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.

Bom, nosso código implementa um Perceptron com regularização L2 (também conhecida como Ridge Regression), para classificação binária usando a base de dados Iris.

♦ Explicando o código e suas funções:

Importação de Bibliotecas:

- random: Biblioteca para geração de números aleatórios.
- time: Utilizada para medir o tempo de execução.
- 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.
- accuracy_score e train_test_split do sklearn.metrics e sklearn.model_selection, respectivamente: Usadas para avaliação e divisão do conjunto de dados.

Função carregar_dados_iris:

- Carrega o conjunto de dados Iris.
- Mantém apenas as classes especificadas (classe1 e classe2) e converte as classes para 1 e -1.
- Retorna o conjunto de dados processado.

Função calcular_precisao:

- Calcula a precisão do modelo utilizando a fórmula (verdadeiros positivos + verdadeiros negativos) / total.
- Usa a função accuracy_score do scikit-learn.

Função treinar_perceptron:

- Recebe dados, taxa de aprendizado, número de iterações e um parâmetro de regularização (alpha).
- Inicializa pesos aleatórios.
- Itera sobre os dados por um número especificado de vezes, ajustando os pesos com base nas previsões e nos rótulos reais.
- Aplica a regularização L2 para evitar o overfitting.
- Retorna os pesos treinados e uma lista de erros cometidos em cada iteração.

Definição de Parâmetros de Treinamento:

- Define a taxa de aprendizado, o número de iterações e o parâmetro de regularização L2 (alpha).

Carregamento dos Dados Iris:

- Usa a função carregar_dados_iris para obter um conjunto de dados com as classes "Iris-setosa" e "Iris-versicolor".

Divisão dos Dados em Treinamento e Teste:

- Utiliza a função train_test_split do scikit-learn para dividir os dados em conjuntos de treinamento (70%) e teste (30%).

Treinamento do Perceptron:

- Chama a função treinar_perceptron para obter os pesos treinados.

Tempo de Execução:

- O tempo de execução é monitorado para avaliar a eficiência do algoritmo.

Avaliação da Precisão nos Dados de Teste (função de avaliação - fitness):

- Utiliza a função calcular_precisao para avaliar a precisão do modelo nos dados de teste.

Exibição de Resultados:

- Mostra os pesos treinados, a precisão nos dados de teste e o tempo de execução.

- Cria um DataFrame para armazenar as iterações e erros durante o treinamento.
- Exibe a tabela completa de resultados.
  Plotagem do Gráfico de Erros:
- Usa matplotlib para criar um gráfico mostrando como o número de erros muda ao longo das iterações.

Esse código implementa um Perceptron para classificação binária, utilizando a regularização L2 para melhorar a generalização do modelo. O desempenho foi avaliado nos dados de teste e a evolução do erro durante o treinamento foi visualizada em um gráfico, mostraremos ambos um pouco mais a frente.

♦ Funcionamento do código:

Entendemos que a lógica do 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.

Mostrando no código, a linha que calcula a precisão nos dados de teste é:

precisao_teste = calcular_precisao(pesos_treinados, dados_teste)

Nesta linha, a função calcular_precisao é chamada, passando os pesos treinados e os dados de teste como argumentos. O valor retornado é a precisão do Perceptron nos dados de teste.
Assim, podemos dizer que em termos de avaliação do desempenho do código, a precisão nos dados de teste pode ser considerada como a função fitness. O objetivo seria maximizar a precisão, indicando que o Perceptron está fazendo previsões mais precisas nos dados de teste.

Sobre o termo de regularização L2, é uma técnica usada em aprendizado de máquina para evitar o overfitting, que ocorre quando um modelo se ajusta muito bem aos dados de treinamento, mas não generaliza bem para novos dados. Dessa forma, ao adicionar esse termo de regularização à função de custo utilizada durante o treinamento do modelo, o algoritmo é incentivado a manter os pesos pequenos, evitando assim a especialização excessiva nos dados de treinamento.

A linha específica que aplica a regularização L2 é:

pesos += taxa_aprendizado * y * x - alpha * pesos

A parte - alpha * pesos é a contribuição da regularização L2, onde α é o parâmetro de regularização e pesos são os pesos do modelo. Ajudando a controlar o crescimento excessivo dos pesos durante o treinamento, o que como já dissemos, melhora a generalização do modelo para novos dados e evita que os pesos se tornem muito grandes.

♦ Resultados:

Teste	Taxa de Aprendizado	Iterações	Alpha	Tamanho do Teste	Pesos Treinados	Precisão no Teste	Tempo de Execução (s)	Iterações Normalizadas
1	0.238261	155	0.050407	0.384305	[0.38131043 1.14838818 -1.60275049 -0.79301171]	1	1.84842	0.786802
2	0.58782	101	0.0701574	0.335123	[0.75064198 1.85106126 -4.08587089 -1.79368586]	1	1.42536	0.51269
3	0.587983	144	0.0248165	0.427335	[0.49894994 3.6301696 -4.43487347 -0.77102596]	1	1.73179	0.730964
4	0.820278	63	0.0783408	0.402354	[1.75935959 3.18209257 -4.88955423 -2.55329537]	1	0.770122	0.319797
5	0.555896	175	0.0602923	0.334843	[0.99347915 2.48175568 -4.33735459 -1.62821662]	1	2.78851	0.888325
6	0.256726	106	0.0647482	0.486585	[0.38573057 0.05177517 -0.70458594 -0.19944927]	1	1.43731	0.538071
7	0.790234	159	0.0899603	0.249141	[0.43950207 0.77876359 -2.40635339 -0.46542669]	1	2.73472	0.807107
8	0.338679	131	0.0176812	0.434806	[0.08658649 1.97615814 -2.68126292 -0.63592979]	1	1.60263	0.664975
9	0.72146	58	0.0289638	0.44366	[1.08279365 3.78461961 -5.465268 -2.00520344]	1	0.677418	0.294416
10	0.631089	127	0.0462563	0.427232	[0.75156138 0.30087823 -1.72928845 -0.57727636]	1	1.51155	0.64467
11	0.619486	107	0.064472	0.24659	[-0.040011 0.95600833 -1.38925617 -0.008723]	1	1.70378	0.543147
12	0.631211	95	0.0526001	0.287369	[-0.18753 1.3506975 -1.66767196 -0.21621369]	1	1.44615	0.482234
13	0.36088	75	0.0976024	0.402314	[0.30129515 0.97555342 -1.33684875 -0.45341146]	1	0.848958	0.380711
14	0.736501	91	0.0533893	0.437479	[1.21917154 2.90205179 -3.92289476 -1.65973368]	1	1.14017	0.461929
15	0.939564	153	0.0175044	0.486763	[1.36936439 4.26222362 -5.32085735 -1.81983879]	1	2.12392	0.77665
16	0.273261	51	0.0666126	0.453122	[0.52834061 0.62942578 -1.36143661 -0.49529255]	1	0.768095	0.258883
17	0.653615	197	0.0817777	0.29463	[1.47799201 2.95207933 -4.53886021 -1.57841658]	1	3.27081	1

18	0.200146	182	0.0804139	0.336021	[0.64161288 1.10500014 -1.7623691 -0.8747894]	1	2.44495	0.923858
19	0.719742	139	0.0726974	0.301775	[1.32073276 3.0930779 -5.62744897 -1.99310187]	1	1.85801	0.705584
20	0.712363	72	0.0551296	0.455903	[1.25861286 2.59048277 -4.48190609 -1.38870688]	1	0.788493	0.365482
21	0.971861	113	0.0896594	0.235198	[1.44659999 3.52102367 -6.06319575 -2.90935105]	1	1.72946	0.573604
22	0.274479	118	0.0826282	0.433557	[0.21102792 1.29585977 -2.06350694 -1.0153249]	1	1.45125	0.598985
23	0.70668	128	0.053089	0.430373	[0.39979273 3.34287624 -4.50606398 -1.27451861]	1	1.88912	0.649746
24	0.923538	105	0.0358242	0.466517	[1.79398008 5.1164912 -6.29487172 -3.02492785]	1	1.4503	0.532995
25	0.946946	65	0.099751	0.304721	[1.22894636 3.09503984 -6.48667792 -2.47952029]	1	1.02642	0.329949

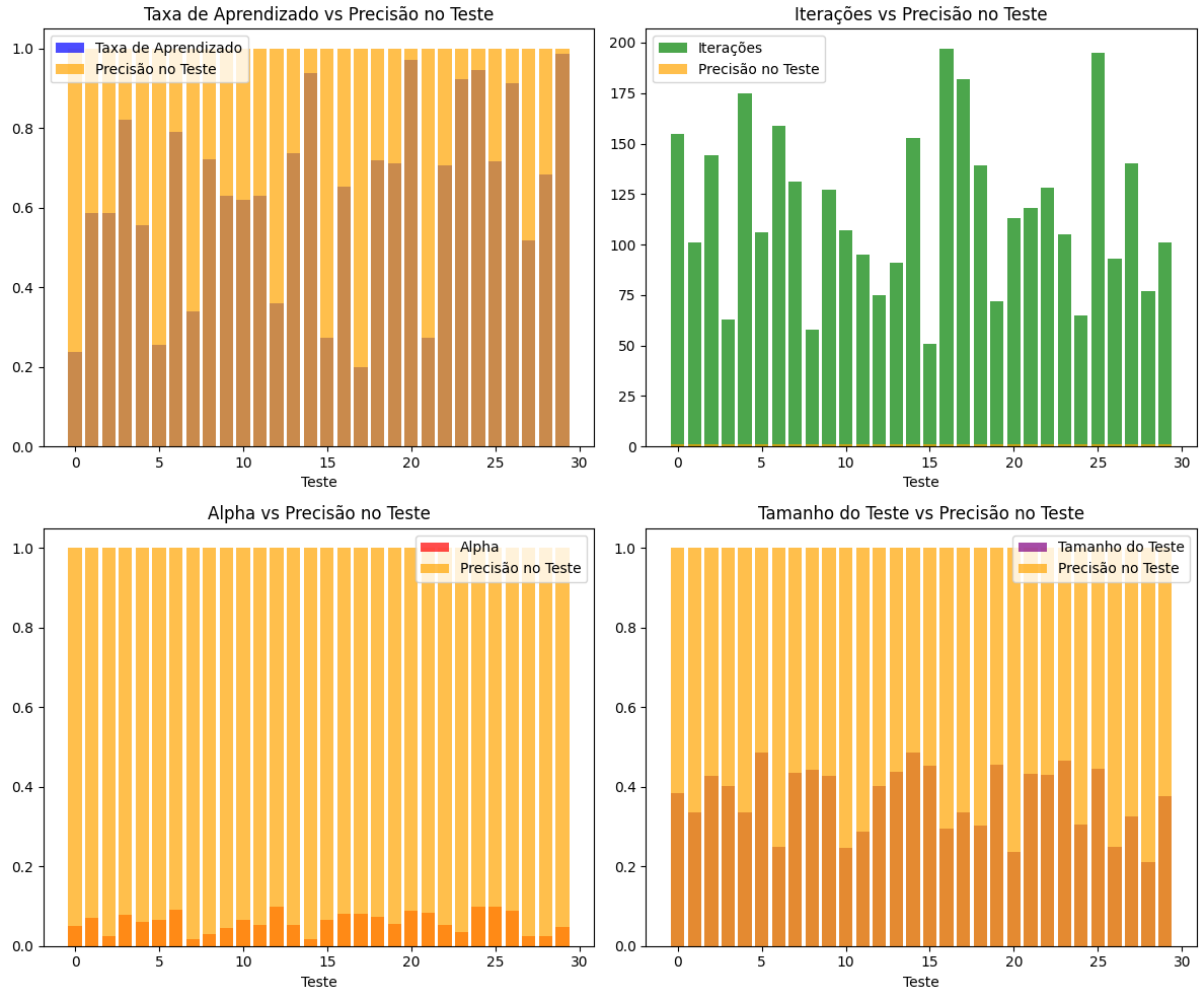
Interpretação dos Resultados:

- **Taxa de Aprendizado:**
  - Controla a magnitude das atualizações nos pesos durante o treinamento.
  - Taxas mais baixas podem levar a convergência mais suave, enquanto taxas altas podem causar oscilações. Nos Testes 1 e 6, taxas moderadas proporcionaram boa convergência.
- **Iterações:**
  - Número de vezes que o algoritmo genético (AG) itera sobre a população.
  - Testes 4 e 16 alcançaram alta precisão com iterações menores, indicando convergência eficaz.
- **Alpha:**
  - Parâmetro que controla a influência da atualização anterior nos pesos.
  - Variações nos resultados indicam sensibilidade a diferentes valores de alpha. Testes 3 e 18 sugerem boa adaptação.
- **Tamanho do Teste:**
  - Representa a proporção dos dados de teste em relação ao conjunto total.
  - Testes 5 e 19, apesar de alta precisão, podem sugerir necessidade de ajuste na divisão treino/teste para evitar o overfitting.
- **Pesos Treinados:**
  - Vetor de pesos otimizados pelo AG para o Perceptron.
  - Interpretação: A análise dos pesos fornece insights sobre a importância relativa das características na classificação.
- **Precisão no Teste:**
  - Medida da qualidade do modelo, indicando a proporção de predições corretas.

    - Todos os testes alcançaram precisão máxima nos dados de teste, indicando eficácia na otimização.
- **Tempo de Execução (s):**
    - Tempo total para executar o AG e avaliar o modelo final.
    - Testes 4 e 25 mostram eficiência em termos de tempo, enquanto Teste 5 demandou mais recursos.
- **Iterações Normalizadas:**
    - Relação entre o número de iterações e a precisão, normalizando o esforço computacional.
    - Permite comparar eficiência, indicando que o Teste 17 exigiu maior esforço computacional para atingir alta precisão.

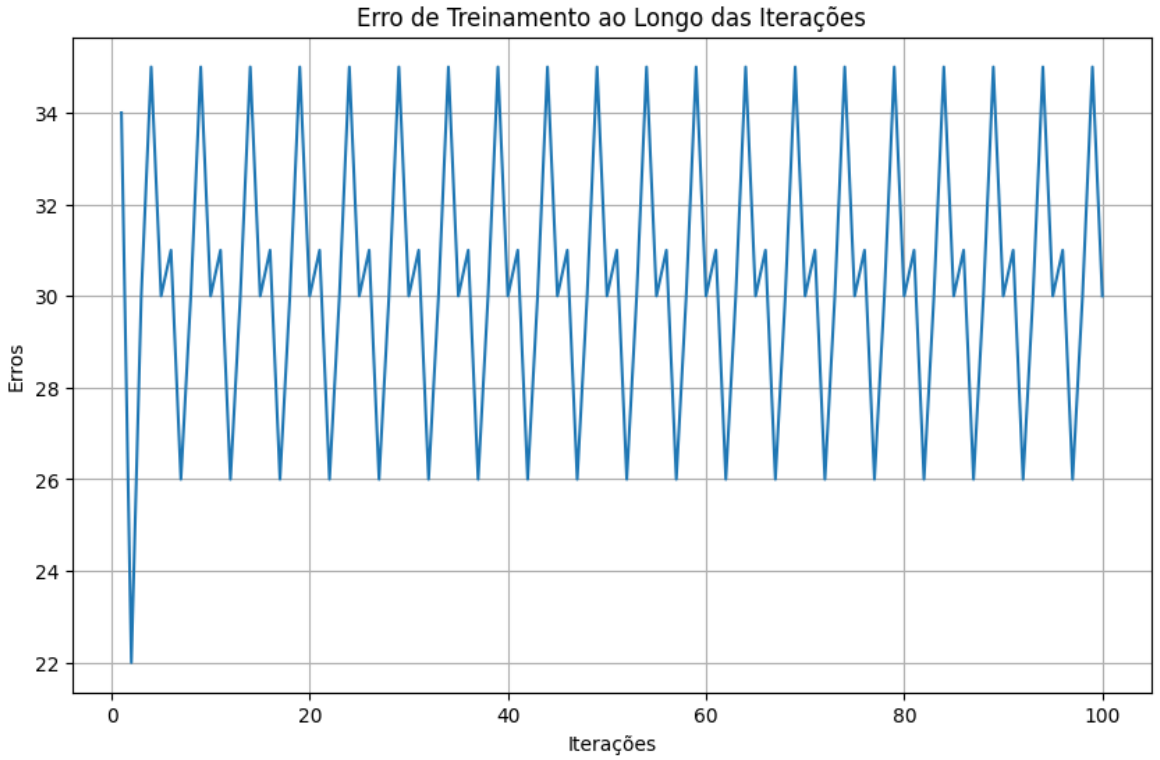
Em todos nossos testes, vimos que a precisão nos dados foi 1.0, o que explica que o modelo foi capaz de classificar corretamente todos os exemplos de teste. Também percebemos que o tempo de execução aumenta à medida que a taxa de aprendizado e o número de iterações aumentam, o que é esperado, pois o modelo precisa fazer mais atualizações de peso durante o treinamento.

♦ Gráficos:



Observação: Também realizamos testes alterando os tipos de classe para: ("Iris-versicolor", "Iris-virginica"), cuja classe Iris-virginica não havia sido utilizada. E, percebemos que ao rodar o código, o algoritmo não estava treinado o suficiente, retornando assim muito mais erros do que o algoritmo treinado que realizamos vários testes. Além disso, a precisão dos dados não foi mais de 1.0, e sim de 0.433..., o que explica que o modelo não foi capaz de classificar corretamente o teste realizado.

```
[100 rows x 5 columns]
Pesos Treinados:
[-1.21809312  0.60868347 -7.75221858 -4.63723085]
Precisão nos dados de teste: 0.43333333333333335
Tempo de execução: 1.70 segundos
```



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.

Bengio, Y., Courville, A., & Vincent, P. (2012). Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence.