

IIT Madras

ONLINE DEGREE

Machine Learning Program

Demonstration: Multi class Image Classification with Perceptron

(Refer Slide Time: 0:10)



```
+ Code + Text
RAM 8 GB
Disk 100 GB
JIT Madras
B.Tech Degree

Multiclass Classifier (OneVsAll)

• We know that the perceptron is a binary classifier. However, MNIST dataset contains 10 classes. Then how can we extend the idea to handle multi-class problems?
• Solution: Combine multiple binary classifiers and devise a suitable scoring metric.
• Sklearn makes it extremely easy without modifying a single line of code that we have written for the binary classifier.
• Sklearn does this by counting a number of unique elements (10 in this case) in the label vector y_train and converting labels using LabelBinarizer to fit each binary classifier (Remember binary classifier requires binary labels, Tautology :))
• That's all!

1 from sklearn.linear_model import Perceptron
2 from sklearn.preprocessing import LabelBinarizer

[51] 1 clf = Perceptron(random_state=1729)

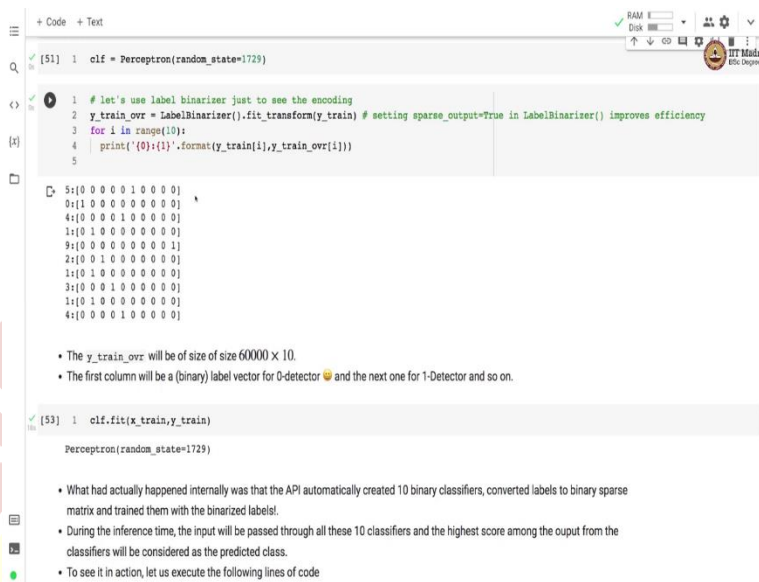
[52] 1 # let's use label binarizer just to see the encoding
2 y_train_ovr = LabelBinarizer().fit_transform(y_train) # setting sparse_output=True in LabelBinarizer() improves efficiency
3 for i in range(10):
4     print('({0}:{1})'.format(y_train[i], y_train_ovr[i]))
5

5:([0 0 0 0 0 1 0 0 0 0])
0:([1 0 0 0 0 0 0 0 0 0])
4:([0 0 0 0 1 0 0 0 0 0])
1:([0 1 0 0 0 0 0 0 0 0])
9:([0 0 0 0 0 0 0 0 0 1])
2:([0 0 1 0 0 0 0 0 0 0])
1:([0 1 0 0 0 0 0 0 0 0])
```

Namaste, welcome to the next video of machine learning practice course. In this video, we will implement multiclass M is digit classification with perceptron. So in the last video, we saw a 0 detector with perceptron. Now what we will do is we will combine multiple binary classifiers and devise a suitable scoring metric for solving the multi class classification problem.

So here, sklearn makes it very easy for us without modifying a single line of code written for binary classification to achieve the multi class classification. All that we are going to do is we are going to use a LabelBinarizer. And we will convert the label into the binary representation.

(Refer Slide Time: 1:05)



```
[51] 1 clf = Perceptron(random_state=1729)

2 # let's use LabelBinarizer just to see the encoding
3 y_train_ovr = LabelBinarizer().fit_transform(y_train) # setting sparse_output=True in LabelBinarizer() improves efficiency
4 for i in range(10):
5     print('{0}:{1}'.format(y_train[i], y_train_ovr[i]))
```

```
5:[0 0 0 0 0 1 0 0 0 0]
9:[1 0 0 0 0 0 0 0 0 0]
4:[0 0 0 0 1 0 0 0 0 0]
1:[0 1 0 0 0 0 0 0 0 0]
9:[0 0 0 0 0 0 0 0 0 1]
2:[0 0 1 0 0 0 0 0 0 0]
1:[0 1 0 0 0 0 0 0 0 0]
3:[0 0 0 1 0 0 0 0 0 0]
1:[0 1 0 0 0 0 0 0 0 0]
4:[0 0 0 0 1 0 0 0 0 0]
```

- The `y_train_ovr` will be of size of size 60000×10 .
- The first column will be a (binary) label vector for 0-detector and the next one for 1-Detector and so on.

```
[53] 1 clf.fit(x_train, y_train)
```

Perceptron(random_state=1729)

- What that actually happened internally was that the API automatically created 10 binary classifiers, converted labels to binary sparse matrix and trained them with the binarized labels.
- During the inference time, the input will be passed through all these 10 classifiers and the highest score among the output from the classifiers will be considered as the predicted class.
- To see it in action, let us execute the following lines of code

We instantiate a perceptron object and we call the binary label transformer which is LabelBinarizer we call fit transform on LabelBinarizer by passing the the label vector. And you can see that the LabelBinarizer has gotten us this particular representation of our labels. So, our original level was 5. So, you can see that the sixth value is 1 the label was 0 the first value is 1 the other values are 0.

So, for 4 you can see that the fifth value is 1 and rest of the values are 0. So, you can see that after you performing labelbinarization our label vector has become a matrix of size $60,000 \times 10$. And this is the training label metrics. And remember that we had 60,000 examples in training and 10,000 examples were set aside for test. So, the first column in this particular label matrix is a binary label vector for the 0 detector, the next one for the for detecting digit 1 and so on.

(Refer Slide Time: 2:36)

```
+ Code + Text
classifiers will be considered as the predicted class.
• To see it in action, let us execute the following lines of code

[54] 1 print('Shape of Weight matrix:{0} and bias vector:{1}'.format(clf.coef_.shape,clf.intercept_.shape))
      Shape of Weight matrix:(10, 784) and bias vector:(10,)

• So it is a matrix of size 10 X 784 where each row represents the weights for a single binary classifier.
• Important difference to note is that there is no sigmoid function associated with the perceptron.
• The class of a perceptron that outputs the maximum score for the input sample is considered as the predicted class.

[55] 1 scores = clf.decision_function(x_train[0].reshape(1,-1))
      2 print(scores)
      3 print('The predicted class: ',np.argmax(scores))

[[-431.88492118  154.366213   -65.23234141  -90.95338716  -189.29568627
 -137.14385236  -99.64604383  -159.69859285  -136.86391288  -199.26120723]]
The predicted class: 1

[56] 1 print('Predicted output:\n')
      2 print(clf.predict(x_train[0].reshape(1,-1)))

Predicted output:

['5']

[57] 1 # get the prediction for all training samples
      2 y_hat = clf.predict(x_train)

[58] 1 print(classification_report(y_train,y_hat))
```

So, now, we simply call the fit method on the perceptron object by passing the feature matrix and the label matrix. And you can see that the shape of the weight matrix has become 10,784. So, we have 784 weights for each of 10 classes. And there are 10 values for buyers one corresponding to each class. So, here when we call the decision function on the perceptron object, we basically get 10 different values of the decision function.

And we can choose the one we can choose the + that has got the highest value of the decision function and you can see that the highest value corresponds to the class 1.

(Refer Slide Time: 3:48)

```
+ Code + Text
['5']

[57] 1 # get the prediction for all training samples
      2 y_hat = clf.predict(x_train)

[58] 1 print(classification_report(y_train,y_hat))


              precision    recall  f1-score   support

0               0.98        0.95        0.97         5923
1               0.94        0.98        0.96         6742
2               0.89        0.90        0.90         5958
3               0.86        0.87        0.87         6131
4               0.89        0.94        0.91         5842
5               0.81        0.88        0.85         5421
6               0.92        0.97        0.94         5918
7               0.91        0.94        0.92         6265
8               0.92        0.77        0.84         5851
9               0.92        0.82        0.87         5949

 accuracy          0.90
 macro avg         0.90
 weighted avg      0.91

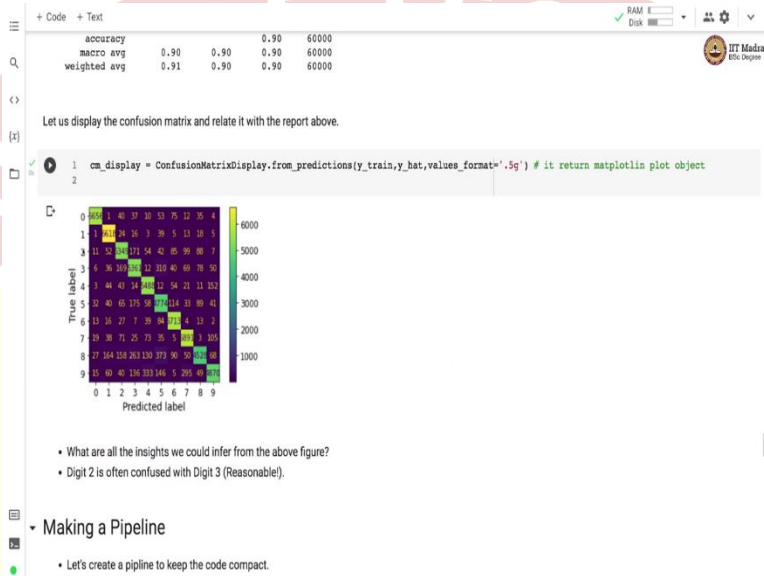
Let us display the confusion matrix and relate it with the report above.

[59] 1 cm_display = ConfusionMatrixDisplay.from_predictions(y_train,y_hat,values_format='.5g') # it return matplotlib plot object
      2
```



We can use classification report in order to get a precision recall and f1 score for each of the each of the classes. And you can see that for class 0 we have precision of 0.98 and recall a 0.95 with f1 score of 0.97 for class 7 we have precision of 0.91 recall of 0.94 and f1 score of 0.92. And overall accuracy is 0.90.

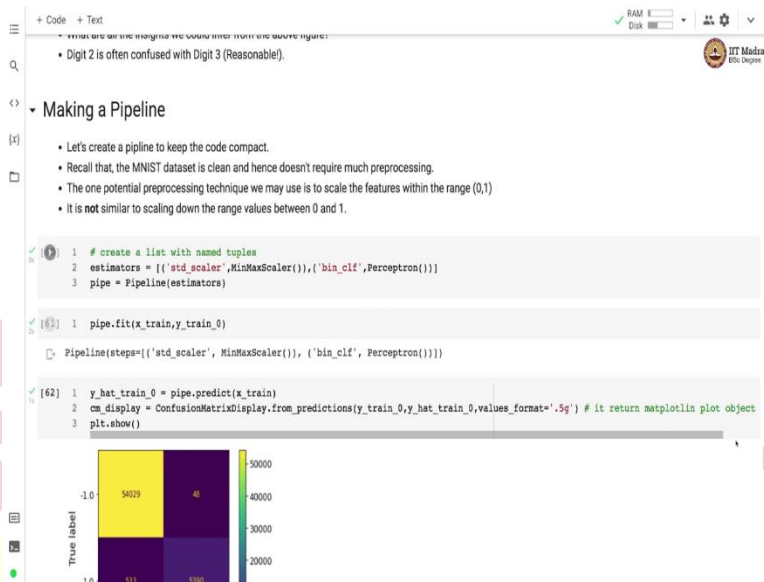
(Refer Slide Time: 4:26)



We can display the confusion matrix for all the for all the classes and you can see that how the labels are accurately predicted. So, accurate predictions are on the diagonal and the misclassification are of diagonal. So, you can see that you can see that there are some misclassification for digit 8 as well as for digit 9. And digit 8 for example is getting confused with digit 3 and also with digit 5.

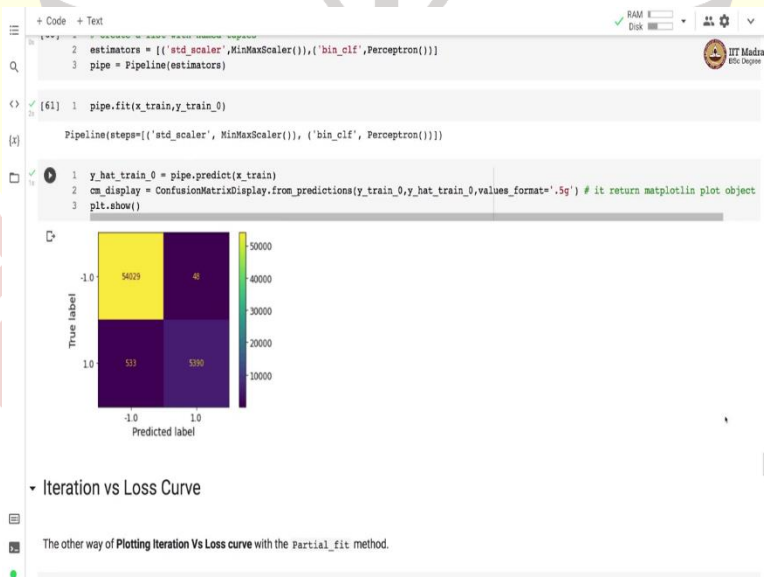
And whereas, digit 9 is getting confused with digit 4, there is some confusion over here. You can also see that there is a confusion between digit 2 and digit 3.

(Refer Slide Time: 5:27)



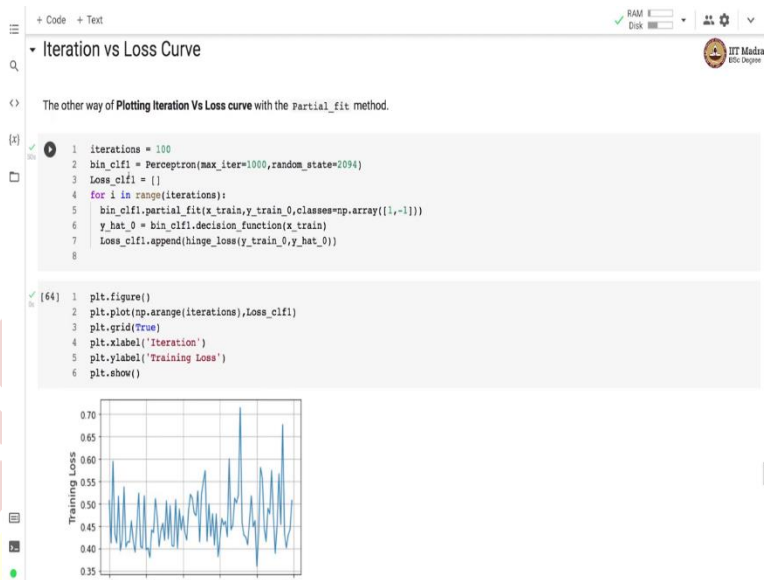
So, we have seen in relation that it is usually a good idea to write all the pre processing operation and our estimator in a pipeline. So, we are creating a pipeline for the perceptron. So, we are first performing scaling with min max scalar followed by the perceptron estimator and we call the fit function on the pipeline object.

(Refer Slide Time: 5:58)



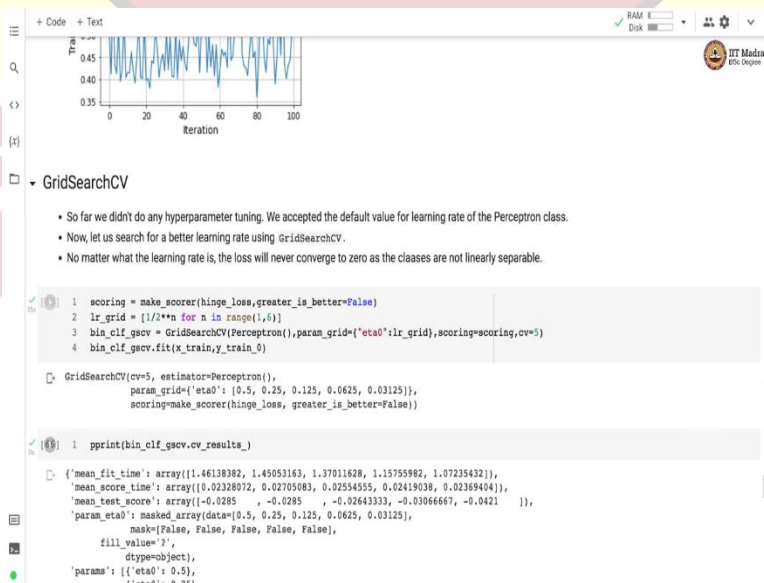
And we can get a confusion matrix and this is a confusion matrix for the 0 detector.

(Refer Slide Time: 6:09)



So, in machine learning techniques, we were generally advocating use of what is called as learning curve, where we have iteration on x axis and training loss on y axis. So, there is no direct way of obtaining such a learning curve in sklearn. So, here we are making use of partial fit method in order to plot this learning curve. So, what happens is, here we use or make use of partial fit, and we run the partial fit in an iterative manner. And for each iteration, we record the loss and then bring that particular loss against the iteration.

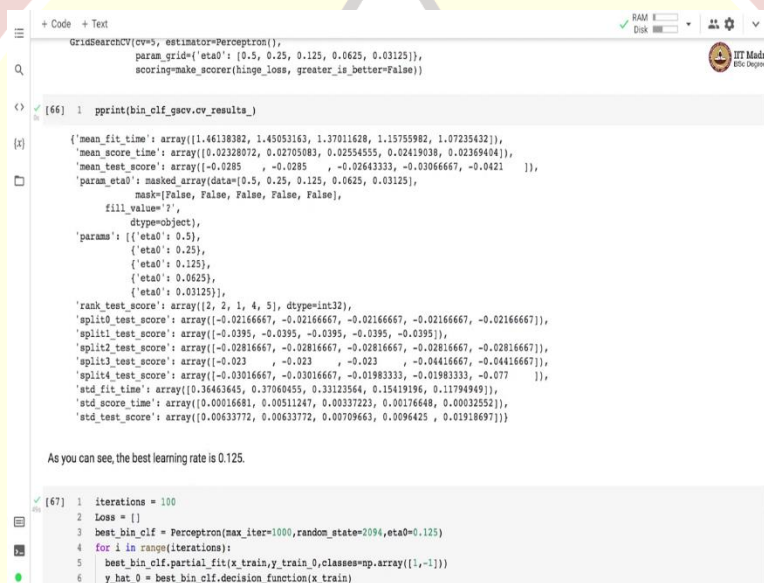
(Refer Slide Time: 7:00)



So far, we have not done any hyper parameter tuning for perceptron. So we accepted the default values of learning rate for perceptron. So now here, what we will do is we will try to perform a grid search for the learning rate of perceptron. So, here we specify the parameter grid with the parameter eta 0 which is the initial learning rate. And we are setting a parameter grid with different values which are in the range when 1 by 2 raise to basically these are inverse powers of 2 between 1 and 6.

So, you can see that we are trying various values for learning rate which are 0.5, 0.25, 0.125, 0.0625, 0.03125 these are different values that are tried out for the learning rate.

(Refer Slide Time: 8:11)



```

+ Code + Text
GridSearchCV(cv=5, estimator=Perceptron(),
              param_grid={'eta0': [0.5, 0.25, 0.125, 0.0625, 0.03125]},
              scoring=make_scorer(hinge_loss, greater_is_better=False))

[66] 1 pprint(bin_clf_gscv_results_)

{'mean_fit_time': array([1.46138382, 1.45053163, 1.37011628, 1.15755982, 1.07235432]),
 'mean_score_time': array([0.02328072, 0.02705083, 0.02545555, 0.02419038, 0.02369404]),
 'mean_test_score': array([-0.0285, -0.0285, -0.02643333, -0.03066667, -0.0421]),
 'param_eta0': masked_array(data=[0.5, 0.25, 0.125, 0.0625, 0.03125],
                             mask=[False, False, False, False, False],
                             fill_value='?',
                             dtype=object),
 'rank_test_score': array([2, 2, 1, 4, 5], dtype=int32),
 'split0_test_score': array([-0.02166667, -0.02166667, -0.02166667, -0.02166667, -0.02166667]),
 'split1_test_score': array([-0.0395, -0.0395, -0.0395, -0.0395, -0.0395]),
 'split2_test_score': array([-0.02816667, -0.02816667, -0.02816667, -0.02816667, -0.02816667]),
 'split3_test_score': array([-0.023, -0.023, -0.023, -0.04416667, -0.04416667]),
 'split4_test_score': array([-0.03016667, -0.03016667, -0.01983333, -0.01983333, -0.077]),
 'std_fit_time': array([0.36463645, 0.37060455, 0.33123564, 0.15419196, 0.11794949]),
 'std_score_time': array([0.00016681, 0.00511247, 0.00337223, 0.00176648, 0.00032552]),
 'std_test_score': array([0.00633772, 0.00633772, 0.00709463, 0.0096425, 0.01918697])}

As you can see, the best learning rate is 0.125.

[67] 1 iterations = 100
     2 Loss = []
     3 best_bin_clf = Perceptron(max_iter=1000, random_state=2094, eta0=0.125)
     4 for i in range(iterations):
     5     best_bin_clf.partial_fit(x_train, y_train, classes=np.array([1, -1]))
     6     y_hat_0 = best_bin_clf.decision_function(x_train)
  
```

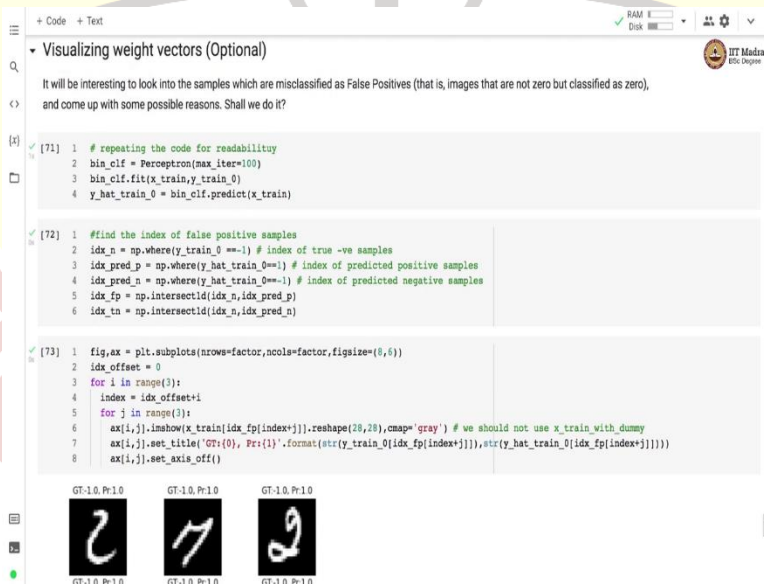
And after performing the grid search, you can see that the best learning rate was 0.125. And why this is the best learning rate? Because you can see that the ranking based on the test score the learning rate of 0.125 got the rank of 1. So, this is the best learning rate obtained to the grid search.

(Refer Slide Time: 8:47)



We can compare the training loss with the default learning rate of 1 and a one octane to the grid search. And you can see that the training loss is much lower for the one that is discovered to the grid search.

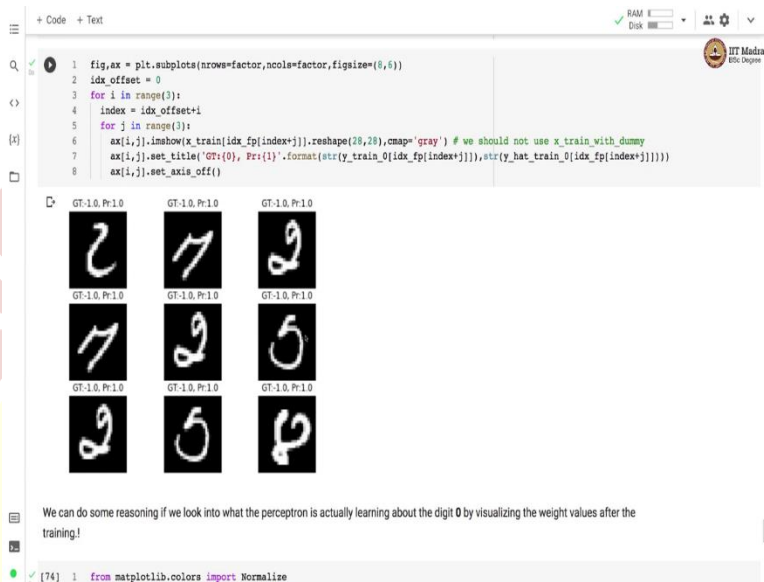
(Refer Slide Time: 9:12)



We can also obtain the best estimator through grid search using best_estimator_member variable. And for the best estimator for the 0 detector, we got the f1 score of 0.95 for class 1. So, as an exercise it will be interesting to compare the classification report when the learning rate was 1 you

can find is earlier in the in the colab. So finally, what we will do is we will visualize the weight vectors obtained through the perceptron algorithm.

(Refer Slide Time: 10:01)



So here, we have various digits. So, the ground truth is - 1, but it is predicted to be of + 1. And remember this is these are all we are solving we are showing the weight vectors learn to the 0 detector. So, that is why we have two labels - 1 and + 1. So, this one is 7, but it is predicted to be 0. So, these are some of the some of the misclassification which are predicted to be 0 even though they are they are not 0.

(Refer Slide Time: 10:40)



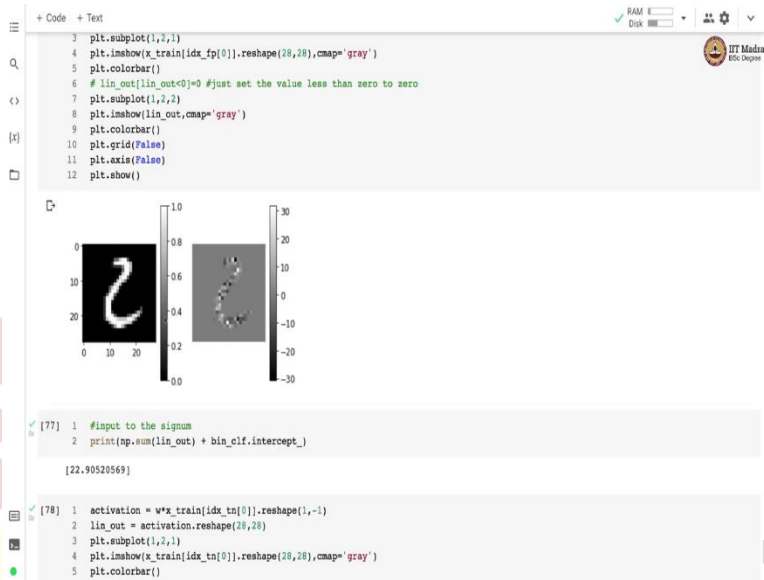
So, let us try to first print the weight vector of all the of the 0 predictor. And this is how the weight vector looks like the fender values or the fender color denotes a higher value of the weights, whereas the darker shades represent lower values of the of the weights. So, the lower values are here or - 60 and higher values are upward of 20.

(Refer Slide Time: 11:14)



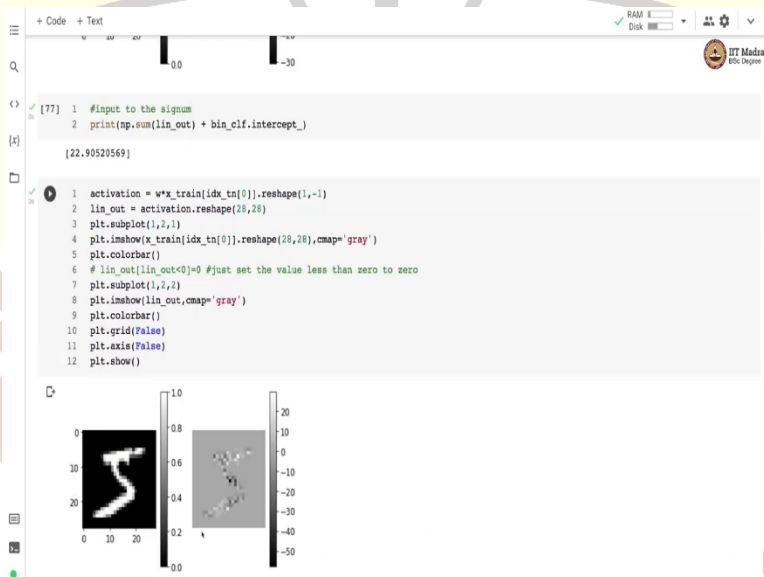
So, we can calculate the activation by multiplying the weight vector and the under pixel values. And we can look at the activation map of different digits.

(Refer Slide Time: 11:29)



So, for this particular digit, this activation map that we obtained by multiplying the pixel values with the weights, and you can see that when we perform the linear combination, the value that we obtain is 22.9, which is > 0 . And hence, we basically label this particular image as 0.

(Refer Slide Time: 12:00)



Let us look at another image which is which has 5 in it and this activation map or 5.

(Refer Slide Time: 12:08)



And if we calculate the linear combination, the value is coming out to be - 293.33. So, since this value is < 0 we label this particular image with - 1. So, this is basically non 0 image. So, this is how you can visualize activations and weight vectors. So in this video, we studied how to perform multi class classification with perceptron. So, what we did is we took the label vector applied label by riser transformation to it.

And obtain one hot encoding representation of labels. And after obtaining the one hot encoding representation of the label. We use exactly the same code that we used for the binary case and perform multi class classification with perceptron. We also studied how to visualize the weights and activation maps for different digits. And this will help us in in detecting any issues in prediction.