Chapter 2 : Section 3

**Polynomial Regression**

Introduction to Polynomial Regressions

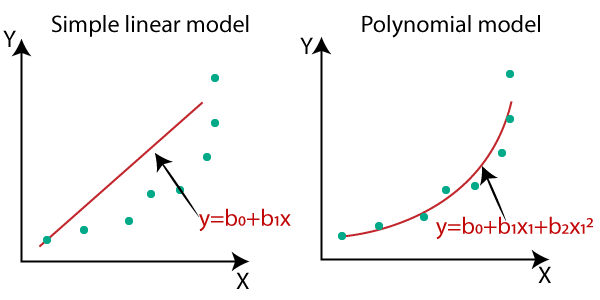
From this section we are making some new kind of regressors which are slightly more advanced regressors especially for ***SVR*** or ***random*** ***Forrest***. This polynomial regression model that we're about to build right now is not that much advanced compared to simple regression and multiple regression because we will just add a *polynomial* *term* in the *multiple* *linear* *regression* equation. But SVR or Random Forrest will be based on more complex theory.

**2.3.1 Polynomial Regression**

Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

Notice there is only one feature is present.

* It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the *Multiple Linear regression* equation to convert it into *Polynomial* *Regression*.
* It is a *linear model* with some modification in order to *increase* the *accuracy*. Linearity refers to the coefficients (because they are the unknown here) ***rather than feature variables***.
* The dataset used in *Polynomial regression* for training is of *non-linear nature*. Eg: Population growth.
* It makes use of a *linear regression* model to fit the *complicated* and *non-linear functions* and datasets.
* In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model.
* Need for Polynomial Regression: If we apply a ***linear model*** on a ***linear*** ***dataset***, then it provides us a good result as we have seen in ***Simple Linear Regression***, but if we apply the same model without any modification on a ***non-linear dataset***, then it will produce a ***drastic*** output. Due to which ***loss*** ***function*** will ***increase***, the ***error*** ***rate*** will be ***high***, and ***accuracy*** will be ***decreased***.
* So for such cases, where ***data points are arranged in a non-linear fashion***, we need the Polynomial Regression model. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



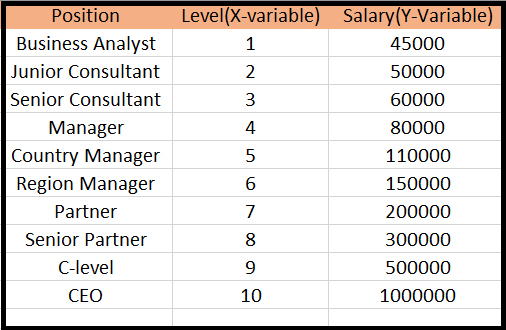
In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.

* Why Linear: A Polynomial Regression algorithm is also called Polynomial LINEAR Regression because it *does not depend on the variables*, instead, it *depends on the coefficients*, which are arranged in a linear fashion (Generally polynomial equations are Non-Linear). Hence the Linear or Non-linear refers to the coefficients. For example: is a *non-linear regression*.
* Equation of the Polynomial Regression Model:
* Simple Linear Regression equation:
* Multiple Linear Regression equation:
* Polynomial Regression equation:
* We can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The ***Simple*** and ***Multiple*** ***Linear equations are also Polynomial equations*** with a ***single*** ***degree***, and the ***Polynomial regression equation*** is ***Linear equation*** with the ***nth degree***. (In mathematics Polynomial equations are nonlinear equations. Here in Regression it *does not depend on the variables*). So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

**2.3.2 Implementation of Polynomial Regression using Python**

Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

* Problem Description: There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff.
* So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a *non-linear relationship* between the Position levels and the salaries. Our goal is to build a ***Bluffing detector regression model***, so HR can hire an honest candidate. Below are the steps to build such a model.



* Steps for Polynomial Regression: The main steps involved in *Polynomial* *Regression* are given below:

1. Data *Pre-processing*
2. *Build* a *Linear* *Regression* model and *fit* it to the dataset
3. *Build* a *Polynomial* *Regression* model and *fit* it to the dataset
4. *Visualize* the result for *Linear* *Regression* and *Polynomial* *Regression* model.
5. *Predicting* the output.

* Why Linear & Polynomial both: Here, we will build the *Linear* *regression* model as well as *Polynomial* *Regression* to see the ***results between the predictions***. And Linear regression model is for *reference*.

1. Step1 - Data Pre-processing: The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will *not use feature scaling,* and also we will *not* *split* our *dataset* into training and test set. It has two reasons:

* The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.
* In this model, we want very accurate predictions for salary, so the model should have enough information.

**Pre-processing step**

**import** pandas **as** pNd

**import** matplotlib **as** pLt

**import** numpy **as** nPy

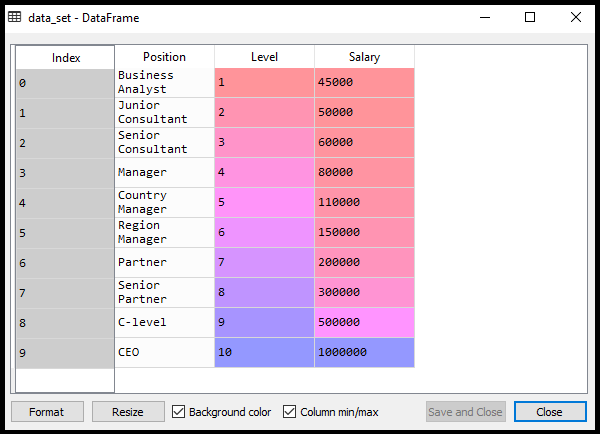
dataSet = pNd**.read\_csv**("Position\_Salaries.csv")

X = dataSet**.**iloc[:, 1:2]**.**values

#*X = dataSet.iloc[:, [1]].values # same as above*

y = dataSet**.**iloc[:, 2]**.**values

* We will consider only two columns (Salary and Levels). Because " Levels " indicate " Positions".
* Trick to make feature-matrix: For x-variable, we have taken parameters as **[:,1:2]**, because we want 1 index(levels), and included :2 to make it as a matrix (we could also use **dataSet.iloc[:, [1]]**).



* Here we will ***predict*** the output for ***level 6.5*** because the candidate has 4+ years' experience as a regional manager, so he must be somewhere between levels 7 and 6.

#*---------- Simple Linear model for Referace -------------*

**from** sklearn**.**linear\_model **import** LinearRegression

lin\_regressor\_1 = **LinearRegression**()    #*object for Simple Linear Regression*

lin\_regressor\_1**.fit**(X, y)

#*----------- Polynomial regressor --------------------*

**from** sklearn**.**preprocessing **import** PolynomialFeatures

#*Set the polynomial for X*

poly\_regressor = **PolynomialFeatures**(degree= 2)

X\_poly = poly\_regressor**.fit\_transform**(X) #*Generates Polynomial Feature-Matrix*

lin\_regressor\_2 = **LinearRegression**()    #*object for Polinomial Regression*

lin\_regressor\_2**.fit**(X\_poly, y)

#*------------  Visualize the result  --------------*

y\_lin\_pred = lin\_regressor\_1**.predict**(X)

#*y\_poly\_pred = lin\_regressor\_2.predict(X): Does not work- Have to use Polynomial Feature-Matrix*

y\_poly\_pred = lin\_regressor\_2**.predict**(X\_poly)

pLt**.scatter**(X, y, color = "red")

pLt**.plot**(X, y\_lin\_pred, color = "blue")

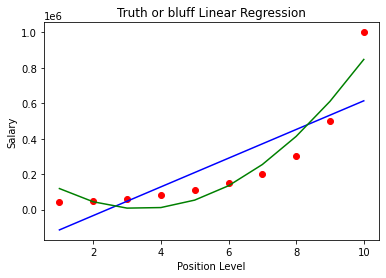
pLt**.title**('Truth or bluff Linear Regression')

pLt**.xlabel**('Position Level')

pLt**.ylabel**("Salary")

pLt**.plot**(X, y\_poly\_pred, color = "green")

pLt**.show**()



1. Step 2 - Building the Linear regression model:

In building polynomial regression, we will take the ***Linear*** ***regression*** model as reference and compare both the ***results***. The code is given below:

#*---------- Simple Linear model for Referace -------------*

**from** sklearn**.**linear\_model **import** LinearRegression

lin\_regressor\_1 = **LinearRegression**()    #*object for Simple Linear Regression*

lin\_regressor\_1**.fit**(X, y)

In the above code, we have created the Simple Linear model using ***lin\_regressor\_1*** object of ***LinearRegression*** class and fitted it to the dataset variables (X and y).

1. Step 3 - Building the Polynomial regression model:

Now we will build the Polynomial Regression model, but it will be a little different from the Simple Linear model. Because here we will use ***PolynomialFeatures*** class of ***preprocessing*** library. We are using this class to add some extra features to our dataset.

#*----------- Polynomial regressor --------------------*

**from** sklearn**.**preprocessing **import** PolynomialFeatures

#*Set the polynomial for X*

poly\_regressor = **PolynomialFeatures**(degree= 2)

X\_poly = poly\_regressor**.fit\_transform**(X) #*Generates Polynomial Feature-Matrix*

lin\_regressor\_2 = **LinearRegression**()    #*object for Polinomial Regression*

lin\_regressor\_2**.fit**(X\_poly, y)

|  |  |
| --- | --- |
| * ***poly\_regressor.fit\_transform(X)*** used converting our feature matrix into Polynomial Feature Matrix, * The parameter value***(degree= 2)*** depends on our choice. We can choose it according to our Polynomial features. * After executing the code, we will get another matrix ***X\_poly***, which can be seen under the variable explorer option: * Next, we have used another ***LinearRegression*** object, namely ***lin\_regressor\_2***, to fit our x\_poly vector to the linear model. * The ***1st column*** is the "***constant-Column***" Recall Multiple-Linear-regression (this column is automatically created). ***2nd column*** is for ***1st-degrre*** term and 3rd column is the ***2nd –degree*** term. |  |

* Polynomial Feature-Matrix: This ***poly\_regressor*** object is going to be a transformer tool that will transform our matrix of feature ***X*** into a new matrix of features that we're going to call ***X\_poly*** which will be a new matrix of features containing not only this independent variable but also  **i.e. it actually adds the polynomial terms**. The can be set to any integer (2, 3, 4, .. ) using *PolynomialFeatures(****degree= 2****)*.

**from** sklearn**.**preprocessing **import** PolynomialFeatures

poly\_regressor = **PolynomialFeatures**(degree= 2) #*Set the degree of polynomial for X*

X\_poly = poly\_regressor**.fit\_transform**(X) #*Generates Polynomial Feature-Matrix*

* These three lines actually transformed our original *matrix of features* **X** into our *new matrix of features* containing the original *independent* *variable* **X** and its associated *polynomial* *terms*.
* Include this X\_poly fit into a multiple regression model: In the following two lines we created a new linear regression object ***lin\_regressor\_2*** that we fitted to this new matrix ***X\_poly*** and now original dependent variable vector ***y***.

lin\_regressor\_2 = **LinearRegression**()    #*object for Polinomial Regression*

lin\_regressor\_2**.fit**(X\_poly, y)

1. Step 4 - Visualizing the result for Linear & Polynomial regression:

|  |  |
| --- | --- |
| #*======= Visualizing Simple  Linear ============*  y\_lin\_pred = lin\_regressor\_1**.predict**(X)  pLt**.scatter**(X, y, color = "red")  pLt**.plot**(X, y\_lin\_pred, color = "blue")  pLt**.title**('Truth or bluff Linear Regression')  pLt**.xlabel**('Position Level')  pLt**.ylabel**("Salary")  pLt**.show**() |  |

* If we consider this output to predict the value of CEO, it will give a salary of approx. 600000$, which is far away from the real value. So we need a curved model to fit the dataset other than a straight line.

|  |  |
| --- | --- |
| #*======= Polynomial ============*  #***y\_poly\_pred = lin\_regressor\_2.predict(X)****: Does not work- Have to use Polynomial Feature-Matrix*  y\_poly\_pred = lin\_regressor\_2**.predict**(X\_poly)  pLt**.scatter**(X, y, color = "red")  pLt**.title**('Truth or bluff Linear Regression')  pLt**.xlabel**('Position Level')  pLt**.ylabel**("Salary")  pLt**.plot**(X, y\_poly\_pred, color = "green") |  |

* Notice: ***X\_poly*** is used with ***lin\_regressor\_2*** and we cannot use ***X***. The reason is ***lin\_regressor\_1*** is the model built upon Single feature ***X*** but ***lin\_regressor\_2*** is the model built upon extra polynomial terms of feature variable .

y\_poly\_pred = lin\_regressor\_2**.predict**(X\_poly)

**Fitting the model by Increasing degree.**

|  |  |
| --- | --- |
| For degree= 3: If we change the degree=3, then we will give a more accurate plot, as shown in the below image. | Degree= 4: Hence we can get more accurate results by increasing the degree of Polynomial. |
| ML Polynomial Regression | ML Polynomial Regression |

1. Step – 5: Predicting the final result with the Linear Regression model:

Now, we will predict the final output using the ***Linear regression*** model to see whether an employee is saying truth or bluff. So, for this, we will use the ***predict()*** method and will pass the value ***6.5***. Below is the code for it:

#*----------------- Prediction -------------------*

lin\_pred = lin\_regressor\_1**.predict**([[6.5]])  #*Simple linear Regerssion*

**print**(lin\_pred)

poly\_pred = lin\_regressor\_2**.predict**(poly\_regressor**.fit\_transform**([[6.5]]))  #*Polynl linear Regerssion*

**print**(poly\_pred)

As we can see, the predicted output for the Polynomial Regression is [158862.45265153], which is much closer to real value hence, we can say that future employee is saying true.

Practiced version

#*--------------- Import Libraries --------------*

**import** pandas **as** pNd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** nPy

#*--------------- Data Preprocessing  -------------*

dataSet = pNd**.read\_csv**("Position\_Salaries.csv")

X = dataSet**.**iloc[:, 1:2]**.**values

#*X = dataSet.iloc[:, [1]].values # same as above*

y = dataSet**.**iloc[:, 2]**.**values

#*---------- Simple Linear model for Referace -------------*

**from** sklearn**.**linear\_model **import** LinearRegression

lin\_regressor\_1 = **LinearRegression**()    #*object for Simple Linear Regression*

lin\_regressor\_1**.fit**(X, y)

#*----------- Polynomial regressor --------------------*

**from** sklearn**.**preprocessing **import** PolynomialFeatures

#*Set the polynomial for X*

poly\_regressor = **PolynomialFeatures**(degree= 4)

X\_poly = poly\_regressor**.fit\_transform**(X) #*Generates Polynomial Feature-Matrix*

lin\_regressor\_2 = **LinearRegression**()    #*object for Polinomial Regression*

lin\_regressor\_2**.fit**(X\_poly, y)

#*------------  Visualize the result  --------------*

    #*======= Simple  Linear ============*

y\_lin\_pred = lin\_regressor\_1**.predict**(X)

pLt**.scatter**(X, y, color = "red")

pLt**.plot**(X, y\_lin\_pred, color = "blue")

pLt**.title**('Truth or bluff Linear Regression')

pLt**.xlabel**('Position Level')

pLt**.ylabel**("Salary")

pLt**.show**()

    #*======= Polynomial ============*

#*y\_poly\_pred = lin\_regressor\_2.predict(X): Does not work- Have to use Polynomial Feature-Matrix*

y\_poly\_pred = lin\_regressor\_2**.predict**(X\_poly)

pLt**.scatter**(X, y, color = "red")

pLt**.title**('Truth or bluff Linear Regression')

pLt**.xlabel**('Position Level')

pLt**.ylabel**("Salary")

pLt**.plot**(X, y\_poly\_pred, color = "green")

pLt**.show**()

#*----------------- Prediction -------------------*

lin\_pred = lin\_regressor\_1**.predict**([[6.5]])  #*Simple linear Regerssion*

**print**(lin\_pred)

poly\_pred = lin\_regressor\_2**.predict**(poly\_regressor**.fit\_transform**([[6.5]]))  #*Polynl linear Regerssion*

**print**(poly\_pred)

#*python prctc\_polnm\_rgsn.py*

**Instructor version**

#*Polynomial Regression*

#*Importing the libraries*

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**import** pandas **as** pd

#*Importing the dataset*

dataset = pd**.read\_csv**('Position\_Salaries.csv')

X = dataset**.**iloc[:, 1:-1]**.**values

y = dataset**.**iloc[:, -1]**.**values

#*Training the Linear Regression model on the whole dataset*

**from** sklearn**.**linear\_model **import** LinearRegression

lin\_reg = **LinearRegression**()

lin\_reg**.fit**(X, y)

#*Training the Polynomial Regression model on the whole dataset*

**from** sklearn**.**preprocessing **import** PolynomialFeatures

poly\_reg = **PolynomialFeatures**(degree = 4)

X\_poly = poly\_reg**.fit\_transform**(X)

poly\_reg**.fit**(X\_poly, y)

lin\_reg\_2 = **LinearRegression**()

lin\_reg\_2**.fit**(X\_poly, y)

#*Visualising the Linear Regression results*

plt**.scatter**(X, y, color = 'red')

plt**.plot**(X, lin\_reg**.predict**(X), color = 'blue')

plt**.title**('Truth or Bluff (Linear Regression)')

plt**.xlabel**('Position level')

plt**.ylabel**('Salary')

plt**.show**()

#*Visualising the Polynomial Regression results*

plt**.scatter**(X, y, color = 'red')

plt**.plot**(X, lin\_reg\_2**.predict**(poly\_reg**.fit\_transform**(X)), color = 'blue')

plt**.title**('Truth or Bluff (Polynomial Regression)')

plt**.xlabel**('Position level')

plt**.ylabel**('Salary')

plt**.show**()

#*Visualising the Polynomial Regression results (for higher resolution and smoother curve)*

X\_grid = np**.arange**(min(X), max(X), 0.1)

X\_grid = X\_grid**.reshape**((len(X\_grid), 1))

plt**.scatter**(X, y, color = 'red')

plt**.plot**(X\_grid, lin\_reg\_2**.predict**(poly\_reg**.fit\_transform**(X\_grid)), color = 'blue')

plt**.title**('Truth or Bluff (Polynomial Regression)')

plt**.xlabel**('Position level')

plt**.ylabel**('Salary')

plt**.show**()

#*Predicting a new result with Linear Regression*

lin\_reg**.predict**([[6.5]])

#*Predicting a new result with Polynomial Regression*

lin\_reg\_2**.predict**(poly\_reg**.fit\_transform**([[6.5]]))

**New Template for Non-Linear Regression**

Follow the Instructor version

#*Regression Template*

#*Importing the libraries*

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**import** pandas **as** pd

#*Importing the dataset*

dataset = pd**.read\_csv**('Position\_Salaries.csv')

X = dataset**.**iloc[:, 1:2]**.**values

y = dataset**.**iloc[:, 2]**.**values

#*Splitting the dataset into the Training set and Test set*

"""from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)"""

#*Feature Scaling*

"""from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

sc\_y = StandardScaler()

y\_train = sc\_y.fit\_transform(y\_train.reshape(-1,1))"""

#*Fitting the Regression Model to the dataset*

#*Create your regressor here*

#*Predicting a new result*

y\_pred = regressor**.predict**(6.5)

#*Visualising the Regression results*

plt**.scatter**(X, y, color = 'red')

plt**.plot**(X, regressor**.predict**(X), color = 'blue')

plt**.title**('Truth or Bluff (Regression Model)')

plt**.xlabel**('Position level')

plt**.ylabel**('Salary')

plt**.show**()

#*Visualising the Regression results (for higher resolution and smoother curve)*

X\_grid = np**.arange**(min(X), max(X), 0.1)

X\_grid = X\_grid**.reshape**((len(X\_grid), 1))

plt**.scatter**(X, y, color = 'red')

plt**.plot**(X\_grid, regressor**.predict**(X\_grid), color = 'blue')

plt**.title**('Truth or Bluff (Regression Model)')

plt**.xlabel**('Position level')

plt**.ylabel**('Salary')

plt**.show**()