Regressão Linear

Prática 03: Regressão de Cume

Profª Deborah Magalhães
Monitor: Davi Luis de Oliveira

Universidade
Federal do Piauí

Olá!

Curso: Bacharelado em Sistema de Informação

Disciplina: Sistemas Inteligentes

- **Predição com Regressão de Cume**

Você pode me encontrar em **deborah.vm@gmail.com**
(Dúvidas e sugestões serão bem-vindas =D)

Passo 1: Importar as bibliotecas

Prática 02: Regressão de Cume

1. Importando as bibliotecas necessárias

```
import graphlab
import random
import math
import numpy
from matplotlib import pyplot as plt
%matplotlib inline
```

Passo 2: Gerando os dados sinteticamente

2. Gerando artificialmente os dados

```
random.seed(98103)
n = 30
x = graphlab.SArray([random.random() for i in range(n)]).sort()
```

```
x
```

```
dtype: float
Rows: 30
[0.03957894495006575, 0.04156809967912256, 0.0724319480800758, 0.1502
890446221763, 0.16133414450223427, 0.19195631279497838, 0.23283391714
465285, 0.25990098016580054, 0.3801458148686865, 0.432444723507992, 0
.47056698189428126, 0.4714946037956341, 0.47870640066103853, 0.490535
53924712967, 0.5467800590828905, 0.5696803579782542, 0.60793245364620
45, 0.6202375373443129, 0.630093133764472, 0.6450096693254694, 0.6467
576040906915, 0.6990897790220533, 0.7902450464374043, 0.8103846511814
395, 0.829320894073608, 0.8501115576007019, 0.8863684369527574, 0.891
```

Passo 2: Gerando os dados sinteticamente

```
y = x.apply(lambda x: math.sin(4*x))
```

```
y
```

```
dtype: float
Rows: 30
[0.15765527330715118, 0.16550731513895361, 0.28569137317201587, 0.565
5963310738573, 0.6014673638641537, 0.6945723182799316, 0.802417733662
6097, 0.8622036562183355, 0.9987395887969133, 0.9873888679370947, 0.9
518836143289069, 0.9507399491702666, 0.9414033048470645, 0.9243965533
386744, 0.8160088904398481, 0.759712729130455, 0.6517297472208435, 0.
6136242987242347, 0.5820277559882728, 0.5325021703779891, 0.526571194
9509843, 0.3384164444332101, -0.019386317630007324, -0.09977963765619
023, -0.17478846590569042, -0.25597249131754973, -0.3929901249560022,
-0.4104720422761508, -0.4581930879372578, -0.5280908755021222]
```

Passo 2: Gerando os dados sinteticamente

Adicionando aos dados um ruído gaussiano

```
random.seed(1)
e = graphlab.SArray([random.gauss(0,1.0/3.0) for i in range(n)])
y = y + e
```

Passo 3: Criar um SFrame

3. Dispor os dados gerados em formato de SFrame

```
data = graphlab.SFrame({'X1':x,'Y':y})
```

```
data
```

X1	Y
0.0395789449501	0.587050191026
0.0415680996791	0.648655851372
0.0724319480801	0.307803309485
0.150289044622	0.310748447417
0.161334144502	0.237409625496
0.191956312795	0.705017157224
0.232833917145	0.461716676992
0.259900980166	0.383260507851

Passo 4: Definir as características do modelo de regressão

4. Definir uma função para criar as características do modelo de regressão polinomial de qualquer grau

```
def polynomial_features(data, deg):
    data_copy=data.copy()
    for i in range(1,deg):
        data_copy['X'+str(i+1)]=data_copy['X'+str(i)]*data_copy['X1']
    return data_copy
```

Passo 5: Definir uma função para ajustar um modelo de regressão linear polinomial do grau "deg" aos dados "data"

```
def regressao_polinomial_cume(data, deg, l2_penalty):

        model =
graphlab.linear_regression.create(polynomial_features(data,deg),

                                  target='Y', l2_penalty=l2_penalty,

                                  validation_set=None,verbose=False)

        return model
```

Passo 6: Imprimir os coeficientes do modelo

6. Definir a função que imprime os coeficientes do modelo

```
def print_coefficients(model):

    deg = len(model.coefficients['value'])-1
    w = list(model.coefficients['value'])

    print 'Coeficientes do polinômio de grau ' + str(deg) + ':'
    w.reverse()
    print numpy.poly1d(w)
```

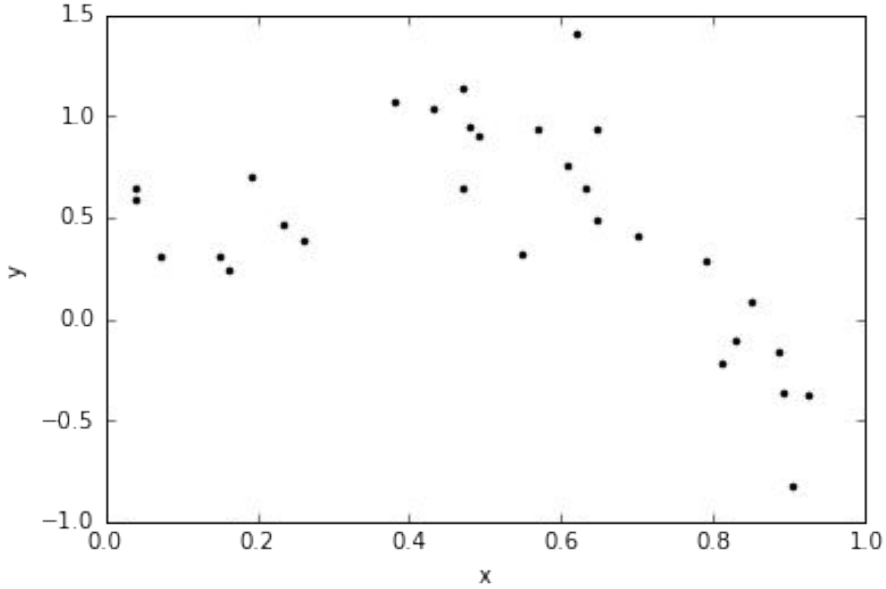
Passo 7: Plotar dados e previsões

7. Defina a função para plotar dados e previsões

```
def plot_data(data):
    plt.plot(data['X1'],data['Y'],'k.')
    plt.xlabel('x')
    plt.ylabel('y')

plot_data(data)
```

Passo 7: Plotar dados e previsões



Passo 7: Plotar dados e previsões

```
def plot_predicoes(data, model):
        plot_data(data)

        deg = len(model.coefficients['value'])-1

        x_pred = graphlab.SFrame({'X1':[i/200.0 for i in range(200)]})
        y_pred = model.predict(polynomial_features(x_pred,deg))

        plt.plot(x_pred['X1'], y_pred, 'g-', label='degree ' + str(deg) + ' fit')
        plt.legend(loc='upper left')
        plt.axis([0,1,-1.5,2])
```

Passo 8: modelo de regressão (grau 16) com lambda pequeno

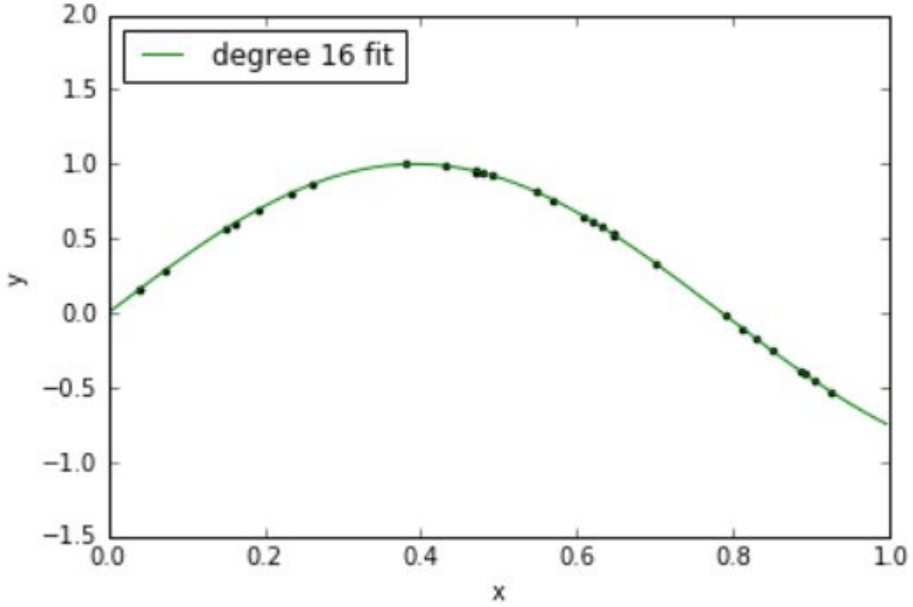
8. Encontre um modelo de regressão de cume utilizando um polinômio de grau 16 usando com um parâmetro de ajuste muito pequeno

```
model = regressao_polinomial_cume(data, deg=16, l2_penalty=1e-25)
print_coefficients(model)
```

```
Coeficientes do polinômio de grau 16:
             16             15             14             13
-4.537e+05 x  + 1.129e+06 x  + 4.821e+05 x  - 3.81e+06 x
              12             11             10             9
 + 3.536e+06 x  + 5.753e+04 x  - 1.796e+06 x  + 2.178e+06 x
              8             7             6             5             4
 - 3.662e+06 x + 4.442e+06 x - 3.13e+06 x + 1.317e+06 x - 3.356e+05 x
             3        2
 + 5.06e+04 x - 4183 x + 160.8 x - 1.621
```

Passo 8: modelo de regressão (grau 16) com lambda pequeno

```
plot_predicoes(data,model)
```



Passo 9: modelo de regressão (grau 16) com lambda grande

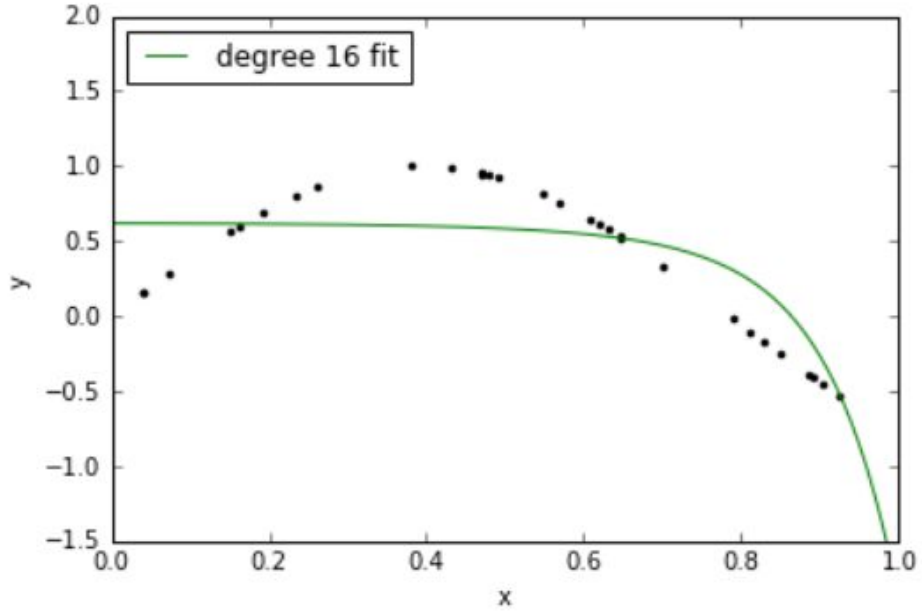
9. Encontre um modelo de regressão de cume utilizando um polinômio de grau 16 usando com um parâmetro de ajuste muito grande

```
model = regressao_polinomial_cume(data, deg=16, l2_penalty=100)
print_coefficients(model)
```

```
Coeficientes do polinômio de grau 16:
        16          15          14          13          12          11
-0.301 x  - 0.2802 x  - 0.2604 x  - 0.2413 x  - 0.2229 x  - 0.205 x
           10          9          8          7          6          5
 - 0.1874 x  - 0.1699 x - 0.1524 x - 0.1344 x - 0.1156 x - 0.09534 x
            4          3          2
 - 0.07304 x - 0.04842 x - 0.02284 x - 0.002257 x + 0.6416
```

Passo 9: modelo de regressão (grau 16) com lambda grande

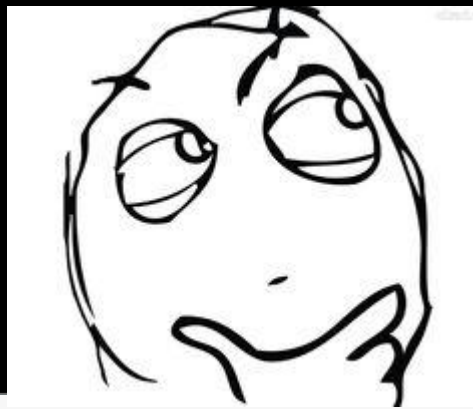
```
plot_predicoes(data,model)
```

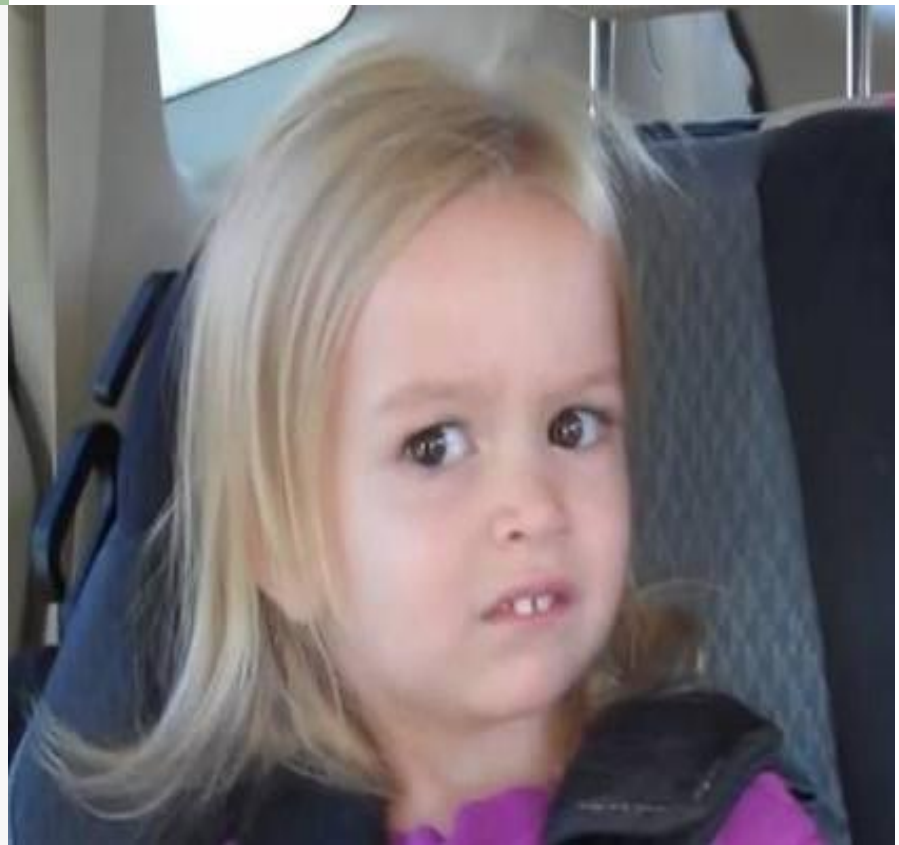


Passo 10: Plotar dados e previsões

```
for l2_penalty in [1e-25, 1e-10, 1e-6, 1e-3, 1e2]:
        model = polynomial_ridge_regression(data, deg=16, l2_penalty=l2_penalty)
        print 'lambda = %.2e' % l2_penalty
        print_coefficients(model)
        print '\n'
        plt.figure()
        plot_predicoes(data,model)
        plt.title('Ridge, lambda = %.2e' % l2_penalty)
```

Qual o melhor coeficiente?





Dúvidas?Sugestões? Inquietações? Aconselhamentos?

- Desabafe em: **deborah.vm@gmail.com**