UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Projeto 03 - Computação Bioinspirada

Aluno: Thiago Vasconcelos Braga

Matrícula: 11921BSI225

UBERLÂNDIA, NOVEMBRO 2023.

1) Introdução

Esse trabalho trata-se da combinação dos algoritmos bioinspirados, a rede neural Perceptron e um algoritmo genético. Foi implementado 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. Porém, a etapa de treinamento do perceptron foi substituída por um algoritmo genético que irá evoluir indivíduos das populações geradas, cada indivíduo é um vetor de pesos do perceptron, e através das etapas de fitness(aptidão), crossover e mutação, tentaremos obter o melhor indivíduo possível para ser utilizado como vetor de pesos.

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]

atributos_setosa = atributos[classes == 0]#ApenasSetosa

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

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, o primeiro é utilizado para verificar a aptidão dos vetores de peso gerados pelo algoritmo genético, já na etapa de testes, colocamos o Perceptron em prática para que ele classifique as flores do conjunto de testes utilizando os pesos definidos. 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) Aplicação do algoritmo genético para obter vetor de pesos:

Esta etapa substitui o que seria o treinamento do Perceptron, ao invés de realizar a etapa de treinamento, foi utilizado um algoritmo genético para obter o melhor vetor de pesos possível, seguindo os seguintes passos:

01) Inicializar a primeira população, cada indivíduo da população gerada é um vetor de pesos+bias.

02) Iterar pelo número total de gerações definido, em cada iteração seguir os seguintes passos:

- Para cada indivíduo na população, verifica a sua aptidão;
- Seleciona o melhor indivíduo e adiciona em uma nova população;
- Seleciona dois indivíduos da população atual para efetuar o cruzamento;
- Efetua o cruzamento e a mutação para gerar um novo filho;
- Adiciona filho gerado na nova população;
- Troca a população antiga pela nova população gerada

03) Após finalizada as iterações, o vetor de pesos e bias é o indivíduo da população com melhor aptidão(fitnesss)

```
populacao = inicializar_populacao(tamanho_populacao, dimensao_individuo)

for geracao in range(num_geracoes):
    aptidoes = []
    for individuo in populacao:
        apt = aptidao(individuo)
        aptidoes.append(apt);
    melhor_individuo = populacao[np.argmin(aptidoes)]
    # populacao que contem apenas o melhor indivíduo, será a próxima populacao
    populacao_selecionada = [melhor_individuo]
    #Gera filhos
    while len(populacao_selecionada) < tamanho_populacao:
        #seleciona dois indices da populacao para serem os pais
        indices = np.random.choice(len(populacao), size=2, replace=False)
        pai1 = populacao[indices[0]]
        pai2 = populacao[indices[1]]
        filho = crossover(pai1, pai2)
        filho = mutacao(filho, taxa_mutacao)
        populacao_selecionada.append(filho)
    populacao = populacao_selecionada

melhor_pesos = melhor_individuo[:-1]
melhor_bias = melhor_individuo[-1]
```

Imagem 04 - Aplicação do algoritmo genético.

2.4.1) Definição de atributos do algoritmo genético

Inicialmente foram definidos os parâmetros do algoritmo genético, sendo o tamanho da população de 100 indivíduos, a dimensão dos indivíduos é o número de posições do vetor de pesos, sendo uma posição para cada atributo e mais uma posição ao final para armazenar o bias.

```
#Genético
tamanho_populacao = 100
dimensao_individuo = atributos_treinamento.shape[1] + 1  #Adiciona o viés na ultima posição
taxa_mutacao = 0.2
num_geracoes = 200
```

Imagem 05 - Parâmetro do algoritmo genético.

2.4.2) Inicialização da população

A população inicial é gerada pela função 'inicializar_populacao', que cria indivíduos em que cada posição tem valores reais aleatórios, entre –1 e 1 até o tamanho da população definido.

```
def inicializar_populacao(tamanho_populacao, dimensao_individuo):
    populacao = []
    for j in range(tamanho_populacao):
        individuo = np.random.uniform(-1, 1, dimensao_individuo)
        populacao.append(individuo)
    return populacao
```

Imagem 06 - Inicialização da população.

2.4.3) Aptidão de cada indivíduo na população

A função de aptidão recebe um indivíduo, que é um vetor de pesos e classifica o conjunto de treinamento utilizando este vetor, para cada elemento classificado adiciona o resultado da classificação na variável 'previsao', calcula-se a diferença entre o esperado e a previsão, que é o erro, eleva-se o erro ao quadrado e adiciona na variável 'erro_quadrado_total'. Finalizada a classificação dos elementos, divide-se a soma dos erros elevados ao quadrado pelo tamanho do conjunto de treinamento, obtendo o erro médio quadrático obtido pelo classificador utilizando o vetor de pesos, que é o indivíduo da população no algoritmo genético, quanto menor o erro, melhor foi o desempenho do Perceptron utilizando aquele indivíduo como vetor de pesos.

```python
#Para cada individuo calcula a aptidao EMQ
def aptidao(individuo):       You, 15 hours ago • Primeiro Commit
    pesos = individuo[:-1]
    viés = individuo[-1]
    erro_quadrado_total = 0
    for i in range(atributos_treinamento.shape[0]):
        entrada = atributos_treinamento[i]
        alvo = classes_treinamento[i]
        soma_ponderada = np.dot(entrada, pesos) + viés
        previsao = 1 if soma_ponderada >= 0 else 0
        erro = alvo - previsao
        erro_quadrado_total += erro ** 2
    emq = erro_quadrado_total / atributos_treinamento.shape[0]
    return emq
```

Imagem 07 - Função de aptidão(fitness).

2.4.4) Cruzamento(Crossover)

Nesta etapa, é realizado o cruzamento entre dois pais selecionados, onde é selecionado um ponto de corte aleatório, e os dois pais são concatenados, de maneira que à partir do ponto de corte utiliza-se os cromossomos do pai2 e antes os cromossomos do pai 1.

```python
def crossover(pai1, pai2):
    ponto_corte = np.random.randint(0, len(pai1))
    #Une pais de acordo com o ponto de corte definido
    filho = np.concatenate((pai1[:ponto_corte], pai2[ponto_corte:]))
    return filho
```

Imagem 08 - Função de crossover(cruzamento).

2.4.5) Mutação

Na etapa de mutação, a função ‘mutacao’ a taxa de mutação e um indivíduo, e gera um número aleatório entre –1 e 1, se o número for menor que a taxa de mutação definida, o individuo é mutado, dessa maneira, fazemos com que a mutação ocorra em algumas posições do cromossomo e outras não.

```python
def mutacao(individuo, taxa_mutacao):
    #Verifica numero aleatório gerado e compara com a taxa de mutação para mutar o filho
    for i in range(len(individuo)):
        if np.random.uniform(-1, 1) < taxa_mutacao:       You, 1 second ago • Uncommitted changes
            individuo[i] = np.random.uniform(-1, 1)
    return individuo
```

Imagem 09 - Função de mutação.

2.5) Testes da aplicação:

Nesta etapa, o perceptron classifica dados de teste baseado no vetor de pesos obtido por meio do algoritmo genético, 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 10 – Testes do Perceptron.

2.6) Resultados de execução dos testes

2.6.1) Acurácia por tamanho do conjunto de testes

Foram realizados 3 testes variando o tamanho do conjunto de testes, utilizando o tamanho de 0.1, 0.3 e 0.5. Podemos constatar, conforme gráfico e tabela abaixo, que a combinação do algoritmo genético com o Perceptron obteve o mesmo desempenho que o Perceptron nos conjuntos de testes de 0.1 e 0.3, e no conjunto de testes de 0.5 a combinação trouxe uma acurácia ainda maior que o algoritmo Perceptron sozinho.

	Conjunto de testes	**Acurácia Perceptron+Genético**	**Acurácia Perceptron**
Teste 01	0.1	90,00%	90,00%
Teste 02	0.3	90,00%	90%
Teste 03	0.5	94,00%	92%

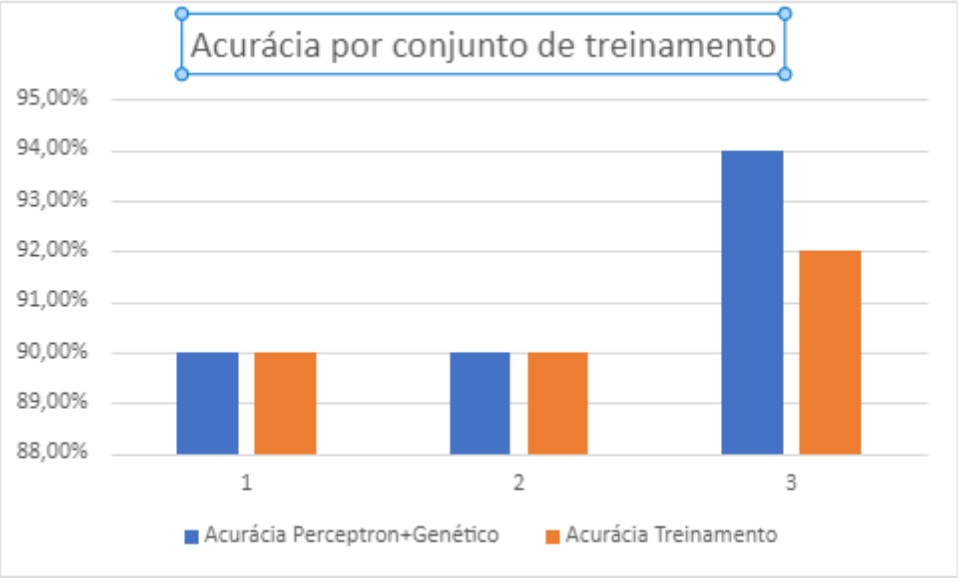

Imagem 11 - Gráficos de resultados