

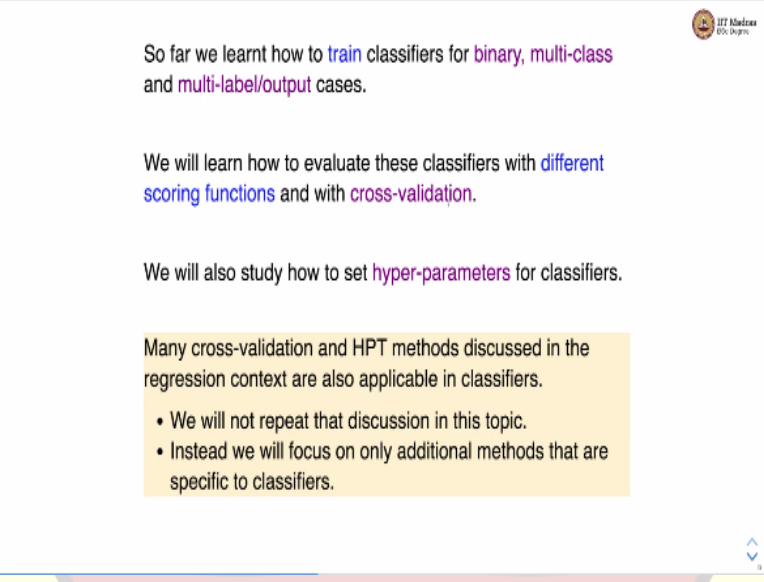
IIT Madras

ONLINE DEGREE

Machine Learning Practice
B. Sc in Programming and Data Science
Diploma Level
Indian Institute of Technology, Madras
Evaluating Classifiers

Namaste welcome to the next video of machine learning practice course. In this video, we will study sklearn utilities for evaluating classifiers.

(Refer Slide Time: 0:21)



So far we learnt how to train classifiers for binary, multi-class and multi-label/output cases.

We will learn how to evaluate these classifiers with different scoring functions and with cross-validation.

We will also study how to set hyper-parameters for classifiers.

Many cross-validation and HPT methods discussed in the regression context are also applicable in classifiers.

- We will not repeat that discussion in this topic.
- Instead we will focus on only additional methods that are specific to classifiers.

So, far we learnt how to train classifiers for binary multi-class and multi-label and output cases. Now, we will learn how to evaluate these classifiers with different scoring functions and with cross validation. We will also study how to set hyper parameters for these classifiers.

Many cross validation and hyper parameter tuning methods discussed in the regression context are also applicable in the classifiers. We will not repeat this discussion and instead will focus only on additional methods that are specific to classifiers

`sklearn.model_selection` module provides the following three stratified APIs to create folds such that the overall class distribution is replicated in individual folds.


- `StratifiedKFold`
- `RepeatedStratifiedKFold`
- `StratifiedShuffleSplit`

Note: Folds obtained via `StratifiedShuffleSplit` may not be completely different.

Let us first study stratified cross-validation iterators. There may be issues like class imbalance in classification, which tend to impact the cross validation force. The overall class distribution and the ones in the folds may be different and this has implication in effective model training, `sklearn.model_selection` module provides three stratified APIs to create folds such that, overall class distribution is replicated in individual folds.

This stratified k-fold, repeated stratified k fold, and stratified shuffle split. The force obtained by stratified shuffle split may not be completely different. So, stratified k fold, and repetitive stratified k-fold is roughly equivalent to k-fold and stratified shuffle split is equivalent to shuffle split without certification.

(Refer Slide Time: 2:05)



LogisticRegressionCV

- Support in-built cross validation for optimizing hyperparameters
- The following are key parameters for HPT and cross validation

cv specifies	scoring specifies	cs specifies
cross validation iterator	scoring function to use for HPT	regularization strengths to experiment with.

- Choosing the best hyper-parameters

refit = True	Scores averaged across folds, values corresponding to the best score are selected and final refit with these parameters
refit = False	the coefs, intercepts and C that correspond to the best scores across folds are averaged.

So, we have logistic regression CV, which has support for inbuilt cross validation for optimizing hyper parameters. The following are key parameters for hyper parameter tuning and cross validation. CV that specifies cross validation iterator, scoring that specifies scoring function to use for hyper parameter tuning. And cs specifies regularization strength to experiment with.

Let us see how to choose the best hyper parameters. We can set refit to true and when we set refit to true, we score the average across force and values corresponding to the best core are selected. The final refit is performed with these parameters. If we set refit to false, then we get the coefficient intercept and value of C, that corresponds to the best course across folds based on averaging.

(Refer Slide Time: 3:09)



Now let's look at classification metrics implemented in sklearn.

Classification metrics

sklearn.metrics implements a bunch of classification scoring metrics based on true labels and predicted labels as inputs.

```
accuracy_score    balanced_accuracy_score  
top_k_accuracy_score    roc_auc_score  
precision_score    recall_score    f1_score
```

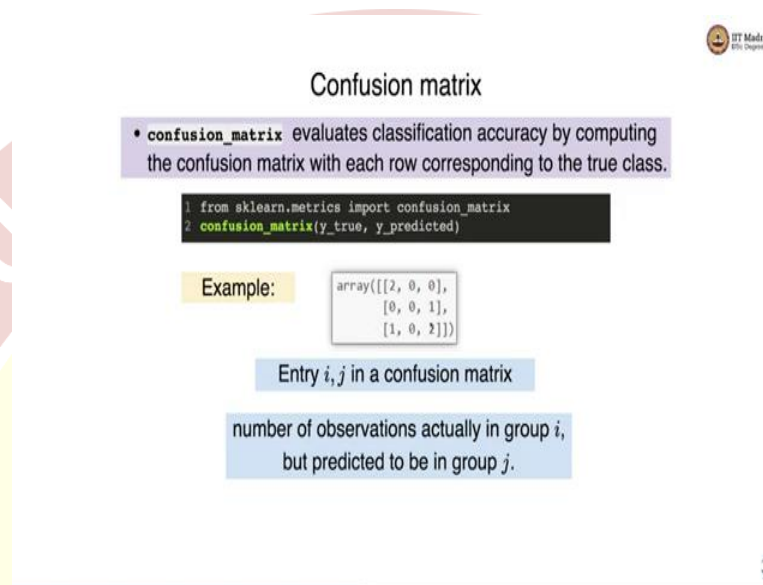
```
score(actual_labels, predicted_labels)
```

Now, let us look at classification metrics implemented in sklearn, sklearn implements a bunch of classification scoring metrics based on true labels and predicted labels as input. Most of the scoring functions has the following signature. So, this score is the new of the scoring function, it takes actual labels and predicted labels as input and returns the score.

There are variety of scoring metrics, accuracy_score, balance_accuracy_score, top_k_accuracy_score, roc_auc_score, precision_score, recall_score, and f1_score. So, these are variety of scoring metrics, that are implemented by sklearn. So, balanced_accuracy_score helps us to get accuracy score in case of imbalanced data set.

Whereas, `roc_auc_score` gives us the score under the area under the curve of the roc curve. And precision, recaller, `f1_score`, as we have been studying so far are the measures derived from the confusion matrix.

(Refer Slide Time: 4:20)



Confusion matrix

- `confusion_matrix` evaluates classification accuracy by computing the confusion matrix with each row corresponding to the true class.

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_true, y_predicted)
```

Example:

```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

Entry i, j in a confusion matrix

number of observations actually in group i ,
but predicted to be in group j .

Let us study confusion matrix, confusion matrix evaluates classification accuracy by computing the confusion matrix with each row corresponding to the true class. Confusion matrix is implemented in `sklearn.metric`. So, we can import the confusion matrix from `sklearn.metric` and we can compute the confusion matrix by providing the actual labels and the predicted labels.

The i, j th entry in the confusion matrix gives us the number of observations actually in group i , that are predicted to be in group j . So, here there are three labels and you can see that this value 2. So, this is the, this is the number of entries from label 0, which is predicted to be level 0. For example this 1 are the entries from label 0, label 2, but they are actually predicted as label 0.

So, this is some kind of a misclassification. This is another example of misclassification, where actual label is 1, but the predicted label is 2. And is an example of correct classification, where the actual level is 2 and predicted label is also 2.

(Refer Slide Time: 5:48)



Confusion matrix can be displayed with `ConfusionMatrixDisplay` API in `sklearn.metrics`.

- Confusion matrix

```
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
```

- From estimators

```
ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
```

- From predictions

```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

So, we can display confusion matrix in a pretty manner using confusion matrix display. So, we can get the confusion matrix display based on the pre-calculated confusion matrix, just as we saw you can also calculate confusion matrix from estimator by providing the estimator object and the examples and actual labels.

Or we can also plot the confusion matrix from prediction, where we provide the actual labels and the predicate labels and based on that the confusion matrix display is a is computed and then we have to call `plot.show` to in order to show this confusion matrix on the screen.

(Refer Slide Time: 6:42)



The `classification_report` function builds a text report showing the main classification metrics.

```
1 from sklearn.metrics import classification_report
2 print(classification_report(y_true, y_predicted))
```

	precision	recall	f1-score	support
class 0	0.67	1.00	0.80	2
class 1	0.00	0.00	0.00	1
class 2	1.00	0.50	0.67	2
accuracy			0.60	5
macro avg	0.56	0.50	0.49	5
weighted avg	0.67	0.60	0.59	5

So, we have `classification_report` utility that builds a text report showing the main classification metrics. So, `classification_report` utility is again implemented in `sklearn.metrics`. So, it takes the actual labels under predicted labels and produces a classification report.

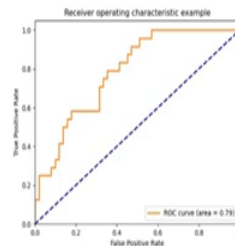
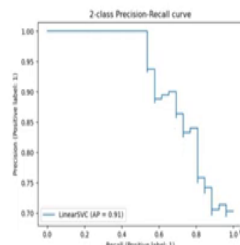
So, it has got precision recall and f1 score by the class label, it also has accuracy as well as micro and weighted averages. We will see what micro, what macro and weighted averages in the subsequent slides.

(Refer Slide Time: 7:29)



Classifier Performance across probability thresholds

```
1 from sklearn.metrics import precision_recall_curve
2 precision, recall, thresholds = precision_recall_curve(y_true, y_predicted)
```



```
1 from sklearn.metrics import roc_curve
2 fpr, tpr, thresholds = metrics.roc_curve(y_true, y_scores, pos_label=2)
```


Sklearn also supports precision recall curve and roc auc curve, that helps us to gauge the performance of the classifier across probability thresholds. So, precision_recall_curve is again implemented in sklearn.metrics. And as like any other metric, it just takes the actual labels and the predicted labels.

And it gives us the value of precision recall for every threshold. We can take the output and plot it in form of a precision recall curve, where we have recall on X-axis and precision on the Y-axis. In the same manner, we can also calculate roc curve. So, in case of roc curve again we gave the actual label under predicted label and we get false positive rate, true positive rate for every threshold.

And when we plot it we have fpr, or false positive rate on the X axis and true positive rate on the Y axis. So, we get area under the roc curve as a measure from the roc curve, whereas from the precision recall curve we get what is called as average precision.

(Refer Slide Time: 8:51)

How to extend binary metric to multiclass or multilabel problems?

- Treat data as a collection of binary problems, one for each class.
- Then, average binary metric calculations across the set of classes.
- Can be done using **average** parameter.

macro	calculates the mean of the binary metrics
weighted	computes the average of binary metrics in which each class's score is weighted by its presence in the true data sample.
micro	gives each sample-class pair an equal contribution to the overall metric
samples	calculates the metric over the true and predicted classes for each sample in the evaluation data, and returns their average
None	returns an array with the score for each class

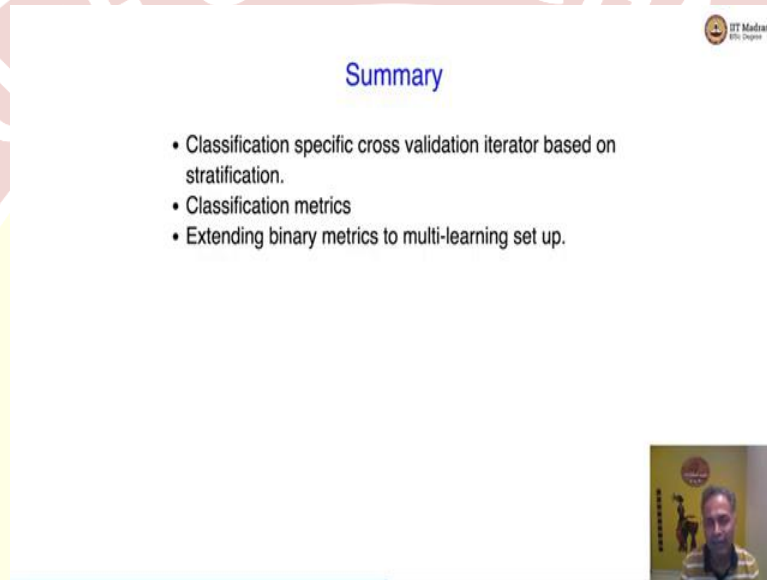
So, far we studied classification metrics for binary classification. Let us study how to extend these metrics to multiclass and multilabel problems. The key idea here is to treat the data as a collection of binary problems one for each class. And in average binary metric calculations across the set of classes.

So, this is typically done with the average parameter and average takes various values, that are macro, weighted, micro, samples, and none. So, macro calculates the mean of binary matrix,

weighted average computes the average of binary matrix in which each, in which each class is weighted by the presence of the number of samples.

Micro gives each sample class pair an equal contribution to the overall matrix. Samples calculate the metric over true and predicted classes for each sample in the evaluation data and returns their averages. And when we say average equal to none, we get an array with score for each class.

(Refer Slide Time: 10:06)



Summary

- Classification specific cross validation iterator based on stratification.
- Classification metrics
- Extending binary metrics to multi-learning set up.

In this video, we studied classification specific cross validation iterators. They mainly make use of stratification. We also studied different classification metrics implemented in sklearn and how to extend the binary metrics to multi-learning set up.