Redes Neurais
Trabalho de Cálculo

Juliana Câmara, Josiel Santana e Stefany de Oliveira

Maio de 2023

1 Objetivo do Trabalho

O objetivo desse trabalho é elaborar um algoritmo em Python usando Redes Neurais para realizar análise de crédito de diferentes clientes de um dataset selecionado, tudo isso utilizando de conceitos aprendidos durante a aula de Cálculo.

2 Descrição do Dataset

Para esse trabalho será usado o dataset **Credit Score Classification Dataset** disponibilizado na plataforma Kaggle. O conjunto de dados conta com 164 linhas e 8 colunas, sendo elas:

- *Age*: Idade dos clientes, com o menor valor sendo 25 anos e o maior sendo 53 anos.
- *Gender*: Gênero desse clientes podendo ser 'Female' (52% do dataset) ou 'Male' (48% do dataset).
- *Income*: Salário dos clientes, sendo um número inteiro entre 25.000 a 163.000.
- *Education*: Nível de escolaridade dos clientes, podendo ser 'Bachelor's Degree', 'Master Degree', entre outros.
- *Marital Status*: Estado civíl do cliente, podendo ser 'Married' (53% do dataset) ou 'Single' (47% do dataset).
- *Number of Children*: Número de filhos de cada cliente, com o menor valor sendo 0 filhos e o maior sendo 3 filhos.
- *Home Ownership*: Coluna relacionada a moradia do cliente, podendo ser casa própria 'Owned' (68% do dataset) ou alugada 'Rented' (32% do dataset).
- *Credit Score*: Status de crédito do cliente, podendo ser 'High', 'Average' ou 'Low'.

3 Etapas do processo

3.1 Análise de Componentes Principais

Com o objetivo de reduzir o número de colunas do dataset perdendo o mínimo de informações possíveis, foi usado um algoritmo de PCA da biblioteca scikit-learn. O PCA (Principal Component Analysis), ou Análise de Componentes Principais, é um algoritmo de redução de dimensionalidade comumente utilizado em análise de dados e aprendizado de máquina. Nesse processo, os valores da variável categórica Credit Score foram mudados de 'High' para 1 e de 'Low' para 0, em 'Home Ownership' os valores 'Owned' foram mudados para 1 e 'Rented' foram mudados para 0, e em 'Education' os valores 'High School Diploma' foram mudados para 1, 'Associate's Degree' para 2, 'Bachelor's Degree' para 3, 'Master's Degree' para 4 e 'Doctorate' para 5.
Com isso foi observada a seguinte correlação entre as colunas do dataset:

	Age	Income	Education	Number of Children	Home Ownership	Credit Score
Age	1.000000	0.629114	0.108018	-0.161488	0.625316	0.569973
Income	0.629114	1.000000	0.129200	-0.098330	0.633115	0.675786
Education	0.108018	0.129200	1.000000	0.227614	0.424971	0.501959
Number of Children	-0.161488	-0.098330	0.227614	1.000000	0.346181	0.323021
Home Ownership	0.625316	0.633115	0.424971	0.346181	1.000000	0.872656
Credit Score	0.569973	0.675786	0.501959	0.323021	0.872656	1.000000

Após isso, todas as variaveis do dataset passaram a ser númericas e foi possível fazer a padronização de dados usando a função StandardScaler do Sklearn. Não houve a necessidade de remover valores ou colunas nulas do dataset pois o dataset já estava limpo.

	Age	Income	Education	Number of Children	Home Ownership
0	-1.944368	-1.295720	-0.198922	-0.886593	-2.395171
1	-1.311088	0.257348	0.508357	1.295789	0.417507
2	-0.677808	-0.519186	1.215637	0.204598	0.417507
3	-0.044528	1.033882	-1.613481	-0.886593	0.417507
4	0.588753	0.257348	-0.198922	2.386980	0.417507
...	...	...	...	...	...
123	1.602001	0.956229	-0.906202	-0.886593	0.417507
124	-1.437744	-1.994601	-1.613481	-0.886593	-2.395171
125	-0.171184	-0.907453	-0.198922	1.295789	0.417507
126	0.462097	-0.130919	0.508357	-0.886593	0.417507
127	1.095377	-0.441532	1.215637	0.204598	0.417507

Fazendo a aplicação do PCA com 5 fatores, obtivemos os seguintes resultados:

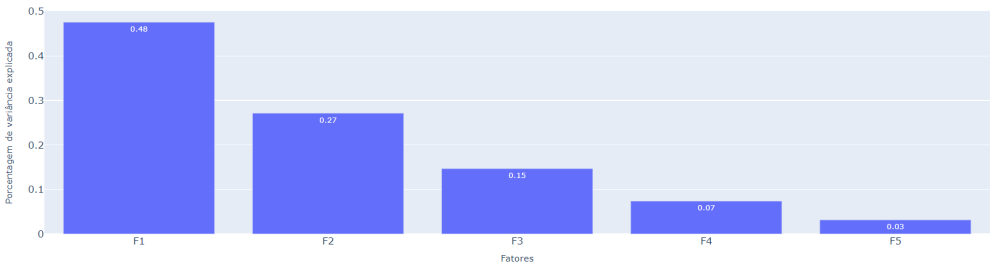


Gráfico com a porcentagem de variância explicada dos cinco fatores.

	F1	Credit Score
0	3.269016	1.0
1	0.026263	1.0
2	0.009543	1.0
3	-0.219353	1.0
4	-0.867648	1.0
...	...	...
121	-0.496444	1.0
123	-1.242632	1.0
124	3.784826	1.0
126	-0.480175	1.0
127	-0.957501	1.0

Dataset após a aplicação do PCA sendo concatenado com a coluna de Credit Score.

Entretando, ao analisar a correlação entre as variáveis do dataset notamos que as colunas 'Home Ownership' e 'Credit Score' tem uma alta correlação, então foi decidido utilizá-las no resto do trabalho.

3.2 Função do Erro

A função do erro utilizada para o desenvolvimento desse algoritmo foi a fórmula do erro quadrático médio (MSE - Mean squared Error) e ela foi utilizada para medir a discrepância entre os valores previstos e os valores reais em um problema de regressão. Essa função é amplamente utilizada na avaliação de desempenho de redes neurais e fornece uma medida objetiva da qualidade das estimativas realizadas pelo modelo. A fórmula é demonstrada por:

$$Erro = \frac{(w_1novo - w1)^2 + (w_2novo - w2)^2 + (b_1novo - b1)^2 + (b_2novo - b2)^2}{4}$$

Onde:

- w1 é o peso antigo
- w1 é o novo peso
- w2 é o peso antigo
- w2 é o novo peso
- b1 é o viés antigo
- b1 é o novo viés
- b2 é o viés antigo
- b2 é o novo viés
- Ele é dividido por 4 para obter a média

Ela foi utilizada na função descent e na função descentV:

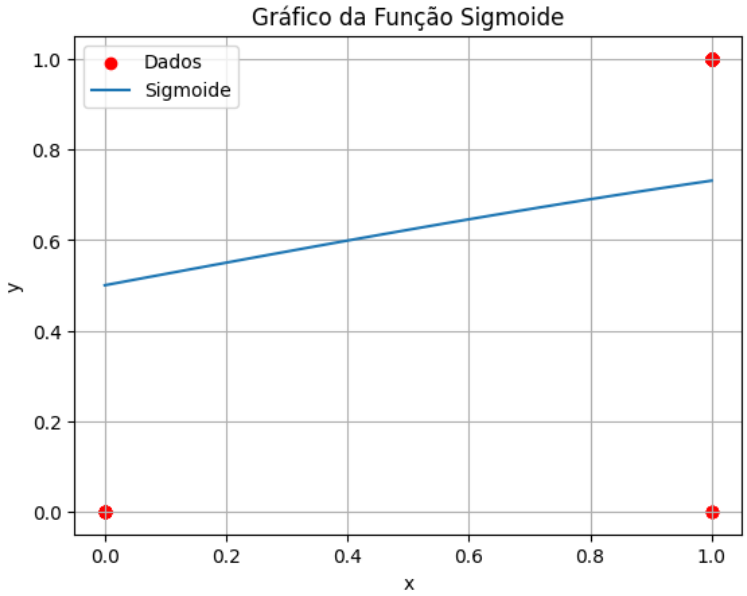
```
# gradiente desc
def descent(w1, w2, b1, b2, dataset):
    lr = 0.1
    err = 1
    i = 0
    tol = 10 **(-4)
    while err >= tol:
        dedw1, dedw2, dedb1, dedb2 = grad(w1, w2, b1, b2, dataset)

        w1_novo = w1 - lr * dedw1
        w2_novo = w2 - lr * dedw2
        b1_novo = b1 - lr * dedb1
        b2_novo = b2 - lr * dedb2

        errnovo = ((w1_novo - w1) ** 2 + (w2_novo - w2) ** 2 + (b1_novo - b1) ** 2 + (b2_novo - b2) ** 2) / 4
        print(f"Erro: {errnovo}, w1 = {w1_novo}, w2 = {w2_novo}, b1 = {b1_novo}, b2 = {b2_novo}")

        w1 = w1_novo
        w2 = w2_novo
        b1 = b1_novo
        b2 = b2_novo

        i += 1

        err = errnovo

    return w1, w2, b1, b2, i
```

3.3 Função Sigmoide

A função sigmoide foi utilizada como função de ativação em neurônios artificiais nas redes neurais. Ela mapeia a soma ponderada dos sinais de entrada para uma saída no intervalo entre 0 e 1, permitindo que os neurônios modelados pela função sigmoide respondam de forma não linear aos estímulos. Ela é representada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

O gráfico dos dados da coluna Home Ownership no eixo y e da coluna Credit Score (variável resposta) no eixo x:



Foi declarada uma função no código para esse cálculo:

```
# Função sigmoide
def sigma(x):
    return 1 / (1 + np.exp(-x))
```

3.4 Função do Gradiente

Para a função do gradiente, foi usado o seguinte conceito:
Seja $z1 = w1 \cdot x + b1$, $a1 = \sigma(z1)$, $z2 = w2 \cdot a1 + b2$ e $a2 = \sigma(z2)$.

1. Derivada do erro em relação a w_1:

$$\frac{\partial E}{\partial w_1} = (a_2 - y) \cdot \sigma'(z_2) \cdot w_2 \cdot \sigma'(z_1) \cdot x$$

2. Derivada do erro em relação a w_2:

$$\frac{\partial E}{\partial w_2} = (a_2 - y) \cdot \sigma'(z_2) \cdot a_1$$

3. Derivada do erro em relação a b_1:

$$\frac{\partial E}{\partial b_1} = (a_2 - y) \cdot \sigma'(z_2) \cdot w_2 \cdot \sigma'(z_1)$$

4. Derivada do erro em relação a b_2:

$$\frac{\partial E}{\partial b_2} = (a_2 - y) \cdot \sigma'(z_2)$$

Observe que σ' representa a derivada da função de ativação σ. Ela é representada por:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

As variáveis a_1 e a_2 são as saídas das camadas oculta e de saída, respectivamente, e z_1 e z_2 são os valores pré-ativação nas respectivas camadas. No código está representado dessa maneira:

```
# Cálculo das derivadas parciais
def grad(w1, w2, b1, b2, dataset):
    dedw1 = 0
    dedw2 = 0
    dedb1 = 0
    dedb2 = 0

    for i in train.index:
        x = train['Home Ownership'][i]
        y = train['Credit Score'][i]

        z1 = w1 * x + b1
        a1 = sigma(z1)
        z2 = w2 * a1 + b2
        a2 = sigma(z2)

        # Derivada do erro em relação a W1
        dedw1 += (a2 - y) * sigma_derivada(z2) * w2 * sigma_derivada(z1) * x

        # Derivada do erro em relação a W2
        dedw2 += (a2 - y) * sigma_derivada(z2) * a1

        # Derivada do erro em relação a B1
        dedb1 += (a2 - y) * sigma_derivada(z2) * w2 * sigma_derivada(z1)

        # Derivada do erro em relação a B2
        dedb2 += (a2 - y) * sigma_derivada(z2)

    return dedw1, dedw2, dedb1, dedb2
```

3.5 Função Neural

Para a função neural, foi utilizado o seguinte conceito:

$$\frac{\partial E}{\partial w_1} = \sum_{i=1}^{n} -(y - y_i) \cdot y_i \cdot (1 - y_i) \cdot w_2 \cdot \sigma(w_1 \cdot x_i + b_1) \cdot (1 - \sigma(w_1 \cdot x_i + b_1)) \cdot x_i$$

$$\frac{\partial E}{\partial w_2} = \sum_{i=1}^{n} -(y - y_i) \cdot y_i \cdot (1 - y_i) \cdot \sigma(w_1 \cdot x_i + b_1)$$

$$\frac{\partial E}{\partial b_1} = \sum_{i=1}^{n} -(y - y_i) \cdot y_i \cdot (1 - y_i) \cdot w_2 \cdot \sigma(w_1 \cdot x_i + b_1) \cdot (1 - \sigma(w_1 \cdot x_i + b_1))$$

$$\frac{\partial E}{\partial b_2} = \sum_{i=1}^{n} -(y - y_i) \cdot y_i \cdot (1 - y_i)$$

No código a seguir é importante destacar que o y_i é representado por $f(x)$:

```
def neural(ts, xs):
    w1 = xs[0]
    w2 = xs[1]
    b1 = xs[2]
    b2 = xs[3]

    def f(xi):
        return sigma(w2 * sigma(w1 * xi + b1) + b2)

    dedw1 = sum(
        [-(y - f(x)) * f(x) * (1 - f(x)) * w2 * sigma(w1 * x + b1) * (1 - sigma(w1 * x + b1)) * x
        for i in train.index
        for x, y in [(train['Home Ownership'][i], train['Credit Score'][i])]]
    )
    dedw2 = sum(
      [-(y - f(x) ) * f(x) * (1 - f(x)) * sigma(w1 * x + b1)
      for i in train.index
      for x, y in [(train['Home Ownership'][i], train['Credit Score'][i])]]
    )

    dedb1 = sum(
        [-(y - f(x)) * f(x) * (1 - f(x)) * w2 * sigma(w1 * x + b1) * (1 - sigma(w1 * x + b1))
         for i in train.index
         for x, y in [(train['Home Ownership'][i], train['Credit Score'][i])]]
    )
    dedb2 = sum(
        [-(y - f(x)) * f(x) * (1 - f(x))
         for i in train.index
         for x, y in [(train['Home Ownership'][i], train['Credit Score'][i])]]
    )

    return [dedw1, dedw2, dedb1, dedb2]
```

3.6 Parâmetros Utilizados

No começo do código, em duas funções chamadas grad e descent, são feitos os cálculos dos pesos e os vieses iniciais e depois, se necessário, recalculados na

função descentV.
O ideal seria os pesos e os vieses estarem corretos quando são passados na função descentV para ele só precisar de um passo para calcular. No entanto, se for preciso, ele começa calculando os pesos e os vieses baseado no que foi calculado na função descent.
Os códigos da função grad, descent e descent V estão a seguir, respectivamente:

```
# Cálculo das derivadas parciais
def grad(w1, w2, b1, b2, dataset):
    dedw1 = 0
    dedw2 = 0
    dedb1 = 0
    dedb2 = 0

    for i in train.index:
        x = train['Home Ownership'][i]
        y = train['Credit Score'][i]

        z1 = w1 * x + b1
        a1 = sigma(z1)
        z2 = w2 * a1 + b2
        a2 = sigma(z2)

        # Derivada do erro em relação a W1
        dedw1 += (a2 - y) * sigma_derivada(z2) * w2 * sigma_derivada(z1) * x

        # Derivada do erro em relação a W2
        dedw2 += (a2 - y) * sigma_derivada(z2) * a1

        # Derivada do erro em relação a B1
        dedb1 += (a2 - y) * sigma_derivada(z2) * w2 * sigma_derivada(z1)

        # Derivada do erro em relação a B2
        dedb2 += (a2 - y) * sigma_derivada(z2)

    return dedw1, dedw2, dedb1, dedb2
```

```python
# Descida do gradiente
def descent(w1, w2, b1, b2, dataset):
    lr = 0.1
    err = 1
    i = 0
    tol = 10 **(-4)
    while err >= tol:
        dedw1, dedw2, dedb1, dedb2 = grad(w1, w2, b1, b2, dataset)

        w1_novo = w1 - lr * dedw1
        w2_novo = w2 - lr * dedw2
        b1_novo = b1 - lr * dedb1
        b2_novo = b2 - lr * dedb2

        err = math.sqrt((w1_novo - w1) ** 2 + (w2_novo - w2) ** 2 + (b1_novo - b1) ** 2 + (b2_novo - b2) ** 2)

        w1 = w1_novo
        w2 = w2_novo
        b1 = b1_novo
        b2 = b2_novo

        i += 1
        #print(i)

    return w1, w2, b1, b2, i
```

```python
def descentV(grad, lr, i, err, xts):
    tol = 10 ** (-6)
    print(f"Valores iniciais de xts: {xts}")
    while err >= tol:
        dfdxs = grad(xts)
        xsnovo = [xt - lr * grad for (xt, grad) in zip(xts, dfdxs)]
        errnovo = sum([(xnovo - xt) ** 2 for (xnovo, xt) in zip(xsnovo, xts)])
        xts = xsnovo
        i += 1
        err = errnovo
        print(f"Erro: {errnovo}, dfdxs {dfdxs}, xts: {xsnovo}, i: {i}")
    return xts, i
```

```python
p = descentV(lambda xs: print(xs) or neural(train, xs), 0.1, 0, 9999, [w1, w2, b1, b2])
fp = p[0]
```

3.7 Acurácia

A acurácia do algoritmo foi calculado pela fórmula:

$$\text{acurácia} = \frac{acertos}{total}$$

O valor está entre 0 e 1, o ideal seria entre 0.95 e 1. O algoritmo para prever se as pessoas que possuem Home Ownership possuem credit score alto é de 1, ou seja, 100%.
O código retorna:

```python
correct_predictions = 0
total_samples = len(test.index)

print("Descent")
print(p)
print("1 - Possui casa própria e 0 - Não possui casa própria")
for i in test.index:
  x = test['Home Ownership'][i]
  y = test['Credit Score'][i]
  print(f"Home ownership? {x}")
  prediction = predict(x, fp)
  print(f"Predict: {prediction}")
  print(f"Real: {y}\n")
  # Verificar se a previsão está correta
  if (prediction >= 0.5 and y == 1) or (prediction < 0.5 and y == 0):
      correct_predictions += 1
accuracy = correct_predictions / total_samples
print(f"Acurácia: {accuracy}")
```

```
Acurácia: 1.0
```

4 Conclusões

Com base no dataset e o algoritmo ultilizado nesse trabalho podemos concluir que a situação de moradia da pessoa, no caso se ela tem casa própria ou alugada, impacta de uma forma grande no Score.

5 Código Fonte

5.1 Redes Neurais

5.2 PCA e correlação - Dataset