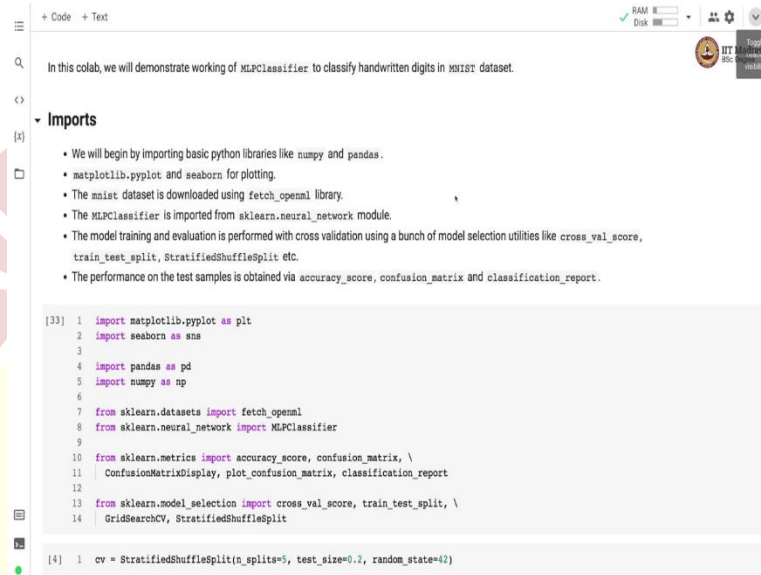


IIT Madras
ONLINE DEGREE

Neural Networks
Professor. Doctor. Ashish Tendulkar
Indian Institute of Technology, Madras
Multilayer Perceptron Classifier on MNIST

(Refer Slide Time: 00:11)



The screenshot shows a Google Colab notebook interface. At the top, there's a text box stating: "In this colab, we will demonstrate working of MLPClassifier to classify handwritten digits in mnist dataset." Below this, there's an "Imports" section with a list of libraries to be imported: numpy, pandas, matplotlib.pyplot, seaborn, fetch_openml, MLPClassifier, accuracy_score, confusion_matrix, ConfusionMatrixDisplay, plot_confusion_matrix, classification_report, cross_val_score, train_test_split, GridSearchCV, and StratifiedShuffleSplit. The code is written in a code cell, with line numbers 1 through 14. The code imports the necessary libraries and creates a StratifiedShuffleSplit object with n_splits=5, test_size=0.2, and random_state=42.

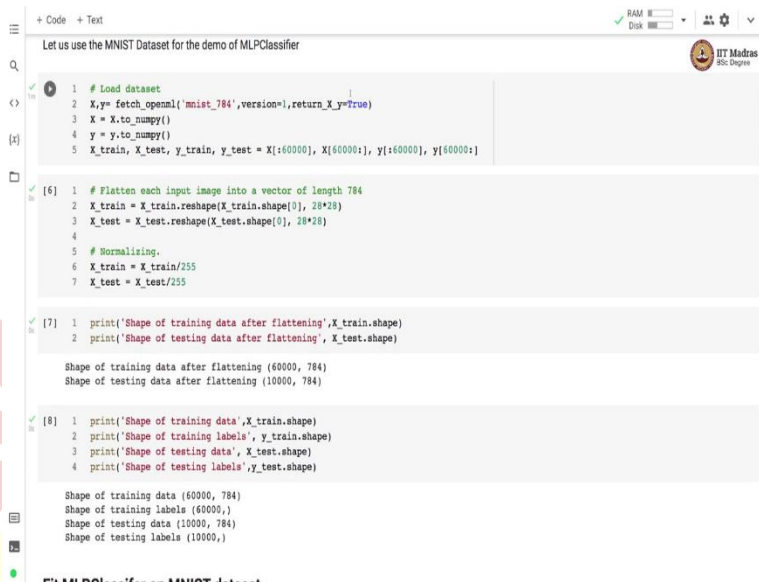
```
[33] 1 import matplotlib.pyplot as plt
      2 import seaborn as sns
      3
      4 import pandas as pd
      5 import numpy as np
      6
      7 from sklearn.datasets import fetch_openml
      8 from sklearn.neural_network import MLPClassifier
      9
     10 from sklearn.metrics import accuracy_score, confusion_matrix, \
     11     ConfusionMatrixDisplay, plot_confusion_matrix, classification_report
     12
     13 from sklearn.model_selection import cross_val_score, train_test_split, \
     14     GridSearchCV, StratifiedShuffleSplit

[4] 1 cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
```

Namaste! Welcome to the next video of Machine Learning Practice Course. In this collab will demonstrate working of MLPClassifier to classify handwritten digits in this dataset. We will begin by importing basic Python libraries like pandas and numpy, matplotlib and seaborn for plotting. The mnist dataset is downloaded using fetch _openml library. The MLPClassifier is imported from sklearn.neural network module.

The model training and evaluation is performed with cross-validation using a bunch of model selection utilities like cross _val _score, train _test _split GridSearchCV and StratifiedShuffleSplit. The performance on the test sample is obtained via accuracy _score, confusion _matrix and classification _report, which are imported from sklearn.metrics module. We create StratifiedShuffleSplit with number of splits = 5 and setting a size 20% examples as test. We will be using this particular cross-validation strategy for training the model.

(Refer Slide Time: 01:27)



```
1 # Load dataset
2 X,y= fetch_openml('mnist_784',version=1,return_X_y=True)
3 X = X.to_numpy()
4 y = y.to_numpy()
5 X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

[6] 1 # Flatten each input image into a vector of length 784
2 X_train = X_train.reshape(X_train.shape[0], 28*28)
3 X_test = X_test.reshape(X_test.shape[0], 28*28)
4
5 # Normalizing.
6 X_train = X_train/255
7 X_test = X_test/255

[7] 1 print('Shape of training data after flattening',X_train.shape)
2 print('Shape of testing data after flattening', X_test.shape)

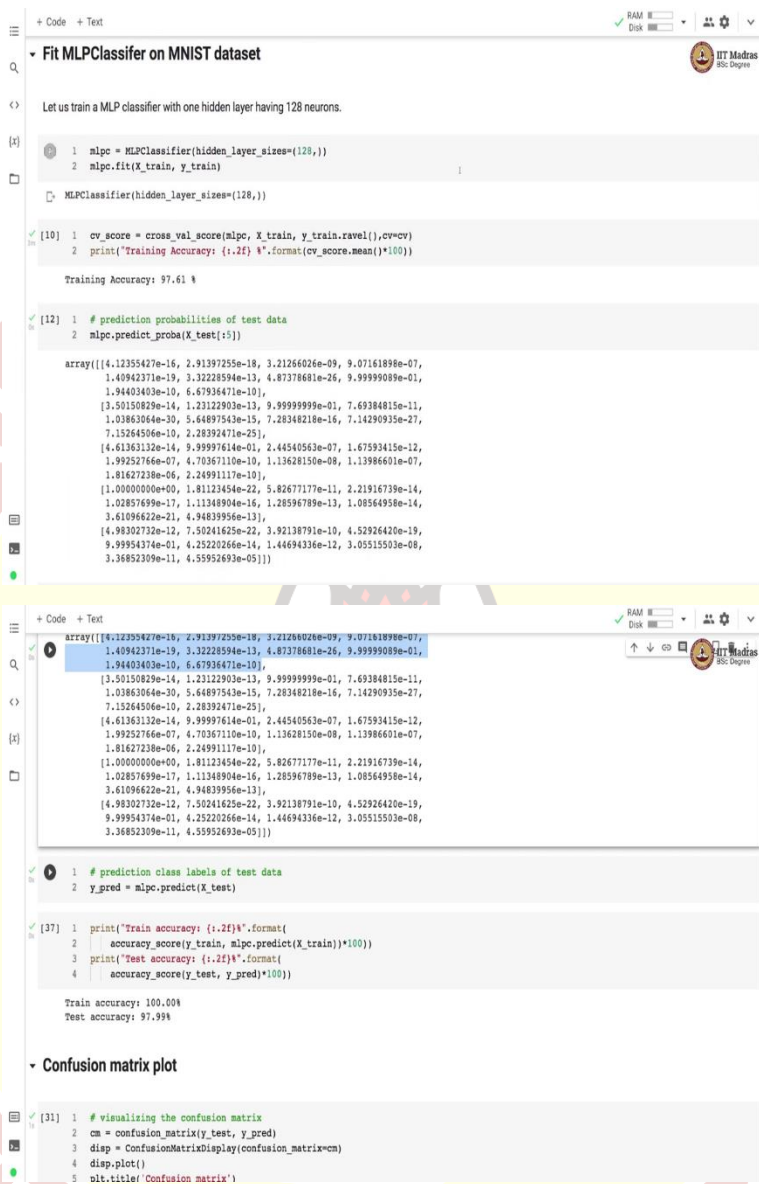
Shape of training data after flattening (60000, 784)
Shape of testing data after flattening (10000, 784)

[8] 1 print('Shape of training data',X_train.shape)
2 print('Shape of training labels', y_train.shape)
3 print('Shape of testing data', X_test.shape)
4 print('Shape of testing labels',y_test.shape)

Shape of training data (60000, 784)
Shape of training labels (60000,)
Shape of testing data (10000, 784)
Shape of testing labels (10000,)
```

Let us use the MNIST dataset and that MNIST dataset is fetched using `fetch_openml` appear by specifying `mnist_784` as the dataset string. As usual, we use first 60,000 examples as studying example, and remaining 10,000 examples as test examples. You flatten each input image into a vector of length 784. As we have been doing throughout this course, we normalize each image by dividing the pixel value by 255 in both training and test set. After flattening, we have training examples, there are 60,000 training examples, each represented with 784 features. And there are 10,000 test examples and each example is represented with 784 features.

(Refer Slide Time: 02:20)



The image displays two screenshots of a Jupyter Notebook interface. The top screenshot shows the initial setup and training of an MLPClassifier. The code includes importing the necessary libraries, creating an MLPClassifier with 128 hidden neurons, fitting it to the training data, and calculating the cross-validation score. The output shows a training accuracy of 97.61%. The bottom screenshot shows the prediction of probabilities for the test data and the calculation of the test accuracy. The output shows a test accuracy of 97.99%.

```
+ Code + Text
Fit MLPClassifier on MNIST dataset

Let us train a MLP classifier with one hidden layer having 128 neurons.

1 mlpc = MLPClassifier(hidden_layer_sizes=(128,))
2 mlpc.fit(X_train, y_train)

MLPClassifier(hidden_layer_sizes=(128,))

[10] 1 cv_score = cross_val_score(mlpc, X_train, y_train.ravel(), cv=cv)
2 print('Training Accuracy: {:.2f} %'.format(cv_score.mean()*100))

Training Accuracy: 97.61 %

[12] 1 # prediction probabilities of test data
2 mlpc.predict_proba(X_test[:5])

array([[4.12355427e-16, 2.91397255e-18, 3.21266026e-09, 9.07161898e-07,
1.40942371e-19, 3.32228594e-13, 4.87378681e-26, 9.99999089e-01,
1.94403403e-10, 6.67936471e-10],
[3.50150829e-14, 1.23122903e-13, 9.99999999e-01, 7.69384815e-11,
1.03863064e-30, 5.64897543e-15, 7.28348218e-16, 7.14290935e-27,
7.15264506e-10, 2.28392471e-25],
[4.61363132e-14, 9.99997614e-01, 2.44540563e-07, 1.67593415e-12,
1.99252766e-07, 4.70367110e-10, 1.13628150e-08, 1.13986601e-07,
1.81627238e-06, 2.2499117e-10],
[1.00000000e+00, 1.81123454e-22, 5.82677177e-11, 2.21916739e-14,
1.02857699e-17, 1.11348904e-16, 1.28596789e-13, 1.08564958e-14,
3.61096622e-21, 4.94839956e-13],
[4.98302732e-12, 7.50241625e-22, 3.92138791e-10, 4.52926420e-19,
9.99954374e-01, 4.25220266e-14, 1.44694336e-12, 3.05515503e-08,
3.36852309e-11, 4.59952693e-05]])

+ Code + Text

array([[4.12355427e-16, 2.91397255e-18, 3.21266026e-09, 9.07161898e-07,
1.40942371e-19, 3.32228594e-13, 4.87378681e-26, 9.99999089e-01,
1.94403403e-10, 6.67936471e-10],
[3.50150829e-14, 1.23122903e-13, 9.99999999e-01, 7.69384815e-11,
1.03863064e-30, 5.64897543e-15, 7.28348218e-16, 7.14290935e-27,
7.15264506e-10, 2.28392471e-25],
[4.61363132e-14, 9.99997614e-01, 2.44540563e-07, 1.67593415e-12,
1.99252766e-07, 4.70367110e-10, 1.13628150e-08, 1.13986601e-07,
1.81627238e-06, 2.2499117e-10],
[1.00000000e+00, 1.81123454e-22, 5.82677177e-11, 2.21916739e-14,
1.02857699e-17, 1.11348904e-16, 1.28596789e-13, 1.08564958e-14,
3.61096622e-21, 4.94839956e-13],
[4.98302732e-12, 7.50241625e-22, 3.92138791e-10, 4.52926420e-19,
9.99954374e-01, 4.25220266e-14, 1.44694336e-12, 3.05515503e-08,
3.36852309e-11, 4.59952693e-05]])

1 # prediction class labels of test data
2 y_pred = mlpc.predict(X_test)

[37] 1 print('Train accuracy: {:.2f} %'.format(
2 accuracy_score(y_train, mlpc.predict(X_train))*100))
3 print('Test accuracy: {:.2f} %'.format(
4 accuracy_score(y_test, y_pred)*100))

Train accuracy: 100.00%
Test accuracy: 97.99%

Confusion matrix plot

[31] 1 # visualizing the confusion matrix
2 cm = confusion_matrix(y_test, y_pred)
3 disp = ConfusionMatrixDisplay(confusion_matrix=cm)
4 disp.plot()
5 plt.title('Confusion matrix')
```

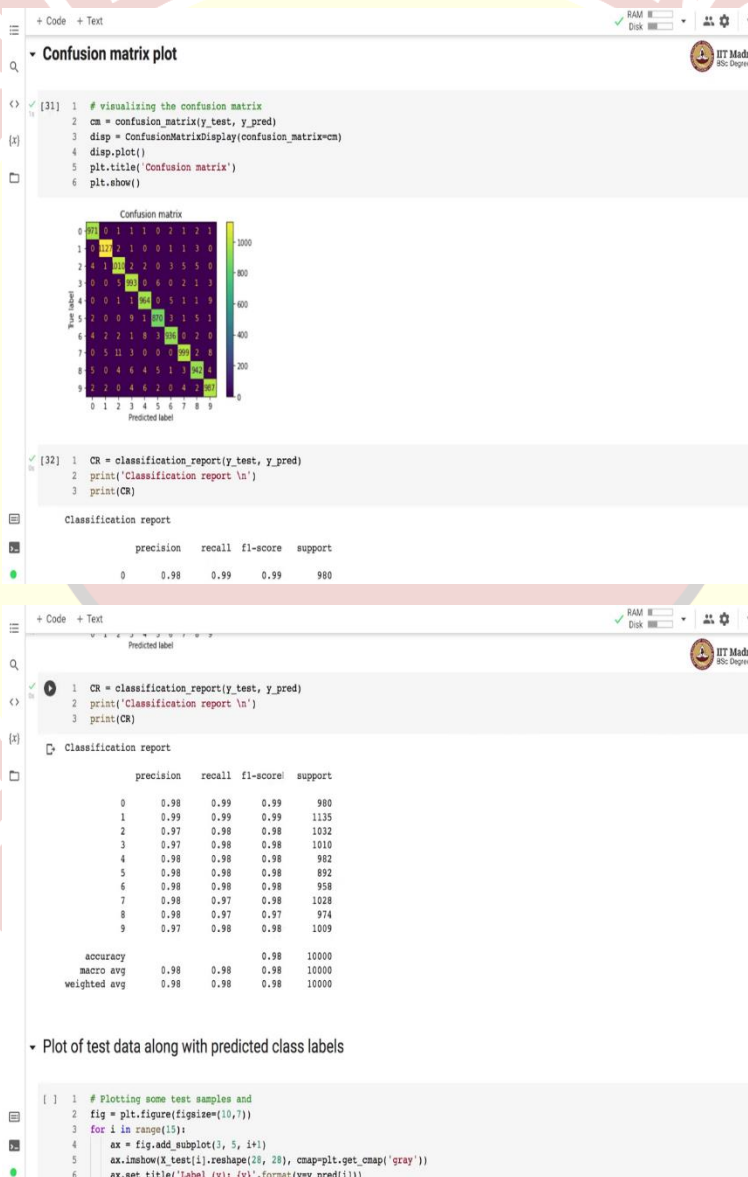
Let us train MLPClassifier with 1 hidden layer having 128 neurons. So, we instantiate MLPClassifier by specifying hidden_layer_sizes, and we used 1 hidden layer with 128 neurons. And we call the fit method by supplying the training feature matrix and training labels as input. After training the MLPClassifier, we obtained cross valid score using the cross-validation which is StratifiedShuffleSplit as specified earlier.

And this cross-validation uses five-fold cross-validation. So, we obtained accuracy of 97.61 % in the cross validated region, we can predict probability on the test data using this model by calling predict_proba method on the MLPClassifier object. And here we have shown probability vectors

for first five test examples. So, each vector that you see here is a probability distribution of that example, over 10 different classes in handwritten digit dataset.

We can use predict method to get class labels of test set. So, now we have predictions for test set in y_pred variable. We use this particular value or these predictions for calculating test accuracy, we also obtain train accuracy and compare them. So, we obtain almost 100% training accuracy and the test accuracy is close to 98%.

(Refer Slide Time: 04:05)

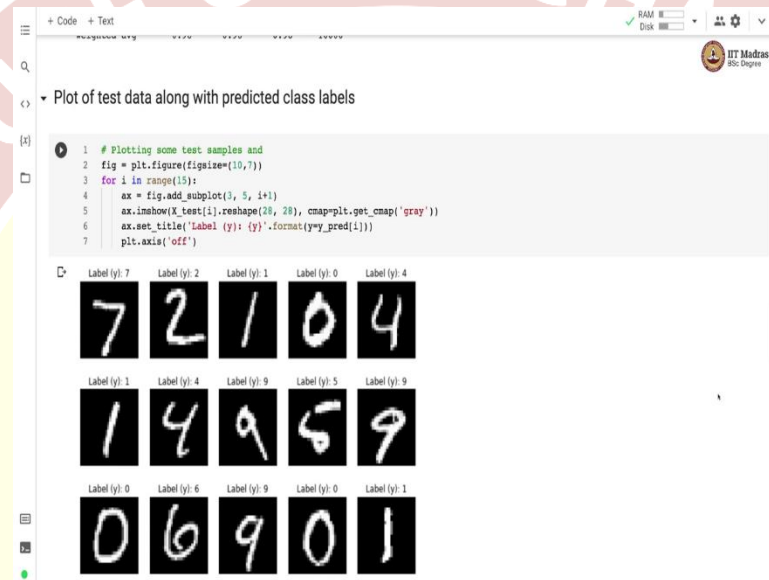


You plot the confusion _matrix for the test examples and you can see that this is almost a perfect confusion _matrix with very little confusion among different classes. So, our diagonal entries are

very small, if you compare this confusion _matrix with other confusion _matrix that we had obtained with other classifiers in the course. The classification _report shows that the test accuracy average test accuracy is 9 %.

And for most of the classes the accuracy is $> \text{ or } = 97\%$. The highest accuracy that was obtained is 0.99 or 99% on class 0 and 1.

(Refer Slide Time: 04:50)



Here we have plotted the test data with their with the predicted labels, and we have plotted 15 examples over here and you can see that most of the examples have been correctly predicted using our MLPClassifier.

(Refer Slide Time: 05:13)


```
+ Code + Text
Visualization of multi layer perceptron weights in hidden layer

• Looking at the learned coefficients of a neural network can provide insight into the learning behavior.
• The input data consists 784 features in the dataset.
• We have used one hidden layer with 128 neurons. Therefore weight matrix has the shape (784, 128).
• We can therefore visualize a single column of the weight matrix as a 28 x 28 pixel image.

[23] 1 w = mlpc.coefs_
      2 w = np.array(w[0])
      3 w.shape
      (784, 128)

[24] 1 w1 = np.array(w[:,0])
      2 w1.shape
      (784, )

[25] 1 w_matrix = w1.reshape(28,28)
      2 fig = plt.figure()
      3 plt.imshow(w_matrix,cmap='gray')
      4 plt.grid(False)
      5 plt.axis(False)
      6 plt.colorbar()
      7 plt.show()
```



Finally, we will visualize the weights that are learned by multi layer perceptron or MLPClassifier. Since you have used 1 hidden layer with 128 neurons, the weight matrix will have shape of 784 by 128. We visualize a single column of weight matrix as 28×28 pixel image. So, you can see that the shape of the weight matrix for the hidden layer is 784 by 128. And for w1 the shape is 784.

(Refer Slide Time: 05:51)

```
+ Code + Text
(784, )

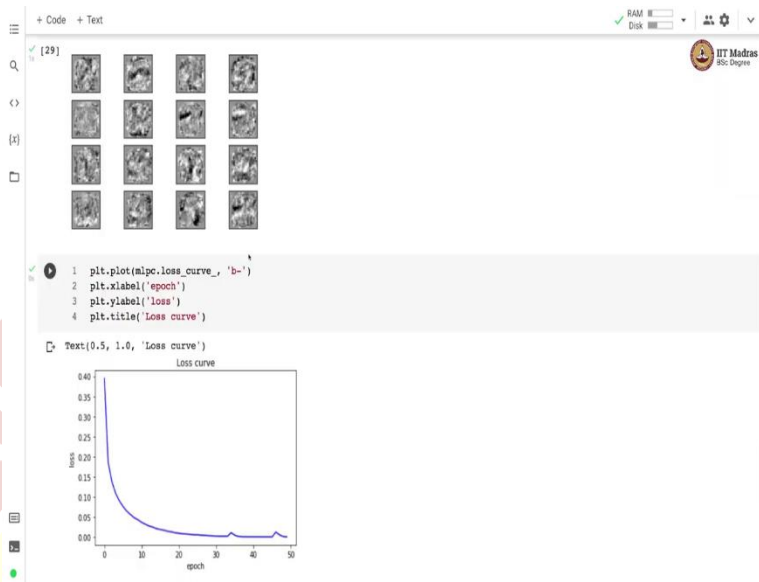
1 w_matrix = w1.reshape(28,28)
2 fig = plt.figure()
3 plt.imshow(w_matrix,cmap='gray')
4 plt.grid(False)
5 plt.axis(False)
6 plt.colorbar()
7 plt.show()

[29] 1 fig, axes = plt.subplots(4, 4)
      2 vmin, vmax = mlpc.coefs_[0].min(), mlpc.coefs_[0].max()
      3 for coef, ax in zip(mlpc.coefs_[0].T, axes.ravel()):
      4     ax.imshow(coef.reshape(28, 28), cmap=plt.cm.gray,
      5             vmin=0.5 * vmin, vmax=0.5 * vmax)
      6     ax.set_xticks(())
      7     ax.set_yticks(())
      8 plt.show()
```



And this is a sample matrix the weight matrix that is learned by our MLPClassifier.

(Refer Slide Time: 05:59)



Finally, we look at loss curve of the train classifier. And you can see that the training loss goes down consistently as we train the model for more epochs, which is a sign of well-trained classifier. So, in this video, we demonstrated how `MLPClassifier` can be used for training, a model for recognizing handwritten digits from MNIST dataset, you can use `MLPClassifier` for many such kind of task, not only in the image recognition but it can also be used for structured datasets.