UNIVERSIDADE FEDERAL
DE MINAS GERAIS

ATIVIDADE AVALIATIVA 3
CLASSIFICADORES NÃO LINEARES - PERCEPTRON

ALUNO: FELIPPE VELOSO MARINHO
MATRÍCULA: 2021072260
DISCIPLINA: REDES NEURAIS ARTIFICIAIS

1 - O aluno deve amostrar duas distribuições normais no espaço $R2$, ou seja, duas distribuições com duas variáveis cada (Ex: $x1$ e $x2$), gerando um conjunto de dados com duas classes. As distribuição são caracterizadas como $\mathcal{N}$(2, 2, σ = 0.4) e $\mathcal{N}$(4, 4, σ = 0.4), como pode ser visualizado na Fig. 1. O número de amostras será de 200 para cada classe. Dica: Consulte o item Funções úteis no R para ver como gerar estes dados.

O objetivo da atividade é treinar um classificador linear do tipo Adaline para resolver o problema de classificação dos dados do enunciado usando a solução direta. Sendo um para uma classe e -1 para outra. Por ser um sistema de N equações e n incógnitas, pode ser representado na forma matricial de 3.

$$\mathbf{X}\mathbf{w} = \mathbf{y} \tag{3}$$

em que $\mathbf{X}$, $\mathbf{w}$ e $\mathbf{y}$ são representados em 4, 5 e 6.

$$\mathbf{x} = \begin{bmatrix} x_{n_1} & x_{n-1_1} & \cdots & x_{1_1} & 1 \\ x_{n_2} & x_{n-1_2} & \cdots & x_{1_2} & 1 \\ \vdots & \vdots & \cdots & \vdots & \\ x_{n_N} & x_{n-1_N} & \cdots & x_{1_N} & 1 \end{bmatrix} \tag{4}$$

$$\mathbf{w} = \begin{bmatrix} w_n \\ w_{n-1} \\ \vdots \\ w_1 \\ w_0 \end{bmatrix} \tag{5}$$

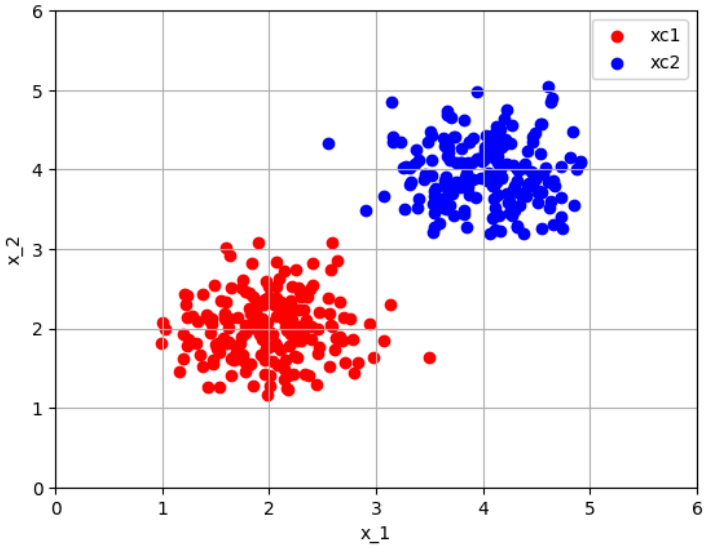
$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{6}$$

A matriz $\mathbf{w}$ pode ser obtida por meio da pseudoinversa, conforme Equação 7.

$$\mathbf{w} = \mathbf{X}^{+}\mathbf{y} \tag{7}$$

No código, primeiramente definimos os parâmetros da atividade e geramos amostras randômicas para xc1 e xc2. Com isso, criamos uma matriz com 200 amostras de 2 dimensões (x, y) deslocadas nas posições [2, 2] e [4, 4] em xc1 e xc2 respectivamente. Quanto maior os valores dos parâmetros σ = s1 = s2 = 0.4 maior é a dispersão das

amostras, por isso multiplicamos o valor com os pontos na matriz para manter os pontos menos dispersos.



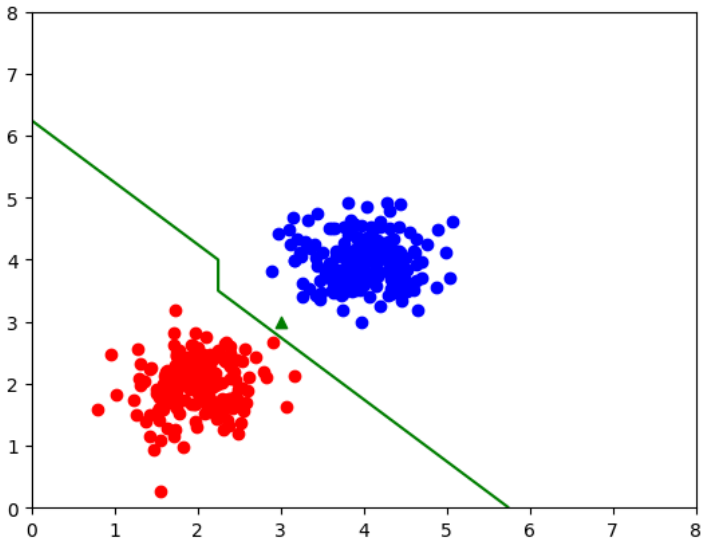
Depois de gerarmos os dados do problema podemos modelar o Adaline. Sendo X a matriz com a combinação dos pontos de xc1 e xc2, y a saída e W(pesos) como a pseudoinversa de X * Y(saída), criamos nosso modelo para predição.

```
# Predições
yhat1 = np.sign(X[0, :] @ W) # predição para o primeiro ponto
yhat40 = np.sign(X[39, :] @ W) # predição para o quadragésimo ponto
```

Na linha acima, estamos fazendo a predição para o primeiro ponto de dados. Aqui, X é uma matriz que contém os dados de entrada, onde cada linha representa um ponto de dados e cada coluna representa uma característica. X[0, :] seleciona a primeira linha da matriz X, que corresponde ao primeiro ponto de dados. W é um vetor de pesos que foi aprendido pelo modelo Adaline. O operador @ realiza a multiplicação matricial entre X[0, :] e W, e np.sign() retorna o sinal do resultado dessa multiplicação. Portanto, *yhat1* será igual a 1 se o resultado for positivo, -1 se for negativo e 0 se for zero.

Criamos um *ponto_teste* para calcularmos a previsão neste ponto. Para isso, usamos “np.sign” para retornar -1 caso o valor seja menor que 0 e 1 se o escalar for maior que 0. Com valores para definir a superfície de separação e utilizando a classe “enumerate” percorremos a matriz completa classificando os pontos.

Plotando a superfície agora podemos verificar a divisória entre os dois pontos bem definida.



2 - Resolva o mesmo problema anterior implementando o treinamento através da regra Delta vista em sala de aula. Dica: consulte as notas de aula e os slides para ver como implementar a função de treinamento. Com o mesmo conjunto de dados utilizado anteriormente (não gere os dados novamente) Gere um o gráfico com o melhor hiperplano de separação e compare com o do exercício anterior.

O Adaline é um tipo de rede neural artificial simples projetada para classificação binária. Ele se ajusta a uma função linear aos dados de entrada para prever a classe de saída. O ajuste dos parâmetros (ou pesos) é realizado através de um processo iterativo que minimiza uma função de custo, geralmente o erro quadrático médio (MSE - Mean Squared Error).

Para que o treinamento de um modelo como o Adaline seja feito, seguimos os processos:

```
# Inicialização dos pesos
wt = np.random.uniform(-0.5, 0.5, (n, 1))
```

- Inicialização dos Pesos Iniciais: Os pesos são inicializados aleatoriamente. No código, isso é feito com `wt = np.random.uniform(-0.5, 0.5, (n, 1))`, onde `n` é o número de características de entrada mais um para o bias (viés).

```
# Treinamento do Adaline
n_epocas = 0
erro_epoca = tolerancia + 1
erro_evec = np.zeros(max_epocas)

while n_epocas < max_epocas and erro_epoca > tolerancia:
    erro_i2 = 0
```

```
    xseq = np.random.permutation(N)
    for i in range(N):
        irand = xseq[i]
        yhati = np.dot(X[irand], wt)
        erro_i = Y[irand] - yhati
        gradiente = eta * erro_i * X[irand, :, np.newaxis]
        wt += gradiente
        erro_i2 += (erro_i ** 2)
    erro_epoca = erro_i2 / N
    erro_evec[n_epocas] = erro_epoca
    n_epocas += 1
```

- **Loop de Treinamento | Épocas:** Uma época corresponde a uma passagem completa através de todo o conjunto de dados de treinamento. O número máximo de épocas foi definido como 1000, mas o treinamento pode parar mais cedo se o erro se tornar menor que uma tolerância definida.

- **Embaralhamento dos Dados:** A cada época, os índices dos dados são embaralhados para garantir que a ordem de apresentação dos exemplos não afete o aprendizado.

- **Atualização dos Pesos:** Para cada exemplo no conjunto de dados, o modelo faz uma predição (`yhati = np.dot(X[irand], wt)`), calcula o erro (`erro_i = Y[irand] - yhati`), e então usa esse erro para ajustar os pesos. A atualização é feita na direção que minimiza o erro quadrático, proporcionalmente à taxa de aprendizado (`eta`), conforme a fórmula de atualização de pesos: `wt += eta * erro_i * X[irand, :, np.newaxis]`.

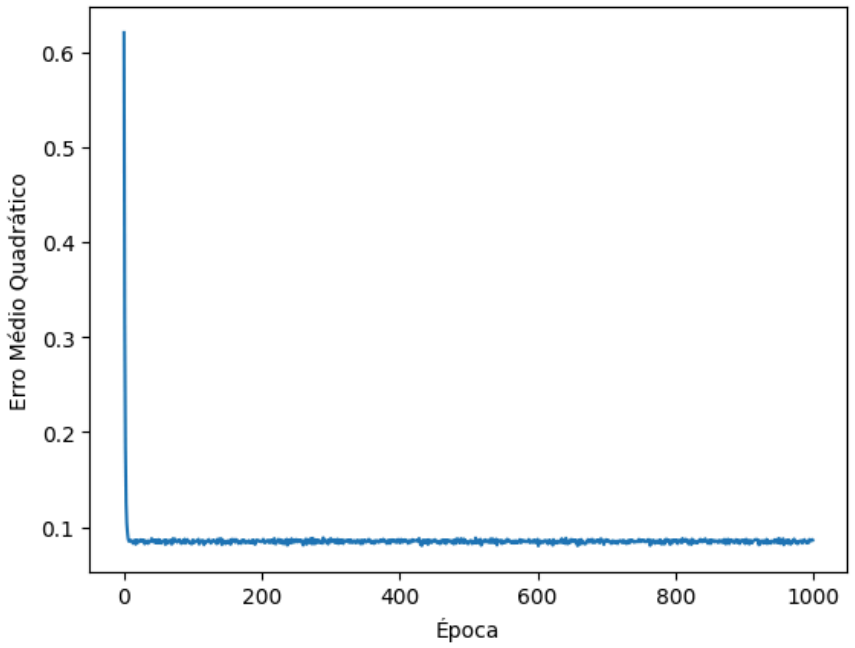
- **Cálculo do Erro Quadrático Médio (MSE):** Durante cada época, o erro quadrático é acumulado e, no final da época, é calculado o MSE dividindo o erro quadrático acumulado pelo número de amostras. Esse valor é verificado a cada época para determinar se o processo de treinamento deve continuar ou parar.

Antes de prosseguir, a diferença entre os dados de treino e os de teste são:
- Dados de Treino: São os dados usados para ajustar os parâmetros do modelo. No caso do Adaline, são utilizados para atualizar os pesos.

- Dados de Teste: Não especificamente mencionados no código fornecido, mas em geral, são um conjunto de exemplos usados apenas para avaliar o desempenho do modelo treinado. Eles ajudam a verificar se o modelo generaliza bem para novos dados, não vistos durante o treinamento.

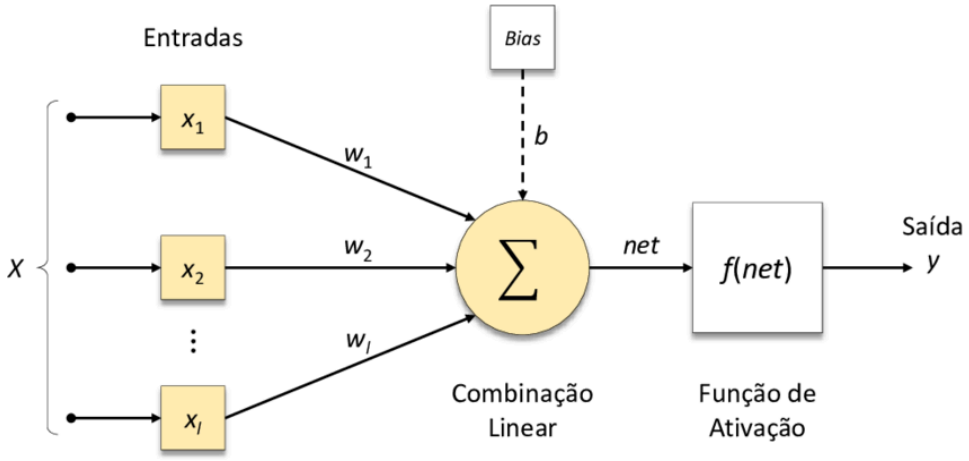
Importância das Épocas e Erros
No gráfico exibido um plot demonstrando a relação de erro médio quadrático em relação ao número de Épocas. Ou seja, quando maior o número de épocas, menor é o Erro Médio Quadrático.
- As Épocas servem para refinar continuamente os pesos do modelo. Múltiplas passagens são frequentemente necessárias para que o modelo aprenda eficazmente a partir de todo o conjunto de dados.

- Erros medem quão longe as predições do modelo estão dos valores reais. A minimização desses erros durante o treinamento é crucial para garantir que o modelo não só se ajuste aos dados de treino mas também generalize bem para novos dados.



3 - Transforme o algoritmo de treinamento do Adaline no exercício 2 para implementar um perceptron e resolva o mesmo problema sem gerar dados novos e compare as três soluções. Gere o gráfico do hiperplano de separação aqui também.

Primeiramente, o Perceptron, assim como o Adaline, realiza classificações binárias. Sendo basicamente um classificador linear que separa duas classes utilizando uma função de ativação do tipo degrau.

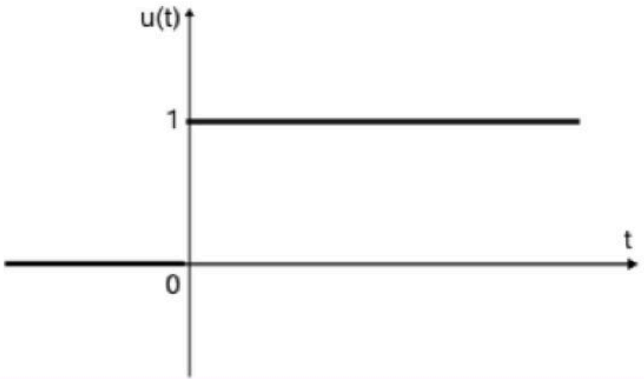


A estrutura do perceptron é definida como o diagrama acima.

- Entradas (feature): x1, x2, …, xn;
- Pesos associados às entradas: w1, w2, … , wn;
- Bias: b (peso adicional que ajuda o modelo a ajustar-se melhor aos dados)

A função de ativação utilizada para problemas de classificação é a função degrau:

$$u(t)=\begin{cases} 0, & t<0 \\ 1, & t>0 \end{cases}$$



O perceptron aprende ajustando os pesos e o bias com base nos erros cometidos nas previsões durante o treinamento. O objetivo é encontrar uma linha ou plano que separa os conjuntos de treinamento. Os pesos são ajustados pela fórmula:

$$W_{new} = W_{old} + \eta(Y_{real} - Y_{previsto})x$$

Onde:

- η é a taxa de aprendizado (um pequeno número positivo),
- Yreal é a classe real do exemplo,
- Yprevisto é a classe prevista pelo modelo,
- X são as entradas do exemplo.

No código definimos como:

```
def step_function(x):
    return np.where(x >= 0, 1, -1)
```

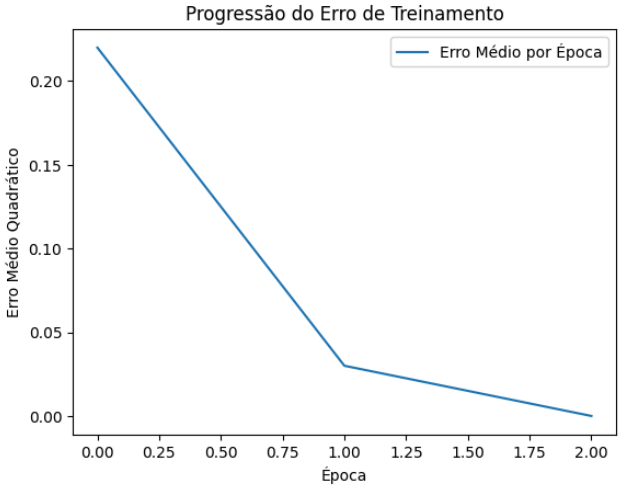
O laço para treinamento do modelo foi definido no código abaixo seguindo uma lógica parecida a utilizada no Adeline onde:

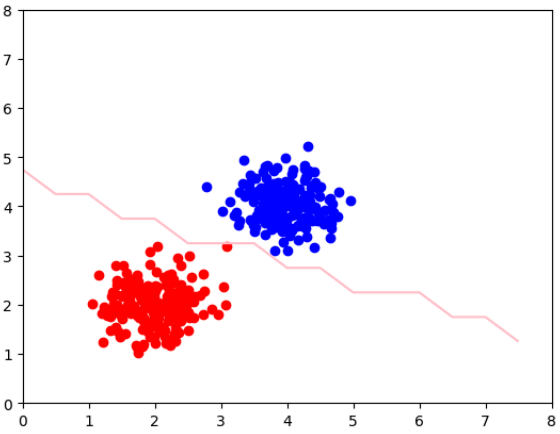
- erro_i2 acumula o quadrado dos erros para calcular o erro médio quadrático da época.
- xseq é uma permutação aleatória dos índices dos exemplos, usado para iterar sobre os exemplos em ordem aleatória a cada época (isso ajuda a evitar ciclos em dados ordenados).
- yhati é a predição do modelo, calculada pela função de ativação aplicada ao produto escalar entre os pesos e as características do exemplo.
- erro_i é a diferença entre a predição e o valor real.
- gradiente calcula o ajuste necessário para os pesos.
- erro_medio calcula o erro médio quadrático da época atual, que é usado para verificar se o treinamento deve continuar.

```
# Inicialização de pesos
wt = np.random.uniform(-0.5, 0.5, n)

n_epocas = 0
erro_epoca = tolerancia + 1
erro_evec = []

while n_epocas < max_epocas and erro_epoca > tolerancia:
    erro_i2 = 0
    xseq = np.random.permutation(N)

    for i in range(N):
        irand = xseq[i]
        yhati = step_function(X[irand, :] @ wt)
        erro_i = Y[irand] - yhati
        gradiente = eta * erro_i * X[irand, :]

        # Atualização dos pesos
        wt += gradiente

        # Acumulação do erro quadrado
        erro_i2 += erro_i ** 2

    # Atualização do número de épocas
    n_epocas += 1
    erro_medio = erro_i2 / N
    erro_evec.append(erro_medio)
```

```
    erro_epoca = erro_medio
```

Com isso chegamos ao resultado:





Percebe-se que o hiperplano de separação do Perceptron (em rosa) tem uma maior variância e insconsistência do que se comparado ao do Adaline.

Sendo assim, o Adaline possui maior flexibilidade em relação aos dados, com um algoritmo pioneiro para treinamento de redes de múltiplas camadas que é a regra Delta.