Foundations of Data Science
Assignment-3

Achyuta Krishna V	2018A7PS0165H
Jonathan Samuel J	2018AAPS0460H
Shubh Deep	2018A7PS0162H

Aim of the assignment :

To implement Polynomial Regression after adequate pre-processing of data using:

1. Gradient Descent
2. Stochastic Gradient Descent

Backed up by Lasso and Ridge Regularization

The Model

Our method involved developing 10 different models of degrees from 1 to 10. Each model of degree x contains all polynomial combinations of the features with degree less than or equal to x as the independent variables. In our case, the dataset contains 2 features: Age and BMI. A model of degree 2 for our dataset will have 5 independent variables: age, bmi, age*bmi, age^2, bmi^2 . Assuming age to be x_1, bmi to be x_2 and the target variable charge to be y, the resulting model would be:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

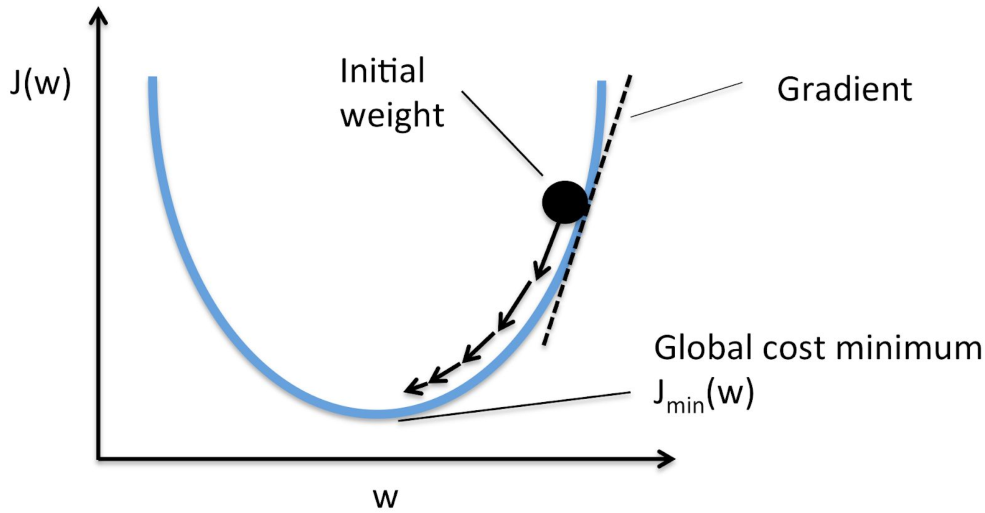
To generate the feature matrix for different degrees, we use the `PolynomialFeatures()` function available in the scikit-learn library in python. This helps us generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the degree specified by us.

To get our matured polynomial features, we split the dataset into a feature set and a target set and then pass the feature set to the `poly_features(X, deg)` method defined by us.

After getting our polynomial features and the target set, we use the `splitData()` method to divide the dataset into a training set and testing set. We then use the standardize() method to standardize our datasets using the min-max normalization method.

Gradient Descent Algorithm

Gradient descent is an iterative optimization algorithm used to find the values of parameters(coefficients of the features) to minimize a cost function. The algorithm arrives at an optimal value of the global minimum by taking steps proportional to the negative of the gradient at the current point.



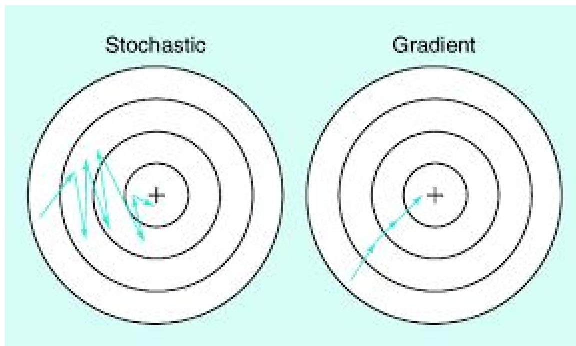
To implement this algorithm we defined a function, gradient_descent() which has a loop which runs for the number of iterations passed to the function. At each iteration of the loop, we first calculate the prediction by taking the dot product of our feature matrix and our current model. We then calculate the derivative of the cost function by using vectorized equations. Finally, we update our model(θ) by subtracting the product of the derivative and learning rate from the current model. After every 50th iteration, we calculate the training error as the Root Mean Square of all errors based on our current model and print its value.

To use the above function we define a separate function, gradient_descent_model() which obtains an optimal model by calling the gradient_descent() function. After obtaining the model, the testing and training errors are calculated as the root mean square of all errors and their values are printed.

To obtain models of degrees from 1 to 10 we have another loop of 10 iterations. At the i^{th} iteration of the loop, we calculate the matured polynomial feature matrix for a polynomial of degree i+1. We then split the datasets into training and testing sets. We then standardize both our training and testing sets and finally call the gradient_descent_model() function by passing the arguments at the end of the loop.

Stochastic Gradient Descent Algorithm

Stochastic Gradient Descent is identical to gradient descent except here we select only one random data point in each iteration and use it to compute the gradient to reach the optimal value or the global minimum. This is done in order to decrease the computation power and time needed to train models with huge data sets.



To implement this algorithm we defined a function, stochastic_gradient_descent(). This function first shuffles the feature and target matrix so that the points we choose at each iteration of our algorithm are random. After this we have a loop similar to the gradient descent algorithm where we calculate the prediction value, and the derivative value based on just one random point and use these values to update our model.Similar to the previous method we output the RMSE of our model after every 50th iteration Also,we shuffle the dataset again once a full cycle of all points is completed and we can shuffle-start again.

To use the above function we define a separate function, stochastic_gradient_descent_model() which obtains an optimal model by calling the stochastic_gradient_descent() function. After obtaining the model, the testing and training errors are calculated as the root mean square of all errors and their values are printed.

To obtain models of degrees from 1 to 10 we have another loop of 10 iterations. At the i^{th} iteration of the loop, we calculate the matured polynomial feature matrix for a polynomial of degree i+1. We then split the datasets into training and testing sets. We then standardize both our training and testing sets and finally call the stochastic_gradient_descent_model() function by passing the arguments at the end of the loop.

Regularization

Ridge Regularization

In ridge regularization the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients. The cost function after modification is:

$$\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} w_j^2$$

When the new cost function is differentiated, the update step simplifies to

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

To implement ridge regularization, we modified the update features step of our original gradient descent functions. The updated functions gradient_descent_ridge() and stochastic_gradient_descent_ridge() have the same loop as their non-regularized counterparts. These functions now take a new argument, the regularization parameter(λ). The update step of our regression algorithm is now changed and becomes

```
for i in range(num_iters):
      pred = X.dot(theta)
      cost_der = (1/m)*np.dot(X.transpose(), (pred-Y))
      theta = theta*(1 - alpha*lam) - alpha*cost_der
```

The update step is changed similarly in stochastic_gradient_descent_ridge() function as well.

Lasso Regularization

In lasso regularization the cost function is altered by adding a penalty equivalent to the absolute value of the coefficients. The cost function after modification is:

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} |w_j|$$

To implement lasso regularization, two new functions were defined, gradient_descent_lasso() and stochastic_gradient_desscent_lasso(). Both these methods have two nested loops. The outer loop runs for the number of iterations passed to the function. The inner loop traverses every feature in our model. At each iteration it checks the value of the weight of that feature. If the feature is negative, the penalty term is subtracted in the update feature and if it’s positive, the penalty term in the derivative of the cost function is added. For the 0th weight in our feature, the penalty term is ignored as only the coefficients of our features and not the bias column are regularized.

```
if j == 0:
       theta[j] = theta[j] + alpha*2*np.sum(Y - Y_pred) / m
else:
       if theta[j] > 0 :
           theta[j] = theta[j] - alpha*( - ( 2 * (X[:, j]).dot(Y - Y_pred)) + lam ) / m
       else :
           theta[j] = theta[j] - alpha*( - ( 2 * ( X[:, j] ).dot(Y - Y_pred))- lam ) / m
```

Similarly,the stochastic_grdient_descent_lasso() method is changed as well where X and Y are defined such that they consider only one random data point at each iteration.

Errors

Without Regularization

The following tables show the minimum training and testing errors (RMSE) achieved by us when using the 2 algorithms without regularization.

Gradient Descent		
Degree	**Training Error**	**Testing Error**
1	12429.457840641535	13009.745375207587
2	12325.864056635874	12963.346663554885
3	12307.072736991606	12985.509385093808
4	12305.745261851276	13009.877859362405
5	12308.04928449048	13025.900462908057
6	12309.941944316204	13032.481026879872
7	12310.557808869966	13032.139190097356
8	12310.098041588537	13027.682732900297
9	12309.008291995753	13021.45148437517
10	12307.631589247554	13014.957506622986

Stochastic Gradient Descent		
Degree	**Training Error**	**Testing Error**
1	12424.089498909932	13000.588858942969
2	12321.98603471101	12952.472416613264
3	12316.837139548254	12983.926103318845
4	12338.862208514023	13028.725833254322
5	12374.68763394292	13077.191743158704
6	12419.521974251391	13128.991109631692
7	12471.848584276437	13184.079333396992
8	12531.56794074707	13243.765554455098
9	12596.567741680266	13308.210392166637
10	12666.533389739761	13377.09735625821

With Regularization

The following tables show the minimum training, validation and testing errors achieved by us when using the 2 algorithms with regularization

Ridge Regularization

Gradient Descent				
Degree	**Training Error**	**Validation Error**	**Testing Error**	**Best Penalty Value**
1	12341.512654617767	13325.183402542412	12817.613556825718	0.015458344010213065
2	12430.855752858219	13501.0840172 4584	13119.533248473397	0.2128584524604823
3	12302.734327266777	13362.108066731947	12753.0231916 1517	0.0037737464563282197
4	12304.856712142442	13387.827712223389	12775.155831841754	0.021558177635465414
5	12303.622746870366	13387.232259513823	12761.634733148523	0.00822361195021104
6	12313.502725070064	13436.249107957934	12821.605430197413	0.06136515359108852
7	12347.466221298715	13522.9083831 9875	12942.745544085416	0.1817256634972827
8	12298.0029868 2153	13357.583691176565	12762.425970921671	0.0012927404473989101
9	12359.7835201 4738	13550.706273961565	12966.260270350751	0.2035395614260218
10	12317.273592996282	13450.506146047519	12836.5045869 4072	0.07975314113789722

Stochastic Gradient Descent				
Degree	Training Error	Validation Error	Testing Error	Best Penalty Value
1	12467.436294075855	13409.551674104063	12911.383067940811	0.10737358283603637
2	12349.779158915333	13301.852443388418	12707.8437635 3357	0.10944041427958573
3	12356.535967181662	13320.679673088158	12554.257147935588	0.02561599160219985
4	12322.397274355992	13359.270740104781	12639.108376739112	0.1506386047554381
5	12354.63228007274	13388.186558532261	12571.53540914129	0.065092701573216
6	12374.954609197484	13372.002098464935	12524.386584169426	0.013666413776350583
7	12365.564588593215	13402.688486384352	12555.969190321834	0.047888043322065865
8	12363.03100444519	13410.281282705311	12564.859764356512	0.061262181930983095
9	12352.992946445487	13440.80459704115	12616.901466210045	0.12186754505308717
10	12383.917722251934	13329.995002021047	12488.547889582509	0.0014680602522822994

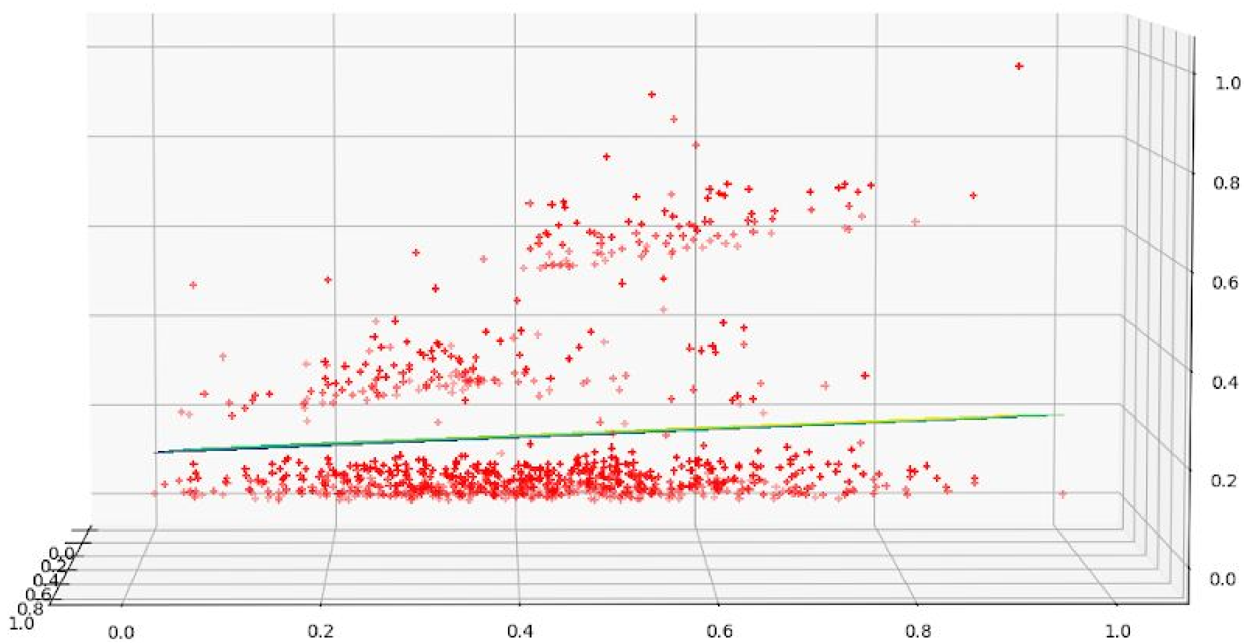
Lasso Regularization

Gradient Descent				
Degree	**Training Error**	**Validation Error**	**Testing Error**	**Best Penalty Value**
1	12369.808507548654	13332.47388531869	12820.274345727219	0.04695617846033018
2	12314.09631648974	13323.026009903024	12767.370773847393	0.05464566607764565
3	12306.895252028226	13353.54730698739	12766.118138072992	0.9710168923821021
4	12307.167464872877	13375.306329042738	12771.995569625724	0.8590197580990305
5	12308.186883439808	13387.365477408594	12775.385632643054	0.8072438873945543
6	12308.891983019053	13388.591288952397	12775.00334047284	0.9611952879918535
7	12309.467891024095	13385.938661699873	12777.442649404158	0.9718966006156333
8	12309.452136989443	13385.306752604953	12777.538245593525	0.9541433478649468
9	12308.888424469442	13386.27074970387	12774.487421938606	0.41902255471752015
10	12304.062524930405	13377.825218924589	12747.486855151015	0.13884979808376263

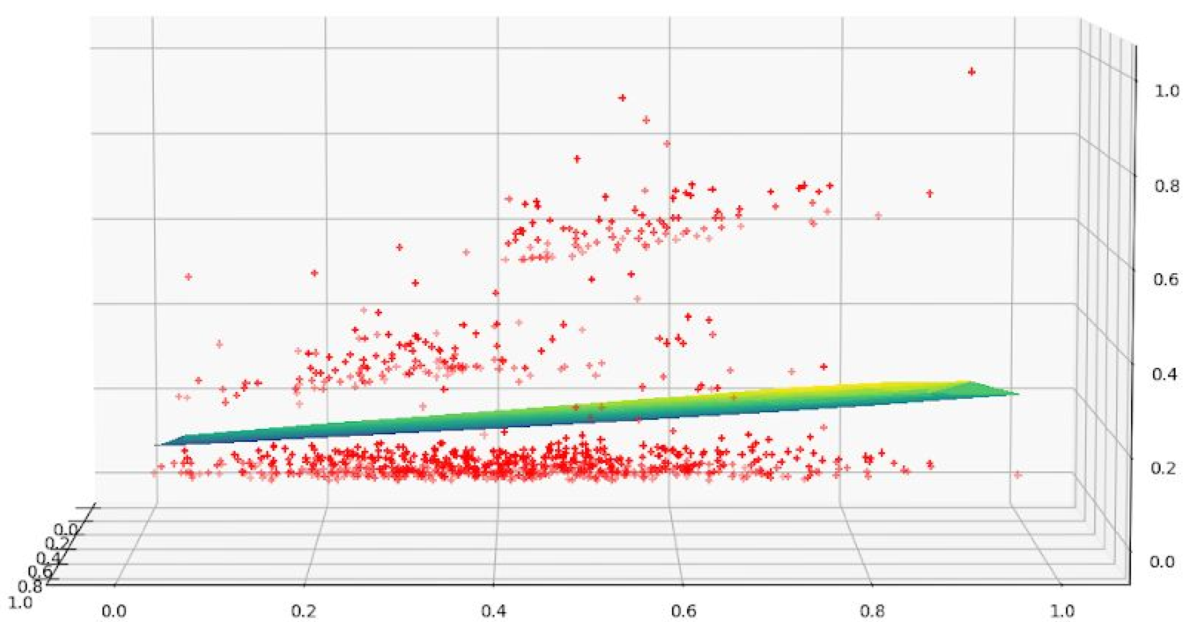
Stochastic Gradient Descent				
Degree	**Training Error**	**Validation Error**	**Testing Error**	**Best Penalty Value**
1	12595.15258636861	13573.162158126368	13149.893170498919	0.0027416359532291823
2	13151.25303474986	14148.69736255332	13796.024920018464	0.791582101392465
3	12444.686219870684	13496.247154289207	13092.308119590063	0.022481754008752408
4	12316.139851333703	13400.296599522511	12840.61636511133	0.005268103546186675
5	13148.010094742678	14145.208557302421	13791.535738402617	0.9030226404724753
6	12833.64607055728	13883.56743813412	13584.300008697439	0.05216712137353685
7	12468.862489529207	13554.32561404493	13168.432596145849	0.023318054098146135
8	12350.702114187761	13473.41629763026	12973.604962760075	0.01032594792649022
9	12424.784831740863	13517.713205380946	13104.524573711855	0.019185463106915934
10	12336.473091855096	13470.611181954042	12932.114787255416	0.007116444193793203

Surface Plots

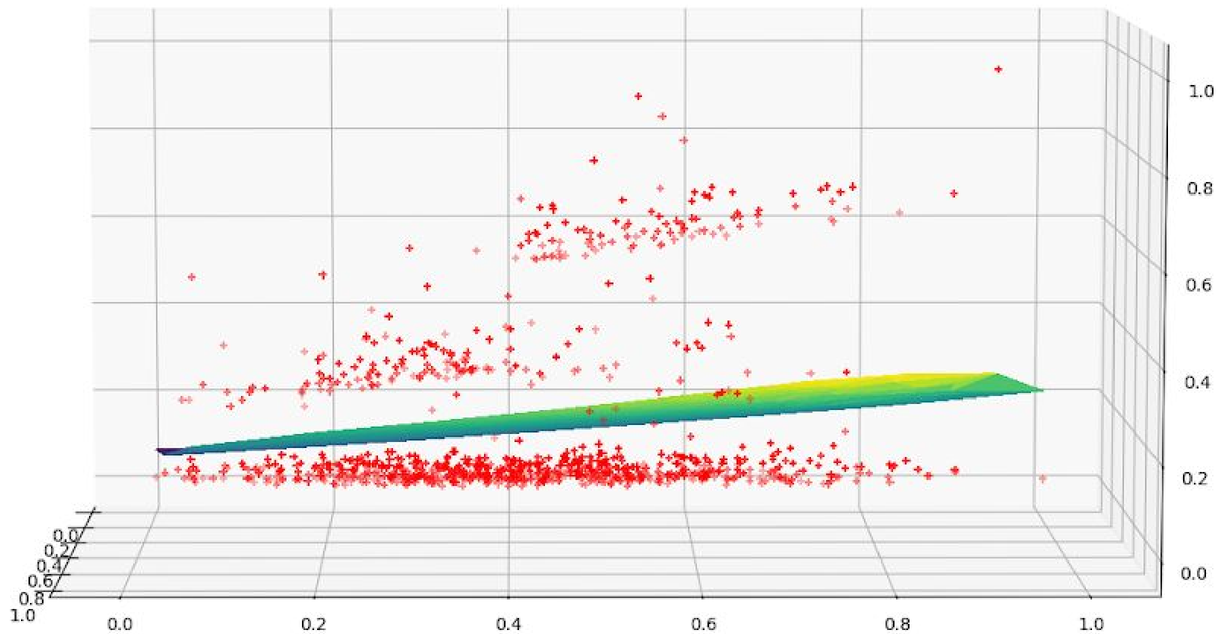
Degree : 1



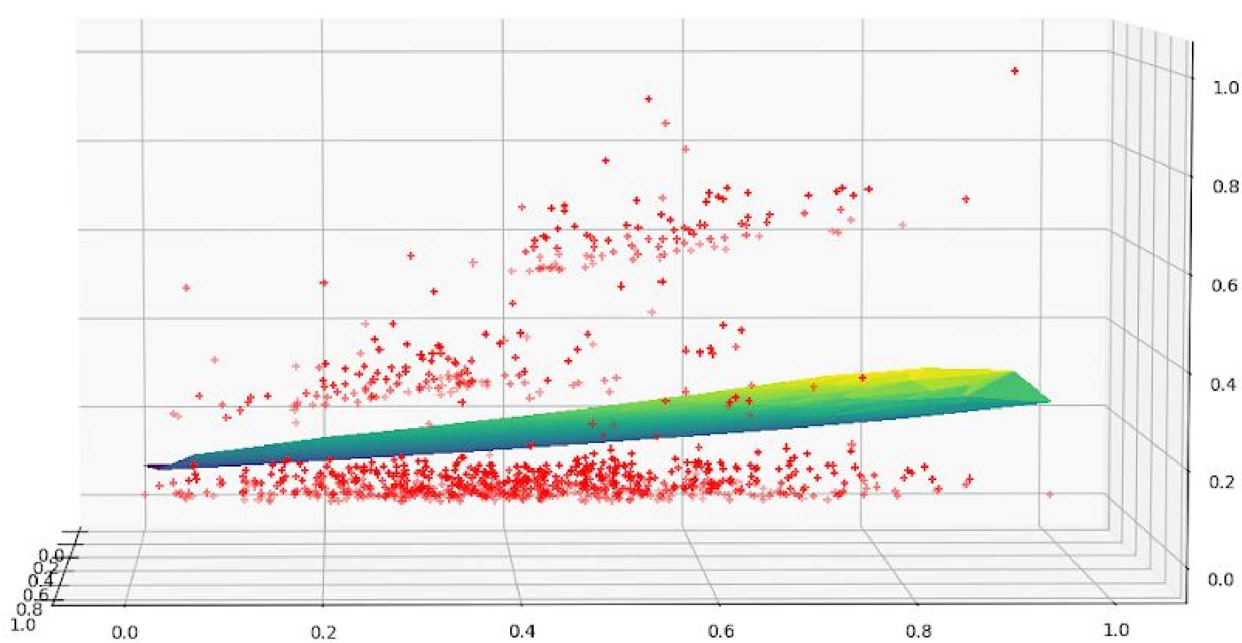
Degree : 2



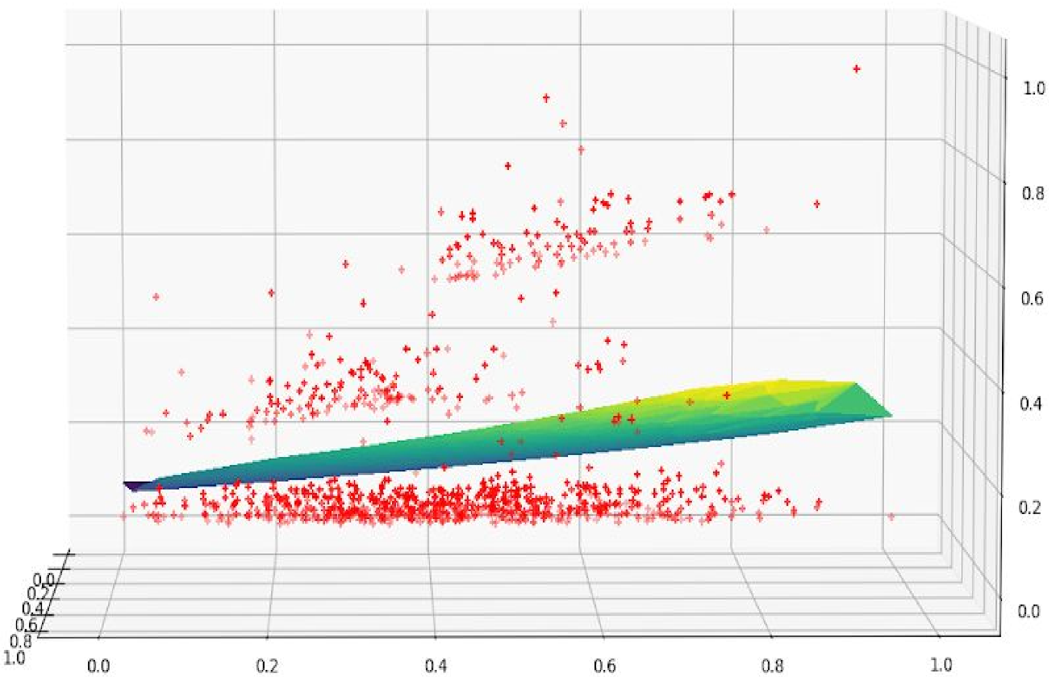
Degree : 3



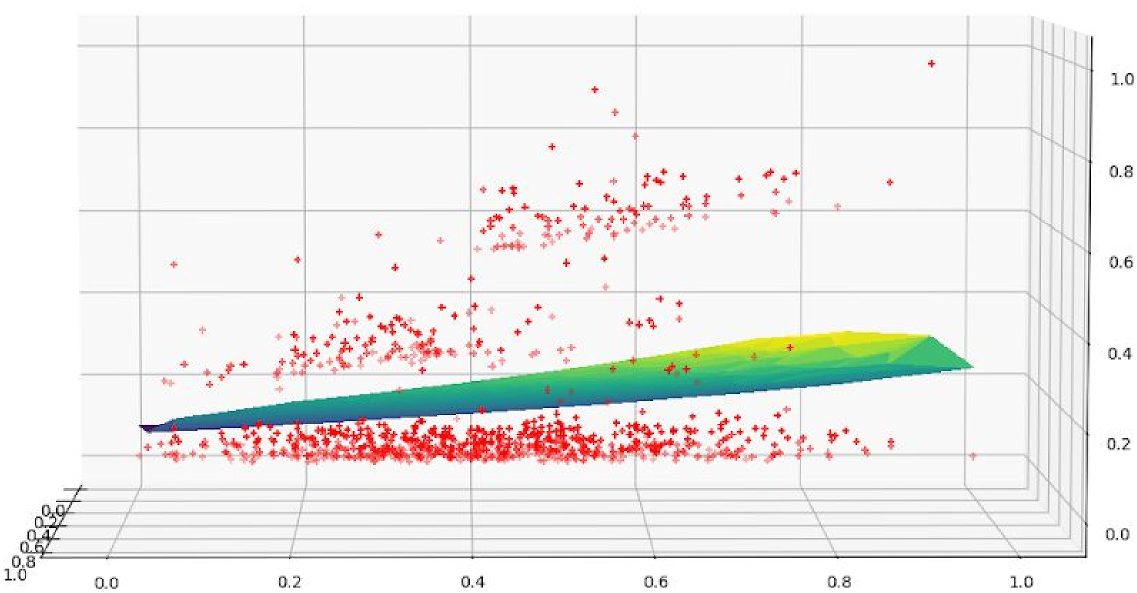
Degree : 4



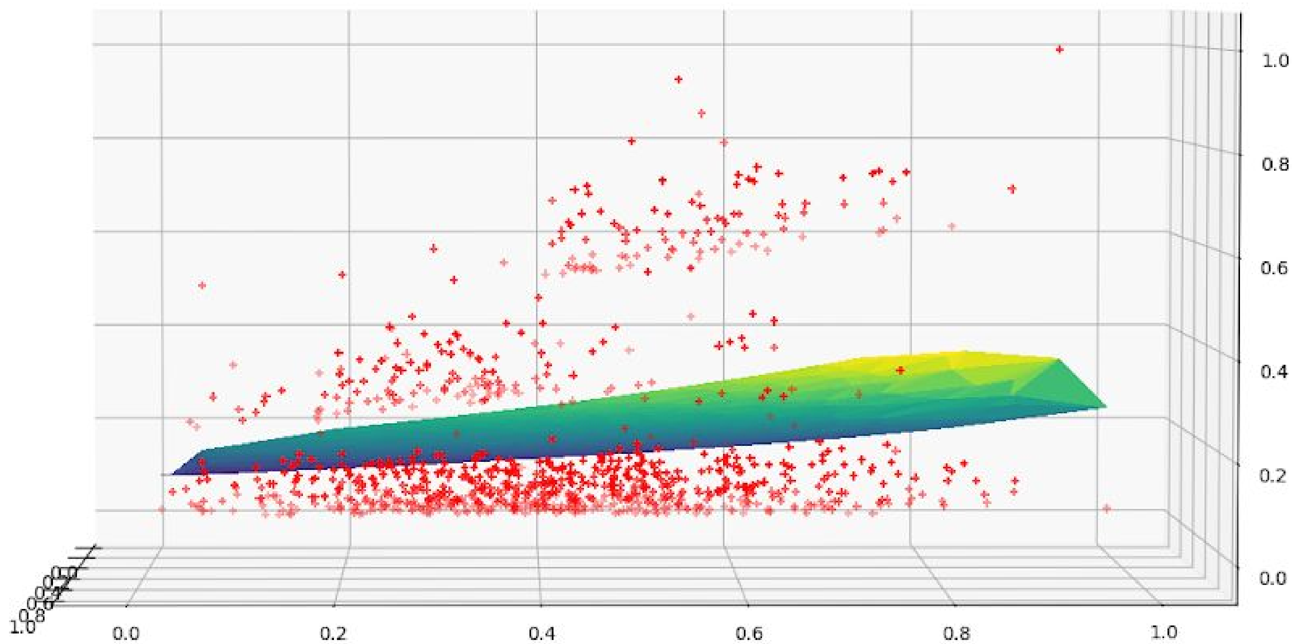
Degree : 5



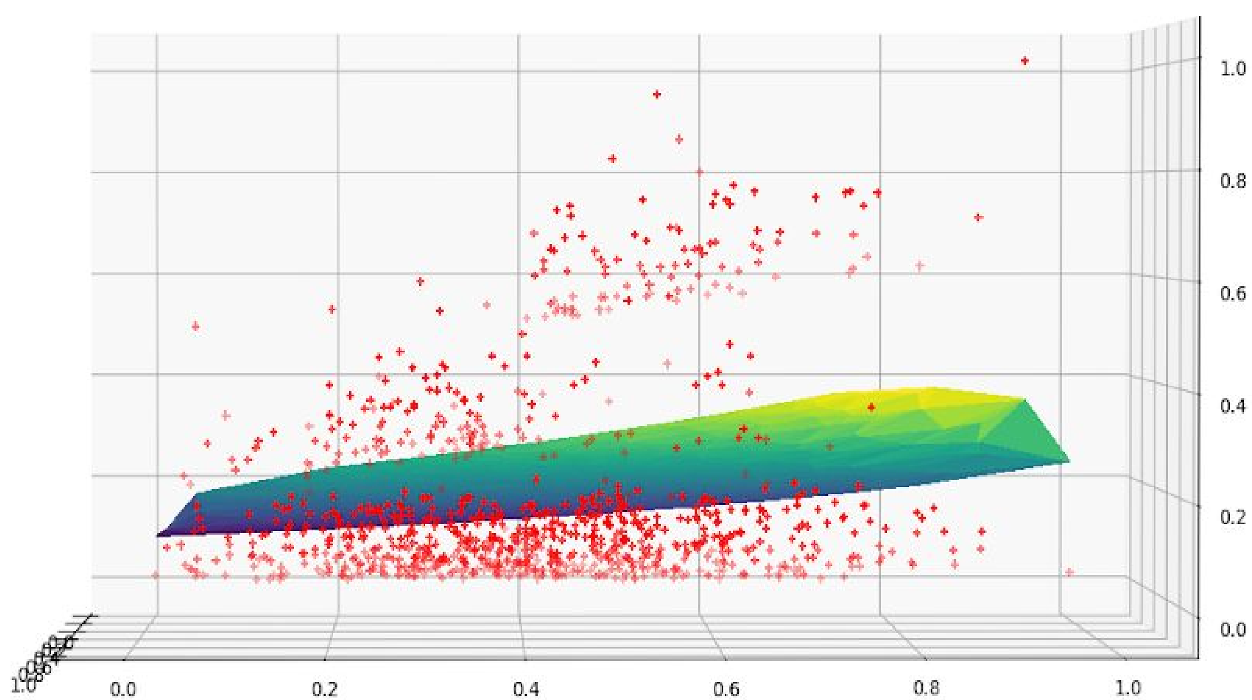
Degree : 6



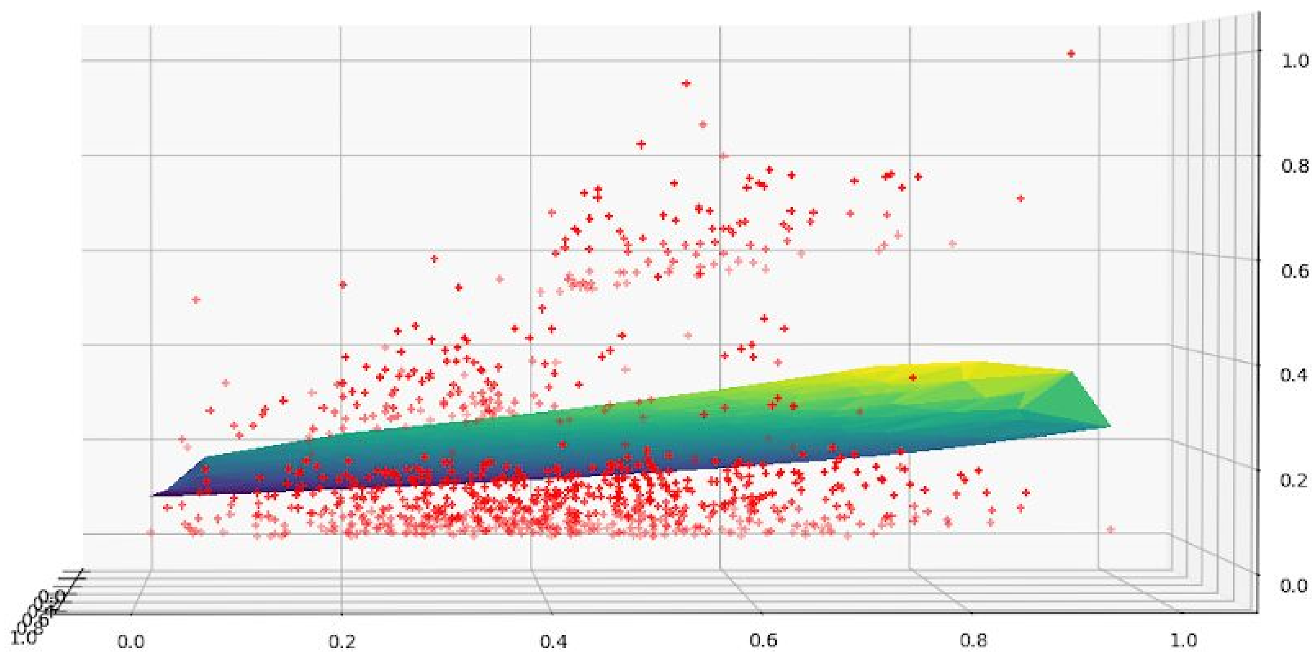
Degree : 7



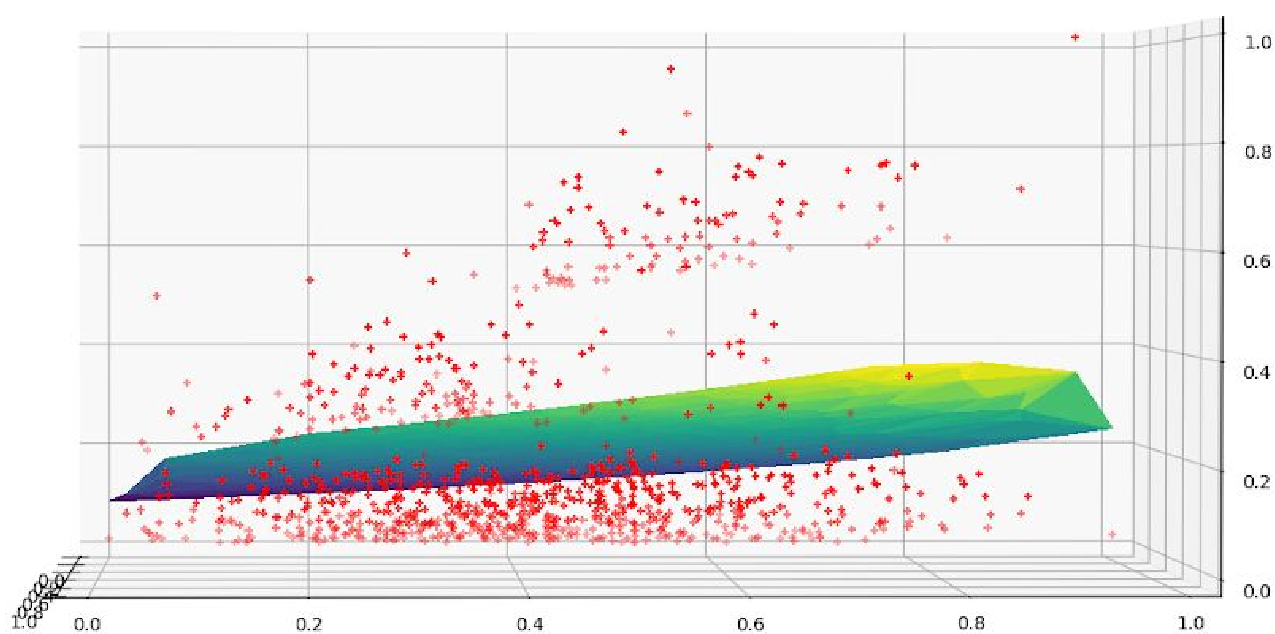
Degree : 8



Degree : 9



Degree : 10



How overfitting works?

We can see that as we increase the polynomial degree our model tries to fit as many as many data points as possible. In degree 1 we see the model is a plane very close to the bottom points because they are a lot compared to the points in the top. As we increase the degree our model tries to accommodate the top points also. So this is not good because the model tries to decrease the training error but sacrifices the testing error.

Therefore as the degree increases, overfitting increases which gives us bad testing error.

Questions to Ponder on

1. **What happens to the training and testing error as polynomials of higher degree are used for prediction?**

Ans: As the degree of our polynomial used for prediction increases, the training error decreases but the testing error increases. This happens because as the degree of our polynomial increases, the flexibility of the model increases which allows it to pass through more training points. As a result of this increased flexibility, the model starts describing the noise in the data too well. This model is said to be overfitting the data. As a result, the testing errors increase because the model starts describing the noise in the data and ignores the actual relationship between the feature and the target variables.

2. **Does a single global minimum exist for Polynomial Regression as well? If yes, justify.**

Ans: Yes, a single global minimum exists for polynomial regression as well. Even though the curve we are trying to fit is a polynomial of greater than or equal to 2 degrees, the error function obtained (ie., the sum of squared errors), is quadratic on the coefficients of the fitted polynomial similar to linear regression. Hence, the error function is convex and thus has only a single global minimum value.

3. **Which form of regularization curbs overfitting better in your case? Can you think of a case when Lasso regularization works better than Ridge?**

Ans: In our case, Ridge Regularization had slightly better results than Lasso Regularization. Lasso regularization performs better than Ridge regularization when we have a small number of significant features and other features have no effect on the target value. In such a case, Lasso regression can result in selecting only the relevant such that the coefficients of the insignificant features become zero. In the case of ridge regression, the coefficients can get close to zero but not exactly zero. Thus, lasso regression helps us achieve feature selection

4. **How does the regularization parameter affect the regularization process and weights? What would happen if a higher value for λ (> 2) was chosen?**

Ans: The regularisation parameter λ, gives a value of the importance given to the regularisation of the weights as compared to the minimisation of the error function. If λ is small, then very little importance is given to regularisation of the weights. Hence it can lead to a case of overfitting. If a higher value for λ is chosen, importance is given to regularisation and we might end up with a model that is not a good fit at the cost of regularising the weights.

5. **Regularization is necessary when you have a large number of features but limited training instances. Do you agree with this statement?**

Ans: Yes, regularisation is required when we have a large number of features but limited number of training instances. With a limited number of training examples, the predicted model would tend to overfit the data points. Hence, regularization is required to get a generalised model.

6. **If you are provided with D original features and are asked to generate new matured features of degree N, how many such new matured features will you be able to generate? Answer in terms of N and D.**

Ans: The number of matured features of degree N for a dataset having D original features will be

$$Number\ of\ features = \frac{(N+D)!}{(N!)(D!)}$$

7. **What is bias-variance trade off and how does it relate to overfitting and regularization.**

Ans: Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Variance is the amount that the estimate of the target function will change if different training data was used. If a model has a high bias, then there is considerable difference between the predicted value and the actual value. Hence, the model is said to be underfitting. On the other hand, if the variance is high, then on using a different training data, the target function changes considerably. This is because the predicted model overfits the data. We would want our model to neither be underfitting nor be overfitting, and therefore would want to minimise both the bias and the variance. Usually, the decrease in one leads to the increase in another, and this leads to a trade-off between bias and variance. Regularization tends to reduce the degree of overfitting, ie. the variance. However, too much regularization (by using a high value for the regularization parameter) can lead to an increase in bias.