Github: https://github.com/JoaoVictorfss/percepton2_iris/tree/master

Link do codigo do segundo trabalho:
https://github.com/DanielGoncalves666/Computacao_Bio-Inspirada/tree/main/Projeto%201

Pegar o trabalho 2 finalizado

- remover a lógica da função de treinamento e manter o resto
- mesclar o trabalho 1 para treinamento -> classificar de acordo com o vetor de pesos(melhor indivíduo) e realizar eventuais alterações. Fitness: aplicar o cromossomo a cada entrada e verificar se ela foi testada corretamente, o fitness será a quantidade de acertos. A quantidade máxima de fitness depende da quantidade máxima de entrada(parâmetro para treinamento/porcentagem)
- usar a base iris
- mesmos parâmetros de entrada e saída do trabalho anterior

Etapas:

Executado antes do treinamento…

Variavel global para o vetor com os cromossomos

-> _pop: vetor de indivíduo

Função para iniciar população de pesos: Luiz

-> Deve ser gerada uma população de pesos para iniciar a variável _pop
-> Recebe como parâmetro o tamanho da população
-> Cada cromossomo é um vetor de 5 posições
-> Código para gerar um cromossomo:

```
[random.uniform(-1, 1) for _ in range(5)]
```

…Durante o treinamento

Função para calcular fitness/avaliação com base na precisão: Daniel

calcular a aptidão de cada cromossomo para usar no algoritmo genético

-> Recebe como parâmetros: espécies disponíveis; porcentagem da base inicial; classes

-> Aplicar a junção aditiva e a função de ativação para verificar a quantidade de acertos para cada cromossomo

-> Alterar população

Após a execução das funções anteriores, teremos uma lista de população, onde cada indivíduo terá seu cromossomo(peso) e sua aptidão

Função de seleção: Arthur

-> usar torneio
-> criar população intermediária do mesmo tamanho da população
-> retorna a população intermediária(vetor de indivíduo)

Github: https://github.com/JoaoVictorfss/percepton2_iris/tree/master

Função de crossover: Luiz

-> Recebe a população intermediária obtida na função de seleção
-> Pega par de indivíduos da população e aplica o crossover de 1 ponto
-> Retorna nova população intermediária dos filhos

Função de mutação: Daniel

-> Recebe a população intermediária retornada pela função de crossover e aplica o mecanismo de mutação

Fluxo

```
treinar_percepton:
    para cada geração fazer:
        calcular fitness()
        pop_inter = selecao()
        pop_intermediaria = crossover(pop_inter)
        pop_final = mutacao(pop_intermediaria)
        _pop = pop_final
```

```
class Individuo:
  cromossomo: char*
  aptidao: int
```