MovieGroupProcess

class **MovieGroupProcess**(n_clusters: int = 8, alpha: float = 0.1, beta: float = 0.1, multiple_occurance: bool=True)

Class for fitting a dirichlet mixture model with the movie group process algorithm described in Yin & Wang's paper (2014).

Source

Hyperparameters

n_clusters: int, *default 8*

Number of mixture components in the model.

alpha: float, *default 0.1*

Willingness of a document joining an empty cluster.

beta: float, *default 0.1*

Willingness to join clusters, where the terms in the document are not present.

multiple_occurance: bool, *default True*

Specifies whether a term should only be counted once in each document.

If set to False, Formula 3 will be used, else Formula 4 will be used.

Attributes

cluster_word_distribution: matrix *of shape (n_clusters, n_vocab)*

Contains the amount a word occurs in a certain cluster.

cluster_doc_count: array *of shape (n_clusters,)*

Array containing how many documents there are in each cluster.

cluster_word_count: array *of shape (n_clusters,)*

Contains the amount of words there are in each cluster.

vectorizer: CountVectorizer

BOW vectorizer from sklearn.

It is used to transform documents into bag of words embeddings.

n_vocab: int

Number of total vocabulary items seen during fitting.

n_documents: int

Total number of documents seen during fitting.

Methods

def **fit**(documents: Iterable[str], n_iterations: int = 30, **vectorizer_kwargs) -> MovieGroupProcess

Fits the model with the MGP algorithm described in Yin and Wang (2014).

Parameters

documents: iterable of str

Stream of documents to fit the model with.

n_iterations: int, *default 30*

Number of iterations used for fitting the model.
Results usually improve with higher number of iterations.

****vectorizer_kwargs**

The rest of the arguments supplied are passed to sklearn's CountVectorizer.

For a detailed list of arguments consult the documentation

Returns

MovieGroupProcess

The same model fitted.

def **transform**(embeddings: scipy.sparse.csr_matrix) -> np.ndarray

Predicts mixture component labels from BOW representations
of the provided documents produced by self.vectorizer.
This function is mostly here for sklearn compatibility.

Parameters

embedings: sparse array *of shape (n_documents, n_vocab)*

BOW embeddings of documents

Returns

array *of shape (n_documents, n_clusters)*

Matrix of probabilities of documents belonging to each cluster.

def **predict**(documents: Iterable[str]) -> np.ndarray

Predicts mixture component labels for the given documents.

Parameters

documents: iterable of str

Stream of text documents.

Returns

array *of shape (n_documents, n_clusters)*

Matrix of probabilities of documents belonging to each cluster.

def **top_words**(top_n: int) -> List[Dict[str, int]]

Calculates the top words for each cluster.

Parameters

top_n: int, *default 10*

Top N words to return. If None, all words are returned.

Returns

list of dict of str to int

Dictionary for each cluster mapping the words to number of occurances.

def **most_important_words**(top_n: int) -> List[Dict[str, int]]

Calculates the most important words for each cluster, where

```
importance = n_occurances_in_component/(log(n_occurances_in_corpus) + 1)
```

Parameters

top_n: int, *default 10*

Top N words to return. If None, all words are returned.

Returns

list of dict of str to int

Dictionary for each cluster mapping the words to number of occurances.

def **visualize**()

Visualizes the model with pyLDAvis for inspection of the different
mixture components :)

Returns

pyLDAvis graph

Interactive visualization of the clusters