# Assignment-1

Shivanshu Shekhar

March 13, 2023

## 1 Introduction

Linear Regression is a statistical method used to model the relationship between a dependent variable (also called the response variable) and one or more independent variables (also called predictors or explanatory variables). The goal of linear regression is to find the best linear relationship between the dependent variable and the independent variables.

In simple linear regression, there is only one independent variable and the relationship between the dependent variable and independent variable is represented by a straight line. The equation of the line is given by:

$y = \beta_0 + \beta_1 x + \epsilon$

where y is the dependent variable, x is the independent variable, $\beta_0$ and $\beta_1$ are the intercept and slope coefficients respectively, and $\epsilon$ is the error term that represents the unexplained variation in y.

## 2 Questions-1

### 2.1 Train, Test, Val split

We first import pandas to load the CSV files to python, we use the following lines of code to do that:

```
df1 = pd.read_csv("./data1.txt", delim_whitespace=True)
df2 = pd.read_csv("./data2.txt", delim_whitespace=True)
```

After getting the data loaded into python we get all the columns and convert them to NumPy arrays:

```
data1 = df1[df1.columns].to_numpy()
data2 = df2[df2.columns].to_numpy()
```

Given that the last column of the loaded data is the target values, we slice the data such that we take 70% for train, 20% for val, and 10% for test and leave the last row as the target variable for the same.

We do this with the following:

```
X_train = data[:int(0.7*data.shape[0]) + 1, :-1]
y_train = data[:int(0.7*data.shape[0]) + 1, -1].reshape(-1,1)
X_val = data[int(0.7*data.shape[0]): int(0.9*data.shape[0]) + 1, :-1]
y_val = data[int(0.7*data.shape[0]): int(0.9*data.shape[0]) + 1, -1].reshape(-1,1)
X_test = data[int(0.9*data.shape[0]): , :-1]
y_test = data[int(0.9*data.shape[0]): , -1].reshape(-1,1)
```

## 2.2 Fitting different degree polynomials

We fit the data to our model by using the normal equation:

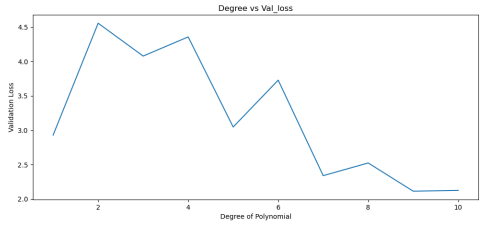
$$W_{ML} = (\phi^T \phi)^{-1} \phi^T y$$

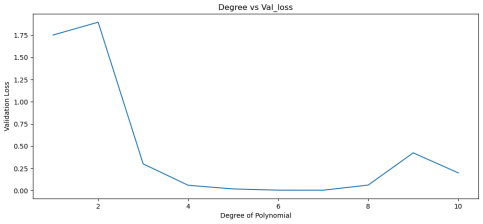
To create $\phi$ we basically append $X^i$ for i $\in [0, degree]$ for each feature. This is done by the **transform** function.

The **normal_equation** function gives us the weight vector($W_{ML}$) value and we use this value to get the validation loss. The loss used here is Mean Squared Error loss.

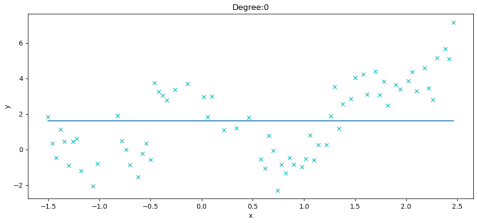
All these steps are combined in one final function named **solve** which returns the best degree of fit for the data.

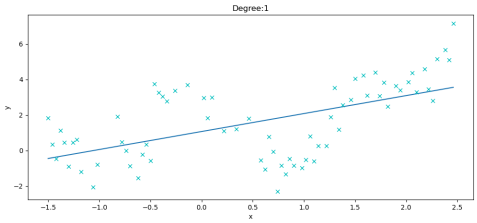
We next plot the validation loss vs degree of the polynomial for both datasets.

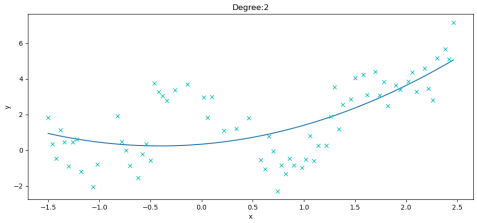


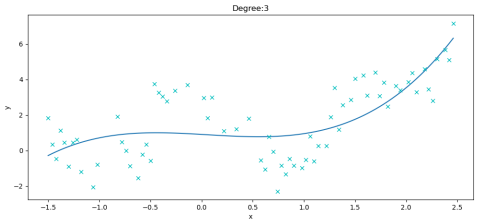



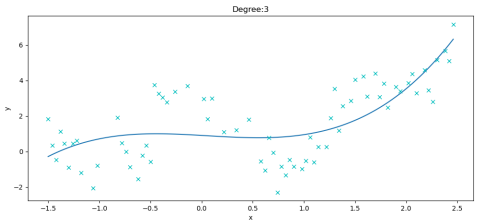

The plots of training data with the polynomial fitted are shown below:

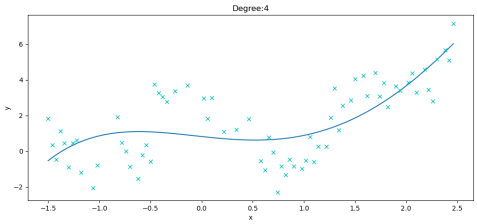


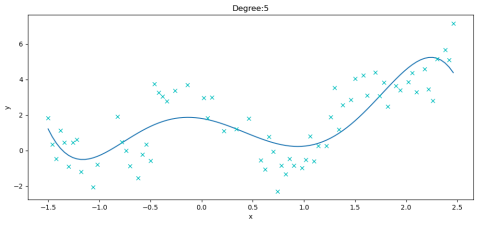

Degree:1
x
y

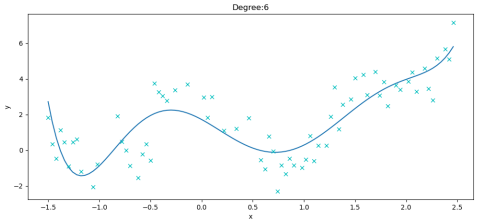

Degree:2
x
y

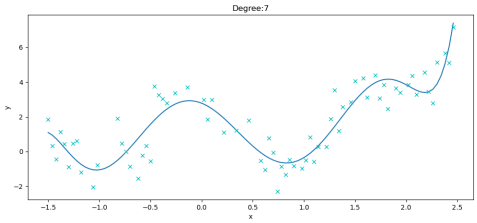

Degree:3
x
y

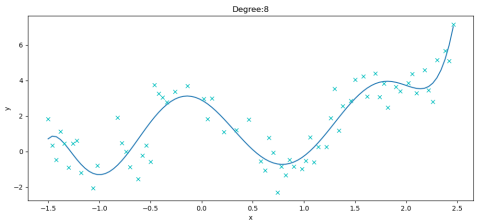

Degree:3
x
y

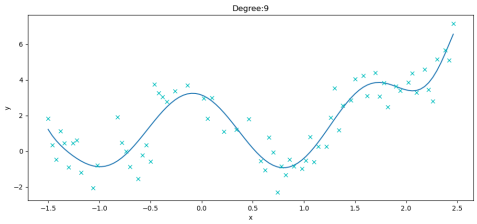

Degree:4
x
y

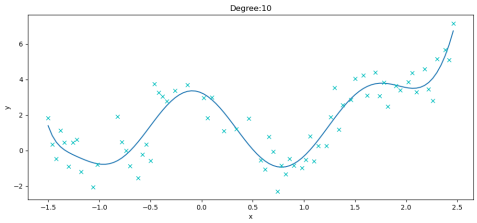

Degree:5
x
y

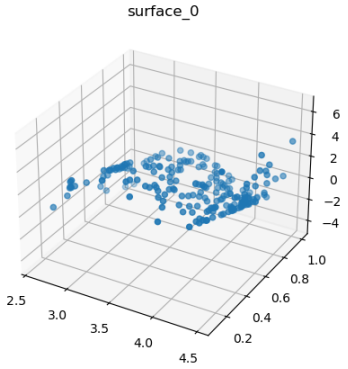

Degree:6
x
y

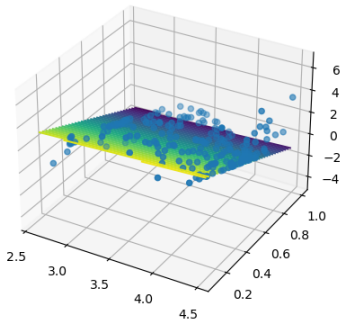

Degree:7
x
y

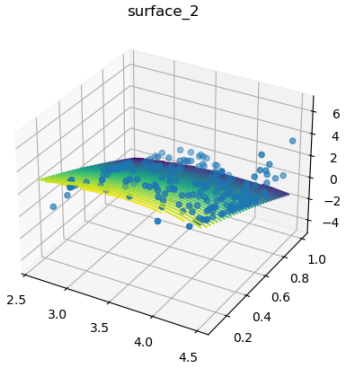

Degree:8
x
y

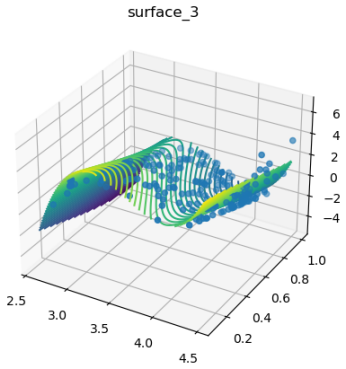

Degree:9
x
y

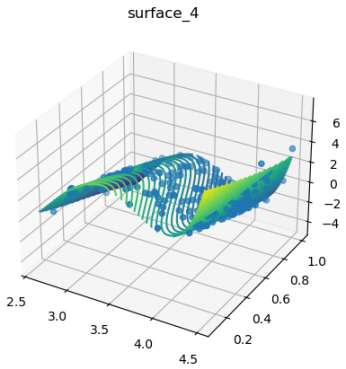

Degree:10
y
x


surface_0

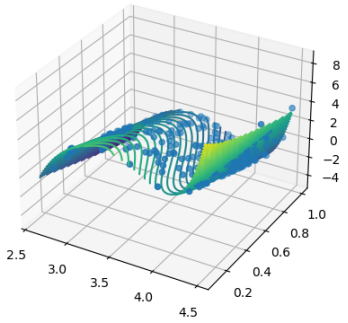

surface_1


surface_2
6
4
2
0
−2
−4
1.0
0.8
0.6
0.4
0.2
2.5
3.0
3.5
4.0
4.5

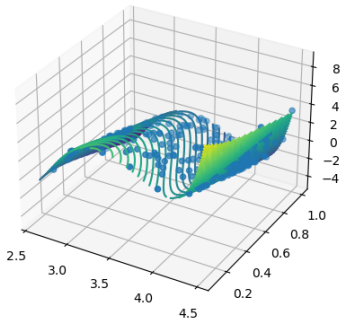

surface_3
6
4
2
0
−2
−4
1.0
0.8
0.6
0.4
0.2
2.5
3.0
3.5
4.0
4.5


surface_4
6
4
2
0
−2
−4
1.0
0.8
0.6
0.4
0.2
2.5
3.0
3.5
4.0
4.5

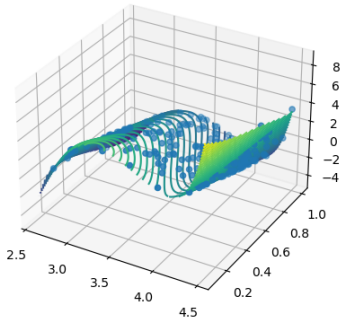
surface_5

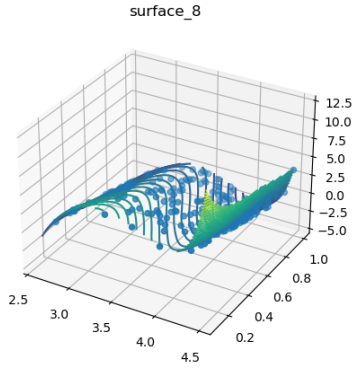


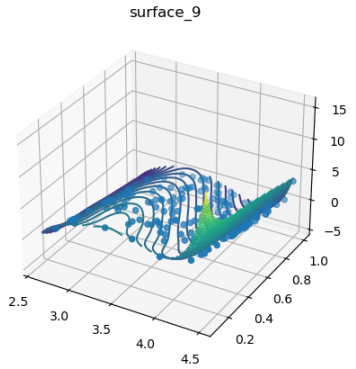

surface_6

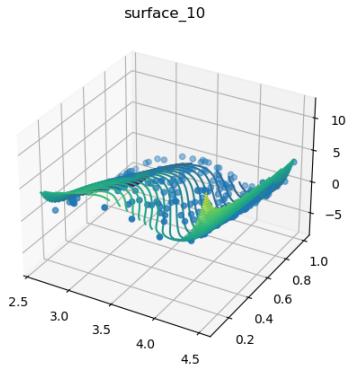



surface_7






## 2.3 Ridge regression and lambda

Ridge regression is a regularization technique used in linear regression models to address the problem of overfitting. In traditional linear regression, the

goal is to minimize the sum of the squared residuals between the predicted and actual values. However, when there are many predictors (features) in the model, the coefficients can become large and the model may start to overfit, meaning it performs well on the training data but poorly on new, unseen data.

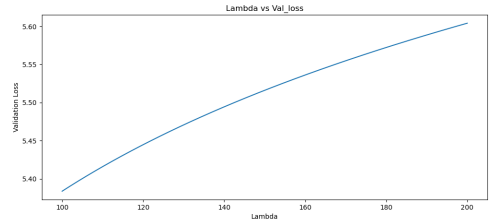
Ridge regression adds a penalty term to the cost function in linear regression that shrinks the coefficients towards zero, reducing the effect of the predictors that are less important in explaining the target variable. The penalty term is determined by a hyperparameter called lambda ($\lambda$), which controls the amount of shrinkage. As lambda increases, the coefficients are pushed closer to zero, and the model becomes simpler and less prone to overfitting.

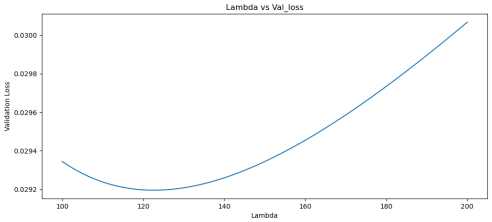
$$J(\beta) = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2 \tag{1}$$

We did the same transformations for ridge regression as well. Still, here the only difference is that we already know the degree of the polynomial that best fits the model with regard to validation accuracy. In ridge regression, we are finding validation loss for various values of the lambda parameter.

The code for this is almost the same just that here we are varying lambda instead of the degree. We define a function named **solve2** that does the aforementioned for us.

We next plot the validation loss vs lambda graphs for both datasets.



# 3 Question-2

We import the data into python as we did back in question 1:

```
dataT = pd.read_csv("./Train.csv")

X_train = dataT[dataT.columns[:-1]].to_numpy()
X_train = transform(X_train)
y_train = dataT[dataT.columns[-1]].to_numpy().reshape(-1,1)

dataV = pd.read_csv("./test.csv")

X_val = dataV[dataV.columns[:-1]].to_numpy()
X_val = transform(X_val)
y_val = dataV[dataV.columns[-1]].to_numpy().reshape(-1,1)
```

## 3.1 Obtaining $W_{ML}$

We know that:

$$W_{ML} = (\phi^T \phi)^{-1} \phi^T y$$

So we simply estimate $W_{ML}$ from here.

## 3.2 Gradient Descent

We know that the Least Squared Error is:

$$L(W) = 1/2 * (XW - y)^T (XW - y)$$

This gives us:

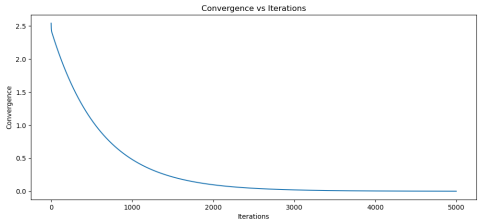
$$\nabla L(W) = X^T (XW - y)$$

Using this we can write our gradient descent update rule as:

$$W_{t+1} = W_t - \eta \nabla L(W)$$

So we update W till convergence(or some large number of iterations), we also need to set a small learning rate $\eta$ for the training to be successful.

We observe that $W_R$ converges to $W_{ML}$ in approximately 3000 iterations.




We also transform X like in question-1 for the aforementioned reasons.

### 3.3 Gradient Descent for Ridge Regression

It is almost the same as the above analysis, just that here the gradient of loss function with respect to W changes slightly because the loss function itself changes.

The new loss function is:

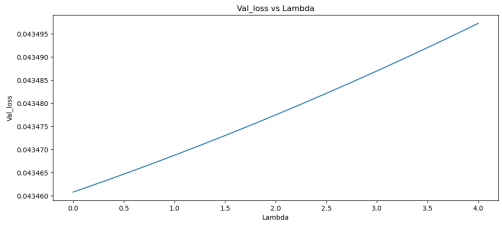
$$L(W) = 1/2 * (XW - y)^T(XW - y) + \lambda\|W\|^2/2$$

So the corresponding gradient becomes:

$$\nabla L(W) = X^T(XW - y) + \lambda W$$

The update rule remains the same.

We plot the validation error vs lambda next.




Next we choose the best $W_R$ from the various $W_R$'s obtained for the various lambda values and then we compare this to $W_{ML}$ and we see that $W_R$ performed better on the validation set although the best regularization lambda coefficient is 0.0, I suspect that this is due to early stopping nature of gradient descent which brings in regularization by not updating the weights to be as high as $W_ML$, I also saw that if we used k-fold validation instead of normal validation then we are getting regularization term around 1.

We did the following from the lines of code given below:

```
la = np.linspace(0, 4, num=50)

best = []
prev_loss = 10000

for l in tqdm(la):
    W = np.zeros_like(W_ML)
    for i in range(it):
        W = grad_update(X_train, W ,y_train, lr, l)
    y_pred = X_val@W
    curr_loss = loss(y_pred, y_val)
    y.append(curr_loss)
    if curr_loss<= prev_loss:
        best.clear()
        best.append(l); best.append(W)
        #print(l)
        prev_loss = curr_loss
```

We got best $W_R$ as 0.3659610 and we got $W_ML$ as 0.3721524

# 4 Conclusion

In this assignment we learned how to use normal equations and gradient descent to fit the data to linear models, we also saw how the model starts to fit into the noise of the data with increasing complexity and how we can tune the lambda parameter to get the optimal solution between model complexity and data fitting.