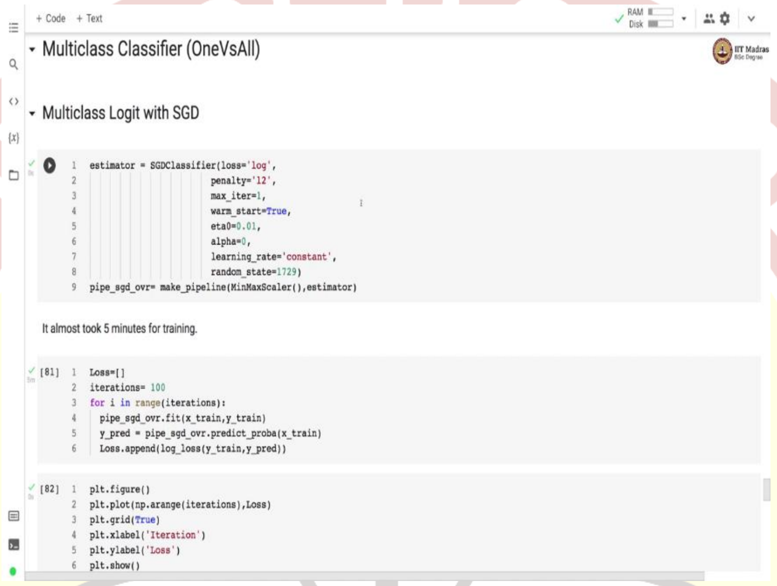
INDIAN INSTITUTE OF TECHNOLOGY MADRAS • सिद्धिर्भवति कर्मजा •

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Doctor Ashish Tendulkar
Indian Institute of Technology Madras
Demonstration: Multiclass classifier on MNIST Dataset

(Refer Slide Time: 00:10)



Multiclass Classifier (OneVsAll)

Multiclass Logit with SGD

```
estimator = SGDClassifier(loss='log',
                          penalty='l2',
                          max_iter=1,
                          warm_start=True,
                          eta0=0.01,
                          alpha=0,
                          learning_rate='constant',
                          random_state=1729)
pipe_sgd_ovr= make_pipeline(MinMaxScaler(),estimator)
```

It almost took 5 minutes for training.

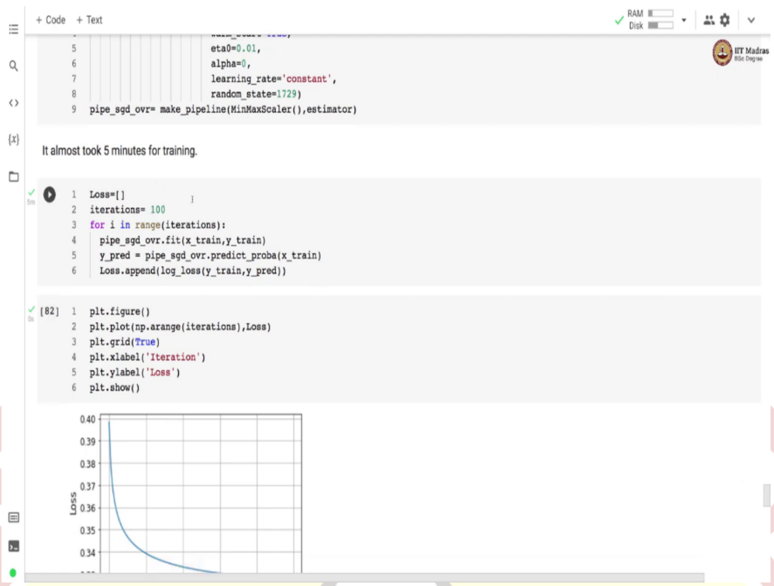
```
[81]
Loss=[]
iterations= 100
for i in range(iterations):
  pipe_sgd_ovr.fit(x_train,y_train)
  y_pred = pipe_sgd_ovr.predict_proba(x_train)
  Loss.append(log_loss(y_train,y_pred))
```

```
[82]
plt.figure()
plt.plot(np.arange(iterations),Loss)
plt.grid(True)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.show()
```

Namaste! Welcome to the next video of Machine Learning Practice course. In this video, we will use multiclass classification in order to solve the MNIST digit classification problem. In the last few videos, we solved a binary classification problem in order to detect the digit 0 from the other digits. So here, we solve the actual MNIST digit recognition problem, where we will train a multiclass classifier that recognizes one digit from another.

So first, we use the SGDclassifier for multiclass classification. So, we train SGDclassifier without regularization, here, we set alpha to 0, we use the cross-entropy loss and the learning rate is constant and is set to 0.01. We have two stages in the pipeline, first is the scaling followed by the SGDclassifier.

(Refer Slide Time: 01:15)

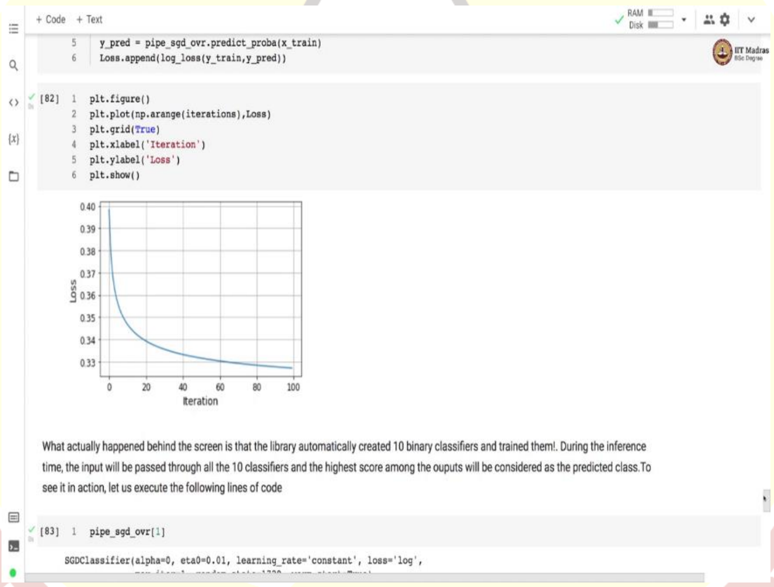


```
                          eta0=0.01,
                          alpha=0,
                          learning_rate='constant',
                          random_state=1729)
pipe_sgd_ovr= make_pipeline(MinMaxScaler(),estimator)
```

It almost took 5 minutes for training.

```
Loss=[]
iterations= 100
for i in range(iterations):
  pipe_sgd_ovr.fit(x_train,y_train)
  y_pred = pipe_sgd_ovr.predict_proba(x_train)
  Loss.append(log_loss(y_train,y_pred))
```

```
[82]
plt.figure()
plt.plot(np.arange(iterations),Loss)
plt.grid(True)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.show()
```



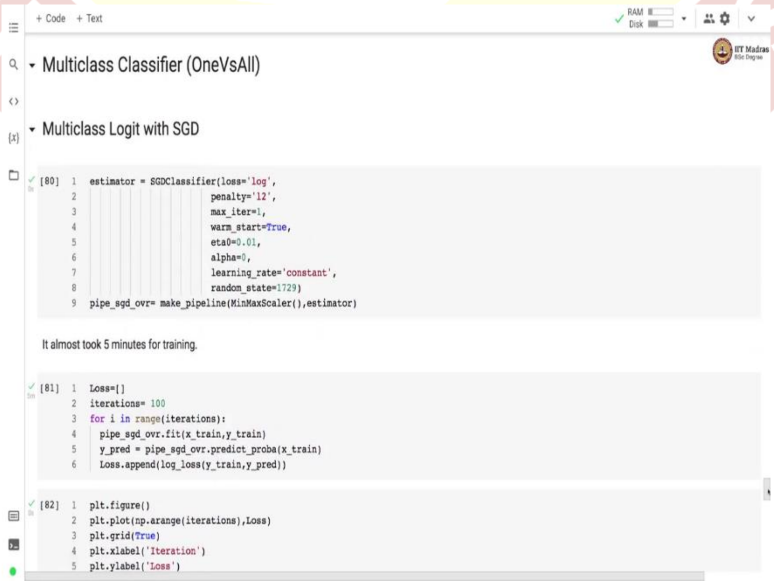
```
  y_pred = pipe_sgd_ovr.predict_proba(x_train)
  Loss.append(log_loss(y_train,y_pred))
```

```
[82]
plt.figure()
plt.plot(np.arange(iterations),Loss)
plt.grid(True)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.show()
```

What actually happened behind the screen is that the library automatically created 10 binary classifiers and trained them!. During the inference time, the input will be passed through all the 10 classifiers and the highest score among the ouputs will be considered as the predicted class.To see it in action, let us execute the following lines of code

```
[83]
pipe_sgd_ovr[1]
```

```
SGDClassifier(alpha=0, eta0=0.01, learning_rate='constant', loss='log',
```



Multiclass Classifier (OneVsAll)

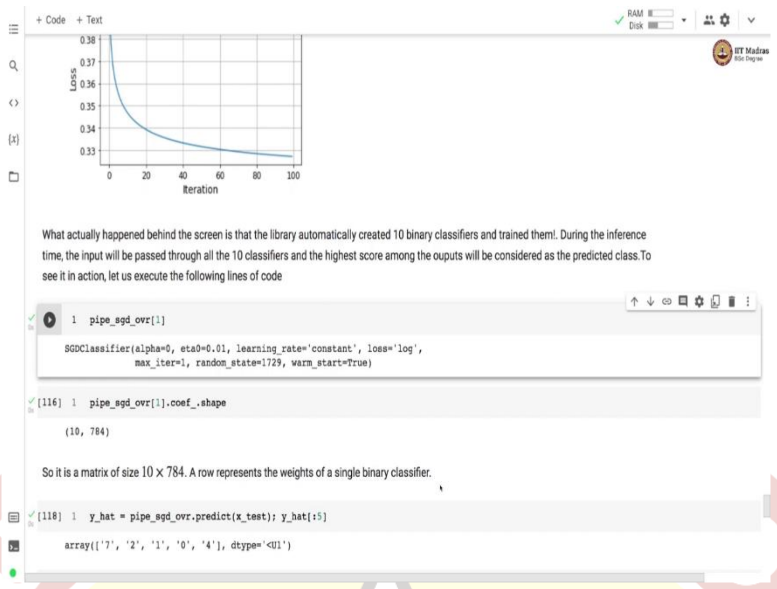
Multiclass Logit with SGD

```
[80]
estimator = SGDClassifier(loss='log',
                          penalty='l2',
                          max_iter=1,
                          warm_start=True,
                          eta0=0.01,
                          alpha=0,
                          learning_rate='constant',
                          random_state=1729)
pipe_sgd_ovr= make_pipeline(MinMaxScaler(),estimator)
```

It almost took 5 minutes for training.

```
[81]
Loss=[]
iterations= 100
for i in range(iterations):
  pipe_sgd_ovr.fit(x_train,y_train)
  y_pred = pipe_sgd_ovr.predict_proba(x_train)
  Loss.append(log_loss(y_train,y_pred))
```

```
[82]
plt.figure()
plt.plot(np.arange(iterations),Loss)
plt.grid(True)
plt.xlabel('Iteration')
plt.ylabel('Loss')
```



What actually happened behind the screen is that the library automatically created 10 binary classifiers and trained them!. During the inference time, the input will be passed through all the 10 classifiers and the highest score among the ouputs will be considered as the predicted class.To see it in action, let us execute the following lines of code

```
pipe_sgd_ovr[1]
```

```
SGDClassifier(alpha=0, eta0=0.01, learning_rate='constant', loss='log',
              max_iter=1, random_state=1729, warm_start=True)
```

```
[116] pipe_sgd_ovr[1].coef_.shape
```

```
(10, 784)
```

So it is a matrix of size 10×784. A row represents the weights of a single binary classifier.

```
[118] y_hat = pipe_sgd_ovr.predict(x_test); y_hat[:5]
```

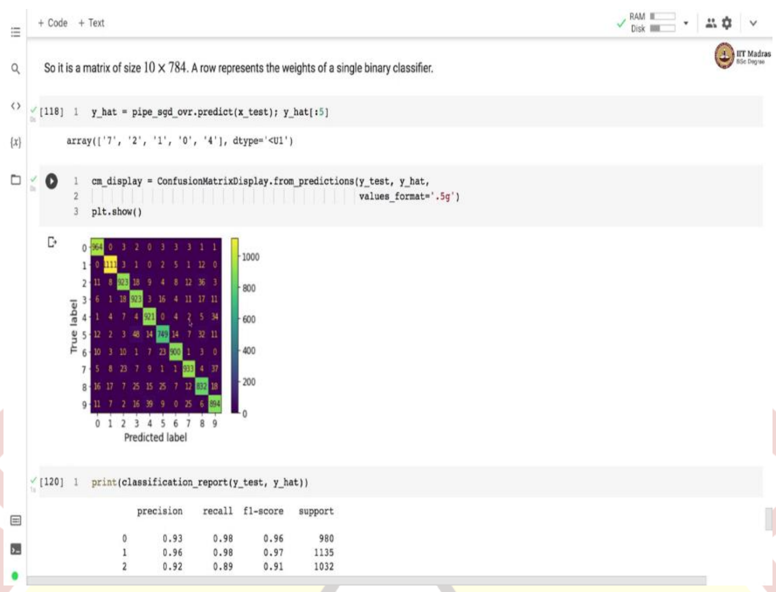
```
array(['7', '2', '1', '0', '4'], dtype='<U1')
```

Let us train the SGDclassifier for 100 iterations, it took almost five minutes for training and we have plotted the learning curve. So, you can see that this learning curve is like an ideal classifier with every iteration the loss is coming down. So, here we did not do anything special to accommodate the multiclass setting.

So, what is happening actually behind the scene is that library automatically created 10 different binary classifiers, and trained them. And during the inference, the input will be passed through all the 10 classifiers and the highest score among the output will be considered as the predicted class.

Since, we are using pipelined the SGDclassifier is the second stage in the pipeline, and it can be accessed as pipe _sgd _ovr, which is the pipeline object and the index 1 of it will be the SGDclassifier. We can look at the shape of the weight matrix of this multi-class SGDclassifier, and we found that the shape is $\frac{10}{784}$. So, each row in this matrix which is of the size $\frac{10}{784}$ represent the weights of a single binary classifier.

(Refer Slide Time: 02:52)



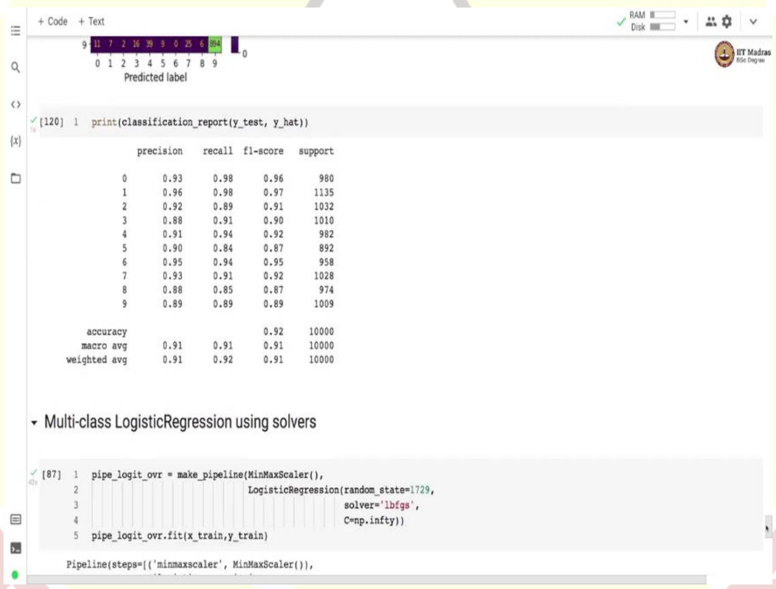
So it is a matrix of size 10×784. A row represents the weights of a single binary classifier.

```
[118] y_hat = pipe_sgd_ovr.predict(x_test); y_hat[:5]

array(['7', '2', '1', '0', '4'], dtype='<U1')
```

```
cm_display = ConfusionMatrixDisplay.from_predictions(y_test, y_hat,
                                                      values_format='.5g')
plt.show()
```

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	964	0	3	2	0	3	3	3	1	1
1	0	1111	3	1	0	2	5	1	12	0
2	11	8	923	18	9	4	8	12	36	3
3	6	1	18	923	3	16	4	11	17	11
4	1	4	7	4	921	0	4	2	5	34
5	12	2	3	48	14	749	14	7	32	11
6	10	3	10	1	7	23	900	1	3	0
7	5	8	23	7	9	1	1	933	4	37
8	16	17	7	25	15	25	7	12	832	18
9	11	7	2	16	39	9	0	25	6	894

```
[120] print(classification_report(y_test, y_hat))

              precision    recall  f1-score   support

           0       0.93      0.98      0.96       980
           1       0.96      0.98      0.97      1135
           2       0.92      0.89      0.91      1032
```



```
[120] print(classification_report(y_test, y_hat))

              precision    recall  f1-score   support

           0       0.93      0.98      0.96       980
           1       0.96      0.98      0.97      1135
           2       0.92      0.89      0.91      1032
           3       0.88      0.91      0.90      1010
           4       0.91      0.94      0.92       982
           5       0.90      0.84      0.87       892
           6       0.95      0.94      0.95       958
           7       0.93      0.91      0.92      1028
           8       0.88      0.85      0.87       974
           9       0.89      0.89      0.89      1009

    accuracy                           0.92     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.92      0.91     10000
```

Multi-class LogisticRegression using solvers

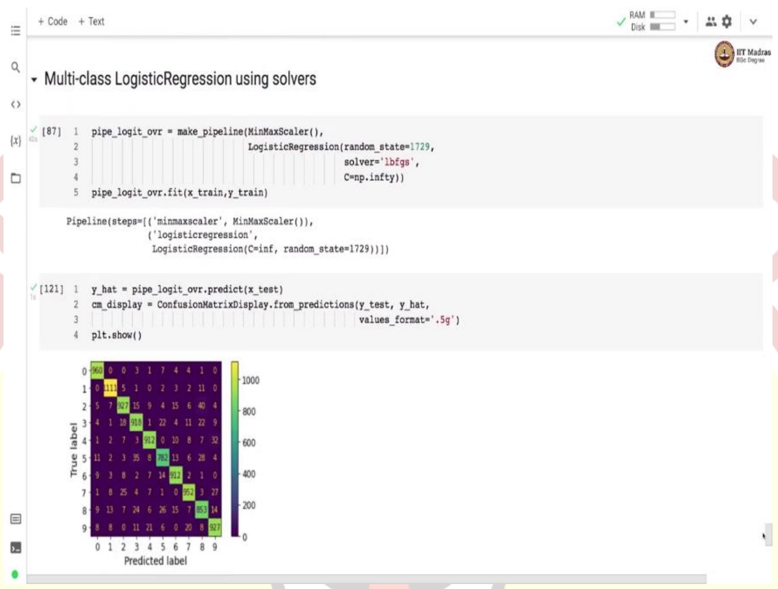
```
[87] pipe_logit_ovr = make_pipeline(MinMaxScaler(),
                                    LogisticRegression(random_state=1729,
                                                       solver='lbfgs',
                                                       C=np.infty))
     pipe_logit_ovr.fit(x_train,y_train)

Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
```

We obtain the prediction on test set with the train classifier. And here we have printed the predictions for first five examples. And you can see that we get a single prediction for each example in the test set, which is an image. Next, we obtain a confusion matrix based on the prediction on the test set.

And you can see that in this confusion matrix, there is some confusion between label 2 and label 8, then label 5 and label 8, label 5 and label 3. So, there are some of these confusions that you can see from the confusion matrix. There is also a confusion between label 9 and label 4. And let us obtain a classification report for this multiclass classifier.

So, you can see the precision recall and f1 number for every class. And for most of the classes, the numbers are more than 90% precision recall as well as f1-scores, except for Class 8 which has got f1-score of 0.87 and for Class 5 we have 0. 87 of f1-score. The recall is lower than the other classes.

(Refer Slide Time: 04:34)



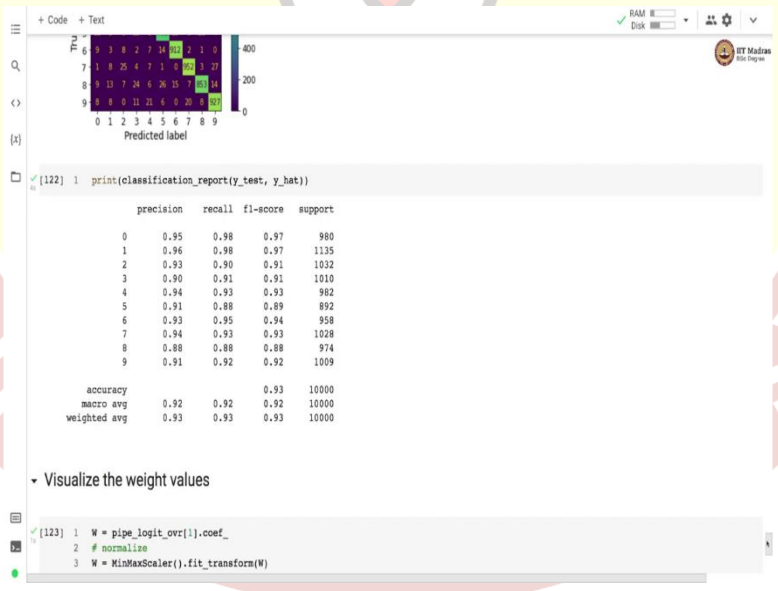
Multi-class LogisticRegression using solvers

```
[87]
pipe_logit_ovr = make_pipeline(MinMaxScaler(),
                               LogisticRegression(random_state=1729,
                                                  solver='lbfgs',
                                                  C=np.infty))
pipe_logit_ovr.fit(x_train,y_train)
```

```
Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                ('logisticregression',
                 LogisticRegression(C=inf, random_state=1729))])
```

```
[121]
y_hat = pipe_logit_ovr.predict(x_test)
cm_display = ConfusionMatrixDisplay.from_predictions(y_test, y_hat,
                                                     values_format='.5g')
plt.show()
```

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	960	0	0	3	1	7	4	4	1	0
1	0	1111	5	1	0	2	3	2	11	0
2	5	7	927	15	9	4	15	6	40	4
3	4	1	18	918	1	22	4	11	22	9
4	1	2	7	3	912	0	10	8	7	32
5	11	2	3	35	8	782	13	6	28	4
6	9	3	8	2	7	14	912	2	1	0
7	1	8	25	4	7	1	0	952	3	27
8	9	13	7	24	6	26	15	7	853	14
9	8	8	0	11	21	6	0	20	8	927



```
[122]
print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       980
           1       0.96      0.98      0.97      1135
           2       0.93      0.90      0.91      1032
           3       0.90      0.91      0.91      1010
           4       0.94      0.93      0.93       982
           5       0.91      0.88      0.89       892
           6       0.93      0.95      0.94       958
           7       0.94      0.93      0.93      1028
           8       0.88      0.88      0.88       974
           9       0.91      0.92      0.92      1009

    accuracy                           0.93     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.93      0.93      0.93     10000
```

Visualize the weight values

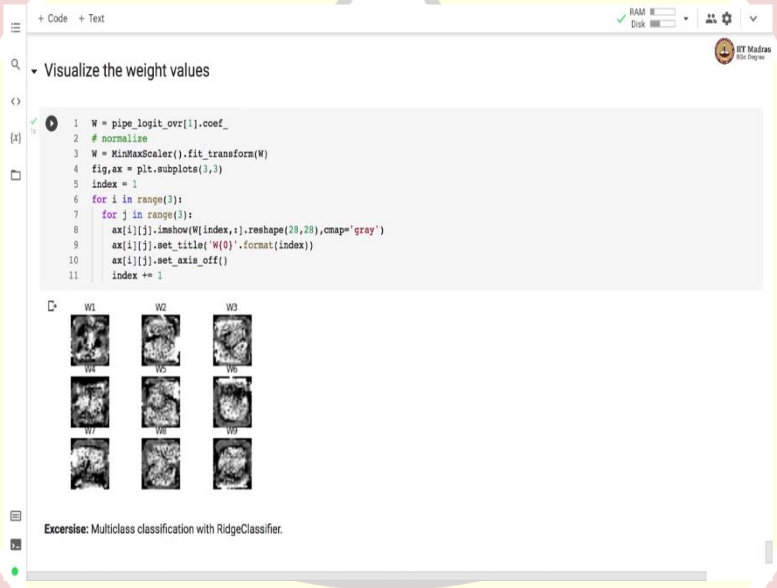
```
[123]
W = pipe_logit_ovr[1].coef_
# normalize
W = MinMaxScaler().fit_transform(W)
```

We can also set up a multiclass classifier with logistic regression with lbfgs solver and here we have again trained the logistic regression without regularization by setting the value of C to np.infty. After training the classifier we obtain the predictions on the test set and also plot the confusion matrix.

Observe that there are similar confusions that we discussed in the earlier confusion matrix. We obtained a classification report on the predictions on the test set. And you can see that the precision recall and f1-score are comparable to the classification report that we obtained earlier. Here, the numbers for class 8 and class 5 are better than what we obtained with SGDclassifier and overall f1-score for most of the classes is higher than what we obtain with SGDclassifier.

So, as an exercise what you can do is, you can try to introduce regularization, and train this multiclass classifier. Also train this multiclass classifier with cross-validation and perform hyper-parameter tuning. So, you have to follow exactly the same process that we followed for 0 detector and then compare the performance for different classifiers.

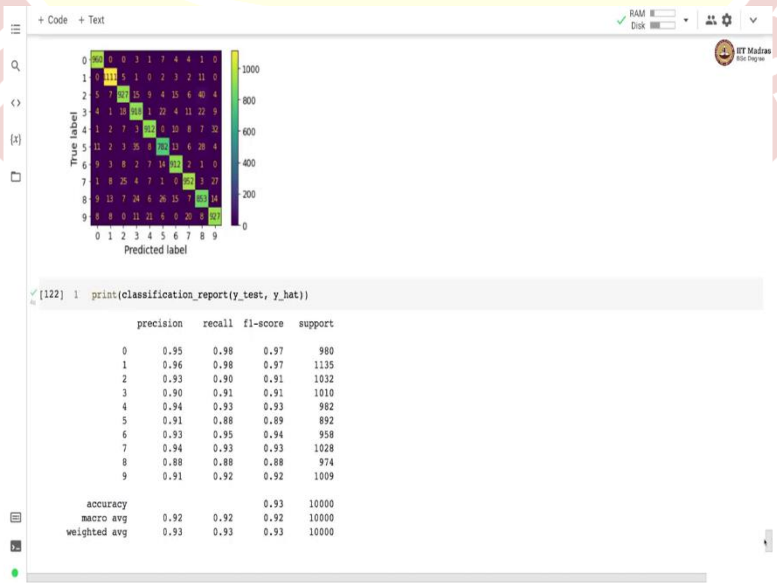
(Refer Slide Time: 06:09)



Visualize the weight values

```python
W = pipe_logit_ovr[1].coef_
# normalize
W = MinMaxScaler().fit_transform(W)
fig,ax = plt.subplots(3,3)
index = 1
for i in range(3):
  for j in range(3):
    ax[i][j].imshow(W[index,:].reshape(28,28),cmap='gray')
    ax[i][j].set_title('W{0}'.format(index))
    ax[i][j].set_axis_off()
    index += 1
```

Excersise: Multiclass classification with RidgeClassifier.



	0	1	2	3	4	5	6	7	8	9
0	960	0	0	3	1	7	4	4	1	0
1	0	1111	5	1	0	2	3	2	11	0
2	5	7	927	15	9	4	15	6	40	4
3	4	1	18	918	1	22	4	11	22	9
4	1	2	7	3	912	0	10	8	7	32
5	11	2	3	35	8	782	13	6	28	4
6	9	3	8	2	7	14	912	2	1	0
7	1	8	25	4	7	1	0	952	3	27
8	9	13	7	24	6	26	15	7	853	14
9	8	8	0	11	21	6	0	20	8	927

```
[122] print(classification_report(y_test, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       980
           1       0.96      0.98      0.97      1135
           2       0.93      0.90      0.91      1032
           3       0.90      0.91      0.91      1010
           4       0.94      0.93      0.93       982
           5       0.91      0.88      0.89       892
           6       0.93      0.95      0.94       958
           7       0.94      0.93      0.93      1028
           8       0.88      0.88      0.88       974
           9       0.91      0.92      0.92      1009

    accuracy                           0.93     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.93      0.93      0.93     10000
```

So, here we are visualized weight values for weight vectors W1, W2, W3 all the way up to W9. And you can see that these different weight vectors have learned different set of patterns as is evident from the weights that are assigned to each pixel. In this video, we trained a couple of multiclass classification models with SGDclassifier, and logistic regression classifiers. Now, you know how to set up the multiclass classification problems and train them using a SGDclassifier and logistic regression classifier.