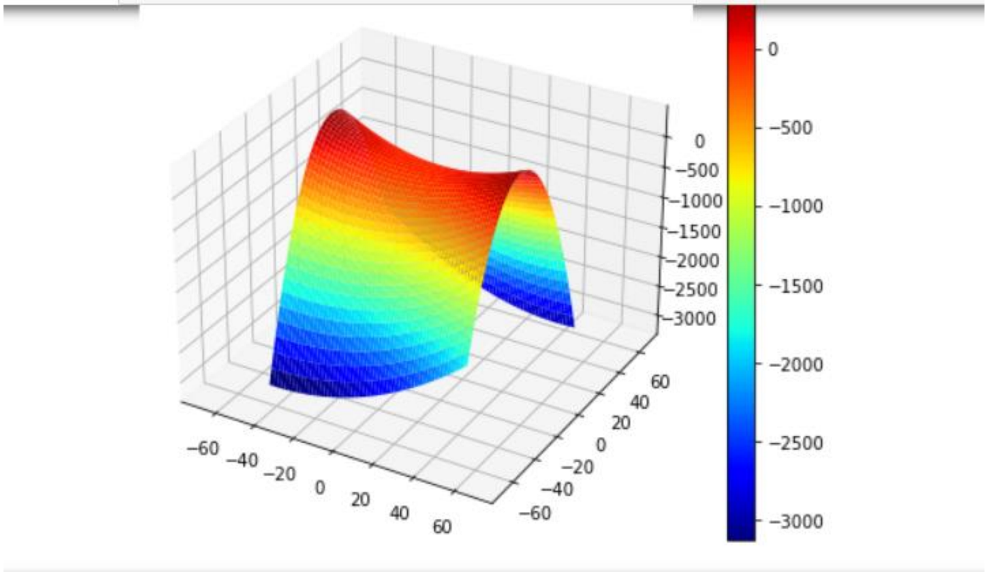


```
plt.show()
```

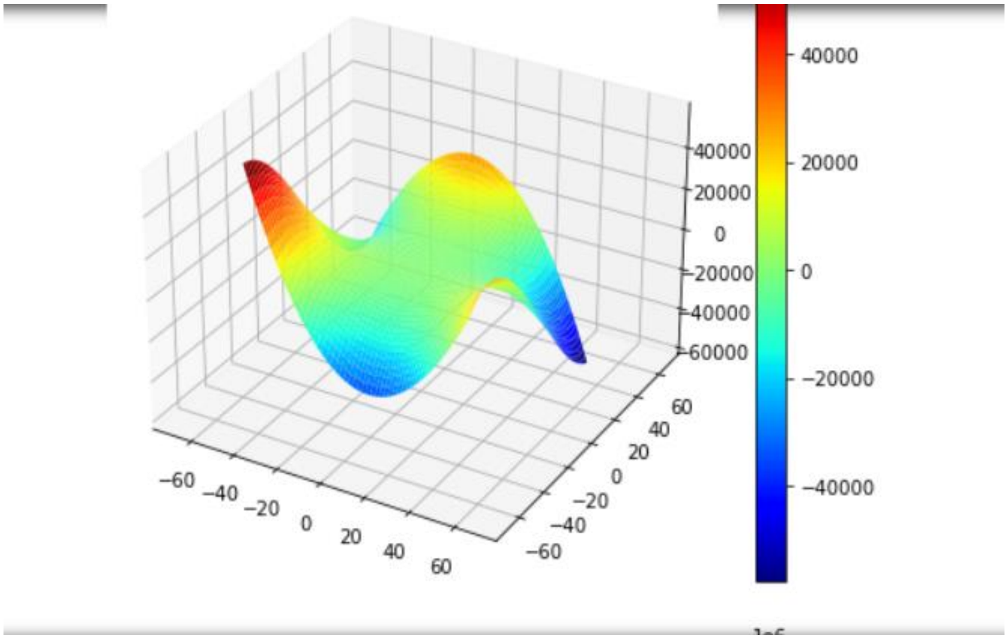
Degree 1



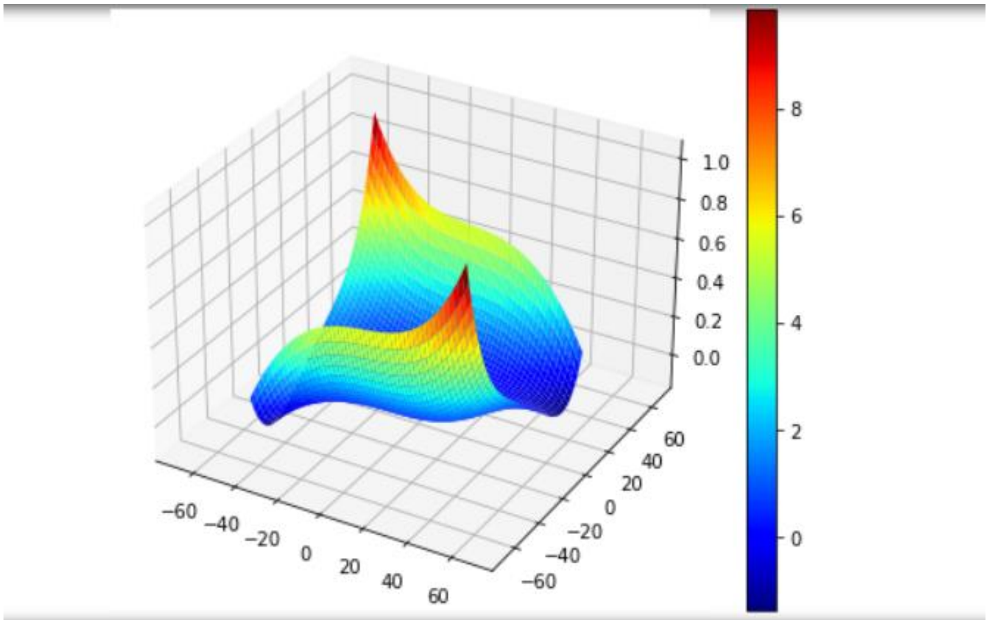
Degree 2



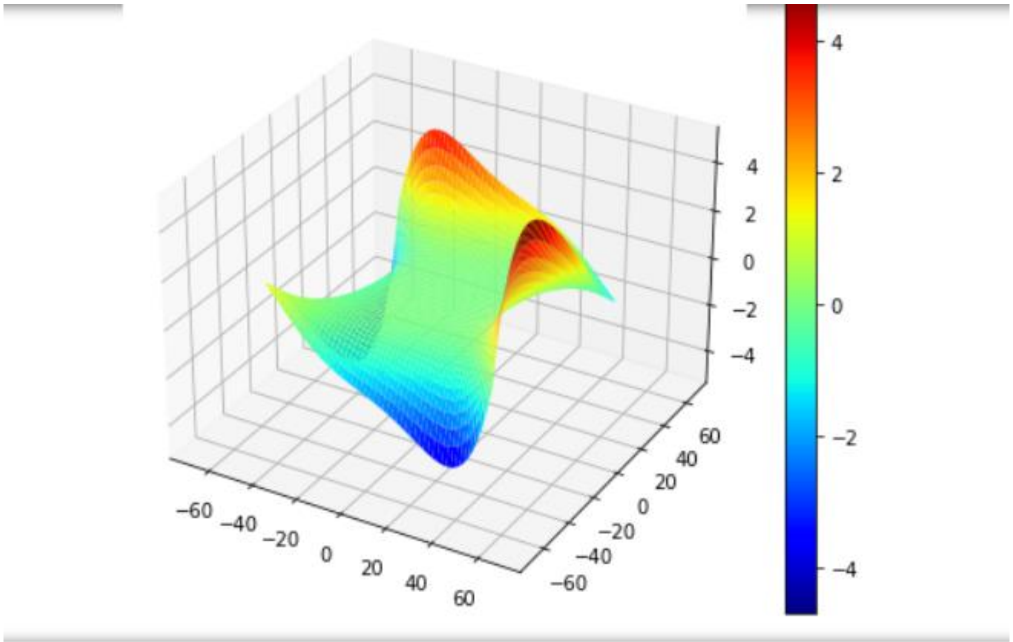
Degree 3



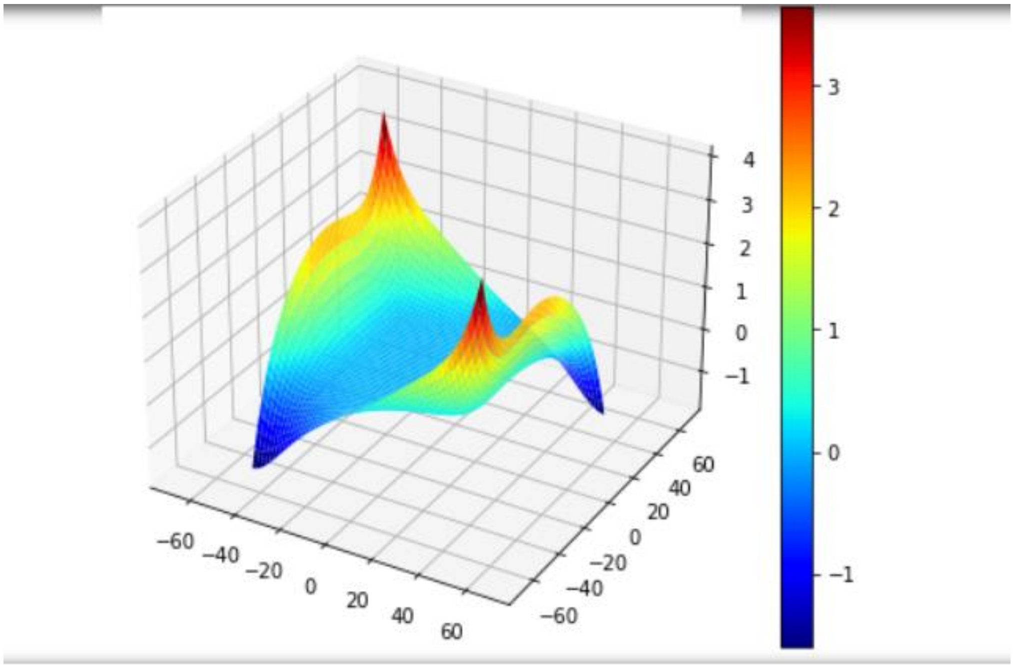
Degree 4



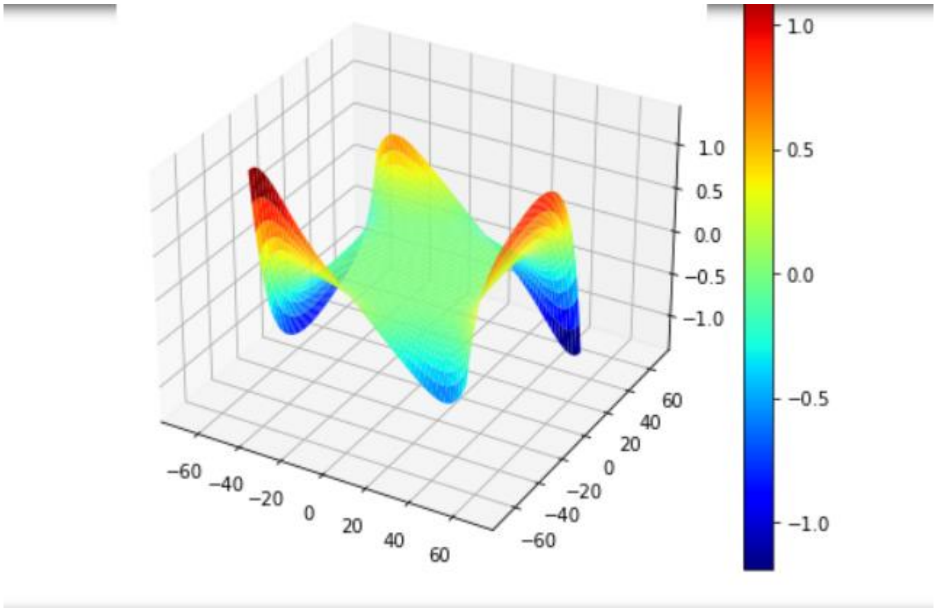
Degree 5



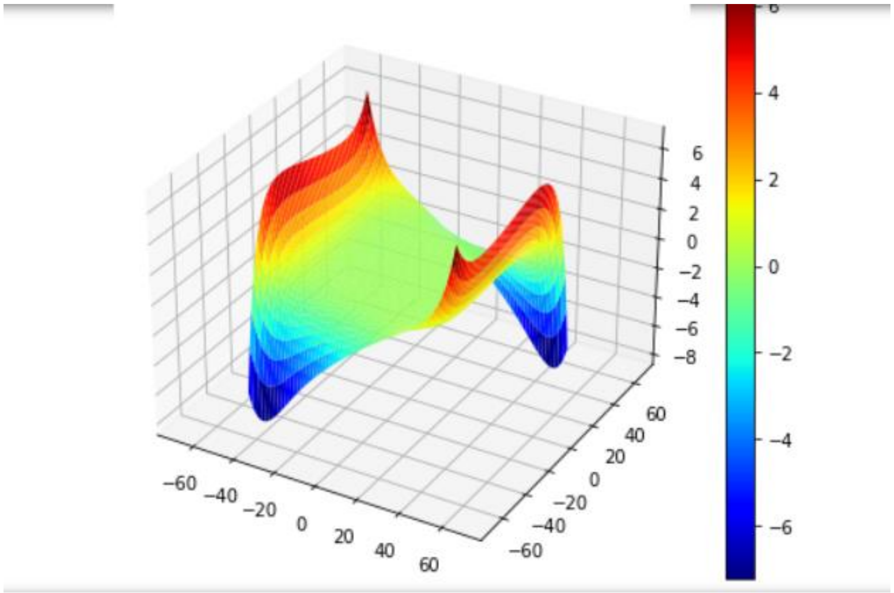
Degree 6



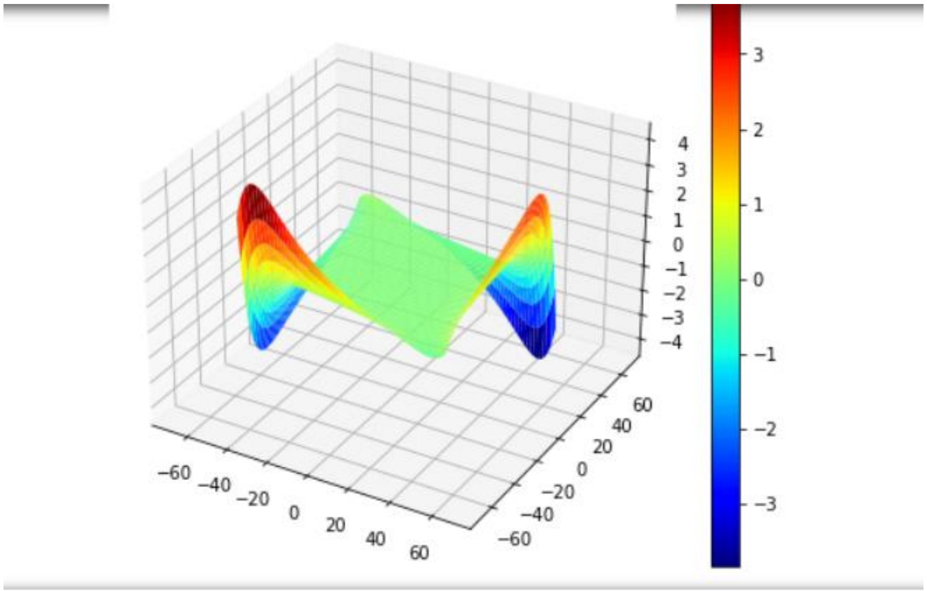
Degree 7



Degree 8



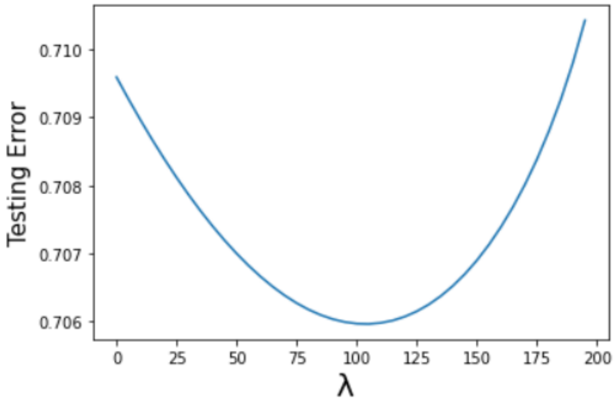
Degree 9



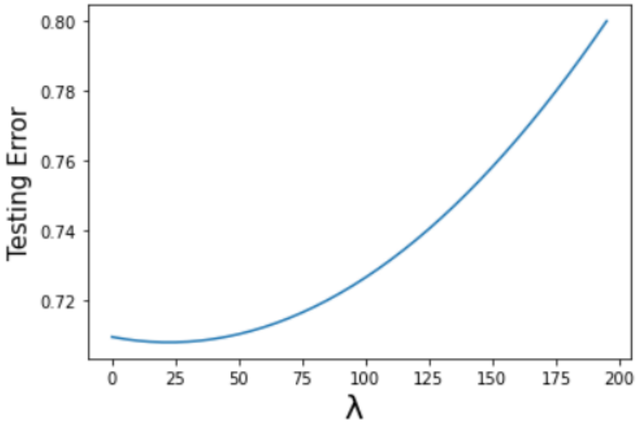
4.

Model	q	Optimal	Training Error	Testing Error
Unregularized	-	Optimal degree = 2	0.8599	0.9340
Regularized	0.5	λ = 110	0.87437	0.68344
	1	λ = 25	0.87437	0.68344
	2	λ = 5	0.87437	0.68342
	4	λ = 0.01	0.87437	0.68332

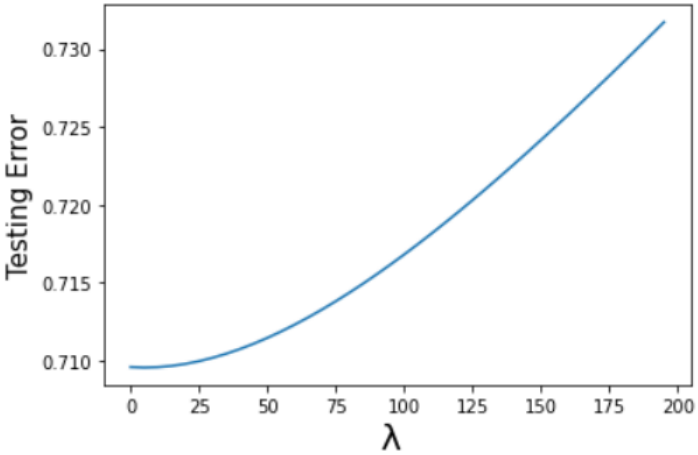
q = 0.5



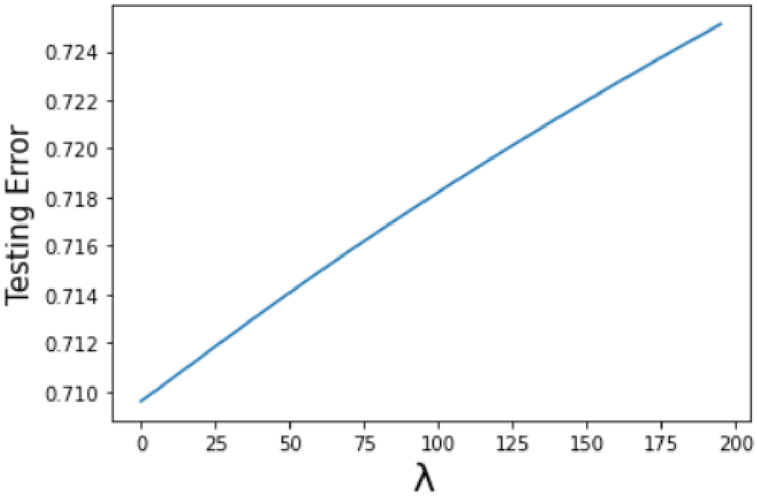
q = 1



q = 2



q = 4



Conclusions: As the above data suggests the unregularized model is better with degree 2. And for the regularized model, all the q values perform pretty much similarly. The only difference regularized and unregularized implementations have is that the testing errors are controlled and hence preventing the overfitting of the model by penalising the weights. Here regularized losses are more since the regularization is applied for degree 1 polynomial and not for the best fit degree polynomial and the unregularized loss which are tabulated above is for the best degree of 4. Hence the difference.

1C)

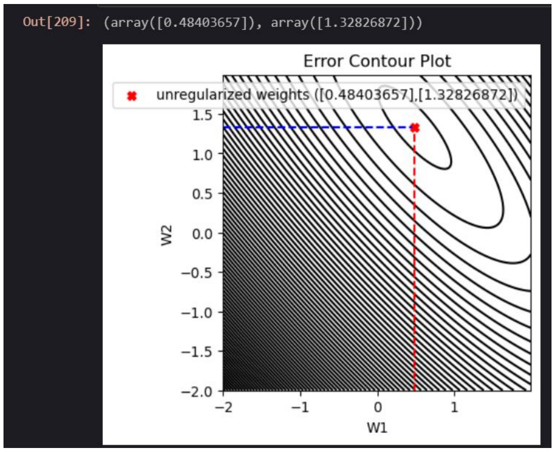
In this assignment (1C), we generate the contour plots for the error function defined in terms of the chemical parameters x1 and x2 and two weight functions w1,w2 expressed as weighed average error against the target values.
The given data set contains three columns, namely, x1, x2 and the target (t). First the dataset file in .csv format is uploaded to the Google Colab.
At the beginning of the python code various necessary libraries are imported, namely, numpy as np, pandas as pd, matplotlib.pyplot as plt, and random.
A function named normalise(X) is defined to estimate the standard normal variable z=(x-mean)/sd.
In the next step, the rows are randomized in the df. The normalized parameters are then stored in X_train and Y_train.

A function is defined for gradient descent method with 1000 epochs. A function named costfunc is also defined, to estimate the function cost. A mesh grid is then generated for X and Y arrays and the surface equation for the error is defined as Z array.

Another function named in_range is defined to generate the points required for plotting. Plotting of the error function contours , constraint function, w1 and w2 unregularized lines, transition path of weights, and the point of regularized weights on the specific contour of the optimum value of the error function satisfying the constraint function are plotted using matplotlib function ‘plt.plot’ for the given combinations of q and η.
Finally the mean square error (MSE) is also estimated for the four cases.

Error contour Plot

Unregularized weights are 0.4840 and 1.3282

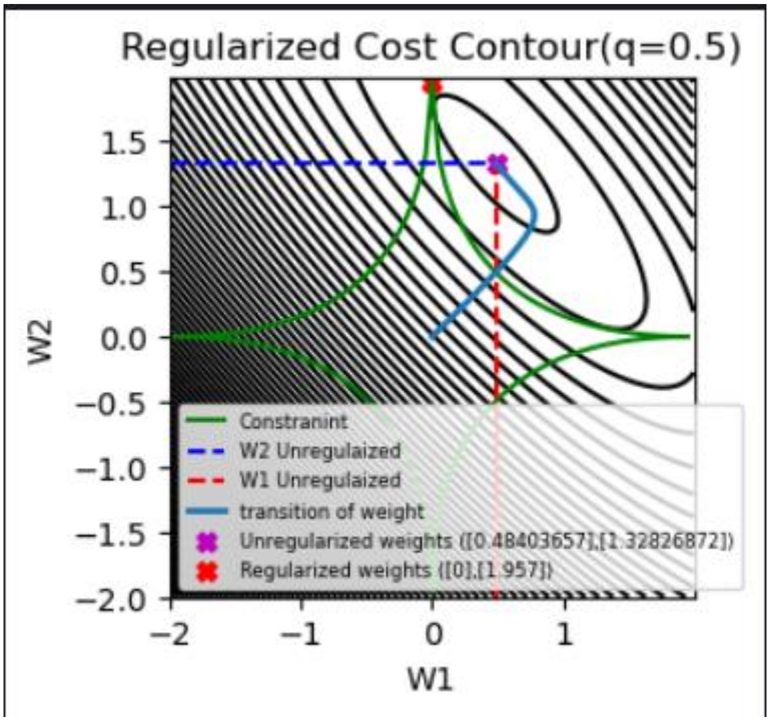


```
Out[209]:  (array([0.48403657]), array([1.32826872]))
```

In the following graphs , green coloured region shows the constraint region and the black ones show the contours of the cost function.

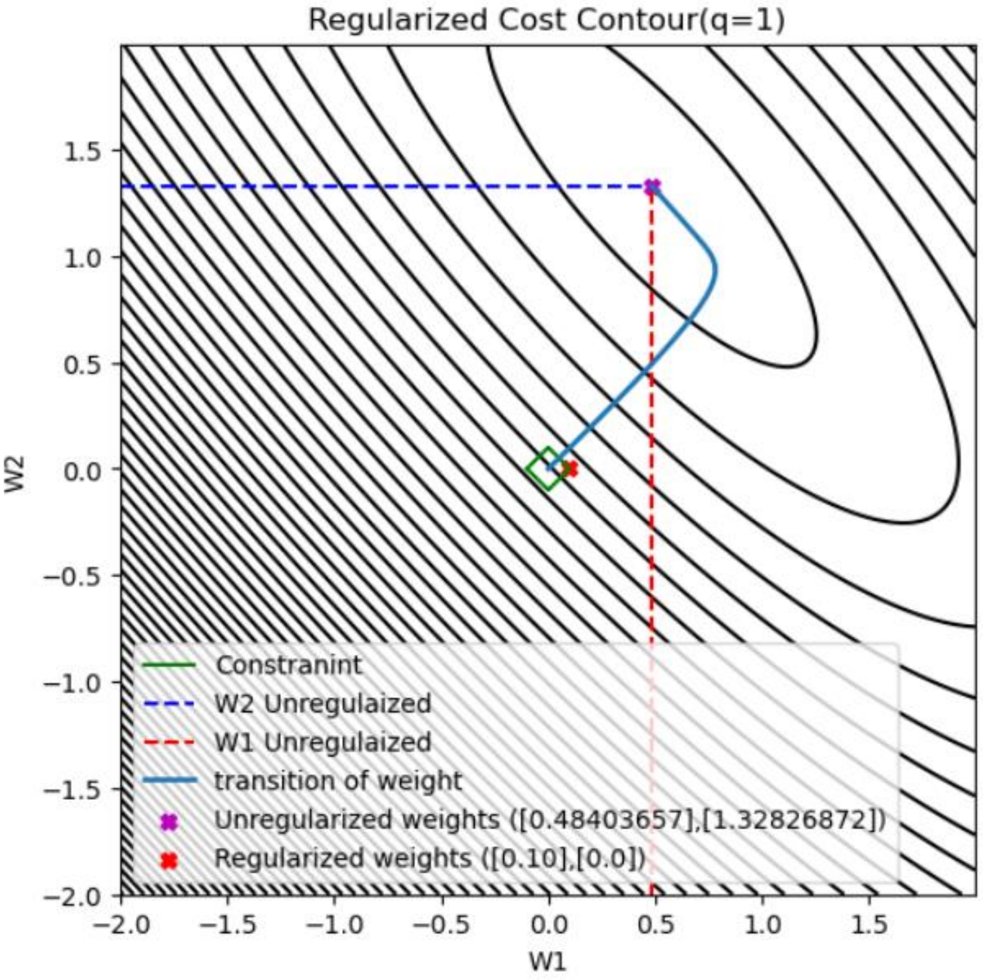
(a) $q = 0.5$ and $\eta = 1.4$

Regularized weights are 0 , 1.957



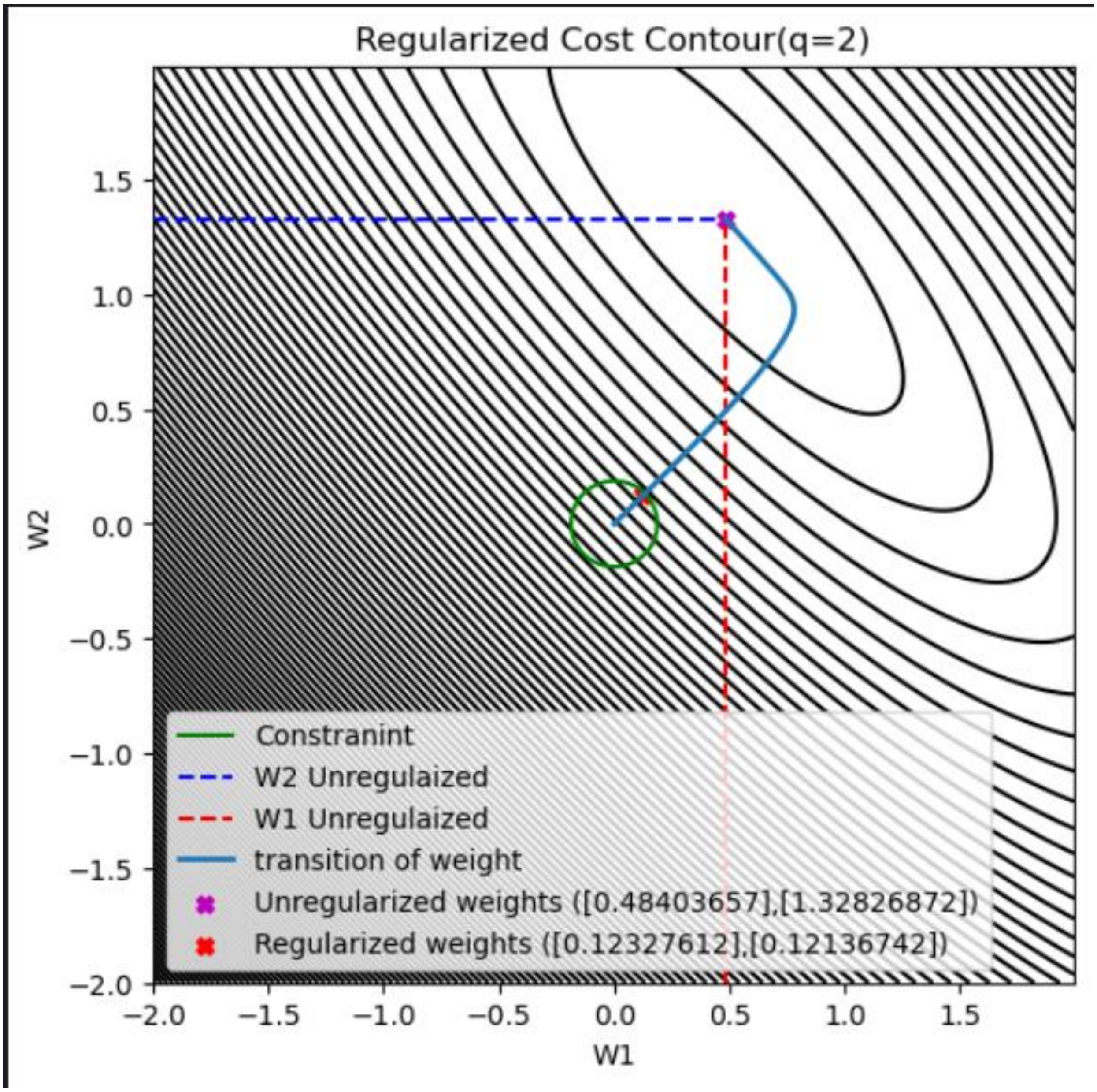
(b) $q = 1$ and $\eta = 0.1$

Regularized weights are 0.1,0



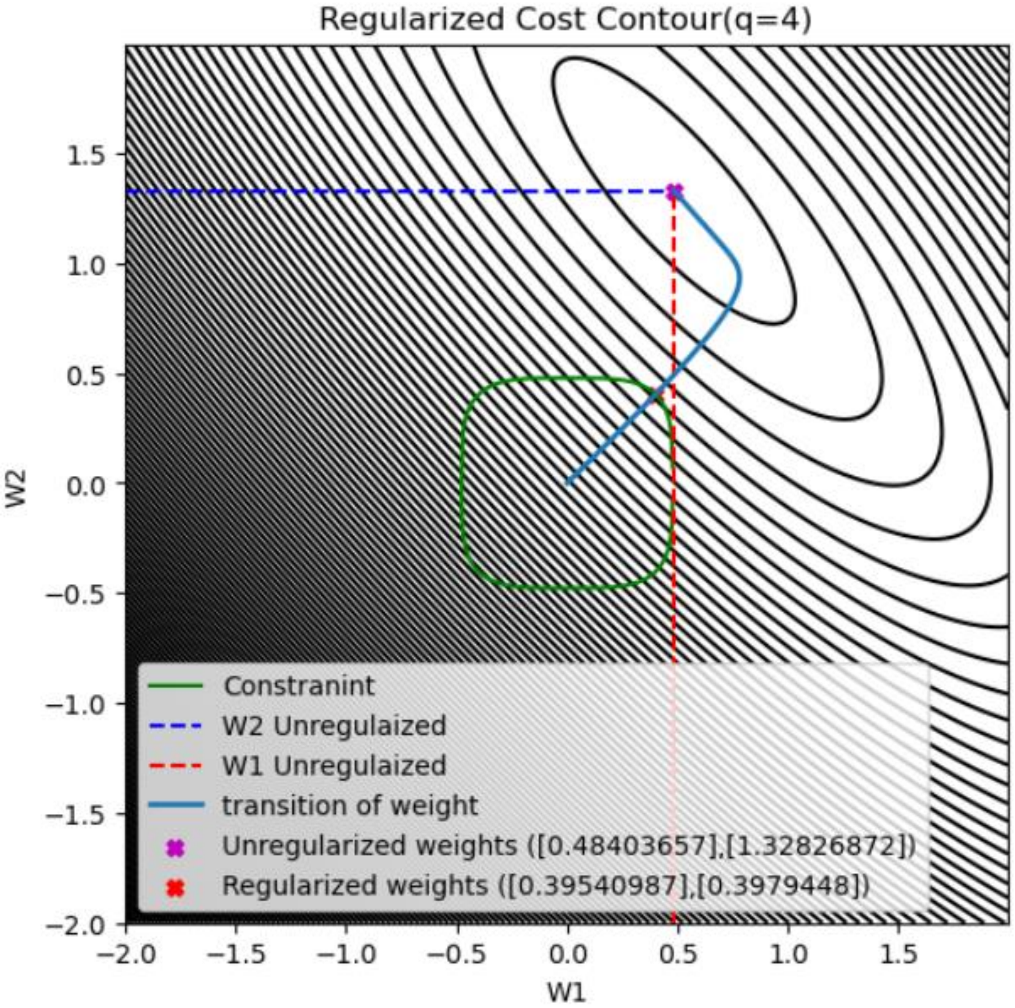
(c) q = 2 and η = 0.035

Regularised weights are 0.123, 0.121



(d) q = 4 and η = 0.052

Regularized weights are 0.395,0.3979



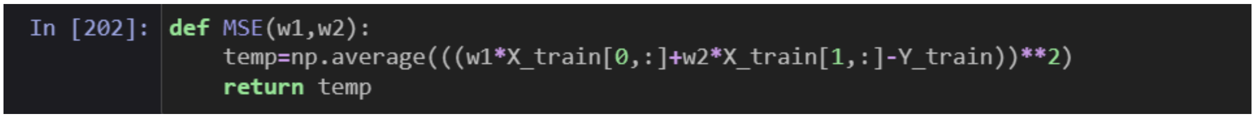
(iii) Mean Squared Errors

After finding the optimal weights we are finding the average and variance of data (unnormalized) and then we are calculating the Mean Squared error(MSE) function.

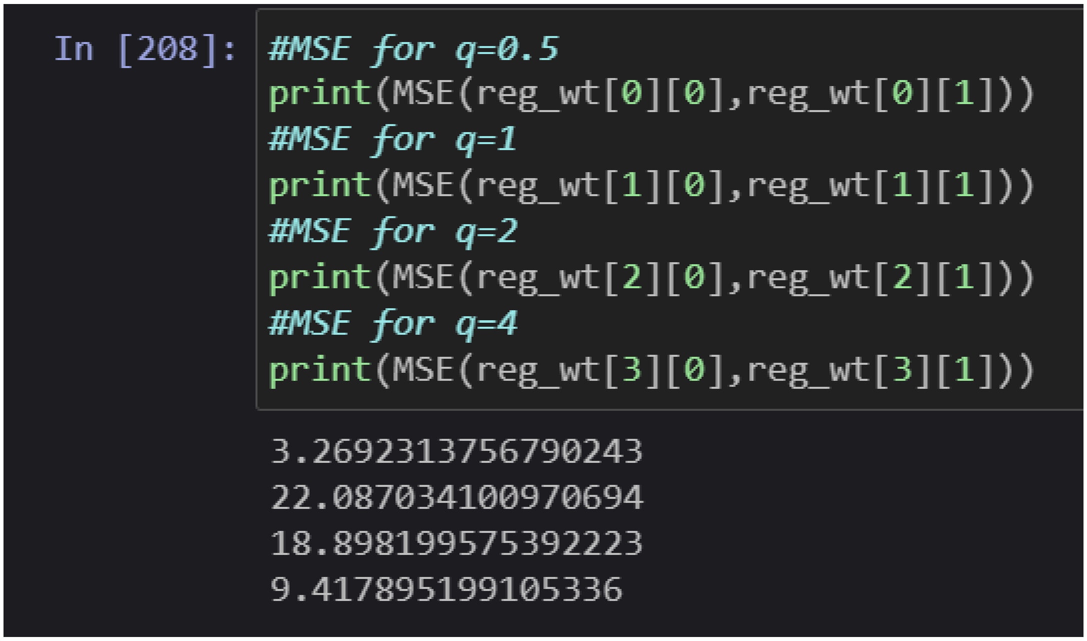
MSE = Σ (y^ - t)^2 / n

Here n is no of examples.

y^ is predicted output and t is actual value.



```
In [202]: def MSE(w1,w2):
              temp=np.average(((w1*X_train[0,:]+w2*X_train[1,:]-Y_train))**2)
              return temp
```



```
In [208]: #MSE for q=0.5
          print(MSE(reg_wt[0][0],reg_wt[0][1]))
          #MSE for q=1
          print(MSE(reg_wt[1][0],reg_wt[1][1]))
          #MSE for q=2
          print(MSE(reg_wt[2][0],reg_wt[2][1]))
          #MSE for q=4
          print(MSE(reg_wt[3][0],reg_wt[3][1]))

          3.2692313756790243
          22.087034100970694
          18.898199575392223
          9.417895199105336
```

MSE for q = 0.5 - > 3.269

MSE for q = 1 -> 22.087

MSE for q = 2 -> 18.898

MSE for q = 3 -> 9.4178