Polynomial Regression in Python

November 26, 2021

Topics: Machine Learning

Polynomial regression is a machine learning model used to model non-linear relationships between dependent and independent

Getting Started with Polynomial Regression in Python

Examples of cases where polynomial regression can be used include modeling population growth, the spread of diseases, and epidemics.

Table of contents

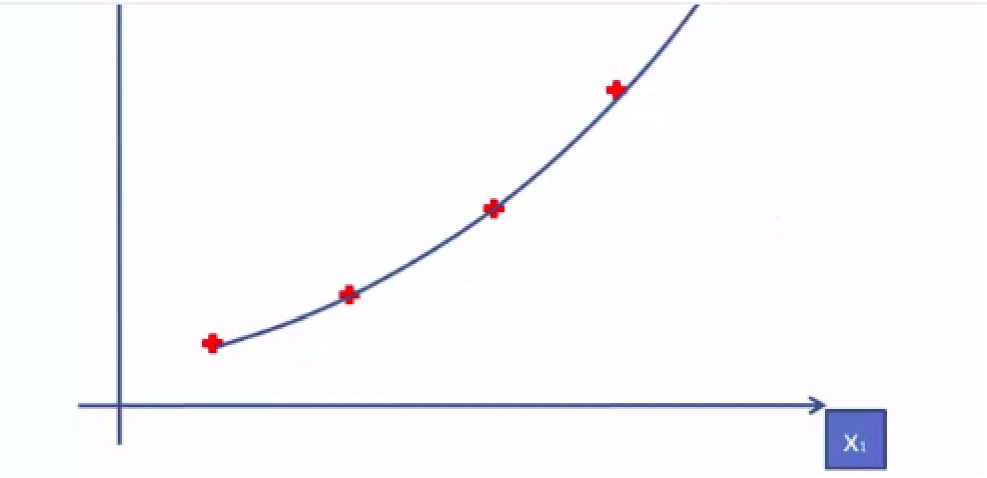
Prerequisites

A general understanding of Python and the Linear Regression Model will be helpful for the reader to follow along.

Suppose we want to predict the value of a real-valued target variable y using real-valued inputs of the observed variable x. Suppose our training set comprises of N training examples, with $\mathbf{x}$ being features variable written $\mathbf{x} \equiv (x_1, \ldots, x_N)^T$, and a corresponding target variable y given as, $\mathbf{y} \equiv (y_1, \ldots, y_N)^T$ with N = 10.

Now, suppose we fit a polynomial curve on this data:

The obtained curve would look like the one in the figure below:



Usually, when fitting a curve, the goal is to exploit the training set and learn the underlying regularities of the data to predict the target variable values on new values of the input variable x. The general task here is to discover the underlying function from which the training data was generated.

To find this function, we need to generalize the finite data of the training set optimally. However, the training set is typically corrupted with random noise. This situation poses a challenge.

value of y we are trying to predict using the input value x.

Polynomial curve fitting

The polynomial function we use to fit the data is of the form:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 +, \ldots, + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Where;

M is the polynomial order.

x^j is input variable x raised to the power j.

$\mathbf{w} = w_0, . \ . \ . , w_M$ denotes a vector of weights.

From the polynomial hypothesis above, we note that this function $y(x, \mathbf{w})$ is a non-linear function of x. However, this function is a linear function of the weights $\mathbf{w}$. Functions that are non-linear in the input variable but linear in

Now the task is to determine the value of $\mathbf{w}$ and M.

Obtaining the $\mathbf{w}$

To determine the values of the coefficients, we first fit a polynomial to the training dataset. After that, we find the error function, i.e., $E(\mathbf{w})$, which measures the misfit between the fitted curve $y(x, \mathbf{w})$ and the data points of the training set.

One of the most used error functions in machine learning is the sum of squares of the errors between the predictions $y(x_n, \mathbf{w})$ and the corresponding target values for each point x_n y_n fin the data.

This error function is of the form:

$$E(\mathbf{w}) = \frac{1}{2} \sum (y(x_n, w) - y_n)^2$$

Where:

y_n is the actual value of the input variable.

The good thing with this error function is that it's non-negative, and it can be zero only and only if the function $y(x, \mathbf{w})$ passes exactly through each point of the training set.

To solve the problem of the curve fitting, we choose the value of $\mathbf{w}$ for which $E(\mathbf{w})$ is minimized as small as possible. Since the error function is quadratic, we can solve it using techniques such as the *Least Squares* or the *gradient descent* optimizer.

We thus obtain the optimal set of the parameters $\mathbf{w}$ for which $E(\mathbf{w})$ is minimized as small as possible. The unique set of solutions for $\mathbf{w}$ we obtain through minimizing the error function is denoted as $\mathbf{w}^\star$. From this, we represent our polynomial function as:

$$y(x, \mathbf{w}^\star)$$

Our only remaining discussion now is how we choose the value of M for our polynomial function. Selecting a value of M is a problem that results in the problem of *over-fitting*.

Hence a poor generalization of the data in both cases. Therefore, as the goal is to achieve good generalization, we need to select a value of M that helps in generalizing the model better. Now that we know what a Polynomial Regression is let's use this knowledge and develop a prediction model.

Implementing the polynomial regression model

The dataset we will use in this tutorial can be obtained from here. Our task in this implementation session is to predict a team member's salary depending on their position level in the company.

To make this session more enjoyable, we shall work along with both the linear regression and the polynomial regression to know when to choose a polynomial regression model over the linear regression model.

Sect

Step 1: Importing the libraries

We import the following libraries:

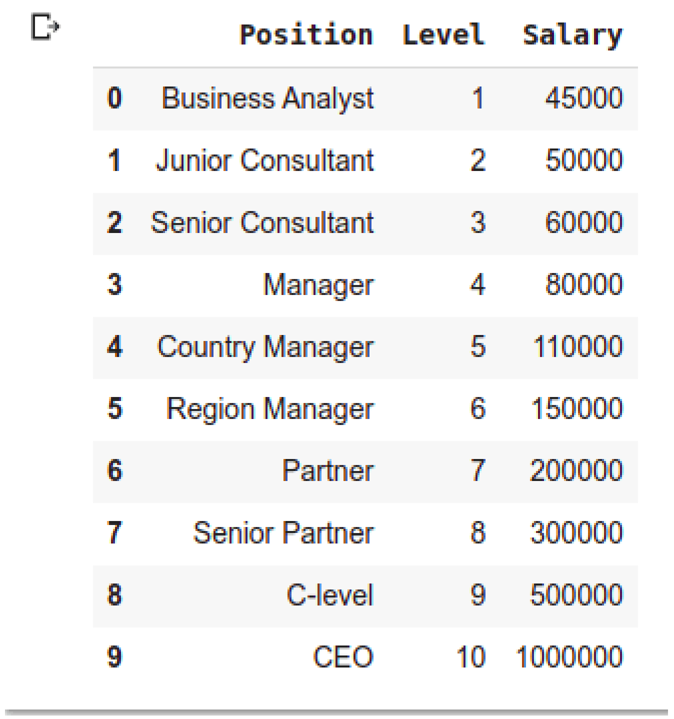
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Step 2: Importing the dataset

In this step, we import the dataset and create a dataframe.

```
dataset = pd.read_csv('/content/drive/MyDrive/Position_Salaries.csv') # reac
dataset
```

This code prints the following dataset.



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

This dataset contains information on how employees of a particular company are paid.

The dataset contains the following information:

Salary: The salary of a team member. For different levels, there are significant differences in the salary. Thus, we need to find a way to generalize the data.

Splitting the data into features and labels

```
X = dataset.iloc[:, 1:-1].values # extracts features from the dataset
y = dataset.iloc[:, -1].values # extracts the labels from the dataset
```

Step 3: Training the linear regression model on the whole dataset

In this step, we will train the linear regression model on the entire dataset. The code below explains how this is done.

```
from sklearn.linear_model import LinearRegression # import the Linear Regres
lin_reg = LinearRegression() # creat model object
```

Output

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fal
```

The output indicates the linear regression model has been trained on the whole dataset.

Step 4: Training the polynomial regression model on the whole dataset

In this step, we will train the polynomial regression model on the whole dataset. The code below explains how this is done.

```
X_poly = poly_regr.fit_transform(X) # transforms the features to the polynom
lin_reg_2 = LinearRegression() # creates a linear regression object
lin_reg_2.fit(X_poly, y) # fits the linear regression object to the polynomi
```

Output

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fal
```

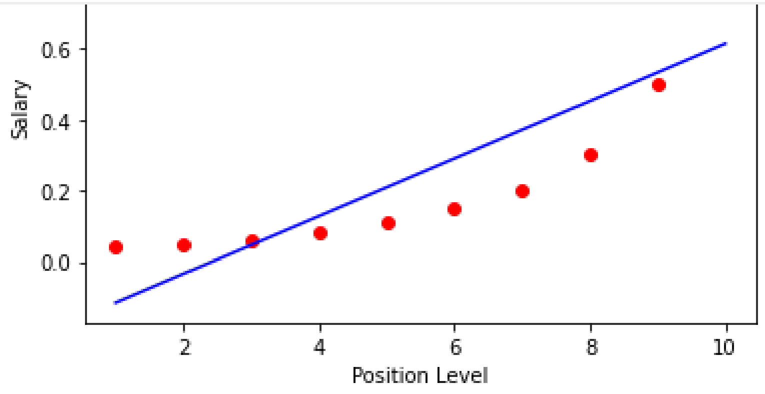
The output indicates the polynomial regression model has been trained on the whole dataset.

Step 5: The visualization of linear regression results

Sect

```
plt.scatter(X, y, color = 'red') # plotting the training set
plt.plot(X, lin_reg.predict(X), color = 'blue') # plotting the linear regres
plt.title('Truth or Bluff (Linear Regression)') # adding a tittle to our plo
plt.xlabel('Position Level') # adds a label to the x-axis
plt.ylabel('Salary') # adds a label to the y-axis
plt.show() # prints our plot
```

The code above plots the data and fit a linear regression model on it, as shown below.



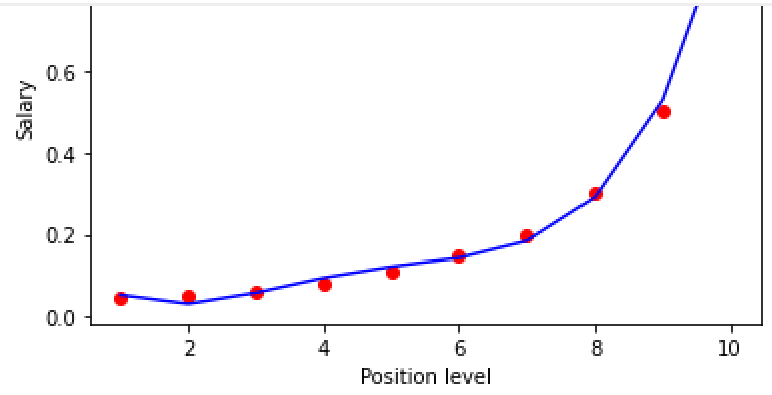
As seen from the plot above, the linear regression model does not fit the data well.

Step 6: The polynomial regression results visualization

To visualize the polynomial regression results, let’s execute the code below.

```
plt.title('Truth or Bluff (Polynomial Regression)') # adding a tittle to our
plt.xlabel('Position level') # adding a label to the x-axis
plt.ylabel('Salary') # adding a label to the y-axis
plt.show() # prints our plot
```

The code above plots the data and fit a polynomial regression model on it, as shown below.

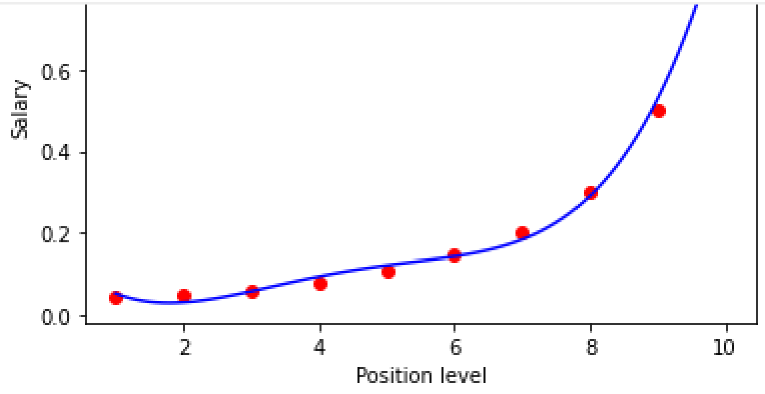


Step 7: The polynomial regression results visualization (for higher resolution and smoother curve)

In this step, we plot the polynomial regression results on a higher resolution (100 points per axis) to get a smoother curve. To visualize the results of this model, let’s execute the following code.

```
plt.scatter(X, y, color = 'red') # plots the training set
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color =
plt.title('Truth or Bluff (Polynomial Regression)') # adds tittle to the plo
plt.xlabel('Position level') # adds label to the x-axis
plt.ylabel('Salary') # adds label to the y-axis
plt.show() # prints our plot
```

The code above yields the plot below.



Step 8: A new result prediction with linear regression

Here, we predict a new output with the linear regression model. Let's execute the code below and see the output.

```
lin_reg.predict([[6.5]]) # predicting a new result with linear regression
```

```
array([330378.78787879])
```

The variable $X = 6.5$ is exact between $X = 6$ and $X = 7$. Thus we expect the model to predict a salary value between 150000 and 200000. With linear regression, this is not the case.

It overshoots the expected salary actually by almost two times. This indicates that linear regression is not suitable for this problem.

Step 9: A new result prediction with polynomial regression

Here, we predict a new output with the polynomial regression model. Let’s execute the code below and see the output.

```
array([158862.45265155])
```

Executing the code above returns a predicted salary as 158862.45265155. This value lies within the range of our expectations, and thus we can conclude the polynomial regression is suitable for this problem.

Conclusion

In this session, we have learned the knowledge behind the polynomial regression. Specifically, we learned how to obtain an optimal set of the parameters from the error function and avoid underfitting and overfitting in polynomial regression.

Then we implemented this model on a real dataset, and we were able to visualize its graph and use it to make predictions. I hope this session has

Happy coding!

Peer Review Contributions by: Lalithnarayan C

Sect

2

0 Comments

Sort By Best

The EngEd community is subject to Section's moderation policy.

Be the first to comment...

LOGIN SIGNUP

Similar Articles

Model Monitoring and Detecting drifts in ML Models using Deepchecks

Read More

Implementing Undersampling, Oversampling, and SMOTE Techniques in Deep Neural Networks

Read More

Multivariate Time Series using Auto ARIMA

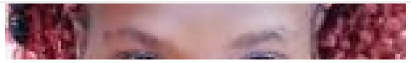
Read More

EngEd Author Bio

Faith Mwangangi

Faith Mwangangi is an undergraduate student undertaking her Bachelor of Science in Computer Science. She is interested in AI. She loves listening to musics as well.

Sect



Join our Slack community

Add to Slack

Company

About

Careers

Legals

Resources

Blog

Case Studies

Content Library

Solution Briefs

Partners

Changelog

Pricing

Support

Docs

Community Slack

Help & Support

Platform Status

Section supports many open source projects including: