Name: Shahnawaz Khan
Entry Number: 2020CSB1123

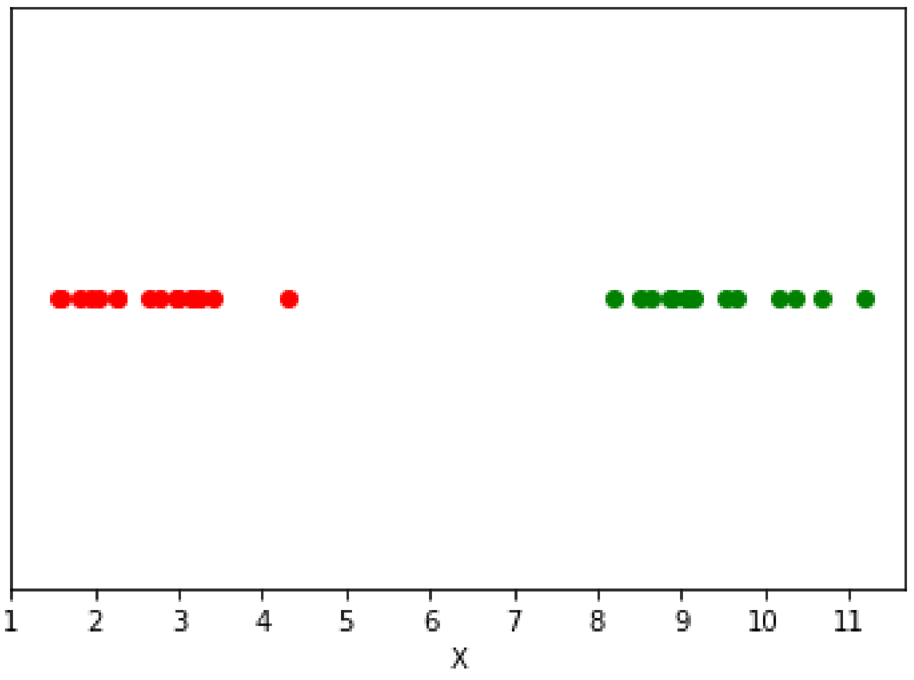
Machine Learning(CS503)

Assignment 1

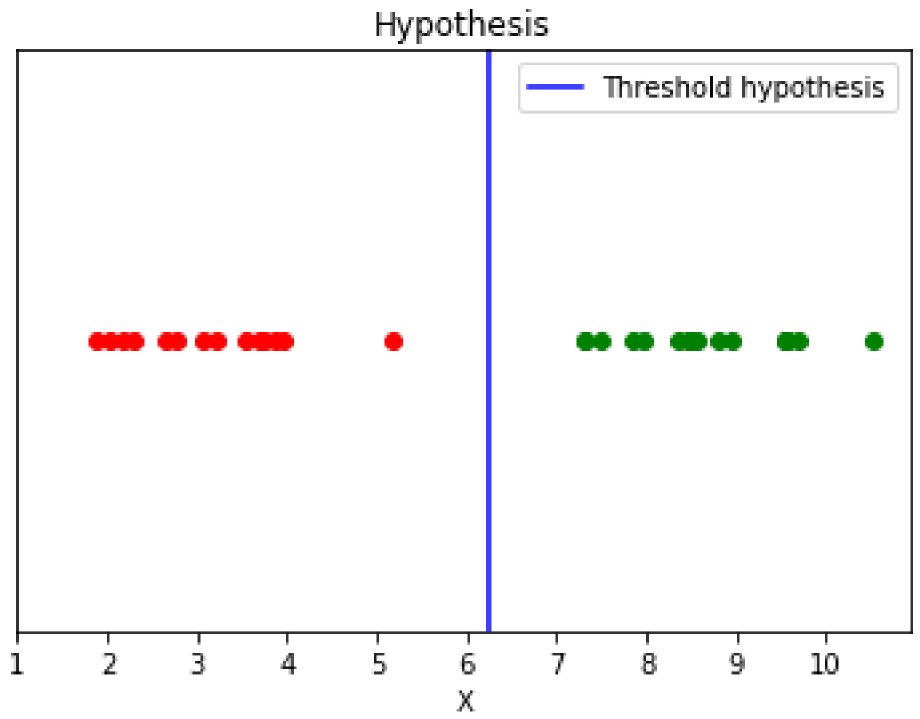
---

1. Task 1

   (a) The data-set generated is plotted as follows:



   (b) In this problem, we use Empirical Risk Minimisation algorithm to get a hypothesis. For that, we can make two sorted arrays denoting the x value of the data points whose y values are 0 and 1, respectively. For each x whose y value is 1, we binary search on the other array and calculated the absolute error on each value. The one which minimises the error is chosen by taking average of the next green point and the current red point to maintain uniformity somehow. Using this algorithm, we get the following results.



(c) As we know the the x values are normally distributed with $N_0(9,1)$ and $N_1(3,1)$ for $y_i = 0$ and $y_i = 1$ respectively, we can take the threshold θ^* and solve for the θ^* which minimizes the generalization error. The generalisation error L is given by the following equation:

$$L = \frac{1}{2}\int_{-\infty}^{\theta^*} \frac{1}{\sqrt{2\pi}} \exp\left((t-9)^2\right)\, dt + \frac{1}{2}\int_{\theta^*}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left((t-3)^2\right)\, dt$$

Now, we differentiate the above equation w.r.t θ^* and put it to zero to get optimal θ^*.

$$\frac{\mathrm{d}L}{\mathrm{d}\theta^*} = \frac{1}{2}\frac{1}{\sqrt{2\pi}} \exp\left((\theta^*-3)^2\right) - \frac{1}{2}\frac{1}{\sqrt{2\pi}} \exp\left((\theta^*-9)^2\right) = 0$$

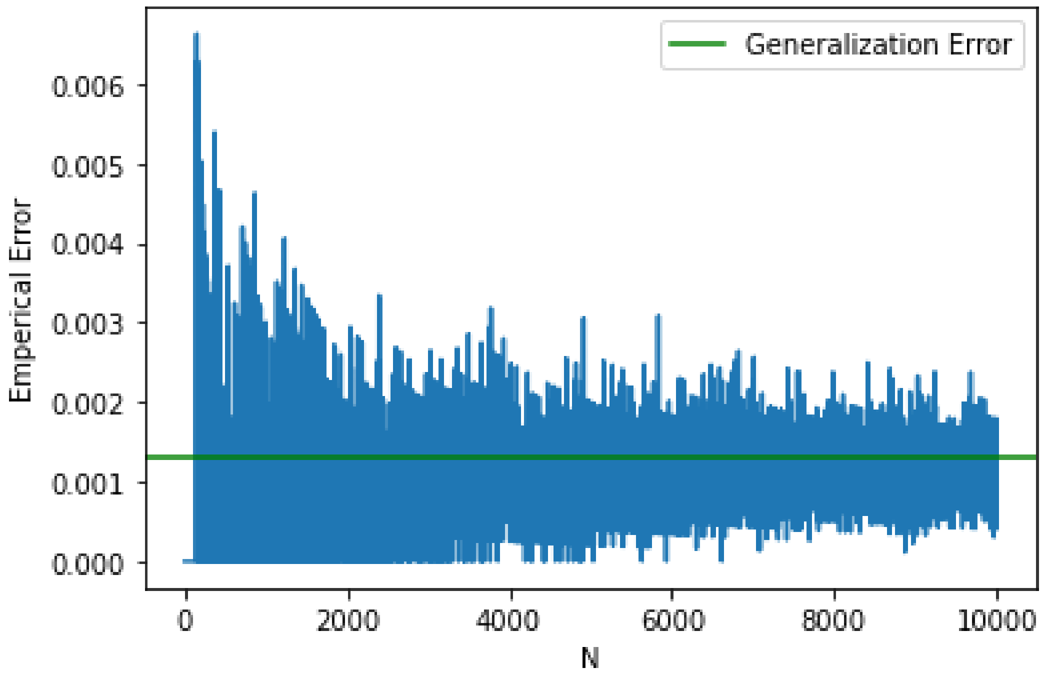
The equation simplifies to the following

$$(\theta^*-3)^2 = (\theta^*-9)^2$$

Hence, the value of θ^* is 6.

To get the generalisation error L, we can put θ^* as 6 in the above equation and get the value of L easily by looking at the z table. The value of L we get from this is **0.0013** approximately.

After plotting the empirical error vs N graph for 10000 values of N, we get the following graph.



Here, we can observe that, as N increases, the empirical error is coming close to the generalisation error.

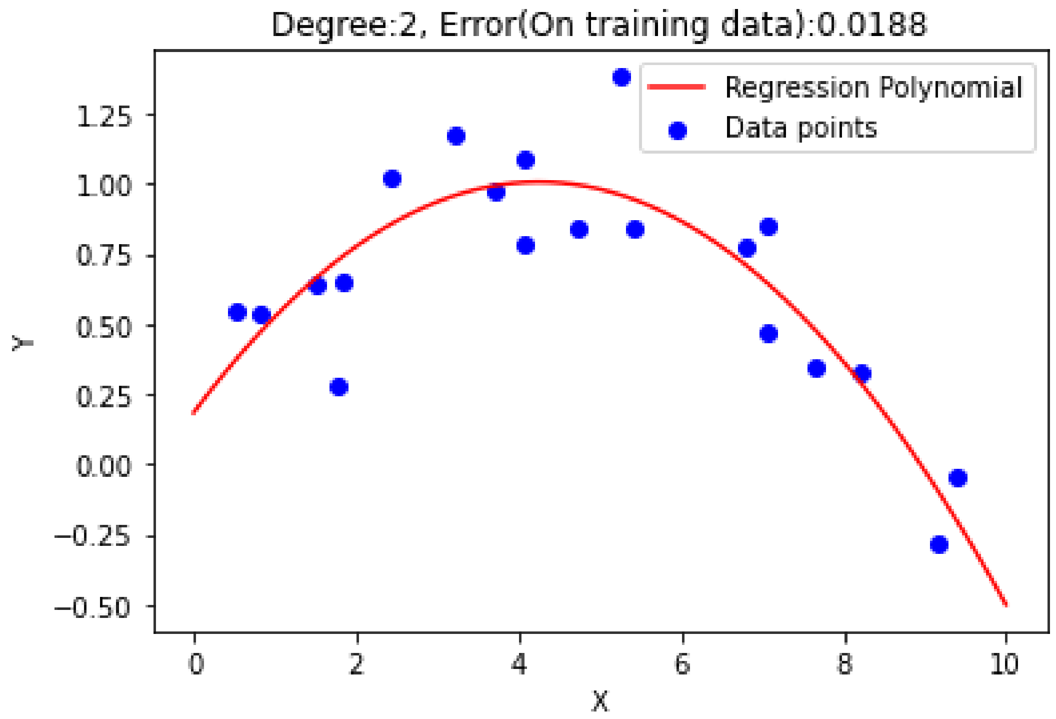
To get the minimum value of N to reach θ^* (Error being bounded by ϵ with probability atleast $1-\delta$), we can use the formula

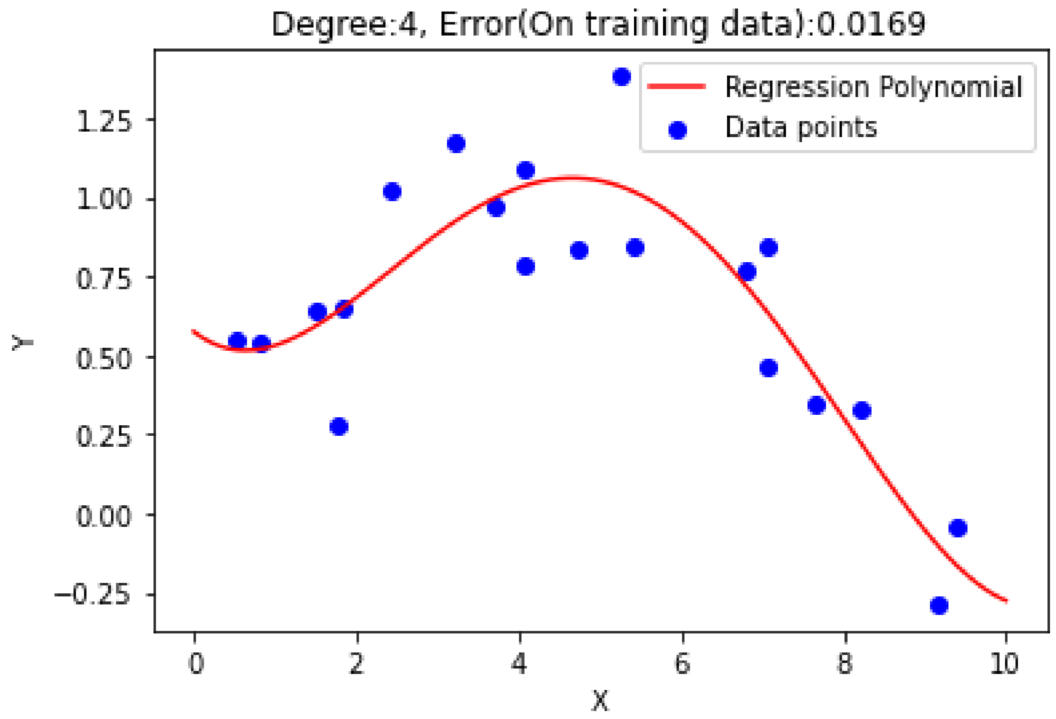
$$m(\epsilon, \delta) = \lceil \frac{log(2/\delta)}{\epsilon} \rceil$$

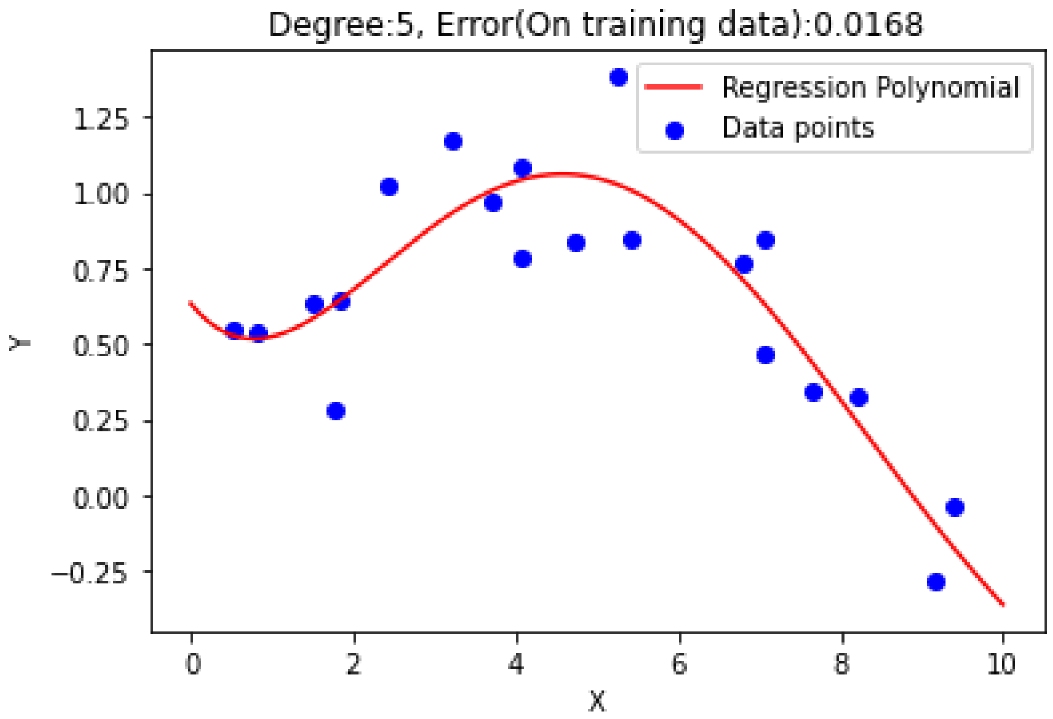
Putting $\epsilon = 0.0003$ and $\delta = 0.99$, we get, $N = 2344$

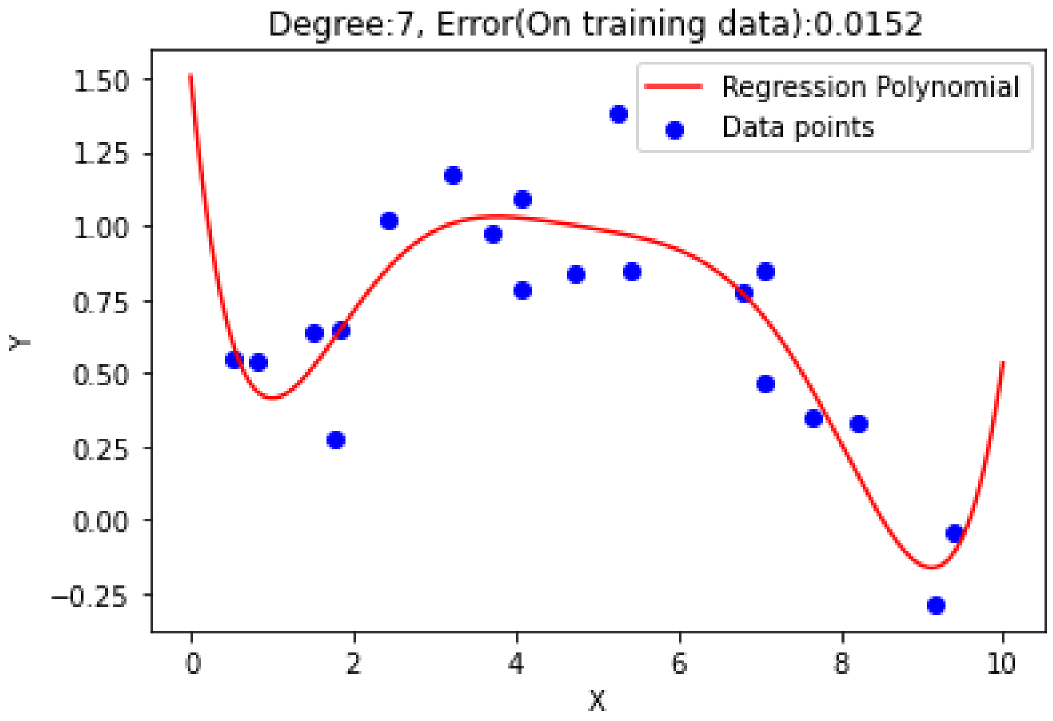
2. Task 2

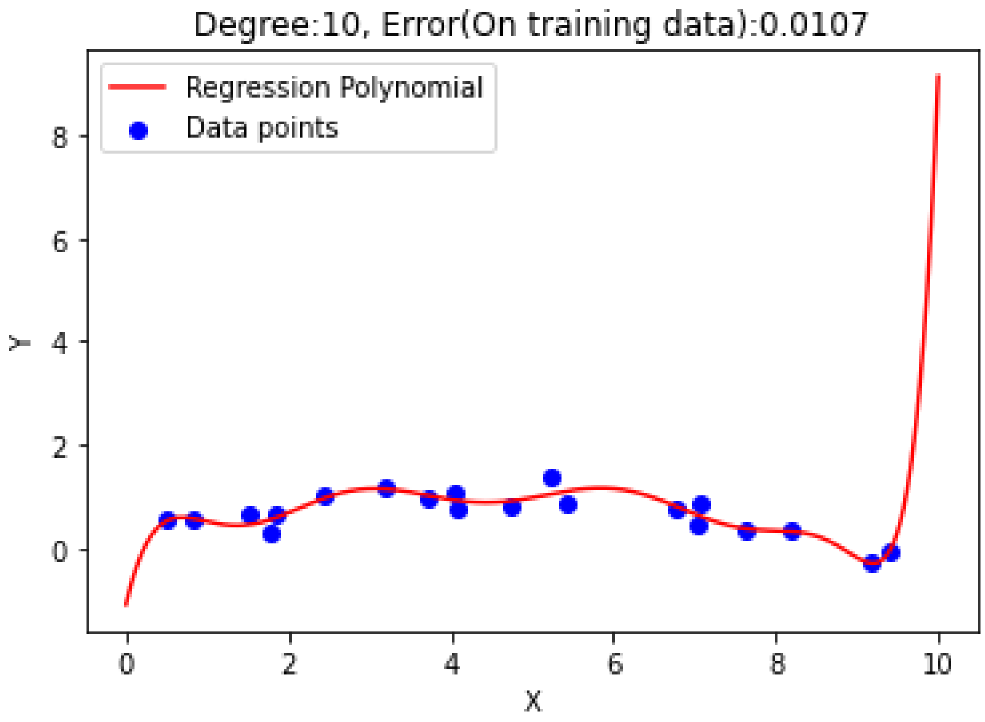
   (a) The data-set generated is plotted as follows along with the trained polynomial:


Degree:2, Error(On training data):0.0188
Regression Polynomial
Data points
Y
X
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0
2
4
6
8
10


Degree:4, Error(On training data):0.0169
Regression Polynomial
Data points
Y
X
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
2
4
6
8
10

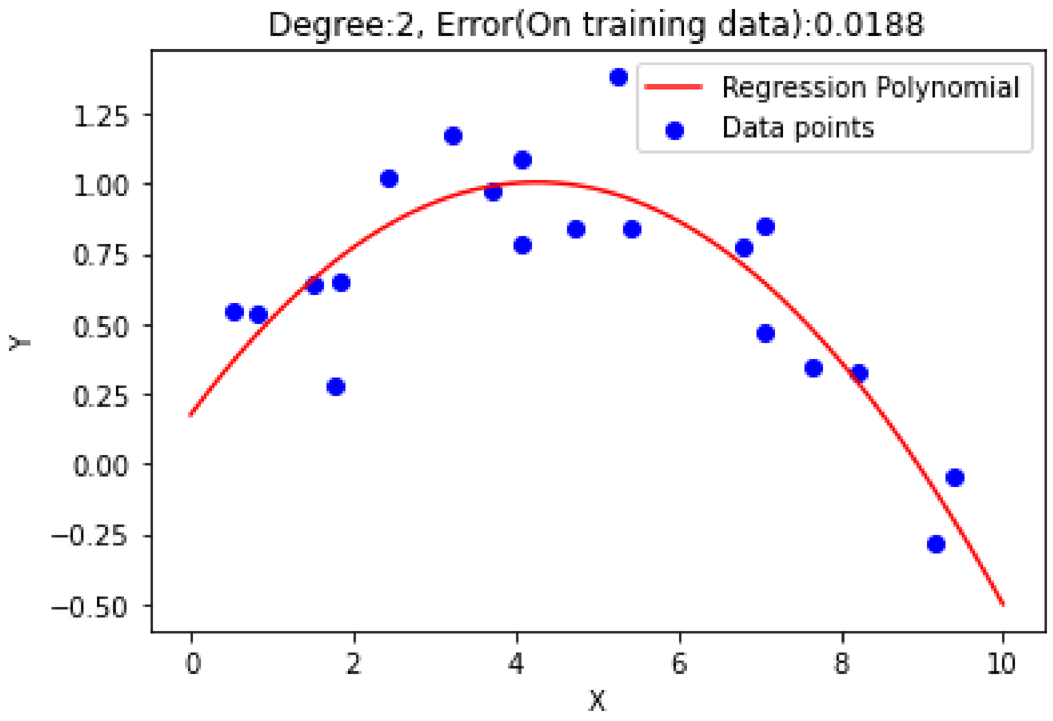

Degree:5, Error(On training data):0.0168
Regression Polynomial
Data points
Y
X
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
2
4
6
8
10

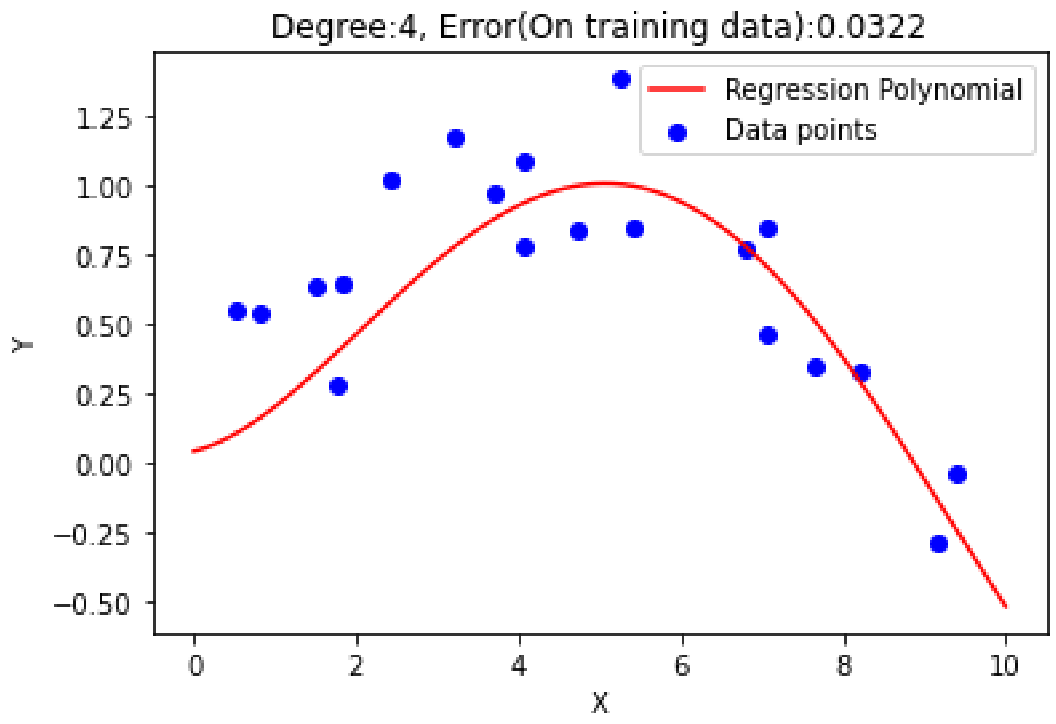

Degree:7, Error(On training data):0.0152
Regression Polynomial
Data points
Y
X
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
2
4
6
8
10

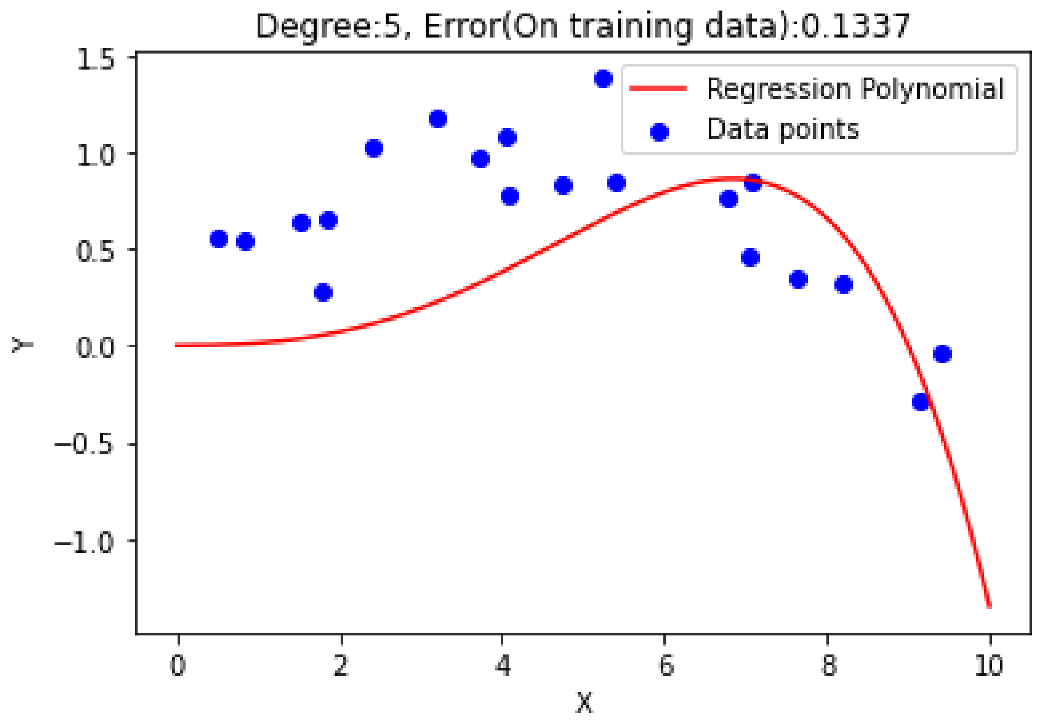


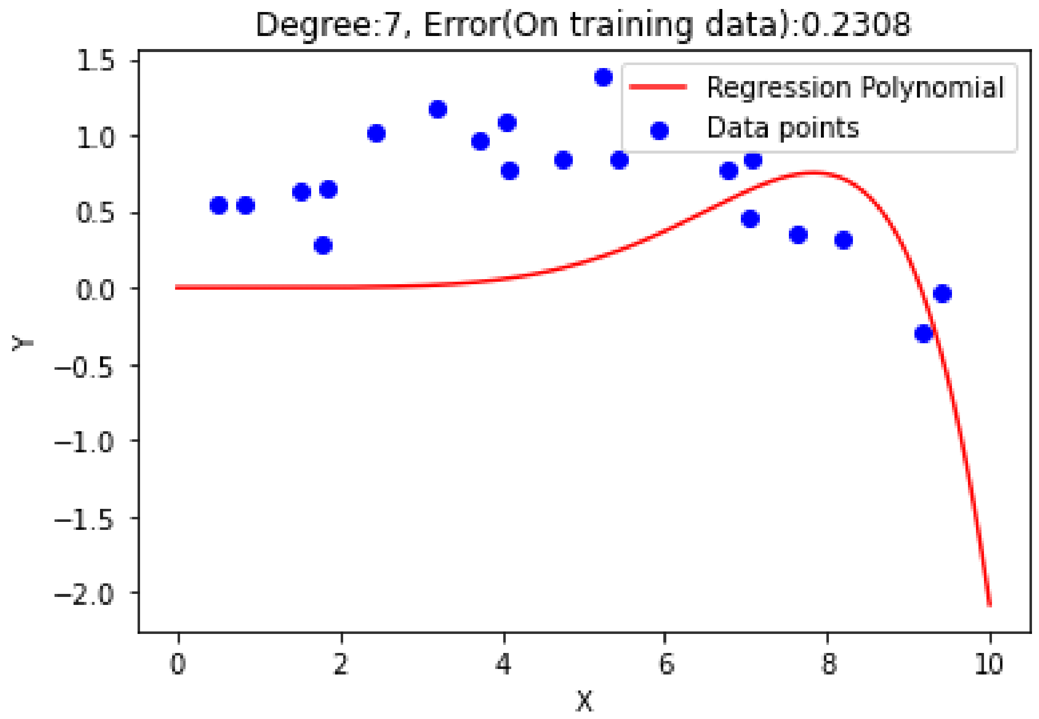
Here, we can observe that as degree of polynomial is increasing, the data tends to fit more on the trained curve. The error is also decreasing as we increase M. But as we increase M, we get higher value of coefficients and the curve becomes more sensitive to small changes and can result in over-fitting. In very few cases, the error is more for M=10 case because the coefficients are becoming large.

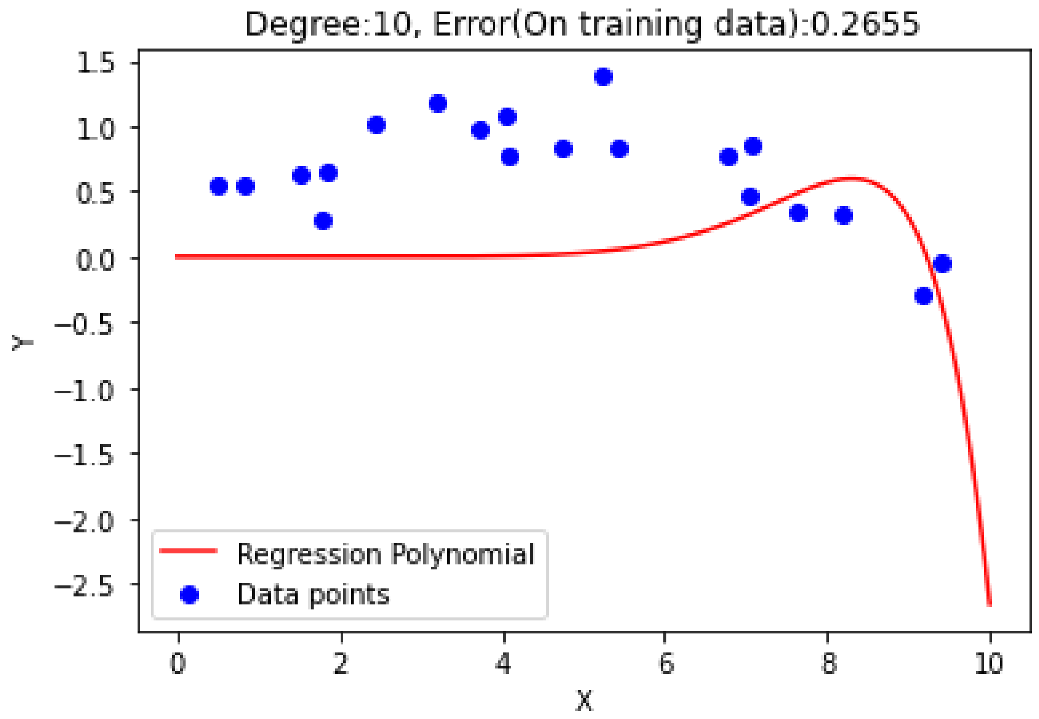
(b) Running the gradient descent for 1 million iterations, we get the following curves.




Degree:4, Error(On training data):0.0322
Regression Polynomial
Data points
Y
X
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0
2
4
6
8
10


Degree:5, Error(On training data):0.1337
Regression Polynomial
Data points
Y
X
1.5
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10



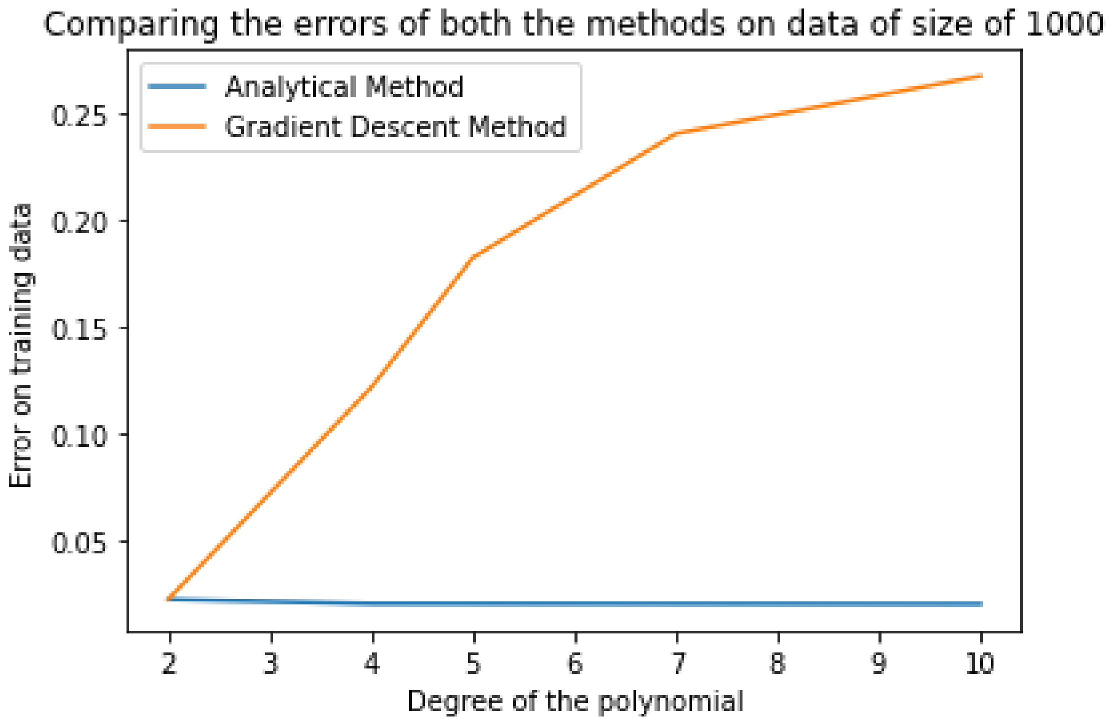


The error by this method is somewhat more than that of the analytical method. The reason is that the error is minimum while using analytical method(As per the mathematical equations). But in gradient descend method, the error never reaches optimum but it converges to that point which results in this error becoming more than analytical method. Also, the error is increasing with M because higher degree polynomial have very high error with small change in coefficient.

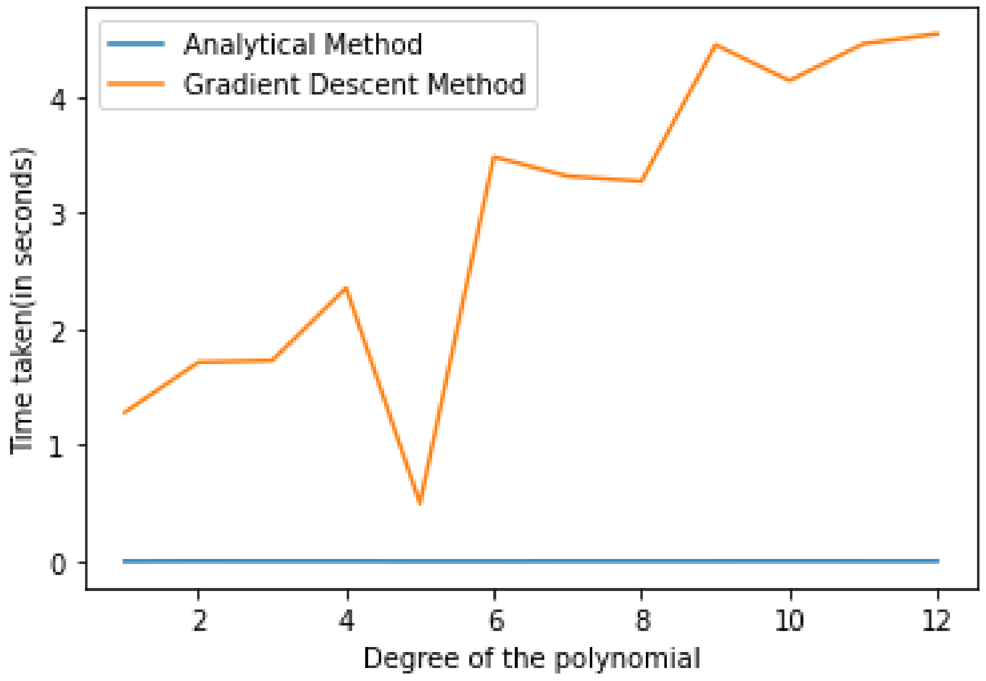
We use non-constant learning rate to solve this problem. Initially the learning rate is 1 and it becomes half everytime the error increases after taking a step. By this way we get the best results. Also, the toleranace helps in terminating the learning when we reach within the acceptable limit for the error(The limit also depends on the epochs. That is way default value of tolerance is taken as $1/epochs$).

More small we make the learning rate, the better accurately we can predict the coefficients. However, with smaller learning rate, we need large number of iterations to converge because the changes will be slow. Increasing the number of iterations can make the algorithm slow.

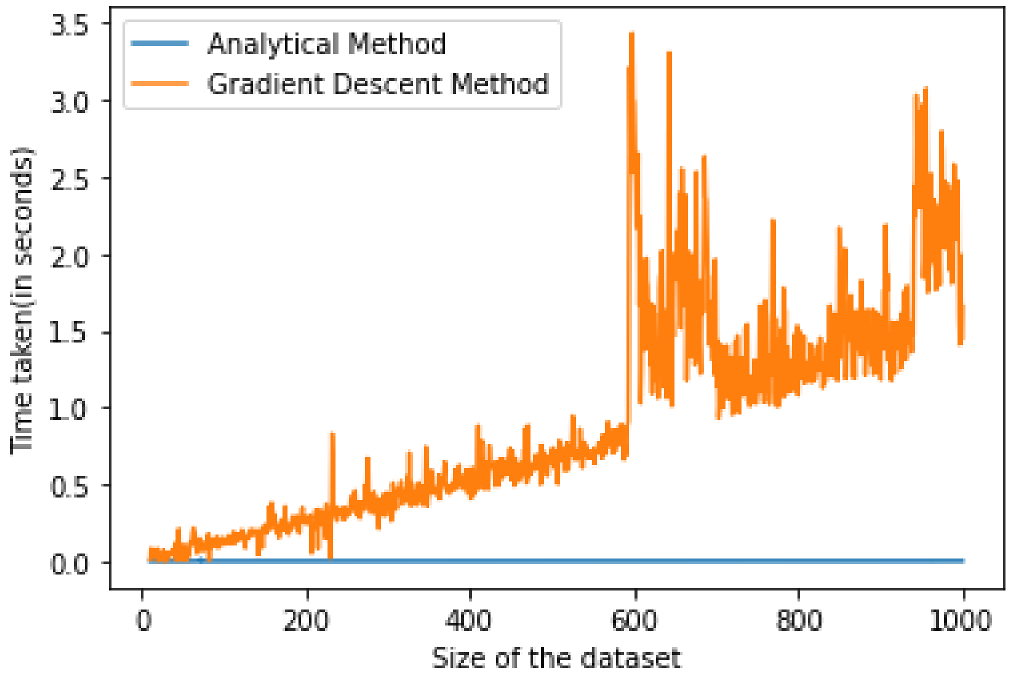
(c) Here, we plot the training data error obtained by both the methods on degrees 2,4,5,7,10. We can observe that errors are almost same on lower degrees but the difference becomes high on higher degrees polynomials. This is because gradient descend approaches optimal points which we can easily get by analytical method.



(d) We can observe that more time is being consumed when we increase the degree of polynomial. For higher degree polynomials, the gradient descent can work faster than the lower degree polynomial. But at higher degrees, the matrix X will become very huge and it may be difficult to calculate polynomial regression using any of the two methods due to high memory and time requirements. The time taken by the gradient descend is more than the time taken by the analytical method at lower values of M and N. On the fixed dataset of size 1000, following graph is obtained.

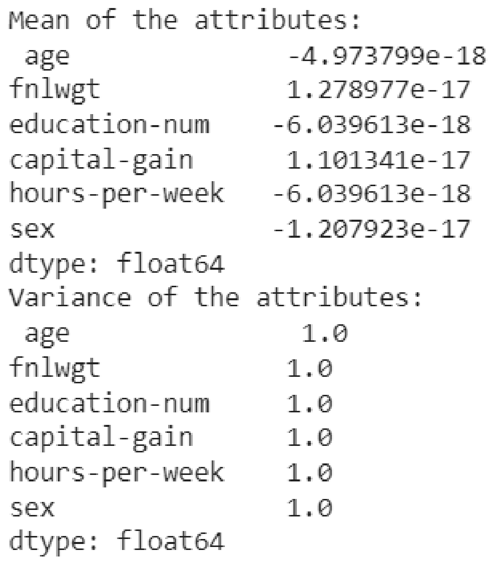


(e) On large dataset, the time taken by gradient descent method is more compared to that of analytical method. The value of M is fixed at 5 here. We tested till dataset of size of 1000 because this computation is slow. Here also, we can see that gradient descent is performing worse than the analytical method. This is because we tested for lower values of M and N. At higher values, gradient descent will perform better because it doesn't involve inverse and multiply operations on large matrices.



3. Task 3

(a) We gave the binary label 0 to *Females* and 1 to *Males*. Then after normalising the data, we got the following means and variances of the attributes. The means are close to zero and variances are 1 of the modified data-set.


```
Mean of the attributes:
 age              -4.973799e-18
fnlwgt             1.278977e-17
education-num    -6.039613e-18
capital-gain       1.101341e-17
hours-per-week   -6.039613e-18
sex              -1.207923e-17
dtype: float64
Variance of the attributes:
 age               1.0
fnlwgt            1.0
education-num     1.0
capital-gain      1.0
hours-per-week    1.0
sex               1.0
dtype: float64
```

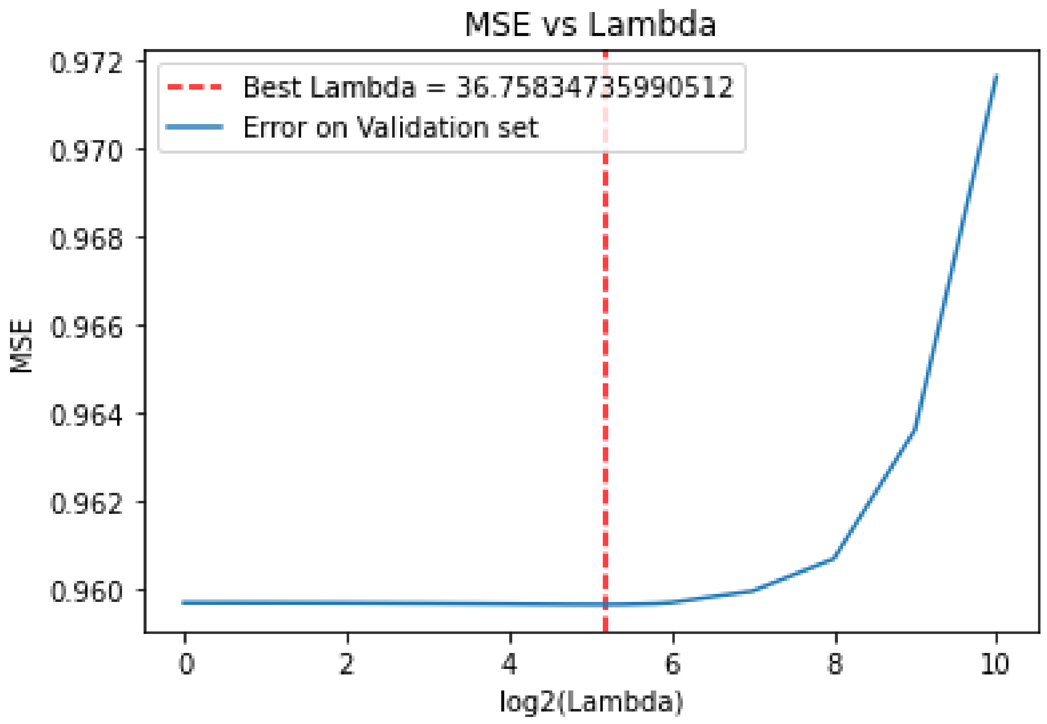
(b) We create datasets for K-folds cross validation using pandas sample method which randomly samples the data-set. The leftover(In our case there will be no leftovers) is given to the first fold.

(c) We generate the stratified K folds samples using the same sample method. We first partition the data-set into two parts one for each males and females. Then we select from them while ensuring we take samples in the ratio of there occurances in the actual data-set. All the leftover examples(due to leftover sample not divisible by K) is given to the first fold.

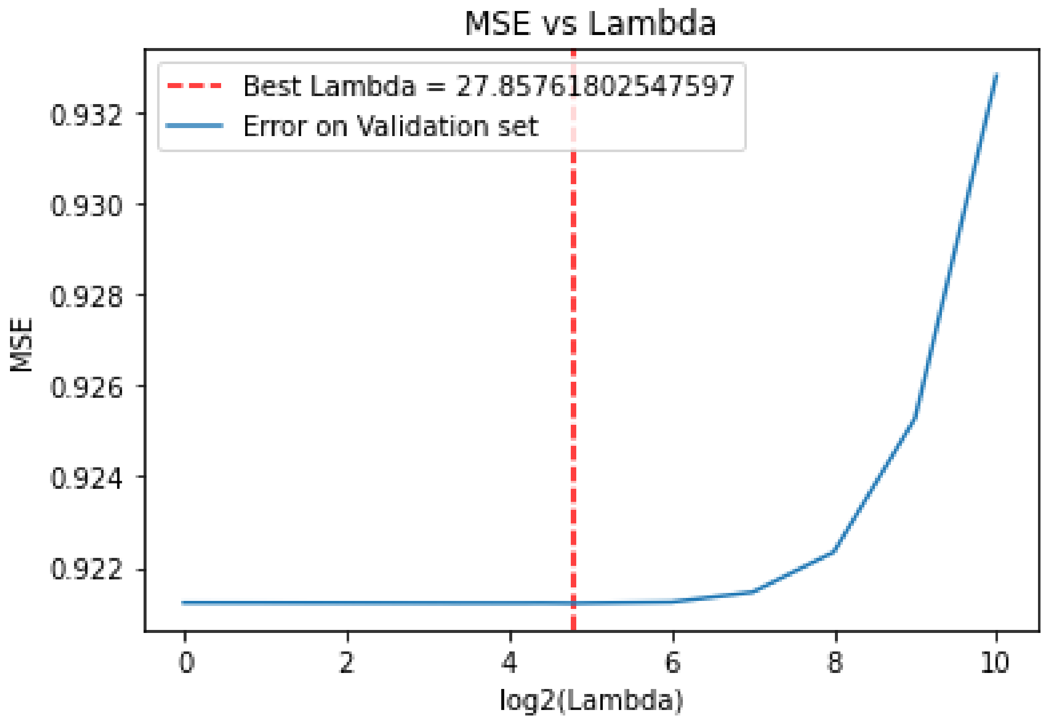
(d) We run the gradient descend at $epochs = 1000$ and $LearningRate$ as 0.00001. We use the formula for the gradient based on the error formula in LASSO regularization. This yields the weights with high accuracy.

(e) Here also, we use the same function as the algorithm for LASSO regression doesn't change with the change in the data-set.

(f) We generate 11 values at first level for log of λ starting from 0 to 10. When we find the optimal λ, let's say at $\log \lambda = x$, then at second level, we take 10 values of $\log \lambda$ from $x - 1$ to $x + 1$. Since the learning is slow, we cannot test on many values of λ. Splitting it into first and second level allows us to reach optimum with less computing time.
This way we can approximately find the optimal value of λ. In the K-Fold data-set, for each fold, we take $K-1$ folds as training set and the rest as validation set. We calculate MSE on this validation set and for each λ, we take the average of errors from validation set as the error for that particular λ. Doing this, we obtain the following graph, which gives us the approximate optimal value of λ.

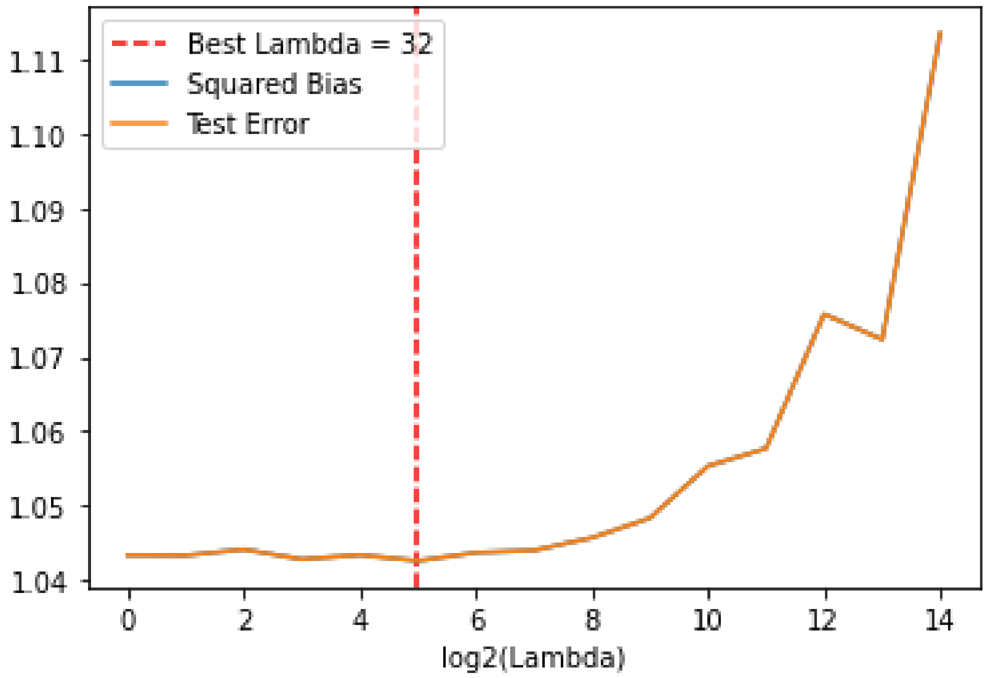


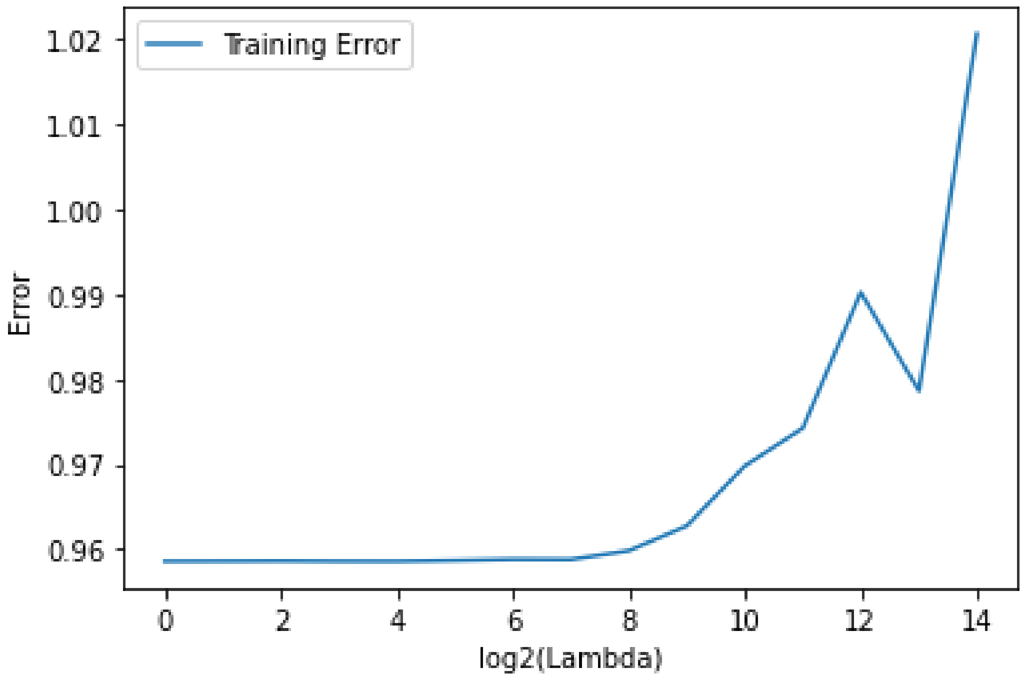
(g) Here also, we used the same technique as in 3.6. We obtain the following results.

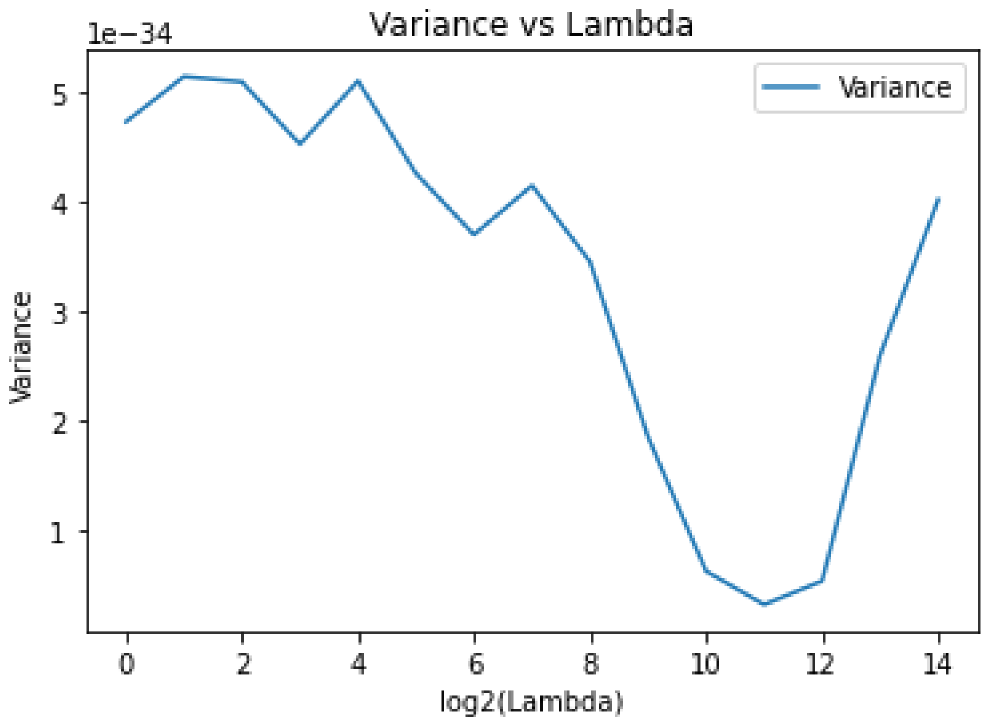


Here, the value of optimal λ is found to be 27.857... which is less than that found in the previous problem. However, this trend varies on different runs. We also observe that on same value of λ, we get lower error on stratisfied K fold compared to random dataset.

(h) In this problem, we take each possible $K-1$ folds as independent training data and calculate weights for each available lambdas. Then we calculate training error on this data-set and test error on the test set reserved initially as 20% of the complete data-set. Then we get the variance of the y values calculated by the weights and the actual y values in the training. We obtain the following plots.







Here the test error and bias are almost same. Hence, we can not distinguish their colors in the chart

We can see that at those where variance is increasing, the bias is decreasing and vice versa. Here also, we are getting the value of λ nearly same as we were getting in the previous problems.

The optimal value is obtained when there is a balance between the bias and the variance. In this particular case, the variance is very low (because datasets are generated using K fold validation). Because of this the test error is dominated by bias.