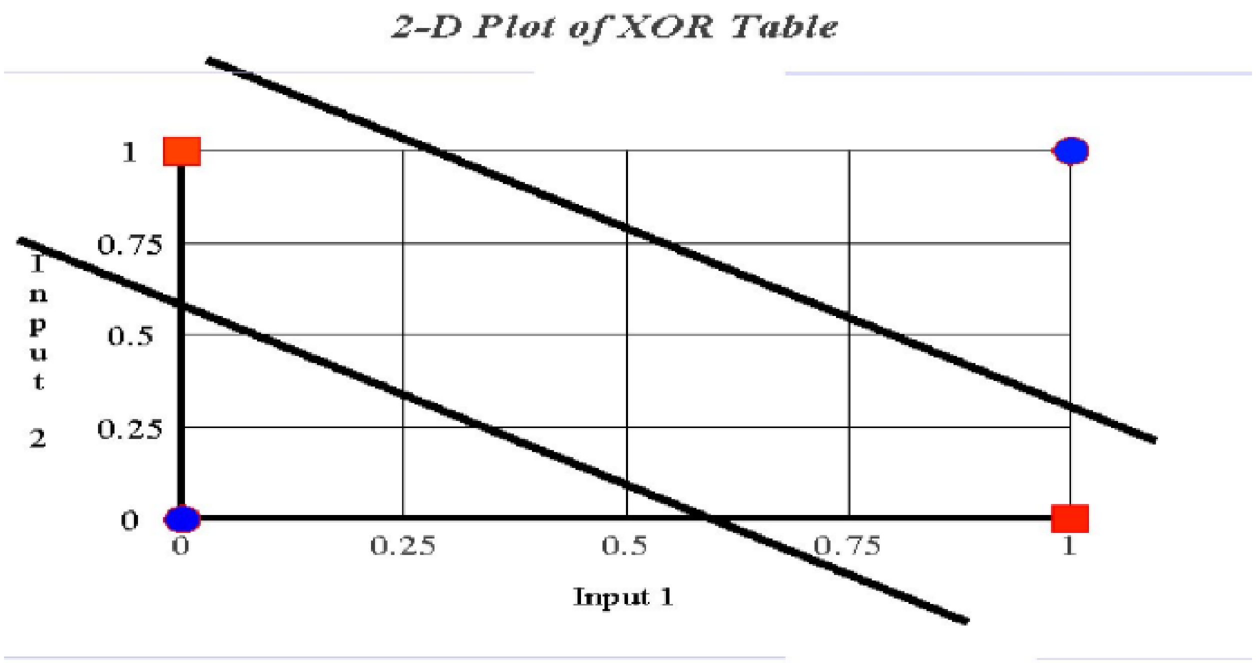
นาย สัณหณัฐ พรมจรรย์ 63070501069

# Quiz 7

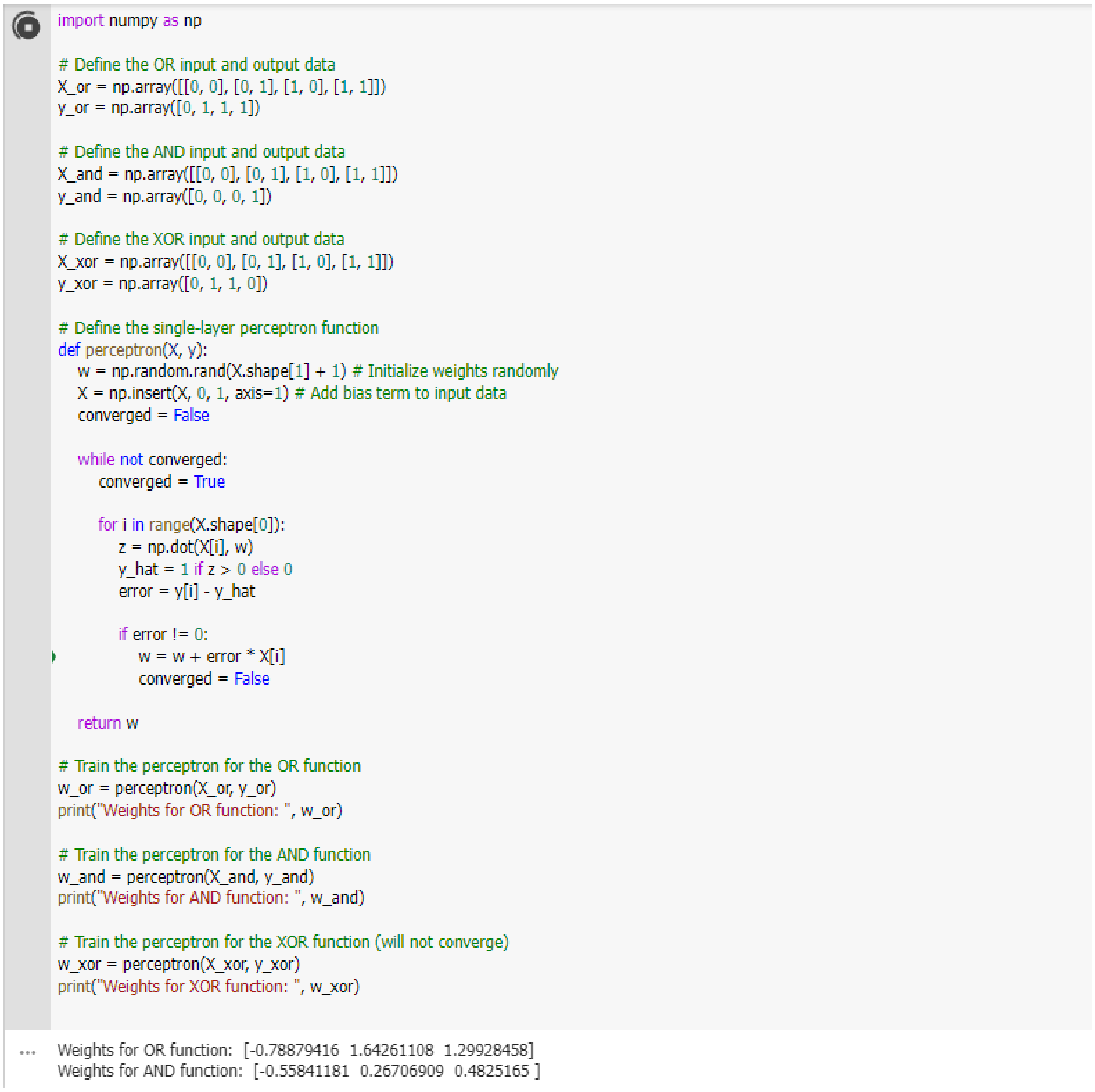
1. *15 points*. 1 hour. Show that the OR, AND will work for a single layer perceptron, but the XOR will not find a solution.

| $I_1$ | $I_2$ | AND | OR | XOR |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |



Since XOR outputs cannot be separated using a straight line, they cannot be classified using a single layer perceptron.

**Link:** https://colab.research.google.com/drive/1UrSTCLvWxBlDj1AsuLa45VqGk-QVKbjr?usp=sharing



```python
import numpy as np

# Define the OR input and output data
X_or = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_or = np.array([0, 1, 1, 1])

# Define the AND input and output data
X_and = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_and = np.array([0, 0, 0, 1])

# Define the XOR input and output data
X_xor = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_xor = np.array([0, 1, 1, 0])

# Define the single-layer perceptron function
def perceptron(X, y):
    w = np.random.rand(X.shape[1] + 1) # Initialize weights randomly
    X = np.insert(X, 0, 1, axis=1) # Add bias term to input data
    converged = False

    while not converged:
        converged = True

        for i in range(X.shape[0]):
            z = np.dot(X[i], w)
            y_hat = 1 if z > 0 else 0
            error = y[i] - y_hat

            if error != 0:
                w = w + error * X[i]
                converged = False

    return w

# Train the perceptron for the OR function
w_or = perceptron(X_or, y_or)
print("Weights for OR function: ", w_or)

# Train the perceptron for the AND function
w_and = perceptron(X_and, y_and)
print("Weights for AND function: ", w_and)

# Train the perceptron for the XOR function (will not converge)
w_xor = perceptron(X_xor, y_xor)
print("Weights for XOR function: ", w_xor)
```

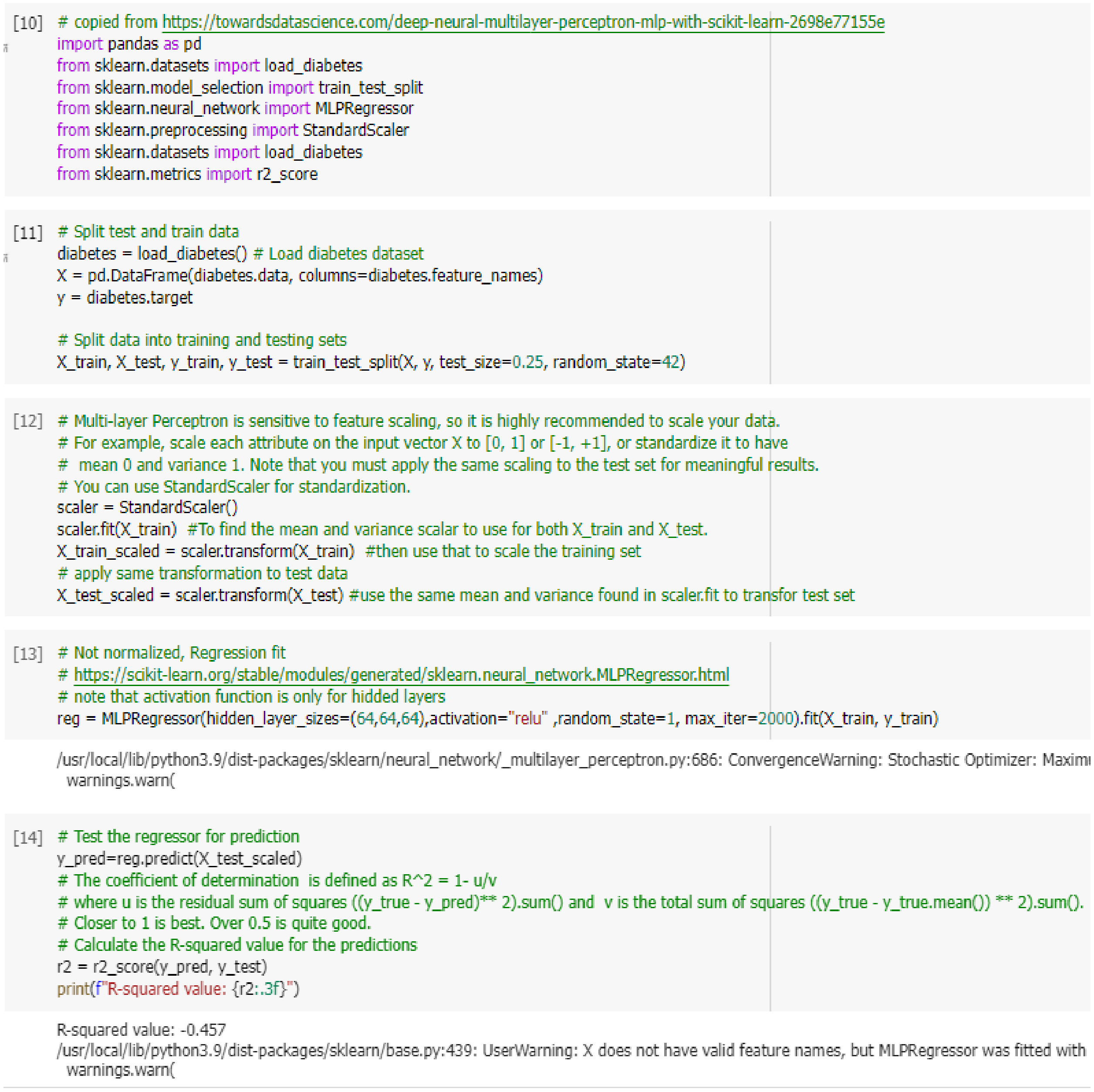
```
Weights for OR function:  [-0.78879416  1.64261108  1.29928458]
Weights for AND function:  [-0.55841181  0.26706909  0.4825165 ]
```

We train the perceptron on the OR and AND functions and print out the resulting weights. As we can see, the perceptron is able to learn these functions using a single layer.

However, when we try to train the perceptron on the XOR function, the function will not converge. This is because the XOR function is not linearly separable and cannot be learned using a single-layer perceptron.

2. *15 points*. 1 hour. Use multilayer perceptron on sklearn's diabetes data meant for regression using 25% test data, find the R-square.

Link: https://colab.research.google.com/drive/1XzJ1ZvNl40MhXxP8z6x5KQkLBWkUYOUR?usp=sharing



```
[10] # copied from https://towardsdatascience.com/deep-neural-multilayer-perceptron-mlp-with-scikit-learn-2698e77155e
     import pandas as pd
     from sklearn.datasets import load_diabetes
     from sklearn.model_selection import train_test_split
     from sklearn.neural_network import MLPRegressor
     from sklearn.preprocessing import StandardScaler
     from sklearn.datasets import load_diabetes
     from sklearn.metrics import r2_score
```

```
[11] # Split test and train data
     diabetes = load_diabetes() # Load diabetes dataset
     X = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
     y = diabetes.target

     # Split data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

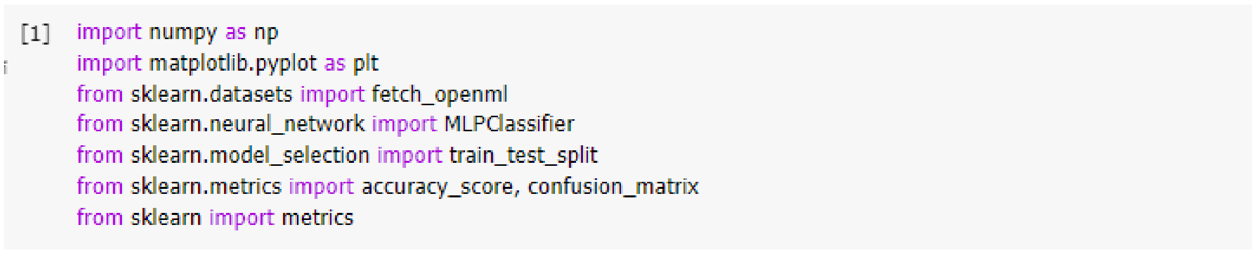
```
[12] # Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data.
     # For example, scale each attribute on the input vector X to [0, 1] or [-1, +1], or standardize it to have
     #  mean 0 and variance 1. Note that you must apply the same scaling to the test set for meaningful results.
     # You can use StandardScaler for standardization.
     scaler = StandardScaler()
     scaler.fit(X_train)  #To find the mean and variance scalar to use for both X_train and X_test.
     X_train_scaled = scaler.transform(X_train)  #then use that to scale the training set
     # apply same transformation to test data
     X_test_scaled = scaler.transform(X_test) #use the same mean and variance found in scaler.fit to transfor test set
```

```
[13] # Not normalized, Regression fit
     # https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
     # note that activation function is only for hidded layers
     reg = MLPRegressor(hidden_layer_sizes=(64,64,64),activation="relu" ,random_state=1, max_iter=2000).fit(X_train, y_train)

     /usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximu
       warnings.warn(
```

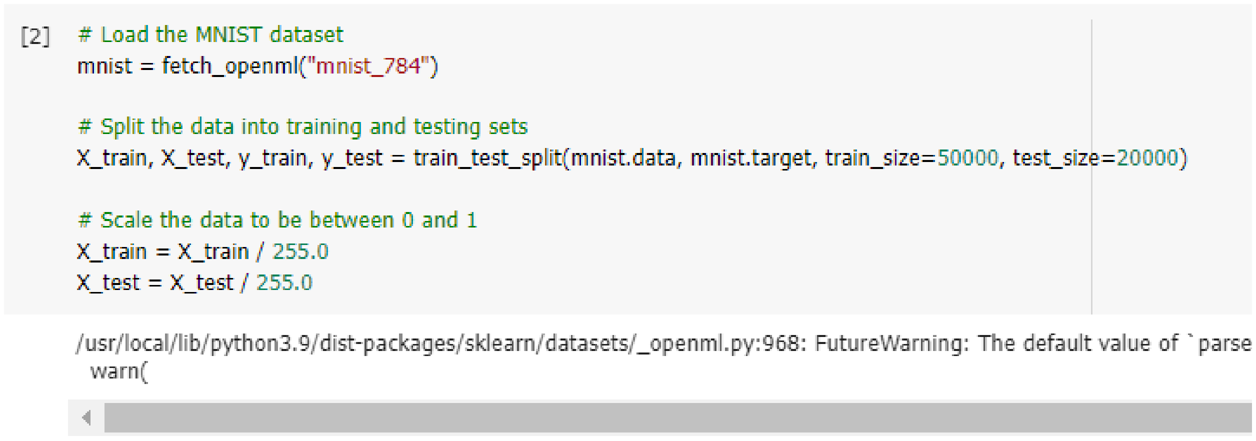
```
[14] # Test the regressor for prediction
     y_pred=reg.predict(X_test_scaled)
     # The coefficient of determination  is defined as R^2 = 1- u/v
     # where u is the residual sum of squares ((y_true - y_pred)** 2).sum() and  v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum().
     # Closer to 1 is best. Over 0.5 is quite good.
     # Calculate the R-squared value for the predictions
     r2 = r2_score(y_pred, y_test)
     print(f"R-squared value: {r2:.3f}")

     R-squared value: -0.457
     /usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MLPRegressor was fitted with
       warnings.warn(
```

3. *20 points*. 2 hours. Follow the instructions in the following site for MLPClassifier for MNIST data. https://towardsdatascience.com/classifying-handwritten-digits-using-a-multilayer-perceptron-classifier-mlp-bc8453655880 . Use 50,000 data samples for training and 20,000 for testing. Note you should scale your data to be 0-1 by dividing it by 255. Report your accuracy and print out the confusion matrix.

Link: https://colab.research.google.com/drive/1SxtJNAn6TTaFkQq4M_SDHkNL83rTou4u?usp=sharing

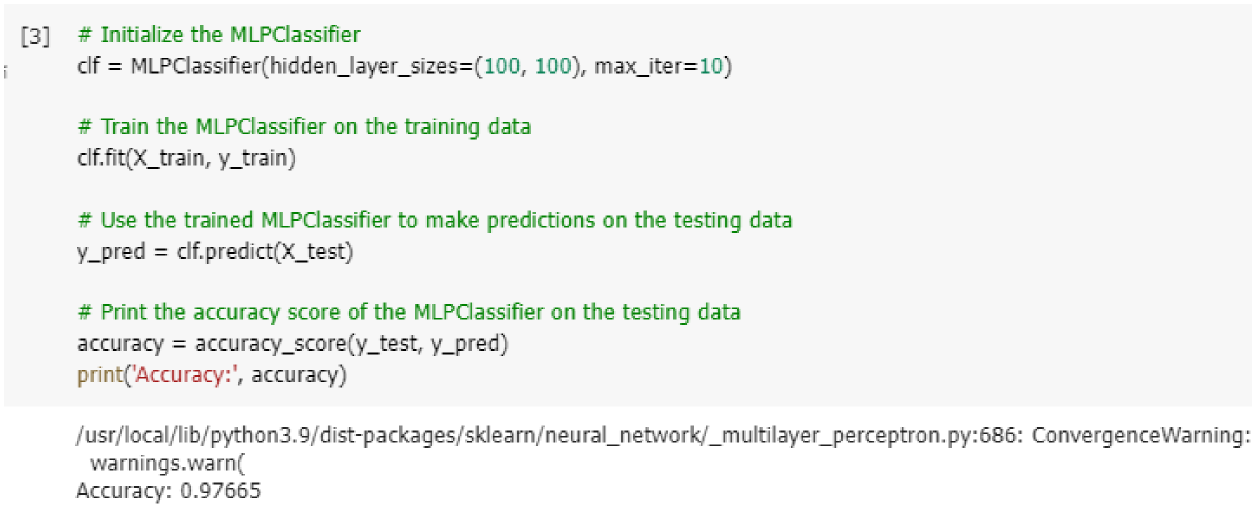


```
[1] import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.datasets import fetch_openml
    from sklearn.neural_network import MLPClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, confusion_matrix
    from sklearn import metrics
```



```
[2] # Load the MNIST dataset
    mnist = fetch_openml("mnist_784")

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target, train_size=50000, test_size=20000)

    # Scale the data to be between 0 and 1
    X_train = X_train / 255.0
    X_test = X_test / 255.0
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parse
  warn(
```



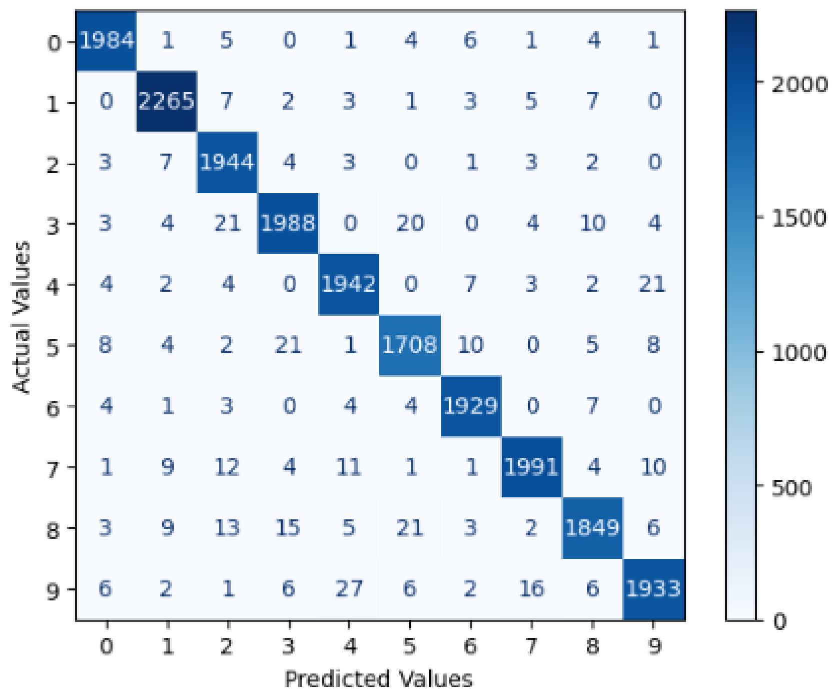
```
[3] # Initialize the MLPClassifier
    clf = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=10)

    # Train the MLPClassifier on the training data
    clf.fit(X_train, y_train)

    # Use the trained MLPClassifier to make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Print the accuracy score of the MLPClassifier on the testing data
    accuracy = accuracy_score(y_test, y_pred)
    print('Accuracy:', accuracy)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning:
  warnings.warn(
Accuracy: 0.97665
```

```
[4]  # Show the confusion matrix of the MLPClassifier on the testing data
     print('\nConfusion Matrix:')
     confusion_matrix(y_test, y_pred)
```

```
Confusion Matrix:
array([[1984,    1,    5,    0,    1,    4,    6,    1,    4,    1],
       [   0, 2265,    7,    2,    3,    1,    3,    5,    7,    0],
       [   3,    7, 1944,    4,    3,    0,    1,    3,    2,    0],
       [   3,    4,   21, 1988,    0,   20,    0,    4,   10,    4],
       [   4,    2,    4,    0, 1942,    0,    7,    3,    2,   21],
       [   8,    4,    2,   21,    1, 1708,   10,    0,    5,    8],
       [   4,    1,    3,    0,    4,    4, 1929,    0,    7,    0],
       [   1,    9,   12,    4,   11,    1,    1, 1991,    4,   10],
       [   3,    9,   13,   15,    5,   21,    3,    2, 1849,    6],
       [   6,    2,    1,    6,   27,    6,    2,   16,    6, 1933]])
```

```
# Confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
classes = np.unique(mnist.target)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
cm_display.plot(cmap=plt.cm.Blues)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

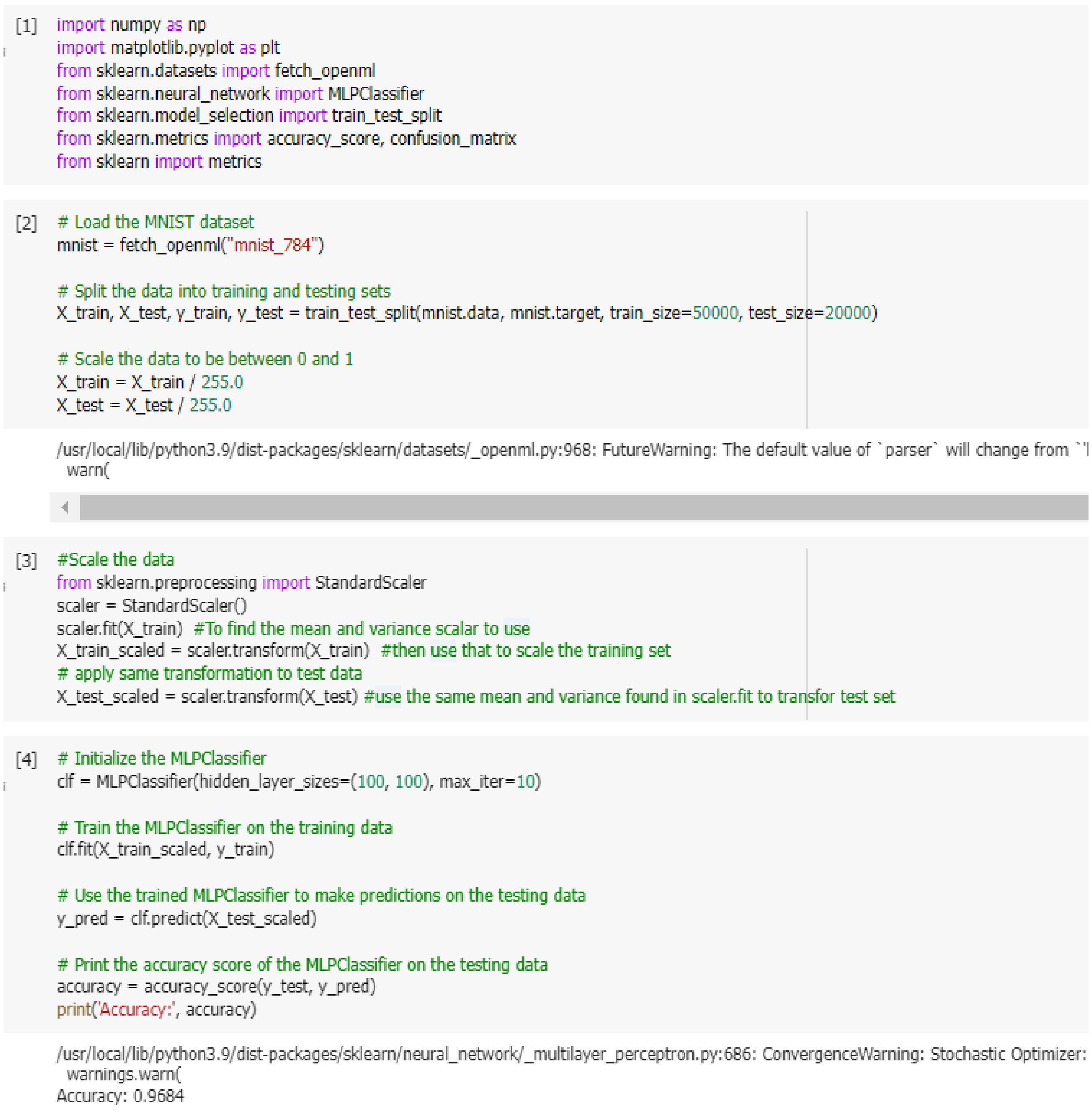


| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1984 | 1 | 5 | 0 | 1 | 4 | 6 | 1 | 4 | 1 |
| 1 | 0 | 2265 | 7 | 2 | 3 | 1 | 3 | 5 | 7 | 0 |
| 2 | 3 | 7 | 1944 | 4 | 3 | 0 | 1 | 3 | 2 | 0 |
| 3 | 3 | 4 | 21 | 1988 | 0 | 20 | 0 | 4 | 10 | 4 |
| 4 | 4 | 2 | 4 | 0 | 1942 | 0 | 7 | 3 | 2 | 21 |
| 5 | 8 | 4 | 2 | 21 | 1 | 1708 | 10 | 0 | 5 | 8 |
| 6 | 4 | 1 | 3 | 0 | 4 | 4 | 1929 | 0 | 7 | 0 |
| 7 | 1 | 9 | 12 | 4 | 11 | 1 | 1 | 1991 | 4 | 10 |
| 8 | 3 | 9 | 13 | 15 | 5 | 21 | 3 | 2 | 1849 | 6 |
| 9 | 6 | 2 | 1 | 6 | 27 | 6 | 2 | 16 | 6 | 1933 |

4. *10 points*. 0.5 hours. Repeat problem #3, but also normalize the input data by dividing by variance on the scaled 0-1 data by using sklearn's StandardScaler. Report your accuracy and see if it improves from problem #3.

It improves from problem #3

Link: https://colab.research.google.com/drive/1JJMlFeSBcjYHg-_dpfhtYl05GadTXvaj?usp=sharing



```
[1] import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.datasets import fetch_openml
    from sklearn.neural_network import MLPClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, confusion_matrix
    from sklearn import metrics
```

```
[2] # Load the MNIST dataset
    mnist = fetch_openml("mnist_784")

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target, train_size=50000, test_size=20000)

    # Scale the data to be between 0 and 1
    X_train = X_train / 255.0
    X_test = X_test / 255.0
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `parser` will change from `'
  warn(
```

```
[3] #Scale the data
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler.fit(X_train)  #To find the mean and variance scalar to use
    X_train_scaled = scaler.transform(X_train)  #then use that to scale the training set
    # apply same transformation to test data
    X_test_scaled = scaler.transform(X_test) #use the same mean and variance found in scaler.fit to transfor test set
```

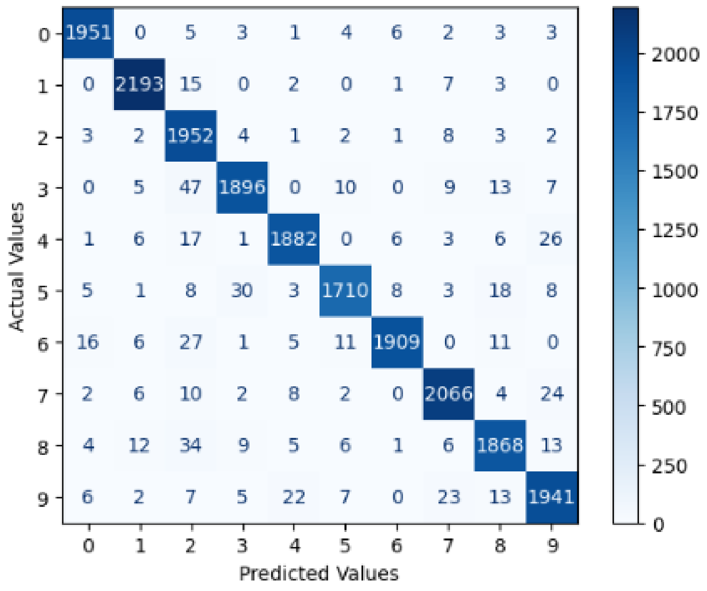
```
[4] # Initialize the MLPClassifier
    clf = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=10)

    # Train the MLPClassifier on the training data
    clf.fit(X_train_scaled, y_train)

    # Use the trained MLPClassifier to make predictions on the testing data
    y_pred = clf.predict(X_test_scaled)

    # Print the accuracy score of the MLPClassifier on the testing data
    accuracy = accuracy_score(y_test, y_pred)
    print('Accuracy:', accuracy)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer:
  warnings.warn(
Accuracy: 0.9684
```

```
[5]  # Show the confusion matrix of the MLPClassifier on the testing data
     print("\nConfusion Matrix:")
     confusion_matrix(y_test, y_pred)
```

```
Confusion Matrix:
array([[1951,    0,    5,    3,    1,    4,    6,    2,    3,    3],
       [   0, 2193,   15,    0,    2,    0,    1,    7,    3,    0],
       [   3,    2, 1952,    4,    1,    2,    1,    8,    3,    2],
       [   0,    5,   47, 1896,    0,   10,    0,    9,   13,    7],
       [   1,    6,   17,    1, 1882,    0,    6,    3,    6,   26],
       [   5,    1,    8,   30,    3, 1710,    8,    3,   18,    8],
       [  16,    6,   27,    1,    5,   11, 1909,    0,   11,    0],
       [   2,    6,   10,    2,    8,    2,    0, 2066,    4,   24],
       [   4,   12,   34,    9,    5,    6,    1,    6, 1868,   13],
       [   6,    2,    7,    5,   22,    7,    0,   23,   13, 1941]])
```

```
[6]  # Confusion matrix
     cm = metrics.confusion_matrix(y_test, y_pred)
     classes = np.unique(mnist.target)
     cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classes)
     cm_display.plot(cmap=plt.cm.Blues)
     plt.xlabel('Predicted Values')
     plt.ylabel('Actual Values')
     plt.show()
```



|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1951 | 0 | 5 | 3 | 1 | 4 | 6 | 2 | 3 | 3 |
| 1 | 0 | 2193 | 15 | 0 | 2 | 0 | 1 | 7 | 3 | 0 |
| 2 | 3 | 2 | 1952 | 4 | 1 | 2 | 1 | 8 | 3 | 2 |
| 3 | 0 | 5 | 47 | 1896 | 0 | 10 | 0 | 9 | 13 | 7 |
| 4 | 1 | 6 | 17 | 1 | 1882 | 0 | 6 | 3 | 6 | 26 |
| 5 | 5 | 1 | 8 | 30 | 3 | 1710 | 8 | 3 | 18 | 8 |
| 6 | 16 | 6 | 27 | 1 | 5 | 11 | 1909 | 0 | 11 | 0 |
| 7 | 2 | 6 | 10 | 2 | 8 | 2 | 0 | 2066 | 4 | 24 |
| 8 | 4 | 12 | 34 | 9 | 5 | 6 | 1 | 6 | 1868 | 13 |
| 9 | 6 | 2 | 7 | 5 | 22 | 7 | 0 | 23 | 13 | 1941 |