

IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Doctor Ashish Tendulkar
Indian Institute of Technology Madras
Demonstration: Zero detector with Ridge Classifier

(Refer Slide Time: 00:10)



```
1 # initialize new variable names with all -1
2 y_train_0 = -1*np.ones((len(y_train)))
3 y_test_0 = -1*np.ones((len(y_test)))
4
5 # find indices of digit 0 image
6 indx_0 = np.where(y_train == '0')
7
8 # use those indices to modify y_train_0,y_test_0
9 y_train_0[indx_0] = 1
10 indx_0 = np.where(y_test == '0')
11 y_test_0[indx_0] = 1
```

• First take a look into the parameters of the class

```
RidgeClassifier(alpha=1.0, *, fit_intercept=True, normalize='deprecated', copy_X=True,
max_iter=None, tol=0.001, class_weight=None, solver='auto', positive=False, random_state=None)
```

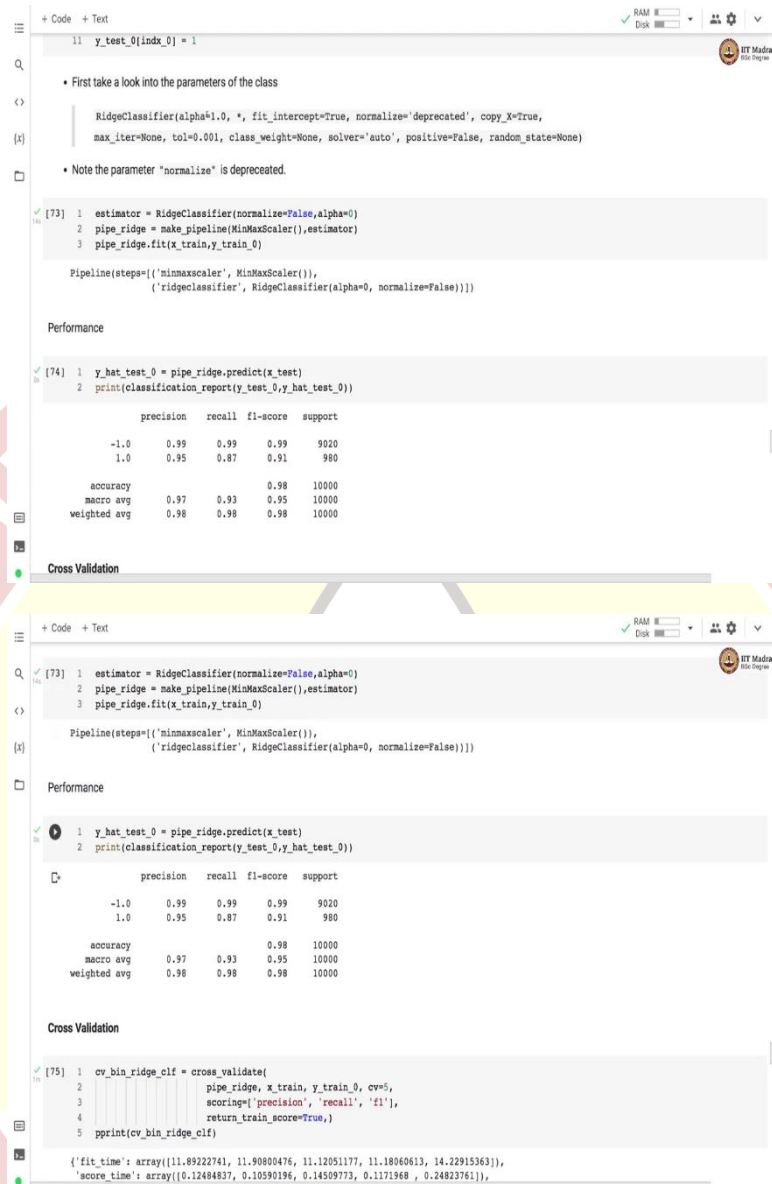
• Note the parameter "normalize" is deprecated.

```
[ ] 1 estimator = RidgeClassifier(normalize=False,alpha=0)
2 pipe_ridge = make_pipeline(MinMaxScaler(),estimator)
3 pipe_ridge.fit(x_train,y_train_0)
```

Namaste! Welcome to the next video of Machine Learning Practice course. In this video, we will use Ridge Classifier to implement the zero-detector model. The ridge classifier cast the classification problem, as a least squares classification problem, and finds the optimal weights using matrix decomposition techniques like SVD. To train a ridge classifier, the label should be +1 and -1.

The ridge classifier by default uses L2 regularization, but we will be setting the value of regularization rate alpha to 0 and train it without any regularization. So, first we construct the labels. So, for the negative examples, we assign labels of -1, and for positive examples, we assign a label of +1. Here the positive example corresponds to images of digit 0, and negative examples are images of rest of the digits.

(Refer Slide Time: 01:38)



The image displays two screenshots of a Jupyter Notebook interface. The top screenshot shows the initial setup of a Ridge Classifier and a pipeline. The bottom screenshot shows the performance metrics and cross-validation results for the same pipeline.

Top Screenshot:

```
11 y_test_0[indx_0] = 1
```

• First take a look into the parameters of the class

```
RidgeClassifier(alpha=1.0, *, fit_intercept=True, normalize='deprecated', copy_X=True,
max_iter=None, tol=0.001, class_weight=None, solver='auto', positive=False, random_state=None)
```

• Note the parameter "normalize" is deprecated.

```
[73] 1 estimator = RidgeClassifier(normalize=False, alpha=0)
     2 pipe_ridge = make_pipeline(MinMaxScaler(), estimator)
     3 pipe_ridge.fit(x_train, y_train_0)
```

Pipeline(steps=[('minmaxscaler', MinMaxScaler()), ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))])

Performance

```
[74] 1 y_hat_test_0 = pipe_ridge.predict(x_test)
     2 print(classification_report(y_test_0, y_hat_test_0))
```

	precision	recall	f1-score	support
-1.0	0.99	0.99	0.99	9020
1.0	0.95	0.87	0.91	980
accuracy			0.98	10000
macro avg	0.97	0.93	0.95	10000
weighted avg	0.98	0.98	0.98	10000

Cross Validation

Bottom Screenshot:

```
[73] 1 estimator = RidgeClassifier(normalize=False, alpha=0)
     2 pipe_ridge = make_pipeline(MinMaxScaler(), estimator)
     3 pipe_ridge.fit(x_train, y_train_0)
```

Pipeline(steps=[('minmaxscaler', MinMaxScaler()), ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))])

Performance

```
1 y_hat_test_0 = pipe_ridge.predict(x_test)
2 print(classification_report(y_test_0, y_hat_test_0))
```

	precision	recall	f1-score	support
-1.0	0.99	0.99	0.99	9020
1.0	0.95	0.87	0.91	980
accuracy			0.98	10000
macro avg	0.97	0.93	0.95	10000
weighted avg	0.98	0.98	0.98	10000

Cross Validation

```
[75] 1 cv_bin_ridge_clf = cross_validate(
     2     pipe_ridge, x_train, y_train_0, cv=5,
     3     scoring=['precision', 'recall', 'f1'],
     4     return_train_score=True,
     5 )
pprint(cv_bin_ridge_clf)
```

```
{ 'fit_time': array([11.89222741, 11.90800476, 11.12051177, 11.18060613, 14.22915363]),
  'score_time': array([0.12484837, 0.10590196, 0.14509773, 0.1171948 , 0.24823761]),
```

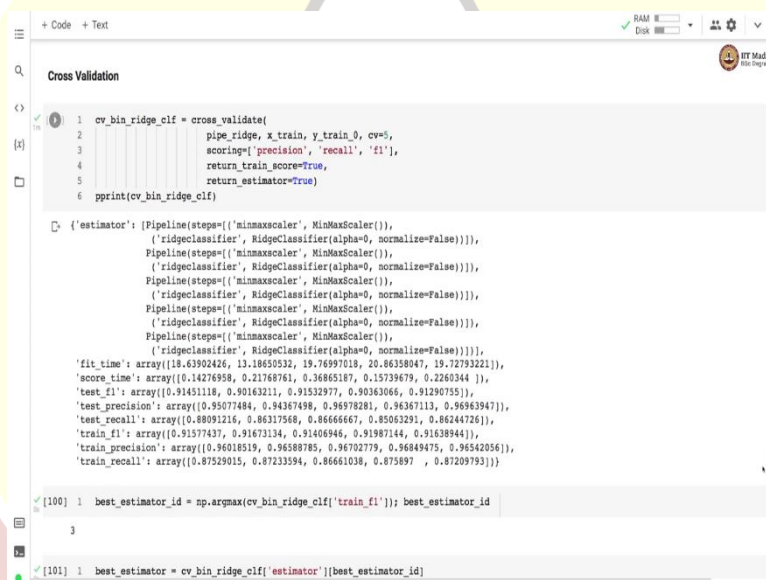
Let us look at the ridge classifier, as implemented in a sklearn and its parameters. So alpha is the regularization rate, it is set to 1 by default, fit intercept is true by default. Then solver is auto, so ridge classifier automatically selects the solver based on the data, and note that, a normalized parameter is now deprecated.

So, here we define a pipeline object with two stages. The first stage is, the MinMaxScaler that is used for the processing or scaling of each pixel value between 0 and 1. And a second step is the ridge classifier that implements the least square classification method. We train this model by calling of fit on the pipeline object by passing the feature matrix and label vector.

So, after calling the fit, the model is trained, and we use the trained model to make predictions on the test set. And we evaluate the performance of the classifier on the test set with classification report API. So, you first make prediction on the test set and we get the predicted labels. We compare these predicted labels with the actual labels and obtain a classification report that contains measures like precision, recall and f1-score.

So, if you compare for the class 1, which corresponds to the digit 0 has got lower recall than the methods that you use. So far, the precision is comparable it is 0.95 it is slightly lower than the earlier methods, but recall is quite low than the earlier methods, recall is 0.87 in this case, and f1-score is 0.91. On the negative example side precision recall and f1-scores are comparable to the earlier methods.

(Refer Slide Time: 03:56)



```
+ Code + Text
Cross Validation

1 cv_bin_ridge_clf = cross_validate(
2     pipe_ridge, x_train, y_train_0, cv=5,
3     scoring=['precision', 'recall', 'f1'],
4     return_train_score=True,
5     return_estimator=True)
6 pprint(cv_bin_ridge_clf)

{'estimator': (Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))])),
 'fit_time': array([18.63902426, 13.18650532, 19.76997018, 20.86358047, 19.72793221]),
 'score_time': array([0.14276958, 0.21768761, 0.36865187, 0.15739679, 0.2260344 ]),
 'test_f1': array([0.91451118, 0.90163211, 0.91532977, 0.90363066, 0.91290755]),
 'test_precision': array([0.95077484, 0.94367498, 0.96978281, 0.96367113, 0.96963947]),
 'test_recall': array([0.88091216, 0.86317568, 0.86666667, 0.85063291, 0.86244726]),
 'train_f1': array([0.91577437, 0.91673134, 0.91406946, 0.91987144, 0.91638944]),
 'train_precision': array([0.96018519, 0.96588785, 0.96702779, 0.96849475, 0.96542056]),
 'train_recall': array([0.87529015, 0.87233594, 0.86661038, 0.875897 , 0.87209793])

[100] 1 best_estimator_id = np.argmax(cv_bin_ridge_clf['train_f1']); best_estimator_id
3

[101] 1 best_estimator = cv_bin_ridge_clf['estimator'][best_estimator_id]
```

The first screenshot shows the initial code for training a ridge classifier with cross-validation. The code defines a pipeline with a MinMaxScaler and a RidgeClassifier, then uses `cross_validate` to evaluate it. The output shows the best estimator ID (3) and the best estimator object.

```

3 scoring='precision', 'recall', 'f1',
4 return_train_score=True,
5 return_estimator=True
6 pprint(cv_bin_ridge_clf)

{'estimator': [Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))]),
               Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                                ('ridgeclassifier', RidgeClassifier(alpha=0, normalize=False))])],
 'fit_time': array([18.63902426, 13.18650532, 19.76997018, 20.86358047, 19.72793221]),
 'score_time': array([0.14276958, 0.21768761, 0.36865187, 0.15739679, 0.2260344 ]),
 'test_f1': array([0.91451118, 0.90163211, 0.91532977, 0.90363066, 0.91290755]),
 'test_precision': array([0.95077484, 0.94367498, 0.96978281, 0.96367113, 0.96963947]),
 'test_recall': array([0.88091216, 0.86317568, 0.86666667, 0.85063291, 0.86244726]),
 'train_f1': array([0.91577437, 0.91673134, 0.91406946, 0.91987144, 0.91638944]),
 'train_precision': array([0.88091216, 0.86317568, 0.86666667, 0.85063291, 0.86244726]),
 'train_recall': array([0.91577437, 0.91673134, 0.91406946, 0.91987144, 0.91638944])}

[100] 1 best_estimator_id = np.argmax(cv_bin_ridge_clf['train_f1']); best_estimator_id
3

[101] 1 best_estimator = cv_bin_ridge_clf['estimator'][best_estimator_id]

Let's evaluate the performance of the best classifier on the test set:

[114] 1 y_hat_test_0 = best_estimator.predict(x_test)
      2 print(classification_report(y_test, y_hat_test_0))

```

The second screenshot shows the output of the `classification_report` function, which provides a detailed summary of the classifier's performance on the test set.

```

[100] 1 best_estimator_id = np.argmax(cv_bin_ridge_clf['train_f1']); best_estimator_id
3

[101] 1 best_estimator = cv_bin_ridge_clf['estimator'][best_estimator_id]

Let's evaluate the performance of the best classifier on the test set:

[114] 1 y_hat_test_0 = best_estimator.predict(x_test)
      2 print(classification_report(y_test, y_hat_test_0))

              precision    recall  f1-score   support

    -1.0         0.99         0.99         0.99         9020
     1.0         0.95         0.88         0.91          980

 accuracy          0.97         0.94         0.98        10000
  macro avg          0.97         0.94         0.95        10000
 weighted avg          0.98         0.98         0.98        10000

Further exploration
Let's see what these classifiers learnt about the digit 0

```

In order to train the ridge classifier in a robust manner, we use cross-validation-based training. So, we use `cross_validate` API, and pass the pipeline object as an estimator. We also pass the feature vector and the labels. We set the number of cross-validation folds to 5 and use precision recall and f1 as scoring functions.

We request the API to return the training score and estimators by setting those respective flags to true. So, after the model is trained, what you see is, you see five different estimators that were trained on different folds, you have the time it took to fit these estimators on different folds, and the scoring time and f1-scored precision and recall on test and training sets.

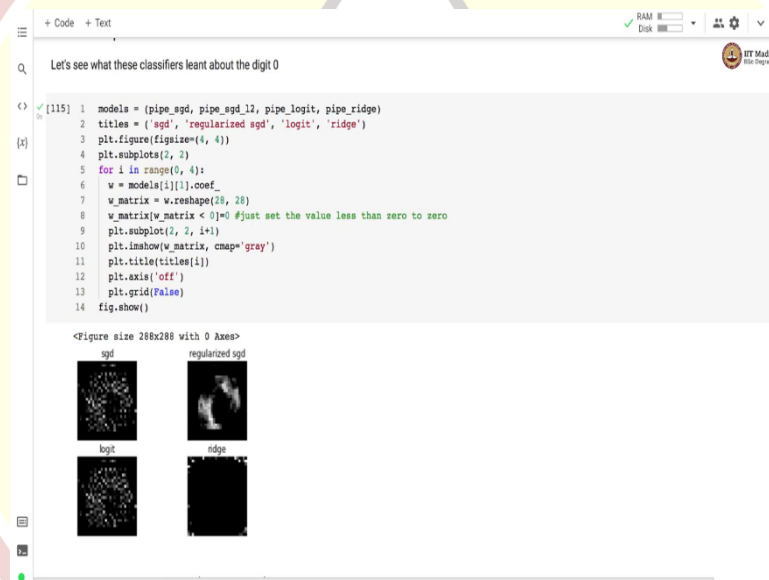
So, among these different estimators, we can find the best estimator that has the best score, which is best f1-score on the training set. So, we use `np.argmax` on the train f1-score and find

out the index of the best estimator. And you can see that the best estimator is at index 3, because we start from index 0. This is the best estimator that we have, which is the fourth estimator on the list.

We obtain the best estimator and use it for predicting labels on the test set. We use the predicted label to evaluate the performance of the classifier with classification report API. We obtained precision recall and f1-score. You can compare this classification report with the earlier classification report where we trained ridge classifier without cross-validation.

The recall has improved by 1% point, earlier the recall was 0.87, now we have recall a 0.88. The precision has remained unchanged.

(Refer Slide Time: 06:24)



Now let us see, what these different classifiers have learned about digits 0. So, we have different classifiers. The first couple of them based on the stochastic gradient descent, we have one with logistic regression, and the one that we that we implemented in this video is based on the ridge classifier.

What we have done is, we obtained the weight vector for different modules, and we reshape the weight vector into 28×28 grid. And if vector, if the if the weight is less than 0, then we set it to 0. And you can see that when you plot the weight matrix for these four classifiers, you can see different patterns that were learned for weight.

So, these are the pixels that have got positive values for a SGD classifier for regularized SGD. There are certain pixels in this region which are present here, which had got lower weight. Age are driven down to 0 and these pixels have gotten the higher values. The classifier obtained based on logistic regression is very similar to what we obtained with SGD.

The classifier obtained with ridge is quite different from the other three methods, and this mainly happens probably because many of these pixels were having values less than 0. And that is why we are not showing that we have set those weight values to 0, and that is why you see a black patch in between where other classifiers have put certain weights on those pixels.

Surprisingly, the ridge classifier put some weight on the pixels at the boundary. So, in this video we used another classification technique, which is ridge classifier, which uses least square classification technique for training the model for detecting the digit 0 in the image. We also visualized a weight matrix learned by different classifiers.

