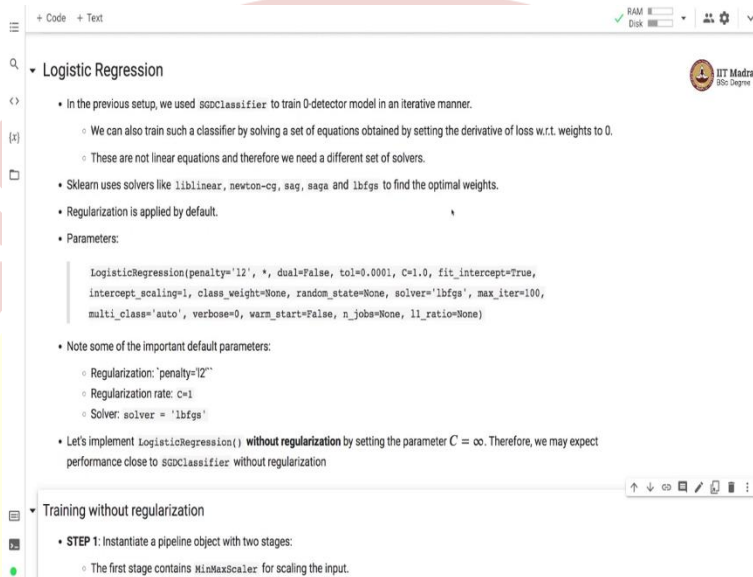


IIT Madras

ONLINE DEGREE

Machine Learning Techniques
Professor Doctor Ashish Tendulkar
Indian Institute of Technology Madras
Demonstration: MNIST Digits Classification using Logistic Regression

(Refer Slide Time: 00:09)

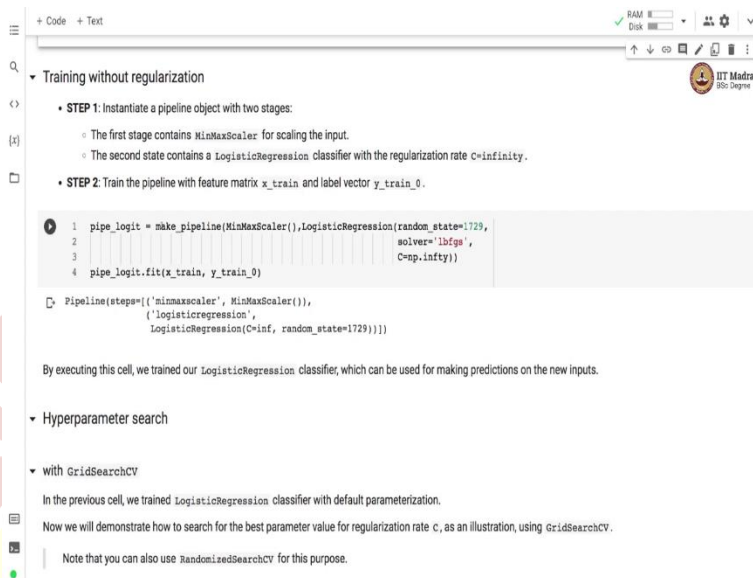


Namaste! Welcome to the next video of Machine Learning Practice course. In this video, we will build a 0-detector model with logistic regression. In the previous setup, we use SGDClassifier to train the 0-detector model in an iterative manner. We can also train such a classifier by solving a set of equations obtained by setting the derivative of loss with respect to weights to 0.

So, the resulting equations are not linear equations, and hence, we need a different set of solvers for solving these equations. So, sklearn uses solvers like liblinear, newton-cg, sag, saga and lbfgs to obtain optimal weights in case of logistic regression. Regularization is applied by default. This is the default parameterization of the logistic regression classifier in sklearn. Note some of the important default parameter.

The regularization that is used is l2 regularization, with regularization rate c set to 1, and the default solver is lbfgs. Let us implement logistic regression without regularization by setting the parameter c to infinity. So, by setting c to infinity, we expect the performance of the logistic regression model without regularization to be close to the SGDClassifier without regularization.

(Refer Slide Time: 01:50)



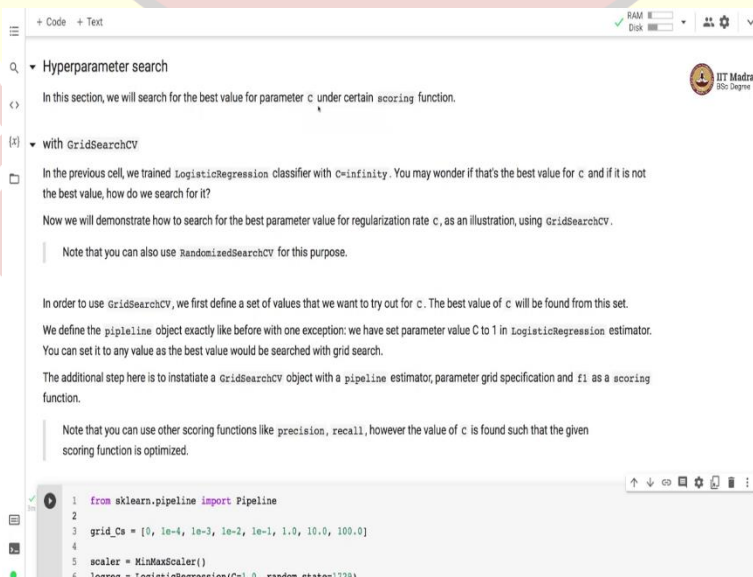
The image shows a Jupyter Notebook interface with a code cell. The code defines a pipeline with two stages: `MinMaxScaler` and `LogisticRegression`. The `LogisticRegression` is initialized with `random_state=1729`, `solver='lbfgs'`, and `C=np.inf`. The pipeline is then trained using `pipe_logit.fit(x_train, y_train_0)`. Below the code, there is a text box explaining that the pipeline consists of a `MinMaxScaler` and a `LogisticRegression` with `C=inf` and `random_state=1729`. It also mentions that the model can be used for making predictions on new inputs.

```
1 pipe_logit = make_pipeline(MinMaxScaler(), LogisticRegression(random_state=1729,
2 solver='lbfgs',
3 C=np.inf))
4 pipe_logit.fit(x_train, y_train_0)
```

By executing this cell, we trained our `LogisticRegression` classifier, which can be used for making predictions on the new inputs.

So, in order to train a logistic regression model without regularization we first instantiate a pipeline object with two stages. The first stage contains `MinMaxScaler` for scaling the input. And the second stage contains logistic regression classifier with regularization rate C equal to infinity. In the second step, we train the pipeline with feature matrix x train and label vector y train 0 as inputs. By executing this cell, we train our logistic regression classifier which can now be used for making predictions on the new inputs.

(Refer Slide Time: 02:39)



The image shows a Jupyter Notebook interface with a code cell. The code imports `Pipeline` from `sklearn.pipeline`, defines a `grid_cs` list of regularization parameters, creates a `MinMaxScaler` and a `LogisticRegression` with `C=1.0` and `random_state=1729`. The code is partially visible, showing the import and the definition of the pipeline components.

```
1 from sklearn.pipeline import Pipeline
2
3 grid_cs = [0, 1e-4, 1e-3, 1e-2, 1e-1, 1.0, 10.0, 100.0]
4
5 scaler = MinMaxScaler()
6 logreg = LogisticRegression(C=1.0, random_state=1729)
```

```
+ Code + Text
1 from sklearn.pipeline import Pipeline
2
3 grid_Cs = [0, 1e-4, 1e-3, 1e-2, 1e-1, 1.0, 10.0, 100.0]
4
5 scaler = MinMaxScaler()
6 logreg = LogisticRegression(C=1.0, random_state=1729)
7
8 pipe = Pipeline(steps=[('scaler', scaler),
9                        ('logistic', logreg)])
10
11 pipe_logit_cv = GridSearchCV(
12     pipe,
13     param_grid={'logistic_C': grid_Cs},
14     scoring='f1')
15 pipe_logit_cv.fit(x_train, y_train_0)

GridSearchCV(estimator=Pipeline(steps=[('scaler', MinMaxScaler()),
                                       ('logistic',
                                        LogisticRegression(random_state=1729))]),
              param_grid={'logistic_C': [0, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0,
                                         100.0]}),
              scoring='f1')
```

The GridSearchCV finds the best value of c and refits the estimator by default on the entire training set. This gives us the logistic regression classifier with best value of c.

We can check the value of the best parameter by accessing the best_params_ member variable of the GridSearchCV object.

```
[10] 1 pipe_logit_cv.best_params_
{'logistic_C': 0.1}
```

```
+ Code + Text
GridSearchCV(estimator=Pipeline(steps=[('scaler', MinMaxScaler()),
                                       ('logistic',
                                        LogisticRegression(random_state=1729))]),
              param_grid={'logistic_C': [0, 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0,
                                         100.0]}),
              scoring='f1')
```

The GridSearchCV finds the best value of c and refits the estimator by default on the entire training set. This gives us the logistic regression classifier with best value of c.

We can check the value of the best parameter by accessing the best_params_ member variable of the GridSearchCV object.

```
[10] 1 pipe_logit_cv.best_params_
{'logistic_C': 0.1}
```

and the best score can be found in best_score_ member variable and can be obtained as follows:

```
[11] 1 pipe_logit_cv.best_score_
0.9579654953103655
```

The best estimator can be accessed with best_estimator_ member variable.

```
[12] 1 pipe_logit_cv.best_estimator_
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('logistic', LogisticRegression(C=0.1, random_state=1729))])
```

With LogisticRegressionCV

In this section, we will search for the best value of parameter c under certain scoring function. In the previous cell, we trained logistic regression classifier with the value of c set to infinity. You may wonder if that's the best value for c , and if it is not the best value, how do search for the best value. Now, we will demonstrate how to search for the best parameter value for regularization rate c using GridSearchCV. And this particular regularization rate c is just used as an illustration. You can also launch parameter search for something like l1 ratio.

Note that you can also make use of RandomizedSearchCV for finding the best parameter value. However, we are making use of GridSearchCV as an illustration. In order to use GridSearchCV, we first define a set of values for c . The best value of c will be found from this set. We define a

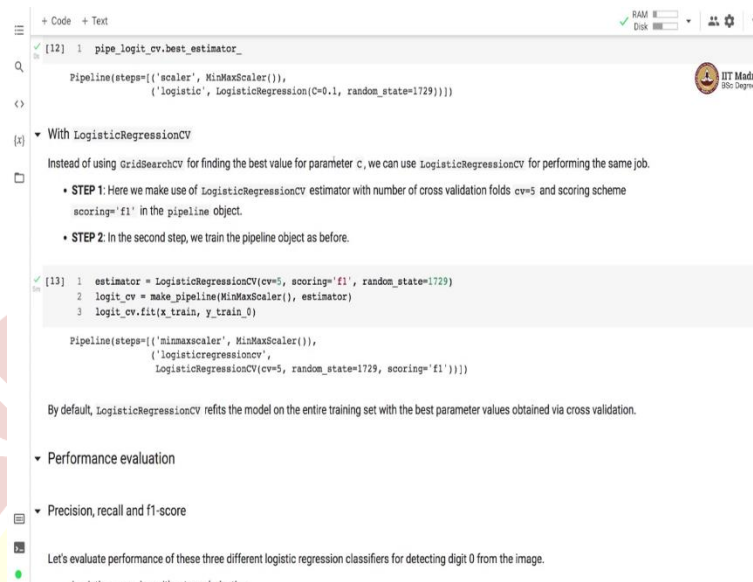
pipeline object just like before with one small exception. We set the value of parameter c equal to 1 in the logistic regression estimator. You can set this value to any other value and the best value of c would be searched with grid search.

There is an additional step to instantiate a GridSearchCV object with pipeline estimator and parameter grid specification. We also have to set up, we also have to specify the scoring function, and in this case, we use f1 as a scoring function. So, instead of using f1 as a scoring function, you also have a choice of using scoring functions like precision and recall. However, note that the value of the c that we have to find will be optimized for the scoring function that you specify.

So, the GridSearchCV finds the best value of c and refits the estimator by default on the entire training set. This gives us a logistic regression classifier with best value of c . We can check out the value of the best parameter by accessing the best _params _member variable of GridSearchCV object. So, we can access it by specifying this member variable of the GridSearchCV object and you can see that the parameter value c , the best value of the parameter c is 0.1.

The best score can be found in best _score _member variable of the GridSearchCV object. And here we obtained the best score of 0.9579. That's a best value of the f1 score. The best estimator can be accessed with best _estimator _member variable of the GridSearchCV object.

(Refer Slide Time: 06:15)



The screenshot shows a Jupyter Notebook interface with the following content:

```
[12] 1 pipe_logit_cv.best_estimator_

Pipeline(steps=[('scaler', MinMaxScaler()),
                 ('logistic', LogisticRegression(C=0.1, random_state=1729))])
```

▼ With LogisticRegressionCV

Instead of using GridSearchCV for finding the best value for parameter *c*, we can use LogisticRegressionCV for performing the same job.

- STEP 1: Here we make use of LogisticRegressionCV estimator with number of cross validation folds *cv=5* and scoring scheme *scoring='f1'* in the pipeline object.
- STEP 2: In the second step, we train the pipeline object as before.

```
[13] 1 estimator = LogisticRegressionCV(cv=5, scoring='f1', random_state=1729)
      2 logit_cv = make_pipeline(MinMaxScaler(), estimator)
      3 logit_cv.fit(x_train, y_train_0)

Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                 ('logisticregressioncv', LogisticRegressionCV(cv=5, random_state=1729, scoring='f1'))])
```

By default, LogisticRegressionCV refits the model on the entire training set with the best parameter values obtained via cross validation.

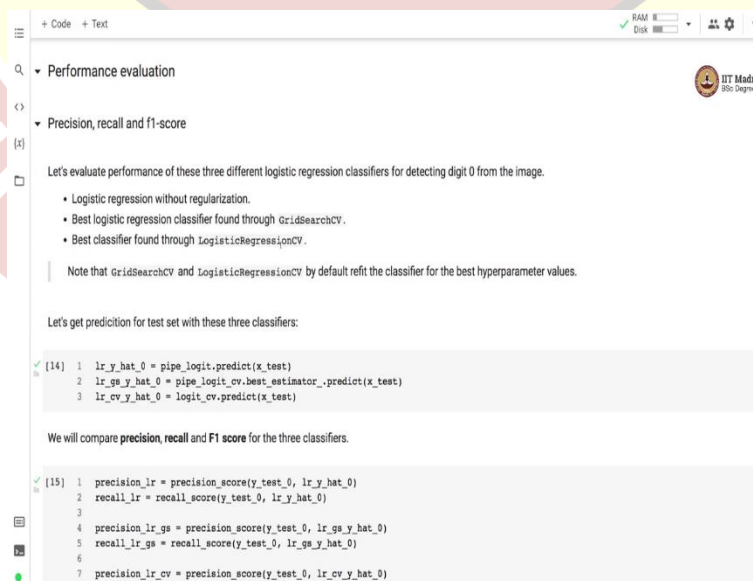
▼ Performance evaluation

▼ Precision, recall and f1-score

Let's evaluate performance of these three different logistic regression classifiers for detecting digit 0 from the image.

Instead of using GridSearchCV for finding the best value of parameter *c*, we can use LogisticRegressionCV for performing the same job. Here we make use of LogisticRegressionCV estimator with number of cross-validation fold CV set to 5 and scoring scheme set to f1. In the second step, we train the pipeline object as before. So, here, we define a pipeline object with estimator equal to LogisticRegressionCV. By default, LogisticRegressionCV refits the model on the entire training set with the best parameter values obtained via cross-validation.

(Refer Slide Time: 07:05)



The screenshot shows a Jupyter Notebook interface with the following content:

▼ Performance evaluation

▼ Precision, recall and f1-score

Let's evaluate performance of these three different logistic regression classifiers for detecting digit 0 from the image.

- Logistic regression without regularization.
- Best logistic regression classifier found through GridSearchCV.
- Best classifier found through LogisticRegressionCV.

Note that GridSearchCV and LogisticRegressionCV by default refit the classifier for the best hyperparameter values.

Let's get prediction for test set with these three classifiers:

```
[14] 1 lr_y_hat_0 = pipe_logit.predict(x_test)
      2 lr_gs_y_hat_0 = pipe_logit_cv.best_estimator_.predict(x_test)
      3 lr_cv_y_hat_0 = logit_cv.predict(x_test)
```

We will compare precision, recall and F1 score for the three classifiers.

```
[15] 1 precision_lr = precision_score(y_test_0, lr_y_hat_0)
      2 recall_lr = recall_score(y_test_0, lr_y_hat_0)
      3
      4 precision_lr_gs = precision_score(y_test_0, lr_gs_y_hat_0)
      5 recall_lr_gs = recall_score(y_test_0, lr_gs_y_hat_0)
      6
      7 precision_lr_cv = precision_score(y_test_0, lr_cv_y_hat_0)
```



```
+ Code + Text
RAM 8 GB
Disk 100 GB

[15] 1 precision_lr = precision_score(y_test_0, lr_y_hat_0)
2 recall_lr = recall_score(y_test_0, lr_y_hat_0)
3
4 precision_lr_gs = precision_score(y_test_0, lr_gs_y_hat_0)
5 recall_lr_gs = recall_score(y_test_0, lr_gs_y_hat_0)
6
7 precision_lr_cv = precision_score(y_test_0, lr_cv_y_hat_0)
8 recall_lr_cv = recall_score(y_test_0, lr_cv_y_hat_0)

1 print (f"LogReg: precision={precision_lr}, recall={recall_lr}")
2 print (f"GridSearch: precision={precision_lr_gs}, recall={recall_lr_gs}")
3 print (f"LogRegCV: precision={precision_lr_cv}, recall={recall_lr_cv}")

LogReg: precision=0.9515151515151515, recall=0.961224469759183
GridSearch: precision=0.9564336372847011, recall=0.963265306122449
LogRegCV: precision=0.9564370967741935, recall=0.9663265306122449

Note that all three classifiers have roughly the same performance as measured with precision and recall.

• The LogisticRegression classifier obtained through GridSearchCV has the highest precision - marginally higher than the other two classifiers.
• The LogisticRegression classifier obtained through LogisticRegressionCV has the highest recall - marginally higher than the other two classifiers.

Using PR curve

[17] 1 y_scores_lr = pipe_logit.decision_function(x_test)
2 precisions_lr, recalls_lr, thresholds_lr = precision_recall_curve(
3     y_test_0, y_scores_lr)
4
```

Let us evaluate performance of these three different logistic regression classifiers for detecting digit 0 from the image. So, we have three classifiers, logistic regression without regularization, best logistic regression classifier found through GridSearchCV, and best classifier found through LogisticRegressionCV. So, again, reminding you that GridSearchCV and LogisticRegressionCV, by default, refit the classifier for the best hyper-parameter values that define.

Let us get the predictions for test set with these three classifiers. So, we have three sets of prediction for test sets with logistic regression, without regularization, the one with GridSearchCV and the one that is found through LogisticRegressionCV. Then we calculate the precision and recall values for these three sets of predictions. And here we compare, we print the precision recall values for each of the classifier and we can compare them.

So, you can see that all the three classifiers roughly have the same performance as measured with precision and recall. The logistic regression classifier obtained through GridSearchCV has highest precision, whereas logistic regression classifier obtained through LogisticRegressionCV has got the highest recall value, and both the highest precision and highest recall are marginally higher than the remaining two classifiers.

(Refer Slide Time: 09:02)

```
+ Code + Text
Using PR-curve

[17] 1 y_scores_lr = pipe_logit.decision_function(x_test)
2 precisions_lr, recalls_lr, thresholds_lr = precision_recall_curve(
3     y_test_0, y_scores_lr)
4
5 y_scores_lr_gs = pipe_logit_cv.decision_function(x_test)
6 precisions_lr_gs, recalls_lr_gs, thresholds_lr_gs = precision_recall_curve(
7     y_test_0, y_scores_lr_gs)
8
9 y_scores_lr_cv = logit_cv.decision_function(x_test)
10 precisions_lr_cv, recalls_lr_cv, thresholds_lr_cv = precision_recall_curve(
11     y_test_0, y_scores_lr_cv)
```

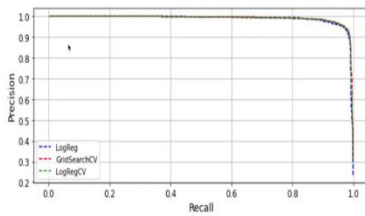
We have all the quantities for plotting the PR curve. Let's plot PR curves for all three classifiers:

```
[18] 1 plt.figure(figsize=(10,4))
2 plt.plot(recalls_lr[1:], precisions_lr[1:], 'b--', label='LogReg')
3 plt.plot(recalls_lr_gs[1:], precisions_lr_gs[1:], 'r--', label='GridSearchCV')
4 plt.plot(recalls_lr_cv[1:], precisions_lr_cv[1:], 'g--', label='LogRegCV')
5
6 plt.ylabel('Precision')
7 plt.xlabel('Recall')
8 plt.grid(True)
9 plt.legend(loc='lower left')
10 plt.show()
```



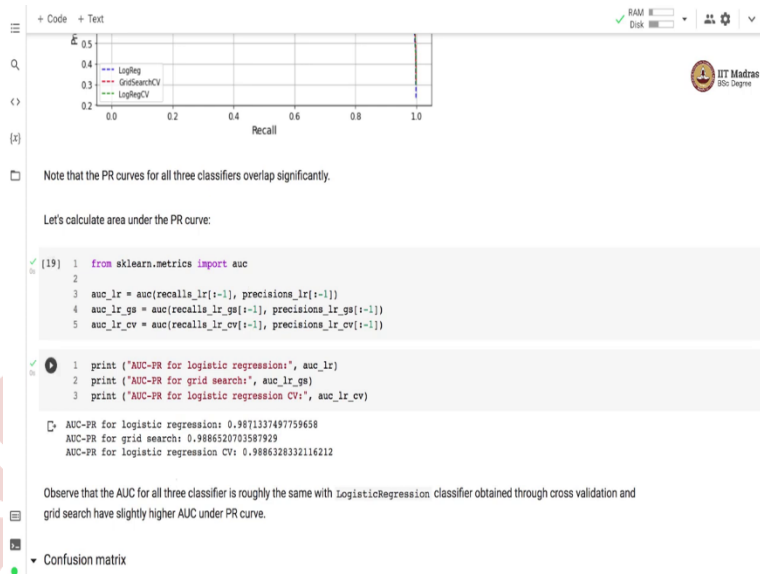
We have all the quantities for plotting the PR curve. Let's plot PR curves for all three classifiers:

```
1 plt.figure(figsize=(10,4))
2 plt.plot(recalls_lr[1:], precisions_lr[1:], 'b--', label='LogReg')
3 plt.plot(recalls_lr_gs[1:], precisions_lr_gs[1:], 'r--', label='GridSearchCV')
4 plt.plot(recalls_lr_cv[1:], precisions_lr_cv[1:], 'g--', label='LogRegCV')
5
6 plt.ylabel('Precision')
7 plt.xlabel('Recall')
8 plt.grid(True)
9 plt.legend(loc='lower left')
10 plt.show()
```



Note that the PR curves for all three classifiers overlap significantly.

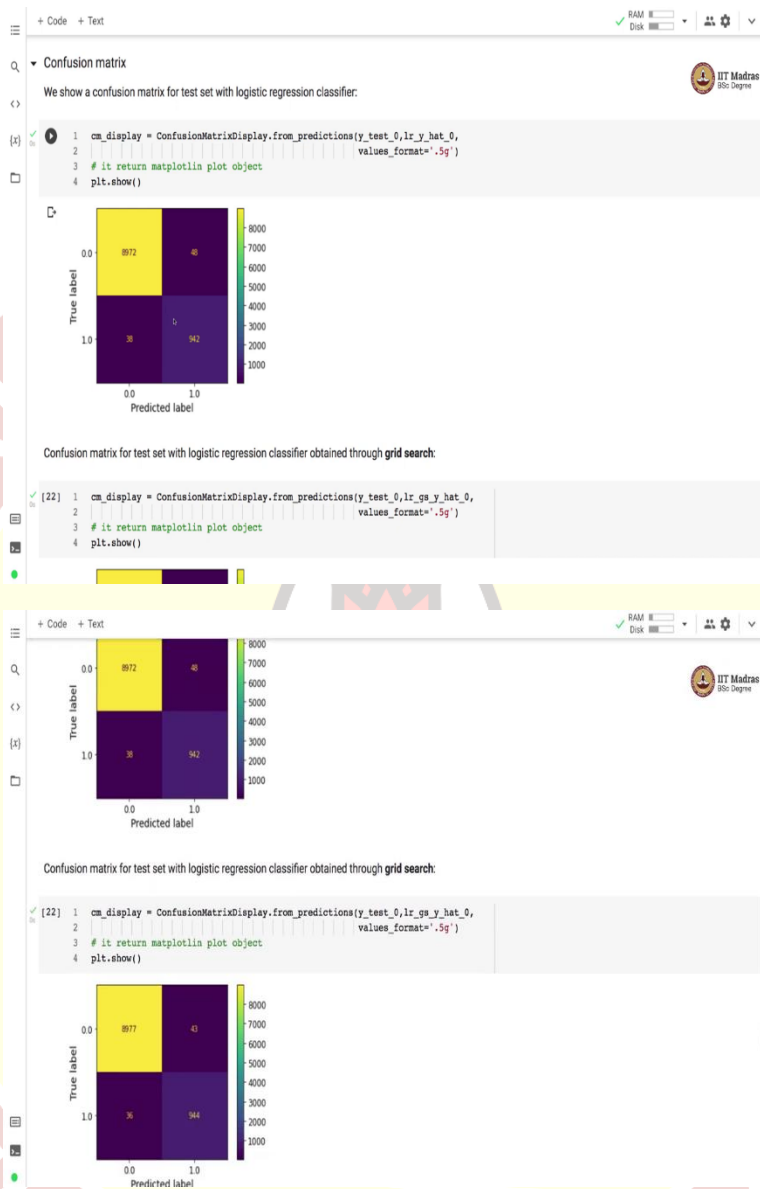
Let's calculate area under the PR curve:



Instead of comparing these three classifiers on single value of precision and recall, we will use PR curves to study their behavior across different values of precision and recall. So, here we use decision functions to obtain the probability for each example of belonging to class 1. And at different probability thresholds, we get the values of precision and recall through `precision_recall_curve` function.

So, we perform the same procedure for all the three classifiers and then we print the, we use these PR values for plotting the PR curve. So, here we plotted PR curves for all the three classifiers in the same plot. You can observe that PR curves for all the three classifiers overlap significantly. Let us calculate the area under the curve of these PR curves. So, we can make use of AUC function from sklearn metric for calculating the area under the PR curve. And observe that the AUC for all three classifiers is roughly the same.

(Refer Slide Time: 10:36)





We can also look at the confusion matrix for the test set with the three classifiers. So, this is the confusion matrix for logistic regression classifier without any regularization and mostly the default parameterization. So, you can see that there are 48 and 38 examples that are misclassified from class 0 and class 1. So, 38 examples are such that the true label was 1, but they are predicted to be 0 and 48 examples from class 0 are predicted in the class 1.

Now, we can compare this confusion matrix with the one obtained through grid search. And you can see that the misclassification has gone down marginally in case of the grid search based logistic regression classifier. And you can see that the number of examples from class 1 that are correctly classified has gone up by 2.

Let us compare this classifier, these two classifiers with the confusion matrix of logistic regression classifier obtained through cross-validation. And you can see that the number of correctly classified examples from class 1 have gone up further in this case. So, here you can see that the number of misclassified example from class 1 have also gone down.

So, in this video, we use logistic regression classifier for 0-detector. We train three logistic regression classifiers, one without regularization, and then the remaining two logistic regression classifiers that perform cross-validation for finding the best parameter.