Aplicações em Python

Atividade Prática – Machine Learning com Python

1. **Importação de Bibliotecas:** As bibliotecas necessárias são importadas: pandas e numpy para manipulação de dados.
   scikit-learn para carregar e preparar o conjunto de dados.
   tensorflow para construir, treinar e avaliar o modelo de Machine Learning.

2. **Carregamento dos Dados:** O conjunto de dados Iris é carregado utilizando a função load_iris() da biblioteca scikit-learn, que contém informações sobre as características das flores e suas respectivas espécies.

3. **Pré-processamento dos Dados:** Divisão do Conjunto de Dados: Os dados são divididos em conjuntos de treinamento (80%) e teste (20%) utilizando train_test_split(), garantindo que o modelo possa ser avaliado em dados que não viu durante o treinamento.
   Normalização dos Dados: Os dados de entrada são normalizados utilizando StandardScaler() para que todas as características tenham uma escala similar, melhorando a eficiência do treinamento.

4. **Construção do Modelo:** Um modelo de rede neural é criado utilizando a API do TensorFlow:
   - A primeira camada oculta possui 10 neurônios e utiliza a função de ativação relu.
   - A segunda camada oculta também possui 10 neurônios e utiliza relu.
   - A camada de saída tem 3 neurônios (uma para cada espécie) e utiliza a função de ativação softmax para calcular as probabilidades de cada classe.

5. **Treinamento do Modelo:** O modelo é compilado com um otimizador (adam) e uma função de perda (sparse_categorical_crossentropy), e em seguida, é treinado com os dados de treinamento por um número determinado de épocas, permitindo que o modelo aprenda a classificar as flores.

   Avaliação do Modelo: A precisão do modelo é avaliada utilizando os dados de teste, fornecendo uma métrica de desempenho que indica a porcentagem de previsões corretas.

6. **Previsões com o Modelo Treinado:** O modelo treinado é utilizado para fazer previsões sobre os dados de teste, permitindo comparar as classes previstas com as classes reais para verificar a eficácia do modelo.

Unidade 4 – Gustavo Nascimento Cazzine

```python
# Importar bibliotecas necessárias
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Carregar o conjunto de dados Iris
iris = load_iris()
X = iris.data  # características (comprimento e largura das sépalas e pétalas)
y = iris.target  # rótulos (espécies de flores)

# Dividir o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalizar os dados
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Construir o modelo de rede neural
model = keras.Sequential([
    layers.Dense(10, activation='relu', input_shape=(X_train.shape[1],)),  # camada oculta
    layers.Dense(10, activation='relu'),  # outra camada oculta
    layers.Dense(3, activation='softmax')  # camada de saída para 3 classes
])

# Compilar o modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Treinar o modelo com os dados de treinamento
model.fit(X_train, y_train, epochs=50)  # você pode ajustar o número de épocas conforme necessário

# Avaliar a precisão do modelo usando os dados de teste
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Precisão do modelo: {accuracy * 100:.2f}%')

# Fazer previsões com o modelo treinado
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)  # converter as previsões em classes
print(f'Classes previstas: {predicted_classes}')
print(f'Classes reais: {y_test}')
```

Unidade 4

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.4700 - loss: 0.9255
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5248 - loss: 0.8881
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5306 - loss: 0.9155
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5585 - loss: 0.8943
Epoch 5/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5494 - loss: 0.9108
Epoch 6/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5723 - loss: 0.8945
Epoch 7/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6313 - loss: 0.8908
Epoch 8/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6527 - loss: 0.8543
Epoch 9/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6829 - loss: 0.8684
Epoch 10/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6898 - loss: 0.8647
Epoch 11/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7006 - loss: 0.8651
Epoch 12/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7317 - loss: 0.8628
Epoch 13/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7933 - loss: 0.8081
Epoch 14/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7727 - loss: 0.8080
Epoch 15/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.7987 - loss: 0.8147
Epoch 16/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8550 - loss: 0.8015
Epoch 17/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8479 - loss: 0.7965
Epoch 18/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8229 - loss: 0.7778
Epoch 19/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8492 - loss: 0.7881
Epoch 20/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8248 - loss: 0.8047
Epoch 21/50
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8817 - loss: 0.6417
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8777 - loss: 0.6702
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8873 - loss: 0.6565
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8496 - loss: 0.6600
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8465 - loss: 0.6643
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8735 - loss: 0.6199
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8537 - loss: 0.6390
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8527 - loss: 0.6382
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8790 - loss: 0.5932
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8687 - loss: 0.6019
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8844 - loss: 0.5761
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8833 - loss: 0.5579
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8865 - loss: 0.5527
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8854 - loss: 0.5531
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8865 - loss: 0.5370
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8896 - loss: 0.5338
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8729 - loss: 0.5128
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8950 - loss: 0.5103
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8731 - loss: 0.4984
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8627 - loss: 0.4843
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 484ms/step - accuracy: 0.9333 - loss: 0.4115
Precisão do modelo: 93.33%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
Classes previstas: [1 0 2 1 2 0 1 2 1 1 2 0 0 0 0 2 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
Classes reais: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```