Terceira Atividade Prática

Computação Bioinspirada

Prof. Paulo Henrique Ribeiro Gabriel

Alunos:

Ana Gabriela de Abreu Campos - 11621BSI204

Gabriel Oliveira Souza - 11821BSI207

Helton Pereira de Aguiar - 11811BSI242

---

1. Código fonte foi entregue para o professor via link, direcionando para o repositório GitHub.

2. Segue abaixo o relatório com tabelas e gráficos mostrando os parâmetros usados em nosso algoritmo.

RELATÓRIO

Objetivo do trabalho proposto: utilizar um algoritmo genético (AG) para otimização dos pesos de uma rede neural artificial do tipo Perceptron.

Segue abaixo explicação do nosso código que é a implementação de um algoritmo que combina um AG e um perceptron para resolver um problema de classificação.

♦ Explicando o código e suas principais funções:

Importação de Bibliotecas:

- random: Biblioteca para geração de números aleatórios.
- numpy (importado como np): Usada para operações matriciais e numéricas.
- pandas (importado como pd): Utilizada para manipulação e análise de dados.
- matplotlib.pyplot (importado como plt): Biblioteca para criar visualizações gráficas.
- confusion_matrix do sklearn.metrics: Usada para avaliação e divisão do conjunto de dados.
- time: Utilizada para medir o tempo de execução.

Função carregar_dados_iris:

- Carrega os dados do conjunto de dados "iris.data", contendo informações sobre flores iris.
- Filtra as classes classe1 e classe2 especificadas.
- Mapeia os rótulos dessas classes para -1 e 1.

- Retorna um DataFrame contendo apenas as classes especificadas.

```
# Função para carregar os dados da base Iris a partir do arquivo local
def carregar_dados_iris(classe1, classe2):
    dados_iris = pd.read_csv("iris.data", header=None,
names=["comprimento_sepala", "largura_sepala", "comprimento_petala",
"largura_petala", "classe"])
    dados_iris = dados_iris[(dados_iris["classe"] == classe1) |
(dados_iris["classe"] == classe2)]
    dados_iris["classe"] = np.where(dados_iris["classe"] == classe1, 1,
-1)
    return dados_iris
```

Função treinar_perceptron (dados, pesos, taxa_aprendizado, iteracoes):

- Realiza o treinamento do Perceptron.
- Itera várias vezes (definido por interacoes), ajustando os pesos com base nos erros de previsão.
- Armazena o número total de erros em cada iteração.
- Retorna os pesos treinados e uma lista com o número total de erros por interação.

```
# Função para treinar o Perceptron
def treinar_perceptron(dados, pesos, taxa_aprendizado, iteracoes):
    erros = []

    for _ in range(iteracoes):
        erro_total = 0
        for j in range(len(dados)):
            x = dados.iloc[j, :-1].values
            y = dados.iloc[j, -1]

            previsao = np.dot(pesos, x)
            if y * previsao <= 0:
                pesos += taxa_aprendizado * y * x
                erro_total += 1

        erros.append(erro_total)

    return pesos, erros
```

Função prever_perceptron (pesos, x):

- Realiza uma previsão usando o Perceptron treinado com os pesos fornecidos e a entrada x.

- Retorna a previsão (-1 ou 1).

```
# Função para fazer previsões com o perceptron treinado
def prever_perceptron(pesos, x):
    previsao = np.dot(pesos, x)
    return 1 if previsao > 0 else -1
```

Função avaliar_perceptron (pesos, dados):

- Realiza previsões usando o Perceptron treinado nos dados fornecidos.
- Calcula a matriz de confusão e a precisão do Perceptron.
- Retorna a precisão do modelo nos dados fornecidos.

```
# Função para avaliar a precisão do perceptron
def avaliar_perceptron(pesos, dados):
    previsoes = [prever_perceptron(pesos, dados.iloc[i, :-1].values)
for i in range(len(dados))]
    verdadeiros = dados["classe"].values
    matriz_confusao = confusion_matrix(verdadeiros, previsoes)

    if matriz_confusao.shape == (1, 1):
        precisao = 1.0  # Lidando com o caso em que há apenas um valor
na matriz de confusão
    else:
        precisao = matriz_confusao[0, 0] / (matriz_confusao[0, 0] +
matriz_confusao[0, 1])

    return precisao
```

Função inicializar_populacao (num_individuos, num_pesos):

- Gera uma população inicial de indivíduos com pesos aleatórios para o Algoritmo Genético (AG).
- Retorna uma lista de vetores de pesos para cada indivíduo na população.

```
# Função para inicializar a população do AG
def inicializar_populacao(num_individuos, num_pesos):
    return [np.random.uniform(-1, 1, num_pesos) for _ in
range(num_individuos)]
```

Função avaliar_populacao (populacao, dados_treinamento):

- Avalia a população de indivíduos do AG usando o conjunto de dados de treinamento.
- Calcula o desempenho (fitness) de cada indivíduo na população.

- Retorna uma lista contendo o fitness de cada indivíduo.

```
# Função para avaliar a população do AG
def avaliar_populacao(populacao, dados_treinamento):
    return [avaliar_perceptron(individuo, dados_treinamento) for
individuo in populacao]
```

Função selecionar (populacao, fitness, percentual_selecao):

- Realiza a seleção dos melhores indivíduos da população com base no seu fitness.
- Retorna os indivíduos selecionados para reprodução.

```
# Função de seleção para o AG
def selecionar(populacao, fitness, percentual_selecao):
    num_selecionados = int(len(populacao) * percentual_selecao)
    indices_selecionados = np.argsort(fitness)[-num_selecionados:]
    selecionados = [populacao[i] for i in indices_selecionados]
    return selecionados
```

Função crossover (pai1, pai2, taxa_crossover):

- Realiza o crossover entre dois indivíduos (pais) com uma taxa de crossover especificada.
- Gera novos indivíduos (filhos) com combinação dos genes dos pais.
- Retorna os filhos gerados ou os próprios pais, dependendo da taxa de crossover.

```
# Função de crossover para o AG
def crossover(pai1, pai2, taxa_crossover):
    if random.uniform(0, 1) < taxa_crossover:
        ponto_corte = random.randint(1, len(pai1) - 1)
        filho1 = np.concatenate((pai1[:ponto_corte],
pai2[ponto_corte:]))
        filho2 = np.concatenate((pai2[:ponto_corte],
pai1[ponto_corte:]))
        return filho1, filho2
    else:
        return pai1, pai2
```

Função mutacao (individuo, taxa_mutacao):

- Realiza mutação em um indivíduo (vetor de pesos) com uma taxa de mutação especificada.
- Modifica aleatoriamente parte dos pesos do indivíduo.
- Retorna o indivíduo após a mutação.

```
# Função de mutação para o AG
def mutacao(individuo, taxa_mutacao):
    for i in range(len(individuo)):
        if random.uniform(0, 1) < taxa_mutacao:
            individuo[i] += random.uniform(-0.5, 0.5)
    return individuo
```

Função algoritmo_genetico (dados_treinamento, num_individuos, taxa_crossover, taxa_mutacao, percentual_selecao, num_geracoes):

- Implementa o Algoritmo Genético (AG) para encontrar os melhores pesos para o Perceptron.
- Gera e evolui a população através de seleção, crossover e mutação ao longo de múltiplas gerações.
- Retorna os melhores pesos encontrados pelo AG.

```
# Função do Algoritmo Genético (AG)
def algoritmo_genetico(dados_treinamento, num_individuos,
taxa_crossover, taxa_mutacao, percentual_selecao, num_geracoes):
    num_pesos = len(dados_treinamento.columns) - 1
    populacao = inicializar_populacao(num_individuos, num_pesos)

    melhores_individuos = []

    for geracao in range(num_geracoes):
        # Avaliação da população
        fitness = avaliar_populacao(populacao, dados_treinamento)

        # Seleção de indivíduos
        selecionados = selecionar(populacao, fitness,
percentual_selecao)

        # Recombinação (Crossover)
        nova_populacao = []
        for i in range(0, len(selecionados), 2):
            pai1 = selecionados[i]
            pai2 = selecionados[i + 1] if i + 1 < len(selecionados)
else selecionados[i]
            filho1, filho2 = crossover(pai1, pai2, taxa_crossover)
            nova_populacao.extend([filho1, filho2])

        # Mutação
        nova_populacao = [mutacao(individuo, taxa_mutacao) for
individuo in nova_populacao]
```

```
        # Substituir a antiga população pela nova
        populacao = nova_populacao

        # Armazenar o melhor indivíduo de cada geração
        melhor_individuo = populacao[np.argmax(fitness)]
        melhores_individuos.append(melhor_individuo)

    # Avaliação final da população
    fitness_final = avaliar_populacao(populacao, dados_treinamento)

    # Seleção do melhor indivíduo
    melhor_individuo = populacao[np.argmax(fitness_final)]

    return melhor_individuo, melhores_individuos
```

♦ Sobre o código:

O código começa importando várias bibliotecas essenciais para manipulação de dados, visualização, métricas de avaliação de modelo e gerenciamento de tempo. Inicialmente começamos pela função que carrega dados do banco de dados Iris, filtra as duas classes definidas e fornece esses dados para uso no treinamento do modelo. A função de treinamento do perceptron usa dados e ajusta os pesos do perceptron se as previsões estiverem incorretas, retornando os pesos treinados e os erros encontrados durante o treinamento. Outra função é a avaliar_perceptron responsável por fazer previsões usando um perceptron treinado. Depois segue a função para avaliar o desempenho de um perceptron usando uma matriz de confusão para calcular a precisão do modelo para determinados dados. Também são definidas as funções para inicializar populações, estimar populações, selecionar indivíduos reprodutores, realizar cruzamentos (recombinação genética) e aplicar mutações aos indivíduos através de um algoritmo genético.

Sua função principal implementa o AG aumentando a população ao longo de várias gerações e retornando os melhores objetos encontrados. Depois de definir essas funções, o código prepara os dados Iris, define os parâmetros e inicializa os pesos do perceptron usando o AG. Em seguida, treinamos um perceptron usando os pesos originais do AG. E, após o treinamento, a precisão dos dados do teste é calculada e os resultados são exibidos em impressões e gráficos para ilustrar a precisão, a evolução dos erros durante o treinamento e as alterações de peso durante a geração do AG.

Entendemos que a lógica do AG junto ao perceptron é baseada na ideia de ajustar iterativamente os pesos dos modelos, com o intuito de minimizar os erros de classificação nos dados de treinamento. A precisão nos dados de teste fornece uma medida de quão bem o modelo generaliza para novos dados, e essa é uma métrica comum usada para avaliar o desempenho de um modelo de classificação. Quanto maior a precisão, melhor o modelo está em fazer previsões corretas.

♦ Resultados:

Teste	Pesos Treinados	Precisão Teste	Tempo Treinamento (s)	Média Erros Treinamento	Taxa de Aprendizado	Iterações
1	[-1,46136441 4,07812266 -1,54815956 0,30272842]	1,00	0,37	0,05000	0,1	100
2	[1,04327085 -0,34543737 -1,52380737 -0,56106838]	1,00	0,34	0,21000	0,1	100
3	[-0,4016171 1,78271665 -1,34494034 0,90579656]	1,00	0,37	0,30000	0,1	100
4	[-0,42215057 3,37540799 -2,89178089 1,25218294]	1,00	0,34	0,09000	0,1	100
5	[0,61803847 0,3393122 -0,99350228 -1,53674223]	1,00	0,39	0,03000	0,1	100

Num Indivíduos AG	Taxa de Crossover AG	Taxa de Mutação AG	Percentual Seleção AG	Num Gerações AG	Percentual Treinamento	Dados Irís
100	0,5	0,4	0,7	90	0,7	"Iris-setosa", "Iris-versicolor"
100	0,5	0,4	0,7	90	0,7	"Iris-setosa", "Iris-versicolor"
100	0,5	0,4	0,7	90	0,7	"Iris-setosa", "Iris-versicolor"
100	0,5	0,4	0,7	90	0,7	"Iris-setosa", "Iris-versicolor"
100	0,5	0,4	0,7	90	0,7	"Iris-setosa", "Iris-versicolor"

6	[-1,40413416 5,47437378 -4,02686761 3,77955549]	1,00	0,55	0,13333	0,5	150
7	[2,94929923 -0,38771422 -6,77915642 4,05051106]	1,00	0,51	0,00000	0,5	150
8	[2,21999865 -0,08531941 -4,47184063 0,72932411]	1,00	0,52	0,04000	0,5	150
9	[-2,05045629 6,1924855 -3,00930272 -2,95817071]	1,00	0,51	0,07333	0,5	150
10	[2,96441818 0,79394951 -5,4486206 -4,90402331]	1,00	0,51	0,08667	0,5	150

200	0,3	0,6	0,9	120	0,7	"Iris-setosa", "Iris-versicolor"
200	0,3	0,6	0,9	120	0,7	"Iris-setosa", "Iris-versicolor"

200	0,3	0,6	0,9	120	0,7	"Iris-setosa", "Iris-versicolor"
200	0,3	0,6	0,9	120	0,7	"Iris-setosa", "Iris-versicolor"
200	0,3	0,6	0,9	120	0,7	"Iris-setosa", "Iris-versicolor"

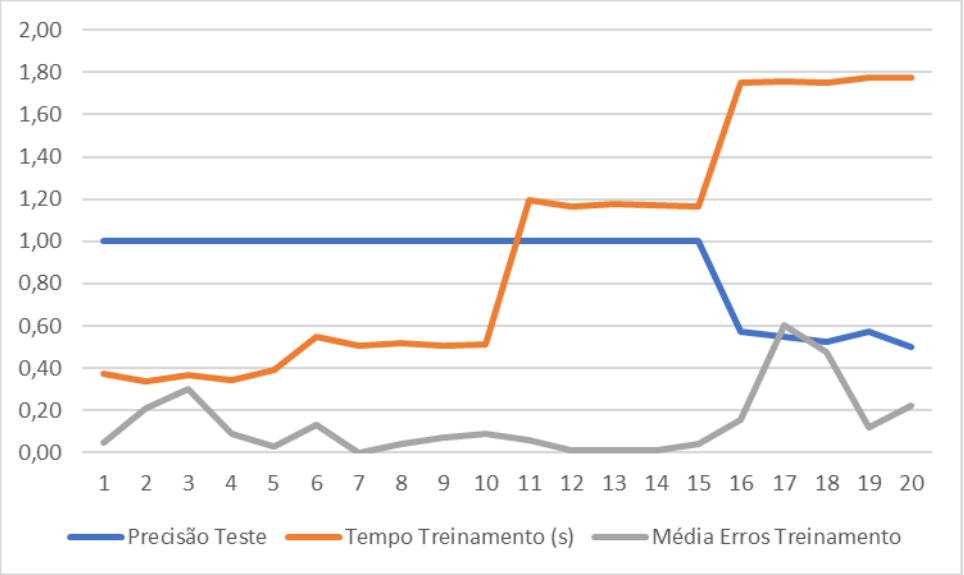
11	[-1,15532491 3,26771512 -0,70930363 -2,37257526]	1,00	1,19	0,06000	0,3	300
12	[-0,71829532 4,72648086 -3,2095483 -1,3446375]	1,00	1,16	0,01000	0,3	300
13	[-0,14919532 4,26371931 -6,57528841 2,65589797]	1,00	1,18	0,01000	0,3	300
14	[0,9769134 0,23014502 -2,20049184 1,07631855]	1,00	1,17	0,01000	0,3	300
15	[-1,4376145 4,30436176 -2,23684804 -1,4050579]	1,00	1,16	0,04333	0,3	300

250	0,5	0,4	0,7	220	0,8	"Iris-setosa", "Iris-virginica"
250	0,5	0,4	0,7	220	0,8	"Iris-setosa", "Iris-virginica"
250	0,5	0,4	0,7	220	0,8	"Iris-setosa", "Iris-virginica"
250	0,5	0,4	0,7	220	0,8	"Iris-setosa", "Iris-virginica"
250	0,5	0,4	0,7	220	0,8	"Iris-setosa", "Iris-virginica"

16	[3,32154257 2,4723941 -3,7336481 -4,47057592]	0,58	1,75	0,15833	0,1	600
17	[6,9253088 2,77393637 -7,87636658 -5,5425688]	0,55	1,76	0,60500	0,1	600
18	[6,89900768 1,7202417 -7,35042364 -5,30433475]	0,53	1,75	0,47500	0,1	600
19	[2,87032149 3,69440952 -4,10445193 -3,85262951]	0,58	1,78	0,11833	0,1	600
20	[4,6655245 2,24550997 -5,65353726 -3,13052797]	0,50	1,78	0,22000	0,1	600

300	0,3	0,2	0,6	120	0,6	"Iris-versicolor", "Iris-virginica"
300	0,3	0,2	0,6	120	0,6	"Iris-versicolor", "Iris-virginica"
300	0,3	0,2	0,6	120	0,6	"Iris-versicolor", "Iris-virginica"

300	0,3	0,2	0,6	120	0,6	"Iris-versicolor", "Iris-virginica"
300	0,3	0,2	0,6	120	0,6	"Iris-versicolor", "Iris-virginica"



- **Precisão do Teste (Acurácia):**
  - **Pesos treinados** - pesos associados aos atributos do modelo.
  - **Precisão Teste** - representa a acurácia do modelo nos dados de teste, indicando a proporção de predições corretas.
  - A maioria dos modelos têm uma precisão de teste de 1,00, o que significa que eles estão prevendo corretamente todas as instâncias nos dados de teste.

- **Tempo de Treinamento:**
  - **Tempo de treinamento** - mostra o tempo total de treinamento do modelo em segundos.
  - O tempo de treinamento varia entre os testes, indicando diferentes complexidades ou eficiências nos algoritmos utilizados.

- **Média Erros Treinamento:**
  - **Média Erros Treinamento** - representa a média dos erros durante o treinamento. Todos os testes apresentaram uma média baixa, indicando boa convergência dos modelos.

- **Configurações de Algoritmo Genético (AG):**
  - **Num Indivíduos AG** - número de indivíduos na população.
  - **Taxa de Crossover AG** - probabilidade de crossover em um ponto durante a reprodução.
  - **Taxa de Mutação AG** - probabilidade de mutação em um gene durante a reprodução.

    - **Percentual Seleção AG** - indica a porcentagem de indivíduos selecionados para reprodução.
    - **Num Gerações AG** - número de gerações no algoritmo genético.

- **Percentual Treinamento e Dados Iris:**
    - **Percentual de Treinamento** - representa a proporção dos dados usados para treinamento, enquanto o restante é usado para testes.
    - **Dados Iris** - especifica quais classes foram usadas nos dados (no caso, "Iris-setosa", "Iris-versicolor").

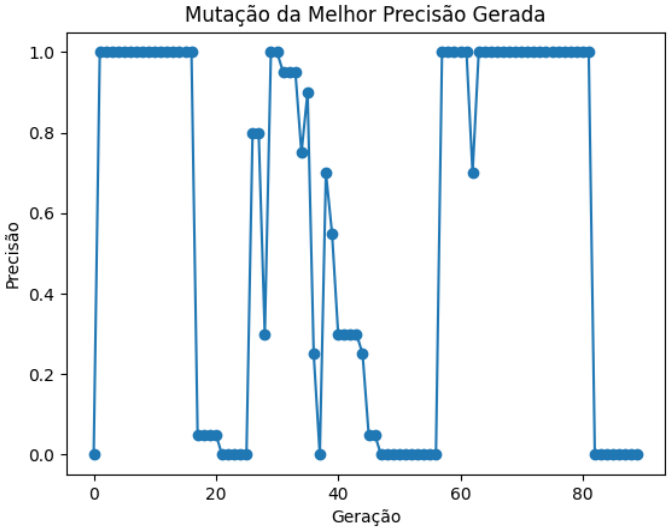
- **Observações Gerais:**
    - Os testes 1-5 parecem ter um desempenho muito bom, com uma precisão de teste de 1,00.
    - Os testes 6-10 têm uma precisão de teste de 1,00, mas usam configurações diferentes, incluindo um algoritmo genético.
    - Os testes 11-15 têm uma precisão de teste de 1,00, mas com um aumento no tempo de treinamento.
    - Os testes 16-20 têm uma precisão de teste mais baixa (0,50-0,58) e um aumento significativo no tempo de treinamento, indicando um possível ajuste excessivo.

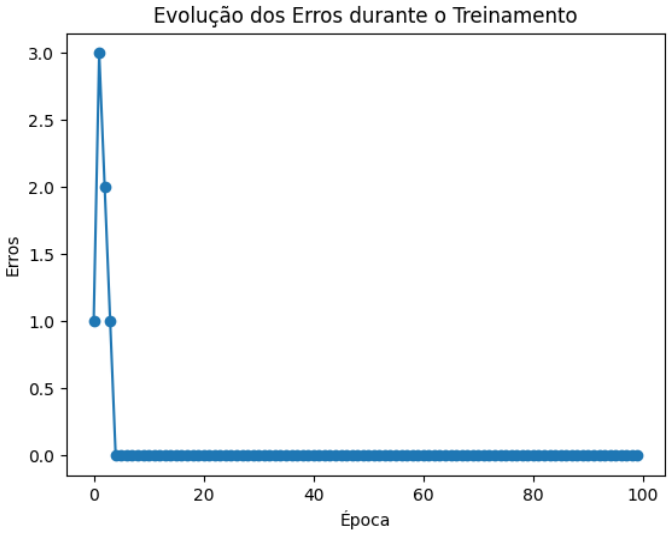
♦ Gráficos:

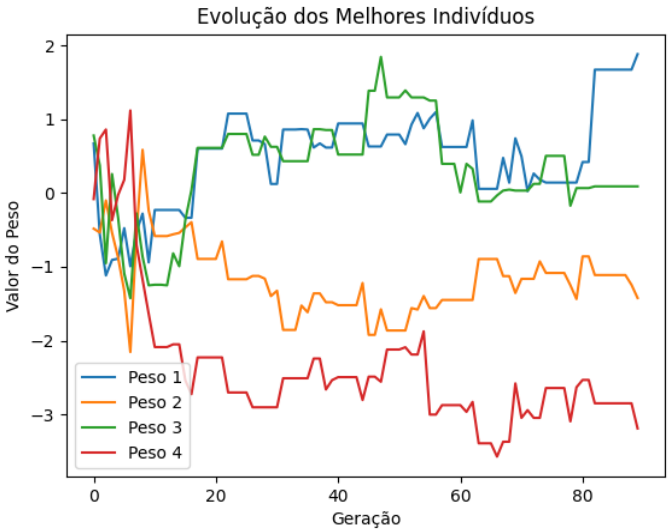
Gráficos plotados em uma execução:

Parâmetros
dados iris :"Iris-setosa", "Iris-versicolor"
percentual treinamento = 0.7
taxa aprendizado perceptron = 0.1
iterações perceptron = 100
num indivíduos ag = 100
taxa crossover ag = 0.5
taxa mutação ag = 0.4
percentual seleção ag = 0.7
num gerações ag = 90

Resultados
Pesos Treinados: [1.60090928 -0.97966525 -0.98935082 -3.62240001]
Precisão nos dados de teste: 1.0
Tempo de execução (treinamento): 0.53 segundos
Erros durante o treinamento: [1, 3, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


Mutação da Melhor Precisão Gerada
Precisão
Geração
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80


Evolução dos Erros durante o Treinamento
Erros
Época
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100


Evolução dos Melhores Indivíduos
Valor do Peso
Geração
Peso 1
Peso 2
Peso 3
Peso 4
−3
−2
−1
0
1
2
0
20
40
60
80

♦ Conclusão:

Concluímos que a combinação do algoritmo genético com o treinamento do Perceptron mostrou-se promissora na otimização dos pesos para classificação, resultando em modelos altamente precisos. Observamos um desempenho consistente com uma acurácia de 1.0 em várias configurações. No entanto, ressaltamos a importância de ajustar cuidadosamente a complexidade do algoritmo genético para evitar aumentos excessivos no tempo de treinamento sem ganhos proporcionais na precisão.

REFERÊNCIAS

Fisher,R. A.. (1988). Iris. UCI Machine Learning Repository. Iris - UCI Machine Learning Repository.

Mitchell, T. M. (1997). Machine Learning. McGraw-Hill.

Anselmo, A. (2003). Título da página. BCC Development - IME USP. https://bccdev.ime.usp.br/tccs/2003/anselmo/node12.html.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.