Generalização de Métodos Lineares

1. Introdução

Foi visto em outros tópicos, a implementação de Regressões Lineares como também de métodos de regularização, mas todo esse campo de modelagem foi feito sobre dados que tinham um certo comportamento linear, ou seja, poderiam ser descritos por uma Regressão Linear. Mas a partir de transformações no conjunto de dados, será extrapolado todos os conceitos desenvolvidos na Regressão Linear para variáveis não-lineares.

2. Variáveis Não-Lineares

Dado a equação para a Regressão Linear Múltipla, definida da forma a seguir:

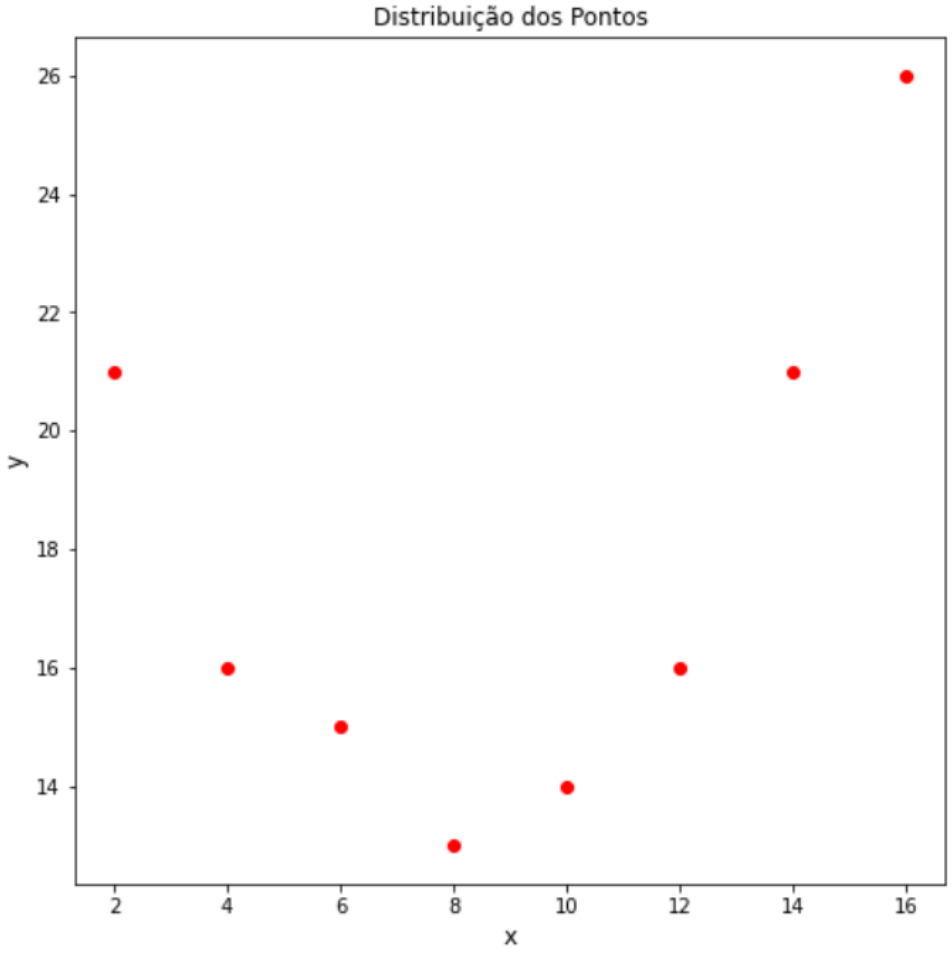
$$Y = \beta_0 X_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

Note que a única **condição de linearidade** necessária para a aplicação da Regressão seria com relação aos **coeficientes**, ou seja todos os coeficientes $ \beta_0, \beta_1, \beta_2, ..., \beta_n $ devem ser lineares entre si. Dessa forma, pode se aplicar uma Regressão Linear independentemente de como é composto os termos de X, mas deve-se realizar uma **transformação** em X para se adequar o modelo. Por exemplo, dado o conjunto de dados a seguir:

```
# conjunto de dados X e Y
x = np.array([2, 4, 6, 8, 10, 12, 14, 16])
y = np.array([21, 16, 15, 13, 14, 16, 21, 26])
```

Ao visualizar a distribuição destes dados, têm-se uma figura construída com o código abaixo:

```
 # Cria a figura e define o tamanho
plt.figure(figsize = (8, 8))

# Cria o gráfico de pontos
plt.plot(x, y, 'ro')

# Adiciona um título
plt.title('Distribuição dos Pontos', fontsize = 12)

# ajusta os eixos do gráfico
plt.ylabel("y", fontsize = 12)
plt.xlabel("x", fontsize = 12)

# Mostra o gráfico
plt.show()
```



Fonte: Let's Code

A figura claramente tem um comportamento **polinomial**, mais especificamente pode ser aproximada a uma **equação quadrática**. Portanto, mantendo os coeficientes da Regressão Linear lineares entre si, pode utilizar a seguinte fórmula para aproximação:

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2$$

O processo de transformação dos dados consiste em substituir o termo x por novas variáveis na forma z_i:

$$z_1 = x$$

$$z_2 = x^2$$

Dessa forma, ao invés de passar apenas o valor de x para a construção do Y, serão necessários passar ambos os parâmetros z_1 e z_2. Com a transformação perceba que a equação utilizada para aproximar recaí na equação geral da Regressão Linear Múltipla, agora em relação a z:

$$Y = \beta_0 + \beta_1 z_1 + \beta_2 z_2$$

Agora, a construção das diversas variáveis z_i pode ser feita a partir de uma função do *Scikit-Learn* chamada de `PolynomialFeatures`, onde de acordo com o grau n do polinômio a ser aproximado, a função calcula todos os termos de potência em relação a variável x. A seguir tem-se a implementação em *Python* do `PolynomialFeatures`:

```
# Carrega a função PolynomialFeatures
from sklearn.preprocessing import PolynomialFeatures

# Define a transformação nos dados
transf = PolynomialFeatures(degree = 2,           # grau do polinômio
                            include_bias = False) # se deseja adicionar um
termo de vies na transformação

# Reshape dos dados x
x = x.reshape((-1, 1)) # importante o reshape, pois transforma-se os dados em
diversas colunas

# Ajusta os dados em relação ao grau do polinômio
transf.fit(x)
```

```
# Transforma os dados incluindo uma nova coluna com valores quadráticos
trans_x = transf.transform(x)

# Print dos dados originais
print("Dados Originais X: \n", x)

# Print dos dados transformados
print("Dados Transformados Z: \n", trans_x)
```

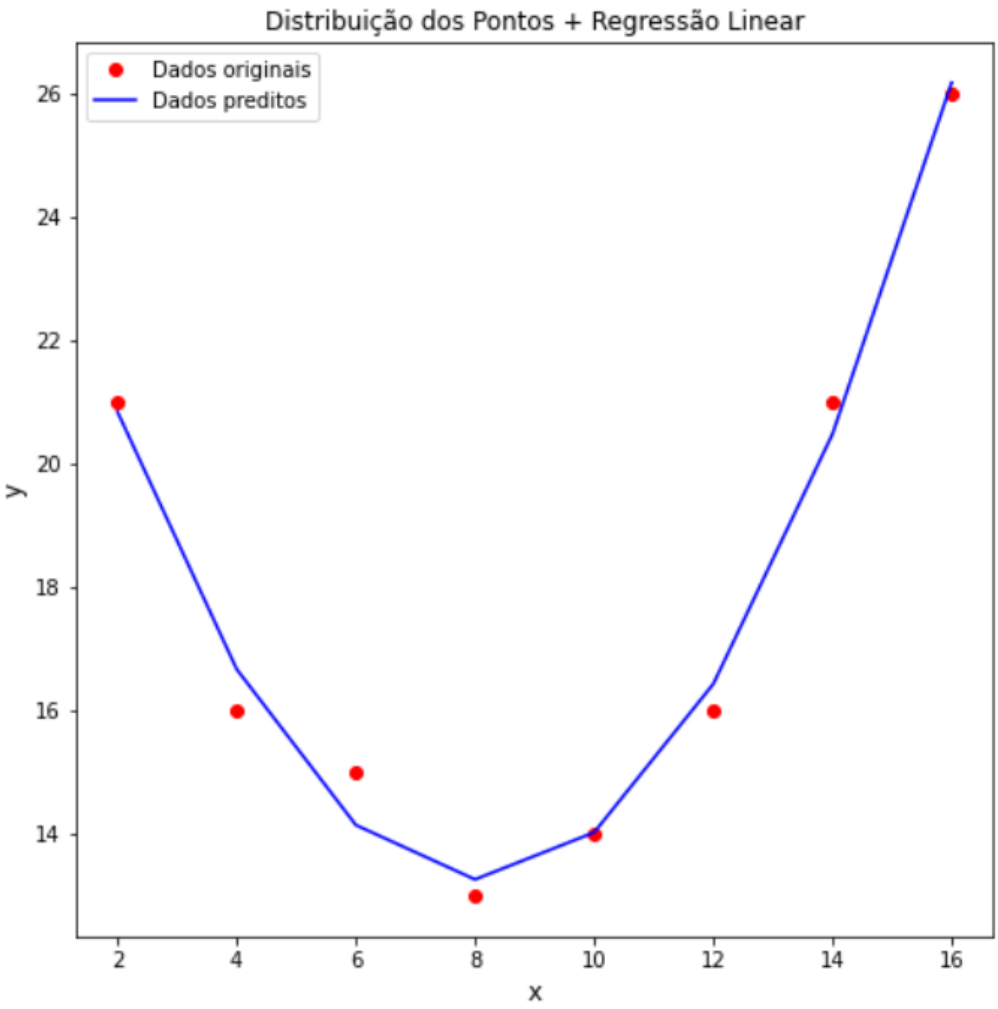
Após a construção dos dados transformados, basta aplicar a Regressão Linear Múltipla como em modelos anteriores:

```
# Carrega a função da Regressão Linear
from sklearn.linear_model import LinearRegression

# Instância o Modelo
model = LinearRegression()

# Treina o modelo
model.fit(trans_x)

# Gera as predições
y_pred = model.predict(trans_x)
```

Dessa forma, consegue se levantar como foi o ajuste da curva em relação ao modelo partindo de variáveis não-lineares:

```
# Cria a figura e define o tamanho
plt.figure(figsize = (8, 8))

# Cria os pontos dos dados originais
plt.plot(x, y, 'ro', label = 'Dados originais')

# Cria a curva oriunda da regressão linear
plt.plot(x, y_pred, 'b-', label = 'Dados preditos')

# Cria um título
plt.title('Distribuição dos Pontos + Regressão Linear')

# Ajusta os eixos
plt.ylabel("y", fontsize=12)
plt.xlabel("x", fontsize=12)
```

```
# Define uma legenda
plt.legend()

# Mostra o gráfico
plt.show()
```



Fonte: Let´s Code

Materiais Complementares

Documentação no Scikit-Learn sobre o PolynomialFeatures;

Referências

James, Gareth, et al. An Introduction to Statistical Learning: With Applications in R. Alemanha, Springer New York, 2013;

Bruce A., Bruce P. Estatística Prática para Cientistas de Dados. Segunda Edição, Alta books, 2019;

Tópico anterior

Próximo Tópico