sklearn.preprocessing.OrdinalEncoder

»

class `sklearn.preprocessing.OrdinalEncoder`(*categories='auto'*, *dtype=<class 'numpy.float64'>*) ¶ [source]

Encode categorical features as an integer array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are converted to ordinal integers. This results in a single column of integers (0 to n_categories - 1) per feature.

Read more in the User Guide.

Parameters: **categories** : *'auto' or a list of lists/arrays of values.*

Categories (unique values) per feature:

- 'auto' : Determine categories automatically from the training data.
- list : `categories[i]` holds the categories expected in the ith column. The passed categories should not mix strings and numeric values, and should be sorted in case of numeric values.

The used categories can be found in the `categories_` attribute.

dtype : *number type, default np.float64*

Desired dtype of output.

Attributes: **categories_** : *list of arrays*

The categories of each feature determined during fitting (in order of the features in X and corresponding with the output of `transform`).

See also:

`sklearn.preprocessing.OneHotEncoder`

performs a one-hot encoding of categorical features.

`sklearn.preprocessing.LabelEncoder`

encodes target labels with values between 0 and n_classes-1.

Examples

Given a dataset with two features, we let the encoder find the unique values per feature and transform the data to an ordinal encoding.

```
>>> from sklearn.preprocessing import OrdinalEncoder
>>> enc = OrdinalEncoder()
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
>>> enc.fit(X)
...
OrdinalEncoder(categories='auto', dtype=<... 'numpy.float64'>)
>>> enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> enc.transform([['Female', 3], ['Male', 1]])
array([[0., 2.],
       [1., 0.]])
```

```
>>> enc.inverse_transform([[1, 0], [0, 1]])
array([['Male', 1],
       ['Female', 2]], dtype=object)
```

Methods

`fit`(self, X[, y])	Fit the OrdinalEncoder to X.
`fit_transform`(self, X[, y])	Fit to data, then transform it.
`get_params`(self[, deep])	Get parameters for this estimator.
`inverse_transform`(self, X)	Convert the data back to the original representation.
`set_params`(self, **params)	Set the parameters of this estimator.
`transform`(self, X)	Transform X to ordinal codes.

`__init__`(*self, categories='auto', dtype=<class 'numpy.float64'>*) [source]

`fit`(*self, X, y=None*) [source]

Fit the OrdinalEncoder to X.

Parameters:	**X** : *array-like, shape [n_samples, n_features]* The data to determine the categories of each feature.
Returns:	self

`fit_transform`(*self, X, y=None, **fit_params*) [source]

Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters:	**X** : *numpy array of shape [n_samples, n_features]* Training set. **y** : *numpy array of shape [n_samples]* Target values.

Returns: **X_new** : *numpy array of shape [n_samples, n_features_new]*

Transformed array.

`get_params`(*self*, *deep=True*) [source]

Get parameters for this estimator.

Parameters: **deep** : *boolean, optional*

If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns: **params** : *mapping of string to any*

Parameter names mapped to their values.

`inverse_transform`(*self*, *X*) [source]

Convert the data back to the original representation.

Parameters: **X** : *array-like or sparse matrix, shape [n_samples, n_encoded_features]*

The transformed data.

Returns: **X_tr** : *array-like, shape [n_samples, n_features]*

Inverse transformed array.

`set_params`(*self*, ***params*) [source]

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Returns: self

`transform`(*self*, *X*) [source]

Transform X to ordinal codes.

Parameters: **X** : *array-like, shape [n_samples, n_features]*

The data to encode.

Returns: **X_out** : *sparse matrix or a 2-d array*

Transformed input.