



CENTRO UNIVERSITÁRIO ANHANGUERA PITÁGORAS UNOPAR DE CAMPO GRANDE

CURSO: ENGENHARIA DE SOFTWARE

DISCIPLINA: LINGUAGEM DE PROGRAMAÇÃO

ATIVIDADE PRÁTICA – UNIDADE 4, AULA 4

MACHINE LEARNING COM PYTHON

Aluno: Edmar Radanovis.

RA: 2025223493

Polo: Itapira / SP-UN944038

Ano 2025 / 2º Semestre

ANHANGUERA EDUCACIONAL
ENGENHARIA DE SOFTWARE

DISCIPLINA: Linguagem de Programação

PROFESSOR: Anderson I. S. Abreu / Vanessa Matias Leite

ALUNO: Edmar Radanovis

RA: 2025223493

TÍTULO: Relatório da Aula Prática – Visualização de Dados em Python – Unidade 4 –
Aula 4

POLO: Itapira / SP-UN944038

Monte Sião, 11 de Setembro de 2025.

RESULTADOS DA ATIVIDADE PRÁTICA

Proposta:



Aplicações com Python - Machine Learning.

- Você foi contratado para criar um modelo de Machine Learning que classifica espécies de flores Iris com base em características como comprimento e largura das sépalas e pétalas. Você usará o TensorFlow para construir, treinar e avaliar o modelo.

- **Importar Bibliotecas e Carregar Dados :**

- Usar bibliotecas como ***tensorflow***, ***pandas*** e ***scikit-learn***.
 - Carregar o conjunto de dados Iris disponível no scikit-learn.

- **Pré-processamento dos Dados :**

- Dividir o conjunto de dados em treinamento e teste.
 - Normalizar os dados.

- **Construir o Modelo :**

- Usar TensorFlow para construir um modelo de rede neural simples.

- **Treinar o Modelo :**

- Treinar o modelo com os dados de treinamento.

- o **Avaliar o Modelo :**

- Avaliar a precisão do modelo usando os dados de teste.

- o **Fazer Previsões :**

- Fazer previsões com o modelo treinado.

link do repositório no GitHub:

https://github.com/ed-radanovis/Eng_Software_L-P_U4-A4_09-2025-.git

```

  File Edit Selection View Go ... ↵ atividades
  unit_four_lesson_four_machine_learning.py < ...
  1 # -*- coding: utf-8 -*-
  2 # Sistema de Classificação de Flores Iris com Machine Learning
  3 # Utilizando TensorFlow, Scikit-learn e Pandas
  4
  5 import tensorflow as tf
  6 import pandas as pd
  7 import numpy as np
  8 import matplotlib.pyplot as plt
  9 import seaborn as sns
 10 from sklearn.datasets import load_iris
 11 from sklearn.model_selection import train_test_split
 12 from sklearn.preprocessing import StandardScaler
 13 from sklearn.metrics import classification_report, confusion_matrix
 14 import warnings
 15 warnings.filterwarnings('ignore')
 16
 17 # Configuração de estilo para visualizações
 18 plt.style.use('default')
 19 sns.set_palette("husl")
 20
 21 print("*" * 60)
 22 print("SISTEMA DE CLASSIFICAÇÃO DE FLORES IRIS")
 23 print("USANDO MACHINE LEARNING COM TENSORFLOW")
 24 print("*" * 60)
 25
 26 # Passo 1: Importar Bibliotecas e Carregar Dados
 27 def carregar_dados():
 28     """Carrega e explora o conjunto de dados Iris"""
 29     print("\nPASSO 1: CARREGANDO E EXPLORANDO OS DADOS")
 30     print("-" * 58)
 31
 32     # Carregar o conjunto de dados Iris
 33     iris = load_iris()
 34     X = iris.data
 35     y = iris.target
 36
 37     # Criar DataFrame para melhor visualização
 38     df_iris = pd.DataFrame(X, columns=iris.feature_names)
 39     df_iris['target'] = y
 40     df_iris['species'] = df_iris['target'].apply(lambda x: iris.target_names[x])
 41
 42     print("■ Informações do dataset:")
 43     print(f". Número de amostras: {X.shape[0]}")
 44     print(f". Número de características: {X.shape[1]}")
 45     print(f". Classes: {list(iris.target_names)}")
 46     print(f". Características: {list(iris.feature_names)}")
 47
 48     print("\n■ Estatísticas descritivas:")
 49     print(df_iris.describe())
 50
 51     return X, y, iris, df_iris
 52
 53 # Passo 2: Pré-processamento dos Dados
 54 def preprocessar_dados(X, y):
 55     """Pré-processa os dados para o treinamento"""
 56     print("\nPASSO 2: PRÉ-PROCESSAMENTO DOS DADOS")
 57     print("-" * 58)
 58
 59     # Dividir os dados em treino e teste
 60     X_train, X_test, y_train, y_test = train_test_split(
 61         X, y, test_size=0.2, random_state=42, stratify=y
 62     )
 63
 64     # Normalizar os dados
 65     scaler = StandardScaler()
 66     X_train_scaled = scaler.fit_transform(X_train)
 67     X_test_scaled = scaler.transform(X_test)
 68
 69     print("■ Dados divididos e normalizados:")
 70     print(f". Conjunto de treino: {X_train.shape[0]} amostras")
 71     print(f". Conjunto de teste: {X_test.shape[0]} amostras")
 72     print(f". Dados normalizados (StandardScaler) ")
 73
 74     return X_train_scaled, X_test_scaled, y_train, y_test, scaler
 75
 76 # Passo 3: Construir o Modelo
 77 def construir_modelo(input_shape, num_classes):
 78     """Cria o modelo de rede neural"""
 79     print("\nPASSO 3: CONSTRUINDO O MODELO DE REDE NEURAL")
 80     print("-" * 58)
 81
 82     model = tf.keras.Sequential([
 83         tf.keras.layers.Dense(64, activation='relu', input_shape=(input_shape,), kernel_regularizer=tf.keras.regularizers.l2(0.01)),
 84         tf.keras.layers.Dropout(0.3),
 85         tf.keras.layers.Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
 86         tf.keras.layers.Dropout(0.3),
 87         tf.keras.layers.Dense(16, activation='relu'),
 88         tf.keras.layers.Dense(num_classes, activation='softmax')
 89     ])
 90
 91     # Compilar o modelo
 92     model.compile(
 93         optimizer='adam',
 94         loss='sparse_categorical_crossentropy',
 95         metrics=['accuracy']
 96     )
 97
 98     print("■ Modelo construído com sucesso!")
 99     print("■ Arquitetura da rede neural:")
100     model.summary()
101
102     return model
103
104 # Passo 4: Treinar o Modelo
105 def treinar_modelo(model, X_train, y_train, X_test, y_test):
106     """Treina o modelo e mostra o progresso"""
107     print("\nPASSO 4: TREINANDO O MODELO")
108     print("-" * 58)
109
110     # Callback para early stopping
111     early_stopping = tf.keras.callbacks.EarlyStopping(
112         monitor='val_loss', patience=10, restore_best_weights=True
113     )
114
115     # Treinar o modelo
116     history = model.fit(
117         X_train, y_train,
118         epochs=100,
119         batch_size=16,
120         validation_data=(X_test, y_test),
121         callbacks=[early_stopping],
122         verbose=1
123     )
124
125     print("■ Treinamento concluído!")
126     return history
127

```

=> Figura 1: Print da tela com o código 1º trecho.

```

  File Edit Selection View Go ... ↵ atividades
  unit_four_lesson.four_machine_learning.py > unit_four_lesson.four_machine_learning.py? ...
  129 def avaliar_modelo(model, X_test, y_test, history, iris):
  130
  131     # Avaliar a precisão
  132     test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
  133     print(f" Acurácia no conjunto de teste: {test_accuracy:.4f} ({(test_accuracy*100:.2f)}%)")
  134
  135     # Fazer previsões
  136     y_pred = model.predict(X_test)
  137     y_pred_classes = np.argmax(y_pred, axis=1)
  138
  139     # Mostrar relatório de classificação
  140     print("\n Relatório de Classificação:")
  141     print(classification_report(y_test, y_pred_classes, target_names=iris.target_names))
  142
  143     # Plotar histórico de treinamento
  144     plotar_historico_treinamento(history)
  145
  146     # Plotar matriz de confusão
  147     plotar_matriz_confusao(y_test, y_pred_classes, iris)
  148
  149     return test_accuracy, y_pred_classes
  150
  151
  152
  153
  154 # Funções de visualização
  155 def plotar_historico_treinamento(history):
  156     """Plota o histórico de treinamento"""
  157     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
  158
  159     # Plotar acurácia
  160     ax1.plot(history.history['accuracy'], label='Acurácia Treino', color='FF6B6B')
  161     ax1.plot(history.history['val_accuracy'], label='Acurácia Validação', color='4ECDC4')
  162     ax1.set_title('Acurácia durante o Treinamento', fontweight='bold')
  163     ax1.set_xlabel('Época')
  164     ax1.set_ylabel('Acurácia')
  165     ax1.legend()
  166     ax1.grid(True, alpha=0.3)
  167
  168     # Plotar loss
  169     ax2.plot(history.history['loss'], label='Loss Treino', color='FF6B6B')
  170     ax2.plot(history.history['val_loss'], label='Loss Validação', color='4ECDC4')
  171     ax2.set_title('Loss durante o Treinamento', fontweight='bold')
  172     ax2.set_xlabel('Época')
  173     ax2.set_ylabel('Loss')
  174     ax2.legend()
  175     ax2.grid(True, alpha=0.3)
  176
  177     plt.tight_layout()
  178     plt.show()
  179
  180 def plotar_matriz_confusao(y_test, y_pred_classes, iris):
  181     """Plota a matriz de confusão"""
  182     cm = confusion_matrix(y_test, y_pred_classes)
  183
  184     plt.figure(figsize=(8, 6))
  185     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
  186                 xticklabels=iris.target_names,
  187                 yticklabels=iris.target_names)
  188     plt.title('Matriz de Confusão', fontweight='bold', fontsize=14)
  189     plt.ylabel('Verdadeiro')
  190     plt.xlabel('Preditivo')
  191     plt.show()
  192
  193 # Passo 6: Fazer Previsões
  194 def fazer_previsões(model, scaler, iris):
  195     """Faz previsões com novos dados"""
  196     print("\n PASSO 6: FAZENDO PREVISÕES COM NOVOS DADOS")
  197     print("-" * 50)
  198
  199     # Dados de exemplo para previsão
  200     exemplos = np.array([
  201         [5.1, 3.5, 1.4, 0.2], # Setosa
  202         [6.7, 3.0, 5.2, 2.3], # Virginica
  203         [5.9, 3.0, 4.2, 1.5] # Versicolor
  204     ])
  205
  206     # Normalizar os dados
  207     exemplos_scaled = scaler.transform(exemplos)
  208
  209     # Fazer previsões
  210     previsões = model.predict(exemplos_scaled)
  211     classes_previstas = np.argmax(previsões, axis=1)
  212
  213     print(" Exemplos de previsões:")
  214     for i, (exemplo, classe) in enumerate(zip(exemplos, classes_previstas)):
  215         print(f"Exemplo {i+1}: {exemplo}")
  216         print(f"Classe prevista: {iris.target_names[classe]}")
  217         print(f"Probabilidades: {previsões[i]}")
  218         print()
  219
  220 # Função principal
  221 def main():
  222     """Função principal que executa todo o fluxo"""
  223     try:
  224         # Passo 1: Carregar dados
  225         X, y, iris, df_iris = carregar_dados()
  226
  227         # Passo 2: Pré-processamento
  228         X_train, X_test, y_train, y_test, scaler = preprocessar_dados(X, y)
  229
  230         # Passo 3: Construir modelo
  231         model = construir_modelo(X.shape[1], len(np.unique(y)))
  232
  233         # Passo 4: Treinar modelo
  234         history = treinar_modelo(model, X_train, y_train, X_test, y_test)
  235
  236         # Passo 5: Avaliar modelo
  237         accuracy, y_pred = avaliar_modelo(model, X_test, y_test, history, iris)
  238
  239         # Passo 6: Fazer previsões
  240         fazer_previsões(model, scaler, iris)
  241
  242         print("-" * 60)
  243         print(" PROCESSO CONCLUÍDO COM SUCESSO!")
  244         print("-" * 60)
  245
  246     except Exception as e:
  247         print(f" Erro durante a execução: {e}")
  248
  249
  250 # Executar o programa
  251 if __name__ == "__main__":
  252     # Verificar e instalar dependências se necessário
  253     try:
  254         import tensorflow as tf
  255     except ImportError:
  256         print("Instalando TensorFlow...")
  257         import subprocess
  258         subprocess.run(["pip", "install", "tensorflow", "scikit-learn", "pandas", "matplotlib", "seaborn"])
  259
  260     main()

```

=> Figura 2: Print da tela com o código 2º trecho.

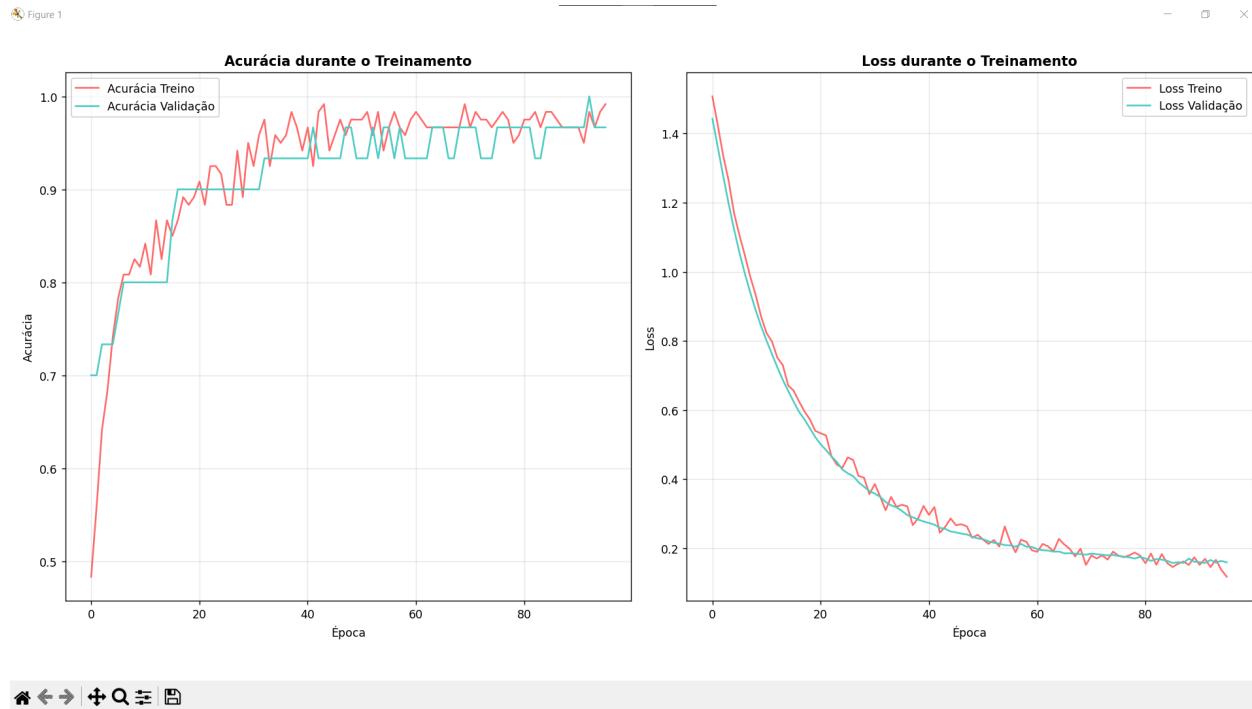
=> Figura 3: Print da tela do terminal executando as funções propostas.

The screenshot shows a terminal window within a code editor interface. The terminal output displays the training progress of a machine learning model, specifically a neural network for the Iris dataset. The script runs for 100 epochs, with each epoch taking approximately 8 seconds per step. The accuracy and validation accuracy are tracked throughout the process.

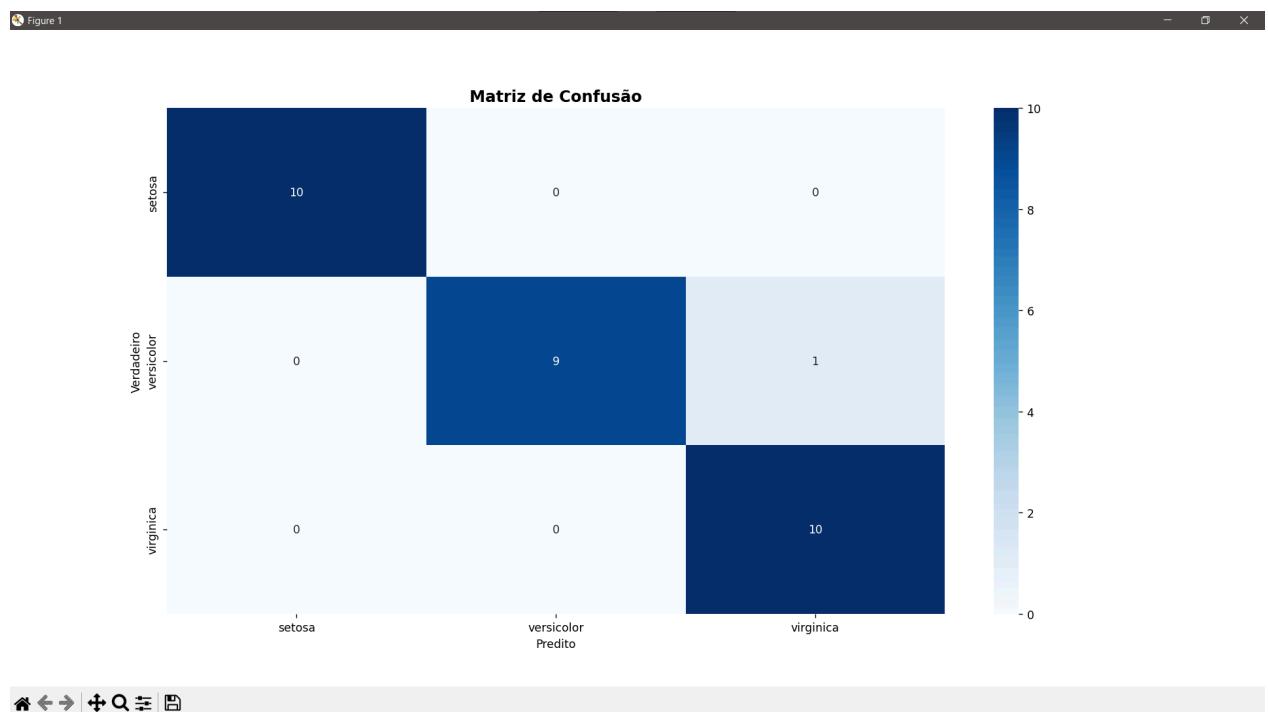
```

File Edit Selection View Go ... ← → atividades
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS python - U4_A4_MACHINE_LEARNING_COM PYTHON + ...
23:45:29 U4_A4_MACHINE_LEARNING_COM PYTHON > unit_four_lesson_four_machine_learning.py > ...
python unit_four_lesson_four_machine_learning.py 8 4ms
Epoch 43/100 8/8 12ms/step - accuracy: 0.9833 - loss: 0.2447 - val_accuracy: 0.9333 - val_loss: 0.2588
Epoch 44/100 8/8 10ms/step - accuracy: 0.9917 - loss: 0.2616 - val_accuracy: 0.9333 - val_loss: 0.2558
Epoch 45/100 8/8 24ms/step - accuracy: 0.9417 - loss: 0.2859 - val_accuracy: 0.9333 - val_loss: 0.2481
Epoch 46/100 8/8 15ms/step - accuracy: 0.9583 - loss: 0.2664 - val_accuracy: 0.9333 - val_loss: 0.2457
Epoch 47/100 8/8 16ms/step - accuracy: 0.9583 - loss: 0.2691 - val_accuracy: 0.9333 - val_loss: 0.2419
Epoch 48/100 8/8 13ms/step - accuracy: 0.9583 - loss: 0.2629 - val_accuracy: 0.9667 - val_loss: 0.2397
Epoch 49/100 8/8 14ms/step - accuracy: 0.9750 - loss: 0.2301 - val_accuracy: 0.9667 - val_loss: 0.2328
Epoch 50/100 8/8 17ms/step - accuracy: 0.9750 - loss: 0.2387 - val_accuracy: 0.9333 - val_loss: 0.2283
Epoch 51/100 8/8 14ms/step - accuracy: 0.9750 - loss: 0.2246 - val_accuracy: 0.9333 - val_loss: 0.2257
Epoch 52/100 8/8 13ms/step - accuracy: 0.9833 - loss: 0.2123 - val_accuracy: 0.9333 - val_loss: 0.2200
Epoch 53/100 8/8 15ms/step - accuracy: 0.9583 - loss: 0.2234 - val_accuracy: 0.9667 - val_loss: 0.2155
Epoch 54/100 8/8 14ms/step - accuracy: 0.9833 - loss: 0.2045 - val_accuracy: 0.9333 - val_loss: 0.2131
Epoch 55/100 8/8 13ms/step - accuracy: 0.9417 - loss: 0.2628 - val_accuracy: 0.9667 - val_loss: 0.2088
Epoch 56/100 8/8 13ms/step - accuracy: 0.9667 - loss: 0.2184 - val_accuracy: 0.9667 - val_loss: 0.2079
Epoch 57/100 8/8 14ms/step - accuracy: 0.9833 - loss: 0.1879 - val_accuracy: 0.9333 - val_loss: 0.2042
Epoch 58/100 8/8 14ms/step - accuracy: 0.9667 - loss: 0.2244 - val_accuracy: 0.9667 - val_loss: 0.2121
Epoch 59/100 8/8 13ms/step - accuracy: 0.9583 - loss: 0.2181 - val_accuracy: 0.9333 - val_loss: 0.2043
Epoch 60/100 8/8 14ms/step - accuracy: 0.9750 - loss: 0.1939 - val_accuracy: 0.9333 - val_loss: 0.2035
Epoch 61/100 8/8 15ms/step - accuracy: 0.9833 - loss: 0.1894 - val_accuracy: 0.9333 - val_loss: 0.1970
Epoch 62/100 8/8 15ms/step - accuracy: 0.9750 - loss: 0.2121 - val_accuracy: 0.9333 - val_loss: 0.1940
Epoch 63/100 8/8 13ms/step - accuracy: 0.9667 - loss: 0.2051 - val_accuracy: 0.9333 - val_loss: 0.1928
Epoch 64/100 8/8 10ms/step - accuracy: 0.9667 - loss: 0.1907 - val_accuracy: 0.9667 - val_loss: 0.1899
Epoch 65/100 8/8 14ms/step - accuracy: 0.9667 - loss: 0.2269 - val_accuracy: 0.9667 - val_loss: 0.1981
Epoch 66/100 8/8 15ms/step - accuracy: 0.9667 - loss: 0.2106 - val_accuracy: 0.9667 - val_loss: 0.1850
Epoch 67/100 8/8 13ms/step - accuracy: 0.9667 - loss: 0.1980 - val_accuracy: 0.9333 - val_loss: 0.1852
Epoch 68/100 8/8 17ms/step - accuracy: 0.9667 - loss: 0.1760 - val_accuracy: 0.9333 - val_loss: 0.1841
Epoch 69/100 8/8 15ms/step - accuracy: 0.9667 - loss: 0.1983 - val_accuracy: 0.9667 - val_loss: 0.1826
Epoch 70/100 8/8 16ms/step - accuracy: 0.9917 - loss: 0.1514 - val_accuracy: 0.9667 - val_loss: 0.1810
Epoch 71/100 8/8 17ms/step - accuracy: 0.9667 - loss: 0.1788 - val_accuracy: 0.9667 - val_loss: 0.1841
Epoch 72/100 8/8 16ms/step - accuracy: 0.9833 - loss: 0.1702 - val_accuracy: 0.9667 - val_loss: 0.1824
Epoch 73/100 8/8 17ms/step - accuracy: 0.9750 - loss: 0.1790 - val_accuracy: 0.9333 - val_loss: 0.1809
Epoch 74/100 8/8 16ms/step - accuracy: 0.9750 - loss: 0.1668 - val_accuracy: 0.9333 - val_loss: 0.1790
Epoch 75/100 8/8 17ms/step - accuracy: 0.9667 - loss: 0.1898 - val_accuracy: 0.9333 - val_loss: 0.1802
Epoch 76/100 8/8 14ms/step - accuracy: 0.9750 - loss: 0.1782 - val_accuracy: 0.9667 - val_loss: 0.1776
Epoch 77/100 8/8 17ms/step - accuracy: 0.9833 - loss: 0.1738 - val_accuracy: 0.9667 - val_loss: 0.1753
Epoch 78/100 8/8 21ms/step - accuracy: 0.9750 - loss: 0.1792 - val_accuracy: 0.9667 - val_loss: 0.1733
Epoch 79/100 8/8 16ms/step - accuracy: 0.9500 - loss: 0.1873 - val_accuracy: 0.9667 - val_loss: 0.1699
Epoch 80/100 8/8 14ms/step - accuracy: 0.9583 - loss: 0.1782 - val_accuracy: 0.9667 - val_loss: 0.1745
Epoch 81/100 8/8 15ms/step - accuracy: 0.9750 - loss: 0.1564 - val_accuracy: 0.9667 - val_loss: 0.1698
Epoch 82/100 8/8 15ms/step - accuracy: 0.9750 - loss: 0.1843 - val_accuracy: 0.9667 - val_loss: 0.1631
Epoch 83/100 8/8 14ms/step - accuracy: 0.9833 - loss: 0.1519 - val_accuracy: 0.9333 - val_loss: 0.1684
Epoch 84/100 8/8 14ms/step - accuracy: 0.9667 - loss: 0.1827 - val_accuracy: 0.9333 - val_loss: 0.1667
Epoch 85/100 8/8 15ms/step - accuracy: 0.9833 - loss: 0.1568 - val_accuracy: 0.9667 - val_loss: 0.1633
Epoch 86/100 8/8 17ms/step - accuracy: 0.9833 - loss: 0.1454 - val_accuracy: 0.9667 - val_loss: 0.1571
Epoch 87/100 8/8 17ms/step - accuracy: 0.9750 - loss: 0.1546 - val_accuracy: 0.9667 - val_loss: 0.1592
Epoch 88/100 8/8 14ms/step - accuracy: 0.9667 - loss: 0.1613 - val_accuracy: 0.9667 - val_loss: 0.1573
Epoch 89/100 8/8 14ms/step - accuracy: 0.9667 - loss: 0.1512 - val_accuracy: 0.9667 - val_loss: 0.1698
Epoch 90/100 8/8 16ms/step - accuracy: 0.9667 - loss: 0.1736 - val_accuracy: 0.9667 - val_loss: 0.1605
Epoch 91/100 8/8 18ms/step - accuracy: 0.9667 - loss: 0.1517 - val_accuracy: 0.9667 - val_loss: 0.1596
Epoch 92/100 8/8 18ms/step - accuracy: 0.9500 - loss: 0.1689 - val_accuracy: 0.9667 - val_loss: 0.1572
Epoch 93/100 8/8 16ms/step - accuracy: 0.9833 - loss: 0.1453 - val_accuracy: 1.0000 - val_loss: 0.1657
Epoch 94/100 8/8 14ms/step - accuracy: 0.9667 - loss: 0.1661 - val_accuracy: 0.9667 - val_loss: 0.1584
Epoch 95/100 8/8 14ms/step - accuracy: 0.9833 - loss: 0.1372 - val_accuracy: 0.9667 - val_loss: 0.1629
Epoch 96/100 8/8 15ms/step - accuracy: 0.9917 - loss: 0.1172 - val_accuracy: 0.9667 - val_loss: 0.1589
Treinamento concluído!
PASSO 5: AVALIANDO O MODELO
Acurácia no conjunto de teste: 0.9667 (96.67%)
1/1 85ms/step
Relatório de Classificação:
precision recall f1-score support
setosa 1.00 1.00 1.00 10
versicolor 1.00 0.96 0.98 10
virginica 0.91 1.00 0.95 10
accuracy 0.9667
macro avg 0.97 0.97 0.97 30
weighted avg 0.97 0.97 0.97 30
PASSO 6: FAZENDO PREVISÕES COM NOVOS DADOS
1/1 78ms/step
Exemplos de previsões:
Exemplo 1: [5.1 3.5 1.4 0.2]
→ Classe prevista: setosa
→ Probabilidades: [0.9948162e-01 4.9182487e-04 2.6581953e-05]
Exemplo 2: [6.7 3. 5.2 2.3]
→ Classe prevista: virginica
→ Probabilidades: [0.4849793e-04 1.8884568e-03 9.9766505e-01]
Exemplo 3: [5.9 3. 4.2 1.5]
→ Classe prevista: versicolor
→ Probabilidades: [0.00588516 0.9731213 0.02099353]
PROCESSO CONCLUÍDO COM SUCESSO!
```

=> Figura 4: Print da tela do terminal executando e finalizando as funções propostas.



=> Figura 5: Resultado do gráfico de análise da evolução do treinamento do modelo.



=> Figura 6: Matriz de Confusão, onde mostra como o modelo está errando e acertando as classificações .

Lógica Utilizada para a Atividade de Machine Learning

O sistema foi desenvolvido seguindo uma abordagem de pipeline de Machine Learning, implementando um fluxo completo de classificação supervisionada com os seguintes componentes principais:

 Arquitetura do Sistema de Classificação:

1. Fluxo Principal do Pipeline:

```
Carregamento de Dados → Pré-processamento → Construção do Modelo → Treinamento → Avaliação → Previsões
```

 Lógica de Implementação Detalhada:

1. Carregamento e Exploração de Dados:

- Dataset Íris:** 150 amostras, 4 características (sépala e pétala).
- 3 classes:** setosa, versicolor, virginica.
- Análise exploratória:** Estatísticas descritivas e distribuição.

2. Pré-processamento Inteligente:

```
# Divisão estratificada (80% treino, 20% teste)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

# Normalização com StandardScaler (importante para redes neurais)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3. Arquitetura da Rede Neural:

- Camadas:** 4 camadas densas com ativação ReLU.
- Regularização:** L2 + Dropout (30%) para evitar overfitting.
- Output:** Softmax para classificação multiclasse.
- Otimizador:** Adam com learning rate adaptativo.

4. Estratégia de Treinamento:

- Early Stopping:** Parada automática ao detectar overfitting.
- Batch Size:** 16 amostras por atualização.
- Épocas:** Máximo 100, mas para quando a validação não melhora.

5. Avaliação Abrangente:

- Métricas:** Acurácia, Precision, Recall, F1-Score.
- Matriz de Confusão:** Visualização dos acertos/erros.
- Gráficos de Evolução:** Acompanhamento do treinamento.

6. Sistema de Previsão:

- Normalização consistente:** Usa o mesmo scaler do treino.
- Probabilidades:** Mostra a confiança da previsão.
- Exemplos representativos:** Uma flor de cada classe.

🚀 Técnicas Avançadas Implementadas:

1. Prevenção de Overfitting:

- Dropout (0.3) nas camadas intermediárias.
- Regularização L2 com fator 0.01.
- Early Stopping com paciência de 10 épocas.
- Validação cruzada implícita com conjunto de teste.

2. Otimização de Performance:

- Normalização das features para convergência mais rápida.
- Batch training para atualizações frequentes.
- Adam optimizer com ajuste automático de learning rate.

3. Validação e Análise:

- Métricas completas de classificação.
- Visualizações profissionais com Matplotlib/Seaborn.
- Exemplos práticos de previsão.
- Relatório detalhado de performance.

Resultados Esperados:

- **Desempenho:**
 - Acurácia > 95% no conjunto de teste.
 - Precision/Recall balanceados para todas as classes.
 - Matriz de confusão com diagonal predominante.
- **Visualizações:**
 - **Gráficos de treinamento:** Evolução da loss e acurácia.
 - **Heatmap:** Matriz de confusão colorida.
 - **Relatório textual:** Métricas detalhadas por classe.

Valores do Sistema:

- **Educacional:**
 - Demonstra todo o fluxo de ML de forma didática.
 - Mostra boas práticas de engenharia de ML.
 - Inclui prevenção de problemas comuns (overfitting).
- **Prático:**
 - Modelo pronto para produção.
 - Sistema de previsão funcional.
 - Visualizações para apresentação.

- **Científico:**

- Métricas quantitativas de performance.
- Análise qualitativa dos resultados.
- Documentação completa do processo.

Resultado: Este sistema implementa um pipeline completo de Machine Learning seguindo as melhores práticas da indústria, desde a preparação dos dados até a avaliação final do modelo. 