sklearn.preprocessing.OneHotEncoder

class `sklearn.preprocessing.`**`OneHotEncoder`**(**, categories='auto', drop=None, sparse='deprecated', sparse_output=True, dtype=<class 'numpy.float64'>, handle_unknown='error', min_frequency=None, max_categories=None, feature_name_combiner='concat'*) [source]

Encode categorical 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 (depending on the `sparse_output` parameter)

By default, the encoder derives the categories based on the unique values in each feature. Alternatively, you can also specify the `categories` manually.

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. For a comparison of different encoders, refer to: Comparing Target Encoder with Other Encoders.

Parameters:

categories : *'auto' or a list of array-like, 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.

New in version 0.20.

drop : *{'first', 'if_binary'} or an array-like 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 an unregularized linear regression model.

However, dropping one category breaks the symmetry of the original representation and can therefore induce a bias in downstream models, for instance for penalized linear classification or regression models.

- 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.
- 'if_binary' : drop the first category in each feature with two categories. Features with 1 or more than 2 categories are left intact.
- array : `drop[i]` is the category in feature `X[:, i]` that should be dropped.

When `max_categories` or `min_frequency` is configured to group infrequent categories, the dropping behavior is handled after the grouping.

New in version 0.21: The parameter `drop` was added in 0.21.

Changed in version 0.23: The option `drop='if_binary'` was added in 0.23.

Changed in version 1.1: Support for dropping infrequent categories.

sparse : *bool, default=True*

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

Deprecated since version 1.2: `sparse` is deprecated in 1.2 and will be removed in 1.4. Use `sparse_output` instead.

sparse_output : *bool, default=True*

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

New in version 1.2: `sparse` was renamed to `sparse_output`

dtype : *number type, default=np.float64*

Desired dtype of output.

handle_unknown : *{'error', 'ignore', 'infrequent_if_exist'}, default='error'*

Specifies the way unknown categories are handled during `transform`.

- 'error' : Raise an error if an unknown category is present during transform.
- 'ignore' : When 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.
- 'infrequent_if_exist' : When an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will map to the infrequent category if it exists. The infrequent category will be mapped to the last position in the encoding. During inverse transform, an unknown category will be mapped to the category denoted `'infrequent'` if it exists. If the `'infrequent'` category does not exist, then `transform` and `inverse_transform` will handle an unknown category as with `handle_unknown='ignore'`. Infrequent categories exist based on `min_frequency` and `max_categories`. Read more in the User Guide.

Changed in version 1.1: `'infrequent_if_exist'` was added to automatically handle unknown categories and infrequent categories.

min_frequency : *int or float, default=None*

Specifies the minimum frequency below which a category will be considered infrequent.

- If `int`, categories with a smaller cardinality will be considered infrequent.
- If `float`, categories with a smaller cardinality than `min_frequency * n_samples` will be considered infrequent.

New in version 1.1: Read more in the User Guide.

max_categories : *int, default=None*

Specifies an upper limit to the number of output features for each input feature when considering infrequent categories. If there are infrequent categories, `max_categories` includes the category representing the infrequent categories along with the frequent categories. If `None`, there is no limit to the number of output features.

New in version 1.1: Read more in the User Guide.

feature_name_combiner : *"concat" or callable, default="concat"*

Callable with signature `def callable(input_feature, category)` that returns a string. This is used to create feature names to be returned by `get_feature_names_out`.

`"concat"` concatenates encoded feature name and category with `feature + "_" + str(category)`.E.g. feature X with values 1, 6, 7 create feature names `X_1, X_6, X_7`.

New in version 1.3.

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).

`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.
- `drop_idx_[i] = None` if no category is to be dropped from the feature with index `i`, e.g. when `drop='if_binary'` and the feature isn't binary.
- `drop_idx_ = None` if all the transformed features will be retained.

If infrequent categories are enabled by setting `min_frequency` or `max_categories` to a non-default value and `drop_idx[i]` corresponds to a infrequent category, then the entire infrequent category is dropped.

Changed in version 0.23: Added the possibility to contain `None` values.

`infrequent_categories_` : *list of ndarray*

Infrequent categories for each feature.

n_features_in_ : *int*

Number of features seen during fit.

New in version 1.0.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

feature_name_combiner : *callable or None*

Callable with signature `def callable(input_feature, category)` that returns a string. This is used to create feature names to be returned by `get_feature_names_out`.

New in version 1.3.

See also:

`OrdinalEncoder`
Performs an ordinal (integer) encoding of the categorical features.

`TargetEncoder`
Encodes categorical features using the target.

`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.

`LabelBinarizer`
Binarizes labels in a one-vs-all fashion.

`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
```

One can discard categories not seen during `fit`:

```
>>> enc = OneHotEncoder(handle_unknown='ignore')
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
>>> enc.fit(X)
OneHotEncoder(handle_unknown='ignore')
>>> 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_out(['gender', 'group'])
array(['gender_Female', 'gender_Male', 'group_1', 'group_2', 'group_3'], ...)
```

One can always drop the first column for each feature:

```
>>> 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.]])
```

Or drop a column for feature only having 2 categories:

```
>>> drop_binary_enc = OneHotEncoder(drop='if_binary').fit(X)
>>> drop_binary_enc.transform([['Female', 1], ['Male', 2]]).toarray()
array([[0., 1., 0., 0.],
       [1., 0., 1., 0.]])
```

One can change the way feature names are created.

```
>>> def custom_combiner(feature, category):
...     return str(feature) + "_" + type(category).__name__ + "_" + str(category)
>>> custom_fnames_enc = OneHotEncoder(feature_name_combiner=custom_combiner).fit(X)
>>> custom_fnames_enc.get_feature_names_out()
array(['x0_str_Female', 'x0_str_Male', 'x1_int_1', 'x1_int_2', 'x1_int_3'],
      dtype=object)
```

Infrequent categories are enabled by setting `max_categories` or `min_frequency`.

```
>>> import numpy as np
>>> X = np.array([["a"] * 5 + ["b"] * 20 + ["c"] * 10 + ["d"] * 3], dtype=object).T
>>> ohe = OneHotEncoder(max_categories=3, sparse_output=False).fit(X)
>>> ohe.infrequent_categories_
[array(['a', 'd'], dtype=object)]
>>> ohe.transform([["a"], ["b"]])
array([[0., 0., 1.],
       [1., 0., 0.]])
```

Methods

`fit`(X[, y])	Fit OneHotEncoder to X.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_feature_names_out`([input_features])	Get output feature names for transformation.
`get_metadata_routing`()	Get metadata routing of this object.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(X)	Convert the data back to the original representation.
`set_output`(*[, transform])	Set output container.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Transform X using one-hot encoding.

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

Fit OneHotEncoder to X.

Parameters:

X : *array-like of shape (n_samples, n_features)*
The data to determine the categories of each feature.

y : *None*
Ignored. This parameter exists only for compatibility with `Pipeline`.

Returns:

self
Fitted encoder.

`fit_transform`(*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 : *array-like of shape (n_samples, n_features)*
Input samples.

y : *array-like of shape (n_samples,) or (n_samples, n_outputs), default=None*
Target values (None for unsupervised transformations).

****fit_params : *dict***
Additional fit parameters.

Returns:

X_new : *ndarray array of shape (n_samples, n_features_new)*
Transformed array.

`get_feature_names_out`(*input_features=None*) [source]

Get output feature names for transformation.

Parameters:

input_features : *array-like of str or None, default=None*
Input features.

- If `input_features` is `None`, then `feature_names_in_` is used as feature names in. If `feature_names_in_` is not defined, then the following input feature names are generated: `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
- If `input_features` is an array-like, then `input_features` must match `feature_names_in_` if `feature_names_in_` is defined.

Returns:

feature_names_out : *ndarray of str objects*
Transformed feature names.

`get_metadata_routing`() [source]

Get metadata routing of this object.

Please check User Guide on how the routing mechanism works.

Returns:

routing : *MetadataRequest*
A `MetadataRequest` encapsulating routing information.

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

Get parameters for this estimator.

Parameters:

deep : *bool, default=True*
If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns:

params : *dict*
Parameter names mapped to their values.

property `infrequent_categories_`
Infrequent categories for each feature.

`inverse_transform`(*X*) [source]

Convert the data back to the original representation.

When unknown categories are encountered (all zeros in the one-hot encoding), `None` is used to represent this category. If the feature with the unknown category has a dropped category, the dropped category will be its inverse.

For a given input feature, if there is an infrequent category, 'infrequent_sklearn' will be used to represent the infrequent category.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_encoded_features)*
The transformed data.

Returns:

X_tr : *ndarray of shape (n_samples, n_features)*
Inverse transformed array.

`set_output`(**, transform=None*) [source]

Set output container.

See Introducing the set_output API for an example on how to use the API.

Parameters:

transform : *{"default", "pandas"}, default=None*

Configure output of `transform` and `fit_transform`.

- `"default"`: Default output format of a transformer
- `"pandas"`: DataFrame output
- `None`: Transform configuration is unchanged

Returns:

self : *estimator instance*

Estimator instance.

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

Set the parameters of this estimator.

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

Parameters:

****params : *dict***

Estimator parameters.

Returns:

self : *estimator instance*

Estimator instance.

`transform`(*X*) [source]

Transform X using one-hot encoding.

If there are infrequent categories for a feature, the infrequent categories will be grouped into a single category.

Parameters:

X : *array-like of shape (n_samples, n_features)*

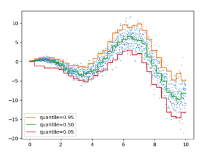
The data to encode.

Returns:

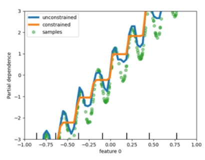
X_out : *{ndarray, sparse matrix} of shape (n_samples, n_encoded_features)*

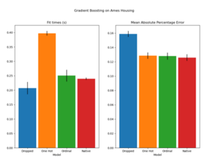
Transformed input. If `sparse_output=True`, a sparse matrix will be returned.

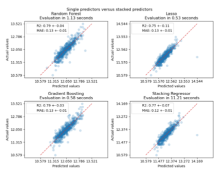
Examples using `sklearn.preprocessing.OneHotEncoder`

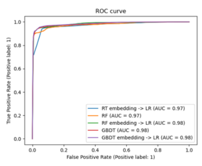

Release Highlights for scikit-learn 1.1

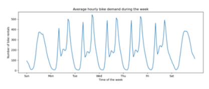
Release Highlights for scikit-learn 1.0

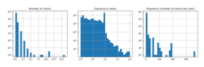

Release Highlights for scikit-learn 0.23

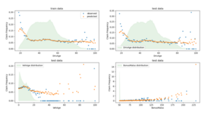

Categorical Feature Support in Gradient Boosting

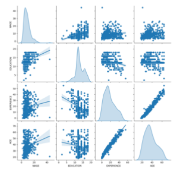

Combine predictors using stacking

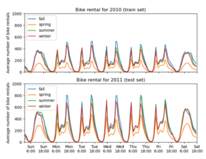

Feature transformations with ensembles of trees


Time-related feature engineering


Poisson regression and non-normal loss

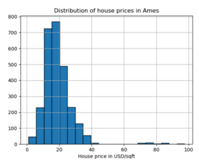

Tweedie regression on insurance claims

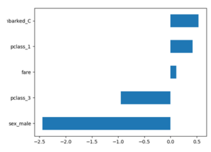

Common pitfalls in the interpretation of coefficients of linear models


Partial Dependence and Individual Conditional Expectation Plots

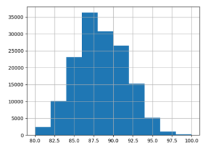
Displaying Pipelines

Displaying estimators and complex pipelines


Evaluation of outlier detection estimators


Introducing the set_output API

Column Transformer with Mixed Types


Comparing Target Encoder with Other Encoders

Toggle Menu