Lab 1 - BCC406/PCC177

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

Regressão Linear

Prof. Eduardo e Prof. Pedro

Data da entrega : 30/03

- Complete o código (marcado com ToDo) e quando requisitado, escreva textos diretamente nos notebooks.
- Execute todo notebook e salve tudo em um PDF nomeado como "NomeSobrenome-Lab1.pdf"
- Envie PELO FORM

Regressão e Descida do Gradiente

Neste estudo dirigido, resolveremos um problema de regressão, usando algoritmos da descida do gradiente para otimização dos pesos. Vamos implementar a descida do gradiente tradicional e uma versão estocástica com mini-lotes.

Vamos aplicar em um problema de predição de preços de casas (**California Housing Dataset**)

Bibliotecas e carregando os dados

Importando as bibliotecas

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error

np.random.seed(406177)
```

A biblioteca Scikit-learn é focada em aprendizagem de máquina e fornece diversos métodos de

classificação, extração de características, etc. Ela também fornece algumas bases de dados clássicas, por meio de objetos do Pandas.

Se você não conhece o pacote Pandas, veja este curso rápido.

Recuperando os dados

```
housing_data = fetch_california_housing()


Features = pd.DataFrame(housing_data.data, columns=housing_data.feature_names)
Target = pd.DataFrame(housing_data.target, columns=['Target'])


df = Features.join(Target)
df
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Lc
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	
...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	

20640 rows × 9 columns

Entendendo e pré-processando os dados (1 ponto)

Entendendo os dados

[ ] ↳ *2 células ocultas*

Pre-processamento : removendo outlier e ajustando escala

Perceba que em 75% dos dados, a renda média (MedInc) é menor que 5 e que o valor da casa (Target) é menor que 3. Vamos remover rendas maiores que 5 e casas com preço maior que 3, para evitar valores espúrios e outliers.

```
print(df.shape)
df = df[df.MedInc < 5]
df = df[df.Target < 3]
df
```

```
(20640, 9)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Lc
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	
7	3.1200	52.0	4.797527	1.061824	1157.0	1.788253	37.84	
8	2.0804	42.0	4.294118	1.117647	1206.0	2.026891	37.84	
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	
...	...	...	...	...	...	...	...	
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	

14653 rows × 9 columns

Normalização - Também vamos deixar as duas variáveis na faixa entre 0 e 1

```
def scale(x):
    min = x.min()
    max = x.max()
    return pd.Series([(i - min)/(max - min) for i in x])

X = scale(df.MedInc)
y = scale(df.Target)
```

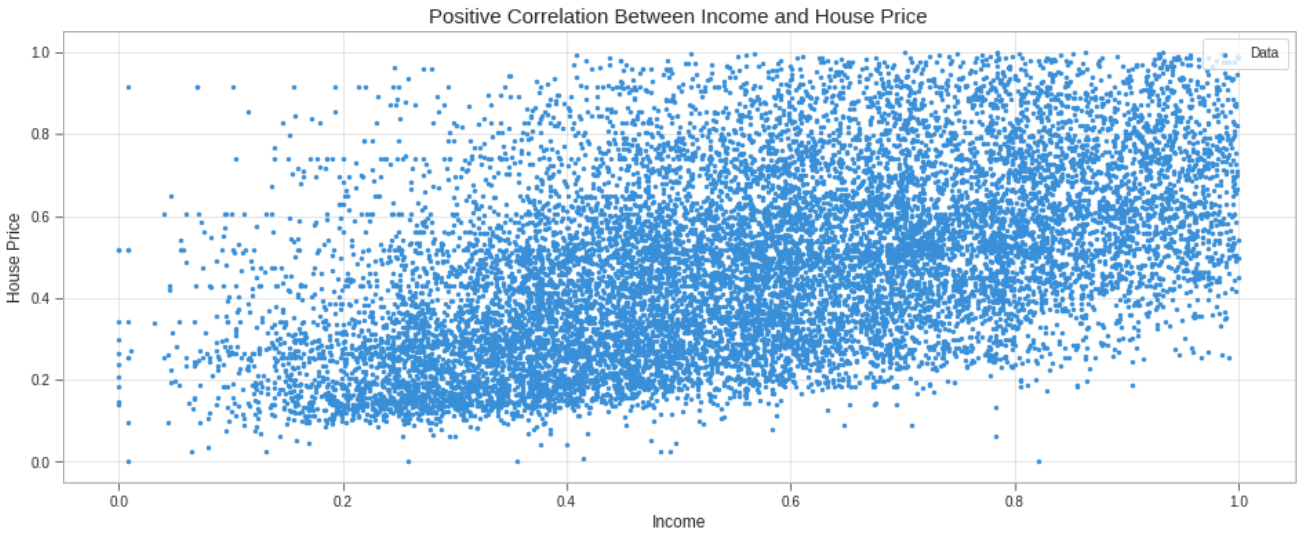
```
#conferindo o valor maximo
X.max(), y.max(), X.size, type(X)
```

```
(1.0, 1.0, 14653, pandas.core.series.Series)
```

Valor esperado: (1.0, 1.0)

Plotando os dados

```
plt.figure(figsize=(16,6))
plt.rcParams['figure.dpi'] = 227
plt.style.use('seaborn-whitegrid')
plt.scatter(X, y, label='Data', c='#388fd8', s=6)
plt.title('Positive Correlation Between Income and House Price', fontSize=15)
plt.xlabel('Income', fontSize=12)
plt.ylabel('House Price', fontSize=12)
plt.legend(frameon=True, loc=1, fontsize=10, borderpad=.6)
plt.tick_params(direction='out', length=6, color='#a0a0a0', width=1, grid_alpha=.6)
plt.show()
```



[1pt] Discussão : Por que os dados devem ser normalizados entre 0 e 1 ?

ToDo : Discorra sobre a questão acima.

Todos os dados usados pelo modelo, tanto de entrada (features) quanto rotulos (para casos de regressão como o exemplo dado) devem ser normalizados, isso é, reposicionados entre 0 e 1 ou deslocados para ter média 0 e variância unitária. Se não forem normalizados, os valores muito grandes ou muito pequenos de variáveis irão influenciar (de forma errada) o modelo nos cáluclos da predição.

Descida do Gradiente (9 pontos)

[4pt] Descida do Gradiente

Vamos resolver uma regressão linear, de uma única característica - MedInc - (vetor de pesos tem uma dimensão), do tipo $Y = mX + b$, e usar erro quadrático médio (mean squared error) como funçào objetivo para a descida do gradiente.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

considerando y chapéu como saída do nosso modelo e y como o rótulo. Expandindo a função é :

$$f(m, b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Derivada em relação a m :

$$\frac{\partial f}{\partial m} = \frac{1}{n}\sum_{i=1}^{n} -2x_i(y_i - (mx_i + b))$$

Derivada em relação a b :

$$\frac{\partial f}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} -2(y_i - (mx_i + b))$$

Clique duas vezes (ou pressione "Enter") para editar

[4pt] Função da descida do gradiente e para plotar a regressão

Vamos implementar uma função chamada *gradient_descent*, seguindo alguns passos:

1. Inicializar m e b aleatoriamente (entre 0 e 1)
2. Iterar por um número de épocas (*epoch*)
3. A cada iteração, calcular o valor predito, o erro quadrático entre o valor predito e y, atualizar os valores de m e b no sentido contrário do gradiente (o ajuste deve ser controlado por um taxa de aprendizado - *Learning Rate (lr)*).
4. Armazenar m, b e erro corrente para análise futura

```
def gradient_descent(X, y, lr=0.05, epoch=10):

    '''
    Descida do Gradiente
    '''

    m, b = np.random.rand(2) # ToDo : inicialize aleatoriamente entre 0 e 1

    log, mse_log = [], [] # listas para armazenar o processo de aprendizado
    N = len(X) # número de instâncias total do conjunto

    for e in range(epoch):

        predict = m*X + b # ToDo : propague (feed-forward) ara obter as predições :
        MSE = 1/N*sum(((predict - y)**2))  # ToDo : calcule o erro quadrático médio

        f_dm = 1/N*sum(-2*X*(y - (m*X + b))) # ToDo : compute a derivada: derivada
        f_db = 1/N*sum(-2*(y - (m*X + b))) #derivada parcial em relação a 'b'


        # atualize m e b
        m -=  lr*f_dm # ToDo : atualize m com base na equação acima. Lembre-se de p
        b -=  lr*f_db # ToDo : atualize b com base na equação acima. Lembre-se de p
```

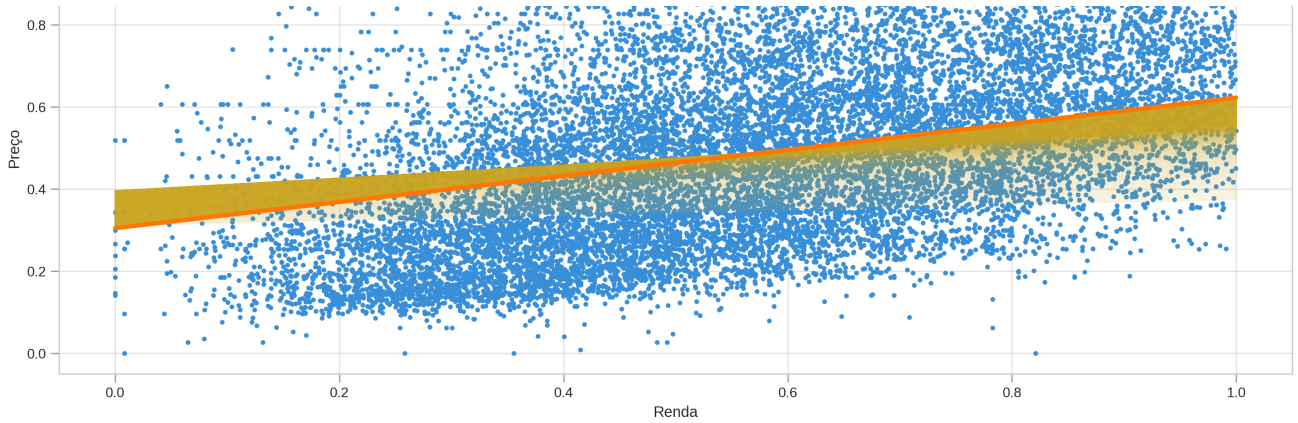
```
        # armazena para uso futuro
        log.append((m, b))
        mse_log.append(MSE)

    return m, b, log, mse_log


def plot_regression(X, y, y_pred, log=None, title="Linear Regression"):

    plt.figure(figsize=(16,6))
    plt.rcParams['figure.dpi'] = 227
    plt.scatter(X, y, label='Data', c='#388fd8', s=6)
    if log != None:
        for i in range(len(log)):
            plt.plot(X, log[i][0]*X + log[i][1], lw=1, c='#caa727', alpha=0.15)
    plt.plot(X, y_pred, c='#ff7702', lw=3, label='Regressao')
    plt.title(title, fontSize=14)
    plt.xlabel('Renda', fontSize=11)
    plt.ylabel('Preço', fontSize=11)
    plt.legend(frameon=True, loc=1, fontsize=10, borderpad=.6)
    plt.tick_params(direction='out', length=6, color='#a0a0a0', width=1, grid_alpha
    plt.show()
```

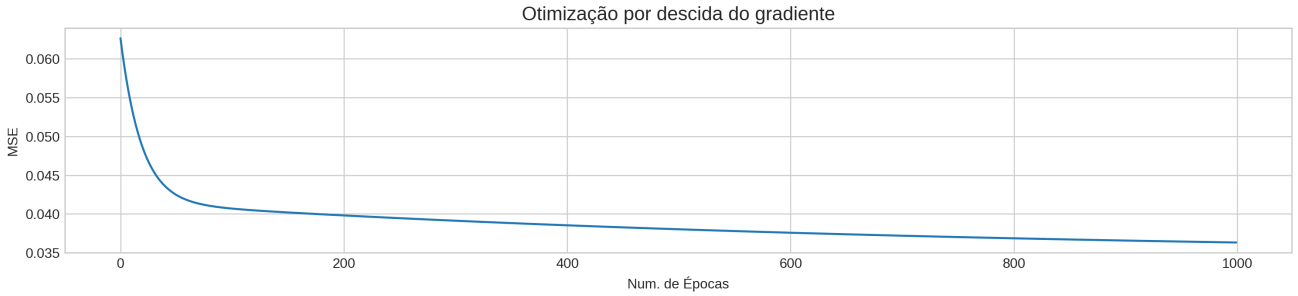
Teste da função

Use a função gradient_descent, para o conjunto de dados de casas (X e y) carregados acima, com uma taxa de aprendizado de 0.01 por 1000 épocas. Analise as curvas plotadas.

```
m, b, log1, mse1 = gradient_descent(X, y, lr=0.01, epoch=1000)

y_pred = m*X + b

print("MSE:",mean_squared_error(y, y_pred))
plot_regression(X, y, y_pred, log=log1, title="Regressao com Descida do Gradiente")

plt.figure(figsize=(16,3))
plt.rcParams['figure.dpi'] = 227
plt.plot(range(len(mse1)), mse1)
plt.title('Otimização por descida do gradiente', fontSize=14)
plt.xlabel('Num. de Épocas')
plt.ylabel('MSE')
plt.show()
```

```
MSE: 0.036309263905409506
```





Valor esperado: MSE: 0.034740339862818236

[4pt] Descida do Gradiente Estocástica

A descida do gradiente estocástica (do inglês Stochastic Gradient Descent - SGD) tem este nome por que ela não é realizada no conjunto de dados inteiro, mas em uma sub-amostragem do conjunto de dados. Esta sub-amostragem é aleatória. Na SGD, o gradiente é aplicado a um sub-conjunto dos dados (um lote ou mini-lote).

A função deve seguir:

1. Inicializar m e b aleatoriamente (entre 0 e 1)
2. Iterar por um número de épocas
3. A cada iteração, amostrar um mini-lote (sub-conjunto) de X (*batch_size*), calcular o valor predito para o mini-lote, calcular o erro quadrático para o mini-lote (entre valor predito e y), atualizar os valores de m e b, no sentido contrário do gradiente (o ajuste deve ser controlado por um taxa de aprendizado - *lr*).
4. Armazenar m, b e erro corrente para análise futura

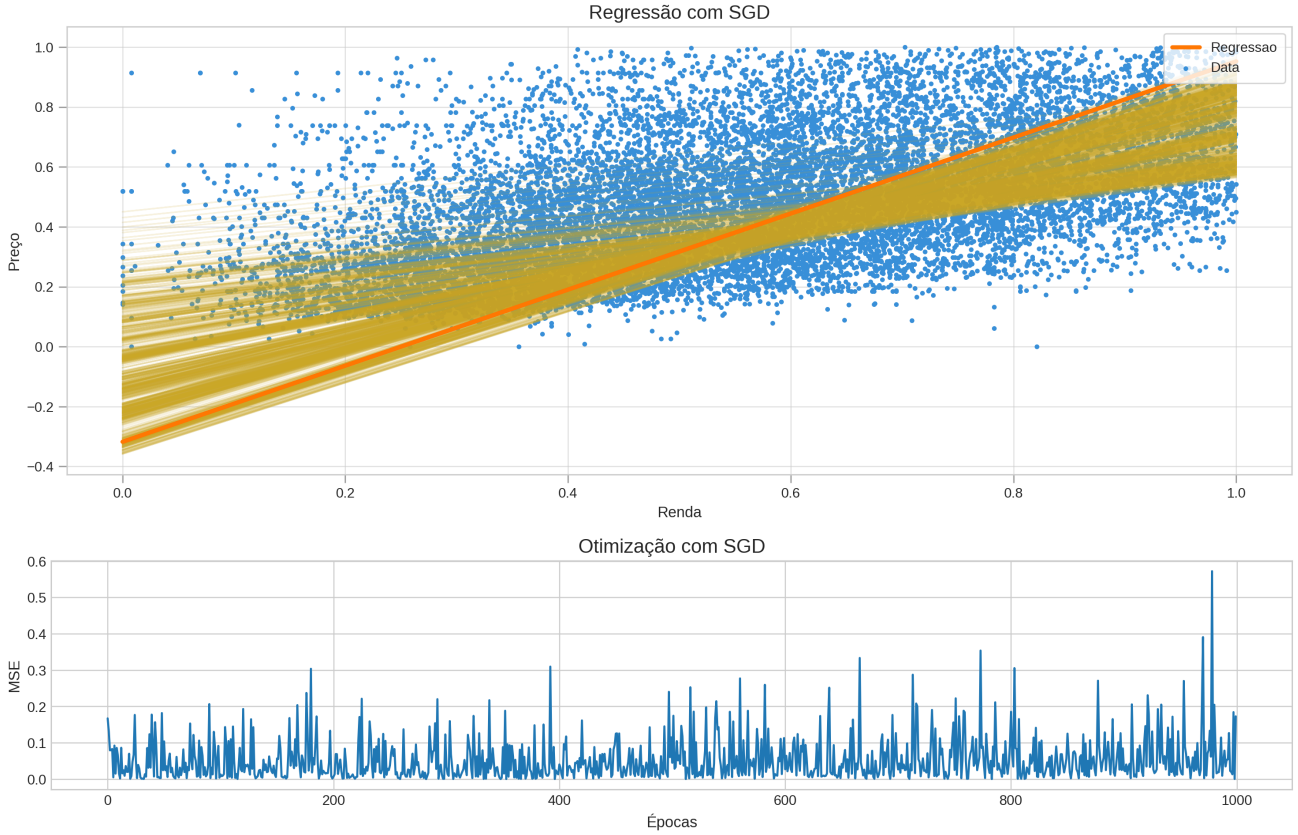
```
def SGD(X, y, lr=0.05, epoch=10, batch_size=2):

    '''
    Descida do Gradiente Estocástica
    '''

    #m, b = 0.002, 0.5 # inicializa os parâmatros
    m, b = np.random.rand(2) # inicializa os parâmatros com valores aleatórios entr
    log, mse_log = [], [] # listas para armazenar o processo de aprendizado

    for _ in range(epoch):

        # ToDo : amostre aleatoriamnte algumas instâncias (até batch_size)
        # Sorteira aleatóriamento batch_size amostras de X, com igual probabilidade
        samples = [np.random.randint(0, X.size) for _ in range(batch_size)]
        mini_batch_X = X[samples] # ToDo : complete
        mini_batch_y = y[samples]  # ToDo : complete
        N = len(mini_batch_X)

        predict = m*mini_batch_X + b # ToDo : propague (feed-forward) ara obter as

        MSE = 1/N*sum((predict - mini_batch_y)**2) # ToDo : complete

        fMini_dm = 1/N*sum(-2*mini_batch_X*(mini_batch_y - predict))  # ToDo : comp

        fMini_db = 1/N+sum(-2*(mini_batch_y - predict))

        # Updating parameters m and b
        m -= lr*fMini_dm # ToDo : atualize m com base na equação acima. Lembre-se d
        b -= lr*fMini_db # ToDo : atualize b com base na equação acima. Lembre-se d

        log.append((m, b))
        mse_log.append(MSE)

    return m, b, log, mse_log
```

Use a função SGD para o conjunto de dados de casas (X e y) carregados acima, com uma taxa

de aprendizado de 0.01 por 1000 épocas.

```
m, b, log2, mse2 = SGD(X, y, lr=0.01, epoch=1000, batch_size=2)

y_pred = m*X + b

print("MSE:",mean_squared_error(y, y_pred))
plot_regression(X, y, y_pred, log=log2, title="Regressão com SGD")

plt.figure(figsize=(16,3))
plt.rcParams['figure.dpi'] = 227
plt.plot(range(len(mse2)), mse2)
plt.title('Otimização com SGD', fontSize=14)
plt.xlabel('Épocas', fontSize=11)
plt.ylabel('MSE', fontSize=11)
plt.show()
```

```
MSE: 0.06910612491306467
```



Valor esperado: MSE: 0.03760346179860297

ψ

```
### [1pt] Discussão : O que você conclui,
curvas de custo (MSE x Épocas) ?

toDo : Discorra sobre a questão acima

O erro final exibido (MSE: 0.069106124913
todos os dados usados, após treinamento c
ajuste da regressão foi inferior ao calcu
inicial.
Ao chamar a função SGD considerando um ba
selecionadas aleatoriamente (sorteio com
amostras em cada época para avaliação da
Nesse caso, na melhor das hipóteses, em 1
2000 amostras (distintas) empregadas no t
instâncias). Esse método usado justifica
regressão do gráfico 'Regressão com SGD'.
maiores (como pe. ex. de 20 amostras), ou
épocas, o ajuste melhora com um MSE em to
0361905258466753.

Quanto ao gráfico 'Otimização com SGD' el
MSE calculados para 2 amostras, sorteadas
função SGD poderia ser alterada para regi
conjunto de dados, ao invés do MSE de cad

Assim feito, o último grafico não revela
aprendizado para o conjunto de dados como
magnitude do erro irá variar a depender d
selecionadas. Lembrando que o sorteio foi
distribuição uniforme, embora para a regr
districuição normal do erro.
```

[1pt] Discussão : O que você conclui, olhando para as duas curvas de custo (MSE x Épocas) ?

toDo : Discorra sobre a questão acima

O erro final exibido (MSE: 0.06910612491306467) foi calculado para todos os dados usados, após treinamento com SGD. Obseva-se que o ajuste da regressão foi inferior ao calculado pela regressão linear inicial. Ao chamar a função SGD considerando um batch_size de 2, serão selecionadas aleatoriamente (sorteio com reposição!) apenas 2 amostras em cada época para avaliação da função de perda (MSE). Nesse caso, na melhor das hipóteses, em 1000 épocas serão no máximo 2000 amostras (distintas) empregadas no treinamento (das 14653 instâncias). Esse método usado justifica o ajuste ruim da reta de regressão do gráfico 'Regressão com SGD'. Se considerarmos lotes maiores (como pe. ex. de 20 amostras), ou se considerarmos mais épocas, o ajuste melhora com um MSE em torno de MSE 0361905258466753.

Quanto ao gráfico 'Otimização com SGD' ele

Quanto ao grafico "Otimizaçao com SGD" ele apenas está exibindo os MSE calculados para 2 amostras, sorteadas em cada época. *Obs.: a função SGD poderia ser alterada para registrar e retornar o MSE do conjunto de dados, ao invés do MSE de cada mini_batch.*

Assim feito, o último grafico não revela uma curva que tende ao aprendizado para o conjunto de dados como um todo. Nesse caso, a magnitude do erro irá variar a depender das apenas 2 amostras selecionadas. Lembrando que o sorteio foi feito considerando uma distribuição uniforme, embora para a regressão linear se espera uma districuição normal do erro.