# sklearn.preprocessing.OneHotEncoder

»

*class* `sklearn.preprocessing.`**`OneHotEncoder`**(*n_values=None*, *categorical_features=None*, *categories=None*, *drop=None*, *sparse=True*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*) [source]

Encode categorical integer features as a one-hot numeric 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 encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array.

By default, the encoder derives the categories based on the unique values in each feature. Alternatively, you can also specify the `categories` manually. The OneHotEncoder previously assumed that the input features take on values in the range [0, max(values)). This behaviour is deprecated.

This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

Note: a one-hot encoding of y labels should use a LabelBinarizer instead.

Read more in the User Guide.

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

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 within a single feature, and should be sorted in case of numeric values.

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

**drop** : *'first' or a list/array of shape (n_features,), default=None.*

Specifies a methodology to use to drop one of the categories per feature. This is useful in situations where perfectly collinear features cause problems, such as when feeding the resulting data into a neural network or an unregularized regression.

- None : retain all features (the default).
- 'first' : drop the first category in each feature. If only one category is present, the feature will be dropped entirely.

- array : `drop[i]` is the category in feature `X[:, i]` that should be dropped.

**sparse** : *boolean, default=True*

Will return sparse matrix if set True else will return an array.

**dtype** : *number type, default=np.float*

Desired dtype of output.

**handle_unknown** : *'error' or 'ignore', default='error'.*

Whether to raise an error or ignore if an unknown categorical feature is present during transform (default is to raise). When this parameter is set to 'ignore' and an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will be all zeros. In the inverse transform, an unknown category will be denoted as None.

**n_values** : *'auto', int or array of ints, default='auto'*

Number of values per feature.

- 'auto' : determine value range from training data.
- int : *number of categorical values per feature.*

  Each feature value should be in `range(n_values)`

- array : *`n_values[i]` is the number of categorical values in*

  `X[:, i]`. Each feature value should be in `range(n_values[i])`

*Deprecated since version 0.20:* The `n_values` keyword was deprecated in version 0.20 and will be removed in 0.22. Use `categories` instead.

**categorical_features** : *'all' or array of indices or mask, default='all'*

Specify what features are treated as categorical.

- 'all': All features are treated as categorical.
- array of indices: Array of categorical feature indices.
- mask: Array of length n_features and with dtype=bool.

Non-categorical features are always stacked to the right of the matrix.

*Deprecated since version 0.20:* The `categorical_features` keyword was deprecated in version 0.20 and will be removed in 0.22. You can use the `ColumnTransformer` instead.

**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`). This includes the category specified in `drop` (if any).

**drop_idx_** : *array of shape (n_features,)*

`drop_idx_[i]` is the index in `categories_[i]` of the category to be dropped for each feature. None if all the transformed features will be retained.

**active_features_** : *array*

Indices for active features, meaning values that actually occur in the training set. Only available when n_values is `'auto'`.

> *Deprecated since version 0.20:* The `active_features_` attribute was deprecated in version 0.20 and will be removed in 0.22.

**feature_indices_** : *array of shape (n_features,)*

Indices to feature ranges. Feature `i` in the original data is mapped to features from `feature_indices_[i]` to `feature_indices_[i+1]` (and then potentially masked by `active_features_` afterwards)

> *Deprecated since version 0.20:* The `feature_indices_` attribute was deprecated in version 0.20 and will be removed in 0.22.

**n_values_** : *array of shape (n_features,)*

Maximum number of values per feature.

> *Deprecated since version 0.20:* The `n_values_` attribute was deprecated in version 0.20 and will be removed in 0.22.

---

**See also:**

**`sklearn.preprocessing.OrdinalEncoder`**
performs an ordinal (integer) encoding of the categorical features.

**`sklearn.feature_extraction.DictVectorizer`**
performs a one-hot encoding of dictionary items (also handles string-valued features).

**`sklearn.feature_extraction.FeatureHasher`**
performs an approximate one-hot encoding of dictionary items or strings.

**`sklearn.preprocessing.LabelBinarizer`**
binarizes labels in a one-vs-all fashion.

**`sklearn.preprocessing.MultiLabelBinarizer`**
transforms between iterable of iterables and a multilabel format, e.g. a (samples x classes) binary matrix indicating the presence of a class label.

### Examples

Given a dataset with two features, we let the encoder find the unique values per feature and transform the data to a binary one-hot encoding.

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> enc = OneHotEncoder(handle_unknown='ignore')
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
>>> enc.fit(X)
...
```

```
...
OneHotEncoder(categorical_features=None, categories=None, drop=None,
   dtype=<... 'numpy.float64'>, handle_unknown='ignore',
   n_values=None, sparse=True)
```

```
>>> enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> enc.transform([['Female', 1], ['Male', 4]]).toarray()
array([[1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
>>> enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])
array([['Male', 1],
       [None, 2]], dtype=object)
>>> enc.get_feature_names()
array(['x0_Female', 'x0_Male', 'x1_1', 'x1_2', 'x1_3'], dtype=object)
>>> drop_enc = OneHotEncoder(drop='first').fit(X)
>>> drop_enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> drop_enc.transform([['Female', 1], ['Male', 2]]).toarray()
array([[0., 0., 0.],
       [1., 1., 0.]])
```

## Methods

| | |
|---|---|
| **fit**(self, X[, y]) | Fit OneHotEncoder to X. |
| **fit_transform**(self, X[, y]) | Fit OneHotEncoder to X, then transform X. |
| **get_feature_names**(self[, input_features]) | Return feature names for output features. |
| **get_params**(self[, deep]) | Get parameters for this estimator. |
| **inverse_transform**(self, X) | Convert the back data to the original representation. |
| **set_params**(self, **params) | Set the parameters of this estimator. |
| **transform**(self, X) | Transform X using one-hot encoding. |

**__init__**(*self*, *n_values=None*, *categorical_features=None*, *categories=None*, *drop=None*, *sparse=True*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*) [source]

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

Fit OneHotEncoder 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*) [source]

Fit OneHotEncoder to X, then transform X.

Equivalent to fit(X).transform(X) but more convenient.

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

The data to encode.

**Returns:** **X_out** : *sparse matrix if sparse=True else a 2-d array*

Transformed input.

`get_feature_names`(*self*, *input_features=None*) [source]

Return feature names for output features.

**Parameters:** **input_features** : *list of string, length n_features, optional*

String names for input features if available. By default, “x0”, “x1”, … “xn_-features” is used.

**Returns:** **output_feature_names** : *array of string, length n_output_features*

`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 back data to the original representation.

In case unknown categories are encountered (all zeros in the one-hot encoding), `None` is used to represent this category.

**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 using one-hot encoding.

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

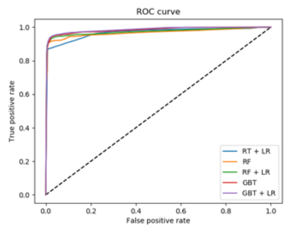
The data to encode.

**Returns:** **X_out** : *sparse matrix if sparse=True else a 2-d array*

Transformed input.

## Examples using `sklearn.preprocessing.OneHotEncoder`

Column Transformer with Mixed Types



Feature transformations with ensembles of trees