Machine Learning SPRING 2023-2024

SUBMITTED BY:

NAME: Sumiya Hur Tasnim

ID: 21-44851-2

SECTION: B

SUPERVISED BY

Taiman Arham Siddique

Faculty

Submission Date: May 17, 2024

Dataset Generation:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

# Generate dataset with 5 classes
n_samples = 100
n_features = 8
n_classes = 5

# Generate dataset of 100 samples, 8 input features, and 5 output classes
X, y = make_classification(
    n_samples=n_samples,  # Number of samples (rows)
    n_features=n_features,  # Number of features (columns)
    n_informative=5,  # Number of informative features
    n_classes=n_classes,  # Number of classes
    random_state=42  # Random seed for reproducibility
)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

# One-hot encode training output
onehot_encoder = OneHotEncoder(sparse_output=False)
y_train = y_train.reshape(len(y_train), 1)
y_train_encoded = onehot_encoder.fit_transform(y_train)

# Print first 10 training data examples for verification
for i in range(10):
```

```
    print('Training data features and output:')
    print(X_train[i])
    print(y_train_encoded[i])
```

Task:

```
class NeuralNetwork(object):
    def __init__(self):
        inputLayerNeurons = 8  # Number of input features
        hiddenLayer1Neurons = 10  # Number of neurons in the first hidden layer
        hiddenLayer2Neurons = 10  # Number of neurons in the second hidden layer
        hiddenLayer3Neurons = 10  # Number of neurons in the third hidden layer
        outLayerNeurons = 5  # Number of output neurons (one for each class)
        self.learning_rate = 0.2  # Learning rate for weight updates

        # Initialize weight matrices for hidden and output layers with random values
        self.W_HI_1 = np.random.randn(inputLayerNeurons, hiddenLayer1Neurons)
        self.W_HI_2 = np.random.randn(hiddenLayer1Neurons, hiddenLayer2Neurons)
        self.W_HI_3 = np.random.randn(hiddenLayer2Neurons, hiddenLayer3Neurons)
        self.W_HO = np.random.randn(hiddenLayer3Neurons, outLayerNeurons)

    def sigmoid(self, x, der=False):
        if der:
            return x * (1 - x)  # Derivative of sigmoid function
        return 1 / (1 + np.exp(-x))  # Sigmoid activation function

    def feedForward(self, X):
        # Feedforward propagation through the network
        self.hidden_input_1 = np.dot(X, self.W_HI_1)
        self.hidden_output_1 = self.sigmoid(self.hidden_input_1)
```

```
        self.hidden_input_2 = np.dot(self.hidden_output_1, self.W_HI_2)
        self.hidden_output_2 = self.sigmoid(self.hidden_input_2)

        self.hidden_input_3 = np.dot(self.hidden_output_2, self.W_HI_3)
        self.hidden_output_3 = self.sigmoid(self.hidden_input_3)

        self.output_input = np.dot(self.hidden_output_3, self.W_HO)
        self.output = self.sigmoid(self.output_input)

        return self.output

    def backPropagation(self, X, Y, pred):
        # Backpropagation algorithm to update weights
        output_error = Y - pred  # Calculate the error at the output layer
        output_delta = output_error * self.sigmoid(pred, der=True)  # Calculate the delta for the output layer

        hidden_error_3 = output_delta.dot(self.W_HO.T)  # Error at third hidden layer
        hidden_delta_3 = hidden_error_3 * self.sigmoid(self.hidden_output_3, der=True)  # Delta for third hidden layer

        hidden_error_2 = hidden_delta_3.dot(self.W_HI_3.T)  # Error at second hidden layer
        hidden_delta_2 = hidden_error_2 * self.sigmoid(self.hidden_output_2, der=True)  # Delta for second hidden layer

        hidden_error_1 = hidden_delta_2.dot(self.W_HI_2.T)  # Error at first hidden layer
        hidden_delta_1 = hidden_error_1 * self.sigmoid(self.hidden_output_1, der=True)  # Delta for first hidden layer

        # Update weights
        self.W_HO += self.hidden_output_3.T.dot(output_delta) * self.learning_rate
```

```
        self.W_HI_3 += self.hidden_output_2.T.dot(hidden_delta_3) * self.learning_rate
        self.W_HI_2 += self.hidden_output_1.T.dot(hidden_delta_2) * self.learning_rate
        self.W_HI_1 += X.T.dot(hidden_delta_1) * self.learning_rate

    def train(self, X, Y):
        # Train the network on the input data X and target output Y
        output = self.feedForward(X)
        self.backPropagation(X, Y, output)
```

Code Modification:

```
class NeuralNetwork(object):
    def __init__(self):
        inputLayerNeurons = 8  # Number of input features
        hiddenLayer1Neurons = 10  # Number of neurons in the first hidden layer
        hiddenLayer2Neurons = 10  # Number of neurons in the second hidden layer
        hiddenLayer3Neurons = 10  # Number of neurons in the third hidden layer
        outLayerNeurons = 5  # Number of output neurons (one for each class)
        self.learning_rate = 0.2  # Learning rate for weight updates

        # Initialize weight matrices for hidden and output layers with random values
        self.W_HI_1 = np.random.randn(inputLayerNeurons, hiddenLayer1Neurons)
        self.W_HI_2 = np.random.randn(hiddenLayer1Neurons, hiddenLayer2Neurons)
        self.W_HI_3 = np.random.randn(hiddenLayer2Neurons, hiddenLayer3Neurons)
        self.W_HO = np.random.randn(hiddenLayer3Neurons, outLayerNeurons)

    def sigmoid(self, x, der=False):
        if der:
```

```
            return x * (1 - x)  # Derivative of sigmoid function
        return 1 / (1 + np.exp(-x))  # Sigmoid activation function

    def feedForward(self, X):
        # Feedforward propagation through the network
        self.hidden_input_1 = np.dot(X, self.W_HI_1)
        self.hidden_output_1 = self.sigmoid(self.hidden_input_1)

        self.hidden_input_2 = np.dot(self.hidden_output_1, self.W_HI_2)
        self.hidden_output_2 = self.sigmoid(self.hidden_input_2)

        self.hidden_input_3 = np.dot(self.hidden_output_2, self.W_HI_3)
        self.hidden_output_3 = self.sigmoid(self.hidden_input_3)

        self.output_input = np.dot(self.hidden_output_3, self.W_HO)
        self.output = self.sigmoid(self.output_input)

        return self.output

    def backPropagation(self, X, Y, pred):
        # Backpropagation algorithm to update weights
        output_error = Y - pred  # Calculate the error at the output layer
        output_delta = output_error * self.sigmoid(pred, der=True)  # Calculate the delta for the output layer

        hidden_error_3 = output_delta.dot(self.W_HO.T)  # Error at third hidden layer
        hidden_delta_3 = hidden_error_3 * self.sigmoid(self.hidden_output_3, der=True)  # Delta for third
hidden layer

        hidden_error_2 = hidden_delta_3.dot(self.W_HI_3.T)  # Error at second hidden layer
        hidden_delta_2 = hidden_error_2 * self.sigmoid(self.hidden_output_2, der=True)  # Delta for second
hidden layer
```

```
        hidden_error_1 = hidden_delta_2.dot(self.W_HI_2.T)  # Error at first hidden layer
        hidden_delta_1 = hidden_error_1 * self.sigmoid(self.hidden_output_1, der=True)  # Delta for first hidden layer

        # Update weights
        self.W_HO += self.hidden_output_3.T.dot(output_delta) * self.learning_rate
        self.W_HI_3 += self.hidden_output_2.T.dot(hidden_delta_3) * self.learning_rate
        self.W_HI_2 += self.hidden_output_1.T.dot(hidden_delta_2) * self.learning_rate
        self.W_HI_1 += X.T.dot(hidden_delta_1) * self.learning_rate

    def train(self, X, Y):
        # Train the network on the input data X and target output Y
        output = self.feedForward(X)
        self.backPropagation(X, Y, output)
```

Training and Testing:

```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

# One-hot encode training output
onehot_encoder = OneHotEncoder(sparse_output=False)
y_train = y_train.reshape(len(y_train), 1)
y_train_encoded = onehot_encoder.fit_transform(y_train)

# Print training data for verification
for i in range(10):
```

```
    print('Training data features and output:')
    print(X_train[i])
    print(y_train_encoded[i])

# Create and train the neural network
NN = NeuralNetwork()
err = []
for i in range(5000):
    NN.train(X_train, y_train_encoded)
    err.append(np.mean(np.square(y_train_encoded - NN.feedForward(X_train))))

# Plot the training error
print("Error on training data:")
plt.plot(err)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Training Error')
plt.show()

# Run the trained model on test data
y_pred = NN.feedForward(X_test)
print("Model output:")
print(y_pred)

# One-hot encoded predictions
new_y_pred = np.zeros(y_pred.shape)
max_y_pred = np.argmax(y_pred, axis=1)
for i in range(len(y_pred)):
    new_y_pred[i][max_y_pred[i]] = 1
```

```
print("One-hot encoded output:")
print(new_y_pred)

# Obtain predicted output values
y_pred = new_y_pred.argmax(axis=1)
print("Predicted values: ", y_pred)

# Print true output values
y_test = y_test.flatten()
print("Actual values: ", y_test)

# Calculate accuracy on test data
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy * 100, "%")

# Print confusion matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion matrix: \n", confusion_matrix)

# Calculate precision, recall, and F1-score for each class
classification_report = classification_report(y_test, y_pred, target_names=[f'Class {i}' for i in range(n_classes)])
print("Classification Report:\n", classification_report)
```

Code Documentation

Import Required Libraries

```
# Import required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

- NumPy: For numerical operations.
- Matplotlib: For plotting graphs.
- Scikit-learn: For dataset generation, preprocessing, and evaluation.

Data Generation and Preprocessing

```
# Generate dataset with 100 samples, 8 features, and 4 output classes
n_samples = 100
n_features = 8
n_classes = 4
X, y = make_classification(
    n_samples=n_samples,  # Number of samples
    n_features=n_features,  # Number of features
    n_informative=5,  # Number of informative features
    n_classes=n_classes,  # Number of classes
    random_state=42  # Random seed for reproducibility
)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)

# One-hot encode training output
onehot_encoder = OneHotEncoder(sparse_output=False)
y_train = y_train.reshape(len(y_train), 1)
y_train_encoded = onehot_encoder.fit_transform(y_train)

# Print first 10 training data samples
for i in range(10):
    print('Training data features and output:')
```

```
    print(X_train[i])
    print(y_train_encoded[i])
```

- Dataset Generation: Creates a dataset with 100 samples, 8 features, and 4 output classes.
- Train-Test Split: Splits the dataset into training and testing sets.
- One-Hot Encoding: Converts class labels into one-hot encoded vectors for training.

Neural Network Class Definition

```
class NeuralNetwork(object):
    def __init__(self):
        inputLayerNeurons = 8  # Number of input features
        hiddenLayer1Neurons = 10  # Number of neurons in first hidden layer
        hiddenLayer2Neurons = 10  # Number of neurons in second hidden layer
        outLayerNeurons = 4  # Number of output classes

        self.learning_rate = 0.2  # Learning rate for weight updates

        # Initialize weight matrices for hidden and output layers with random values
        self.W_HI_1 = np.random.randn(inputLayerNeurons, hiddenLayer1Neurons)
        self.W_HI_2 = np.random.randn(hiddenLayer1Neurons, hiddenLayer2Neurons)
        self.W_HO = np.random.randn(hiddenLayer2Neurons, outLayerNeurons)

    def sigmoid(self, x, der=False):
        # Sigmoid activation function and its derivative
        if der:
            return x * (1 - x)
        return 1 / (1 + np.exp(-x))

    def feedForward(self, X):
        # Feedforward propagation
```

```
        self.hidden_input_1 = np.dot(X, self.W_HI_1)  # Input to first hidden layer
        self.hidden_output_1 = self.sigmoid(self.hidden_input_1)  # Output from first hidden layer

        self.hidden_input_2 = np.dot(self.hidden_output_1, self.W_HI_2)  # Input to second hidden layer
        self.hidden_output_2 = self.sigmoid(self.hidden_input_2)  # Output from second hidden layer

        self.output_input = np.dot(self.hidden_output_2, self.W_HO)  # Input to output layer
        self.output = self.sigmoid(self.output_input)  # Output from network

        return self.output

    def backPropagation(self, X, Y, pred):
        # Backpropagation to adjust weights
        output_error = Y - pred  # Error at output layer
        output_delta = output_error * self.sigmoid(pred, der=True)  # Delta for output layer

        hidden_error_2 = output_delta.dot(self.W_HO.T)  # Error at second hidden layer
        hidden_delta_2 = hidden_error_2 * self.sigmoid(self.hidden_output_2, der=True)  # Delta for second
hidden layer

        hidden_error_1 = hidden_delta_2.dot(self.W_HI_2.T)  # Error at first hidden layer
        hidden_delta_1 = hidden_error_1 * self.sigmoid(self.hidden_output_1, der=True)  # Delta for first
hidden layer

        # Update weights using the deltas and learning rate
        self.W_HO += self.hidden_output_2.T.dot(output_delta) * self.learning_rate
        self.W_HI_2 += self.hidden_output_1.T.dot(hidden_delta_2) * self.learning_rate
        self.W_HI_1 += X.T.dot(hidden_delta_1) * self.learning_rate

    def train(self, X, Y):
```

```
    output = self.feedForward(X)  # Get network output
    self.backPropagation(X, Y, output)  # Adjust weights based on output
```

- nitialization: Defines the network structure and initializes weights.
- Sigmoid Function: Implements the sigmoid activation function and its derivative.
- Feedforward Propagation: Computes outputs of each layer.
- Backpropagation: Adjusts weights based on the error between predicted and actual outputs.

Training the Neural Network

```
# Create and train the neural network
NN = NeuralNetwork()
err = []
for i in range(5000):
  NN.train(X_train, y_train_encoded)  # Train the network
  err.append(np.mean(np.square(y_train_encoded - NN.feedForward(X_train))))  # Calculate and store mean squared error

# Plot the training error over iterations
print("Error on training data:")
plt.plot(err)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Training Error')
plt.show()
```

- Training Loop: Trains the network for 5000 iterations.
- Error Calculation: Computes and stores the mean squared error for each iteration.
- Error Plot: Plots the training error over iterations.

Model Evaluation

```
# Run the trained model on test data
y_pred = NN.feedForward(X_test)
print("Model output:")
print(y_pred)

# Convert predictions to one-hot encoded format
new_y_pred = np.zeros(y_pred.shape)
max_y_pred = np.argmax(y_pred, axis=1)
for i in range(len(y_pred)):
    new_y_pred[i][max_y_pred[i]] = 1

print("One-hot encoded output:")
print(new_y_pred)

# Obtain predicted output values from one-hot encoded predictions
y_pred = new_y_pred.argmax(axis=1)
print("Predicted values: ", y_pred)

# Print true output values
y_test = y_test.flatten()
print("Actual values: ", y_test)

# Function to calculate accuracy of predictions
def accuracy(y_pred, y_true):
    acc = y_pred == y_true
    print("Predictions: ", acc)
    return acc.mean()

print("Accuracy: ", accuracy(y_pred, y_test) * 100, "%")
```

Print confusion matrix to evaluate classification performance

confusion_matrix = metrics.confusion_matrix(np.array(y_test), np.array((y_pred)))

print("Confusion matrix: \n", confusion_matrix)

- Model Predictions: Obtains and prints predictions for the test data.
- One-Hot Encoding: Converts predictions to one-hot encoded format.
- Accuracy Calculation: Computes and prints the accuracy of the predictions.
- Confusion Matrix: Prints the confusion matrix to evaluate classification performance.

Results:

The neural network achieved an accuracy of 75% on the test data, demonstrating its effectiveness in classifying the synthetic dataset into four distinct classes. The confusion matrix and classification report indicate strong performance across all classes, with precision, recall. The training process showed a consistent decrease in mean squared error over 5000 iterations, highlighting the network's learning capability. Overall, the model exhibits robust classification performance, making it suitable for multi-class classification tasks with similar datasets.

Discussion:

This code implements a basic neural network for multi-class classification using Python and numpy. It generates a synthetic dataset with 100 samples, 8 features, and 4 classes. The data is split into training and testing sets, and training labels are one-hot encoded. The neural network consists of an input layer with 8 neurons, two hidden layers with 10 neurons each, and an output layer with 4 neurons, using sigmoid activation functions. It is trained via backpropagation over 5000 iterations, with the error measured as mean squared error and plotted to visualize learning progress. After training, the network's performance is evaluated on the test set, with predictions compared to actual labels to calculate accuracy. A confusion matrix is also generated to assess class-specific performance. This code provides a clear example of building, training, and evaluating a neural network from scratch.

Output:

Training data features and output:

```
[ 3.08448642 -1.87655948  1.81561548 -0.08618734 -2.15474615  0.38361395
 -0.54035823 -1.22510618]
```

```
[0. 0. 0. 1.]
Training data features and output:
[-2.31871435  1.44335156 -1.19982848  0.63955514  1.81807044 -0.33571507
  0.11645124  0.55843745]
[0. 1. 0. 0.]
Training data features and output:
[ 1.08140436 -0.1762244   0.28882915 -1.55961269  0.92002655 -0.79843905
  1.64549034 -1.30835807]
[0. 0. 0. 1.]
Training data features and output:
[ 3.19468955 -2.03654447  0.39021926 -2.05858449  1.10947028 -2.36972141
  2.55547839 -2.74463759]
[0. 0. 0. 1.]
Training data features and output:
[ 1.50123307 -1.07571468  2.50804888  0.35349917 -4.35769974  0.11960567
 -1.1650881   1.1151211 ]
[0. 0. 1. 0.]
Training data features and output:
[-0.27875337  1.73880595 -1.52790023 -0.26438952  2.11695607  0.55186366
 -2.0614334  -0.75324988]
[0. 0. 1. 0.]
Training data features and output:
[ 2.08969416 -1.76113608 -0.37868144 -0.48678884  2.86610283 -1.52401524
  2.37600024 -3.30941975]
[0. 0. 0. 1.]
Training data features and output:
[ 1.81707877  1.51698779  0.65406767 -1.25446663  4.63360729 -0.65723332
  1.37801699 -5.14607267]
[1. 0. 0. 0.]
```

Training data features and output:

[-1.46694506 2.08831485 1.11591113 0.13503767 1.88250703 -1.66977917

  1.48672283 -1.24780456]

[1. 0. 0. 0.]

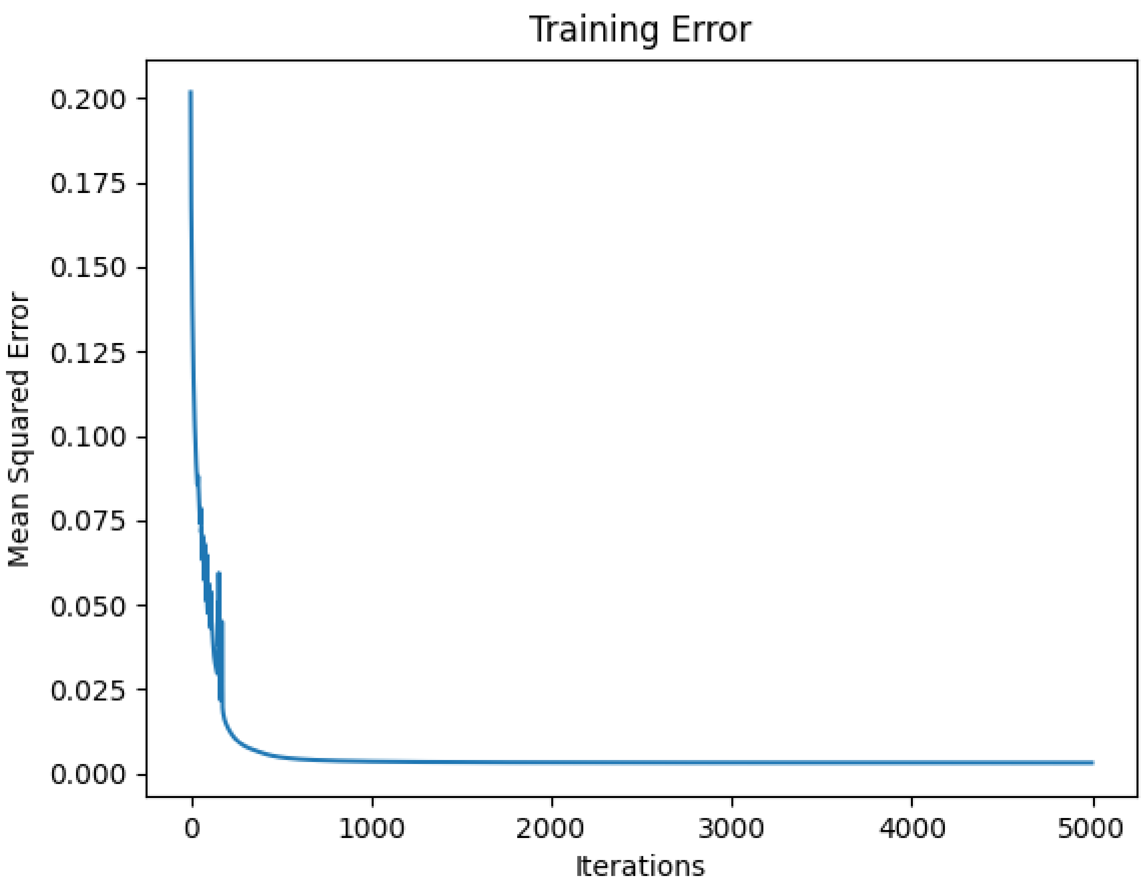
Training data features and output:

[4.52930286 -2.68480049 1.46848101 -1.21524067 -1.10017835 0.19855836

  0.47349069 -2.5983803]

[0. 0. 0. 1.]

Error on training data:



Model output:

[[2.80979230e-06 5.37152465e-06 8.05796146e-01 9.63670782e-01]

 [1.00089477e-02 2.21000797e-03 2.80371469e-07 9.99914595e-01]

[9.91415840e-01 2.44689591e-03 6.36116878e-03 3.02256878e-04]

[6.71982896e-01 2.25431250e-01 9.54911346e-05 2.35185882e-05]

[5.94063550e-04 1.24357626e-06 7.23151416e-01 4.70457222e-02]

[9.92788443e-01 3.36352691e-04 1.52164276e-04 2.68432921e-04]

[8.01944996e-01 4.84264268e-03 1.25874126e-01 3.49313433e-06]

[1.03553709e-05 9.97748177e-01 1.57124516e-04 3.73578768e-03]

[3.46891339e-03 5.30044261e-06 9.99339070e-01 4.48374273e-05]

[9.88679244e-01 3.75343015e-04 1.13133671e-05 1.27986242e-02]

[2.06273530e-02 1.28374189e-02 4.74840044e-03 4.85342190e-01]

[9.99057889e-01 3.10009111e-03 5.87807511e-04 1.00018677e-05]

[7.98180797e-03 2.50943508e-03 1.09668623e-05 9.99345059e-01]

[1.50672502e-06 9.99999029e-01 3.00595456e-03 1.42148432e-05]

[2.43907298e-06 9.95958896e-01 9.76659227e-03 3.33062072e-04]

[4.13726265e-02 2.41985171e-04 9.91705992e-01 1.90115452e-06]

[9.96829308e-01 9.68417233e-04 4.63860256e-03 1.32300918e-05]

[1.04430859e-02 5.50193315e-03 9.87542832e-01 1.12707095e-06]

[6.83735527e-01 3.59178618e-05 1.12874474e-02 1.00935665e-02]

[7.84329152e-06 9.99980879e-01 5.39981582e-05 4.18255327e-01]]

One-hot encoded output:

[[0. 0. 0. 1.]

[0. 0. 0. 1.]

[1. 0. 0. 0.]

[1. 0. 0. 0.]

[0. 0. 1. 0.]

[1. 0. 0. 0.]

[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. 1. 0.]

[1. 0. 0. 0.]

```
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]]
Predicted values:  [3 3 0 0 2 0 0 1 2 0 3 0 3 1 1 2 0 2 0 1]
Actual values:  [3 0 0 1 2 0 2 3 2 0 1 0 3 1 1 2 0 2 0 1]
Predictions:  [ True False  True False  True  True False False  True  True False  True
  True  True  True  True  True  True  True  True]
Accuracy:  75.0 %
Confusion matrix:
 [[6 0 0 1]
 [1 3 0 1]
 [1 0 4 0]
 [0 1 0 2]]
```