

PROJETO DEEP LEARNING

PROF MALGA

código e dataset.

LINK GITHUB

LINK DATASET

sobre o dataset.

O conjunto de dados selecionado foi o "Sleep Health and Lifestyle Dataset" (em tradução livre, "A Saúde do Sono e o Estilo de Vida").

sobre o dataset.

descritivo das colunas.

destacadas em negrito estão as que foram mantidas no processo.

Person ID: An identifier for each individual.

Gender: The gender of the person (Male/Female).

Age: The age of the person in years.

Occupation: The occupation or profession of the person.

Sleep Duration (hours): The number of hours the person sleeps per day.

Quality of Sleep (scale: 1-10): A subjective rating of the quality of sleep, ranging from 1 to 10.

Physical Activity Level (minutes/day): The number of minutes the person engages in physical activity daily.

Stress Level (scale: 1-10): A subjective rating of the stress level experienced by the person, ranging from 1 to 10.

BMI Category: The BMI category of the person (e.g., Underweight, Normal, Overweight).

Blood Pressure (systolic/diastolic): The blood pressure measurement of the person, indicated as systolic pressure over diastolic pressure.

Heart Rate (bpm): The resting heart rate of the person in beats per minute.

Daily Steps: The number of steps the person takes per day.

Sleep Disorder: The presence or absence of a sleep disorder in the person (None, Insomnia, Sleep Apnea).

sobre o dataset e o código.

análise e otimização.

Feita uma análise em busca de encontrar defeitos no dataset. E também, encontrar o melhor jeito de relacionar os dados.

existem valores nulos? e correlação.

Ao empregar as funções `isna()` + `sum()`, foi possível identificar a soma dos valores nulos em cada coluna. Além disso, a função `corr()` foi utilizada para analisar a correlação entre os campos selecionados, verificando sua associação.

```
data.isna().sum()
Python

Age          0
Sleep Duration 0
Quality of Sleep 0
Physical Activity Level 0
Stress Level 0
Heart Rate 0
Daily Steps 0
dtype: int64
```



```
data.corr()
Python
```

	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
Age	1.000000	0.344709	0.473734	0.178993	-0.422344	-0.225606	0.057973
Sleep Duration	0.344709	1.000000	0.883213	0.212360	-0.811023	-0.516455	-0.039533
Quality of Sleep	0.473734	0.883213	1.000000	0.192896	-0.898752	-0.659865	0.016791
Physical Activity Level	0.178993	0.212360	0.192896	1.000000	-0.034134	0.136971	0.772723
Stress Level	-0.422344	-0.811023	-0.898752	-0.034134	1.000000	0.670026	0.186829
Heart Rate	-0.225606	-0.516455	-0.659865	0.136971	0.670026	1.000000	-0.030309
Daily Steps	0.057973	-0.039533	0.016791	0.772723	0.186829	-0.030309	1.000000

sobre o dataset e o código.

normalização dos dados.

Para que cada coluna tenha o peso adequado.

Inicialmente, foi gerado um DataFrame contendo exclusivamente as características desejadas. Em seguida, foram aplicadas duas fórmulas para a normalização dos dados. Por fim, apresentamos os 10 primeiros elementos na tela, juntamente com a média de cada coluna.

```
dataFilt = data[["Age", "Sleep Duration", "Quality of Sleep", "Physical Activity Level", "Heart Rate", "Daily Steps"]]
dataFilt = (dataFilt-dataFilt.min(axis=0))/(dataFilt.max(axis=0)-dataFilt.min(axis=0))
dataFilt = (dataFilt-dataFilt.mean(axis=0))/dataFilt.std(axis=0)
dataFilt.head(10)
```

Python

	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Heart Rate	Daily Steps
0	-1.750750	-1.297149	-1.096811	-0.824314	1.652505	-1.617417
1	-1.635452	-1.171467	-1.096811	0.039791	1.168908	1.967442
2	-1.635452	-1.171467	-1.096811	0.039791	1.168908	1.967442
3	-1.635452	-1.548514	-2.767716	-1.400384	3.586893	-2.359112
4	-1.635452	-1.548514	-2.767716	-1.400384	3.586893	-2.359112
5	-1.635452	-1.548514	-2.767716	-1.400384	3.586893	-2.359112
6	-1.520153	-1.045785	-1.096811	-0.920326	2.861497	-2.050073
7	-1.520153	0.839450	-0.261358	0.759878	-0.040084	0.731284
8	-1.520153	0.839450	-0.261358	0.759878	-0.040084	0.731284
9	-1.520153	0.839450	-0.261358	0.759878	-0.040084	0.731284

```
data.mean()
```

Python

Age	1.519878e-16
Sleep Duration	1.329893e-16
Quality of Sleep	-5.699541e-17
Physical Activity Level	-2.849770e-17
Stress Level	1.994839e-16
Heart Rate	-1.139908e-16
Daily Steps	4.749617e-18
dtype: float64	

sobre o modelo.

modelo.

No presente contexto, objetivamos a determinação do Nível de Estresse fundamentado em características como qualidade do sono, nível de atividade física, entre outros. Nesse sentido, torna-se imprescindível a aplicação de um algoritmo de Regressão Linear.

1. Inicialmente, realiza-se a empilhamento de todas as colunas em uma matriz do NumPy, resultando na divisão imediata entre as características (X) e os rótulos (y).

```
col_Age = np.array(dataFilt["Age"])
col_SleepDuration = np.array(dataFilt["Sleep Duration"])
col_QualityOfSleep = np.array(dataFilt["Quality of Sleep"])
col_PhysicalActivityLevel = np.array(dataFilt["Physical Activity Level"])
col_HeartRate = np.array(dataFilt["Heart Rate"])
col_DailySteps = np.array(dataFilt["Daily Steps"])

X = np.column_stack((col_Age, col_SleepDuration, col_QualityOfSleep, col_PhysicalActivityLevel, col_HeartRate, col_DailySteps))
y = np.array(data['Stress Level'])

Python
```

2. É criada uma classe na qual serão armazenadas todas as variáveis do modelo, tais como o número da camada de entrada, pesos, valores das características, rótulos, entre outros.

Na figura apresentada, estão sendo atribuídos os valores iniciais, como os pesos gerados aleatoriamente em uma matriz do NumPy, o número de neurônios na camada de entrada, o produto das características pelo viés (bias) e os rótulos.

```
class LinearRegression:
    def __init__(self, X, y):
        self.weights0 = np.random.uniform(-1, 1, (1, X.shape[1] + 1)) # Inicializando os pesos
        self.m = X.shape[0] # Numero de samples baseado no tamanho do dataset
        self.X = np.hstack((np.ones((self.m, 1)), X)) # Stackando o valor do Bias 1 que será multiplicado pelo Bias
        self.y = y # Definindo os rótulos
```

3. A função forward realiza a multiplicação dos valores do vetor de entrada pelos pesos. Em seguida, inicia-se a função de ativação, que simplesmente retorna o valor recebido. Por fim, a função retorna o valor recebido da função Linear.

```
def forward(self, weights = None):
    o1 = self.weights0 @ self.X.T # Vetor de saída multiplicando pela matriz de pesos com o @
    self.a1 = self.Linear(o1) # Passando o vetor de saída pela ativação linear
    return self.a1

def Linear(self, x):
    return x # Função de ativação
```

sobre o modelo.

4. Após as previsões da função forward, realiza-se o cálculo dos gradientes para aprimorar a adequação dos erros. Nesse ponto, é calculado o delta, representado pela diferença entre a previsão e os rótulos. Em seguida, efetua-se o cálculo do gradiente conforme a fórmula correspondente, retornando o valor resultante.

```
def calc_grads(self, preds, i):
    delta = preds - self.y # Erro, ou seja, a previsão menos o rótulo
    grads = ((delta @ self.X / self.m)) + 1e-7 * self.weights0 # Calculando o gradiente baseado no peso e na taxa de erro
    return grads
```

5. Na figura a seguir, é apresentada a aplicação da fórmula Mean Squared Error (MSE), em que se realiza o cálculo da raiz quadrada do erro.

```
def mse(self, preds):
    return np.sum((preds - self.y).T @ (preds - self.y)) / (2. * self.m) # Mean Squared Error (MSE) é a raiz quadrada do erro
```

6. Agora, todas as funções são executadas em suas respectivas ordens, utilizando o resultado de uma como parâmetro para a próxima.

É essencial realizar o ajuste dos pesos, o qual é efetuado com base nos gradientes.

Para evitar poluição visual, é estabelecida uma condição para exibir o erro a cada 100 épocas.

```
def train(self, epochs):
    for i in range(epochs):
        preds = self.forward() # Previsões do modelo
        grads = self.calc_grads(preds, i) # Gradientes para o treinamento
        error = self.mse(preds) # Raiz quadrada do erro baseado nas previsões
        self.weights0 = self.weights0 - 0.006 * grads # Ajustando os pesos baseado na matriz de erro - taxa de aprendizado * o gradiente
        if (i % (epochs/10)) == 0: # A cada 100 épocas o erro é impresso
            print(error)
```

sobre o modelo.

7. Por fim, é instanciada a classe e realizado o treinamento com 1000 épocas. Os números refletem a eficácia do modelo, atingindo um último erro de apenas 0.11.

Após o treinamento, são coletadas as 10 primeiras previsões, apenas para fins demonstrativos. Em seguida, é apresentada a comparação entre o valor previsto e o rótulo (valor verdadeiro), destacando a inteligência do modelo.

```
model = LinearRegression(X, y) # Inicializando o modelo na classe LinearRegression
```

```
model.train(1000) # Log de erro
```

```
5980.346493589369  
1794.7470265371471  
538.6171705383325  
161.64312898844753  
48.510359259265876  
14.558347313930387  
4.369082915342575  
1.3112021103599958  
0.39350574497054724  
0.11809630795253649
```

```
preds = model.forward()[0,:10] # Pegando as primeiras 10 previsões
```

```
print(np.round(preds[:10])) # Tentativas do modelo  
print(y[:10]) # Valores reais
```

```
[7. 9. 9. 9. 9. 7. 6. 6. 6.]  
[6 8 8 8 8 7 6 6 6]
```

feito por

Alex Muchau Thiago Martins Escaliante

Ciências da Computação 4º Período, Noite.