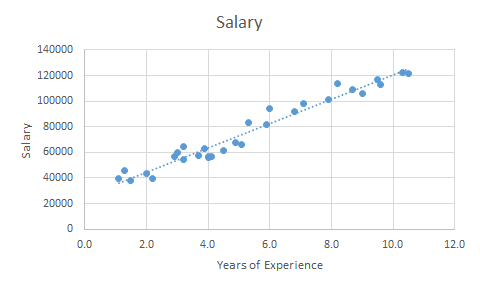
**Machine Learning: Polynomial Regression with Python**

Check out the Linear Regression first

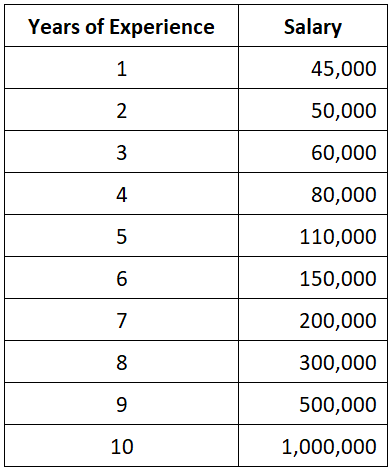
In my previous post, we discussed about [Linear Regression](https://towardsdatascience.com/machine-learning-simple-linear-regression-with-python-f04ecfdadc13). Let’s take a look back. Linear Regression is applied for the data set that their values are **linear**as below example:



Salary based on Years of Experience ([salary\_data.csv](https://s3.us-west-2.amazonaws.com/public.gamelab.fun/dataset/salary_data.csv))

And real life is not that simple, especially when you observe from many different companies in different industries. Salary of 1 YE teacher is different from 1 YE engineer; even 1 YE civil engineer is different from mechanical engineer; and if you compare 2 mechanical engineers from 2 different companies, their salary mostly different as well. So how can we predict the salary of a candidate?

Today, we will use another data set to represent the Polynomial shape.



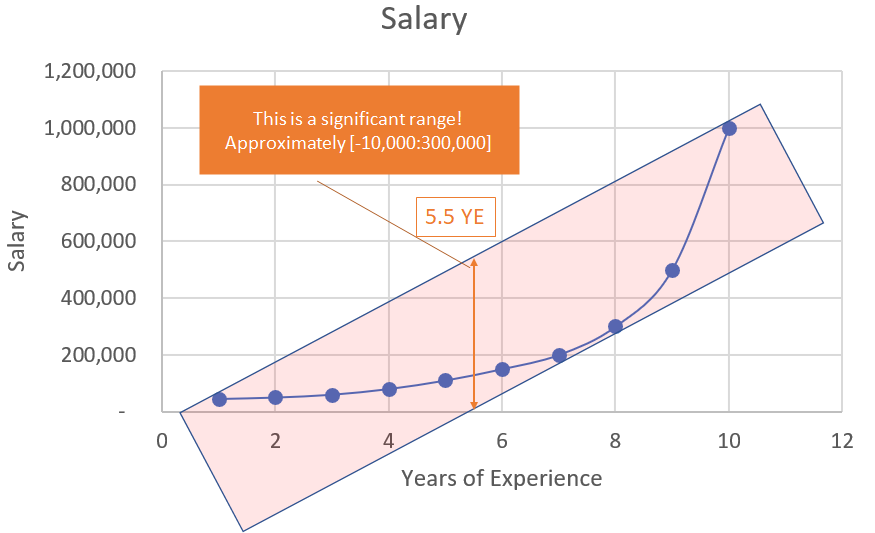
Salary based on Years of Experience ([position\_salaries.csv](https://s3.us-west-2.amazonaws.com/public.gamelab.fun/dataset/position_salaries.csv))

To get an overview of the increment of salary, let’s visualize the data set into a chart:



Salary based on Years of Experience — Plotting

Let’s think about our candidate. He has 5 YE. What if we use the [Linear Regression](https://towardsdatascience.com/machine-learning-simple-linear-regression-with-python-f04ecfdadc13) in this example?



Linear visualization

According to the picture above, the salary range of our candidate could be approximately *from****minus****$10,000 to $300,000*. Why? Look, the salary observations in this scenarios are not linear. **They are in a curved shape!** That’s why applying [Linear Regression](https://towardsdatascience.com/machine-learning-simple-linear-regression-with-python-f04ecfdadc13) in this scenario is not giving you the right value. It’s time for **Polynomial Regression**.

Why Polynomial Regression?

Because it’s much much more accurate!

We are already know the salary of 5 YE is $110,000 and 6 YE is $150,000. It means the salary of 5.5 YE should be between them! And this is how the best value should be:



Polynomial visualization

Let’s compare the gaps between using Linear and Polynomial. Pay attention to the red circle:



Comparison between Linear and Polynomial

It’s too small to see? Zoom it out!



Gaps between Linear and Polynomial

It’s mostly **7.75 times** more accurate than using [Linear Regression](https://towardsdatascience.com/machine-learning-simple-linear-regression-with-python-f04ecfdadc13)!

So how to calculate the salary for our 5.5 YE candidate? We can quick calculate by using the Mean value. Because 5.5 is the average of 5 and 6, so the salary could be calculated as:

*(150,000 + 110,000) / 2 =****$130,000***

*Note: if you don’t know what is Mean value, please read my previous post about*[*Mean, Median, and Mode*](https://medium.com/@nhan.tran/mean-median-an-mode-in-statistics-3359d3774b0b)*. Thanks.*

But it’s not the highest accuracy rate and too manual! Let’s apply the Machine Learning for more accuracy and flexible calculation. Time to start your Spyder IDE!

Polynomial Regression with Python

In this sample, we have to use 4 libraries as numpy, pandas, matplotlib and sklearn. Now we have to import libraries and get the data set first:

|  |
| --- |
|  |
|  | import numpy as np  import matplotlib.pyplot as plt |
|  | import pandas as pd |
|  |  |
|  | # Importing the dataset |
|  | dataset = pd.read\_csv('https://s3.us-west-2.amazonaws.com/public.gamelab.fun/dataset/position\_salaries.csv') |
|  | X = dataset.iloc[:, 1:2].values |
|  | y = dataset.iloc[:, 2].values |

Code explanation:

* dataset: the table contains all values in our csv file
* X: the 2nd column which contains Years Experience array
* y: the last column which contains Salary array

Let’s split our dataset to get training set and testing set (both X and y values per each set)

|  |
| --- |
|  |
|  |
|  |
|  | # 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) |

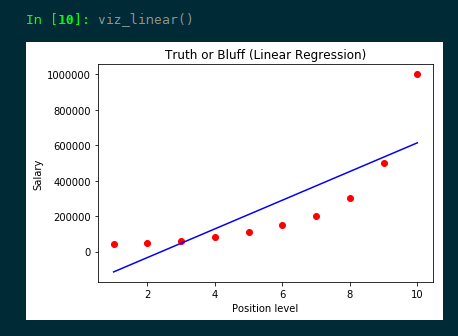
Code explanation:

* test\_size=0.2: we will split our dataset (10 observations) into 2 parts (training set, test set) and the ratio of **test set**compare to dataset is 0.2 (2 observations will be put into the **test set**. You can put it 1/5 to get 20% or 0.2, they are the same. We should not let the test set too big; if it’s too big, we will lack of data to train. Normally, we should pick around 5% to 30%.
* train\_size: if we use the test\_size already, the rest of data will automatically be assigned to train\_size.
* random\_state: this is the seed for the random number generator. We can put an instance of the RandomState class as well. If we leave it blank or 0, the RandomState instance used by np.random will be used instead.

We already have the train set and test set, now we have to build the Regression Model. Firstly, we will build a Linear Regression model and visualize it (it’s no need to include this step in your practice, we just do this for comparison between Linear and Polynomial only):

|  |
| --- |
|  |
|  | # Fitting Linear Regression to the dataset  from sklearn.linear\_model import LinearRegression |
|  | lin\_reg = LinearRegression() |
|  | lin\_reg.fit(X, y) |
|  |  |
|  | # Visualizing the Linear Regression results |
|  | def viz\_linear(): |
|  | 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() |
|  | return |
|  | viz\_linear() |

After calling the viz\_linear() function, you can see a plotting as per below:

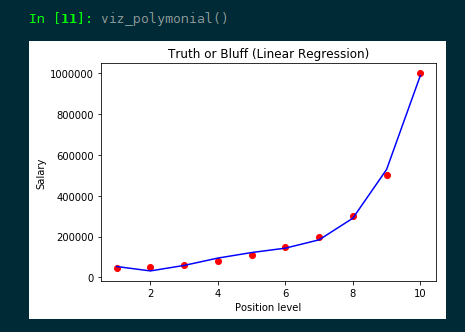


Linear Regression model visualization

In another hand, we will build the Polynomial Regression model and visualize it to see the differences:

|  |
| --- |
|  |
|  |
|  | # Fitting Polynomial Regression to the dataset  from sklearn.preprocessing import PolynomialFeatures |
|  | poly\_reg = PolynomialFeatures(degree=4) |
|  | X\_poly = poly\_reg.fit\_transform(X) |
|  | pol\_reg = LinearRegression() |
|  | pol\_reg.fit(X\_poly, y) |
|  |  |
|  | # Visualizing the Polymonial Regression results |
|  | def viz\_polymonial(): |
|  | plt.scatter(X, y, color='red') |
|  | plt.plot(X, pol\_reg.predict(poly\_reg.fit\_transform(X)), color='blue') |
|  | plt.title('Truth or Bluff (Linear Regression)') |
|  | plt.xlabel('Position level') |
|  | plt.ylabel('Salary') |
|  | plt.show() |
|  | return |
|  | viz\_polymonial() |

After calling the viz\_polynomial() function, you can see a plotting as per below:

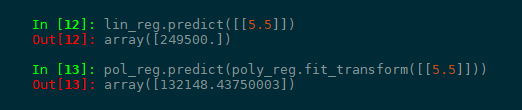


Polynomial Regression model visualization

Last step, let’s predict the value of our candidate (with 5.5 YE) using both Linear Regression model and Polynomial Regression model:

|  |
| --- |
|  |
|  | # Predicting a new result with Linear Regression  lin\_reg.predict([[5.5]]) |
|  | #output should be 249500 |
|  |  |
|  | # Predicting a new result with Polymonial Regression |
|  | pol\_reg.predict(poly\_reg.fit\_transform([[5.5]])) |
|  | #output should be 132148.43750003 |

You can see, the predicted values using Linear Regression model and Polynomial Regression model are totally different!



Comparison between lin\_reg and pol\_reg

Let’s scroll up and check again what we got? According to our data set, our salary should be:

*$110,000 < the salary < $150,000*

But the predicted salary using Linear Regression lin\_reg is **$249,500**. It’s unacceptable *(but still in the range of -10,000 to 300,000 according to Linear Regression)*! What’s about using Polynomial Regression? Our pol\_reg value is **$132,148.43750** which is very close to our **Mean value which is $130,000**.

Bingo! It’s time to let our candidate know we will offer him a best salary in class with $132,148!