Anngladys Winding up 2e937b0 · 2 minutes ago

248 lines (248 loc) · 8.87 KB

Preview Code Blame Raw

In [ ]:

In [1]:
```
# Cell 1: Import Libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore') # Ignore warnings, especially for precis
```

In [2]:
```
# Cell 2: Load and Preprocess Data
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()

print(f"Original Training data shape: {X_train.shape}, Labels shape: {y_tr
print(f"Original Testing data shape: {X_test.shape}, Labels shape: {y_test

# Normalize pixel values to [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Reshape images to (height, width, channels) - MNIST is grayscale, so cha
X_train = np.expand_dims(X_train, -1) # Adds a channel dimension
X_test = np.expand_dims(X_test, -1)

print(f"\nNormalized and Reshaped Training data shape: {X_train.shape}")
print(f"Normalized and Reshaped Testing data shape: {X_test.shape}")

# One-hot encode the labels
num_classes = 10
y_train_one_hot = to_categorical(y_train, num_classes)
y_test_one_hot = to_categorical(y_test, num_classes)

print(f"One-hot encoded Training labels shape: {y_train_one_hot.shape}")
print(f"One-hot encoded Testing labels shape: {y_test_one_hot.shape}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 3
      1 # Cell 2: Load and Preprocess Data
      2 # Load the MNIST dataset
----> 3 (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_
data()
      5 print(f"Original Training data shape: {X_train.shape}, Labels shape:
{y_train.shape}")
      6 print(f"Original Testing data shape: {X_test.shape}, Labels shape:
{y_test.shape}")

NameError: name 'tf' is not defined
```

In [ ]:
```
# Cell 3: Build the CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
```

```
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5), # Helps prevent overfitting
    Dense(num_classes, activation='softmax') # Output layer for 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

In [ ]:

```
# Cell 4: Train the Model
# Use GPU if available (Google Colab usually provides one)
# The training process will automatically use GPU if TF is configured for
history = model.fit(X_train, y_train_one_hot,
                    epochs=10, # You can try fewer epochs (e.g., 5) to sav
                    batch_size=64,
                    validation_split=0.1) # Use 10% of training data for v

print("\nModel training complete!")
```

In [ ]:

```
# Cell 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test_one_hot, verbose=0)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Cell 5: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test_one_hot, verbose=0)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Cell 6: Visualize Model Predictions
# Get 5 random sample images from the test set
sample_indices = np.random.choice(len(X_test), 5, replace=False)
sample_images = X_test[sample_indices]
sample_true_labels = y_test[sample_indices]

# Make predictions
predictions = model.predict(sample_images)
predicted_labels = np.argmax(predictions, axis=1)

plt.figure(figsize=(12, 3))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(sample_images[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {sample_true_labels[i]}\nPred: {predicted_labels[i]}
    plt.axis('off')
```