sklearn.feature_extraction.text.TfidfVectorizer¶

scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

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

Equivalent to CountVectorizer followed by TfidfTransformer.

Read more in the User Guide.

input : string {‘filename’, ‘file’, ‘content’}

> *If ‘filename’, the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.*
>
> *If ‘file’, the sequence items must have a ‘read’ method (file-like object) that is called to fetch the bytes in memory.*
>
> *Otherwise the input is expected to be the sequence strings or bytes items are expected to be analyzed directly.*

encoding : string, ‘utf-8’ by default.

> *If bytes or files are given to analyze, this encoding is used to decode.*

decode_error : {‘strict’, ‘ignore’, ‘replace’}

> *Instruction on what to do if a byte sequence is given to analyze that contains characters not of the given encoding. By default, it is ‘strict’, meaning that a UnicodeDecodeError will be raised. Other values are ‘ignore’ and ‘replace’.*

strip_accents : {‘ascii’, ‘unicode’, None}

> *Remove accents during the preprocessing step. ‘ascii’ is a fast method that only works on characters that have an direct ASCII mapping. ‘unicode’ is a slightly slower method that works on any characters. None (default) does nothing.*

analyzer : string, {‘word’, ‘char’} or callable

> *Whether the feature should be made of word or character n-grams.*
>
> *If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.*

preprocessor : callable or None (default)

> *Override the preprocessing (string transformation) stage while preserving the tokenizing and n-grams generation steps.*

tokenizer : callable or None (default)

> *Override the string tokenization step while preserving the preprocessing and n-grams generation steps. Only applies if* `analyzer == 'word'`.

ngram_range : tuple (min_n, max_n)

> *The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used.*

stop_words : string {'english'}, list, or None (default)

> *If a string, it is passed to _check_stop_list and the appropriate stop list is returned. 'english' is currently the only supported string value.*
>
> *If a list, that list is assumed to contain stop words, all of which will be removed from the resulting tokens. Only applies if* `analyzer == 'word'`.
>
> *If None, no stop words will be used. max_df can be set to a value in the range [0.7, 1.0) to automatically detect and filter stop words based on intra corpus document frequency of terms.*

lowercase : boolean, default True

> *Convert all characters to lowercase before tokenizing.*

token_pattern : string

> *Regular expression denoting what constitutes a "token", only used if* `analyzer == 'word'`. *The default regexp selects tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).*

max_df : float in range [0.0, 1.0] or int, default=1.0

> *When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.*

min_df : float in range [0.0, 1.0] or int, default=1

> *When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.*

max_features : int or None, default=None

> *If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.*
>
> *This parameter is ignored if vocabulary is not None.*

vocabulary : Mapping or iterable, optional

> *Either a Mapping (e.g., a dict) where keys are terms and values are indices in the feature matrix, or an iterable over terms. If not given, a vocabulary is determined from the input documents.*

binary : boolean, default=False

> *If True, all non-zero term counts are set to 1. This does not mean outputs will have only 0/1 values, only that the tf term in tf-idf is binary. (Set idf and normalization to False to get 0/1 outputs.)*

dtype : type, optional

> *Type of the matrix returned by fit_transform() or transform().*

norm : ‘l1’, ‘l2’ or None, optional

> *Norm used to normalize term vectors. None for no normalization.*

use_idf : boolean, default=True

> *Enable inverse-document-frequency reweighting.*

smooth_idf : boolean, 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 : boolean, default=False

> *Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).*

Parameters:

---

vocabulary_ : dict

> *A mapping of terms to feature indices.*

idf_ : array, shape = [n_features], or None

> *The learned idf vector (global term weights) when `use_idf` is set to True, None otherwise.*

stop_words_ : set

> *Terms that were ignored because they either:*
>
> > - *occurred in too many documents (max_df)*
> > - *occurred in too few documents (min_df)*
> > - *were cut off by feature selection (max_features).*
>
> *This is only available if no vocabulary was given.*

Attributes:

See also

`CountVectorizer`
Tokenize the documents and count the occurrences of token and return them as a sparse matrix
`TfidfTransformer`
Apply Term Frequency Inverse Document Frequency normalization to a sparse matrix of occurrence counts.

Notes

The `stop_words_` attribute can get large and increase the model size when pickling. This attribute is provided only for introspection and can be safely removed using delattr or set to None before pickling.

Methods

`build_analyzer`()	Return a callable that handles preprocessing and tokenization
`build_preprocessor`()	Return a function to preprocess the text before tokenization

build_tokenizer()	Return a function that splits a string into a sequence of tokens
decode(doc)	Decode the input into a string of unicode symbols
fit (raw_documents[, y])	Learn vocabulary and idf from training set.
fit_transform (raw_documents[, y])	Learn vocabulary and idf, return term-document matrix.
get_feature_names()	Array mapping from feature integer indices to feature name
get_params([deep])	Get parameters for this estimator.
get_stop_words()	Build or fetch the effective stop words list
inverse_transform (X)	Return terms per document with nonzero entries in X.
set_params (**params)	Set the parameters of this estimator.
transform (raw_documents[, copy])	Transform documents to document-term matrix.

`__init__`(*input=u'content'*, *encoding=u'utf-8'*, *decode_error=u'strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *analyzer=u'word'*, *stop_words=None*, *token_pattern=u'(?u)\\b\\w\\w+\\b'*, *ngram_range=(1, 1)*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=*, *norm=u'l2'*, *use_idf=True*, *smooth_idf=True*, *sublinear_tf=False*)[source]¶

`build_analyzer`()[source]¶

Return a callable that handles preprocessing and tokenization

`build_preprocessor`()[source]¶

Return a function to preprocess the text before tokenization

`build_tokenizer`()[source]¶

Return a function that splits a string into a sequence of tokens

`decode`(*doc*)[source]¶

Decode the input into a string of unicode symbols

The decoding strategy depends on the vectorizer parameters.

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

Learn vocabulary and idf from training set.

raw_documents : iterable

> *an iterable which yields either str, unicode or file objects*

Parameters:

Returns: **self** : TfidfVectorizer

`fit_transform`(*raw_documents*, *y=None*)[source]¶

Learn vocabulary and idf, return term-document matrix.

This is equivalent to fit followed by transform, but more efficiently implemented.

raw_documents : iterable

> *an iterable which yields either str, unicode or file objects*

Parameters:

X : sparse matrix, [n_samples, n_features]

> *Tf-idf-weighted document-term matrix.*

Returns:

`get_feature_names`()[source]¶

Array mapping from feature integer indices to feature name

`get_params`(*deep=True*)[source]¶

Get parameters for this estimator.

deep : boolean, optional

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

Parameters:

params : mapping of string to any

> *Parameter names mapped to their values.*

Returns:

`get_stop_words`()[source]¶

Build or fetch the effective stop words list `inverse_transform`(*X*)[source]¶

Return terms per document with nonzero entries in X.

Parameters: **X** : {array, sparse matrix}, shape = [n_samples, n_features]

X_inv : list of arrays, len = n_samples

> *List of arrays of terms.*

Returns:

`set_params`(***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`(*raw_documents*, *copy=True*)[source]¶

Transform documents to document-term matrix.

Uses the vocabulary and document frequencies (df) learned by fit (or fit_transform).

raw_documents : iterable

> *an iterable which yields either str, unicode or file objects*

copy : boolean, default True

> *Whether to copy X and operate on the copy or perform in-place operations.*

Parameters:

X : sparse matrix, [n_samples, n_features]

> *Tf-idf-weighted document-term matrix.*

Returns: