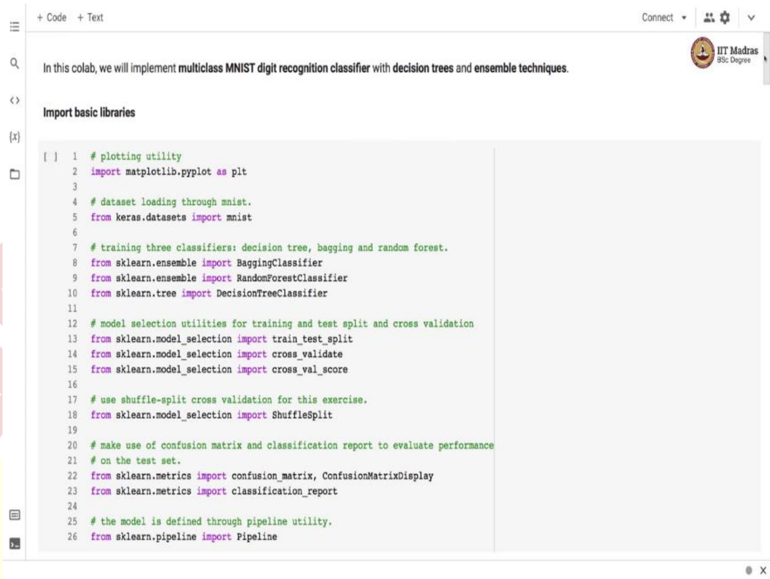
IIT Madras

ONLINE DEGREE

Machine Learning Practice
Professor. Doctor Ashish Tendulkar
Indian Institute of Technology, Madras
Bagging and RandomForestClassifier on MNIST

(Refer Slide Time: 00:10)



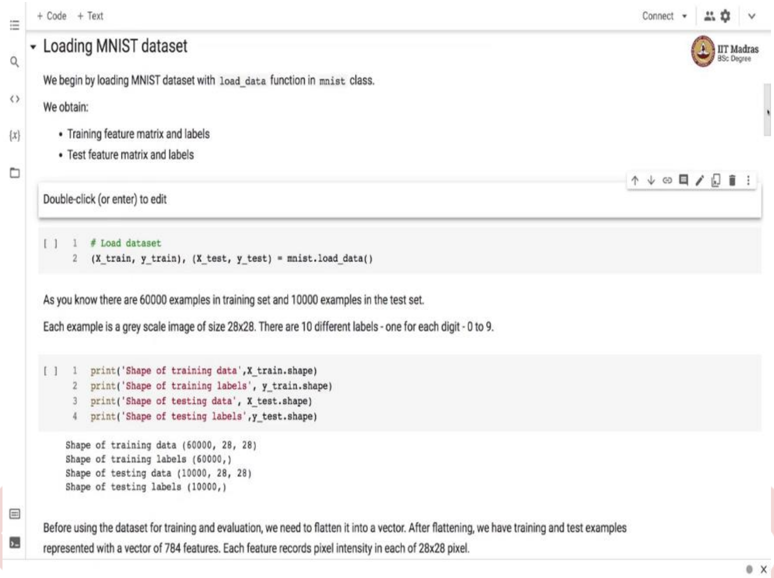
Namaste! Welcome to the next video of Machine Learning Practice Course. In this video, we will implement multiclass MNIST digit recognition classifier with decision trees and ensemble techniques. We will begin by importing basic libraries like matplotlib.pyplot for plotting, then the data will be loaded through MNIST library from keras.datasets module.

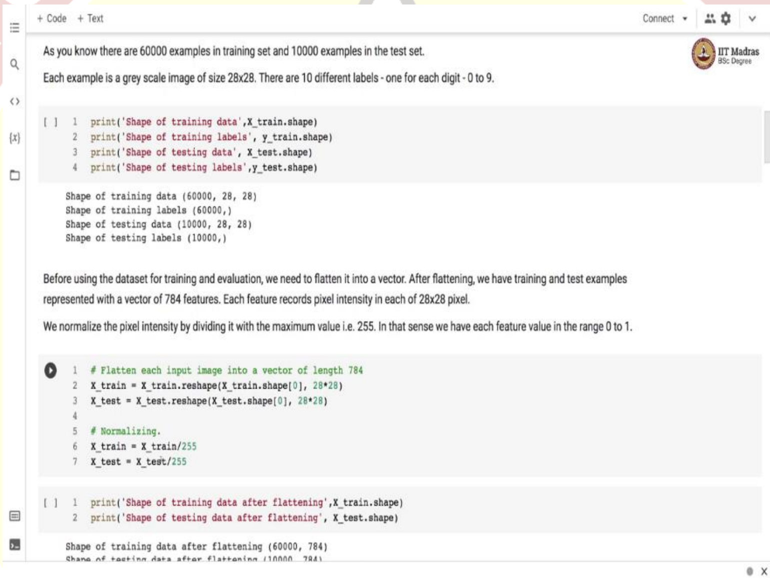
Then there are a bunch of classifiers, 3 classifiers to be specific, the BaggingClassifier, RandomForestClassifier and DecisionTreeClassifier that are imported from the modules. BaggingClassifier and RandomForestClassifiers are implemented as part of sklearn.ensemble module, whereas DecisionTreeClassifier is implemented as part of sklearn.tree module.

Then we import model selection utilities for training and test split which is train _test _split. And for cross-validation, there are a couple of cross-validation utilities that we are importing. We use shuffle split cross-validation for this exercise, and we also import ShuffleSplit from sklearn.model _selection module.

We make use of confusion _matrix and classification _report to evaluate performance on the test set. The confusion _matrix is displayed with ConfusionMatrixDisplay API from sklearn.matrix module. And finally, the model is defined through pipeline utility as we have been doing in all other collapse in this course.

(Refer Slide Time: 01:48)

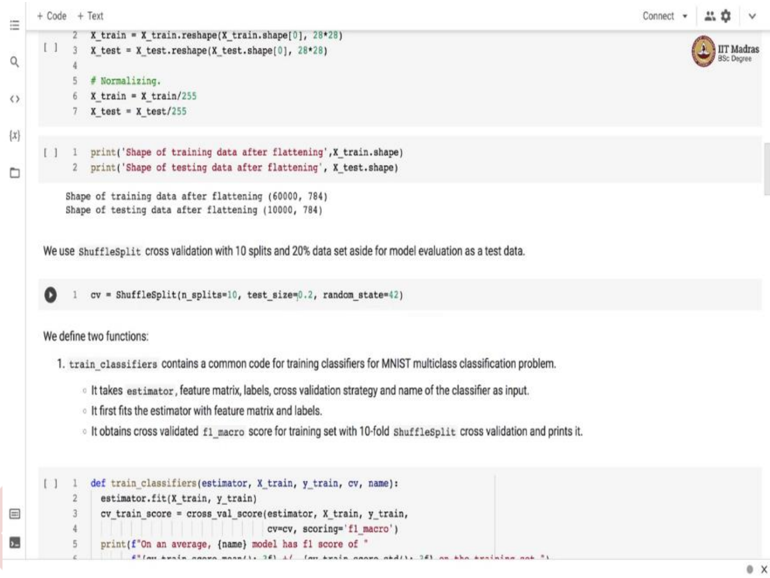




We begin by loading MNIST dataset with load data function in MNIST class. We obtained training feature matrix and labels as well as test feature matrix and labels. As you know, there are 60,000 examples in training set and 10,000 examples in the test set. Every example is a gray scale image of size 28×28. And there are 10 different labels 1 for each digit from 0 to 9.

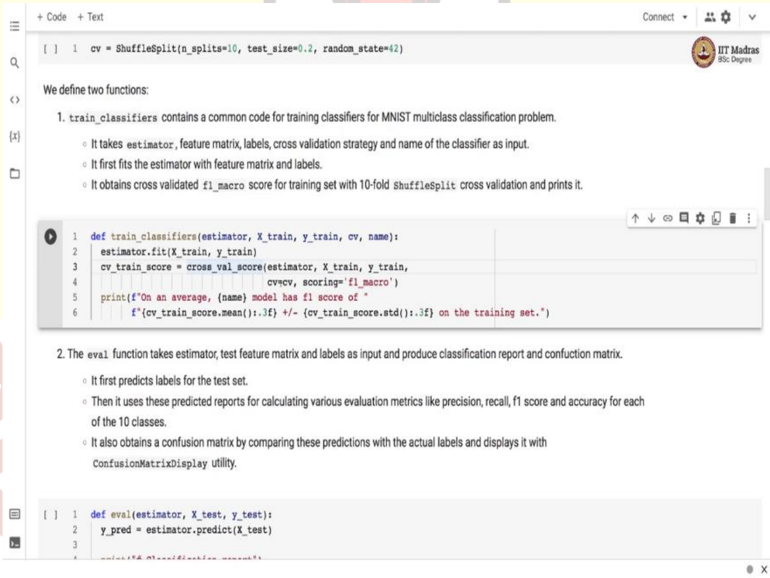
Before using the dataset for training and evaluation, we need to flatten it into a vector. After flattening we have training and test examples represented with 784 features. Each feature records pixel intensity in each of 28×28 pixel image. We normalize the pixel intensity by dividing it with maximum value that is 255. In that sense, we have each feature value now in the range between 0 to 1.

(Refer Slide Time: 02:52)



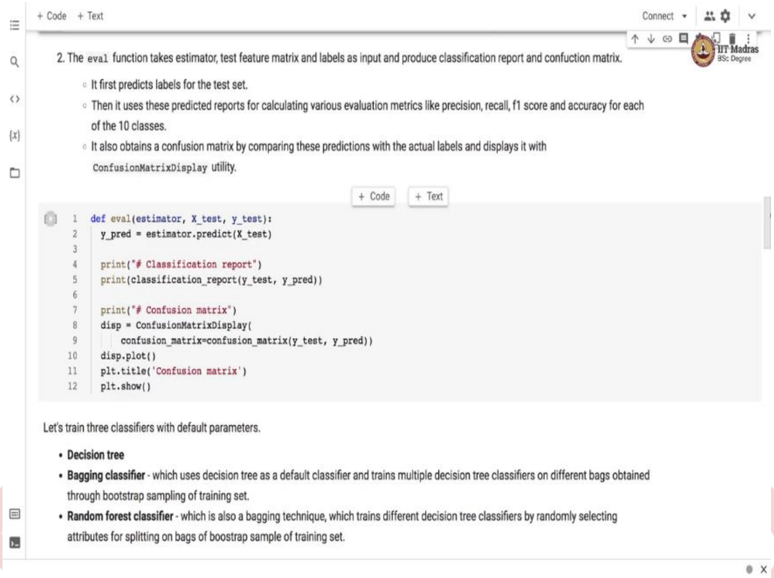
So, you can see that the shape of training data after flattening is 60,000 by 784, whereas the shape of testing data after flattening is 10,000 by 784. Here, we use ShuffleSplit validation with 10 folds and 20% dataset aside for model evaluation as a test data.

(Refer Slide Time: 03:18)



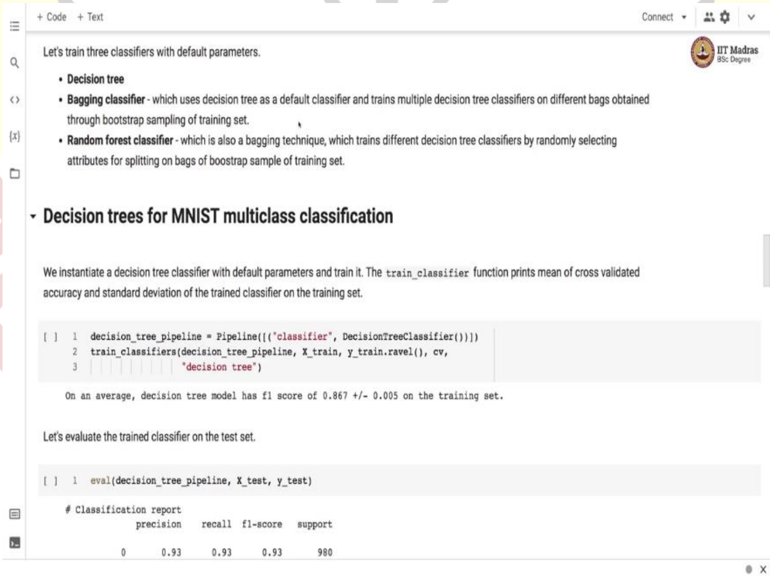
We define 2 functions 1 for training the classifiers and 2 for evaluation. The train classifier function contains common code for training classifiers for MNIST multiclass classification problem. It takes estimator, feature matrix, labels and cross-validation strategy along with the name of the classifier as input. It first fits the estimator with feature matrix and labels as input, and then it obtains cross validated f1 _macro score for training set with 10 for ShuffleSplit cross-validation, and finally it prints the value of f1 _macro.

(Refer Slide Time: 03:58)



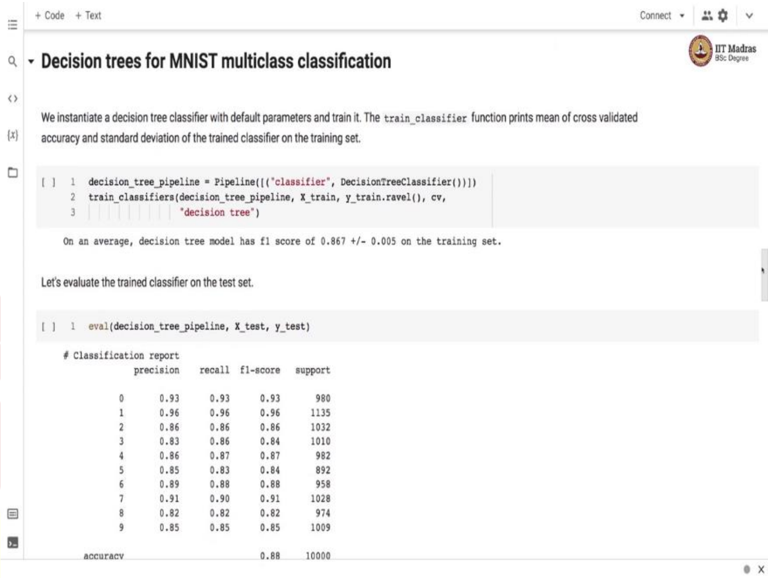
The eval function on the other hand, takes estimator test feature matrix and labels as input and produce classification _report and confusion _matrix as output. It first predicts the labels for the test set then it uses these predicted labels for calculating classification _report, which outputs various evaluation metrics like precision, recall, f1 _score accuracy for each of the 10 classes. It also obtains confusion _matrix by comparing these predictions with the actual label and displays it with ConfusionMatrixDisplay utility.

(Refer Slide Time: 04:42)



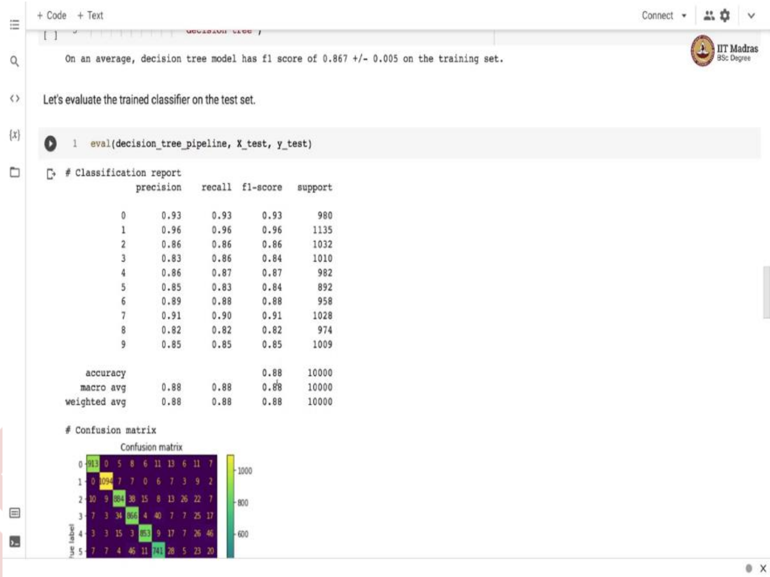
Let us train these 3 classifiers with default parameters. The first classifier is a decision tree, 2 is BaggingClassifier, which also uses decision tree as a default classifier and it trains in fact multiple DecisionTreeClassifiers on different bags obtained through bootstrap sampling or training set. And we also train a RandomForestClassifier, which is also a bagging technique, and it trains different DecisionTreeClassifiers by randomly selecting attributes for splitting on bags or bootstrap sample or training set.

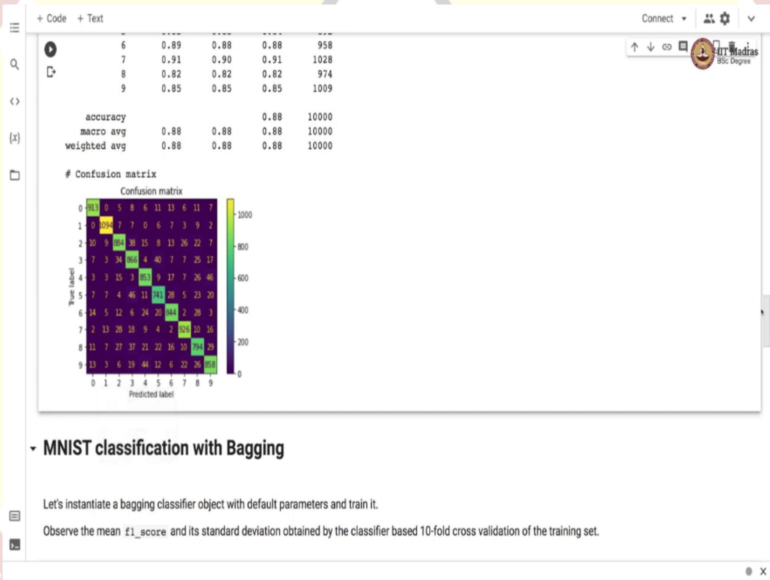
(Refer Slide Time: 05:22)



Let us train the MNIST multiclass classifier with decision trees. We instantiate our DecisionTreeClassifier in the pipeline with default parameters and train it with train _classifiers function. The train _classifiers function brings mean of cross validated accuracy and standard deviation of the trained classifier on the training set. So, here we print f1 _macro score and f1 _macro score from decision tree is 0.86 and the standard deviation is 0.005, which is a very small value. Let us evaluate the DecisionTreeClassifier on the test set.

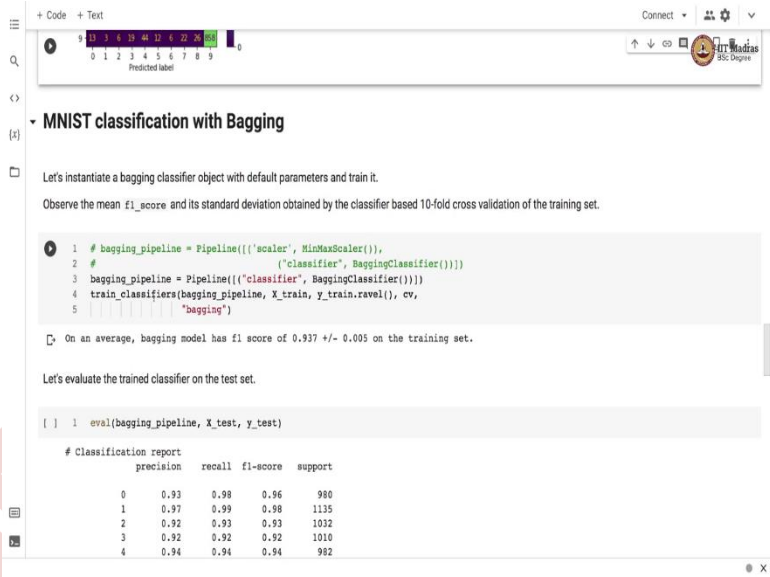
(Refer Slide Time: 06:02)





And you can see that on the test set, it has got accuracy of 0.88. And this is a confusion _matrix for your reference where we have true label on y-axis and predicted labels on the x-axis. There are 10 labels, both on x and y-axis. And on the diagonal, you see the number of images that are correctly classified to their corresponding digits.

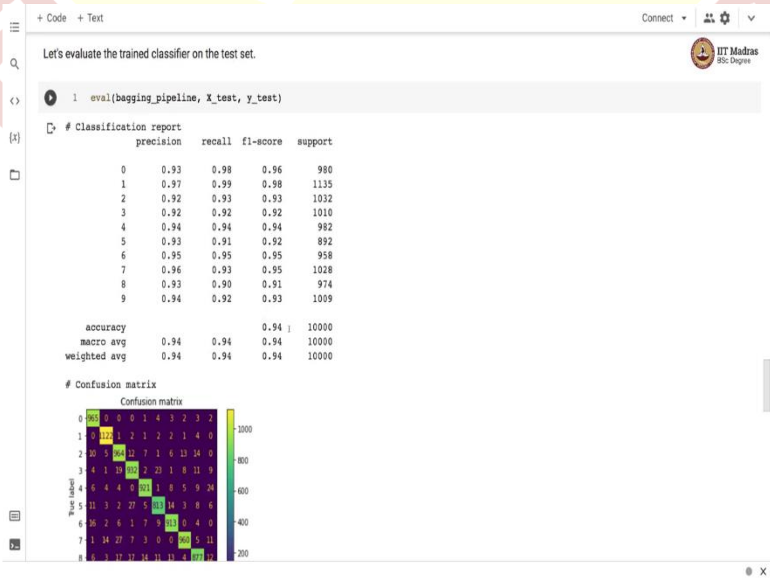
(Refer Slide Time: 06:32)



Next, we train the MNIST classifier with bagging technique. So here, we instantiate a BaggingClassifier with default parameters, and we train it with train _classifiers function. Then we observe f1 _score and standard deviation as obtained by the classifier on the training set based on 10-fold cross-validation.

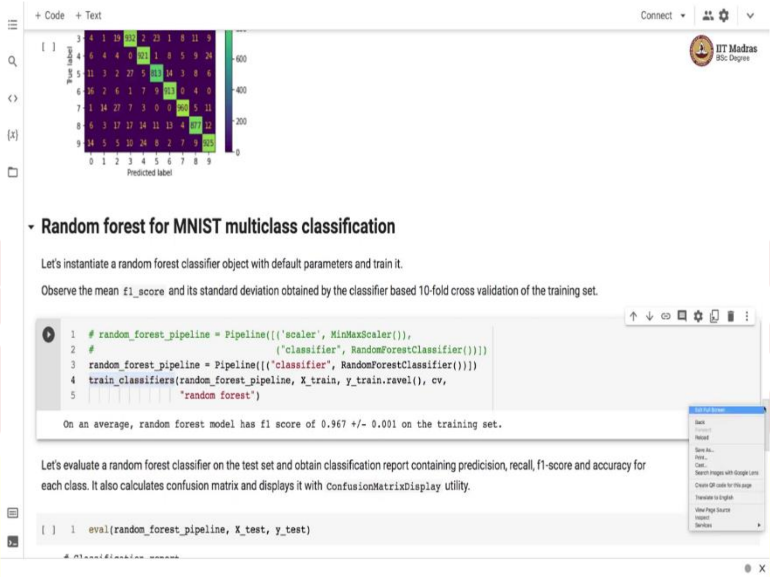
So here, we obtained f1 _score of 0.937 with a small standard deviation on the training set. So, you can see that we are able to get better f1 _score. So, earlier reference code with decision trees was 0.86. And now we have gotten the f1 _score of 0.993, which is about 7% point increase in the f1 _score. And we got this increase just by using bagging which trains multiple DecisionTreeClassifiers.

(Refer Slide Time: 07:25)



Let us evaluate the trained classifier on the test set. So, on the bagging we obtain accuracy of 0.94 which is again 6% point jump over what we obtained with decision tree, in case of decision tree this was 0.88.

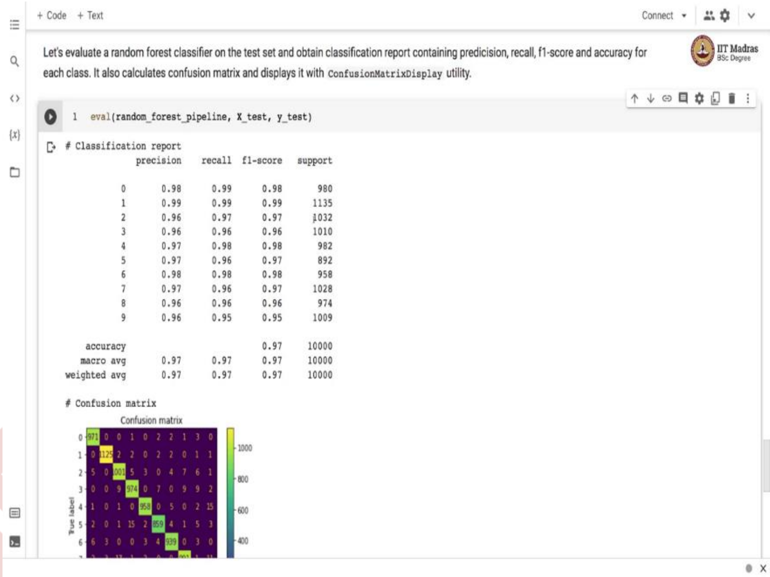
(Refer Slide Time: 07:43)



And finally, we train a RandomForestClassifier for the multiclass classification for recognizing handwritten digits. So here, we instantiate RandomForestClassifier object with default parameters and train it with train _classifiers function, we observe the f1 _score and its standard deviation as obtained by the classifier on the training set based on 10-fold cross-validation.

So, we see that random forest model achieves f1 _score of 0.967 close to 0.997. So random forest, with random forest we are able to get even higher accuracy than the bagging so there is almost 3- 4 % point improvement by using random forest over BaggingClassifier.

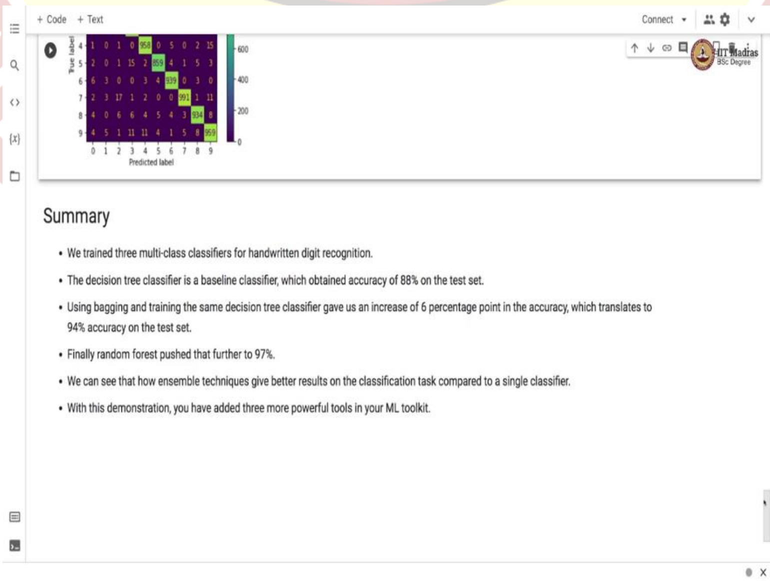
(Refer Slide Time: 08:31)



So let us evaluate a RandomForestClassifier on the test set and obtain classification _report which contains precision recall f1 _score and accuracy for each class. It also calculates confusion _matrix and displays it with ConfusionMatrixDisplay utility. So, you can see that we are often accuracy of 0.97 with RandomForestClassifier and most of the digits are recognized fairly accurately, except for digit 9 which has got a full score of 0.95.

So, this particular accuracy is at par with K-nearest neighbour classifier. K-nearest neighbour classifier also achieves accuracy in this particular range. If you recall, when we train, MNIST digit recognition with K-nearest neighbour classifier.

(Refer Slide Time: 09:21)



So, we trained 3 multiclass classifiers for handwritten digit recognition. The DecisionTreeClassifier is a baseline classifier which obtained accuracy of 88% on the test set. Using bagging and training the same DecisionTreeClassifier gave us an increase of 6% point in accuracy, which translates to 94% accuracy on the test set. Finally, there are no forests pushed the accuracy further to 97%.

So, we can see that how ensemble techniques give better result on classification tasks compared to a single classifier. With this demonstration, you have added 3 more powerful tools in your ML tool kit.