sklearn.feature_extraction.text.TfidfTransformer

class sklearn.feature_extraction.text.**TfidfTransformer**(**, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*) [source]

Transform a count matrix to a normalized tf or tf-idf representation.

Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.

The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

The formula that is used to compute the tf-idf for a term t of a document d in a document set is tf-idf(t, d) = tf(t, d) * idf(t), and the idf is computed as idf(t) = log [n / df(t)] + 1 (if `smooth_idf=false`), where n is the total number of documents in the document set and df(t) is the document frequency of t; the document frequency is the number of documents in the document set that contain the term t. The effect of adding "1" to the idf in the equation above is that terms with zero idf, i.e., terms that occur in all documents in a training set, will not be entirely ignored. (Note that the idf formula above differs from the standard textbook notation that defines the idf as idf(t) = log [n / (df(t) + 1)]).

If `smooth_idf=true` (the default), the constant "1" is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions: idf(t) = log [(1 + n) / (1 + df(t))] + 1.

Furthermore, the formulas used to compute tf and idf depend on parameter settings that correspond to the SMART notation used in IR as follows:

Tf is "n" (natural) by default, "l" (logarithmic) when `sublinear_tf=true`. Idf is "t" when use_idf is given, "n" (none) otherwise. Normalization is "c" (cosine) when `norm='l2'`, "n" (none) when `norm=None`.

Read more in the User Guide.

Parameters:

norm : {'l1', 'l2'} or None, default='l2'
Each output row will have unit norm, either:

- 'l2': Sum of squares of vector elements is 1. The cosine similarity between two vectors is their dot product when l2 norm has been applied.
- 'l1': Sum of absolute values of vector elements is 1. See `preprocessing.normalize`.
- None: No normalization.

use_idf : bool, default=True
Enable inverse-document-frequency reweighting. If False, idf(t) = 1.

smooth_idf : bool, default=True
Smooth idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. Prevents zero divisions.

sublinear_tf : bool, default=False
Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

Attributes:

`idf_` : array of shape (n_features)
Inverse document frequency vector, only defined if `use_idf=True`.

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.

See also:

CountVectorizer
Transforms text into a sparse matrix of n-gram counts.

TfidfVectorizer
Convert a collection of raw documents to a matrix of TF-IDF features.

HashingVectorizer
Convert a collection of text documents to a matrix of token occurrences.

References

[Yates2011]
R. Baeza-Yates and B. Ribeiro-Neto (2011). Modern Information Retrieval. Addison Wesley, pp. 68-74.

[MRS2008]
C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 118-120.

Examples

```
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> from sklearn.pipeline import Pipeline
>>> corpus = ['this is the first document',
...           'this document is the second document',
...           'and this is the third one',
...           'is this the first document']
>>> vocabulary = ['this', 'document', 'first', 'is', 'second', 'the',
...               'and', 'one']
>>> pipe = Pipeline([('count', CountVectorizer(vocabulary=vocabulary)),
...                  ('tfid', TfidfTransformer())]).fit(corpus)
>>> pipe['count'].transform(corpus).toarray()
array([[1, 1, 1, 1, 0, 1, 0, 0],
       [2, 2, 0, 1, 1, 1, 0, 0],
       [1, 0, 0, 1, 0, 1, 1, 1],
       [1, 1, 1, 1, 0, 1, 0, 0]])
>>> pipe['tfid'].idf_
array([1.        , 1.22314355, 1.51082562, 1.        , 1.91629073,
       1.91629073, 1.91629073, 1.91629073])
>>> pipe.transform(corpus).shape
(4, 8)
```

Methods

`fit`(X[, y])	Learn the idf vector (global term weights).
`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.
`set_output`(*[, transform])	Set output container.
`set_params`(**params)	Set the parameters of this estimator.
`set_transform_request`(*[, copy])	Request metadata passed to the `transform` method.
`transform`(X[, copy])	Transform a count matrix to a tf or tf-idf representation.

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

Learn the idf vector (global term weights).

Parameters:

X : *sparse matrix of shape n_samples, n_features)*
A matrix of term/token counts.

y : None
This parameter is not needed to compute tf-idf.

Returns:

self : *object*
Fitted transformer.

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*
Same as input features.

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* idf_**
Inverse document frequency vector, only defined if `use_idf=True`.

Returns:

ndarray of shape (n_features,)

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.

set_transform_request(**, copy: Union[bool, None, str] = '$UNCHANGED$'*) → TfidfTransformer [source]

Request metadata passed to the `transform` method.

Note that this method is only relevant if `enable_metadata_routing=True` (see `sklearn.set_config`). Please see User Guide on how the routing mechanism works.

The options for each parameter are:

- `True`: metadata is requested, and passed to `transform` if provided. The request is ignored if metadata is not provided.
- `False`: metadata is not requested and the meta-estimator will not pass it to `transform`.
- `None`: metadata is not requested, and the meta-estimator will raise an error if the user provides it.
- `str`: metadata should be passed to the meta-estimator with this given alias instead of the original name.

The default (`sklearn.utils.metadata_routing.UNCHANGED`) retains the existing request. This allows you to change the request for some parameters and not others.

New in version 1.3.

Note: This method is only relevant if this estimator is used as a sub-estimator of a meta-estimator, e.g. used inside a `pipeline.Pipeline`. Otherwise it has no effect.

Parameters:

copy : *str, True, False, or None, default=`sklearn.utils.metadata_routing.UNCHANGED`*
Metadata routing for `copy` parameter in `transform`.

Returns:

self : *object*
The updated object.

transform(*X*, *copy=True*) [source]

Transform a count matrix to a tf or tf-idf representation.

Parameters:

X : *sparse matrix of (n_samples, n_features)*
A matrix of term/token counts.

copy : *bool, default=True*
Whether to copy X and operate on the copy or perform in-place operations.

Returns:

vectors : *sparse matrix of shape (n_samples, n_features)*
Tf-idf-weighted document-term matrix.

Examples using `sklearn.feature_extraction.text.TfidfTransformer`

Semi-supervised Classification on a Text Dataset


Clustering text documents using k-means


FeatureHasher and DictVectorizer Comparison