Polynomial Regression (Simple & Multiple)

Explanations:

Polynomial Regression (If NLE)

As we know

equation of line : $y = mx + c$

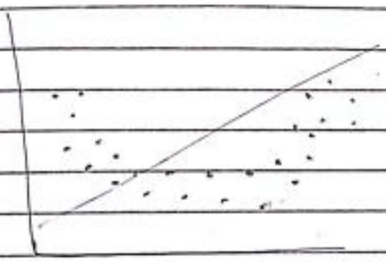
and equation of simple linear Regression

$$y = \alpha_0 + \alpha_1 x$$

=> For multiple linear Regression

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 \text{ —— } \alpha_n x_n$$

This is applicable only when the data is linear but what if data is not linear



In this scenario we extract polynomial feature of input variable in preprocessing stage.

let say e.g. $x \mid y \longrightarrow 2 \mid 6$

- for x we want to make polynomial of degree 2 then we will convert $x \rightarrow x^0, x^1, x^2, y$

its mean $x^0 = 1$, $x^1 = 2$, $x^2 = 4$, $y = 6$

Now our data will be like

x^0	x^1	x^2	y
1	2	4	6

- This way we create a new data for training the extra polynomial feature try to extract this non-linearity relationship.

· Formula of Simple polynomial Regression

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2$$

For degree = 3

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$$

* Now how we will know the perfect value for the degree since this is hyperparameter if we keep it low may be it cause a underfitting and if we keep high then there is a chance of overfitting mean good accuracy in training data but not well perform on testing data so that's why our job is to find out the optimal value

· In case we have two feature (Multiple poly Reg)

x_1 x_2 y

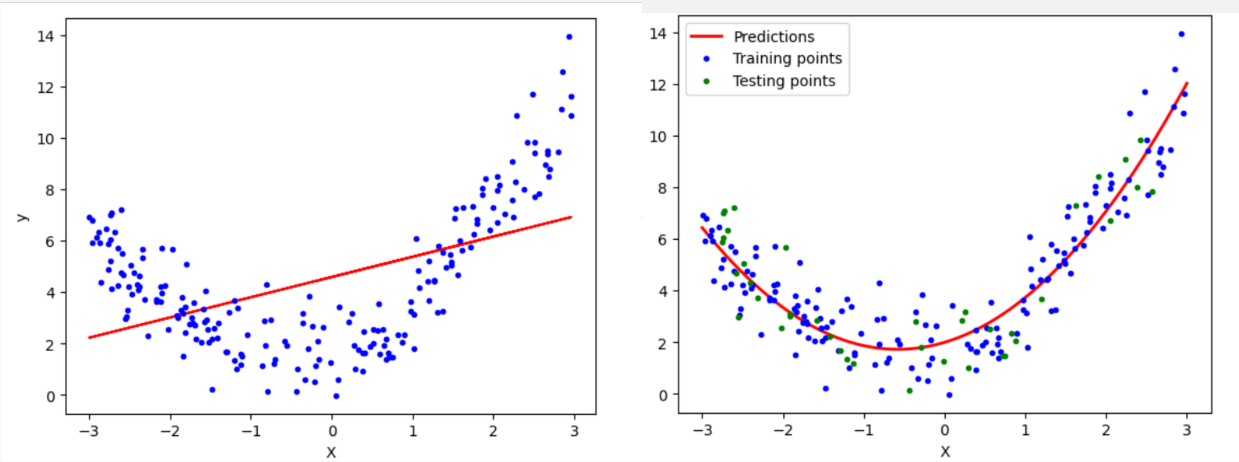
Then for degree 2 our simple polynomial regression would become

$$\Rightarrow y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_1^2 + \alpha_3 x_2 + \alpha_4 x_2^2$$

Question Arise :- Why polynomial is essentially called as Linear Regression.

Ans When we talk about linear Regression we talk about relation b/w y and coefficient of features and you noticed degree of coeff is still one and thus relation b/w y and coefficient is still linear that's we called Polynomial Regression.

Difference Between Linear Regression & Polynomial Regression



Code:

```
import numpy as np

import matplotlib.pyplot as plt


from sklearn.model_selection import train_test_split


from sklearn.linear_model import LinearRegression,SGDRegressor


from sklearn.preprocessing import PolynomialFeatures,StandardScaler


from sklearn.metrics import r2_score


from sklearn.pipeline import Pipeline

X = 6 * np.random.rand(200, 1) - 3

y = 0.8 * X**2 + 0.9 * X + 2 + np.random.randn(200, 1)
```

```
# y = 0.8x^2 + 0.9x + 2
plt.plot(X, y,'b.')
plt.xlabel("X")
plt.ylabel("y")
plt.show()
# Train test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
# Applying linear regression
lr = LinearRegression()
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
r2_score(y_test,y_pred)
plt.plot(X_train,lr.predict(X_train),color='r')
plt.plot(X, y, "b.")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
# Applying Polynomial Linear Regression
# degree 2
poly = PolynomialFeatures(degree=2,include_bias=True)

X_train_trans = poly.fit_transform(X_train)
X_test_trans = poly.transform(X_test)
print(X_train[0])
print(X_train_trans[0])
lr = LinearRegression()
lr.fit(X_train_trans,y_train)
y_pred = lr.predict(X_test_trans)
r2_score(y_test,y_pred)
```

```
print(lr.coef_)
print(lr.intercept_)
X_new=np.linspace(-3, 3, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = lr.predict(X_new_poly)
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.plot(X_train, y_train, "b.",label='Training points')
plt.plot(X_test, y_test, "g.",label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
def Apna_Polynomial(degree):
    X_new=np.linspace(-3, 3, 100).reshape(100, 1)
    X_new_poly = poly.transform(X_new)

    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    Apna_Polynomial = Pipeline([
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ])
    Apna_Polynomial.fit(X, y)
    y_newbig = Apna_Polynomial.predict(X_new)
    plt.plot(X_new, y_newbig,'r', label="Degree " + str(degree), linewidth=2)

    plt.plot(X_train, y_train, "b.", linewidth=3)
```

```
    plt.plot(X_test, y_test, "g.", linewidth=3)
    plt.legend(loc="upper left")
    plt.xlabel("X")
    plt.ylabel("y")
    plt.axis([-3, 3, 0, 10])
    plt.show()
Apna_Polynomial(350)

# 3D polynomial regression
x = 7 * np.random.rand(100, 1) - 2.8
y = 7 * np.random.rand(100, 1) - 2.8

z = x**2 + y**2 + 0.2*x + 0.2*y + 0.1*x*y +2 + np.random.randn(100, 1)
# z = x^2 + y^2 + 0.2x + 0.2y + 0.1xy + 2
import plotly.express as px
df = px.data.iris()
fig = px.scatter_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())
fig.show()
lr = LinearRegression()
lr.fit(np.array([x,y]).reshape(100,2),z)

x_input = np.linspace(x.min(), x.max(), 10)
y_input = np.linspace(y.min(), y.max(), 10)
xGrid, yGrid = np.meshgrid(x_input,y_input)

final = np.vstack((xGrid.ravel().reshape(1,100),yGrid.ravel().reshape(1,100))).T

z_final = lr.predict(final).reshape(10,10)
import plotly.graph_objects as go
```

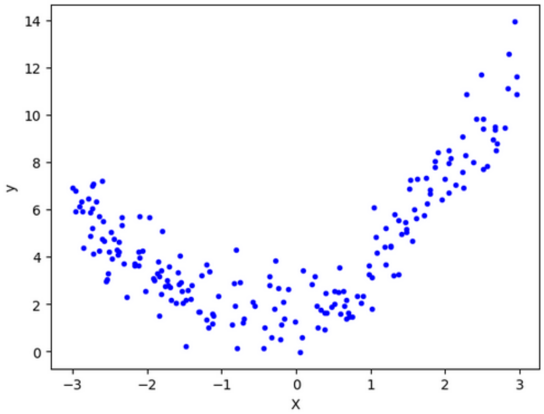
```
fig = px.scatter_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.add_trace(go.Surface(x = x_input, y = y_input, z =z_final ))

fig.show()
X_multi = np.array([x,y]).reshape(100,2)
X_multi.shape
poly = PolynomialFeatures(degree=30)
X_multi_trans = poly.fit_transform(X_multi)
print("Input",poly.n_features_in_)
print("Ouput",poly.n_output_features_)
print("Powers\n",poly.powers_)
X_multi_trans.shape
lr = LinearRegression()
lr.fit(X_multi_trans,z)
X_test_multi = poly.transform(final)
z_final = lr.predict(X_multi_trans).reshape(10,10)
fig = px.scatter_3d(x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.add_trace(go.Surface(x = x_input, y = y_input, z =z_final))

fig.update_layout(scene = dict(zaxis = dict(range=[0,35])))

fig.show()
```

Screenshot:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression,SGDRegressor

from sklearn.preprocessing import PolynomialFeatures,StandardScaler

from sklearn.metrics import r2_score

from sklearn.pipeline import Pipeline
```

In [2]:
```
X = 6 * np.random.rand(200, 1) - 3
y = 0.8 * X**2 + 0.9 * X + 2 + np.random.randn(200, 1)

# y = 0.8x^2 + 0.9x + 2
```

In [3]:
```
plt.plot(X, y,'b.')
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```



In [4]:
```
# Train test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

In [5]:
```
# Applying linear regression
lr = LinearRegression()
```

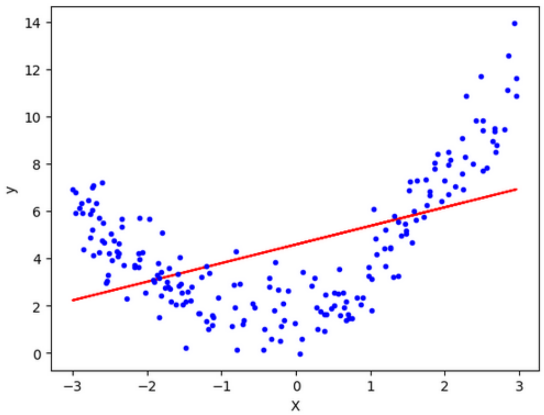
In [6]:
```
lr.fit(X_train,y_train)
```

Out[6]:
```
▾ LinearRegression
LinearRegression()
```

In [7]:
```
y_pred = lr.predict(X_test)
r2_score(y_test,y_pred)
```

Out[7]: -0.08164254561987261

In [8]:
```
plt.plot(X_train,lr.predict(X_train),color='r')
plt.plot(X, y, "b.")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```



```python
In [9]: # Applying Polynomial Linear Regression
        # degree 2
        poly = PolynomialFeatures(degree=2,include_bias=True)

        X_train_trans = poly.fit_transform(X_train)
        X_test_trans = poly.transform(X_test)
```

```python
In [10]: print(X_train[0])
         print(X_train_trans[0])
```

```
[2.25260617]
[1.         2.25260617 5.07423454]
```

```python
In [11]: # include_bias parameter
```

```python
In [12]: lr = LinearRegression()
         lr.fit(X_train_trans,y_train)
```

```
Out[12]: ▾ LinearRegression
         LinearRegression()
```

```python
In [13]: y_pred = lr.predict(X_test_trans)
```

```python
In [14]: r2_score(y_test,y_pred)
```

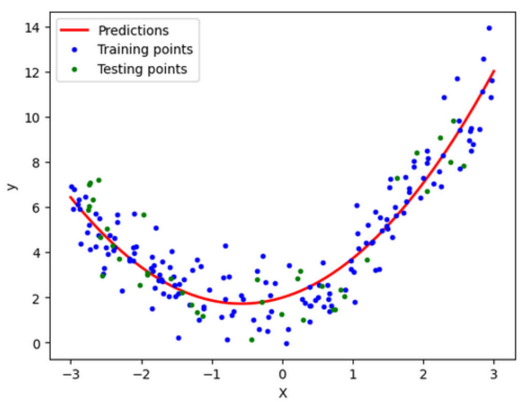
```
Out[14]: 0.8067132798109505
```

```python
In [15]: print(lr.coef_)
         print(lr.intercept_)
```

```
[[0.         0.93128208 0.80366107]]
[1.98613791]
```

```python
In [16]: X_new=np.linspace(-3, 3, 200).reshape(200, 1)
         X_new_poly = poly.transform(X_new)
         y_new = lr.predict(X_new_poly)
```

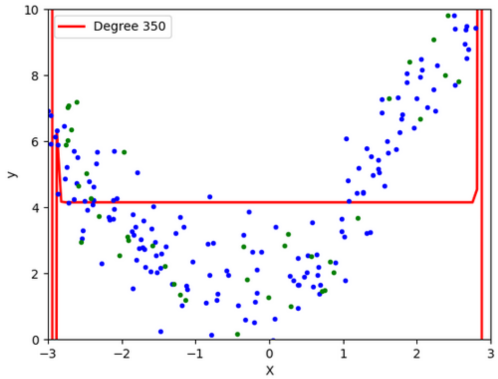
```python
In [17]: plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
         plt.plot(X_train, y_train, "b.",label='Training points')
         plt.plot(X_test, y_test, "g.",label='Testing points')
         plt.xlabel("X")
         plt.ylabel("y")
         plt.legend()
         plt.show()
```




```python
In [18]: def Apna_Polynomial(degree):
             X_new=np.linspace(-3, 3, 100).reshape(100, 1)
             X_new_poly = poly.transform(X_new)

             polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
             std_scaler = StandardScaler()
             lin_reg = LinearRegression()
             Apna_Polynomial = Pipeline([
                     ("poly_features", polybig_features),
                     ("std_scaler", std_scaler),
                     ("lin_reg", lin_reg),
                 ])
             Apna_Polynomial.fit(X, y)
             y_newbig = Apna_Polynomial.predict(X_new)
             plt.plot(X_new, y_newbig,'r', label="Degree " + str(degree), linewidth=2)

             plt.plot(X_train, y_train, "b.", linewidth=3)
             plt.plot(X_test, y_test, "g.", linewidth=3)
             plt.legend(loc="upper left")
             plt.xlabel("X")
             plt.ylabel("y")
             plt.axis([-3, 3, 0, 10])
             plt.show()
```

In [19]:
```
Apna_Polynomial(350)
```

```
C:\python37\Lib\site-packages\sklearn\utils\extmath.py:1066: RuntimeWarning: overflow encountered in square
  temp **= 2
C:\python37\Lib\site-packages\numpy\core\fromnumeric.py:86: RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
```



In [20]:
```
poly.powers_
```

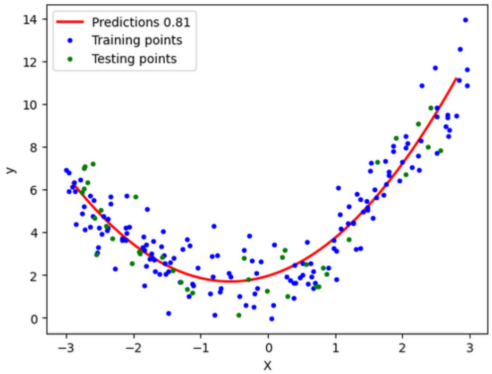
Out[20]:
```
array([[0],
       [1],
       [2]], dtype=int64)
```

In [21]:
```
# Applying Gradient Descent

poly = PolynomialFeatures(degree=2)

X_train_trans = poly.fit_transform(X_train)
X_test_trans = poly.transform(X_test)

sgd = SGDRegressor(max_iter=100)
sgd.fit(X_train_trans,y_train)

X_new=np.linspace(-2.9, 2.8, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = sgd.predict(X_new_poly)

y_pred = sgd.predict(X_test_trans)
```


```
X_new=np.linspace(-2.9, 2.8, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = sgd.predict(X_new_poly)

y_pred = sgd.predict(X_test_trans)
```

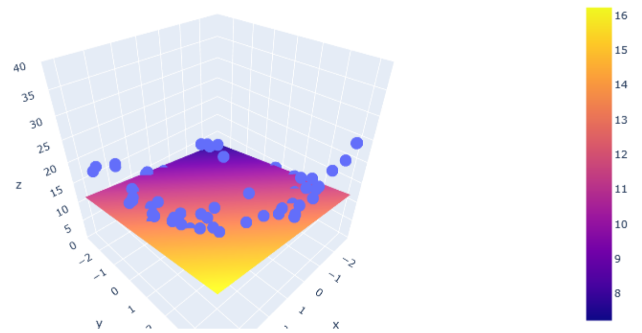

```
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions " + str(round(r2_score(y_test,y_pred),2)))
plt.plot(X_train, y_train, "b.",label='Training points')
plt.plot(X_test, y_test, "g.",label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
C:\python37\Lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



In [22]:
```
# 3D polynomial regression
x = 7 * np.random.rand(100, 1) - 2.8
y = 7 * np.random.rand(100, 1) - 2.8

z = x**2 + y**2 + 0.2*x + 0.2*y + 0.1*x*y +2 + np.random.randn(100, 1)
# z = x^2 + y^2 + 0.2x + 0.2y + 0.1xy + 2
```

In [23]:
```
import plotly.express as px
df = px.data.iris()
fig = px.scatter_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())
fig.show()
```



In [26]:
```
X_multi = np.array([x,y]).reshape(100,2)
X_multi.shape
```
Out[26]:
```
(100, 2)
```

In [27]:
```
poly = PolynomialFeatures(degree=30)
X_multi_trans = poly.fit_transform(X_multi)
```

In [31]:
```
print("Input",poly.n_features_in_)
print("Ouput",poly.n_output_features_)
print("Powers\n",poly.powers_)
```
```
Input 2
Ouput 496
Powers
 [[ 0  0]
 [ 1  0]
 [ 0  1]
 [ 2  0]
 [ 1  1]
 [ 0  2]
 [ 3  0]
 [ 2  1]
 [ 1  2]
 [ 0  3]
```

In [32]:
```
X_multi_trans.shape
```
Out[32]:
```
(100, 496)
```

In [33]:
```
lr = LinearRegression()
lr.fit(X_multi_trans,z)
```
Out[33]:
```
LinearRegression
LinearRegression()
```

In [34]:
```
X_test_multi = poly.transform(final)
```

In [35]:
```
z_final = lr.predict(X_multi_trans).reshape(10,10)
```

In [36]:
```
fig = px.scatter_3d(x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.add_trace(go.Surface(x = x_input, y = y_input, z =z_final))

fig.update_layout(scene = dict(zaxis = dict(range=[0,35])))

fig.show()
```