ML 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:

$y= b_0+b_1x_1+ b_2x_1{}^2+ b_2x_1{}^3+...... b_nx_1{}^n$

- 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.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence, "*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:

The need of Polynomial Regression in ML can be understood in the below points:

- 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.

Simple linear model — $y = b_0 + b_1 x$

Polynomial model — $y = b_0 + b_1 x_1 + b_2 x_1^2$
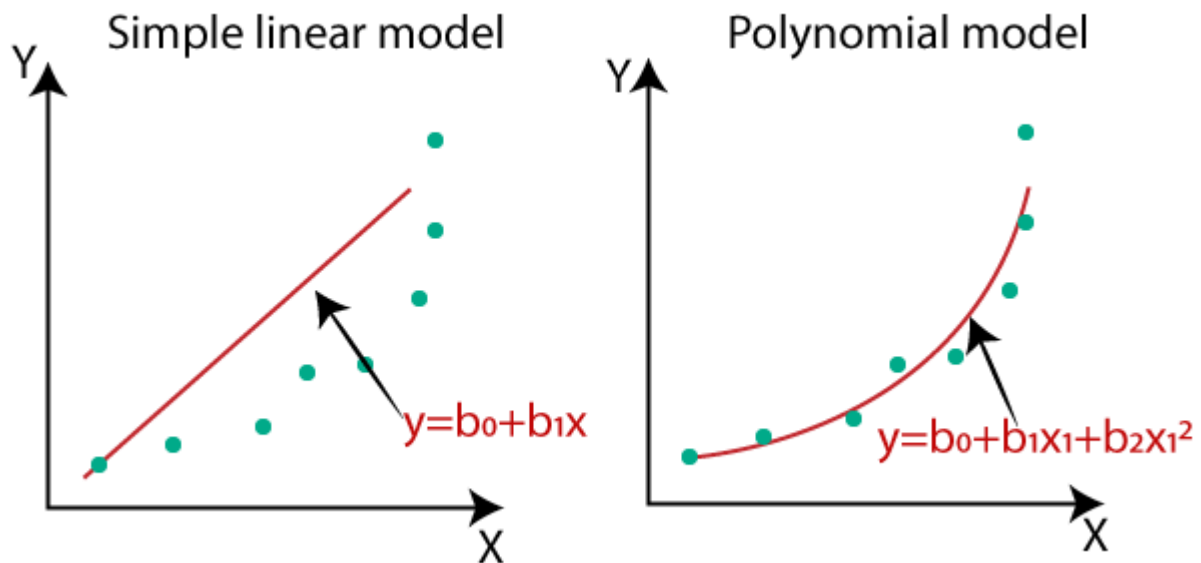
- o In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.

- o Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

Note: 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.

Equation of the Polynomial Regression Model:

Simple Linear Regression equation: $y = b_0 + b_1 x$ ....(a)

Multiple Linear Regression equation: $y = b_0 + b_1 x + b_2 x_2 + b_3 x_3 + ... + b_n x_n$ ....(b)

Polynomial Regression equation: $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + ... + b_n x^n$ ....(c)

PlayNext
Unmute

Current TimeÂ 0:00

/

When we compare the above three equations, 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. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

Note: To better understand Polynomial Regression, you must have knowledge of Simple Linear Regression.

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.

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Steps for Polynomial Regression:

The main steps involved in Polynomial Regression are given below:

- o Data Pre-processing
- o Build a Linear Regression model and fit it to the dataset
- o Build a Polynomial Regression model and fit it to the dataset
- o Visualize the result for Linear Regression and Polynomial Regression model.
- o Predicting the output.

Note: 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.

Data Pre-processing Step:

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:

- o 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.
- o In this model, we want very accurate predictions for salary, so the model should have enough information.

The code for pre-processing step is given below:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('Position_Salaries.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, 1:2].values
11. y= data_set.iloc[:, 2].values

Explanation:

o In the above lines of code, we have imported the important Python libraries to import dataset and operate on it.

o Next, we have imported the dataset '**Position_Salaries.csv**', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).

o After that, we have extracted the dependent(Y) and independent variable(X) from the dataset. 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.

Output:

By executing the above code, we can read our dataset as:

As we can see in the above output, there are three columns present (Positions, Levels, and Salaries). But we are only considering two columns because Positions are equivalent to the levels or may be seen as the encoded form of Positions.

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.

Building the Linear regression model:

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

1. #Fitting the Linear Regression to the dataset
2. from sklearn.linear_model **import** LinearRegression
3. lin_regs= LinearRegression()
4. lin_regs.fit(x,y)

In the above code, we have created the Simple Linear model using **lin_regs** object of **LinearRegression** class and fitted it to the dataset variables (x and y).

Output:

```
Out[5]:   LinearRegression(copy_X=True,   fit_intercept=True,   n_jobs=None,
normalize=False)
```

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.

1.   #Fitting the Polynomial regression to the dataset
2.   from sklearn.preprocessing **import** PolynomialFeatures
3.   poly_regs= PolynomialFeatures(degree= 2)
4.   x_poly= poly_regs.fit_transform(x)
5.   lin_reg_2 =LinearRegression()
6.   lin_reg_2.fit(x_poly, y)

In the above lines of code, we have used **poly_regs.fit_transform(x)**, because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model. 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:

x_poly - NumPy array

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25
5	1	6	36
6	1	7	49
7	1	8	64
8	1	9	81
9	1	10	100

Format   Resize   ☑ Background color

Save and Close   Close

Next, we have used another LinearRegression object, namely **lin_reg_2**, to fit our **x_poly** vector to the linear model.

Output:

```
Out[11]:  LinearRegression(copy_X=True,  fit_intercept=True,  n_jobs=None,
normalize=False)
```

**Visualizing the result for Linear regression:**

Now we will visualize the result for Linear regression model as we did in Simple Linear Regression. Below is the code for it:

1. #Visulaizing the result **for** Linear Regression model
2. mtp.scatter(x,y,color="blue")
3. mtp.plot(x,lin_regs.predict(x), color="red")
4. mtp.title("Bluff detection model(Linear Regression)")
5. mtp.xlabel("Position Levels")

6. mtp.ylabel("Salary")
7. mtp.show()

Output:

Bluff detection model(Linear Regression)
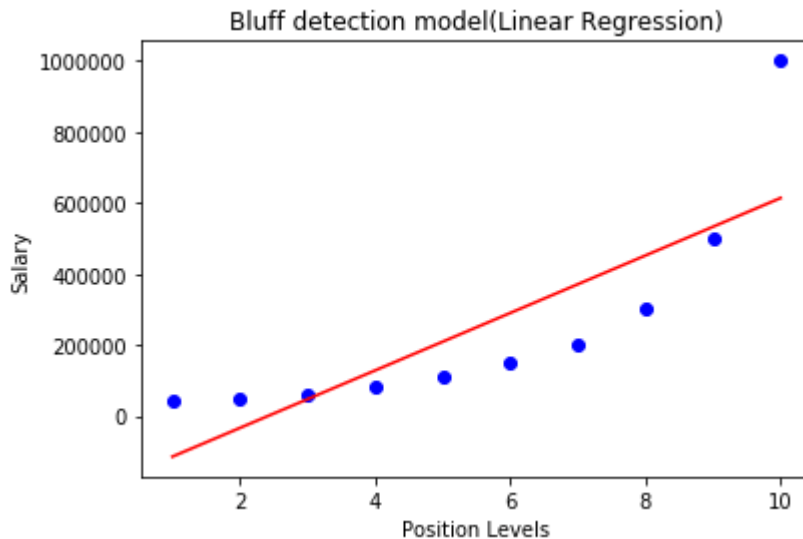
In the above output image, we can clearly see that the regression line is so far from the datasets. Predictions are in a red straight line, and blue points are actual values. If we consider this output to predict the value of CEO, it will give a salary of approx. 600000$, which is far away from the real value.

So we need a curved model to fit the dataset other than a straight line.

**Visualizing the result for Polynomial Regression**

Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.

Code for this is given below:

1. #Visulaizing the result **for** Polynomial Regression
2. mtp.scatter(x,y,color="blue")
3. mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
4. mtp.title("Bluff detection model(Polynomial Regression)")
5. mtp.xlabel("Position Levels")
6. mtp.ylabel("Salary")
7. mtp.show()

In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x), instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.
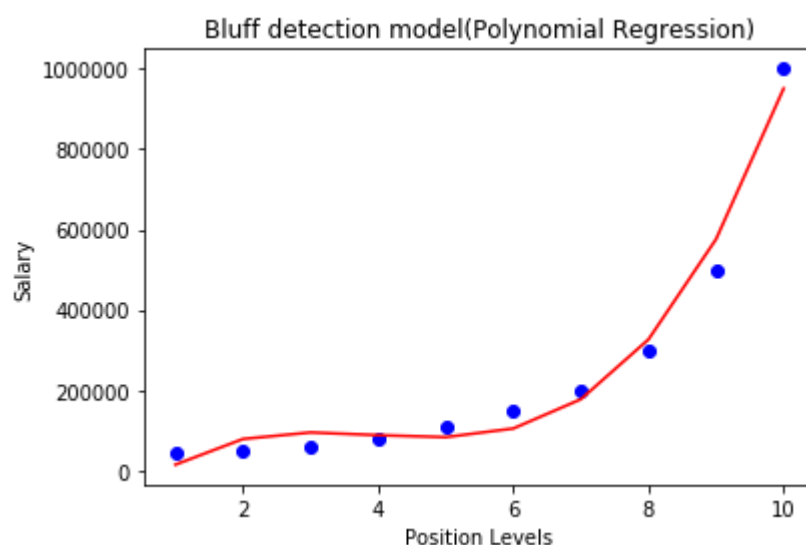
Output:

Bluff detection model(Polynomial Regression)
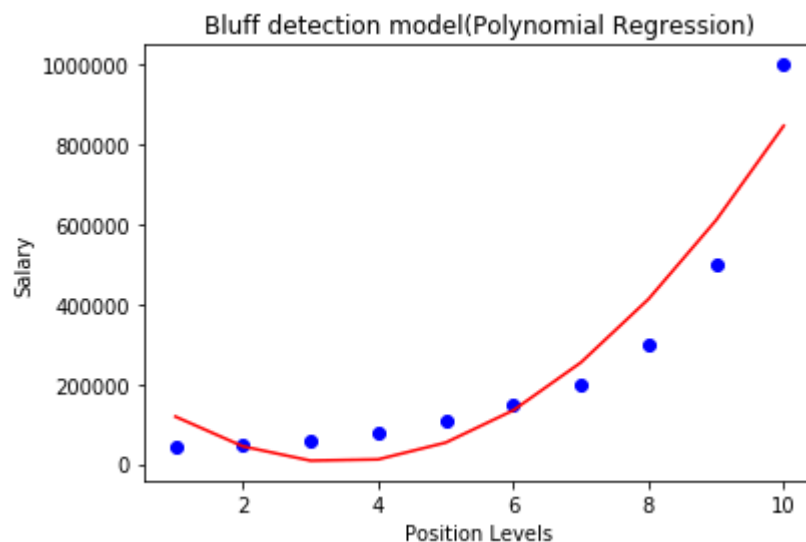
As we can see in the above output image, the predictions are close to the real values. The above plot will vary as we will change the degree.

For degree= 3:

If we change the degree=3, then we will give a more accurate plot, as shown in the below image.

Bluff detection model(Polynomial Regression)

SO as we can see here in the above output image, the predicted salary for level 6.5 is near to 170K$-190k$, which seems that future employee is saying the truth about his salary.

Degree= 4: Let's again change the degree to 4, and now will get the most accurate plot. Hence we can get more accurate results by increasing the degree of Polynomial.
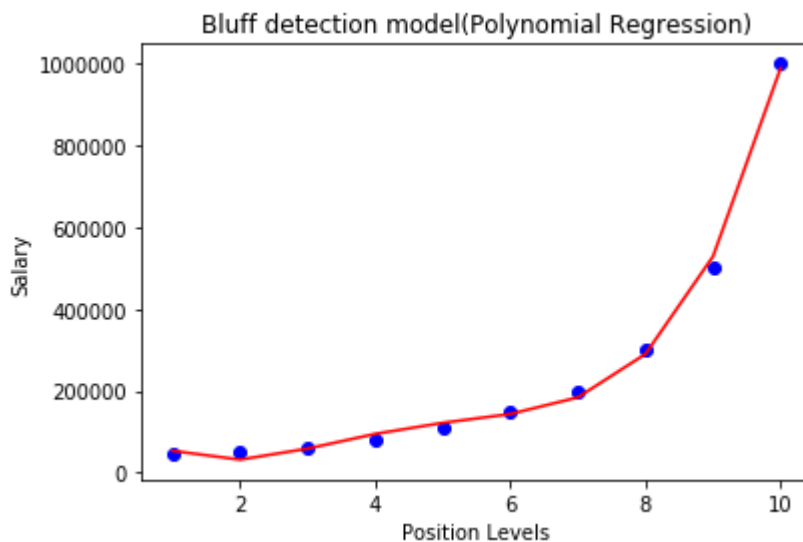
Predicting the final result with the Linear Regression model:

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:

1. lin_pred = lin_regs.predict([[6.5]])
2. print(lin_pred)

Output:

```
[330378.78787879]
```

Predicting the final result with the Polynomial Regression model:

Now, we will predict the final output using the Polynomial Regression model to compare with Linear model. Below is the code for it:

1. poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
2. print(poly_pred)

Output:

```
[158862.45265153]
```

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.