Arnav Sudan
2019TT10954

Go to Handwritten Digit Recognition using logistic regression

Study of different parameters in Polynomial curve fitting.

1. Introduction

Given points (xi, yi) in the file https://web.iitd.ac.in/~seshan/a1/group23.txt
The relationship is of the form y = w0 + w1x + w2x^2 + … + WM X^M + E

Our aim is to identify the polynomial and estimate of the noise variance using regression by minimising the **error function** using **gradient descent.**

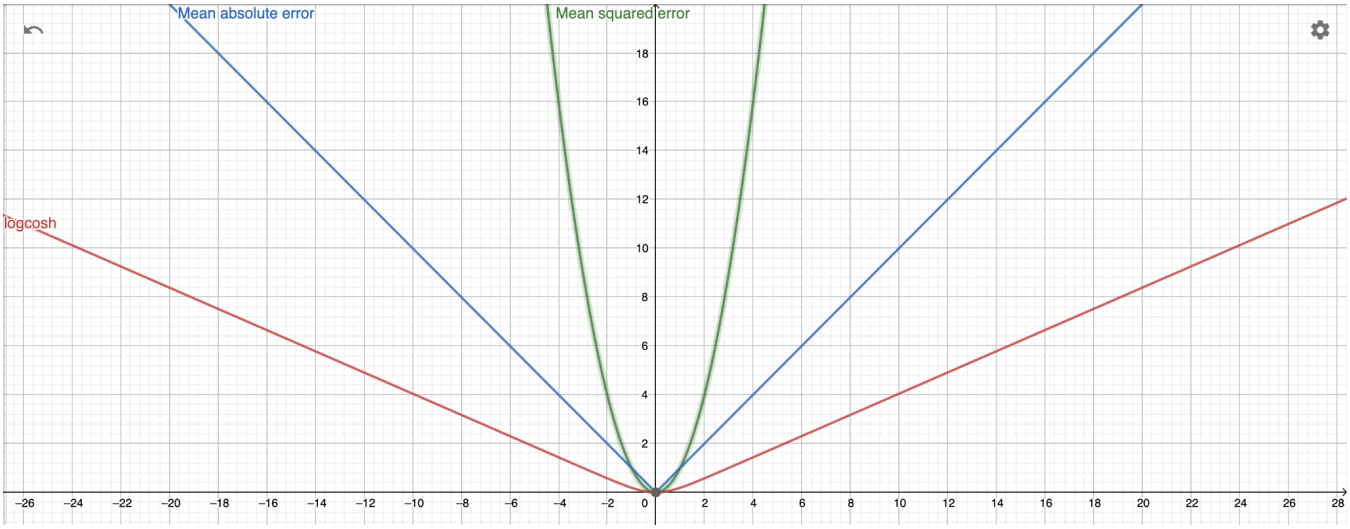
We have studied the following parameters

- Error functions
  - Mean Squared Error
  - Mean Absolute Error
  - Huber
  - Logcosh
  - Mean Squared Log Error
- Degree
- Regularisation
- Number of data points for training

2. Error functions used

For all the experiments with different error functions the following parameters were used:

- No regularisation(lambda = 0)
- Training points = 80
- Testing points = 20
- Degree = 6



2.1 Mean squared error(MSE)

Mean squared error(also called Quadratic loss) function is the most commonly used regression loss function.
It's only concerned with the average magnitude of error irrespective of their direction.
As it has a square term, it isn't robust to outliers.

Formula used

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

Observation

Due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions. In the experiment we observed that the initial decrease in error when error was high was high and in only 888000 iterations the mean squared error was reduced to the minimum(0.2222) with learning rate 0.1.

Results

```
The polynomial is:
477.3073868228952*X^0 15.302680792148 54*X^1 88.74976413350606*X^2 1638.5876078881572*X^3 806.936805259228*X^4 17595.8
5537477385*X^5 0.0368965027924299*X^6

The Training root mean square error is 0.6666481977915485
The Testing root mean square error is 0.892877412930049
noise variance: 0.7972300745206572
```

2.2 Mean absolute error(MAE)

MAE is the sum of absolute differences between our target and predicted variables. So it measures the average magnitude of errors in a set of predictions, without considering their directions.
Unlike MSE, MAE may miss the minima as the gradient is higher and MAE is more robust to outliers since it does not make use of square.

Formula used

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

Observation

As MAE doesn't have a square term it is taking a lot of iterations to reach minimum value. Sum of mean squared errors in 160000th iteration is 11.535093250034878 with learning rate = 0.1 and has missed the minima in our experiment. The experiment had to be reconducted with learning rate = 0.001 as with the previous learning rate the gradient missed the minimum value and even with learning rate = 0.001 the minima was missed.

Results

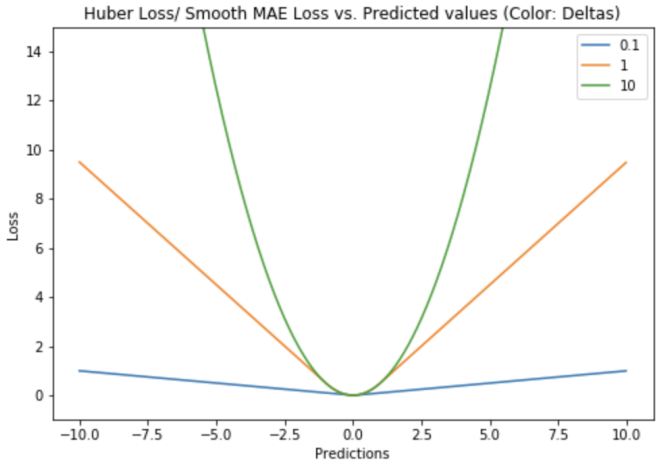
Learning rate = 0.1

```
The polynomial is:
481.5267460116499*X^0 -2.1350953469362572*X^1 134.6279018906703*X^2 1723.119854712826*X^3 694.358680926573*X^4 17523.
307981734393*X^5 70.07914079824549*X^6

The Training root mean square error is 4.803143397824987
The Testing root mean square error is 4.733268037152364
noise variance: 22.403826311528192
```

Learning rate = 0.001

```
The polynomial is:
477.2573626075378*X^0 15.977577688701489*X^1 87.97598394132909*X^2 1634.9207815623079*X^3 810.0971645334602*X^4 1759
9.353403480112*X^5 -2.625437077393102*X^6

The Training root mean square error is 0.6755109039055105
The Testing root mean square error is 0.9195827693363963
noise variance: 0.8456324696603957
```

2.3 Huber



Plot of Hoss Loss (Y-axis) vs. Predictions (X-axis). True value = 0

Can't use higher learning rate varna not converging.

sum of mean squared errors in 1999000th iteration is 0.22224962531666198

Huber loss is less sensitive to outliers in data than the squared error loss. It's also differentiable at 0. It's basically absolute error, which becomes quadratic when error is small. How small that error has to be to make it quadratic depends on a hyperparameter, δ (delta), which can be tuned. Huber loss approaches MSE when $\delta \sim 0$ and MAE when $\delta \sim \infty$ (large numbers.)
The choice of delta is critical because it determines what you're willing to consider as an outlier. Residuals larger than delta are minimized with L1 (which is less sensitive to large outliers), while residuals smaller than delta are minimized "appropriately" with L2.

Formula used

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \leq \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Observation

As we have seen in MAE for even low learning rate MAE misses the minima because of its constantly large gradient.

For MSE, gradient decreases as the loss gets close to its minima, making it more precise.

Huber loss can be really helpful in such cases, as **it curves around the minima** which decreases the gradient. **And it's more robust to outliers than MSE**. As a result, it combines the best properties of both MSE and MAE.
The issue with Huber loss is that it is possible that we will need to train hyperparameter delta, which is an iterative process.

Results

```
parameters used

degree : 6
error Function : <function grad_huber at 0x7fd08b1d7e50>
Lamda for L2 regularisation : 0
Number of training points: 80
Number of testing points: 20


The polynomial is:
477.3075838190162*X^0 15.293071800325684*X^1 88.92313694391228*X^2 1638.5988126903394*X^3 806.5121967302701*X^4 1759
5.859149808693*X^5 0.30696976911541574*X^6

The Training root mean square error is 0.666705429408371
The Testing root mean square error is 0.8922250571158438
noise variance: 0.7960655525453706
```

2.4 Logcosh

Log-cosh is another function used in regression tasks that's smoother than L2. Log-cosh is the logarithm of the hyperbolic cosine of the prediction error.

Advantage:
`log(cosh(x))` is approximately equal to `(x ** 2) / 2` for small `x` and to `abs(x) - log(2)` for large `x`. This means that 'logcosh' is robust to outliers. It has all the advantages of Huber loss, and it's twice differentiable everywhere, unlike Huber loss.

Why do we need a 2nd derivative?
Many ML model implementations like XGBoost use Newton's method to find the optimum, which is why the second derivative (Hessian) is needed.

Formula used

$$L(y, y^p) = \sum_{i=1}^{n} \log(\cosh(y_i^p - y_i))$$

Observations

The decrease in error was very low initially. Curves around minima and more robust to outliers.

Results

```
parameters used

degree : 6
error Function : <function grad_logcosh at 0x7fcfa8095d30>
Lamda for L2 regularisation : 0
Number of training points: 80
Number of testing points: 20


The polynomial is:
477.28673994484 91*X^0 15.429828988876453*X^1 88.88112798658193*X^2 1638.2745732936019*X^3 806.7800796918648*X^4 1759
6.027578185927*X^5 0.08449050670056493*X^6

The Training root mean square error is 0.6671213705975483
The Testing root mean square error is 0.8979719239628647
noise variance: 0.8063535762255689
```

2.5 Mean squared log error

MSLE penalizes underestimates more than overestimates
Use MSLE when doing regression, believing that your target, conditioned on the input, is normally distributed, and you don't want large errors to be significantly more penalized than small ones, in those cases where the range of the target value is large.

In the case of MSE, the presence of outliers can make the error term very high. But, in the case of MLSE the outliers are drastically scaled down thus making it more robust to outliers.

Formula used

$$\frac{1}{n}\sum_{i=1}^{n}(\log(x_i+1) - \log(y_i+1))^2$$

Results

```
parameters used

degree : 6
error Function : <function grad_mean_sqaured_log_error at 0x7fbae00d51f0>
Lamda for L2 regularisation : 0
Number of training points: 80
Number of testing points: 20


The polynomial is:
477.3099681241549*X^0 16.059563645314128*X^1 85.45100842286072*X^2 1635.6420634825456*X^3 817.262817313504*X^4 17598.
030245423928*X^5 -7.581772893623695*X^6

The Training root mean square error is 0.6834371815635446
The Testing root mean square error is 0.9109904261626083
noise variance: 0.8299035565599306
```

2.6 Final conclusion of error function

error function	training error	testing error
MSE	0.666648	0.892877
MAE	0.675511	0.919583
Huber	0.666705	0.892225
Logcosh	0.667121	0.897972
Mean Squared log error	0.683437	0.91099

- Huber and logcosh behave similarly but as logcosh started oscillating when gradient was about 0 so logcosh needs lower learning rate and high iterations for the same error as Huber
- As MSE has squared term so it fits the noise and hence low training error and higher testing error than Huber.
- MAE didn't reach the minimum. To reach its minimum a very low learning rate with very high iterations is needed.

- Mean squared log error didn't fit the data well at all because of the log term thus it has the highest training error also it was very slow while converging.

3. Degree of the polynomial

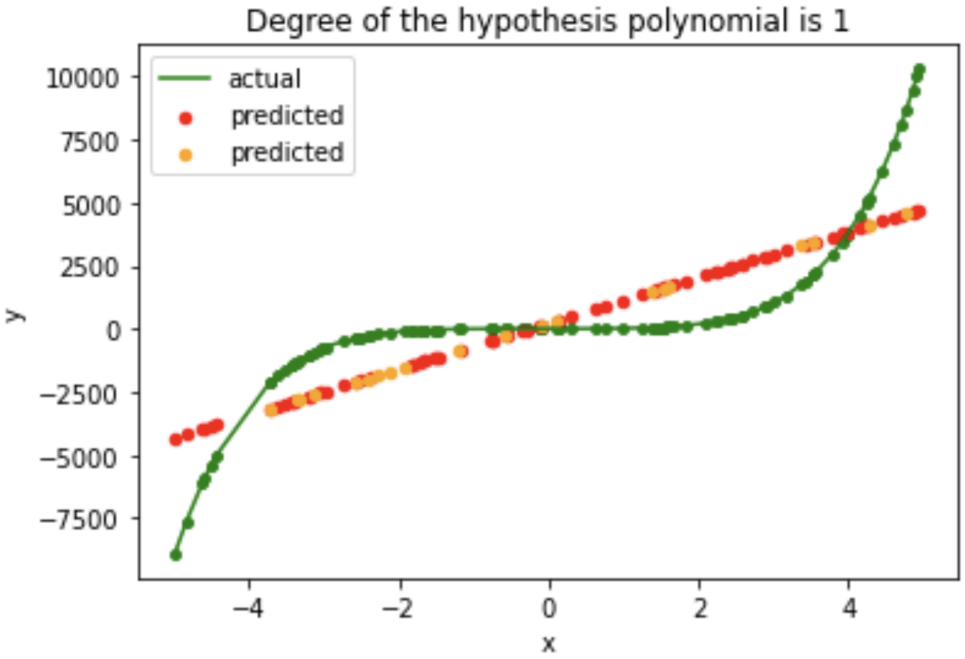
For this experiment the following parameters were used

- no regularisation(lambda = 0)
- Number of training points = 80
- Number of testing points = 20
- Learning rate = 0.01

Note: The orange and red points in the graph are predicted from the testing and training set respectively.

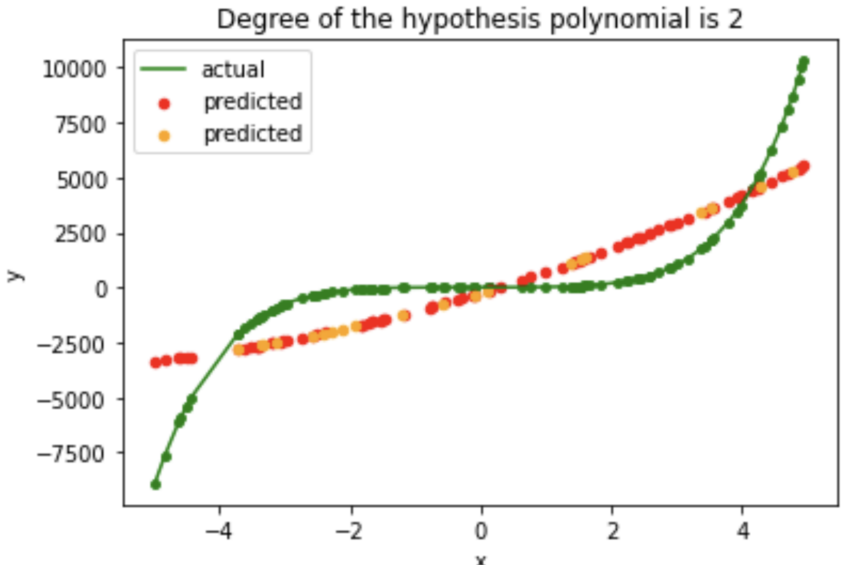
3.1 Degree = 1

```
The polynomial is:
353.71628081206035*X^0 9026.573640220893*X^1

The Training root mean square error is 1889.255724183592
The Testing root mean square error is 1609.961853942211
noise variance: 2591977.1711490415
```



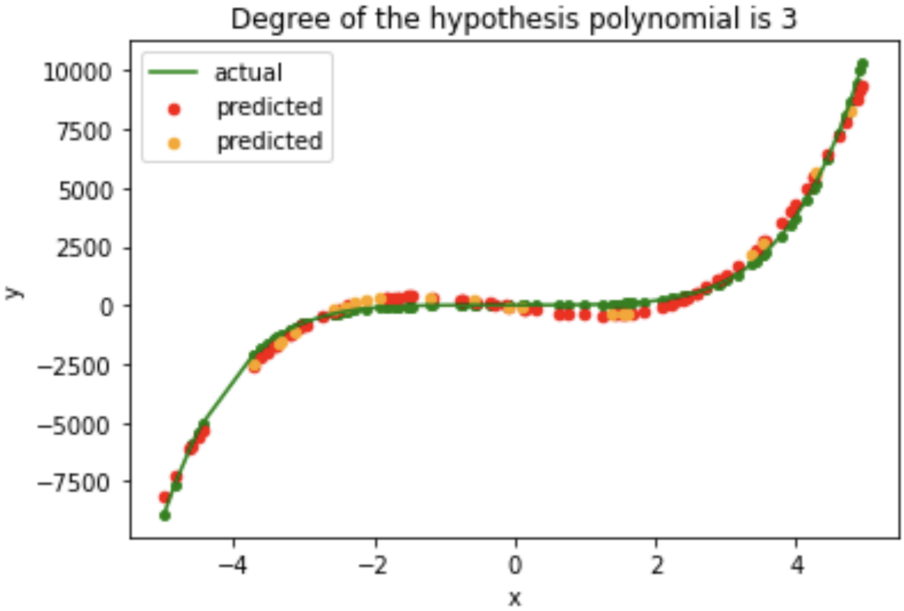
3.2 Degree = 2

```
The polynomial is:
340.80478624695974*X^0 8908.222168164682*X^1 1353.8986898301346*X^2

The Training root mean square error is 1845.2917906932385
The Testing root mean square error is 1506.9847075635892
noise variance: 2271002.908830516
```



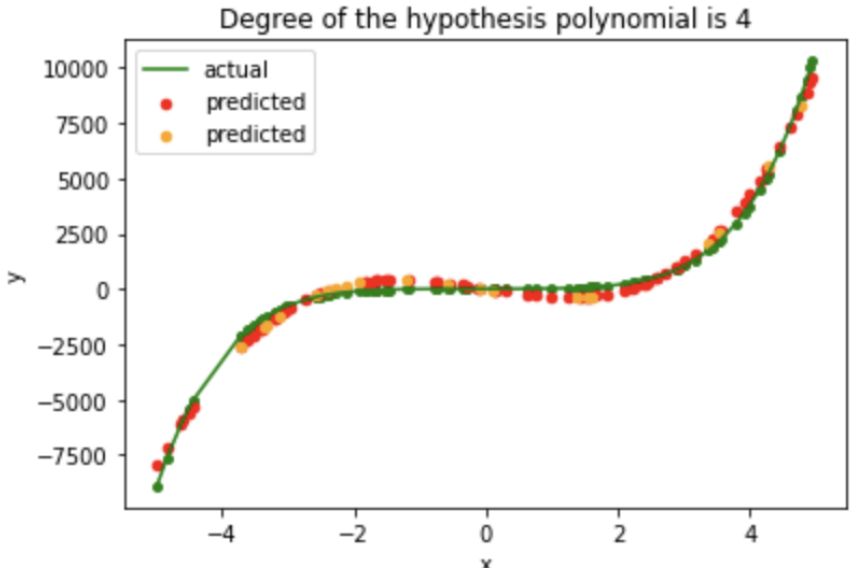
3.3 Degree = 3

```
The polynomial is:
479.67366512389333*X^0 -4523.244051713849*X^1 802.2244260232877*X^2 22071.826805754914*X^3

The Training root mean square error is 365.74640012984275
The Testing root mean square error is 345.7701314545117
noise variance: 119556.98380607033
```



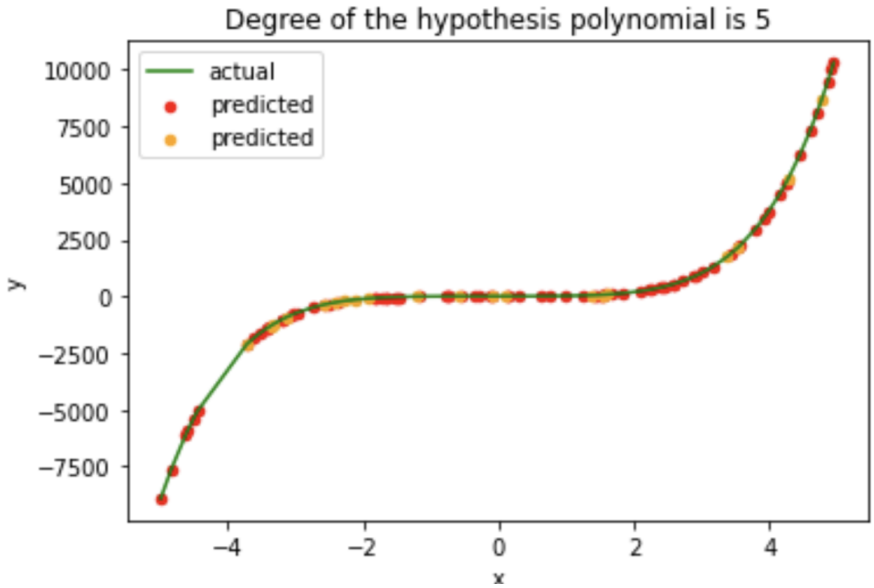
3.4 Degree = 4

```
The polynomial is:
476.95984672374374*X^0 -4495.421172026689*X^1 18.218765270657816*X^2 22019.51542355236*X^3 873.7326251664875*X^4

The Training root mean square error is 358.81584327308195
The Testing root mean square error is 338.22742711293586
noise variance: 114397.79245143635
```



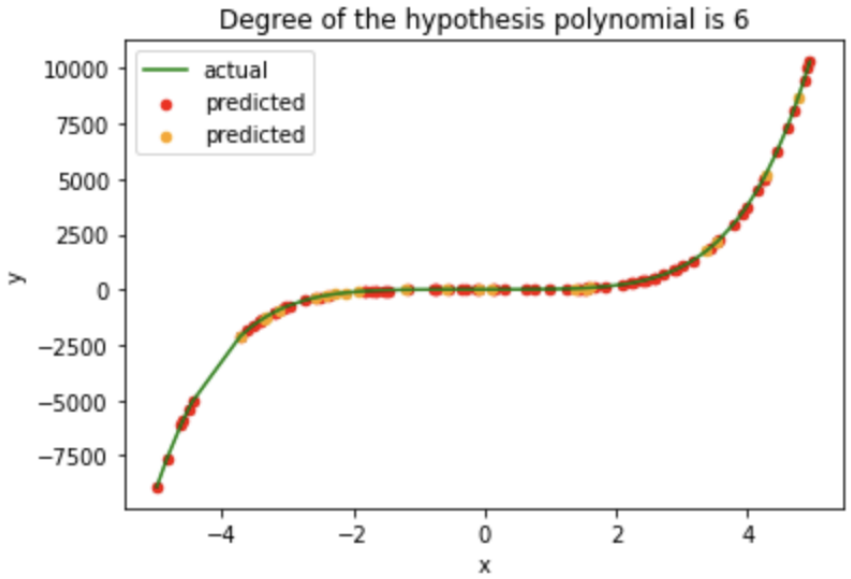
3.5 Degree = 5

```
The polynomial is:
477.30736382867224*X^0 15.318031771157173*X^1 88.72842195180044*X^2 1638.5222893743444*X^3 806.9925985736174*X^4 1759
5.90969789144*X^5

The Training root mean square error is 0.6666446964291595
The Testing root mean square error is 0.8934244483578119
noise variance: 0.7982072449234604
```



3.6 Degree = 6

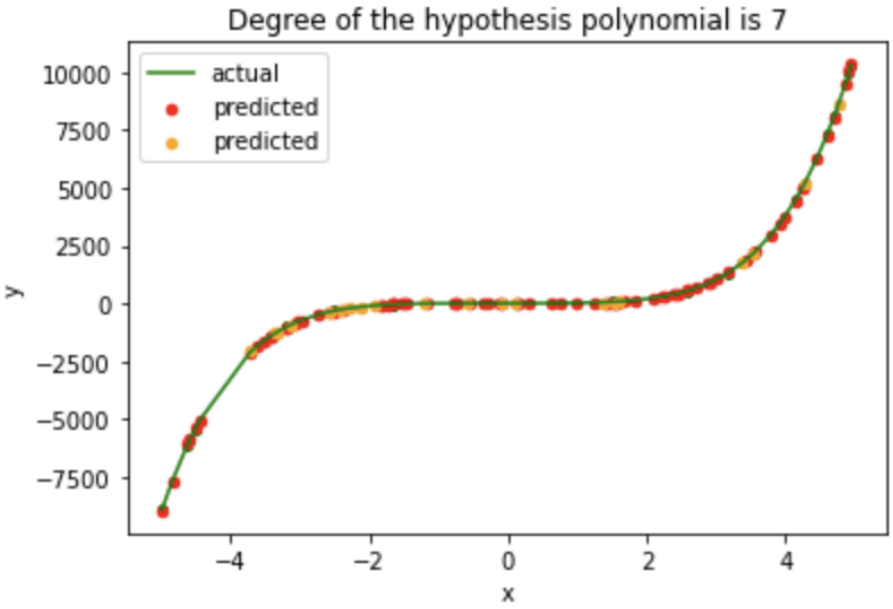
```
The polynomial is:
477.3073709335358*X^0 15.317630901172407*X^1 88.69838408886007*X^2 1638.5242610329558*X^3 807.0838224155298*X^4 1759
5.907719141404*X^5 -0.0665090607884937*X^6

The Training root mean square error is 0.6666418983654495
The Testing root mean square error is 0.8933377697584041
noise variance: 0.7980523708769194
```



3.7 Degree = 7

As it was taking a lot of iterations to reach the minimum, learning rate was increased to 1.

```
The polynomial is:
477.3069554137498*X^0 15.181534153410357*X^1 88.6624507112519*X^2 1639.7137947572817*X^3 807.2117886615982*X^4 17593.
321481190313*X^5 -0.16542185593698205*X^6 1.5739385475462837*X^7

The Training root mean square error is 0.6666349723500002
The Testing root mean square error is 0.891046505166872
noise variance: 0.7939638743700964
```



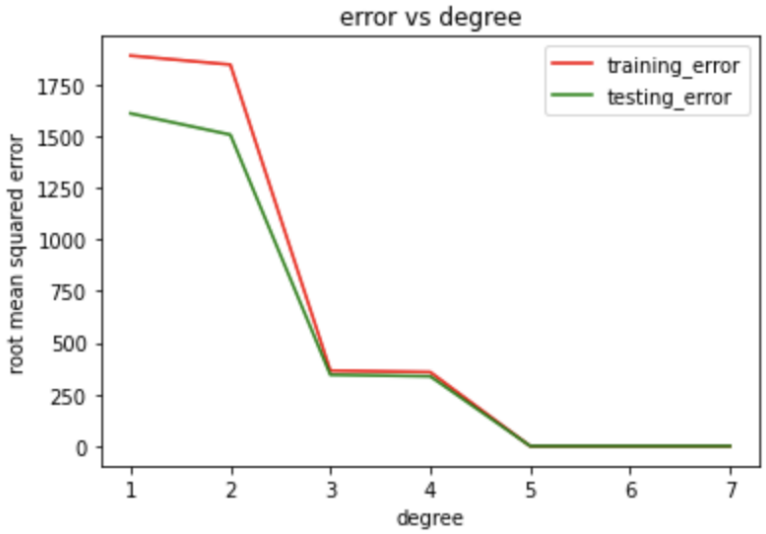
We can see that degree 5 has significantly low training and testing error but as we increase the degree further the model starts fitting the noise and as noise thus the training error decreases but there isn't any much decrease in testing error.

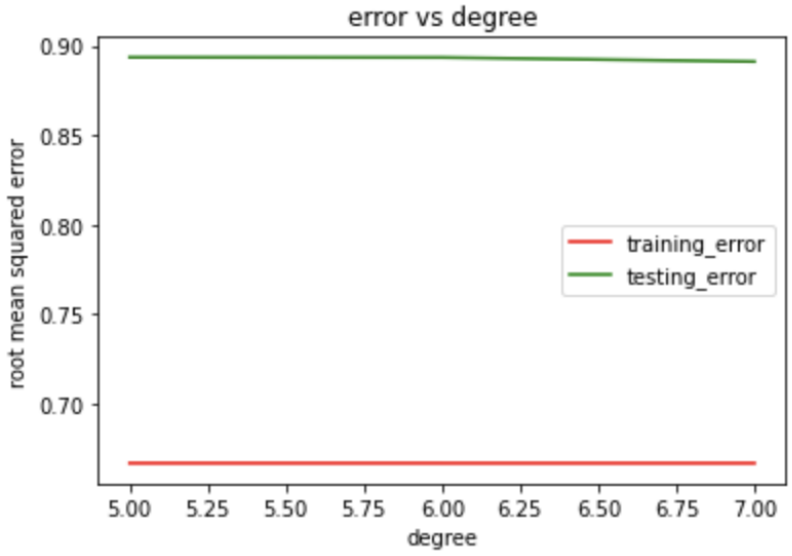
We can see for degree 1,2,3,4 the testing error is lower than training error which implies underfitting and from degree 5 onwards the testing error is more than training error which implies overfitting. We can say that degree 5 and 6 have the best fit as they have the least difference between the testing and training error.

As we increase degree beyond a certain limit ideally training error should decrease(which can be seen from our results) and testing error should increase(but due to the tradeoff between poor generalisation and decrease in testing error due to decrease in training error, the latter one dominates and there is a decrease in testing error also for degree 7)

3.8 Visualisation

degree	training error	testing error
1	1889.26	1609.96
2	1845.9	1506.98
3	365.746	345.77
4	358.816	338.227
5	0.666645	0.893424
6	0.666642	0.893338
7	0.666635	0.891046





4. Regularisation

In this experiment the following parameters were used:

- Mean Squared error
- Number of training points = 80
- Number of testing points = 20

As training error is less than testing error in degree 5 and degree 6 we will use regularisation to make them closer by reducing the testing error and increasing the training error using regularisation.

4.1 Lambda = 1e - 06 | degree 5

```
The polynomial is:
477.3073332815871*X^0 15.198870896998395*X^1 88.72766699130959*X^2 1639.0659051066543*X^3 806.9931941133793*X^4 1759
5.437809088162*X^5

The Training root mean square error is 0.6667142760405544
The Testing root mean square error is 0.8892710199328472
```

4.2 Lambda = 1e - 05 | degree 5

```
The polynomial is:
477.30705773459476*X^0 14.127272169213262*X^1 88.72088168617131*X^2 1643.9547980585853*X^3 806.9985483605219*X^4 1759
1.19387053902*X^5

The Training root mean square error is 0.6735511037090044
The Testing root mean square error is 0.8553072864101724
```

4.3 Lamda = 0.0001 | degree 5

```
The polynomial is:
477.30424103016566*X^0 3.494729426941106*X^1 88.65394635983839*X^2 1692.485174863609*X^3 807.0515394819445*X^4 17549.
05526018806*X^5

The Training root mean square error is 1.1653656259428173
The Testing root mean square error is 0.9271226157385157
```

4.4 Lambda = 0.0001 | degree 6

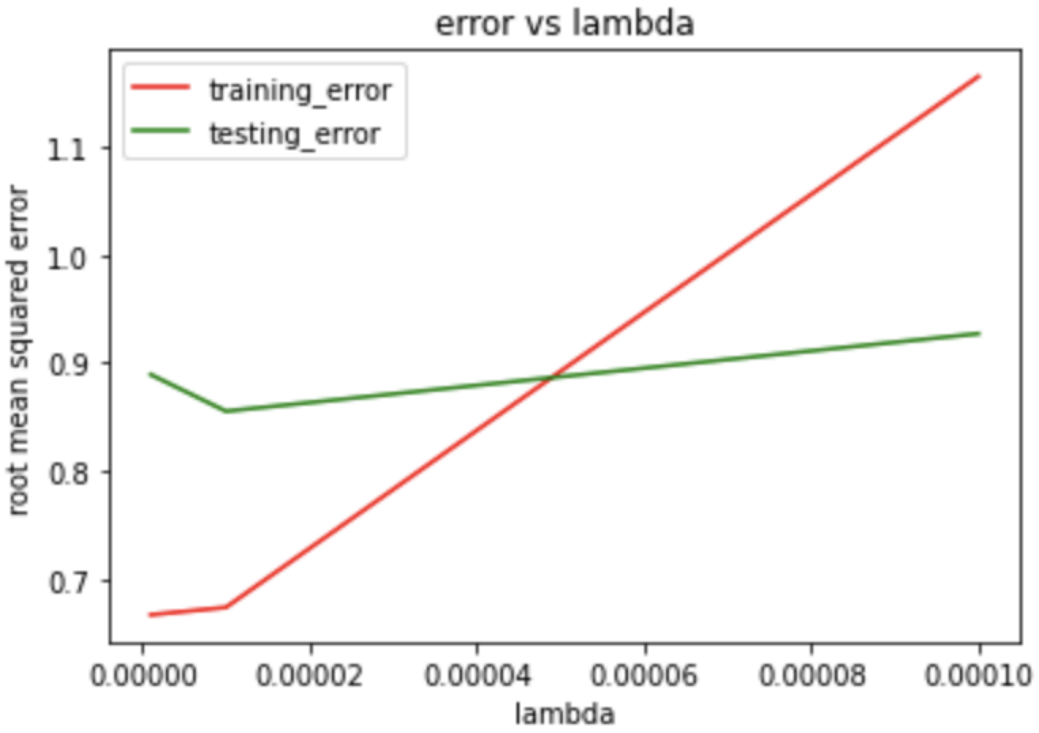
```
The polynomial is:
477.30417631478645*X^0 2.829989294733537*X^1 94.59406452869617*X^2 1695.2698995521341*X^3 789.4443761502359*X^4 1754
6.79495514115*X^5 12.655630807469533*X^6

The Training root mean square error is 1.1911514685959905
The Testing root mean square error is 0.9794941634872961
```

4.5 Lamda = 1e -05 | degree 6

```
The polynomial is:
477.3070590364858*X^0 14.059027978290342*X^1 89.29695888001666*X^2 1644.2409614401386*X^3 805.2936345965404*X^4 1759
0.96122764009*X^5 1.2243235708493092*X^6

The Training root mean square error is 0.6740814802122186
The Testing root mean square error is 0.8546638011969487
```

4.6 Visualisation and observation

degree	lambda	training error	testing error
5	1e-06	0.666714	0.889271
5	1e-05	0.673551	0.855307
5	0.0001	1.16537	0.927122
6	0.0001	1.19115	0.979494
6	1e-05	0.674081	0.854664



From the above graph and data we can see that as lambda increases testing error decreases and training error increases and they both come closer to each other but after a certain lambda training and testing error both increase as by theory of generalisation and **underfitting** occurs.

5. Number of data points

5.1 16 Training and 4 Testing

```
parameters used

degree : 6
error Function : <function grad_mean_squared_error at 0x7f9e582704c0>
Lamda for L2 regularisation : 0
Number of training points: 16
Number of testing points: 4


The polynomial is:
477.3455027362314*X^0 15.434674780437508*X^1 90.25870771471597*X^2 1635.498557899307*X^3 803.6294578556448*X^4 17599.
31939994129*X^5 2.6861029048973264*X^6

The Training root mean square error is 0.5409682022976826
The Testing root mean square error is 1.3792691275797873
noise variance: 1.9023833262947072
```

5.2 32 Training and 8 Testing

```
parameters used

degree : 6
error Function : <function grad_mean_squared_error at 0x7f85be094670>
Lamda for L2 regularisation : 0
Number of training points: 32
Number of testing points: 8


The polynomial is:
477.3683526213569*X^0 14.288846582629526*X^1 93.3227848607182*X^2 1641.1654417588325*X^3 793.8197474816672*X^4 17594.
53795224707*X^5 9.324347336945076*X^6

The Training root mean square error is 0.623927375213056
The Testing root mean square error is 0.9941583643913616
noise variance: 0.9883508534893073
```

5.3 64 Training and 16 Testing

```
parameters used

degree : 6
error Function : <function grad_mean_squared_error at 0x7f96481958b0>
Lamda for L2 regularisation : 0
Number of training points: 64
Number of testing points: 16


The polynomial is:
477.30494249971895*X^0 15.261666170573182*X^1 89.69723621276123*X^2 1640.6750620432947*X^3 804.3175122410801*X^4 1759
3.340802523282*X^5 1.6622569882206168*X^6

The Training root mean square error is 0.6223358705565716
The Testing root mean square error is 0.8787069774562909
noise variance: 0.7721259522303705
```

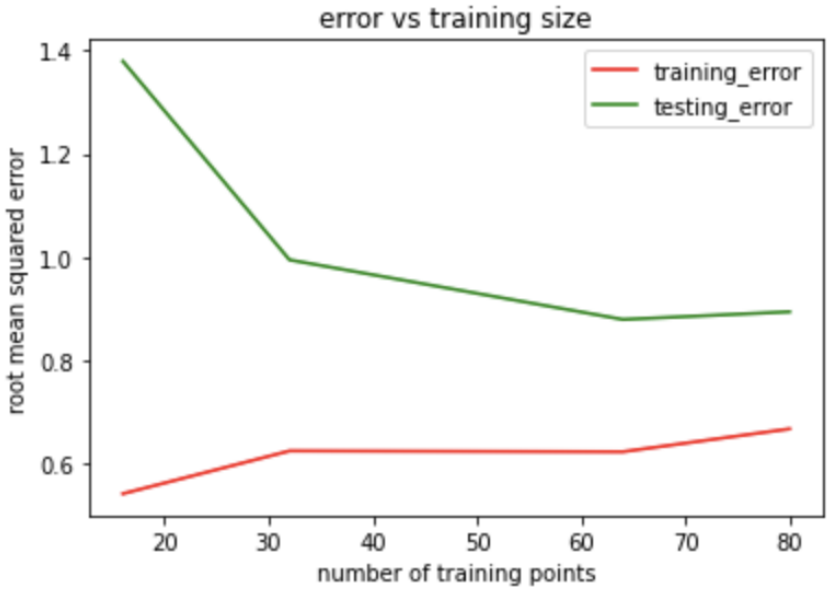
5.4 80 Training and 20 Testing

```
parameters used

degree : 6
error Function : <function grad_mean_squared_error at 0x7ffe0f1978b0>
Lamda for L2 regularisation : 0
Number of training points: 80
Number of testing points: 20


The polynomial is:
477.307364247096 76*X^0 15.323907354618617*X^1 88.67667216436328*X^2 1638.4976686642608*X^3 807.1459455252095*X^4 1759
5.929690619447*X^5 -0.11020288810327672*X^6

The Training root mean square error is 0.6666412897895969
The Testing root mean square error is 0.8935324383541428
noise variance: 0.7984002183910999
```

5.5 Visualisation and observations

From the above results it can be seen that when there are less points in the training data, the model tends to **overfit** the points and decreases the training error but due to theory of generalisation the difference between testing and training error increases and thus the testing error increases.

training size	testing size	training error	testing error
16	4	0.540968	1.37927
32	8	0.623927	0.994158
64	16	0.622336	0.878706
80	20	0.666641	0.893532



6. Conclusion

From the above results we get that Huber error function performed better(testing accuracy) than mean squared error when there was no regularisation as Huber error function was robust to the outliers but when we tried to introduce regularisation to reduce the error we observed that adding regularisation reduces the fitting to the noise and thus mean squared error performed better than Huber function.

In the final model we have used:

- **degree = 6**
- **Mean squared error**
- **Lambda = 0.00001**
- **Training points = 80**
- **Testing points = 20**

Huber oscillating.

```
parameters used

degree : 6
error Function : <function grad_huber at 0x7f8925194e50>
Lamda for L2 regularisation : 1e-05
Number of training points: 80
Number of testing points: 20


The polynomial is:
477.30022645640867*X^0 14.989007213960212*X^1 88.67919202698008*X^2 1639.7154861788333*X^3 807.3893403402558*X^4 1759
4.952854834897*X^5 -0.35473286632923273*X^6

The Training root mean square error is 0.6672442561238824
The Testing root mean square error is 0.8833946435336589
noise variance: 0.7803860962239603
```

7. Final Result

Mean Squared error

```
The polynomial is:
477.3070590364858*X^0 14.059027978290342*X^1 89.29695888001666*X^2 1644.2409614401386*X^3 805.2936345965404*X^4 1759
0.96122764009*X^5 1.2243235708493092*X^6

The Training root mean square error is 0.6740814802122186
The Testing root mean square error is 0.8546638011969487
```

Noise variance = square of testing root mean squared error = 0.73045021307

Handwritten Digit Recognition using logistic regression

1. Introduction

Dataset containing 5000 training examples of handwritten digits(0-9) from https: //web.iitd.ac.in/~seshan/a1/handwritten_image_data.rar. Is used to train and test the logistic regression model.

One vs all multi-class classifiers are used.

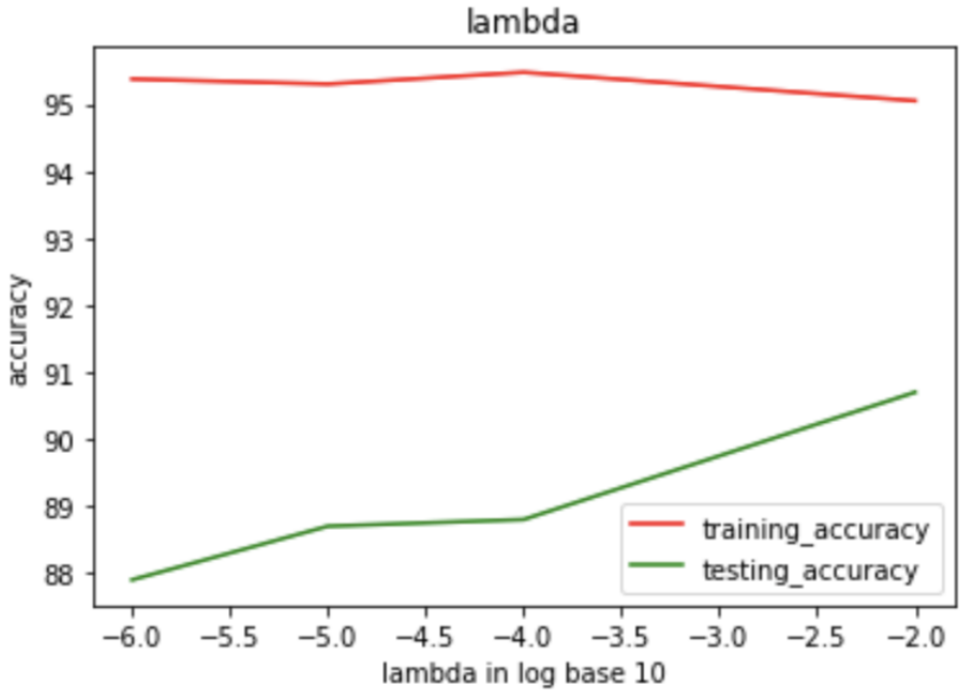
Dataset is split into 80:20 ratio for training and testing respectively.

Cross entropy cost function is used and sigmoid activation is used as instructed in the assignment. Gradient descent is used to minimise the cost function

2. Regularisation

For the experiment iterations = 50000, learning rate = 0.02(for a higher learning rate, cost function was not converging).

Lambda	0	0.000001	0.00001	0.0001	0.01
Train acc	95.25	95.375	95.3	95.475	95.05
Test acc	88.5	87.9	88.7	88.8	90.70



From the above graph and data we can see that as lambda increases testing_accuracy increases and training accuracy decreases and they both come closer to each other but after a certain lambda training and testing accuracy both decrease as by theory of generalisation and **underfitting** occurs.

3. Misclassified data

To observe how the model misclassified the data the arrays were converted to the actual image.
It was observed that the model learned the edges and curves and whenever it saw edges like in 7 in 2,3 it classified 2 as 7.
For classifying 5 it learned an edge on the top and curve in bottom and hence misclassified 3 as 5.

For classifying 8 it learned 2 loops but with a single loop of 9(with the bottom looking as a loop) it misclassified it as 8 and as it learned 8 as 2 loops when the image had the bottom loop not complete it classified the number as 1.

It also misclassified 4 as 9 as they looked similar in the image.

Some misclassified points:

Classified by our model as 7 although labelled as 2

Classified by our model as 7 although labelled as 3

Classified by our model as 5 although labelled as 3

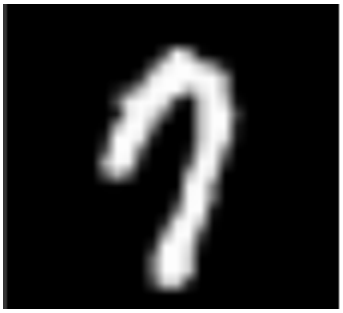
Classified by our model as 4 although labelled as 6

Classified by our model as 8 although labelled as 9

Classified by our model as 7 although labelled as 2

Classified by our model as 1 although labelled as 8

Classified by our model as 8 although labelled as 5



Classified by our model as 9 although labelled as 7

Classified by our model as 6 although labelled as 2

Classified by our model as 0 although labelled as 9

Classified by our model as 9 although labelled as 4