adso

Release 0.0.1

Michele Ciruzzi

Dec 09, 2020

CONTENTS

A topic modelling library built on top of scipy/numpy and nltk.

To install

```
pip install adso
```

adso need to write some files to disk. As default adso uses the `~/.adso` folder, but it can be change setting the enviromental variable `ADSODIR`.

Some examples on how to use adso are available in `tests` and `examples` folders.

CHAPTER

ONE

DOCUMENTATION

1.1 adso

Adso is a topic-modelling library based on numpy and nltk.

Contents

1.1.1 common

adso.**set_seed**(*seed*)

Set random seed for reproducibility.

Take care of python random library and numpy.

Parameters **seed** (*int*) – the value choosen as seed for the random generators

Return type None

1.2 adso.data

Datasets and datased related functions.

Contents

1.2.1 common

adso.data.**load_txt**(*path*, *lines=False*, *label=False*, *extension='txt'*, *encoding='utf-8'*, *ignore_errors=True*)

Load text files as dataset.

Load a dataset composed by text files. This function scan the given directory and all its subdirectories and read all the files in them. The single items for the dataset could be either the files or each lines in each files. In the first case the folder's name will be used as label, in the latter the file's name.

Parameters

- **path** (*Union[str, bytes, os.PathLike]*) – the directory to be scanned for the datatset.
- **lines** (*bool, optional*) – if each item of the datatset is a different file (False) or each line in it (True). Defaults to False.
- **label** (*bool, optional*) – to load the files as a Dataset (False) or a LabelledDataset, i.e. a dataset with labelled data (True). The folder names (if lines=False) or the file names (if lines=False) are used as labels. Defaults to False.
- **extension** (*Union[str, None], optional*) – the extension of the file to be read. If None read all files. Defaults to "txt".
- **encoding** (*str, optional*) – encoding for the files, passed to pathlib.Path.read_text(). Defaults to "utf-8".
- **ignore_errors** (*bool, optional*) – whenever ignore unreadable characters in files or raise an error. Defaults to True.

Returns return a Dataset, as defined in dataset.py file

Return type *Dataset*

1.2.2 Dataset

class adso.data.**Dataset**(*data*)

Bases: object

Dataset class.

__init__(*data*)

Constructor for Dataset class.

Parameters **data** (*List[str]*) – A list of string, each will be considered as one document.

get_data()

Access data stored into the dataset.

Returns Return a list of string, each one is a document

Return type List[str]

class adso.data.**LabelledDataset**(*data*)

Bases: adso.data.dataset.Dataset

Labbelled Dataset class.

A subclass of *Dataset* wich store also labels for documents. Documents and labels are stored in two different lists where each pair share the index.

__init__(*data*)
Contructor for LabelledDataset class.

Parameters **data** (*List[Tuple[str, str]]*) – list of (document, label) tuples of strings.

get_data()
Access data stored into the dataset.

Returns Return a list of string, each one is a document

Return type List[str]

get_labels()
Access labels stored into the dataset.

Returns Return a list of string, each one is a label

Return type List[str]

toDataset()
Convert a labelledDataset to Dataset.

Returns a Dataset with the same documents list but without labels.

Return type *Dataset*

1.2.3 Sample Datasets

adso.data.**load_test_dataset**(*lines=False*)
Load the very simple test dataset as *Dataset*.

Parameters **lines** (*bool, optional*) – consider each line as a document rather than each file. Defaults to False.

Returns the very simple test dataset

Return type *Dataset*

adso.data.**load_labelled_test_dataset**(*lines=False*)
Load the very simple test dataset as *LabelledDataset*.

Parameters **lines** (*bool, optional*) – consider each line as a document rather than each file. Defaults to False.

Returns the very simple test dataset with labels

Return type *LabelledDataset*

adso.data.**load_20newsgroups**(*split='all'*)
Load the 20newsgroups dataset with labels from <http://qwone.com/~jason/20Newsgroups/>`

Parameters **split** (*str, optional*) – which part of the dataset have to be load:

- ”all” for a single *LabelledDataset* with all the available documents
- ”train” for the train split only (deterministic)
- ”test” for the test split only (deterministic)
- ”both” for the tuple (train, test)

Defaults to “all”.

Returns the 20newsgroups dataset with labels, as LabelledDataset

Return type Union[*LabelledDataset*, Tuple[*LabelledDataset*, *LabelledDataset*]]

1.3 adso.transform

Trasform string tensor to numerical.

Contents

class adso.transform.common.**Transformer**
Bases: abc.ABC

Abstract class for data manipulation class.

__init__()
Initialize self. See help(type(self)) for accurate signature.

abstract fit()
Find the parameters necessary to trasform the data.

Parameters **data** (*[type]*) – input data

abstract fit_transform()
Combine fit() and transform().

This is to be preferred to calling fit and transform subsequently.

Parameters **data** (*[type]*) – input data

abstract transform()
Transform the data.

Parameters **data** (*[type]*) – input data

1.3.1 common

adso.transform.**nltk_download**(*id*)
Wrapper for nltk.downloader.download(), to download nltk data inside adso directory.

Parameters **id** (*str*) – id of the required downloadable data. Cfr <http://www.nltk.org/nltk_data/>

1.3.2 Tokenizer

class adso.transform.**Tokenizer**(*tokenizer=<function word_tokenize>*, *stemmer=<function _lowercase>*, *stopwords=None*)

Bases: *adso.transform.common.Transformer*

Transform each document in a corpus in a list of tokens.

__init__(*tokenizer=<function word_tokenize>*, *stemmer=<function _lowercase>*, *stopwords=None*)

Initialize the Tokenizer, specifing callable and parameters to be used.

Parameters

- **tokenizer** (*callable, optional*) – any function which maps a string into a list of strings. Defaults to `nltk.tokenize.word_tokenize()`.
- **stemmer** (*Union[None, callable], optional*) – any function which maps a string into a string, for example a stemmer. Applied after the tokenizer. Defaults to _lowercase.
- **stopwords** (*Union[None, List[str]], optional*) – list of string to be ignored. Removed after calling the stemmer. Defaults to None.

fit(*data*)

Do nothing method.

Return type None

fit_transform(*data*)

Alias for `transform()`.

Parameters **data** (Dataset) – a `data.Dataset` to be tokenized

Returns

a list of list of tokens. Outer list map to documents while inner list map to words in each document.

Return type List[List[str]]

transform(*data*)

Tokenize a given Dataset.

Parameters **data** (Dataset) – a `data.Dataset` to be tokenized

Returns

a list of list of tokens. Outer list map to documents while inner list map to words in each document.

Return type List[List[str]]

1.3.3 Vocab

class adso.transform.**Vocab**(*count*, *max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*)

Bases: `object`

Vocabulary class, used to convert string to int and viceversa.

__init__(*count*, *max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*)

Construct the vocabulary from a `collections.Counter` object with word count.

Parameters

- **count** (*Counter*) – a `collections.Counter` which store the list of words and their count, with which build the vocabulary.
- **max_size** (*Union[None, int], optional*) – maximum number of word to store into the vocabulary. Most common words are kept. None to keep all words. Defaults to None.
- **min_freq** (*float, optional*) – minimum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 0.
- **max_freq** (*float, optional*) – maximum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 1.
- **min_count** (*int, optional*) – minimum number of occurency (i.e. associated value in Counter object) to keep a word. Defaults to 1.

itos
dictionary to access words in vocabulary given the index.
Type Dict[int, str]

stoi
dictionary to access the index of a given word in vocabulary.
Type Dict[str, int]

1.3.4 Vectorizer

class adso.transform.vectorizer.**Vectorizer**
Bases: *adso.transform.common.Transformer*

Abstract class to group vectorizer, i.e. text to number processors.

__init__()
Initialize self. See help(type(self)) for accurate signature.

abstract fit()
Create the vocabulary.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Return type None

abstract fit_transform()
Trasform the collection of tokens in a sequence of numerical values, creating a vocabulary in the process.

Args:

data (List[List[str]]): a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns:

Union[np.array, sp.sparse.spmatrix]: a matrix which maps documents to the rows.

Return type Union[array, spmatrix]

abstract transform()
Trasform the collection of tokens in a sequence of numerical values, given an already fitted vocabulary.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns

a matrix which maps documents to the rows.

Return type Union[np.array, sp.sparse.spmatrix]

CountVectorizer

class adso.transform.**CountVectorizer**(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*)

Bases: *adso.transform.vectorizer.Vectorizer*

Create a document-term matrix with count as values.

__init__(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*)

Initialize the vectorizer and set the parameters.

Parameters

- **max_size** (*Union[None, int], optional*) – maximum number of word to store into the vocabulary. Most common words are kept. None to keep all words. Defaults to None.
- **min_freq** (*float, optional*) – minimum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 0.
- **max_freq** (*float, optional*) – maximum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 1.
- **min_count** (*int, optional*) – minimum number of occurency to keep a word. Defaults to 1.

vocab
the stored vocabulary used to transform the data.

fit(*data*)

Create the vocabulary.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Return type None

fit_transform(*data*)

Return a document-term matrix with count as values. Vocabulary is built on the data passed as input.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns a document-term matrix with count as values.

Return type sp.sparse.spmatrix

transform(*data*)

Return a document-term matrix with count as values. Words not in vocabulary are discarded.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns a document-term matrix with count as values.

Return type sp.sparse.spmatrix

ListVectorizer

class adso.transform.**ListVectorizer**(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*, *max_length=100*)

Bases: `adso.transform.vectorizer.CountVectorizer`

Create a matrix with the ordered list of indeces for each document on the rows.

__init__(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*, *max_length=100*)

Initialize the vectorizer and set the parameters.

Parameters

- **max_size** (*Union[None, int], optional*) – maximum number of word to store into the vocabulary. Most common words are kept. None to keep all words. Defaults to None.
- **min_freq** (*float, optional*) – minimum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 0.
- **max_freq** (*float, optional*) – maximum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 1.
- **min_count** (*int, optional*) – minimum number of occurency to keep a word. Defaults to 1.
- **max_length** (*int, optional*) – maximum lenght of each row, i.e. maximum number of words for each document to keep. The first N words are kept. Defaults to 100.

vocab

the stored vocabulary used to transform the data.

fit(*data*)

Create the vocabulary.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Return type `None`

fit_transform(*data*)

Return a matrix with the ordered list of indeces for each document on the rows. Vocabulary is built on the data passed as input.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns a matrix with the ordered list of indeces for each document on the rows.

Return type np.array

transform(*data*)
Return a matrix with the ordered list of indeces for each document on the rows. Words not in vocabulary are discarded.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns a matrix with the ordered list of indeces for each document on the rows.

Return type np.array

FreqVectorizer

class adso.transform.**FreqVectorizer**(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*)
Bases: `adso.transform.vectorizer.CountVectorizer`

Create a document-term matrix with frequencies as values.

__init__(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*)
Initialize the vectorizer and set the parameters.

Parameters

- **max_size** (*Union[None, int], optional*) – maximum number of word to store into the vocabulary. Most common words are kept. None to keep all words. Defaults to None.
- **min_freq** (*float, optional*) – minimum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 0.
- **max_freq** (*float, optional*) – maximum frequency (as count of word / sum of counts of all words) to keep a word. Defaults to 1.
- **min_count** (*int, optional*) – minimum number of occurency to keep a word. Defaults to 1.

vocab
the stored vocabulary used to transform the data.

fit(*data*)
Create the vocabulary.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Return type None

fit_transform(*data*)
Return a document-term matrix with frequencies as values. Vocabulary is built on the data passed as input.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns a document-term matrix with frequencies as values.

Return type sp.sparse.spmatrix

transform(*data*)
Return a document-term matrix with frequencies as values. Words not in vocabulary are discarded.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns a document-term matrix with frequencies as values.

Return type sp.sparse.spmatrix

TFIDFVectorizer

class adso.transform.**TFIDFVectorizer**(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*, *smooth=False*, *log_df=False*)

Bases: `adso.transform.vectorizer.CountVectorizer`

Create a document-term matrix with TFIDF frequencies as values.

__init__(*max_size=None*, *min_freq=0*, *max_freq=1*, *min_count=1*, *smooth=False*, *log_df=False*)
[summary]

Parameters

- **self** (TFIDFVectorizer) – [description]
- **max_size** (*Union[None, int], optional*) – [description]. Defaults to None.
- **min_freq** (*float, optional*) – [description]. Defaults to 0.
- **max_freq** (*float, optional*) – [description]. Defaults to 1.
- **smooth** (*bool, optional*) – [description]. Defaults to False.
- **log_df** (*bool, optional*) – [description]. Defaults to False.

fit(*data*)
Create the vocabulary.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Return type None

fit_transform(*data*)
Return a document-term matrix with TFIDF frequencies as values. Vocabulary is built on the data passed as input.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns

a document-term matrix with TFIDF frequencies as values.

Return type sp.sparse.spmatrix

transform(*data*)
Return a document-term matrix with TFIDF frequencies as values. Words not in vocabulary are discarded.

Parameters **data** (*List[List[str]]*) – a list of list of tokens. Outer list map to documents while inner list map to words in each document. For example the output of a `transform.Tokenizer`.

Returns

a document-term matrix with TFIDF frequencies as values.

Return type sp.sparse.spmatrix

1.4 adso.topicmodel

Topic modelling algorithms.

Contents

class adso.topicmodel.common.**TopicModel**

Bases: abc.ABC

Abstract class for topic modelling algorithms.

__init__()

Initialize self. See help(type(self)) for accurate signature.

abstract fit(*data*)

Find all the relevant parameters to get the distribution of the topic among the documents.

Parameters **data** – input data, genrally a matrix of terms and documents.

abstract fit_transform(*data*)

Retrieve the distribution of the topic among the documents, computing the parameters at the same time.

This is to be preferred to calling `fit()` and `transform()` subsequently.

Parameters **data** – input data, genrally a matrix of terms and documents.

abstract transform(*data*)

Retrieve the distribution of the topic among the documents from prevously computed parameters.

Parameters **data** – input data, genrally a matrix of terms and documents.

1.4.1 NMF

class adso.topicmodel.**NMF**(*n_topic=10*, *max_iter=200*, *tolerance=0*, *lambdaW=0.5*, *lambdaH=0.5*, *alphaW=0.5*, *alphaH=0.5*, *method='ACLS'*)

Bases: *adso.topicmodel.common.TopicModel*

Nonnegative Matrix Factorization algorithm.

Implemented following Langville et al. (ArXiv:1407.7299) using Alternate Least Squares.

__init__(*n_topic=10*, *max_iter=200*, *tolerance=0*, *lambdaW=0.5*, *lambdaH=0.5*, *alphaW=0.5*, *alphaH=0.5*, *method='ACLS'*)

Initialize NMF algorithm specifing parameters.

Parameters

- **n_topic** (*int, optional*) – number of topic in corpus to estimate. Defaults to 10.

- **max_iter** (*int, optional*) – maximum number of iteration for the algorithm. Defaults to 200.
- **tolerance** (*float, optional*) – maximum relative error between iteration to early stop the algorithm. Defaults to 0.
- **lambdaW** (*float, optional*) – smoothing parameter for W matrix (document-topic). Used in both ACLS and AHCLS variant. Must be positive. Defaults to 0.5.
- **lambdaH** (*float, optional*) – smoothing parameter for H matrix (topic-word). Used in both ACLS and AHCLS variant. Must be positive. Defaults to 0.5.
- **alphaW** (*float, optional*) – sparsity parameter for W matrix in AHCLS algorithm. Must be in (0,1). Defaults to 0.5.
- **alphaH** (*float, optional*) – sparsity parameter for H matrix in AHCLS algorithm. Must be in (0,1). Defaults to 0.5.
- **method** (*str, optional*) – the desired algoritm. One of ["ALS", "ACLS", "AHCLS"]. Defaults to "ACLS".

fit(*data*)

Computate the H matrix (topic-document).

Equivalent to call `fit_transform()` method, which should be used instead.

Parameters **data** (*sp.sparse.spmatrix*) – a document-term matrix for example the output of a subclass of `transform.vectorizer.Vectorizer`.

Returns topic-word H matrix.

Return type sp.sparse.spmatrix

fit_transform(*data*)

Decompose a document-word matrix in two nonnegative document-topic and topic word matrix.

Parameters **data** (*sp.sparse.spmatrix*) – a document-term matrix for example the output of a subclass of `transform.vectorizer.Vectorizer`.

Returns document-topic W matrix, topic-word H matrix and the number of iterations.

Return type Tuple[sp.sparse.spmatrix, sp.sparse.spmatrix, int]

transform(*data*)

Estimate a topic-document matrix given the stored topic-word matrix.

Parameters **data** (*sp.sparse.spmatrix*) – a document-term matrix for example the output of a subclass of `transform.vectorizer.Vectorizer`. Must be of the same type of the one used to fit the model.

Returns topic-document W matrix.

Return type sp.sparse.spmatrix

1.4.2 LDA

class adso.topicmodel.**LDA**(*n_topic=10*, *max_iter=200*, *tolerance=0*, *epsilon=1e-50*)
Bases: *adso.topicmodel.common.TopicModel*

Latent Dirichlet Allocation.

Implemented following Blei, Jordan, Ng 2003.

__init__(*n_topic=10*, *max_iter=200*, *tolerance=0*, *epsilon=1e-50*)
Initialize the algorithm.

Parameters

- **n_topic** (*int, optional*) – number of topics in copus to be estimated. Defaults to 10.
- **max_iter** (*int, optional*) – max number of iterations for iterative steps. Defaults to 200.
- **tolerance** (*float, optional*) – target relative error for early stopping in iterative steps. Defaults to 0.
- **epsilon** (*float, optional*) – small number to avoid -inf in logarithm result. Defaults to 1e-50.

fit(*data*)
Estimate LDA parameters, particularly alpha e beta.

Parameters **data** (*sp.sparse.spmatrix*) – document-term matrix with counts as entry. For example the output of CountVectorizer

Returns

alpha (np.array): Dirichlet priors for LDA model (also stored as instance attributes)

beta (np.array): Multinomial priors for LDA model (also stored as instance attributes)

gamma (np.array): Dirichlet priors for auxiliary model

phi (np.array): Multinomial priors for auxiliary model

likelihood (float): achieved likelihood

iter (int): number of iteration of EM algorithm

Return type Tuple[np.array, np.array, np.array, np.array, float, int]

fit_transform(*data*)
Estimate the probability of each topic in each document.

Perform the estimation of both model's parameters (`fit()`) and topic-document relation (`transform()`).

Parameters **data** (*sp.sparse.spmatrix*) – document-term matrix with counts as entry. For example the output of CountVectorizer

Returns

estimation (sp.sparse.spmatrix): the probability of each document (rows) given the different topics (columns).

alpha (np.array): Dirichlet priors for LDA model (also stored as instance attributes)

beta (np.array): Multinomial priors for LDA model (also stored as instance attributes)

gamma (np.array): Dirichlet priors for auxiliary model

phi (np.array): Multinomial priors for auxiliary model

likelihood (float): achieved likelihood

iter (int): number of iteration of EM algorithm

Return type Tuple[sp.sparse.matrix, np.array, np.array, np.array, np.array, float, int]

transform(*data*)

Estimate the probability of each topic in each document, given the model.

Must be called after a `fit()` method on the same instance.

Parameters **data** (*sp.sparse.spmatrix*) – document-term matrix with counts as entry. For example the output of `CountVectorizer`

Returns

the probability of each document (rows) given the different topics (columns).

Return type sp.sparse.spmatrix

CHAPTER

TWO

EXAMPLES

Some examples on how to use adso package.

2.1 Analyze a very simple dataset with LDA

import

```
import adso
import matplotlib.pyplot as plt
import nltk
import numpy as np
```

set seed

```
adso.set_seed(1234)
```

Download the dataset and select 1000 random elements

```
data = adso.data.load_labelled_test_dataset(lines=True)

print("Number of documents: ", len(data))
```

Out:

```
Number of documents:  12
```

Tokenize the dataset using a stemmer and a stopwords list, removing punctation

```
adso.transform.nltk_download("stopwords")

snowball = nltk.stem.snowball.SnowballStemmer("english")


def stemmer(word):
    ret = snowball.stem(word)
    if ret.isalpha():
        return ret
    else:
        return None


tokenizer = adso.transform.Tokenizer(
```

(continues on next page)

(continued from previous page)

```
    stemmer=stemmer,
    stopwords=nltk.corpus.stopwords.words("english") + [None],
)

tokens = tokenizer.fit_transform(data)

print("First ten tokens of the first document:")
print(tokens[0][:10])
```

Out:

```
[nltk_data] Downloading package stopwords to /home/tnto/.adso/NLTK...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /home/tnto/.adso/NLTK...
[nltk_data]   Package punkt is already up-to-date!
First ten tokens of the first document:
['linear', 'algebra', 'studi', 'matric', 'vector', 'vectori', 'space']
```

Transform the list of tokens in a list of numbers. We will use the absolute frequency.

```
vectorizer = adso.transform.CountVectorizer()
```

Generate the vocabulary.

```
vectorizer.fit(tokens)
vocab = vectorizer.vocab

print("Number of words in vocabulary: ", len(vocab))

print("index of word 'bird': ", vocab["bird"])
print("word at index 1: ", vocab[1])
```

Out:

```
Number of words in vocabulary:  50
index of word 'bird':  1
word at index 1:  bird
```

Create the count matrices from tokens.

```
count_matrix = vectorizer.transform(tokens)
```

LDA

```
LDA = adso.topicmodel.LDA(n_topic=4, tolerance=1e-3, max_iter=200)
ret = LDA.fit_transform(count_matrix)
estimation = ret[0]
beta = ret[2]
print("LDA ended after", ret[6], "iterations, achiving a loglikelihood of", ret[5])
```

Out:

```
Iteration 1 Log-Likelihood -497.4960522897091
Iteration 2 Log-Likelihood -490.3637651912628
Iteration 3 Log-Likelihood -488.2187675212483
Iteration 4 Log-Likelihood -487.4344857012694
```

(continues on next page)

(continued from previous page)

```
Iteration 5 Log-Likelihood -487.2205134256087
LDA ended after 5 iterations, achiving a loglikelihood of -487.2205134256087
```

Check the 10 most characteristic words for each topic

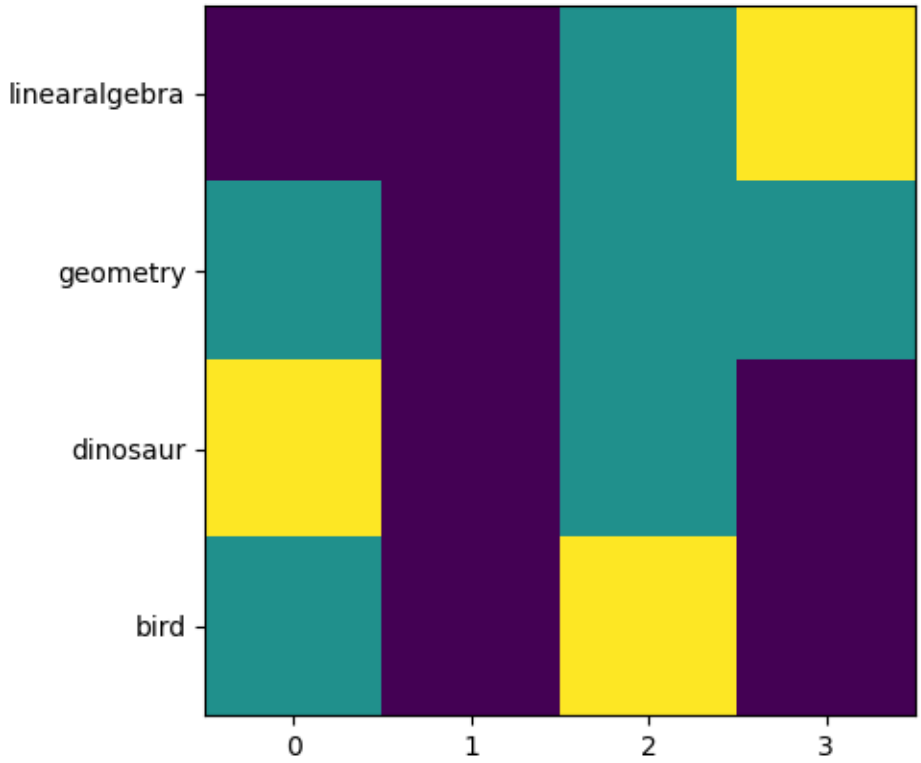
```
for i in range(4):
    print("10 most characteristic words of topic", i)
    print(
        list(
            map(
                lambda j: vocab[j],
                np.argsort(np.squeeze(-beta[i, :].toarray()))[:10].tolist(),
            )
        )
    )
```

Out:

```
10 most characteristic words of topic 0
['dinosaur', 'prove', 'subfield', 'like', 'entiti', 'relat', 'ancient', 'concept',
↪'wide', 'probabl']
10 most characteristic words of topic 1
['reptil', 'fli', 'space', 'matric', 'wing', 'among', 'wide', 'vector', 'mani',
↪'probabl']
10 most characteristic words of topic 2
['bird', 'studi', 'vectori', 'sometim', 'egg', 'fli', 'matric', 'mani', 'descend',
↪'lay']
10 most characteristic words of topic 3
['linear', 'theorem', 'one', 'entiti', 'like', 'use', 'calculus', 'lay', 'vector',
↪'descend']
```

Print the confusion matrix (not diagonalized)

```
print(beta.todense())
print(estimation.todense())

predicted_topic = np.argmax(estimation, axis=1)

listvectorizer = adso.transform.ListVectorizer()
labels = list(map(lambda l: [l], data.get_labels()))

label_topic = np.squeeze(listvectorizer.fit_transform(labels))

confusion = np.zeros((4, 4))
for i in zip(label_topic, predicted_topic):
    confusion[i] += 1

fig, ax = plt.subplots()
ax.imshow(confusion)
ax.set_xticks(np.arange(4))
ax.set_yticks(np.arange(4))
ax.set_yticklabels(list(listvectorizer.vocab.stoi.keys()))
```



Out:

```
[[0.34291811 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.07824034 0.         0.         0.         0.07824034 0.02564681
  0.         0.         0.         0.08572953 0.         0.
  0.         0.         0.         0.         0.         0.08572953
  0.0665698  0.         0.         0.0665698  0.03985091 0.
  0.         0.02494121 0.03899384 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.0665698 ]
 [0.         0.         0.237522   0.17455686 0.         0.
  0.         0.10275189 0.158348   0.         0.         0.
  0.         0.         0.         0.         0.         0.02726827
  0.         0.         0.         0.         0.02502113 0.
  0.         0.         0.         0.         0.         0.
  0.01769463 0.079174   0.         0.01769463 0.04237039 0.
  0.         0.         0.02072957 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.079174   0.01769463]
 [0.         0.32315953 0.         0.06425021 0.         0.
  0.16157976 0.05673078 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.08078988 0.08078988 0.         0.         0.05525809 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

(continues on next page)

(continued from previous page)

```
  0.01674266 0.02928199 0.02289018 0.         0.         0.
  0.02773715 0.08078988 0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.17656598 0.
  0.         0.         0.         0.         0.17656598 0.17656598
  0.09599524 0.         0.         0.         0.09599524 0.03146679
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.08828299
  0.06998748 0.03060106 0.         0.         0.         0.
  0.05797327 0.         0.         0.         0.         0.
  0.         0.        ]]
[[2.56468127e-302 4.43669983e-204 7.40564159e-204 5.55596415e-253]
 [8.57295264e-252 1.00000000e-300 8.07898813e-252 1.76565976e-251]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 5.50454036e-303]
 [7.82403359e-202 1.58347999e-201 1.61579763e-201 9.59952449e-202]
 [4.46517388e-254 1.40095463e-303 0.00000000e+000 1.69494941e-302]
 [2.65286704e-203 7.49728398e-204 1.67426610e-252 6.17870364e-203]
 [3.42918106e-051 2.37521998e-051 1.00000000e-100 1.00000000e-100]
 [8.55279182e-053 1.00000000e-150 9.46275499e-053 3.06010621e-102]
 [1.33716946e-252 3.61848840e-253 1.47069887e-253 0.00000000e+000]
 [7.82403359e-252 2.37521998e-251 7.24162174e-154 5.56515784e-203]
 [0.00000000e+000 3.04700961e-252 1.33403156e-203 0.00000000e+000]
 [1.78606955e-203 3.32757543e-204 3.23159525e-301 9.59952449e-302]]

[Text(0, 0, 'linearalgebra'), Text(0, 1, 'geometry'), Text(0, 2, 'dinosaur'), Text(0,␣
↪3, 'bird')]
```

Total running time of the script: (0 minutes 16.377 seconds)

2.2 Analyze the 20newsgroups dataset with LDA

import

```
import random

import adso
import matplotlib.pyplot as plt
import nltk
import numpy as np
```

set seed

```
adso.set_seed(1234)
```

Download the dataset and select 1000 random elements

```
data = adso.data.load_20newsgroups(split="test")

new_data = []
for i in random.sample(range(len(data)), k=1000):
    new_data.append(data[i])
data = adso.data.LabelledDataset(new_data)

print("Number of documents: ", len(data))
```

Out:

```
Number of documents:  1000
```

Tokenize the dataset using a stemmer and a stopwords list, removing punctation

```
adso.transform.nltk_download("stopwords")

snowball = nltk.stem.snowball.SnowballStemmer("english")


def stemmer(word):
    ret = snowball.stem(word)
    if ret.isalpha():
        return ret
    else:
        return None


tokenizer = adso.transform.Tokenizer(
    stemmer=stemmer,
    stopwords=nltk.corpus.stopwords.words("english") + [None],
)

tokens = tokenizer.fit_transform(data)

print("First ten tokens of the first document:")
print(tokens[0][:10])
```

Out:

```
[nltk_data] Downloading package stopwords to /home/tnto/.adso/NLTK...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /home/tnto/.adso/NLTK...
[nltk_data]   Package punkt is already up-to-date!
First ten tokens of the first document:
['rtaraz', 'ramin', 'taraz', 'subject', 'wing', 'ding', 'organ', 'worcest', 'polytechn
↪', 'institut']
```

Transform the list of tokens in a list of numbers. We will use the absolute frequency.

```
vectorizer = adso.transform.CountVectorizer(max_freq=0.7, min_freq=0.1, max_size=1000)
```

Generate the vocabulary.

```
vectorizer.fit(tokens)
vocab = vectorizer.vocab

print("Number of words in vocabulary: ", len(vocab))

print("index of word 'god': ", vocab["god"])
print("word at index 52: ", vocab[52])
```

Out:

```
Number of words in vocabulary:  1000
index of word 'god':  52
word at index 52:  god
```

Create the count matrices from tokens.

```
count_matrix = vectorizer.transform(tokens)
```

LDA

```
LDA = adso.topicmodel.LDA(n_topic=20, tolerance=0.001, max_iter=100)
ret = LDA.fit_transform(count_matrix)
estimation = ret[0]
beta = ret[2]
print("LDA ended after", ret[6], "iterations, achiving a loglikelihood of", ret[5])
```

Out:

```
Iteration 1 Log-Likelihood -2165701.2114364556
Iteration 2 Log-Likelihood -2160482.3346176264
Iteration 3 Log-Likelihood -2163212.9631883884
Iteration 4 Log-Likelihood -2168112.996969577
Iteration 5 Log-Likelihood -2173911.817309296
Iteration 6 Log-Likelihood -2180224.827983113
Iteration 7 Log-Likelihood -2186886.8326180084
Iteration 8 Log-Likelihood -2193803.660600017
Iteration 9 Log-Likelihood -2200911.3998690704
Iteration 10 Log-Likelihood -2208162.4613385475
Iteration 11 Log-Likelihood -2215519.673691593
Iteration 12 Log-Likelihood -2222953.241613101
Iteration 13 Log-Likelihood -2230438.912128744
Iteration 14 Log-Likelihood -2237956.734111518
Iteration 15 Log-Likelihood -2245490.1476548053
Iteration 16 Log-Likelihood -2253025.2760382164
Iteration 17 Log-Likelihood -2260550.353173516
Iteration 18 Log-Likelihood -2268055.249179338
Iteration 19 Log-Likelihood -2275531.072524249
Iteration 20 Log-Likelihood -2282969.8357664314
Iteration 21 Log-Likelihood -2290364.1765527423
Iteration 22 Log-Likelihood -2297707.127931875
Iteration 23 Log-Likelihood -2304991.933236058
Iteration 24 Log-Likelihood -2312211.901354372
Iteration 25 Log-Likelihood -2319360.2985252636
Iteration 26 Log-Likelihood -2326430.272993087
Iteration 27 Log-Likelihood -2333414.8090984584
Iteration 28 Log-Likelihood -2340306.7076238645
Iteration 29 Log-Likelihood -2347098.589511253
Iteration 30 Log-Likelihood -2353782.9203738156
Iteration 31 Log-Likelihood -2360352.0535009955
Iteration 32 Log-Likelihood -2366798.2892966587
Iteration 33 Log-Likelihood -2373113.9492324963
Iteration 34 Log-Likelihood -2379291.4624240845
Iteration 35 Log-Likelihood -2385323.4628397017
Iteration 36 Log-Likelihood -2391202.8949087705
Iteration 37 Log-Likelihood -2396923.124929807
Iteration 38 Log-Likelihood -2402478.0552294045
Iteration 39 Log-Likelihood -2407862.237542648
Iteration 40 Log-Likelihood -2413070.981643885
Iteration 41 Log-Likelihood -2418100.4549575425
Iteration 42 Log-Likelihood -2422947.7687891275
Iteration 43 Log-Likelihood -2427611.047023004
Iteration 44 Log-Likelihood -2432089.4736624938
```

(continues on next page)

(continued from previous page)

```
Iteration 45 Log-Likelihood -2436383.316426467
Iteration 46 Log-Likelihood -2440493.9247364057
Iteration 47 Log-Likelihood -2444423.7016874366
Iteration 48 Log-Likelihood -2448176.0509292097
Iteration 49 Log-Likelihood -2451755.3006263403
Iteration 50 Log-Likelihood -2455166.607739106
Iteration 51 Log-Likelihood -2458415.8466764167
Iteration 52 Log-Likelihood -2461509.486902886
Iteration 53 Log-Likelihood -2464454.464287028
Iteration 54 Log-Likelihood -2467258.050912735
Iteration 55 Log-Likelihood -2469927.7277550315
Iteration 56 Log-Likelihood -2472471.064093511
Iteration 57 Log-Likelihood -2474895.606878144
LDA ended after 57 iterations, achiving a loglikelihood of -2474895.606878144
```

Check the 10 most characteristic words for each topic

```
for i in range(20):
    print("10 most characteristic words of topic", i)
    print(
        list(
            map(
                lambda j: vocab[j],
                np.argsort(np.squeeze(-beta[i, :].toarray()))[:10].tolist(),
            )
        )
    )
```

Out:

```
10 most characteristic words of topic 0
['becaus', 'thing', 'problem', 'look', 'better', 'think', 'best', 'ever', 'name',
↪'real']
10 most characteristic words of topic 1
['use', 'write', 'may', 'first', 'come', 'chang', 'least', 'w', 'wrote', 'mark']
10 most characteristic words of topic 2
['one', 'game', 'even', 'post', 'said', 'usa', 'god', 'ask', 'might', 'opinion']
10 most characteristic words of topic 3
['mean', 'b', 'long', 'nation', 'h', 'anyth', 'littl', 'stand', 'insid', 'goe']
10 most characteristic words of topic 4
['articl', 'ani', 'line', 'could', 'subject', 'way', 'organ', 'state', 'back', 'help']
10 most characteristic words of topic 5
['also', 'new', 'inform', 'thank', 'f', 'noth', 'idea', 'man', 'object', 'exist']
10 most characteristic words of topic 6
['would', 'whi', 'imag', 'live', 'tell', 'day', 'distribut', 'think', 'still',
↪'version']
10 most characteristic words of topic 7
['interest', 'etc', 'mean', 'littl', 'start', 'reason', 'human', 'term', 'comput',
↪'third']
10 most characteristic words of topic 8
['q', 'must', 'becom', 'na', 'comput', 'across', 'yes', 'happen', 'anyon', 'ca']
10 most characteristic words of topic 9
['well', 'reason', 'someth', 'follow', 'question', 'find', 'start', 'happen',
↪'stephanopoulo', 'true']
10 most characteristic words of topic 10
['would', 'onli', 'doe', 'support', 'group', 'cours', 'like', 'kill', 'work', 'applic
↪']
```

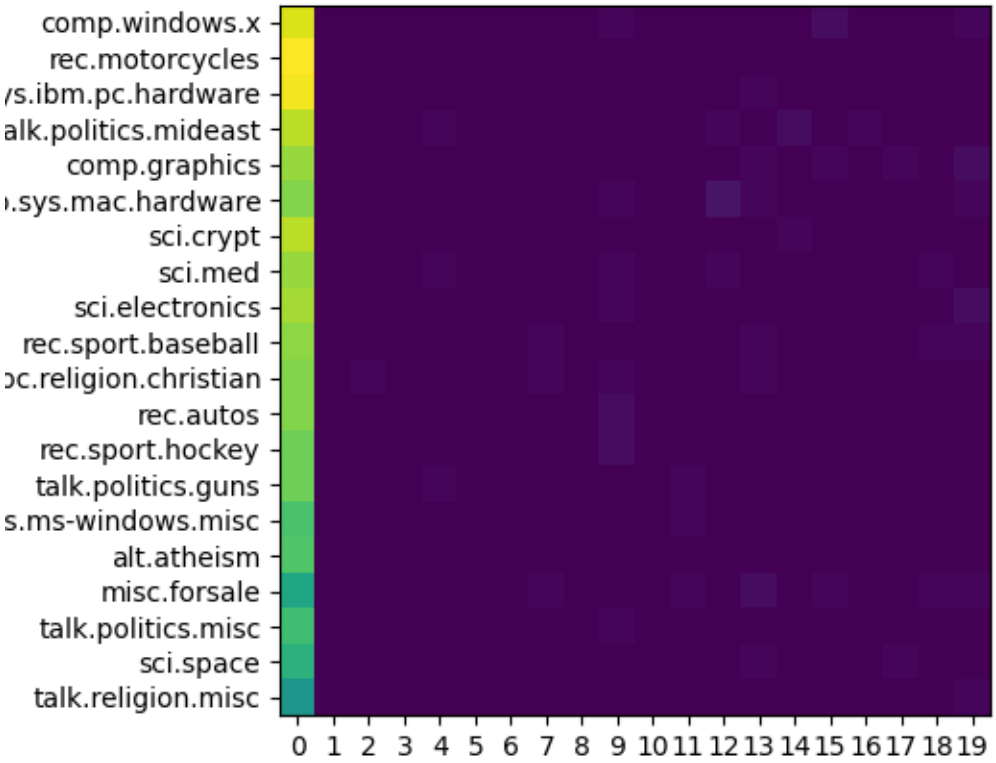
(continues on next page)

(continued from previous page)

```
10 most characteristic words of topic 11
['like', 'time', 'good', 'would', 'work', 'believ', 'mani', 'anoth', 'us', 'world']
10 most characteristic words of topic 12
['say', 'set', 'window', 'c', 'person', 'fire', 'respons', 'public', 'law', 'abov']
10 most characteristic words of topic 13
['anyon', 'comput', 'veri', 'someon', 'question', 'say', 'complet', 'yes', 'russian',
↪'chip']
10 most characteristic words of topic 14
['subject', 'organ', 'line', 'know', 'go', 'right', 'peopl', 'see', 'take', 'year']
10 most characteristic words of topic 15
['get', 'univers', 'write', 'system', 'two', 'much', 'x', 'call', 'program', 'govern']
10 most characteristic words of topic 16
['need', 'make', 'want', 'sinc', 'one', 'talk', 'possibl', 'tri', 'keep', 'enough']
10 most characteristic words of topic 17
['never', 'drive', 'bit', 'human', 'veri', 'involv', 'shall', 'someon', 'nt', 'person
↪']
10 most characteristic words of topic 18
['one', 'make', 'john', 'origin', 'look', 'u', 'befor', 'discuss', 'doe', 'law']
10 most characteristic words of topic 19
['put', 'correct', 'price', 'littl', 'nation', 'detail', 'tank', 'mean', 'h', 'anyth']
```

Print the confusion matrix (not diagonalized)

```
predicted_topic = np.argmax(estimation, axis=1)

listvectorizer = adso.transform.ListVectorizer()
labels = list(map(lambda l: [l], data.get_labels()))

label_topic = np.squeeze(listvectorizer.fit_transform(labels))

confusion = np.zeros((20, 20))
for i in zip(label_topic, predicted_topic):
    confusion[i] += 1

fig, ax = plt.subplots()
ax.imshow(confusion)
ax.set_xticks(np.arange(20))
ax.set_yticks(np.arange(20))
ax.set_yticklabels(list(listvectorizer.vocab.stoi.keys()))
```

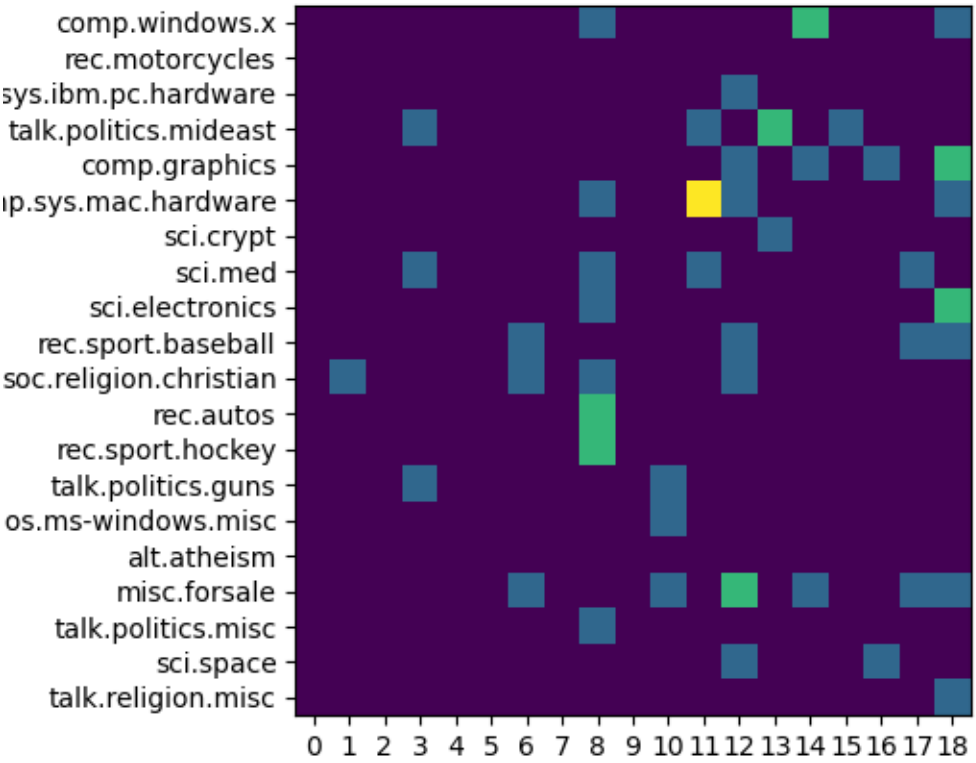


Out:

```
[Text(0, 0, 'comp.windows.x'), Text(0, 1, 'rec.motorcycles'), Text(0, 2, 'comp.sys.
↪ibm.pc.hardware'), Text(0, 3, 'talk.politics.mideast'), Text(0, 4, 'comp.graphics'),
↪ Text(0, 5, 'comp.sys.mac.hardware'), Text(0, 6, 'sci.crypt'), Text(0, 7, 'sci.med
↪'), Text(0, 8, 'sci.electronics'), Text(0, 9, 'rec.sport.baseball'), Text(0, 10,
↪'soc.religion.christian'), Text(0, 11, 'rec.autos'), Text(0, 12, 'rec.sport.hockey
↪'), Text(0, 13, 'talk.politics.guns'), Text(0, 14, 'comp.os.ms-windows.misc'),␣
↪Text(0, 15, 'alt.atheism'), Text(0, 16, 'misc.forsale'), Text(0, 17, 'talk.politics.
↪misc'), Text(0, 18, 'sci.space'), Text(0, 19, 'talk.religion.misc')]
```

Print the confusion matrix skipping the first topic

```
confusion = confusion[:, 1:]
fig, ax = plt.subplots()
ax.imshow(confusion)
ax.set_xticks(np.arange(19))
ax.set_yticks(np.arange(20))
ax.set_yticklabels(list(listvectorizer.vocab.stoi.keys()))
```



Out:

```
[Text(0, 0, 'comp.windows.x'), Text(0, 1, 'rec.motorcycles'), Text(0, 2, 'comp.sys.
↪ibm.pc.hardware'), Text(0, 3, 'talk.politics.mideast'), Text(0, 4, 'comp.graphics'),
↪ Text(0, 5, 'comp.sys.mac.hardware'), Text(0, 6, 'sci.crypt'), Text(0, 7, 'sci.med
↪'), Text(0, 8, 'sci.electronics'), Text(0, 9, 'rec.sport.baseball'), Text(0, 10,
↪'soc.religion.christian'), Text(0, 11, 'rec.autos'), Text(0, 12, 'rec.sport.hockey
↪'), Text(0, 13, 'talk.politics.guns'), Text(0, 14, 'comp.os.ms-windows.misc'),␣
↪Text(0, 15, 'alt.atheism'), Text(0, 16, 'misc.forsale'), Text(0, 17, 'talk.politics.
↪misc'), Text(0, 18, 'sci.space'), Text(0, 19, 'talk.religion.misc')]
```

Total running time of the script: (12 minutes 30.616 seconds)

2.3 Analyze the 20newsgroups dataset with NMF

import

```
import adso

import matplotlib.pyplot as plt

import nltk

import numpy as np
```

set seed

```
adso.set_seed(1234)
```

Download the dataset

```
data = adso.data.load_20newsgroups(split="test")

print("Number of documents: ", len(data))
```

Out:

```
Number of documents:  7532
```

Tokenize the dataset using a stemmer and a stopwords list, removing punctation

```
adso.transform.nltk_download("stopwords")

snowball = nltk.stem.snowball.SnowballStemmer("english")


def stemmer(word):
    ret = snowball.stem(word)
    if ret.isalpha():
        return ret
    else:
        return None


tokenizer = adso.transform.Tokenizer(
    stemmer=stemmer,
    stopwords=nltk.corpus.stopwords.words("english") + [None],
)

tokens = tokenizer.fit_transform(data)

print("First ten tokens of the first document:")
print(tokens[0][:10])
```

Out:

```
First ten tokens of the first document:
['aidler', 'e', 'alan', 'idler', 'subject', 'doctrin', 'origin', 'sin', 'organ',
 ↪'univers']
```

Transform the list of tokens in a list of numbers. We will use the frequency and the TFIDF frequency (a correction for the distribution among the documents).

```
freq = adso.transform.FreqVectorizer(max_freq=0.75, max_size=10000)

tfidf = adso.transform.TFIDFVectorizer(max_freq=0.75, max_size=10000)
```

Generate the vocabulary and share it between the vectorizer.

```
freq.fit(tokens)

# I will write an ad hoc function later
vocab = freq.vocab
```

(continues on next page)

(continued from previous page)

```
print("Number of words in vocabulary: ", len(vocab))

tfidf.vocab = vocab

print("index of word 'god': ", vocab["god"])
print("word at index 32: ", vocab[32])
```

Out:

```
Number of words in vocabulary:  10000
index of word 'god':  32
word at index 32:  god
```

Create the frequency matrices from tokens.

```
freq_matrix = freq.transform(tokens)
tfidf_matrix = tfidf.transform(tokens)
```

NMF1 using frequency matrix and ACLS algorithm

```
NMF1 = adso.topicmodel.NMF(
    n_topic=20, max_iter=100, tolerance=1e-3, lambdaH=0.001, lambdaW=0.001
)
W1, H1, iter1 = NMF1.fit_transform(freq_matrix)
print("NMF1 ended after", iter1, "iterations")
```

Out:

```
Iteration 1 - Error 305.1758155027004
Iteration 11 - Error 257.3421065099152
Iteration 21 - Error 170.63392034256293
Iteration 31 - Error 159.9954495702163
Iteration 41 - Error 154.04000548583855
Iteration 51 - Error 156.2181911808671
Iteration 61 - Error 204.83118800820122
Iteration 71 - Error 162.57089323899004
Iteration 81 - Error 159.4030973037672
Iteration 91 - Error 158.828101953583
NMF1 ended after 100 iterations
```

NMF2 using frequency matrix and AHCLS algorithm

```
NMF2 = adso.topicmodel.NMF(
    n_topic=20,
    max_iter=100,
    tolerance=1e-3,
    lambdaH=0.001,
    lambdaW=0.001,
    alphaH=0.01,
    alphaW=0.01,
    method="AHCLS",
)
W2, H2, iter2 = NMF2.fit_transform(freq_matrix)
print("NMF2 ended after", iter2, "iterations")
```

Out:

```
/home/tnto/Documenti/Universita/Tesi/src/adso/.nox/docs/lib/python3.8/site-packages/
↪scipy/sparse/linalg/dsolve/linsolve.py:144: SparseEfficiencyWarning: spsolve␣
↪requires A be CSC or CSR matrix format
  warn('spsolve requires A be CSC or CSR matrix format',
Iteration 1 - Error 170.77542271706886
Iteration 11 - Error 150.69746064173484
Iteration 21 - Error 149.07915383746712
Iteration 31 - Error 149.05744643993975
NMF2 ended after 31 iterations
```

NMF3 using tfidf matrix and ALS algorithm

```
NMF3 = adso.topicmodel.NMF(n_topic=20, max_iter=100, tolerance=1e-3, method="ALS")
W3, H3, iter3 = NMF3.fit_transform(tfidf_matrix)
print("NMF3 ended after", iter3, "iterations")
```

Out:

```
Iteration 1 - Error 29161773.19350617
Iteration 11 - Error 24063771.987029664
Iteration 21 - Error 23896399.348707188
Iteration 31 - Error 23888636.76562582
NMF3 ended after 31 iterations
```

Check the 10 most characteristic words for the first topic of each model

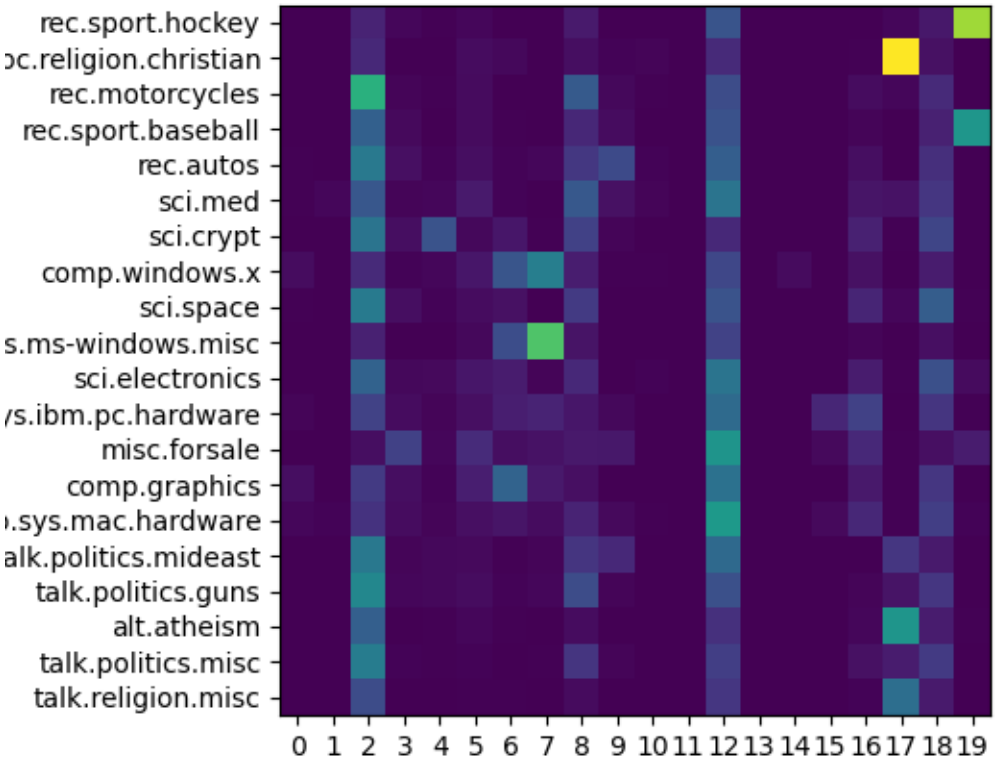
```
print("10 most characteristic words for the first topic of NMF1")
print(
    list(
        map(
            lambda i: vocab[i],
            np.argsort(np.squeeze(-H1[0, :].toarray()))[:10].tolist(),
        )
    )
)
print("10 most characteristic words for the first topic of NMF2")
print(
    list(
        map(
            lambda i: vocab[i],
            np.argsort(np.squeeze(-H2[0, :].toarray()))[:10].tolist(),
        )
    )
)
print("10 most characteristic words for the first topic of NMF3")
print(
    list(
        map(
            lambda i: vocab[i],
            np.argsort(np.squeeze(-H3[0, :].toarray()))[:10].tolist(),
        )
    )
)
```

Out:

```
10 most characteristic words for the first topic of NMF1
['thank', 'ani', 'look', 'help', 'program', 'need', 'pleas', 'advanc', 'could',
↪'inform']
10 most characteristic words for the first topic of NMF2
['one', 'two', 'want', 'time', 'year', 'onli', 'thing', 'card', 'way', 'line']
10 most characteristic words for the first topic of NMF3
['ppd', 'merc', 'asthma', 'cds', 'nova', 'howland', 'teenag', 'mob', 'disc', 'rob']
```

Print the confusion matrix (not diagonalized) for NMF1

```
predicted_topic = np.argmax(W1, axis=1)

listvectorizer = adso.transform.ListVectorizer()
labels = list(map(lambda l: [l], data.get_labels()))

label_topic = np.squeeze(listvectorizer.fit_transform(labels))

confusion = np.zeros((20, 20))
for i in zip(label_topic, predicted_topic):
    confusion[i] += 1

fig, ax = plt.subplots()
ax.imshow(confusion)
ax.set_xticks(np.arange(20))
ax.set_yticks(np.arange(20))
ax.set_yticklabels(list(listvectorizer.vocab.stoi.keys()))
```



Out:

```
[Text(0, 0, 'rec.sport.hockey'), Text(0, 1, 'soc.religion.christian'), Text(0, 2,
↪'rec.motorcycles'), Text(0, 3, 'rec.sport.baseball'), Text(0, 4, 'rec.autos'),␣
↪Text(0, 5, 'sci.med'), Text(0, 6, 'sci.crypt'), Text(0, 7, 'comp.windows.x'),␣
↪Text(0, 8, 'sci.space'), Text(0, 9, 'comp.os.ms-windows.misc'), Text(0, 10, 'sci.
↪electronics'), Text(0, 11, 'comp.sys.ibm.pc.hardware'), Text(0, 12, 'misc.forsale'),
↪ Text(0, 13, 'comp.graphics'), Text(0, 14, 'comp.sys.mac.hardware'), Text(0, 15,
↪'talk.politics.mideast'), Text(0, 16, 'talk.politics.guns'), Text(0, 17, 'alt.
↪atheism'), Text(0, 18, 'talk.politics.misc'), Text(0, 19, 'talk.religion.misc')]
```

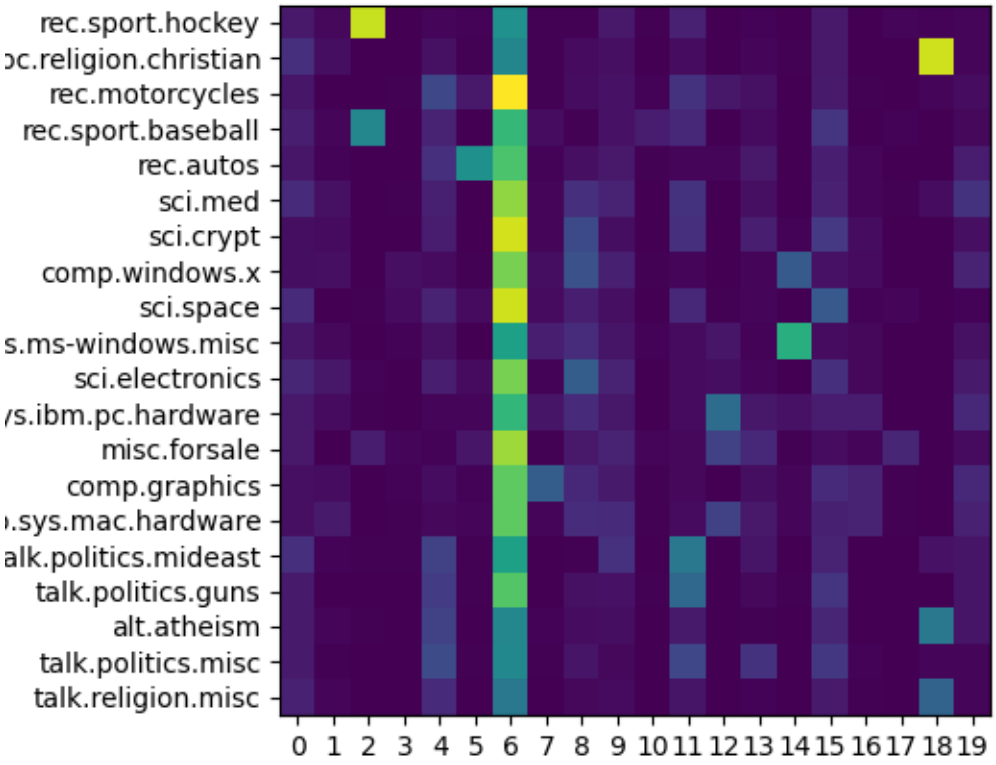
Print the confusion matrix (not diagonalized) for NMF2

```
predicted_topic = np.argmax(W2, axis=1)

listvectorizer = adso.transform.ListVectorizer()
labels = list(map(lambda l: [l], data.get_labels()))

label_topic = np.squeeze(listvectorizer.fit_transform(labels))

confusion = np.zeros((20, 20))
for i in zip(label_topic, predicted_topic):
    confusion[i] += 1

fig, ax = plt.subplots()
ax.imshow(confusion)
```

(continues on next page)

(continued from previous page)

```
ax.set_xticks(np.arange(20))
ax.set_yticks(np.arange(20))
ax.set_yticklabels(list(listvectorizer.vocab.stoi.keys()))
```



Out:

```
[Text(0, 0, 'rec.sport.hockey'), Text(0, 1, 'soc.religion.christian'), Text(0, 2,
↪'rec.motorcycles'), Text(0, 3, 'rec.sport.baseball'), Text(0, 4, 'rec.autos'),␣
↪Text(0, 5, 'sci.med'), Text(0, 6, 'sci.crypt'), Text(0, 7, 'comp.windows.x'),␣
↪Text(0, 8, 'sci.space'), Text(0, 9, 'comp.os.ms-windows.misc'), Text(0, 10, 'sci.
↪electronics'), Text(0, 11, 'comp.sys.ibm.pc.hardware'), Text(0, 12, 'misc.forsale'),
↪ Text(0, 13, 'comp.graphics'), Text(0, 14, 'comp.sys.mac.hardware'), Text(0, 15,
↪'talk.politics.mideast'), Text(0, 16, 'talk.politics.guns'), Text(0, 17, 'alt.
↪atheism'), Text(0, 18, 'talk.politics.misc'), Text(0, 19, 'talk.religion.misc')]
```

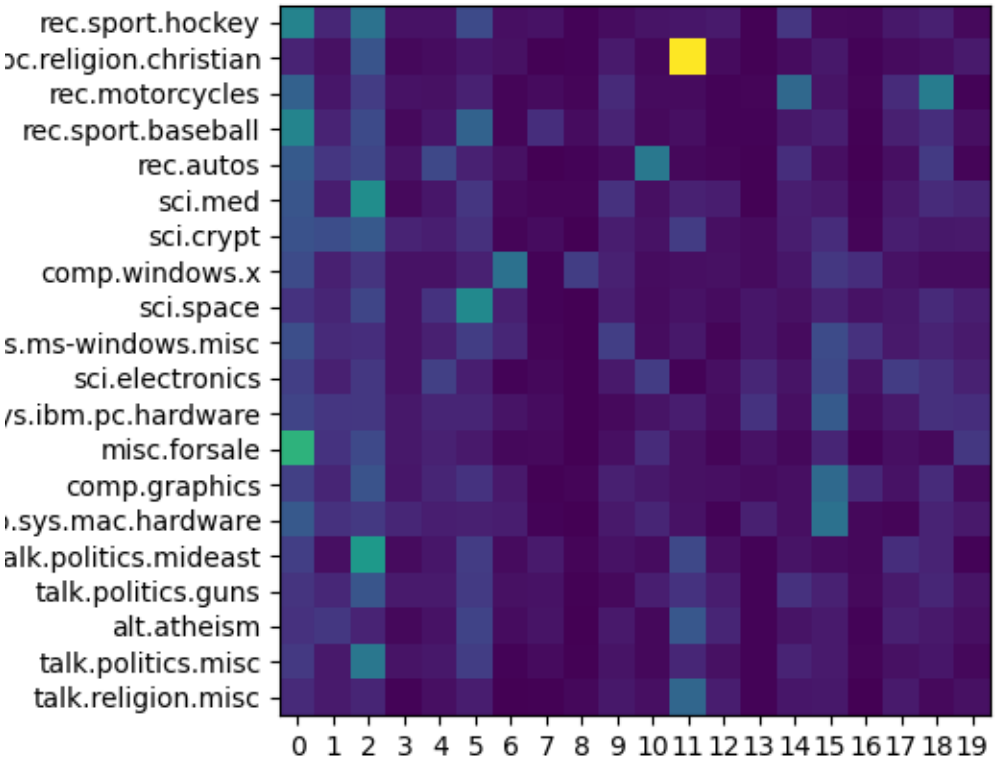
Print the confusion matrix (not diagonalized) for NMF3

```
predicted_topic = np.argmax(W3, axis=1)

listvectorizer = adso.transform.ListVectorizer()
labels = list(map(lambda l: [l], data.get_labels()))

label_topic = np.squeeze(listvectorizer.fit_transform(labels))
```

(continues on next page)

(continued from previous page)

```
confusion = np.zeros((20, 20))
for i in zip(label_topic, predicted_topic):
    confusion[i] += 1

fig, ax = plt.subplots()
ax.imshow(confusion)
ax.set_xticks(np.arange(20))
ax.set_yticks(np.arange(20))
ax.set_yticklabels(list(listvectorizer.vocab.stoi.keys()))
```



Out:

```
[Text(0, 0, 'rec.sport.hockey'), Text(0, 1, 'soc.religion.christian'), Text(0, 2,
↪'rec.motorcycles'), Text(0, 3, 'rec.sport.baseball'), Text(0, 4, 'rec.autos'),␣
↪Text(0, 5, 'sci.med'), Text(0, 6, 'sci.crypt'), Text(0, 7, 'comp.windows.x'),␣
↪Text(0, 8, 'sci.space'), Text(0, 9, 'comp.os.ms-windows.misc'), Text(0, 10, 'sci.
↪electronics'), Text(0, 11, 'comp.sys.ibm.pc.hardware'), Text(0, 12, 'misc.forsale'),
↪ Text(0, 13, 'comp.graphics'), Text(0, 14, 'comp.sys.mac.hardware'), Text(0, 15,
↪'talk.politics.mideast'), Text(0, 16, 'talk.politics.guns'), Text(0, 17, 'alt.
↪atheism'), Text(0, 18, 'talk.politics.misc'), Text(0, 19, 'talk.religion.misc')]
```

Total running time of the script: (15 minutes 21.981 seconds)

CHAPTER

THREE

LICENSE

MIT License

Copyright (c) 2020, Michele Ciruzzi

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

FOUR

INDICES

- genindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

C

D

F

G

I

L

M

N

S

T

V