sklearn.feature_extraction.text.CountVectorizer

class sklearn.feature_extraction.text.CountVectorizer(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *stop_words=None*, *token_pattern='(?u)\b\w\w+\b'*, *ngram_range=(1, 1)*, *analyzer='word'*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=<class 'numpy.int64'>*) [source]

Convert a collection of text documents to a matrix of token counts.

This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.

If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

Read more in the User Guide.

Parameters:

input : *{'filename', 'file', 'content'}, default='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.
- If `'content'`, the input is expected to be a sequence of items that can be of type string or byte.

encoding : *str, default='utf-8'*

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

decode_error : *{'strict', 'ignore', 'replace'}, default='strict'*

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'}, default=None*

Remove accents and perform other character normalization 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.

Both 'ascii' and 'unicode' use NFKD normalization from `unicodedata.normalize`.

lowercase : *bool, default=True*

Convert all characters to lowercase before tokenizing.

preprocessor : *callable, default=None*

Override the preprocessing (strip_accents and lowercase) stage while preserving the tokenizing and n-grams generation steps. Only applies if `analyzer` is not callable.

tokenizer : *callable, default=None*

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

stop_words : *{'english'}, list, default=None*

If 'english', a built-in stop word list for English is used. There are several known issues with 'english' and you should consider an alternative (see Using stop words).

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.

token_pattern : *str, default=r"(?u)\b\w\w+\b"*
Regular expression denoting what constitutes a "token", only used if `analyzer == 'word'`. The default regexp select tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).

If there is a capturing group in token_pattern then the captured group content, not the entire match, becomes the token. At most one capturing group is permitted.

ngram_range : *tuple (min_n, max_n), default=(1, 1)*
The lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted. All values of n such such that min_n <= n <= max_n will be used. For example an `ngram_range` of `(1, 1)` means only unigrams, `(1, 2)` means unigrams and bigrams, and `(2, 2)` means only bigrams. Only applies if `analyzer` is not callable.

analyzer : *{'word', 'char', 'char_wb'} or callable, default='word'*
Whether the feature should be made of word n-gram or character n-grams. Option 'char_wb' creates character n-grams only from text inside word boundaries; n-grams at the edges of words are padded with space.

If a callable is passed it is used to extract the sequence of features out of the raw, unprocessed input.

> *Changed in version 0.21.*

Since v0.21, if `input` is `filename` or `file`, the data is first read from the file and then passed to the given callable analyzer.

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, 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, default=None*
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. Indices in the mapping should not be repeated and should not have any gap between 0 and the largest index.

binary : *bool, default=False*
If True, all non zero counts are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts.

dtype : *type, default=np.int64*
Type of the matrix returned by fit_transform() or transform().

Attributes:

vocabulary_ : *dict*
A mapping of terms to feature indices.

fixed_vocabulary_ : *bool*
ixed vocabulary of term to indices mapping is provided by the user.

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.

See also:

`HashingVectorizer`
Convert a collection of text documents to a matrix of token counts.

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

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.

Examples

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> vectorizer.get_feature_names_out()
array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
       'this'], ...)
>>> print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
>>> vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))
>>> X2 = vectorizer2.fit_transform(corpus)
>>> vectorizer2.get_feature_names_out()
array(['and this', 'document is', 'first document', 'is the', 'is this',
       'second document', 'the first', 'the second', 'the third', 'third one',
       'this document', 'this is', 'this the'], ...)
 >>> print(X2.toarray())
 [[0 0 1 1 0 0 1 0 0 0 0 1 0]
 [0 1 0 1 0 1 0 1 0 0 1 0 0]
 [1 0 0 1 0 0 0 0 1 1 0 1 0]
 [0 0 1 0 1 0 1 0 0 0 0 0 1]]
```

Methods

`build_analyzer`()	Return a callable to process input data.
`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 the input into a string of unicode symbols.

fit(raw_documents[, y])	Learn a vocabulary dictionary of all tokens in the raw documents.
fit_transform(raw_documents[, y])	Learn the vocabulary dictionary and return document-term matrix.
get_feature_names()	DEPRECATED: get_feature_names is deprecated in 1.0 and will be removed in 1.2.
get_feature_names_out([input_features])	Get output feature names for transformation.
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)	Transform documents to document-term matrix.

build_analyzer() [source]

Return a callable to process input data.

The callable handles that handles preprocessing, tokenization, and n-grams generation.

Returns:

analyzer: callable

A function to handle preprocessing, tokenization and n-grams generation.

build_preprocessor() [source]

Return a function to preprocess the text before tokenization.

Returns:

preprocessor: callable

A function to preprocess the text before tokenization.

build_tokenizer() [source]

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

Returns:

tokenizer: callable

A function to split 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.

Parameters:

doc : *bytes or str*

The string to decode.

Returns:

doc: str

of unicode symbols.

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

Learn a vocabulary dictionary of all tokens in the raw documents.

Parameters:

raw_documents : *iterable*
An iterable which generates either str, unicode or file objects.

y : *None*
This parameter is ignored.

Returns:

self : *object*
Fitted vectorizer.

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

Learn the vocabulary dictionary and return document-term matrix.

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

Parameters:

raw_documents : *iterable*
An iterable which generates either str, unicode or file objects.

y : *None*
This parameter is ignored.

Returns:

X : *array of shape (n_samples, n_features)*
Document-term matrix.

get_feature_names() [source]

DEPRECATED: get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.

Array mapping from feature integer indices to feature name.

Returns:

feature_names : *list*
A list of feature names.

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*
d, present here for API consistency by convention.

Returns:

feature_names_out : *ndarray of str objects*

Transformed feature names.

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.

get_stop_words() [source]

Build or fetch the effective stop words list.

Returns:

stop_words: list or None

A list of stop words.

inverse_transform(*X*) [source]

Return terms per document with nonzero entries in X.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*

Document-term matrix.

Returns:

X_inv : *list of arrays of shape (n_samples,)*

List of arrays of terms.

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.

self : *estimator instance*
Estimator instance.

transform(*raw_documents*) [source]

Transform documents to document-term matrix.

Extract token counts out of raw text documents using the vocabulary fitted with fit or the one provided to the constructor.

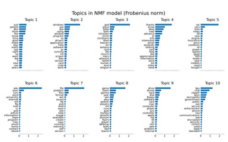
Parameters:

raw_documents : *iterable*
An iterable which generates either str, unicode or file objects.

Returns:

X : *sparse matrix of shape (n_samples, n_features)*
Document-term matrix.

Examples using `sklearn.feature_extraction.text.CountVectorizer`



Topic extraction with Non-negative Matrix Factorization and Latent Dirichlet Allocation

Sample pipeline for text feature extraction and evaluation

Semi-supervised Classification on a Text Dataset