UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Projeto 02 - Computação Bioinspirada

Aluno: Thiago Vasconcelos Braga

Matrícula: 11921BSI225

UBERLÂNDIA, SETEMBRO 2023.

1) Introdução

Esse trabalho trata-se Implementação de um Perceptron para classificação de dados da base Iris por meio dos 4 atributos, sendo eles: comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala. Como trata-se de um algoritmo de classificação binária, optamos pelas classes Versicolor e Virgínica, abaixo discorreremos sobre as etapas implementadas

2) Desenvolvimento

02.1) Carregamento dos dados:

Nesta etapa, é formada uma matriz de atributos, que contém o valor dos 4 atributos para cada uma das flores do conjunto, e um vetor contendo a classificação de cada uma dessas flores, sendo que no vetor de classes, o valor –1 se refere à Virgínica e o valor 1 se refere à classe Versicolor.

```
iris = load_iris()
atributos = iris.data
classes = iris.target

atributos_do_conjunto = atributos[classes != 0]
classes_conjunto = classes[classes != 0]

classes_conjunto = np.where(classes_conjunto == 1, 1, -1)

taxa_aprendizado = 0.1;
epoch = 100
```

Imagem 01 - Código de carregamento dos dados.

02.2) Separação dos conjuntos de treinamento e teste:

O conjunto é dividido em dois subconjuntos, sendo treinamento e teste, na etapa de treinamento ajustamos o Percéptron com os pesos e o viés, para que ele aprenda os dados relevantes para realizar a classificação. Já na etapa de testes, colocamos o Percéptron em prática para que ele classifique as flores do conjunto de testes. Trabalhamos com conjuntos de teste do tamanho 0,1; 0,3 e 0.5, que representa a proporção em relação ao conjunto total.

```
# Divisão entre treinamento e teste
atributos_treinamento, atributos_teste, classes_treinamento, classes_teste = train_test_split(atributos_do_conjunto, cl
tamanho_conjunto = atributos_teste.shape[0]
#Inicialização pesos/bias
```

Imagem 02 - Divisão entre os conjuntos de treinamento e teste.

2.3) Inicialização:

Os vetor de pesos e o bias são inicializados com valores aleatórios.

```
np.random.seed(0)
pesos = np.random.rand(atributos_treinamento.shape[1])
viés = np.random.rand()
```

Imagem 03 - Inicialização do vetor de pesos e viés.

2.4) Treinamento do Perceptron:

Na etapa de treinamento, o algoritmo é executado por um número de épocas(epochs), para cada exemplo no conjunto de treinamento, é feito o cálculo da soma ponderada das características multiplicadas pelos pesos atribuídos a cada uma delas, e adicionado o bias, assim a saída real é comparada com o alvo, se a saída for diferente do alvo, o vetor de pesos e o bias são atualizados, essa é uma etapa muito importante na execução, pois aqui ocorre o aprendizado de fato.

```
for epoca in range(epoch):
    for i in range(atributos_treinamento.shape[0]): #Tamanho conjunto treinamento
        entrada = atributos_treinamento[i]
        objetivo = classes_treinamento[i]
        soma_ponderada = np.dot(entrada, pesos) + viés #Soma ponderada das entradas do neuronio + vies
        ativação = np.sign(soma_ponderada)
        if ativação != objetivo:
            erro = objetivo - ativação
            pesos += taxa_aprendizado * erro * entrada
            viés += taxa_aprendizado * erro
```

Imagem 04 - Código de treinamento do Perceptron.

2.5) Testes do Perceptron:

Nesta etapa, o perceptron classifica dados de teste baseado no seu processo de treinamento, ele vai utilizar a relevância (peso) calculada para cada um dos parâmetros para determinar qual classe uma flor do conjunto pertence, nesta etapa, podemos utilizar parâmetros de avaliação, como a acurácia, em que, através desses dados podemos avaliar o desempenho real do algoritmo.

```
corretos = 0
for i in range(tamanho_conjunto): #Tamanho conjunto de testes
    entrada = atributos_teste[i]
    objetivo = classes_teste[i]
    soma_ponderada = np.dot(entrada, pesos) + viés
    ativação = np.sign(soma_ponderada)
    if ativação == objetivo:      You, 3 days ago • Initial commit
        corretos += 1

acuracia = (corretos / tamanho_conjunto) * 100 #Quantidade de acertos/tamanho do conjunto
```

Imagem 05 – Testes do Perceptron.

2.6) Resultados de execução dos testes

2.6.1) Acurácia por tamanho do conjunto de testes

Considerando uma taxa de aprendizado de 0.1, uma epoch = 100, fizemos alterações no tamanho do conjunto de testes, utilizando o tamanho de 0.1, 0.3 e 0.5, foi observado um ganho

de acurácia de 2% quando utilizado o conjunto de testes de 0.5, ou seja, uma maior quantidade de dados de teste resultou em maior desempenho do algoritmo, portanto, verificamos que um conjunto de treinamento grande não necessariamente melhora o resultado dos testes.

	Taxa de aprendizado	**Conjunto de testes**	**Epoch**	**Acurácia**
Teste 01	0.1	0.1	100	90,00%
Teste 02	0.1	0.3	100	90%
Teste 03	0.1	0.5	100	92%

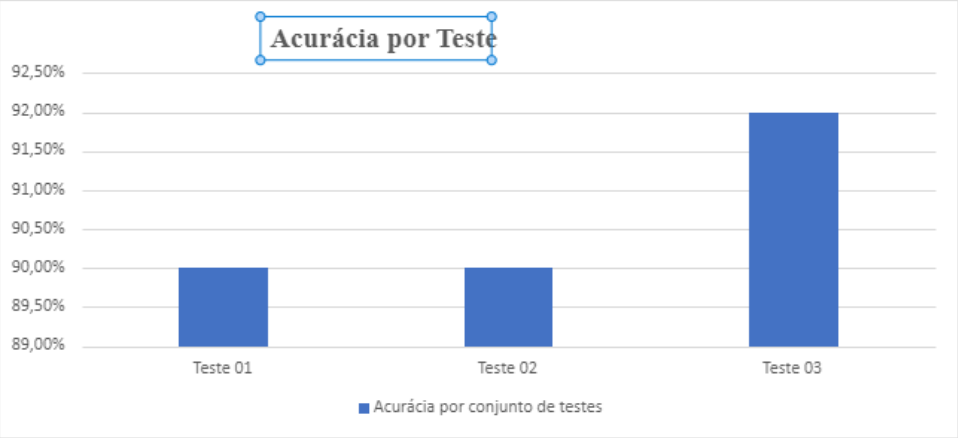

Imagem 06 - Gráfico de acurácia por tamanho do conjunto de testes.

2.6.2) Acurácia por taxa de aprendizado

Utilizando o tamanho do conjunto de testes de 0.5, uma epoch = 100, que foram os atributos que resultaram em maior acurácia no teste anterior, fizemos alterações na taxa de aprendizado, para que fosse avaliado o impacto dessa alteração na acurácia do perceptron, foram utilizados taxas de aprendizado de 0.2, 0.3 e 0.5, os resultados podem ser visualizados abaixo, onde a melhor acurária foi obtida com a taxa de aprendizado de 0.2:

	Taxa de aprendizado	**Conjunto de testes**	**Epoch**	**Acurácia**
Teste 04	0.2	0.5	100	96,00%
Teste 05	0.3	0.5	100	92%
Teste 06	0.5	0.5	100	94%

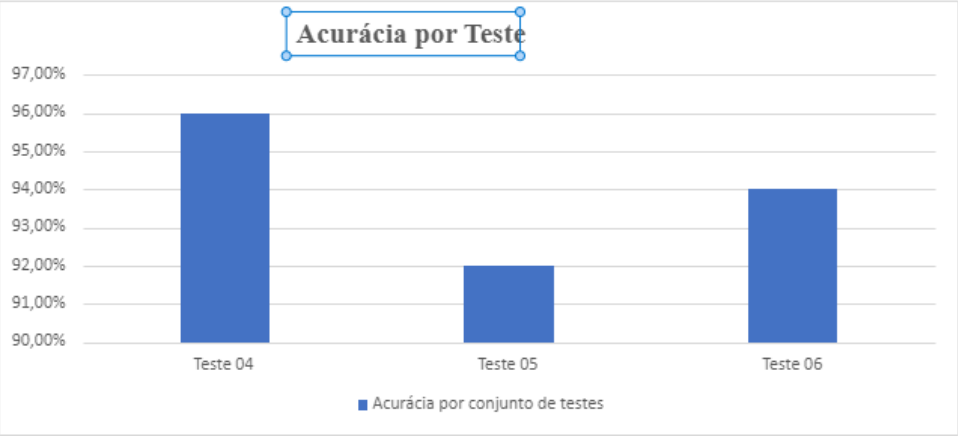

Imagem 07 - Gráfico de acurácia por taxa de aprendizado.

Podemos portanto concluir que as alterações na taxa de aprendizado trouxeram ganhos de desempenho, principalmente utilizando o valor de 0.2.

2.6.3) Acurácia por EPOCH

Utilizando os valores que obtemos o melhor resultado até o momento para a taxa de aprendizado e o tamanho do conjunto de testes, realizamos também alterações na epoch para avaliar o impacto na acurácia. Nos teste 7, 8 e 9, utilizamos o valor de epoch de 200, 500 e 10000, respectivamente. Conforme resultados abaixo, foi mantida a acurácia de 96% e não houve impacto pela alteração da epoch.

	Taxa de aprendizado	**Conjunto de testes**	**Epoch**	**Acurácia**
Teste 07	0.2	0.5	200	96,00%
Teste 08	0.2	0.5	500	96,00%
Teste 06	0.2	0.5	10000	96,00%

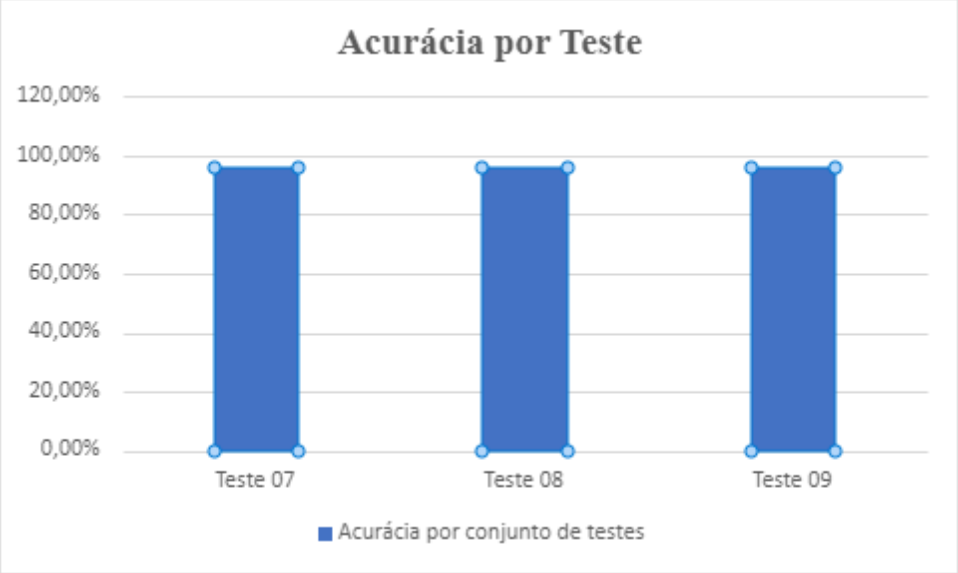

Imagem 08 – Acurácia por Epoch.

2.6.3) Testes com a terceira classe: Setosa

Todos os testes relatados nos itens acima foram realizados também para a terceira classe para entender qual seria o comportamento, ficou evidenciado que todos os elementos da classe setosa foram classificados como Virgínica. Essa classificação se dá pois o perceptron entendeu que os atributos do conjunto de testes com elementos Setosa mais se assemelhavam com as caraterísticas da Virgínica, a qual ele foi treinado.

```
#Classifica setosa sem treinar com os pesos de treinamento anteriores
versicolor = 0
virginica = 0
for j in range(atributos_setosa.shape[0]):
    entrada_setosa = atributos_setosa[i];
    soma_ponderada_setosa = np.dot(entrada_setosa, pesos);
    ativação_setosa = np.sign(soma_ponderada)
    if ativação == -1:
        versicolor += 1
    else:
```

Imagem 08 – Classificação do terceira classe.