6.9. Transforming the prediction target (y)

These are transformers that are not intended to be used on features, only on supervised learning targets. See also Transforming target in regression if you want to transform the prediction target for learning, but evaluate the model in the original (untransformed) space.

6.9.1. Label binarization

6.9.1.1. LabelBinarizer

`LabelBinarizer` is a utility class to help create a label indicator matrix from a list of multiclass labels:

```
>>> from sklearn import preprocessing
>>> lb = preprocessing.LabelBinarizer()
>>> lb.fit([1, 2, 6, 4, 2])
LabelBinarizer()
>>> lb.classes_
array([1, 2, 4, 6])
>>> lb.transform([1, 6])
array([[1, 0, 0, 0],
       [0, 0, 0, 1]])
```

Using this format can enable multiclass classification in estimators that support the label indicator matrix format.

Warning: LabelBinarizer is not needed if you are using an estimator that already supports multiclass data.

For more information about multiclass classification, refer to Multiclass classification.

6.9.1.2. MultiLabelBinarizer

In multilabel learning, the joint set of binary classification tasks is expressed with a label binary indicator array: each sample is one row of a 2d array of shape (n_samples, n_classes) with binary values where the one, i.e. the non zero elements, corresponds to the subset of labels for that sample. An array such as `np.array([[1, 0, 0], [0, 1, 1], [0, 0, 0]])` represents label 0 in the first sample, labels 1 and 2 in the second sample, and no labels in the third sample.

Producing multilabel data as a list of sets of labels may be more intuitive. The `MultiLabelBinarizer` transformer can be used to convert between a collection of collections of labels and the indicator format:

```
>>> from sklearn.preprocessing import MultiLabelBinarizer
>>> y = [[2, 3, 4], [2], [0, 1, 3], [0, 1, 2, 3, 4], [0, 1, 2]]
>>> MultiLabelBinarizer().fit_transform(y)
array([[0, 0, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [1, 1, 0, 1, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 0, 0]])
```

For more information about multilabel classification, refer to Multilabel classification.

6.9.2. Label encoding

`LabelEncoder` is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1. This is sometimes useful for writing efficient Cython routines. `LabelEncoder` can be used as follows:

```
>>> from sklearn import preprocessing
>>> le = preprocessing.LabelEncoder()
>>> le.fit([1, 2, 2, 6])
LabelEncoder()
>>> le.classes_
array([1, 2, 6])
>>> le.transform([1, 1, 2, 6])
array([0, 0, 1, 2])
>>> le.inverse_transform([0, 0, 1, 2])
array([1, 1, 2, 6])
```

It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels:

Toggle Menu

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1])
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```