Foundations of Data Science

**Assignment - 1**

**Documentation**

**Submitted By,**

**Abhirath N B - 2020A7PS0260H**

**Sriram Srivatsan - 2020A7PS0273H**

**Amogh Moses - 2020A7PS1199H**

Contents

1A Prior and Posterior Distribution…….……………………………………….3

(i) Describing likelihood of s………….…………………………………….3

(ii) Describing how we found the posterior distribution………………...…..6

1B Polynomial Regression and Regularization…………………………………9

(i) Brief Description of our Model…………………………………………..9

(ii) Tabulating Training and Testing Errors………………………………...12

(iii) Surface Plots of Predicted Polynomials……………………………….14

(iv) Comparative Analysis of all Models…………………………………..19

1C Visualizing Regularization…………………………………………………20

(i) Error Contour Plot………………………………………………………20

(ii) Constraint Regions and Error Function Contours……………………...20

(iii) Mean Squared Errors………………………………………………….24

**1A - Prior and Posterior Distribution**

**(i) Describing the likelihood of s:**

Given ‘s’ follows a beta distribution with parameters () = (2, 2).

The beta distribution is given by:

P(s) =

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Let s be the probability that the customers liked the new software update.

We know that P(Event happening) + P(Event not happening) = 1

⇒ P(customers liking the update) + P(customers not liking the update) = 1

⇒ P(customers not liking the update) = 1 - P(customers liking the update)

⇒ P(customers not liking the update) = 1 - s

***Initially, out of the 50 customers surveyed, 40 of them liked the update and 10 of them did not like the update (Let this be event E1).***

Let L(s) be the likelihood function

L(s) = P(40 customers liked the update, and 10 did not like the update | s)

Since a customer liking/disliking the update is independent of other customers liking/disliking the update, we can write L(s) as

L(s) = s40 \* (1-s)10

Now we need to choose that ‘s’ that maximizes the likelihood of this event(E1) happening, that is L(s) should be maximum

We know that maxs L(s) = maxs log(L(s)) (if L(s) >= 0)

= maxs log(s40 \* (1-s)10)

maxs L(s) = maxs  [40\*log(s) + 10\*log(1-s)]

For the maximum value of L(s),

L’(s) = 0

⇒ (40/s) - (10/(1-s)) = 0

⇒ 40\*(1-s) = 10\*(s)

⇒ 40 - 40\*s = 10\*s

⇒ 50\*s = 40

⇒ s = 40/50

⇒ s = ⅘

Hence, s = ⅘ maximizes the likelihood of the customers liking the software update after the initial survey.

***After a few days, another survey was conducted in which out of the 30 customers surveyed, 13 of them liked the update, and 17 of them disliked the update (Let this be event E2).***

Let L(s) be the likelihood function

L(s) = P(13 customers liked the update, and 17 did not like the update | s)

Since a customer liking/disliking the update is independent of other customers liking/disliking the update, we can write L(s) as

L(s) = s17 \* (1-s)13

Now we need to choose that ‘s’ that maximizes the likelihood of this event(E2) happening, that is L(s) should be maximum

We know that maxs L(s) = maxs log(L(s)) (if L(s) >= 0)

= maxs log(s13 \* (1-s)17)

maxs L(s) = maxs  [13\*log(s) + 17\*log(1-s)]

For the maximum value of L(s),

L’(s) = 0

⇒ (13/s) - (17/(1-s)) = 0

⇒ 13\*(1-s) = 17\*(s)

⇒ 13 - 13\*s = 17\*s

⇒ 30\*s = 13

⇒ s = 13/30

Hence, s = 13/30 maximizes the likelihood of customers liking the software update after this survey.

***Again, a final survey was conducted in which 70 out of the 100 people surveyed liked the update, and the remaining 30 did not like the update. (Let this be event E3)***

Let L(s) be the likelihood function

L(s) = P(70 customers liked the update, and 30 did not like the update | s)

Since a customer liking/disliking the update is independent of other customers liking/disliking the update, we can write L(s) as

L(s) = s70 \* (1-s)30

Now we need to choose that ‘s’ that maximizes the likelihood of this event(E3) happening, that is L(s) should be maximum

We know that maxs L(s) = maxs log(L(s)) (if L(s) >= 0)

= maxs log(s70 \* (1-s)30)

maxs L(s) = maxs  [70\*log(s) + 30\*log(1-s)]

For the maximum value of L(s),

L’(s) = 0

⇒ (70/s) - (30/(1-s)) = 0

⇒ 70\*(1-s) = 30\*(s)

⇒ 70 - 70\*s = 30\*s

⇒ 100\*s = 70

⇒ s = 70/100

⇒ s = 7/10

Hence, s = 7/10 maximizes the likelihood of customers liking the software update after the final survey.

**(ii) Describing how we found the posterior distribution:**

Given ‘s’ follows a beta distribution with parameters ().

Here, **P(s) denotes the prior distribution.**

Given the probability of customers liking the update is ‘s’, the probability of m customers liking the update and l customers disliking the update is

**P(D|s) = sm \* (1 - s)l**

**This is the likelihood function.**

Now the posterior distribution can be found using Bayes’ theorem.

Bayes Theorem is given by

P(s|D) = P(D|s) \* P(s)

\_\_\_\_\_\_\_\_\_\_\_\_

P(D)

P(D) = P(D, s) \* ds (P(x) = P(x, y) from

Sum rule of Probability)

= P(D|s) \* P(s) ds (P(x, y) = P(x|y) \* P(y) from

Product rule of probability)

The above term P(D) will be constant as it is integral with respect to all possible values of ‘s’.

⇒ **P(D) = constant**

Now, from Bayes’ Theorem,

P(s|D) ∝ P(D|s) \* P(s) (since P(D) is constant)

P(D|s) = sm \* (1 - s)l

P(s) =

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

⇒ P(s|D) ∝ sm \* (1 - s)l \*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ignoring constants , , and, we can write,

P(s|D) ∝

⇒ P(s|D) =

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

After performing some calculations, we get

**P(s|D) = Beta distribution (s; + m, + l)**

**P(s|D) denotes Posterior distribution.**

That is, after having seen D training examples, ‘s’ follows a beta distribution with parameters + m, and + l.

**Using this above-described method in the given question:**

Initially, the values of the parameters and are 2 and 2, respectively.

The prior distribution, in this case, is given by P(s) = Beta Distribution (s; 2, 2)

Given, 40 people liked the update, and 10 people disliked the update.

⇒ m = 40 and l = 10 as described above.

⇒ The posterior distribution is given by

P(s|D) = Beta Distribution (s; 2 + 40, 2 + 10)

⇒ P(s|D) = Beta Distribution (s; 42, 12)

Now the calculated posterior distribution value will be taken as the prior distribution for the next case (second survey).

Given, 13 customers liked the update, and 17 customers disliked the update.

⇒ m = 13 and l = 17 as described above.

⇒ The Posterior distribution after the second survey is given by

P(s|D) = Beta distribution (s; 42 + 13, 12 + 17)

⇒ P(s|D) = Beta distribution (s; 55, 29)

Now the calculated posterior distribution value will be taken as prior distribution for the next case (final survey).

Given, 70 customers liked the update, and 30 customers disliked the update.

⇒ m = 70 and l = 30 as described above.

⇒ The Posterior distribution after the final survey is given by

P(s|D) = Beta distribution (s; 55 + 70, 29 + 30)

⇒ P(s|D) = Beta distribution (s; 125, 59)

**1B - Polynomial Regression and Regularization**

**(i) Brief Description of our model, algorithms, and how we implemented regularization:**

**For degree 1 polynomial:**

We assumed our model to be y = w0 + w1x1 + w2x2

First we split the data in the ratio of 80:20 as training data and testing data respectively.

Let yn be the value of LC50 for the n’th training example.

⇒ **Without regularization, the error function is as below:**

E(w0, w1, w2) = (1/2N) \* (yn - (w0 + w1x1n + w2x2n))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.

In order to reach the minimum, we can use the gradient descent method or stochastic gradient descent method.

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) - \* (∇E)w = w(k),

denotes scaling parameter ( > 0), and

(∇E) =

|  | ∂E/∂w0 |  |
| --- | --- | --- |
|  | ∂E/∂w1 |  |
|  | ∂E/∂w2 |  |

Here ∂E/∂w0 = (1/2N) \* 2 \* (yn - (w0 + w1x1n + w2x2n))\* (-1)

∂E/∂w1 = (1/2N) \* 2 \* (yn - (w0 + w1x1n + w2x2n))\* (-x1n)

∂E/∂w2 = (1/2N) \* 2 \* (yn - (w0 + w1x1n + w2x2n))\* (-x2n)

In **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 exactly the same as the Gradient descent method except that we will choose one random training example (x1t, x2t, yt) 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(w0, w1, w2) = (1/2) \* (yn - (w0 + w1x1n + w2x2n))2

w(k+1) = w(k) - \* (∇E)w = w(k)

∂E/∂w0 = (1/2) \* 2 \* (yt - (w0 + w1x1t + w2x2t))\* (-1)

∂E/∂w1 = (1/2) \* 2 \* (yt - (w0 + w1x1t + w2x2t))\* (-x1t)

∂E/∂w2 = (1/2) \* 2 \* (yt - (w0 + w1x1t + w2x2t))\* (-x2t)

**With regularization, the error function is as below:**

E(w0, w1, w2) = (1/2N) \* (yn - (w0 + w1x1n + w2x2n))2 + 2) \* (|w1|q + |w2|q)

where denotes the balancing factor.

Here, we can again apply the gradient descent and stochastic gradient descent methods as described above using this modified error function.

**Generalizing for polynomial models of any degree:**

The methods described above can be applied to polynomial models of any degree in a similar way.

**How we implemented regularization:**

First, we normalized the x-data using the min-max normalization technique. Then we split the given data into training and testing data in the ratio 80:20. Next, we created a data frame having RDCHI and MLOGP in one table for the training data under the name of DataX.

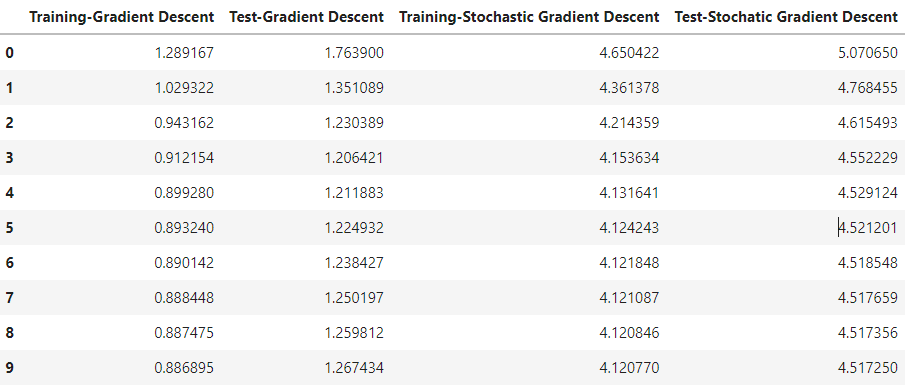
Then we created a data frame having a qualitative response of LC50 in one table for the training data under the name of DataY. To calculate the error using the gradient descent method, we made a table having values of each term in every degree of the polynomial.

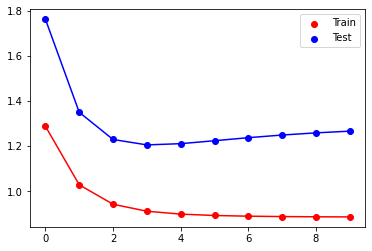
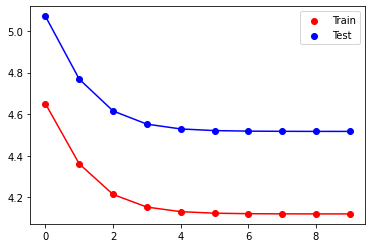
ValX and testXVal store the value of all the x terms possible from degree 0 to degree 9 (like x1, x2, x1\*x2, x12, x22 ……, x19, x29) with respect to each data point in training data and testing data, respectively. Then we fixed the number of iterations as 10000, and for storing the value of ∂E/∂w0, we made an array called costArr. We fixed the learning rate as 0.001.

Then we ran a loop for 10000 iterations and kept on updating the w values using the formula wnew = wold - (learningRate) \* ∂E/∂w0 for each degree. After that, we plotted the values of ∂E/∂w0 with respect to the iterations. We then ran a loop through all the data points and calculated the values of (yi - w \* xi)2 with respect to each data point and then added all the values. Then we divided the sum by 2 \* (the size of the training data) to get the corresponding training error. Then we stored that value in the TrainingErr array. Then we divided the sum of (yi - w \* xi)2 by 2 \* (the size of the testing data) to get the corresponding testing error. Then we stored the value in TestingErr array. We repeated the process for all degrees from 1 to 9.

This is how we implemented regularization.

**(ii) Tabulating the training and testing errors obtained using polynomial regression models of various degrees:**

****

** **

**Plots of training errors and testing errors for polynomial regression models of various degrees using the gradient descent and stochastic gradient descent methods, respectively.**

**Observations on Overfitting:**

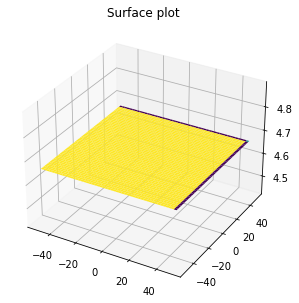
In general, a regression model is said to be overfitting the data if it performs well on the training data but does not perform well on testing data.

From the above plot of training and testing errors for models built using the *gradient descent method*, the polynomial models of degree 3 onwards give almost the same error, which is the least error for the testing data. Among polynomial regression models of degree > 3, we will choose the **polynomial regression model of degree 3 as the best model** by using the principle of Ockham’s Razor (We will choose the simpler model). Hence, this polynomial model of degree 3 is the best model. The polynomial models of degrees higher than 3 fit the training data better than the model of degree 3. So, we can say that the polynomial models of degrees greater than 3 overfit the given data.

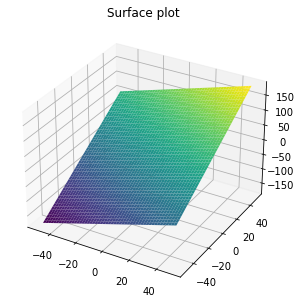
Similarly, from the above plot of training and testing errors for models built using the *stochastic gradient descent method*, the polynomial models of degree 4 onwards give almost the same error, which is the least error for the testing data. Among polynomial regression models of degree > 4, we will choose the polynomial regression model of degree 4 as the best model by using the principle of Ockham’s Razor (We will choose the simpler model). Hence, this polynomial model of degree 4 is the best model. The polynomial models of degrees higher than 4 fit the training data better than the model of degree 4. So, we can say that the polynomial models of degrees greater than 4 overfit the given data.

**(iii) Surface plots of the predicted polynomials:**

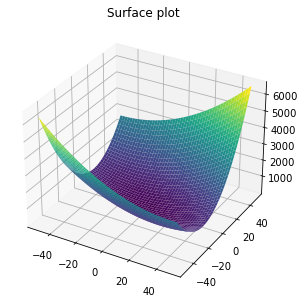
* **Surface plot of degree 0 polynomial:**

****

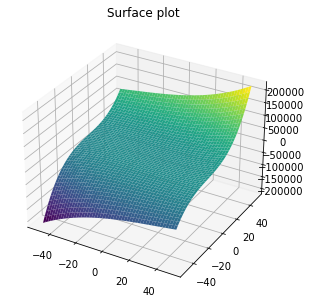
* **Surface plot of degree 1 polynomial:**

****

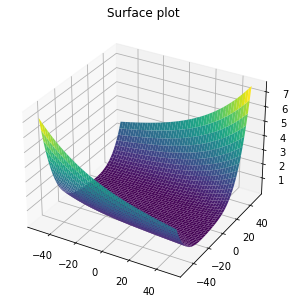
* **Surface plot of degree 2 polynomial:**

****

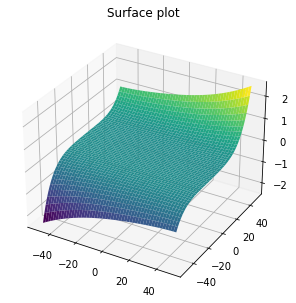
* **Surface plot of degree 3 polynomial:**

****

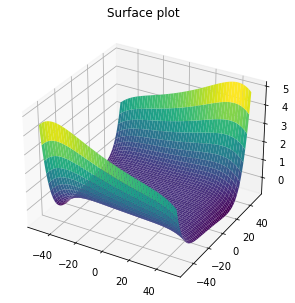
* **Surface plot of degree 4 polynomial:**

****

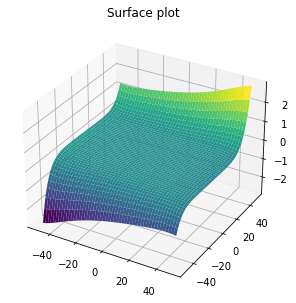
* **Surface plot of degree 5 polynomial:**

****

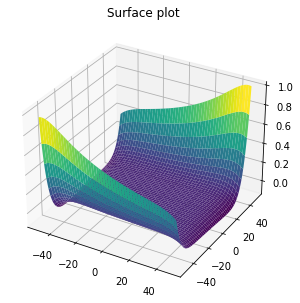
* **Surface plot of degree 6 polynomial:**

****

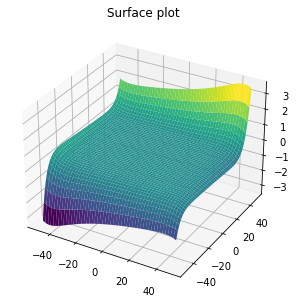
* **Surface plot of degree 7 polynomial:**

****

* **Surface plot of degree 8 polynomial:**

****

* **Surface plot of degree 9 polynomial:**

****

**(iv) Comparative analysis study of the four optimal regularized regression models and best-fit classic polynomial regression model.**

| **Model** | **q** | **Training Error** | **Testing Error** |
| --- | --- | --- | --- |
| Best-fit model  (unregularized degree-3 model is best from 1B (a)) | - | 0.91254 | 1.206421 |
| Regularized | 0.5 | 1.36272555 | 2.72857482 |
|  | 1 | 1.36272555 | 2.72857486 |
|  | 2 | 1.36272555 | 2.72858037 |
|  | 4 | 1.36272555 | 2.74283949 |

From the above table, we can see that the unregularized regression model of degree 3 fits the testing data very well as the value of testing error is the least. The polynomial regression model of degree 1 underfits the training data. Even if we regularize the degree 1 polynomial model, we will not get as powerful a model as the unregularized degree-3 polynomial model. So, we can say that the best-fit unregularized regression model of degree 1 is the best model among these five models. Also, we may be able to obtain a similar model to the best-fit unregularized model of degree 3 if we regularize a polynomial regression model of a degree higher than 3.

**1C - Visualizing Regularization**

**(i), (ii) Plotting error function contours and constraint regions, showing the tangential contour and the point of intersection where the minima occur:**

The polynomial regression model is yn = w1xn1 + w2xn2

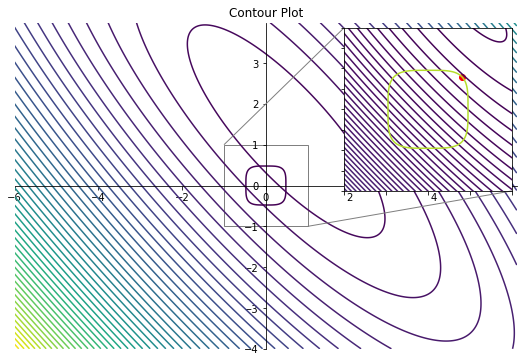
where xn1 and xn2 represent the first and second features of the nth sample.

To find the best model, we minimize

E(w) = (1/2) \* (yn - tn)2 subject to |w1|q + |w2|q

where N is the total number of samples.

* **The plots of the error function contour and constraint region for q = 4 and = 0.052 are as below:**

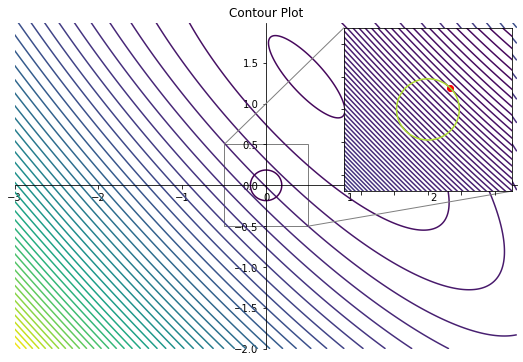


The red-colored point is the point of intersection of the tangential contour and the constraint region |w1|4 + |w2|4 0.052

The values of w1 and w2 obtained from the intersection of the tangential contour and the constraint region are

**w1 = 0.4 and w2 = 0.4**

* **The plots of the error function contour and constraint region for q = 2 and = 0.035 are as below:**

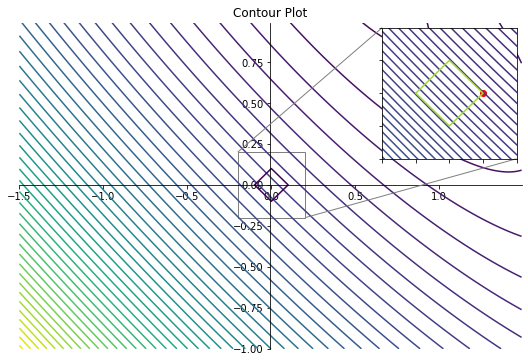


The red-colored point is the point of intersection of the tangential contour and the constraint region |w1|2 + |w2|2 0.035

The values of w1 and w2 obtained from the intersection of the tangential contour and the constraint region are

**w1 = 0.13 and w2 = 0.13**

* **The plots of the error function contour and constraint region for q = 1 and = 0.1 are as below:**

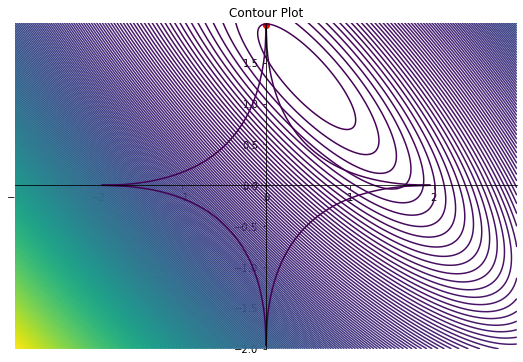


The red-colored point is the point of intersection of the tangential contour and the constraint region |w1|1 + |w2|1 0.1

The values of w1 and w2 obtained from the intersection of the tangential contour and the constraint region are

**w1 = 0.1 and w2 = 0.0**

* **The plots of the error function contour and constraint region for q = 0.5 and = 1.4 are as below:**



The red-colored point is the point of intersection of the tangential contour and the constraint region |w1|0.5 + |w2|0.5 1.4

The values of w1 and w2 obtained from the intersection of the tangential contour and the constraint region are

**w1 = 0.0 and w2 = 1.96**

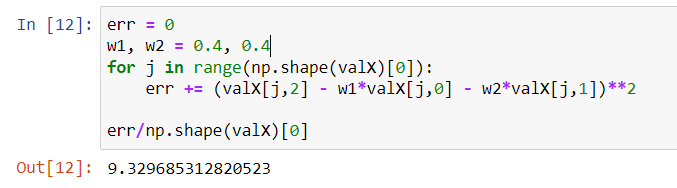
**NOTE:**

**For q = 1, 2 and 4, to make the plots look better we reduced the number of contours, and when we are zooming it we increased the number of contours to the intersection point between them and get the values of w1 and w2.**

**(iii) Calculation of mean squared errors for each polynomial regression model using the obtained weights:**

* **For q = 4 and = 0.052, we get w1 = 0.4 and w2 = 0.4**

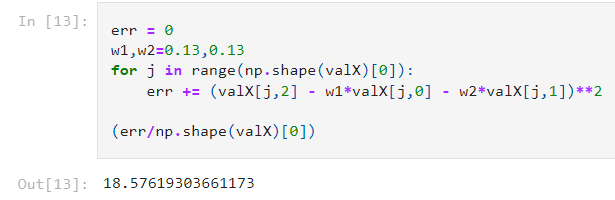
Calculation of mean squared error ((1/N) \*(yn - tn)2) is as shown below:



Thus, the mean squared error value, in this case, is **9.329685312820523**

* **For q = 2 and = 0.035, we get w1 = 0.13 and w2 = 0.13**

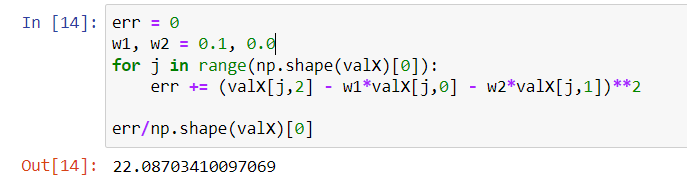
Calculation of mean squared error ((1/N) \*(yn - tn)2) is as shown below:



Thus, the mean squared error value, in this case, is **18.57619303661173**

* **For q = 1 and = 0.1, we get w1 = 0.1 and w2 = 0.0**

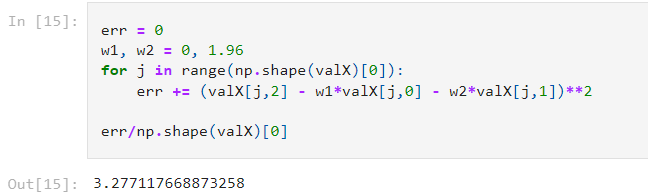
Calculation of mean squared error ((1/N) \*(yn - tn)2) is as shown below:



Thus, the mean squared error value, in this case, is **22.08703410097069**

* **For q = 0.5 and = 1.4 we get w1 = 0.0 and w2 = 1.96**

Calculation of mean squared error ((1/N) \*(yn - tn)2) is as shown below:



Thus, the mean squared error value, in this case, is **3.277117668873258**