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** : 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’, ‘char_wb’} or callable

Whether the feature should be made of word 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.

»

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 ‘english’, a built-in stop word list for English is used.

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, True by default

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 select 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. Indices in the mapping should not be repeated and should not have any gap between 0 and the largest index.

binary : boolean, 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, optional

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

Attributes:

vocabulary_ : dict

A mapping of terms to feature indices.

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`, `TfidfVectorizer`

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 a vocabulary dictionary of all tokens in the raw documents.
`fit_transform`(raw_documents[, y])	Learn the vocabulary dictionary and 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)	Transform documents to document-term matrix.

»

__init__(*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]

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 a vocabulary dictionary of all tokens in the raw documents.

Parameters: **raw_documents** : iterable

An iterable which yields either str, unicode or file objects.

Returns: **self** :

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

Learn the vocabulary dictionary and return term-document matrix.

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

Parameters: **raw_documents** : iterable

An iterable which yields either str, unicode or file objects.

Returns: **X** : array, [n_samples, n_features]

Document-term matrix.

`get_feature_names`() [source]

Array mapping from feature integer indices to feature name

`get_params`(*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.

`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]

Returns: **X_inv** : list of arrays, len = 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 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*) [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 yields either str, unicode or file objects.

Returns: **X** : sparse matrix, [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