DHLAB Documentation

Release 2.21.2

Lars Johnsen, Magnus Breder Birkenes, Andre Kåsen, Yngvil Bey

16 February 2023

CONTENTS

`dhlab` is a python library for doing qualitative and quantitative analyses of the digital texts from *nettbiblioteket* (eng: "the online library") at the National Library of Norway (NLN). *Nettbiblioteket* is the NLN's digital collection of media publications.

CHAPTER

ONE

FUNCTIONALITY

Here are some of the text mining and automatic analyses you can do with `dhlab`:

- *Build corpora* with bibliographic metadata about publications
- word/token frequencies
- text chunks (paragraphs)
- concordances
- collocations
- n-grams
- named entity extraction
- narrative graphs (word dispersion)

Analyses can be performed on both a single document, and on a larger corpus. You can build your own corpus based on bibliographic metadata.

The python package calls the DHLAB API to retrieve and present data from the digital texts.

Further reading <https://nationallibraryofnorway.github.io/DHLAB-kunnskapsbase/>

- Alphabetical Index
- Module Index
- Search

CHAPTER

TWO

GET STARTED

On our official homepage (in Norwegian), you can view and run example jupyter notebooks without installing anything.

Below you'll find installation instructions, and examples of using the dhlab python package.

2.1 Ways to do text analysis with dhlab

- In the python console or an IDE: *Install dhlab*
- Via our streamlit apps
- In our example Jupyter Notebooks
  - on our website: nb.no/dhlab/digital_tekstanalyse/
  - on github: digital_tekstanalyse
- With API-calls to the DH-lab REST-API
- With R on github: dhlab-r

2.2 Installation

2.2.1 Prerequisites

Ensure the following are installed:

- Python (a version between 3.8 - 3.11)
- pip

2.2.2 Installation with pip

Install the latest version of `dhlab` in your (Unix) terminal with pip:

```
pip install -U dhlab
```

2.2.3 Installation from github

- Ensure you have git installed on your system.
- Open your terminal in the file location you will work with DHLAB.

```
git clone https://github.com/NationalLibraryOfNorway/DHLAB.git
cd DHLAB
pip install -U -e .
```

2.3 Run notebooks locally

1. Ensure the following are installed:
   - Jupyter Lab
   - git
   - dhlab
2. Clone the digital_tekstanalyse Github repo (from a terminal or command line):

   ```
   git clone git@github.com:NationalLibraryOfNorway/digital_tekstanalyse.git
   ```

3. Navigate into the cloned repo and start `jupyter-lab` from the terminal

   ```
   cd digital_tekstanalyse
   jupyter-lab
   ```

Once `jupyter lab` is up and running, just open any `.ipynb` notebook file in the left-hand menu and follow the instructions to execute the code in the cells.

2.4 Example Use

2.4.1 Build a corpus

You could start by building your own corpus, e.g. of books published between 1814 and 1905:

```
import dhlab as dh

book_corpus = dh.Corpus(doctype="digibok", from_year=1814, to_year=1905)
```

You can now use `book_corpus` and to do quantitative text analyses. `book_corpus.corpus` is a Pandas dataframe with metadata information about the publications in the corpus.

CHAPTER

THREE

DHLAB

3.1 Sub-packages

api	Python wrappers for API calls to the databases containing NLN's digital archive.
ngram	Extraction of n-gram counts and aggregations.
text	Text analysis functionality, including building text corpora.
wordbank.wordbank	
metadata	Metadata extraction.
images.nbpictures	
visualize	Visualization tools

home << *dhlab*

3.1.1 api

```
from dhlab import api
```

Python wrappers for API calls to the databases containing NLN's digital archive.

Modules

dhlab.api.dhlab_api
dhlab.api.nb_ngram_api
dhlab.api.nb_search_api

home << *dhlab* << *dhlab.api*

dhlab_api

```
from dhlab.api import dhlab_api
```

Functions

collocation([corpusquery, word, before, after])	Make a collocation from a corpus query.
concordance([urns, words, window, limit])	Get a list of concordances from the National Library's database.
concordance_counts([urns, words, window, limit])	Count concordances (keyword in context) for a corpus query (used for collocation analysis).
document_corpus([doctype, author, freetext, ...])	Fetch a corpus based on metadata.
evaluate_documents([wordbags, urns])	Count and aggregate occurrences of topic `wordbags` for each document in a list of `urns`.
find_urns([docids, mode])	Return a list of URNs from a collection of docids.
geo_lookup(places[, feature_class, ...])	From a list of places, return their geolocations
get_chunks([urn, chunk_size])	Get the text in the document `urn` as frequencies of chunks
get_chunks_para([urn])	Fetch chunks and their frequencies from paragraphs in a document (`urn`).
get_dispersion([urn, words, window, pr])	Count occurrences of words in the given URN object.
get_document_corpus(**kwargs)	
get_document_frequencies([urns, cutoff, words])	Fetch frequency counts of `words` in documents (`urns`).
get_metadata([urns])	Get metadata for a list of URNs.
get_places(urn)	Look up placenames in a specific URN.
get_reference([corpus, from_year, to_year, ...])	Reference frequency list of the n most frequent words from a given corpus in a given period.
get_urn_frequencies([urns, dhlabid])	Fetch frequency counts of documents as URNs or DH-lab ids.
get_word_frequencies([urns, cutoff, words])	Fetch frequency numbers for `words` in documents (`urns`).
images([text, part])	Retrive images from bokhylla :param text: fulltext query expression for sqlite :param part: if a number the whole page is shown .
konkordans([urns, words, window, limit])	Wrapper for *concordance()*.
ner_from_urn([urn, model, start_page, to_page])	Get NER annotations for a text (`urn`) using a spacy `model`.
ngram_book([word, title, period, publisher, ...])	Count occurrences of one or more words in books over a given time period.
ngram_news([word, title, period])	Get a time series of frequency counts for `word` in newspapers.
ngram_periodicals([word, title, period, ...])	Get a time series of frequency counts for `word` in periodicals.
pos_from_urn([urn, model, start_page, to_page])	Get part of speech tags and dependency parse annotations for a text (`urn`) with a SpaCy `model`.
reference_words([words, doctype, from_year, ...])	Collect reference data for a list of words over a time period.
show_spacy_models()	Show available SpaCy model names.

continues on next page

Table 1 – continued from previous page

totals([top_words])	Get aggregated raw frequencies of all words in the National Library's database.
urn_collocation([urns, word, before, after, ...])	Create a collocation from a list of URNs.
wildcard_search(word[, factor, freq_limit, ...])	
word_form(word[, lang])	Look up the morphological feature specification of a `word` form.
word_form_many(wordlist[, lang])	Look up the morphological feature specifications for word forms in a `wordlist`.
word_lemma(word[, lang])	Find the list of possible lemmas for a given `word` form.
word_lemma_many(wordlist[, lang])	Find lemmas for a list of given word forms.
word_paradigm(word[, lang])	Find paradigms for a given `word` form.
word_paradigm_many(wordlist[, lang])	Find alternative forms for a list of words.
word_variant(word, form[, lang])	Find alternative `form` for a given `word` form.

home << *dhlab* << *dhlab.api.dhlab_api*

collocation

```
from dhlab.api.dhlab_api import collocation
```

collocation(*corpusquery='norge'*, *word='arbeid'*, *before=5*, *after=0*)

Make a collocation from a corpus query.

Parameters

- **corpusquery** (*str*) – query string
- **word** (*str*) – target word for the collocations.
- **before** (*int*) – number of words prior to `word`
- **after** (*int*) – number of words following `word`

Returns
a dataframe with the resulting collocations

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

concordance

```
from dhlab.api.dhlab_api import concordance
```

concordance(*urns=None*, *words=None*, *window=25*, *limit=100*)

Get a list of concordances from the National Library's database.

Call the API *BASE_URL* endpoint /conc.

Parameters

- **urns** (*list*) – uniform resource names, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`

- **words** (*str*) – Word(s) to search for. Can be an SQLite fulltext query, an fts5 string search expression.
- **window** (*int*) – number of tokens on either side to show in the collocations, between 1-25.
- **limit** (*int*) – max. number of concordances per document. Maximum value is 1000.

Returns
a table of concordances

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

concordance_counts

```
from dhlab.api.dhlab_api import concordance_counts
```

concordance_counts(*urns=None*, *words=None*, *window=25*, *limit=100*)

Count concordances (keyword in context) for a corpus query (used for collocation analysis).

Call the API *BASE_URL* endpoint /conccount.

Parameters

- **urns** (*list*) – uniform resource names, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`
- **words** (*str*) – Word(s) to search for. Can be an SQLite fulltext query, an fts5 string search expression.
- **window** (*int*) – number of tokens on either side to show in the collocations, between 1-25.
- **limit** (*int*) – max. number of concordances per document. Maximum value is 1000.

Returns
a table of counts

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

document_corpus

```
from dhlab.api.dhlab_api import document_corpus
```

document_corpus(*doctype=None*, *author=None*, *freetext=None*, *fulltext=None*, *from_year=None*, *to_year=None*, *from_timestamp=None*, *to_timestamp=None*, *title=None*, *ddk=None*, *subject=None*, *lang=None*, *limit=None*, *order_by=None*)

Fetch a corpus based on metadata.

Call the API *BASE_URL* endpoint /build_corpus.

Parameters

- **doctype** (*str*) – `"digibok"`, `"digavis"`, `"digitidsskrift"` or `"digistorting"`
- **author** (*str*) – Name of an author.

- **freetext** (*str*) – any of the parameters, for example: "digibok AND Ibsen".
- **fulltext** (*str*) – words within the publication.
- **from_year** (*int*) – Start year for time period of interest.
- **to_year** (*int*) – End year for time period of interest.
- **from_timestamp** (*int*) – Start date for time period of interest. Format: YYYYMMDD, books have YYYY0101
- **to_timestamp** (*int*) – End date for time period of interest. Format: YYYYMMDD, books have YYYY0101
- **title** (*str*) – Name or title of a document.
- **ddk** (*str*) – Dewey Decimal Classification identifier.
- **subject** (*str*) – subject (keywords) of the publication.
- **lang** (*str*) – Language of the publication, as a 3-letter ISO code. Example: "nob" or "nno"
- **limit** (*int*) – number of items to sample.
- **order_by** (*str*) – order of elements in the corpus object. Typically used in combination with a limit. Example "random" (random order, the slowest), "rank" (ordered by relevance, faster) or "first" (breadth-first, using the order in the database table, the fastest method)

Returns
a pandas.DataFrame with the corpus information.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

evaluate_documents

```
from dhlab.api.dhlab_api import evaluate_documents
```

evaluate_documents(*wordbags=None*, *urns=None*)

Count and aggregate occurrences of topic wordbags for each document in a list of urns.

Parameters

- **wordbags** (*dict*) – a dictionary of topic keywords and lists of associated words. Example: {"natur": ["planter", "skog", "fjell", "fjord"], ... }
- **urns** (*list*) – uniform resource names, for example: ["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]

Returns
a pandas.DataFrame with the topics as columns, indexed by the dhlabids of the documents.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

find_urns

```
from dhlab.api.dhlab_api import find_urns
```

find_urns(*docids=None*, *mode='json'*)

Return a list of URNs from a collection of docids.

Call the API *BASE_URL* endpoint `/find_urn`.

Parameters

- **docids** (*Union[Dict, DataFrame]*) – dictionary of document IDs (`{docid: URN}`) or a `pandas.DataFrame`.
- **mode** (*str*) – Default 'json'.

Returns
the URNs that were found, in a `pandas.DataFrame`.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

geo_lookup

```
from dhlab.api.dhlab_api import geo_lookup
```

geo_lookup(*places*, *feature_class=None*, *feature_code=None*, *field='alternatename'*)

From a list of places, return their geolocations

Parameters

- **places** (*list*) – a list of place names - max 1000
- **feature_class** (*str*) – which GeoNames feature class to return. Example: `P`
- **feature_code** (*str*) – which GeoNames feature code to return. Example: `PPL`
- **field** (*str*) – which name field to match - default "alternatename".

home << *dhlab* << *dhlab.api.dhlab_api*

get_chunks

```
from dhlab.api.dhlab_api import get_chunks
```

get_chunks(*urn=None*, *chunk_size=300*)

Get the text in the document urn as frequencies of chunks
of the given `chunk_size`.

Calls the API *BASE_URL* endpoint `/chunks`.

Parameters

- **urn** (*str*) – uniform resource name, example: `URN:NBN:no-nb_digibok_2011051112001`
- **chunk_size** (*int*) – Number of tokens to include in each chunk.

Returns
list of dicts with the resulting chunk frequencies, or an empty dict

Return type
Union[*Dict*, *List*]

home << *dhlab* << *dhlab.api.dhlab_api*

get_chunks_para

```
from dhlab.api.dhlab_api import get_chunks_para
```

get_chunks_para(*urn=None*)

Fetch chunks and their frequencies from paragraphs in a document (urn).

Calls the API *BASE_URL* endpoint `/chunks_para`.

Parameters
urn (*str*) – uniform resource name, example: `URN:NBN:no-nb_digibok_2011051112001`

Returns
list of dicts with the resulting chunk frequencies, or an empty dict

Return type
Union[*Dict*, *List*]

home << *dhlab* << *dhlab.api.dhlab_api*

get_dispersion

```
from dhlab.api.dhlab_api import get_dispersion
```

get_dispersion(*urn=None*, *words=None*, *window=300*, *pr=100*)

Count occurrences of words in the given URN object.

Call the API *BASE_URL* endpoint `/dispersion`.

Parameters

- **urn** (*str*) – uniform resource name, example: `URN:NBN:no-nb_digibok_2011051112001`
- **words** (*list*) – list of words. Defaults to a list of punctuation marks.
- **window** (*int*) – The number of tokens to search through per row. Defaults to 300.
- **pr** (*int*) – defaults to 100.

Returns
a `pandas.Series` with frequency counts of the words in the URN object.

Return type
Series

home << *dhlab* << *dhlab.api.dhlab_api*

get_document_corpus

```
from dhlab.api.dhlab_api import get_document_corpus
```

get_document_corpus(***kwargs*)

home << *dhlab* << *dhlab.api.dhlab_api*

get_document_frequencies

```
from dhlab.api.dhlab_api import get_document_frequencies
```

get_document_frequencies(*urns=None*, *cutoff=0*, *words=None*)

Fetch frequency counts of `words` in documents (`urns`).

Call the API *BASE_URL* endpoint `/frequencies`.

Parameters

- **urns** (*list*) – list of uniform resource name strings, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`
- **cutoff** (*int*) – minimum frequency of a word to be counted
- **words** (*list*) – a list of words to be counted - if left None, whole document is returned.

home << *dhlab* << *dhlab.api.dhlab_api*

get_metadata

```
from dhlab.api.dhlab_api import get_metadata
```

get_metadata(*urns=None*)

Get metadata for a list of URNs.

Calls the API *BASE_URL* endpoint /get_metadata.

Parameters

urns (*list*) – list of uniform resource name strings, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`

home << *dhlab* << *dhlab.api.dhlab_api*

get_places

```
from dhlab.api.dhlab_api import get_places
```

get_places(*urn*)

Look up placenames in a specific URN.

Call the API *BASE_URL* endpoint /places.

Parameters

urn (*str*) – uniform resource name, example: `URN:NBN:no-nb_digibok_2011051112001`

home << *dhlab* << *dhlab.api.dhlab_api*

get_reference

```
from dhlab.api.dhlab_api import get_reference
```

get_reference(*corpus='digavis'*, *from_year=1950*, *to_year=1955*, *lang='nob'*, *limit=100000*)

Reference frequency list of the n most frequent words from a given corpus in a given period.

Call the API *BASE_URL* endpoint /reference_corpus.

Parameters

- **corpus** (*str*) – Document type to include in the corpus, can be either `'digibok'` or `'digavis'`.
- **from_year** (*int*) – Starting point for time period of the corpus.
- **to_year** (*int*) – Last year of the time period of the corpus.
- **lang** (*str*) – Language of the corpus, can be one of `'nob,'`, `'nno,'`, `'sme,'`, `'sma,'`, `'smj'`, `'fkv'`
- **limit** (*int*) – Maximum number of most frequent words.

Returns

A `pandas.DataFrame` with the results.

Return type

DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

get_urn_frequencies

```
from dhlab.api.dhlab_api import get_urn_frequencies
```

get_urn_frequencies(*urns=None*, *dhlabid=None*)

Fetch frequency counts of documents as URNs or DH-lab ids.

Call the API *BASE_URL* endpoint `/frequencies`.

Parameters

- **urns** (*list*) – list of uniform resource name strings, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`
- **dhlabid** (*list*) – list of numbers for dhlabid: `[1000001, 2000003]`

home << *dhlab* << *dhlab.api.dhlab_api*

get_word_frequencies

```
from dhlab.api.dhlab_api import get_word_frequencies
```

get_word_frequencies(*urns=None*, *cutoff=0*, *words=None*)

Fetch frequency numbers for `words` in documents (`urns`).

Call the API *BASE_URL* endpoint `/frequencies`.

Parameters

- **urns** (*list*) – list of uniform resource name strings, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`
- **cutoff** (*int*) – minimum frequency of a word to be counted
- **words** (*list*) – a list of words to be counted - should not be left None.

home << *dhlab* << *dhlab.api.dhlab_api*

images

```
from dhlab.api.dhlab_api import images
```

images(*text=None*, *part=True*)

Retrive images from bokhylla :param text: fulltext query expression for sqlite :param part: if a number the whole page is shown … bug prevents these from going thru :param delta: if part=True then show additional pixels around image :parsm hits: number of images

home << *dhlab* << *dhlab.api.dhlab_api*

konkordans

```
from dhlab.api.dhlab_api import konkordans
```

konkordans(*urns=None*, *words=None*, *window=25*, *limit=100*)

Wrapper for *concordance()*.

home << *dhlab* << *dhlab.api.dhlab_api*

ner_from_urn

```
from dhlab.api.dhlab_api import ner_from_urn
```

ner_from_urn(*urn=None*, *model=None*, *start_page=0*, *to_page=0*)

Get NER annotations for a text (urn) using a spacy model.

Parameters

- **urn** (*str*) – uniform resource name, example: URN:NBN:no-nb_digibok_2011051112001
- **model** (*str*) – name of a spacy model. Check which models are available with *show_spacy_models()*

Returns
Dataframe with annotations and their frequencies

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

ngram_book

```
from dhlab.api.dhlab_api import ngram_book
```

ngram_book(*word=['.']*, *title=None*, *period=None*, *publisher=None*, *lang=None*, *city=None*, *ddk=None*, *topic=None*)

Count occurrences of one or more words in books over a given time period.

Call the API *BASE_URL* endpoint /ngram_book.

Filter the selection of books with metadata. Use % as wildcard where appropriate - no wildcards in word or lang.

Parameters

- **word** (*str or list of str*) – Word(s) to search for. Can be several words in a single string, separated by comma, e.g. "ord,ordene,orda".
- **title** (*str*) – Title of a specific document to search through.
- **period** (*tuple of ints*) – Start and end years or dates of a time period, given as (YYYY, YYYY) or (YYYYMMDD, YYYYMMDD).
- **publisher** (*str*) – Name of a publisher.
- **lang** (*str*) – Language as a 3-letter ISO code (e.g. "nob" or "nno")
- **city** (*str*) – City of publication.
- **ddk** (*str*) – Dewey Decimal Classification identifier.
- **topic** (*str*) – Topic of the documents.

Returns
a pandas.DataFrame with the resulting frequency counts of the word(s), spread across years. One year per row.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

ngram_news

```
from dhlab.api.dhlab_api import ngram_news
```

ngram_news(*word=['.']*, *title=None*, *period=None*)

Get a time series of frequency counts for `word` in newspapers.

Call the API *BASE_URL* endpoint `/ngram_newspapers`.

Parameters

- **word** (*str or list of str*) – Word(s) to search for. Can be several words in a single string, separated by comma, e.g. `"ord,ordene,orda"`.
- **title** (*str*) – Title of a specific newspaper to search through.
- **period** (*tuple of ints*) – Start and end years or dates of a time period, given as `(YYYY, YYYY)` or `(YYYYMMDD, YYYYMMDD)`.

Returns

a `pandas.DataFrame` with the resulting frequency counts of the word(s), spread across the dates given in the time period. Either one year or one day per row.

Return type

DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

ngram_periodicals

```
from dhlab.api.dhlab_api import ngram_periodicals
```

ngram_periodicals(*word=['.']*, *title=None*, *period=None*, *publisher=None*, *lang=None*, *city=None*, *ddk=None*, *topic=None*, ***kwargs*)

Get a time series of frequency counts for `word` in periodicals.

Call the API *BASE_URL* endpoint `/ngram_periodicals`.

Parameters

- **word** (*str or list of str*) – Word(s) to search for. Can be several words in a single string, separated by comma, e.g. `"ord,ordene,orda"`.
- **title** (*str*) – Title of a specific document to search through.
- **period** (*tuple of ints*) – Start and end years or dates of a time period, given as `(YYYY, YYYY)` or `(YYYYMMDD, YYYYMMDD)`.
- **publisher** (*str*) – Name of a publisher.
- **lang** (*str*) – Language as a 3-letter ISO code (e.g. `"nob"` or `"nno"`)
- **city** (*str*) – City of publication.
- **ddk** (*str*) – Dewey Decimal Classification identifier.
- **topic** (*str*) – Topic of the documents.

Returns
a `pandas.DataFrame` with the resulting frequency counts of the word(s), spread across years. One year per row.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

pos_from_urn

```
from dhlab.api.dhlab_api import pos_from_urn
```

pos_from_urn(*urn=None*, *model=None*, *start_page=0*, *to_page=0*)

Get part of speech tags and dependency parse annotations for a text (`urn`) with a SpaCy `model`.

Parameters

- **urn** (*str*) – uniform resource name, example: `URN:NBN:no-nb_digibok_2011051112001`
- **model** (*str*) – name of a spacy model. Check which models are available with *show_spacy_models()*

Returns
Dataframe with annotations and their frequencies

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

reference_words

```
from dhlab.api.dhlab_api import reference_words
```

reference_words(*words=None*, *doctype='digibok'*, *from_year=1800*, *to_year=2000*)

Collect reference data for a list of words over a time period.

Reference data are the absolute and relative frequencies of the `words` across all documents of the given `doctype` in the given time period (`from_year` - `to_year`).

Parameters

- **words** (*list*) – list of word strings.
- **doctype** (*str*) – type of reference document. Can be `"digibok"` or `"digavis"`. Defaults to `"digibok"`.

  Note: If any other string is given as the `doctype`, the resulting data is equivalent to what you get with `doctype="digavis"`.

- **from_year** (*int*) – first year of publication
- **to_year** (*int*) – last year of publication

Returns
a DataFrame with the words' frequency data

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

show_spacy_models

```
from dhlab.api.dhlab_api import show_spacy_models
```

show_spacy_models()

Show available SpaCy model names.

home << *dhlab* << *dhlab.api.dhlab_api*

totals

```
from dhlab.api.dhlab_api import totals
```

totals(*top_words=50000*)

Get aggregated raw frequencies of all words in the National Library's database.

Call the API *BASE_URL* endpoint /totals/{top_words}.

Parameters
top_words (*int*) – The number of words to get total frequencies for.

Returns
a `pandas.DataFrame` with the most frequent words.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

urn_collocation

```
from dhlab.api.dhlab_api import urn_collocation
```

urn_collocation(*urns=None*, *word='arbeid'*, *before=5*, *after=0*, *samplesize=200000*)

Create a collocation from a list of URNs.

Call the API *BASE_URL* endpoint `/urncolldist_urn`.

Parameters

- **urns** (*list*) – list of uniform resource name strings, for example: `["URN:NBN:no-nb_digibok_2008051404065", "URN:NBN:no-nb_digibok_2010092120011"]`
- **word** (*str*) – word to construct collocation with.
- **before** (*int*) – number of words preceding the given `word`.
- **after** (*int*) – number of words following the given `word`.

- **samplesize** (*int*) – total number of urns to search through.

Returns
a pandas.DataFrame with distance (sum of distances and bayesian distance) and frequency for words collocated with word.

Return type
DataFrame

home << *dhlab* << *dhlab.api.dhlab_api*

wildcard_search

```
from dhlab.api.dhlab_api import wildcard_search
```

wildcard_search(*word*, *factor=2*, *freq_limit=10*, *limit=50*)

home << *dhlab* << *dhlab.api.dhlab_api*

word_form

```
from dhlab.api.dhlab_api import word_form
```

word_form(*word*, *lang='nob'*)
Look up the morphological feature specification of a word form.

home << *dhlab* << *dhlab.api.dhlab_api*

word_form_many

```
from dhlab.api.dhlab_api import word_form_many
```

word_form_many(*wordlist*, *lang='nob'*)
Look up the morphological feature specifications for word forms in a wordlist.

home << *dhlab* << *dhlab.api.dhlab_api*

word_lemma

```
from dhlab.api.dhlab_api import word_lemma
```

word_lemma(*word*, *lang='nob'*)
Find the list of possible lemmas for a given word form.

home << *dhlab* << *dhlab.api.dhlab_api*

word_lemma_many

```
from dhlab.api.dhlab_api import word_lemma_many
```

word_lemma_many(*wordlist*, *lang='nob'*)

Find lemmas for a list of given word forms.

home << *dhlab* << *dhlab.api.dhlab_api*

word_paradigm

```
from dhlab.api.dhlab_api import word_paradigm
```

word_paradigm(*word*, *lang='nob'*)

Find paradigms for a given `word` form.

Call the API *BASE_URL* endpoint `/paradigm`

Example:

```
word_paradigm('spiste')
# [['adj', ['spisende', 'spist', 'spiste']],
# ['verb', ['spis', 'spise', 'spiser', 'spises', 'spist', 'spiste']]]
```

Parameters

- **word** (*str*) – any word string
- **lang** (*str*) – either “nob” or “nno”

home << *dhlab* << *dhlab.api.dhlab_api*

word_paradigm_many

```
from dhlab.api.dhlab_api import word_paradigm_many
```

word_paradigm_many(*wordlist*, *lang='nob'*)

Find alternative forms for a list of words.

home << *dhlab* << *dhlab.api.dhlab_api*

word_variant

```
from dhlab.api.dhlab_api import word_variant
```

word_variant(*word*, *form*, *lang='nob'*)

Find alternative `form` for a given `word` form.

Call the API *BASE_URL* endpoint `/variant_form`

Example: `word_variant('spiste', 'pres-part')`

Parameters

- **word** (*str*) – any word string
- **form** (*str*) – a morphological feature tag from the Norwegian wordbank "Orbanken".
- **lang** (*str*) – either "nob" or "nno"

home << *dhlab* << *dhlab.api*

nb_ngram_api

```
from dhlab.api import nb_ngram_api
```

Functions

get_ngram(terms[, corpus, lang])	Fetch raw and relative frequencies for the `terms`.
make_word_graph(words[, corpus, cutoff, leaves])	Get galaxy from ngram-database.

home << *dhlab* << *dhlab.api.nb_ngram_api*

get_ngram

```
from dhlab.api.nb_ngram_api import get_ngram
```

get_ngram(*terms*, *corpus='avis'*, *lang='nob'*)

Fetch raw and relative frequencies for the `terms`.

Call the `NGRAM_API`. The frequencies are aggregated per year between 1800-2021.

Parameters

- **terms** (*str*) – comma separated string of words
- **corpus** (*str*) – type of documents to search through

Returns
table of annual frequency counts per term

Return type
dict

home << *dhlab* << *dhlab.api.nb_ngram_api*

make_word_graph

```
from dhlab.api.nb_ngram_api import make_word_graph
```

make_word_graph(*words*, *corpus='all'*, *cutoff=16*, *leaves=0*)

Get galaxy from ngram-database.

Call the *GALAXY_API* endpoint.

Parameters

- **words** (*str*) – comma-separated string of words
- **corpus** (*str*) – document type: `'book'`, `'avis'`, or `'all'`,
- **cutoff** (*int*) – Number of nodes to include.
- **leaves** (*int*) – Set leaves=1 to get the leaves.

Returns

A `networkx.DiGraph` with the results.

Return type

DiGraph

home << *dhlab* << *dhlab.api*

nb_search_api

```
from dhlab.api import nb_search_api
```

Functions

get_data(frase[, media, title])	
get_data_and(frases[, title, media])	
get_df(phrases[, title])	Get `phrases` from documents of `title`, and aggregate their frequencies.
get_df_pd(frase[, media])	
get_json(phrases[, mediatype])	
get_konks(urn, phrase[, window, n])	
get_phrase_info(urn, phrase[, window, n])	
iiif_manifest(urn)	Fetch the IIIF manifest of the images that the book
load_picture(url)	Load the raw image object from a URL.
mods(urn)	
super_search(term[, number, page, mediatype])	Search for a term, return a json object with the matches.
total_search([size, page])	Retrieve the first `size` occurrences from `page`.

home << *dhlab* << *dhlab.api.nb_search_api*

get_data

```
from dhlab.api.nb_search_api import get_data
```

get_data(*frase*, *media='avis'*, *title='jazznytt'*)

home << *dhlab* << *dhlab.api.nb_search_api*

get_data_and

```
from dhlab.api.nb_search_api import get_data_and
```

get_data_and(*frases*, *title='aftenposten'*, *media='avis'*)

home << *dhlab* << *dhlab.api.nb_search_api*

get_df

```
from dhlab.api.nb_search_api import get_df
```

get_df(*phrases*, *title='aftenposten'*)

Get `phrases` from documents of `title`, and aggregate their frequencies.

home << *dhlab* << *dhlab.api.nb_search_api*

get_df_pd

```
from dhlab.api.nb_search_api import get_df_pd
```

get_df_pd(*frase*, *media='bøker'*)

home << *dhlab* << *dhlab.api.nb_search_api*

get_json

```
from dhlab.api.nb_search_api import get_json
```

get_json(*phrases*, *mediatype='aviser'*)

home << *dhlab* << *dhlab.api.nb_search_api*

get_konks

```
from dhlab.api.nb_search_api import get_konks
```

get_konks(*urn*, *phrase*, *window=1000*, *n=1000*)

home << *dhlab* << *dhlab.api.nb_search_api*

get_phrase_info

```
from dhlab.api.nb_search_api import get_phrase_info
```

get_phrase_info(*urn*, *phrase*, *window=1000*, *n=1000*)

home << *dhlab* << *dhlab.api.nb_search_api*

iiif_manifest

```
from dhlab.api.nb_search_api import iiif_manifest
```

iiif_manifest(*urn*)

Fetch the IIIF manifest of the images that the book

home << *dhlab* << *dhlab.api.nb_search_api*

load_picture

```
from dhlab.api.nb_search_api import load_picture
```

load_picture(*url*)

Load the raw image object from a URL.

home << *dhlab* << *dhlab.api.nb_search_api*

mods

```
from dhlab.api.nb_search_api import mods
```

mods(*urn*)

home << *dhlab* << *dhlab.api.nb_search_api*

super_search

```
from dhlab.api.nb_search_api import super_search
```

super_search(*term*, *number=50*, *page=0*, *mediatype='bilder'*)

Search for a term, return a json object with the matches.

Parameters

- **term** (*str*) – Search term, word, or token. Default is the empty string.
- **number** (*int*) – Size of the response. Maximum 50.
- **page** (*int*) – The page number to search from. .. todo:: Fill in description.
- **mediatype** (*str*) – Default 'bilder'. .. todo:: Which other value options are available for this parameter?

Returns

a dict (json object) with the search results.

Return type

dict

home << *dhlab* << *dhlab.api.nb_search_api*

total_search

```
from dhlab.api.nb_search_api import total_search
```

total_search(*size=50*, *page=0*)

Retrieve the first `size` occurrences from `page`.

Wrapper-function for *super_search()* with arguments `medietype="bilder"` and `term=''`.

home << *dhlab*

3.1.2 ngram

```
from dhlab import ngram
```

Extraction of n-gram counts and aggregations.

Modules

dhlab.ngram.nb_ngram

dhlab.ngram.ngram

home << *dhlab* << *dhlab.ngram*

nb_ngram

```
from dhlab.ngram import nb_ngram
```

Functions

nb_ngram(terms[, corpus, smooth, years, ...])	Extract N-gram frequencies from given `terms` and `years`.
ngram_conv(ngrams[, smooth, years, mode])	Construct a dataframe with ngram mean frequencies per year over a given time period.
ngram_conv_old(ngrams[, smooth, years, mode])	Construct a dataframe with ngram mean frequencies per year over a given time period.

home << *dhlab* << *dhlab.ngram.nb_ngram*

nb_ngram

```
from dhlab.ngram.nb_ngram import nb_ngram
```

nb_ngram(*terms*, *corpus='bok'*, *smooth=1*, *years=(1810, 2010)*, *mode='relative'*, *lang='nob'*)

Extract N-gram frequencies from given `terms` and `years`. `lang` param is not supported for corpus=`avis` and will be set to None if `avis` is passed.

Parameters

- **terms** (*str*) – comma
- **corpus** (*str*) –
- **smooth** (*int*) –
- **years** (*tuple*) –
- **mode** (*str*) –
- **lang** (*str*) –

Returns

A sorted Pandas DataFrame index

home << *dhlab* << *dhlab.ngram.nb_ngram*

ngram_conv

```
from dhlab.ngram.nb_ngram import ngram_conv
```

ngram_conv(*ngrams*, *smooth=1*, *years=(1810, 2013)*, *mode='relative'*)

Construct a dataframe with ngram mean frequencies per year over a given time period.

Parameters

- **ngrams** – TODO: FIll in appropriate type and description.
- **smooth** – Smoothing factor for the graph visualisation.

- **years** – Tuple with start and end years for the time period of interest
- **mode** – Frequency measure. Defaults to 'relative'.

Returns
pandas dataframe with mean values for each year

home << *dhlab* << *dhlab.ngram.nb_ngram*

ngram_conv_old

```
from dhlab.ngram.nb_ngram import ngram_conv_old
```

ngram_conv_old(*ngrams*, *smooth=1*, *years=(1810, 2013)*, *mode='relative'*)

Construct a dataframe with ngram mean frequencies per year over a given time period.

Parameters

- **ngrams** – TODO: FIll in appropriate type and description.
- **smooth** (*int*) – Smoothing factor for the graph visualisation.
- **years** (*tuple*) – Tuple with start and end years for the time period of interest
- **mode** (*str*) – Frequency measure. Defaults to 'relative'.

Returns
pandas dataframe with mean values for each year

home << *dhlab* << *dhlab.ngram*

ngram

```
from dhlab.ngram import ngram
```

Classes

Ngram([words, from_year, to_year, doctype, ...])	Top level class for ngrams
NgramBook([words, title, publisher, city, ...])	Extract ngrams using metadata with functions to be inherited.
NgramNews([words, title, city, from_year, ...])	Ngram builder class.

home << *dhlab* << *dhlab.ngram.ngram*

Ngram

```
from dhlab.ngram.ngram import Ngram
```

class Ngram(*words=None*, *from_year=None*, *to_year=None*, *doctype='bok'*, *mode='relative'*, *lang='nob'*, ***kwargs*)

Bases: *DhlabObj*

Top level class for ngrams

Ngram builder class.

Build Ngrams from the National Librarys collections. Use with book corpus or newspaper corpus. Lang parameter is only supported for book (`bok`) corpus. Defaults to `None` if doctype is `avis`.

Parameters

- **words** (*str or list of str, optional*) – words to examine, defaults to None
- **from_year** (*int, optional*) – lower period cutoff, defaults to None
- **to_year** (*int, optional*) – upper period cutoff, defaults to None
- **doctype** (*str, optional*) – `bok` or `avis` , defaults to 'bok'
- **mode** (*str, optional*) – Frequency measure, defaults to 'relative'
- **lang** (*str, optional*) – `nob`, `nno`. Only use with docytype='bok', defaults to 'nob'
- ****kwargs** – Keyword arguments for Ngram._ipython_display_() Ngram.plot()

compare(*another_ngram*)

Divide one ngram by another - measures difference

plot(*smooth=4*, ***kwargs*)

Parameters

smooth – smoothing the curve

home << *dhlab* << *dhlab.ngram.ngram*

NgramBook

```
from dhlab.ngram.ngram import NgramBook
```

class NgramBook(*words=None*, *title=None*, *publisher=None*, *city=None*, *lang='nob'*, *from_year=None*, *to_year=None*, *ddk=None*, *subject=None*, ***kwargs*)

Bases: *Ngram*

Extract ngrams using metadata with functions to be inherited.

Create Dhlab Ngram from metadata

Parameters

- **words** (*str or list of str optional*) – words to examine, defaults to None
- **title** (*_type_, optional*) – _description_, defaults to None
- **publisher** (*_type_, optional*) – _description_, defaults to None
- **city** (*_type_, optional*) – _description_, defaults to None

- **lang** (*str, optional*) – _description_, defaults to 'nob'
- **from_year** (*_type_, optional*) – _description_, defaults to None
- **to_year** (*_type_, optional*) – _description_, defaults to None
- **ddk** (*_type_, optional*) – _description_, defaults to None
- **subject** (*_type_, optional*) – _description_, defaults to None

Returns
description

Return type
type

home << *dhlab* << *dhlab.ngram.ngram*

NgramNews

```
from dhlab.ngram.ngram import NgramNews
```

class NgramNews(*words=None*, *title=None*, *city=None*, *from_year=None*, *to_year=None*, ***kwargs*)

Bases: *Ngram*

Ngram builder class.

Build Ngrams from the National Librarys collections. Use with book corpus or newspaper corpus. Lang parameter is only supported for book (bok) corpus. Defaults to None if doctype is avis.

Parameters

- **words** (*str or list of str, optional*) – words to examine, defaults to None
- **from_year** (*int, optional*) – lower period cutoff, defaults to None
- **to_year** (*int, optional*) – upper period cutoff, defaults to None
- **doctype** (*str, optional*) – bok or avis , defaults to 'bok'
- **mode** (*str, optional*) – Frequency measure, defaults to 'relative'
- **lang** (*str, optional*) – nob, nno. Only use with docytype='bok', defaults to 'nob'
- ****kwargs** – Keyword arguments for Ngram._ipython_display_() Ngram.plot()

home << *dhlab*

3.1.3 text

```
from dhlab import text
```

Text analysis functionality, including building text corpora.

Modules

dhlab.text.chunking	
dhlab.text.conc_coll	
dhlab.text.corpus	
dhlab.text.dhlab_object	
dhlab.text.dispersion	
dhlab.text.geo_data	
dhlab.text.nb_search_text	
dhlab.text.nbtokenizer	Tokenisator for ngramleser (evt. parsing). Lars G Johnsen, Nasjonalbiblioteket, juni 2014
dhlab.text.parse	
dhlab.text.utils	
dhlab.text.wildcard	

home << *dhlab* << *dhlab.text*

chunking

```
from dhlab.text import chunking
```

Classes

Chunks([urn, chunks])	Create chunks from a text.

home << *dhlab* << *dhlab.text.chunking*

Chunks

```
from dhlab.text.chunking import Chunks
```

class Chunks(*urn=None*, *chunks=1000*)

Bases: `object`

Create chunks from a text.

Parameters

- **urn** – str or list

- **chunks** – { ‘para’, ‘avsn’ } or int

to_pandas()

Vectorize into a pandas dataframe with words a index

home << *dhlab* << *dhlab.text*

conc_coll

```
from dhlab.text import conc_coll
```

Functions

find_hits(x)	
make_link(row)	

home << *dhlab* << *dhlab.text.conc_coll*

find_hits

```
from dhlab.text.conc_coll import find_hits
```

find_hits(*x*)

home << *dhlab* << *dhlab.text.conc_coll*

make_link

```
from dhlab.text.conc_coll import make_link
```

make_link(*row*)

Classes

Collocations([corpus, words, before, after, ...])	Create collocations object
Concordance([corpus, query, window, limit])	Wrapper for concordance function
Counts([corpus, words])	Provide counts for a corpus - shouldn't be too large

home << *dhlab* << *dhlab.text.conc_coll*

Collocations

```
from dhlab.text.conc_coll import Collocations
```

class Collocations(*corpus=None*, *words=None*, *before=10*, *after=10*, *reference=None*, *samplesize=20000*, *alpha=False*, *ignore_caps=False*)

Bases: *DhlabObj*

Create collocations object

Parameters

- **corpus** (*dh.Corpus, optional*) – target corpus, defaults to None
- **words** (*str or list, optional*) – target words(s), defaults to None
- **before** (*int, optional*) – words to include before, defaults to 10
- **after** (*int, optional*) – words to include after, defaults to 10
- **reference** (*pd.DataFrame, optional*) – reference frequency list, defaults to None
- **samplesize** (*int, optional*) – _description_, defaults to 20000
- **alpha** (*bool, optional*) – Only include alphabetical tokens, defaults to False
- **ignore_caps** (*bool, optional*) – Ignore capitalized letters, defaults to False

classmethod from_df(*df*)

Typecast DataFrame to Collocation

Parameters
df – DataFrame

Returns
Collocation

keywordlist(*top=200*, *counts=5*, *relevance=10*)

show(*sortby='counts'*, *n=20*)

home << *dhlab* << *dhlab.text.conc_coll*

Concordance

```
from dhlab.text.conc_coll import Concordance
```

class Concordance(*corpus=None*, *query=None*, *window=20*, *limit=500*)

Bases: *DhlabObj*

Wrapper for concordance function

Get concordances for word(s) in corpus

Parameters

- **corpus** – Target corpus, defaults to None
- **query** – word or list or words, defaults to None
- **window** – how many tokens to consider around the target word, defaults to 20

- **limit** – limit returned hits, defaults to 500

show(*n=10*, *style=True*)

home << *dhlab* << *dhlab.text.conc_coll*

Counts

```
from dhlab.text.conc_coll import Counts
```

class Counts(*corpus=None*, *words=None*)

Bases: *DhlabObj*

Provide counts for a corpus - shouldn't be too large

Get frequency list for Corpus

Parameters

- **corpus** – target Corpus, defaults to None
- **words** – list of words to be counted, defaults to None

display_names()

Display data with record names as column titles.

classmethod from_df(*df*)

sum()

Summarize Corpus frequencies

Returns

frequency list for Corpus

home << *dhlab* << *dhlab.text*

corpus

```
from dhlab.text import corpus
```

Classes

Corpus([doctype, author, freetext, ...])	Class representing as DHLAB Corpus

home << *dhlab* << *dhlab.text.corpus*

Corpus

```
from dhlab.text.corpus import Corpus
```

class Corpus(*doctype=None*, *author=None*, *freetext=None*, *fulltext=None*, *from_year=None*, *to_year=None*, *from_timestamp=None*, *to_timestamp=None*, *title=None*, *ddk=None*, *subject=None*, *lang=None*, *limit=10*, *order_by='random'*)

Bases: *DhlabObj*

Class representing as DHLAB Corpus

Primary object for working with dhlab data. Contains references to texts in National Library's collections and metadata about them. Use with *coll*, *conc* or *freq* to analyse using dhlab tools.

add(*new_corpus*)

Utility for appending Corpus or DataFrame to self

coll(*words=None*, *before=10*, *after=10*, *reference=None*, *samplesize=20000*, *alpha=False*, *ignore_caps=False*)

Get collocations of `words` in corpus

conc(*words*, *window=20*, *limit=500*)

Get concodances of `words` in corpus

count(*words=None*)

Get word frequencies for corpus

evaluate_words(*wordbags=None*)

extend_from_identifiers(*identifiers=None*)

freq(*words=None*)

Get word frequencies for corpus

classmethod from_csv(*path*)

Import corpus from csv

classmethod from_df(*df*, *check_for_urn=False*)

Typecast Pandas DataFrame to Corpus class

DataFrame most contain URN column

classmethod from_identifiers(*identifiers*)

Construct Corpus from list of identifiers

sample(*n=5*)

Create random subkorpus with `n` entries

home << *dhlab* << *dhlab.text*

dhlab_object

```
from dhlab.text import dhlab_object
```

Classes

DhlabObj(frame)	DHLAB base class

home << *dhlab* << *dhlab.text.dhlab_object*

DhlabObj

```
from dhlab.text.dhlab_object import DhlabObj
```

class DhlabObj(*frame*)

Bases: object

DHLAB base class

Provides shared utility methods to DHLAB classes.

classmethod from_csv(*path*)

classmethod from_df(*df*)

head(*n=5*)

sort(*by=None*, *asc=False*)

tail(*n=5*)

to_csv(*path*)

to_excel(*path*)

home << *dhlab* << *dhlab.text*

dispersion

```
from dhlab.text import dispersion
```

Classes

Dispersion([urn, wordbag, window, pr])	Wrapper class for get_dispersion

home << *dhlab* << *dhlab.text.dispersion*

Dispersion

```
from dhlab.text.dispersion import Dispersion
```

class Dispersion(*urn=None*, *wordbag=None*, *window=1000*, *pr=100*)

Bases: object

Wrapper class for get_dispersion

Count occurrences of words in the given URN object.

Parameters

- **urn** (*str, optional*) – urn, defaults to None
- **wordbag** (*list, optional*) – words, defaults to None
- **window** (*int, optional*) – The number of tokens to search through per row, defaults to 1000

:param pr:, defaults to 100 :type pr: int, optional

plot(***kwargs*)

home << *dhlab* << *dhlab.text*

geo_data

```
from dhlab.text import geo_data
```

Classes

GeoData([urn, model])	Fetch place data from a text (book, newspaper or ...) identified by URN with an appropriate and available spacy model.
GeoNames(names[, feature_class, feature_code])	Fetch data from a list of names

home << *dhlab* << *dhlab.text.geo_data*

GeoData

```
from dhlab.text.geo_data import GeoData
```

class GeoData(*urn=None*, *model=None*)

Bases: object

Fetch place data from a text (book, newspaper or . . .) identified by URN with an appropriate and available spacy model.

The models are retrieved by instantiating Models.

add_geo_locations(*feature_class=None*, *feature_code=None*)

Get location data for the names in object, attribute self.place_names

home << *dhlab* << *dhlab.text.geo_data*

GeoNames

```
from dhlab.text.geo_data import GeoNames
```

class GeoNames(*names*, *feature_class=None*, *feature_code=None*)

Bases: `object`

Fetch data from a list of names

home << *dhlab* << *dhlab.text*

nb_search_text

```
from dhlab.text import nb_search_text
```

Functions

collocations_from_nb(word, corpus)	Get a concordance, and count the words in it.
count_from_conc(concordance)	From a concordance, count the words in it.
get_all_konks(term, urns)	
graph_from_df(df[, threshold])	
phrase_plots(phrase_sets[, title, fra, til, ...])	
phrase_plots_anno(phrase_sets[, title, fra, ...])	

home << *dhlab* << *dhlab.text.nb_search_text*

collocations_from_nb

```
from dhlab.text.nb_search_text import collocations_from_nb
```

collocations_from_nb(*word*, *corpus*)

Get a concordance, and count the words in it. Assume konks reside a dataframe with columns 'after' and 'before'

home << *dhlab* << *dhlab.text.nb_search_text*

count_from_conc

```
from dhlab.text.nb_search_text import count_from_conc
```

count_from_conc(*concordance*)

From a concordance, count the words in it. Assume konks reside a dataframe with columns 'after' and 'before'

home << *dhlab* << *dhlab.text.nb_search_text*

get_all_konks

```
from dhlab.text.nb_search_text import get_all_konks
```

get_all_konks(*term*, *urns*)

home << *dhlab* << *dhlab.text.nb_search_text*

graph_from_df

```
from dhlab.text.nb_search_text import graph_from_df
```

graph_from_df(*df*, *threshold=100*)

home << *dhlab* << *dhlab.text.nb_search_text*

phrase_plots

```
from dhlab.text.nb_search_text import phrase_plots
```

phrase_plots(*phrase_sets*, *title='aftenposten'*, *fra=1960*, *til=2020*, *step=5*, *rot=0*, *colours=['r', 'b', 'g', 'y', 'm', 'c']*)

home << *dhlab* << *dhlab.text.nb_search_text*

phrase_plots_anno

```
from dhlab.text.nb_search_text import phrase_plots_anno
```

phrase_plots_anno(*phrase_sets*, *title='aftenposten'*, *fra=1960*, *til=2020*, *rot=0*, *colours=['r', 'b', 'g']*)

home << *dhlab* << *dhlab.text*

nbtokenizer

```
from dhlab.text import nbtokenizer
```

Tokenisator for ngramleser (evt. parsing).

Lars G Johnsen, Nasjonalbiblioteket, juni 2014

Tokenisatorens oppgave er å danne token eller ord fra en sekvens med tegn. I utgangspunktet fungerer skilletegn og mellomrom som ordgrenser, men det er unntak, se listen nedenfor. Skilletegn danner som oftest egne token, men spesielt punktum og komma brukes på flere måter, noe det må tas høyde for.

Noen ord (token) har bestanddeler i form av skilletegn, som forkortelser, tall, i tillegg kan ordene selv være bundet sammen med bindestrek:

p-pille (bindestrek) 3.3 (punktum i seksjonsnummerering) etc. (forkortelser) 10 000 (token over mellomrom) 3,14 (desimaltall med komma) co2 (bokstaver og tall i kjemiske formler) co2-forurensning (bokstaver tall pluss bindestrek) 17. (ordenstall som i 17. mai) P. A. Munch (punktum i initialer) … tre eller flere punktum Når punktum følger tall vil tokenisatoren la punktum tilhøre tallet med mindre punktumet følges av mellomrom og stor bokstav.

Punktum tilhører alle forkortelser som tar punktum uavhenging av kontekst. Den kan imidlertid gjøres følsom for påfølgende stor bokstav, men det er altså ikke gjort her. Punktum tillates inne i ord og deler ikke opp ord med punktum i seg.

Alle skilletegn ellers utgjør egne token, bortsett fra § som kan sekvensieres, så § og §§ vil være egne tokener; de benyttes en hel del i lovtekster for entall og flertall.

Tall skrevet med mellomrom blir ett token om de er på formen xx xxx, altså 1 eller 3 siffer etterfulgt av grupper på tre siffer skilt med ett mellomrom. Så 3 1995 vil være to tokener, mens 3 995 blir ett token, 4000 398 blir igjen to token. (Mulig det er endret)

Tall som følger etter § (adskilt med maks ett mellomrom) vil aldri tiltrekke seg punktum.

Øvrige tegn som ikke passer inn med mønstrene over behandles som separate token.

Module Attributes

fork	Her er listen over forkortelser med punktum.
num	Tall som kan slutte på punktum består av hele tall, som tokeniseres
num0	F.eks.
num1	Seksjon 3.2.1 eller 2.3999, kan ikke ha sluttpunktum.
num2	3,5 kan ikke ha sluttpunktum.
num3	Det var .2 prosent økning.
num4	Tre eller flere punktum blir ett token.
num5	Tallord kombinert med ord, f.eks.
parnum0	Paragraftegn kan komme i en eller to (eller flere?) utgaver.
parnum	Tolk tall etter § som heltall uten punktum.
paragrafer	§ eller §§ brukes i lovtekster.
word	Ord er alt som ikke inneholder skillende skilletegn.
catchall	Alle tegn som ikke er et blankt tegn (tab, mellomrom linjeskift osv.), og som ikke er blitt matchet opp tidligere, blir å regne som egne token.
regex	Kombiner alle uttrykkene i rekkefølge og kompiler dem.

Functions

tokenize(text)	Tokenize the input `text` with the `regex` pattern.
tokenize_timer(text)	Time the *tokenize()* function and return the resulting tokens.

home << *dhlab* << *dhlab.text.nbtokenizer*

tokenize

```
from dhlab.text.nbtokenizer import tokenize
```

tokenize(*text*)

Tokenize the input `text` with the `regex` pattern.

home << *dhlab* << *dhlab.text.nbtokenizer*

tokenize_timer

```
from dhlab.text.nbtokenizer import tokenize_timer
```

tokenize_timer(*text*)

Time the *tokenize()* function and return the resulting tokens.

Classes

Tokens(text)	Create a list of tokens from a text with *tokenize()*.

home << *dhlab* << *dhlab.text.nbtokenizer*

Tokens

```
from dhlab.text.nbtokenizer import Tokens
```

class Tokens(*text*)

Bases: object

Create a list of tokens from a text with *tokenize()*.

home << *dhlab* << *dhlab.text*

parse

```
from dhlab.text import parse
```

Classes

Models()	Show the spaCy language models available
NER([urn, model, start_page, to_page])	Provide NER
POS([urn, model, start_page, to_page])	Provide POS and a parse

home << *dhlab* << *dhlab.text.parse*

Models

```
from dhlab.text.parse import Models
```

class Models

Bases: object

Show the spaCy language models available

home << *dhlab* << *dhlab.text.parse*

NER

```
from dhlab.text.parse import NER
```

class NER(*urn=None*, *model=None*, *start_page=0*, *to_page=0*)

Bases: object

Provide NER

home << *dhlab* << *dhlab.text.parse*

POS

```
from dhlab.text.parse import POS
```

class POS(*urn=None*, *model=None*, *start_page=0*, *to_page=0*)

Bases: object

Provide POS and a parse

home << *dhlab* << *dhlab.text*

utils

```
from dhlab.text import utils
```

Functions

urnlist(corpus)	Try to pull out a list of URNs from corpus

home << *dhlab* << *dhlab.text.utils*

urnlist

```
from dhlab.text.utils import urnlist
```

urnlist(*corpus*)

Try to pull out a list of URNs from corpus

home << *dhlab* << *dhlab.text*

wildcard

```
from dhlab.text import wildcard
```

Classes

WildcardWordSearch(word[, factor, ...])	Find a class of words matching a wildcard string

home << *dhlab* << *dhlab.text.wildcard*

WildcardWordSearch

```
from dhlab.text.wildcard import WildcardWordSearch
```

class WildcardWordSearch(*word*, *factor=2*, *freq_limit=10*, *limit=50*)

Bases: *DhlabObj*

Find a class of words matching a wildcard string

Parameters

word – word from a mixture of * and characters

Factor int

the additional length of words to be returned

Freq_lim

the frequency of returned words lower limit

Limit int

number of words returned

home << *dhlab* << dhlab.wordbank

3.1.4 wordbank

```
from dhlab.wordbank import wordbank
```

Classes

WordForm(words[, lang])	Fetch possible forms of a word or list of words
WordLemma(words[, lang])	Fetch possbile lemmas for a given word form
WordParadigm(words[, lang])	Fetch inflection paradigms for a list of words, or just one word
WordbankSuper(frame)	Super class for wordbank classes

home << *dhlab* << *dhlab.wordbank.wordbank*

WordForm

```
from dhlab.wordbank.wordbank import WordForm
```

class WordForm(*words*, *lang='nob'*)

Bases: *WordbankSuper*

Fetch possible forms of a word or list of words

home << *dhlab* << *dhlab.wordbank.wordbank*

WordLemma

```
from dhlab.wordbank.wordbank import WordLemma
```

class WordLemma(*words*, *lang='nob'*)

Bases: *WordbankSuper*

Fetch possbile lemmas for a given word form

home << *dhlab* << *dhlab.wordbank.wordbank*

WordParadigm

```
from dhlab.wordbank.wordbank import WordParadigm
```

class WordParadigm(*words*, *lang='nob'*)

Bases: *WordbankSuper*

Fetch inflection paradigms for a list of words, or just one word

home << *dhlab* << *dhlab.wordbank.wordbank*

WordbankSuper

```
from dhlab.wordbank.wordbank import WordbankSuper
```

class WordbankSuper(*frame*)

Bases: `object`

Super class for wordbank classes

home << *dhlab*

3.1.5 metadata

```
from dhlab import metadata
```

Metadata extraction.

Modules

dhlab.metadata.metadata	
dhlab.metadata.natbib	Tools for querying the Norwegian National Bibliography Marc 21

home << *dhlab* << *dhlab.metadata*

metadata

```
from dhlab.metadata import metadata
```

Functions

get_metadata([urns])	Fetch metadata from a list of urns.

home << *dhlab* << *dhlab.metadata.metadata*

get_metadata

```
from dhlab.metadata.metadata import get_metadata
```

get_metadata(*urns=None*)

Fetch metadata from a list of urns.

Parameters

urns (*list*) – uniform resource names, example: `["URN:NBN:no-nb_digibok_2011051112001", ...]`

home << *dhlab* << *dhlab.metadata*

natbib

```
from dhlab.metadata import natbib
```

Tools for querying the Norwegian National Bibliography Marc 21

Functions

api_call_deco(service)	Decorator for calling a service from DH-lab API
metadata_from_urn(*args, **kwargs)	
metadata_query(*args, **kwargs)	
pretty_print_marc21json(record)	Prints a record from the Norwegian National Bibliography in a readable format

home << *dhlab* << *dhlab.metadata.natbib*

api_call_deco

```
from dhlab.metadata.natbib import api_call_deco
```

api_call_deco(*service*)

Decorator for calling a service from DH-lab API

Parameters
service – Name of service

home << *dhlab* << *dhlab.metadata.natbib*

metadata_from_urn

```
from dhlab.metadata.natbib import metadata_from_urn
```

metadata_from_urn(**args*, ***kwargs*)

home << *dhlab* << *dhlab.metadata.natbib*

metadata_query

```
from dhlab.metadata.natbib import metadata_query
```

metadata_query(**args*, ***kwargs*)

home << *dhlab* << *dhlab.metadata.natbib*

pretty_print_marc21json

```
from dhlab.metadata.natbib import pretty_print_marc21json
```

pretty_print_marc21json(*record*)

Prints a record from the Norwegian National Bibliography in a readable format

Parameters
record – Marc 21 record in json format

home << *dhlab* << dhlab.images

3.1.6 nbpictures

```
from dhlab.images import nbpictures
```

Functions

display_finds(r)	A list of urls in r is displayed as HTML
find_urls(term[, number, page, mediatype])	generates urls from super_search for pictures
find_urls2(term[, number, page])	generates urls from super_search for pictures
find_urns(term)	From result of super_search, to be fed into iiif_manifest
get_metadata_from_url(url)	
get_picture_from_url(url[, width, height])	
get_picture_from_urn(urn[, width, height])	Fetch the Image object with its URN identifier.
get_url(urn[, page, part])	
json2html(meta)	
page_urn(urn[, page])	
pages(urn[, scale])	
total_urls([number, page])	find urls sequentially

home << *dhlab* << *dhlab.images.nbpictures*

display_finds

```
from dhlab.images.nbpictures import display_finds
```

display_finds(*r*)

A list of urls in r is displayed as HTML

home << *dhlab* << *dhlab.images.nbpictures*

find_urls

```
from dhlab.images.nbpictures import find_urls
```

find_urls(*term*, *number=50*, *page=0*, *mediatype='bilder'*)

generates urls from super_search for pictures

home << *dhlab* << *dhlab.images.nbpictures*

find_urls2

```
from dhlab.images.nbpictures import find_urls2
```

find_urls2(*term*, *number=50*, *page=0*)

generates urls from super_search for pictures

home << *dhlab* << *dhlab.images.nbpictures*

find_urns

```
from dhlab.images.nbpictures import find_urns
```

find_urns(*term*)

From result of super_search, to be fed into iiif_manifest

home << *dhlab* << *dhlab.images.nbpictures*

get_metadata_from_url

```
from dhlab.images.nbpictures import get_metadata_from_url
```

get_metadata_from_url(*url*)

home << *dhlab* << *dhlab.images.nbpictures*

get_picture_from_url

```
from dhlab.images.nbpictures import get_picture_from_url
```

get_picture_from_url(*url*, *width=0*, *height=300*)

home << *dhlab* << *dhlab.images.nbpictures*

get_picture_from_urn

```
from dhlab.images.nbpictures import get_picture_from_urn
```

get_picture_from_urn(*urn*, *width=0*, *height=300*)

Fetch the Image object with its URN identifier.

Parameters

- **urn** (*Union[int, str]*) – The uniform resource name number
- **width** (*int*) – Resolution width of the image.
- **height** (*int*) – Resolution height of the image.

Returns

An `Image` object.

home << *dhlab* << *dhlab.images.nbpictures*

get_url

```
from dhlab.images.nbpictures import get_url
```

get_url(*urn*, *page=1*, *part=200*)

home << *dhlab* << *dhlab.images.nbpictures*

json2html

```
from dhlab.images.nbpictures import json2html
```

json2html(*meta*)

home << *dhlab* << *dhlab.images.nbpictures*

page_urn

```
from dhlab.images.nbpictures import page_urn
```

page_urn(*urn*, *page=1*)

home << *dhlab* << *dhlab.images.nbpictures*

pages

```
from dhlab.images.nbpictures import pages
```

pages(*urn*, *scale=800*)

home << *dhlab* << *dhlab.images.nbpictures*

total_urls

```
from dhlab.images.nbpictures import total_urls
```

total_urls(*number=50*, *page=0*)

find urls sequentially

home << *dhlab*

3.1.7 visualize

```
from dhlab import visualize
```

Visualization tools

Modules

dhlab.visualize.cloud	
dhlab.visualize.df	Viz tools for for pandas DataFrame

home << *dhlab* << *dhlab.visualize*

cloud

```
from dhlab.visualize import cloud
```

Functions

cloud(df[, column, top, width, height, ...])	Make and draw a cloud from a pandas dataframe, using *make_cloud()* and *draw_cloud()*.
draw_cloud(sky[, width, height, fil])	Draw a word cloud produced by *make_cloud()*.
make_cloud(json_text[, top, background, ...])	Create a word cloud from a frequency list.

home << *dhlab* << *dhlab.visualize.cloud*

cloud

```
from dhlab.visualize.cloud import cloud
```

cloud(*df*, *column=''*, *top=200*, *width=1000*, *height=1000*, *background='black'*, *file=''*, *stretch=10*, *font_path=None*)

Make and draw a cloud from a pandas dataframe, using *make_cloud()* and *draw_cloud()*.

home << *dhlab* << *dhlab.visualize.cloud*

draw_cloud

```
from dhlab.visualize.cloud import draw_cloud
```

draw_cloud(*sky*, *width=20*, *height=20*, *fil=''*)

Draw a word cloud produced by *make_cloud()*.

home << *dhlab* << *dhlab.visualize.cloud*

make_cloud

```
from dhlab.visualize.cloud import make_cloud
```

make_cloud(*json_text*, *top=100*, *background='white'*, *stretch=<function <lambda>>*, *width=500*, *height=500*, *font_path=None*)

Create a word cloud from a frequency list.

home << *dhlab* << *dhlab.visualize*

df

```
from dhlab.visualize import df
```

Viz tools for for pandas DataFrame

Functions

heatmap(df[, color])	A wrapper for heatmap of a dataframe `df`.

home << *dhlab* << *dhlab.visualize.df*

heatmap

```
from dhlab.visualize.df import heatmap
```

heatmap(*df*, *color='green'*)

A wrapper for heatmap of a dataframe `df`.

3.2 Classes

Corpus([doctype, author, freetext, ...])	Class representing as DHLAB Corpus
Chunks([urn, chunks])	Create chunks from a text.
Collocations([corpus, words, before, after, ...])	Create collocations object
Concordance([corpus, query, window, limit])	Wrapper for concordance function
Counts([corpus, words])	Provide counts for a corpus - shouldn't be too large
GeoData([urn, model])	Fetch place data from a text (book, newspaper or ...) identified by URN with an appropriate and available spacy model.
GeoNames(names[, feature_class, feature_code])	Fetch data from a list of names
NER([urn, model, start_page, to_page])	Provide NER
POS([urn, model, start_page, to_page])	Provide POS and a parse
Models()	Show the spaCy language models available
WildcardWordSearch(word[, factor, ...])	Find a class of words matching a wildcard string
Ngram([words, from_year, to_year, doctype, ...])	Top level class for ngrams
NgramBook([words, title, publisher, city, ...])	Extract ngrams using metadata with functions to be inherited.
NgramNews([words, title, city, from_year, ...])	Ngram builder class.
WordParadigm(words[, lang])	Fetch inflection paradigms for a list of words, or just one word
WordLemma(words[, lang])	Fetch possbile lemmas for a given word form
WordForm(words[, lang])	Fetch possible forms of a word or list of words

home << *dhlab*

3.2.1 Corpus

```
from dhlab import Corpus
```

class Corpus(*doctype=None*, *author=None*, *freetext=None*, *fulltext=None*, *from_year=None*, *to_year=None*, *from_timestamp=None*, *to_timestamp=None*, *title=None*, *ddk=None*, *subject=None*, *lang=None*, *limit=10*, *order_by='random'*)

Bases: *DhlabObj*

Class representing as DHLAB Corpus

Primary object for working with dhlab data. Contains references to texts in National Library's collections and metadata about them. Use with *coll*, *conc* or *freq* to analyse using dhlab tools.

add(*new_corpus*)

Utility for appending Corpus or DataFrame to self

coll(*words=None*, *before=10*, *after=10*, *reference=None*, *samplesize=20000*, *alpha=False*, *ignore_caps=False*)

Get collocations of `words` in corpus

conc(*words*, *window=20*, *limit=500*)

Get concodances of `words` in corpus

count(*words=None*)

Get word frequencies for corpus

freq(*words=None*)

Get word frequencies for corpus

classmethod from_csv(*path*)

Import corpus from csv

classmethod from_df(*df*, *check_for_urn=False*)

Typecast Pandas DataFrame to Corpus class

DataFrame most contain URN column

classmethod from_identifiers(*identifiers*)

Construct Corpus from list of identifiers

sample(*n=5*)

Create random subkorpus with `n` entries

home << *dhlab*

3.2.2 Chunks

```
from dhlab import Chunks
```

class Chunks(*urn=None*, *chunks=1000*)

Bases: `object`

Create chunks from a text.

Parameters

- **urn** – str or list
- **chunks** – {'para', 'avsn'} or int

to_pandas()

Vectorize into a pandas dataframe with words a index

home << *dhlab*

3.2.3 Collocations

```
from dhlab import Collocations
```

class Collocations(*corpus=None*, *words=None*, *before=10*, *after=10*, *reference=None*, *samplesize=20000*, *alpha=False*, *ignore_caps=False*)

Bases: *DhlabObj*

Create collocations object

Parameters

- **corpus** (*dh.Corpus, optional*) – target corpus, defaults to None
- **words** (*str or list, optional*) – target words(s), defaults to None
- **before** (*int, optional*) – words to include before, defaults to 10
- **after** (*int, optional*) – words to include after, defaults to 10
- **reference** (*pd.DataFrame, optional*) – reference frequency list, defaults to None
- **samplesize** (*int, optional*) – _description_, defaults to 20000
- **alpha** (*bool, optional*) – Only include alphabetical tokens, defaults to False
- **ignore_caps** (*bool, optional*) – Ignore capitalized letters, defaults to False

classmethod from_df(*df*)

Typecast DataFrame to Collocation

Parameters
df – DataFrame

Returns
Collocation

home << *dhlab*

3.2.4 Concordance

```
from dhlab import Concordance
```

class Concordance(*corpus=None*, *query=None*, *window=20*, *limit=500*)

Bases: *DhlabObj*

Wrapper for concordance function

Get concordances for word(s) in corpus

Parameters

- **corpus** – Target corpus, defaults to None
- **query** – word or list or words, defaults to None
- **window** – how many tokens to consider around the target word, defaults to 20
- **limit** – limit returned hits, defaults to 500

home << *dhlab*

3.2.5 Counts

```
from dhlab import Counts
```

class Counts(*corpus=None*, *words=None*)

Bases: *DhlabObj*

Provide counts for a corpus - shouldn't be too large

Get frequency list for Corpus

Parameters

- **corpus** – target Corpus, defaults to None
- **words** – list of words to be counted, defaults to None

display_names()

Display data with record names as column titles.

sum()

Summarize Corpus frequencies

Returns
frequency list for Corpus

home << *dhlab*

3.2.6 GeoData

```
from dhlab import GeoData
```

class GeoData(*urn=None*, *model=None*)

Bases: `object`

Fetch place data from a text (book, newspaper or . . .) identified by URN with an appropriate and available spacy model.

The models are retrieved by instantiating *Models*.

add_geo_locations(*feature_class=None*, *feature_code=None*)

Get location data for the names in object, attribute self.place_names

home << *dhlab*

3.2.7 GeoNames

```
from dhlab import GeoNames
```

class GeoNames(*names*, *feature_class=None*, *feature_code=None*)

Bases: `object`

Fetch data from a list of names

home << *dhlab*

3.2.8 NER

```
from dhlab import NER
```

class NER(*urn=None*, *model=None*, *start_page=0*, *to_page=0*)

Bases: object

Provide NER

home << *dhlab*

3.2.9 POS

```
from dhlab import POS
```

class POS(*urn=None*, *model=None*, *start_page=0*, *to_page=0*)

Bases: object

Provide POS and a parse

home << *dhlab*

3.2.10 Models

```
from dhlab import Models
```

class Models

Bases: object

Show the spaCy language models available

home << *dhlab*

3.2.11 WildcardWordSearch

```
from dhlab import WildcardWordSearch
```

class WildcardWordSearch(*word*, *factor=2*, *freq_limit=10*, *limit=50*)

Bases: *DhlabObj*

Find a class of words matching a wildcard string

Parameters

word – word from a mixture of * and characters

Factor int

the additional length of words to be returned

Freq_lim

the frequency of returned words lower limit

Limit int

number of words returned

home << *dhlab*

3.2.12 Ngram

```
from dhlab import Ngram
```

class Ngram(*words=None*, *from_year=None*, *to_year=None*, *doctype='bok'*, *mode='relative'*, *lang='nob'*, ***kwargs*)

Bases: *DhlabObj*

Top level class for ngrams

Ngram builder class.

Build Ngrams from the National Librarys collections. Use with book corpus or newspaper corpus. Lang parameter is only supported for book (`bok`) corpus. Defaults to `None` if doctype is `avis`.

Parameters

- **words** (*str or list of str, optional*) – words to examine, defaults to None
- **from_year** (*int, optional*) – lower period cutoff, defaults to None
- **to_year** (*int, optional*) – upper period cutoff, defaults to None
- **doctype** (*str, optional*) – `bok` or `avis` , defaults to 'bok'
- **mode** (*str, optional*) – Frequency measure, defaults to 'relative'
- **lang** (*str, optional*) – `nob`, `nno`. Only use with docytype='bok', defaults to 'nob'
- ****kwargs** – Keyword arguments for Ngram._ipython_display_() Ngram.plot()

compare(*another_ngram*)

Divide one ngram by another - measures difference

plot(*smooth=4*, ***kwargs*)

Parameters

smooth – smoothing the curve

home << *dhlab*

3.2.13 NgramBook

```
from dhlab import NgramBook
```

class NgramBook(*words=None*, *title=None*, *publisher=None*, *city=None*, *lang='nob'*, *from_year=None*, *to_year=None*, *ddk=None*, *subject=None*, ***kwargs*)

Bases: *Ngram*

Extract ngrams using metadata with functions to be inherited.

Create Dhlab Ngram from metadata

Parameters

- **words** (*str or list of str optional*) – words to examine, defaults to None
- **title** (*_type_, optional*) – _description_, defaults to None
- **publisher** (*_type_, optional*) – _description_, defaults to None
- **city** (*_type_, optional*) – _description_, defaults to None

- **lang** (*str, optional*) – _description_, defaults to 'nob'
- **from_year** (*_type_, optional*) – _description_, defaults to None
- **to_year** (*_type_, optional*) – _description_, defaults to None
- **ddk** (*_type_, optional*) – _description_, defaults to None
- **subject** (*_type_, optional*) – _description_, defaults to None

Returns
description

Return type
type

home << *dhlab*

3.2.14 NgramNews

```
from dhlab import NgramNews
```

class NgramNews(*words=None*, *title=None*, *city=None*, *from_year=None*, *to_year=None*, ***kwargs*)

Bases: *Ngram*

Ngram builder class.

Build Ngrams from the National Librarys collections. Use with book corpus or newspaper corpus. Lang parameter is only supported for book (bok) corpus. Defaults to None if doctype is avis.

Parameters

- **words** (*str or list of str, optional*) – words to examine, defaults to None
- **from_year** (*int, optional*) – lower period cutoff, defaults to None
- **to_year** (*int, optional*) – upper period cutoff, defaults to None
- **doctype** (*str, optional*) – bok or avis , defaults to 'bok'
- **mode** (*str, optional*) – Frequency measure, defaults to 'relative'
- **lang** (*str, optional*) – nob, nno. Only use with docytype='bok', defaults to 'nob'
- ****kwargs** – Keyword arguments for Ngram._ipython_display_() Ngram.plot()

home << *dhlab*

3.2.15 WordParadigm

```
from dhlab import WordParadigm
```

class WordParadigm(*words*, *lang='nob'*)

Bases: *WordbankSuper*

Fetch inflection paradigms for a list of words, or just one word

home << *dhlab*

3.2.16 WordLemma

```
from dhlab import WordLemma
```

class WordLemma(*words*, *lang='nob'*)

Bases: *WordbankSuper*

Fetch possbile lemmas for a given word form

home << *dhlab*

3.2.17 WordForm

```
from dhlab import WordForm
```

class WordForm(*words*, *lang='nob'*)

Bases: *WordbankSuper*

Fetch possible forms of a word or list of words

3.3 Functions

totals	Get aggregated raw frequencies of all words in the National Library's database.

home << *dhlab*

3.3.1 totals

```
from dhlab import totals
```

totals(*top_words=50000*)

Get aggregated raw frequencies of all words in the National Library's database.

Call the API *BASE_URL* endpoint /totals/{top_words}.

Parameters
top_words (*int*) – The number of words to get total frequencies for.

Returns
a `pandas.DataFrame` with the most frequent words.

Return type
DataFrame

3.4 API endpoints

`BASE_URL = 'https://api.nb.no/dhlab'`

REST-API URL adress to fulltext query functions

`NGRAM_API = 'https://api.nb.no/dhlab/nb_ngram/ngram/query'`

URL adress for API calls to ngram-databases

`GALAXY_API = 'https://api.nb.no/dhlab/nb_ngram_galaxies/galaxies/query'`

URL adress for word galaxy API queries

CHAPTER

FOUR

GLOSSARY

IDE
: Integrated Development Environment. Read more on wikipedia.

API
: Application Programming Interface. Read more on wikipedia.

n-gram
: An n-gram is a sequence of linguistic units, typically words or characters.

  Depending on the length n of the 'gram' sequence, we call them unigrams for single tokens, bigrams for sequences of two, trigrams for three, etc.

  As we count the occurrences of these n-grams across a large body of text, called a corpus, we can view patterns of the rhetoric in that corpus. If we additionally can spread samples of the corpus over time, we can see how the use of language develops in that time frame.

  The NB N-gram app offers a visual graph of all uni-, bi-, and trigrams in the digital collection of all Norwegian published textual material (books, newspapers, magazines) between 1810 and 2021.

  The `Ngram`, `NgramBook`, `NgramNews` classes extract ngrams and their frequency lists.

CHAPTER

FIVE

CHANGELOG

5.1 v2.21.0 (2023-02-13)

5.2 v2.20.0 (2023-02-09)

5.2.1 Feat

- **metadata**: update metadata service tools

5.3 v2.19.0 (2023-01-25)

5.3.1 Feat

- added urncount - removed wordcloud message

5.4 v2.18.0 (2023-01-18)

5.4.1 Feat

- **text**: (#133)

5.5 v2.17.0 (2023-01-16)

5.5.1 Feat

- added wildcard search for words

5.6 v2.16.0 (2023-01-16)

5.6.1 Feat

- **heatmap**: add heatmap wrapper to viz package

5.7 v2.15.0 (2023-01-16)

5.7.1 Feat

- **corpus**: (#128)

5.7.2 Fix

- **bump-version**: disable triggering new workflow [skip-ci] (#127)

5.8 v2.14.0 (2023-01-13)

5.8.1 Feat

- **visualize, ngram, wordbank**: (#126)

5.9 v2.13.0 (2022-12-07)

5.9.1 Feat

- **api**: added start page and to page for NER and POS (#122)

5.10 v2.12.0 (2022-12-07)

5.10.1 Feat

- **ngram**: make ngram subclass of dhlabobj (#121)

5.11 v2.10.0 (2022-10-26)

5.11.1 Feat

- added image search in bokhylla books to api

5.12 v2.9.0 (2022-09-05)

5.12.1 Feat

- added reference for words

5.13 v2.8.0 (2022-08-30)

5.13.1 Feat

- added spacy pos parse (pos, lemma, dependency)

5.13.2 Refactor

- **corpus**: ignore index in corpus.add (#84)

5.14 v2.7.0 (2022-08-24)

- automatic bump with features from 2.6.x versions

5.15 v2.6.0 (2022-08-08)

5.15.1 Feat

- code for word evaluations

5.16 v2.5.0 (2022-08-05)

5.16.1 Feat

- geolocation

5.17 v2.4.0 (2022-07-12)

5.17.1 Feat

- ner with spaCy

5.18 v2.3.0 (2022-06-02)

5.18.1 Feat

- added access to Norsk Ordbank, wordbank

5.19 v2.2.0 (2022-05-13)

5.19.1 Feat

- ngram, geodata

5.20 v2.1.0 (2022-05-13)

5.20.1 Feat

- geodata

5.21 v1.0.0 (2022-01-06)

- Set up Github Actions to run automatic linting and testing
- Set up documentation pages
- Include documentation of the code in docstrings

5.21.1 Fix

- address linting issues from flake8
- reformat code

5.21.2 Feat

- add documentation summaries for all modules
- add documentation for the repo
- add docstrings from README.md to nbtext.py
- add pylint config file

5.21.3 Refactor

- reduce code duplication
- update workflow file reference
- change str.format to f-strings
- optimize imports
- rename workflow that packages and publishes dhlab to pypi
- use default publish workflow
- reduce compatible python versions
- update publishing workflow
- type out scope for linting explicitly
- move pylint.yml

5.22 v0.75 (2019-09-09)

- Inital release to pypi

PYTHON MODULE INDEX

d

INDEX

E

F

G

H

I

J

K

L

M

N

P

R

S

T

U

W