

CWordTM Toolkit Usage on BBC News - Alpha Version

This Jupyter notebook demonstrates how to use the package "CWordTM" on the BBC News:

1. Meta Information Features
2. Utility Features
3. Text Visualization - Word Cloud
4. Text Summarization
5. Topic Modeling - LDA, BERTopic and NMF

CWordTM Toolkit's Documentation: <https://cwordtm.readthedocs.io>

```
In [1]: import warnings
warnings.filterwarnings('ignore')
```

1. Meta Information Features

```
In [2]: import cwordtm
from cwordtm import *
```

```
In [3]: cwordtm.__version__
```

```
Out[3]: '0.6.4'
```

```
In [4]: # Show brief module information
print(meta.get_module_info())
```

The member information of the module 'cwordtm'

1. Submodule meta:
 - addin (func, *, timing=False, code=0)
 - addin_all (modname='cwordtm', *, timing=False, code=0)
 - addin_all_functions (submod, *, timing=False, code=0)
 - get_function (mod_name, submodules, func_name, *, timing=False, code=0)
 - get_module_info (detailed=False, *, timing=False, code=0)
 - get_submodule_info (submodname, detailed=False, *, timing=False, code=0)
 - import_module (name, package=None, *, timing=False, code=0)
 - wraps (wrapped, assigned=('__module__', '__name__', '__qualname__', '__doc__', '__annotations__'), updated=('__dict__',), *, timing=False, code=0)
2. Submodule pivot:
 - pivot (df, column='Category', *, timing=False, code=0)
 - stat (df, chi=False, *, timing=False, code=0)
3. Submodule quot:
 - extract_quotation (text, quot_marks, *, timing=False, code=0)
 - match_text (target, sent_tokens, lang, threshold, n=5, *, timing=False, code=0)
 - match Verse (i, ot_list, otdf, df, book, chap, verse, lang, threshold, *, timing=False, code=0)
 - show_ quot (target, source='ot', lang='en', threshold=0.5, *, timing=False, code=0)
 - tokenize (sentence, *, timing=False, code=0)
4. Submodule ta:
 - get_sent_scores (sentences, diction, sent_len, *, timing=False, code=0) -> dict
 - get_sentences (docs, lang='en', *, timing=False, code=0)
 - get_summary (sentences, sent_weight, threshold, sent_len, *, timing=False, code=0)
 - pos_tag (tokens, tagset=None, lang='eng', *, timing=False, code=0)
 - preprocess_sent (text, *, timing=False, code=0)
 - sent_tokenize (text, language='english', *, timing=False, code=0)
 - summary_chi (docs, weight=1.5, sent_len=8, *, timing=False, code=0)
 - summary_en (docs, sent_len=8, *, timing=False, code=0)
 - word_tokenize (text, language='english', preserve_line=False, *, timing=False, code=0)
5. Submodule tm:
 - BTM (textfile, num_topics, chi=False, embed=True)
 - LDA (textfile, num_topics, chi=False)
 - NMF (textfile, num_topics, chi=False)
 - btm_process (doc_file, num_topics=10, source=0, text_col='text', cat=0, chi=False, group=True, eval=False, *, timing=False, code=0)
 - lda_process (doc_file, num_topics=10, source=0, text_col='text', cat=0, chi=False, group=True, eval=False, *, timing=False, code=0)
 - load_bible (textfile, cat=0, group=True, *, timing=False, code=0)
 - load_text (textfile, text_col='text', *, timing=False, code=0)
 - ngrams (sequence, n, *, timing=False, code=0, **kwargs)
 - nmf_process (doc_file, num_topics=10, source=0, text_col='text', cat=0, chi=False, group=True, eval=False, *, timing=False, code=0)
 - pprint (object, stream=None, indent=1, width=80, depth=None, *, compact=False, sort_dicts=True, underscore_numbers=False, timing=False, code=0)
 - process_text (doc, *, timing=False, code=0)
6. Submodule util:
 - add_chi_vocab (*, timing=False, code=0)
 - bible_cat_info (lang='en', *, timing=False, code=0)
 - chi_sent_terms (text, *, timing=False, code=0)
 - chi_stops (*, timing=False, code=0)
 - clean_sentences (sentences, *, timing=False, code=0)
 - clean_text (df, text_col='text', *, timing=False, code=0)
 - extract (df, testament=-1, category='', book=0, chapter=0, verse=0, *, timing=False, code=0)
 - extract2 (df, filter='', *, timing=False, code=0)
 - get_diction (docs, *, timing=False, code=0)
 - get_diction_chi (docs, *, timing=False, code=0)
 - get_diction_en (docs, *, timing=False, code=0)
 - get_list (df, column='book', *, timing=False, code=0)
 - get_sent_terms (text, *, timing=False, code=0)
 - get_text (df, text_col='text', *, timing=False, code=0)
 - get_text_list (df, text_col='text', *, timing=False, code=0)
 - group_text (df, column='chapter', *, timing=False, code=0)
 - is_chi (*, timing=False, code=0)
 - load_text (filepath, nr=0, info=False, *, timing=False, code=0)
 - load_word (ver='web.csv', nr=0, info=False, *, timing=False, code=0)
 - preprocess_text (text, *, timing=False, code=0)
 - remove_noise (text, noise_list, *, timing=False, code=0)
 - reset_rows (*, timing=False, code=0)
 - set_lang (lang='en', *, timing=False, code=0)
 - set_rows (n=None, *, timing=False, code=0)
 - word_tokenize (text, language='english', preserve_line=False, *, timing=False, code=0)
7. Submodule version:
8. Submodule viz:
 - chi_wordcloud (docs, figsize=(15, 10), bg='white', image=0, *, timing=False, code=0)
 - plot_cloud (wordcloud, figsize, *, timing=False, code=0)
 - show_wordcloud (docs, clean=False, figsize=(12, 8), bg='white', image=0, *, timing=False, code=0)

```
In [5]: # Show detailed module information of a submodule
print(meta.get_submodule_info("viz", detailed=True))
```

```

The function(s) of the submodule 'cwordtm.viz':

def chi_wordcloud(docs, figsize=(15, 10), bg='white', image=0):
    """Prepare and show a Chinese wordcloud

    :param docs: The collection of Chinese documents for preparing a wordcloud,
        default to None
    :type docs: pandas.DataFrame
    :param figsize: Size (width, height) of word cloud, default to (15, 10)
    :type figsize: tuple, optional
    :param bg: The background color (name) of the wordcloud, default to 'white'
    :type bg: str, optional
    :param image: The filename of the prescribed image as the mask of the wordcloud,
        or 1/2/3/4 for using an internal image (heart / disc / triangle / arrow),
        default to 0 (No image mask)
    :type image: int or str, optional
    """

    util.set_lang('chi')
    diction = util.get_diction(docs)

    masks = ['heart.jpg', 'disc.jpg', 'triangle.jpg', 'arrow.jpg']

    if image == 0:
        mask = None
    elif image in [1, 2, 3, 4]: # Internal image file
        img_file = files('cwordtm.images').joinpath(masks[image-1])
        mask = np.array(Image.open(img_file))
    elif isinstance(image, str) and len(image) > 0:
        mask = np.array(Image.open(image))
    else:
        mask = None

    font_file = files('cwordtm.data').joinpath('msyh.ttc')
    wordcloud = WordCloud(background_color=bg, colormap='Set2',
                          mask=mask, font_path=str(font_file)) \
        .generate_from_frequencies(frequencies=diction)

    plot_cloud(wordcloud, figsize=figsize)

def plot_cloud(wordcloud, figsize):
    """Plot the prepared 'wordcloud'

    :param wordcloud: The WordCloud object for plotting, default to None
    :type wordcloud: WordCloud object
    :param figsize: Size (width, height) of word cloud, default to None
    :type figsize: tuple
    """

    plt.figure(figsize=figsize)
    plt.imshow(wordcloud)
    plt.axis("off");

def show_wordcloud(docs, clean=False, figsize=(12, 8), bg='white', image=0):
    """Prepare and show a wordcloud

    :param docs: The collection of documents for preparing a wordcloud,
        default to None
    :type docs: pandas.DataFrame
    :param clean: The flag whether text preprocessing is needed,
        default to False
    :type clean: bool, optional
    :param figsize: Size (width, height) of word cloud, default to (12, 8)
    :type figsize: tuple, optional
    :param bg: The background color (name) of the wordcloud, default to 'white'
    :type bg: str, optional
    :param image: The filename of the prescribed image as the mask of the wordcloud,
        or 1/2/3/4 for using an internal image (heart / disc / triangle / arrow),
        default to 0 (No image mask)
    :type image: int or str, optional
    """

    masks = ['heart.jpg', 'disc.jpg', 'triangle.jpg', 'arrow.jpg']

    if image == 0:
        mask = None
    elif image in [1, 2, 3, 4]: # Internal image file
        img_file = files('cwordtm.images').joinpath(masks[image-1])
        mask = np.array(Image.open(img_file))
    elif isinstance(image, str) and len(image) > 0:
        mask = np.array(Image.open(image))
    else:
        mask = None

    if isinstance(docs, pd.DataFrame):
        docs = ' '.join(list(docs.text.astype(str)))
    elif isinstance(docs, pd.Series):

```

```

docs = ' '.join(list(docs.astype(str)))
elif isinstance(docs, list) or isinstance(docs, np.ndarray):
    docs = ' '.join(str(doc) for doc in docs)

if clean:
    docs = util.preprocess_text(docs)

wordcloud = WordCloud(background_color=bg, colormap='Set2', mask=mask) \
    .generate(docs)

plot_cloud(wordcloud, figsize=figsize)

```

```

In [6]: # Show execution time
bbc_news = "BBC/BBC News Train.csv"
df = util.load_text(bbc_news, timing=True)

```

Finished 'load_text' in 0.0577 secs

```

In [7]: # Execute and show code
df = util.load_text(bbc_news, code=1)

def load_text(filepath, nr=0, info=False):
    """Loads and returns the text from the prescribed file path ('filepath').

    :param filepath: The prescribed filepath from which the text is loaded,
        default to None
    :type filepath: str
    :param nr: The number of rows of text to be loaded; 0 represents all rows,
        default to 0
    :type nr: int, optional
    :param info: The flag whether the dataset information is shown,
        default to False
    :type info: bool, optional
    :return: The collection of text with the prescribed number of rows loaded
    :rtype: pandas.DataFrame
    """

    # print("Loading file '%s' ..." %filepath)
    if filepath.lower().endswith('csv'):
        nrows = None
        if nr > 0: nrows = nr
        df = pd.read_csv(filepath, nrows=nrows, encoding='utf-8')
    else:
        noise_list = ['\u3000', '- ', '•']
        tf = open(filepath, encoding='utf-8')
        lines = [remove_noise(line, noise_list) for line in tf.readlines()]
        lines = list(filter(None, lines))

        df = pd.DataFrame({'text': lines})
        if nr > 0: df = df.iloc[:nr]

    if info:
        print("\nDataset Information:")
        df.info()

    return df

>> cwordtm.util.remove_noise
def remove_noise(text, noise_list):
    """Removes a list of substrings in noise_list from the input text.

    :param text: The input text, default to None
    :type text: str
    :param noise_list: The list of substrings to be removed, default to ""
    :type noise_list: list, optional
    :return: The text with the prescribed substrings removed
    :rtype: str
    """

    text = text.rstrip()
    for noise in noise_list:
        text = text.replace(noise, '')
    return text

```

```

In [8]: # Show code without execution
df = util.load_text(bbc_news, code=2)

```

```

def load_text(filepath, nr=0, info=False):
    """Loads and returns the text from the prescribed file path ('filepath').

    :param filepath: The prescribed filepath from which the text is loaded,
        default to None
    :type filepath: str
    :param nr: The number of rows of text to be loaded; 0 represents all rows,
        default to 0
    :type nr: int, optional
    :param info: The flag whether the dataset information is shown,
        default to False
    :type info: bool, optional
    :return: The collection of text with the prescribed number of rows loaded
    :rtype: pandas.DataFrame
    """

    # print("Loading file '%s' ..." %filepath)
    if filepath.lower().endswith('csv'):
        nrows = None
        if nr > 0: nrows = nr
        df = pd.read_csv(filepath, nrows=nrows, encoding='utf-8')
    else:
        noise_list = ['\u3000', '- ', '•']
        tf = open(filepath, encoding='utf-8')
        lines = [remove_noise(line, noise_list) for line in tf.readlines()]
        lines = list(filter(None, lines))

        df = pd.DataFrame({'text': lines})
        if nr > 0: df = df.iloc[:nr]

    if info:
        print("\nDataset Information:")
        df.info()

    return df

>> cwordtm.util.remove_noise
def remove_noise(text, noise_list):
    """Removes a list of substrings in noise_list from the input text.

    :param text: The input text, default to None
    :type text: str
    :param noise_list: The list of substrings to be removed, default to ""
    :type noise_list: list, optional
    :return: The text with the prescribed substrings removed
    :rtype: str
    """

    text = text.rstrip()
    for noise in noise_list:
        text = text.replace(noise, '')
    return text

```

```

In [9]: # Add timing and code reveal features to some other function
from importlib_resources import files
files = meta.addin(files)
files(code=2)

```

```

@package_to_anchor
def files(anchor: Optional[Anchor] = None) -> Traversable:
    """
    Get a Traversable resource for an anchor.
    """
    return from_package(resolve(anchor))

```

2. Utility Features

Load BBC News

```

In [10]: df = util.load_text(bbc_news, info=True)

```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1490 entries, 0 to 1489
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   ArticleId   1490 non-null   int64
1   Text        1490 non-null   object
2   Category    1490 non-null   object
dtypes: int64(1), object(2)
memory usage: 35.0+ KB
```

Preprocessing Text

```
In [11]: text_list = util.get_text_list(df.iloc[:500], text_col='Text')
text = util.preprocess_text(text_list)
```

3. Text Visualization - Word Cloud

```
In [12]: # White background with no image mask
viz.show_wordcloud(text)
```

```
D:\Dev\Anaconda3\lib\site-packages\wordcloud\wordcloud.py:106: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  self.colormap = plt.cm.get_cmap(colormap)
```



```
In [13]: # Black background with the prescribed image as the mask
viz.show_wordcloud(text, bg='black', image='images/disc.png')
```

```
D:\Dev\Anaconda3\lib\site-packages\wordcloud\wordcloud.py:106: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  self.colormap = plt.cm.get_cmap(colormap)
```



```

Corpus loaded!
Text preprocessed!
Text trained!
If no visualization is shown,
you may execute the following commands to show the visualization:
> import pyLDAvis
> pyLDAvis.display(lda.vis_data)
Visualization prepared!

Topics from LDA Model:
[(0,
 '0.006*said" + 0.004*ha" + 0.004*wa" + 0.003*year" + 0.002*best" + '
 '0.002*new" + 0.001*mr" + 0.001*award" + 0.001*film" + 0.001*people'),
 (1,
 '0.008*said" + 0.006*wa" + 0.004*ha" + 0.003*mr" + 0.003*film" + '
 '0.002*people" + 0.002*year" + 0.002*new" + 0.001*time" + 0.001*fir'),
 (2,
 '0.004*said" + 0.004*wa" + 0.003*ha" + 0.002*year" + 0.002*game" + '
 '0.002*new" + 0.002*mobile" + 0.002*service" + 0.001*england" + '
 '0.001*phone'),
 (3,
 '0.006*said" + 0.004*wa" + 0.003*ha" + 0.002*year" + 0.002*new" + '
 '0.002*game" + 0.002*people" + 0.001*number" + 0.001*time" + 0.001*mr'),
 (4,
 '0.006*said" + 0.005*wa" + 0.005*ha" + 0.002*year" + 0.002*new" + '
 '0.001*mr" + 0.001*time" + 0.001*game" + 0.001*uk" + 0.001*fir'),
 (5,
 '0.008*said" + 0.005*wa" + 0.005*mr" + 0.004*ha" + 0.002*year" + '
 '0.002*people" + 0.002*new" + 0.002*government" + 0.002*blair" + '
 '0.001*minister'),
 (6,
 '0.005*wa" + 0.005*said" + 0.005*ha" + 0.002*year" + 0.002*film" + '
 '0.002*people" + 0.002*game" + 0.001*mr" + 0.001*say" + 0.001*time'),
 (7,
 '0.005*said" + 0.005*mr" + 0.005*wa" + 0.003*ha" + 0.003*party" + '
 '0.003*election" + 0.002*labour" + 0.002*tory" + 0.002*blair" + '
 '0.002*tax'),
 (8,
 '0.006*wa" + 0.005*said" + 0.004*ha" + 0.003*year" + 0.001*people" + '
 '0.001*new" + 0.001*world" + 0.001*england" + 0.001*time" + '
 '0.001*film'),
 (9,
 '0.004*mr" + 0.004*said" + 0.004*wa" + 0.003*ha" + 0.002*labour" + '
 '0.002*election" + 0.002*new" + 0.002*year" + 0.001*game" + '
 '0.001*people')]

Model Evaluation Scores:
Coherence: 0.6749119020453799
Perplexity: -11.23703685935349
Topic diversity: 0.0007351396765385262
Topic size distribution: 0.0024154589371980675

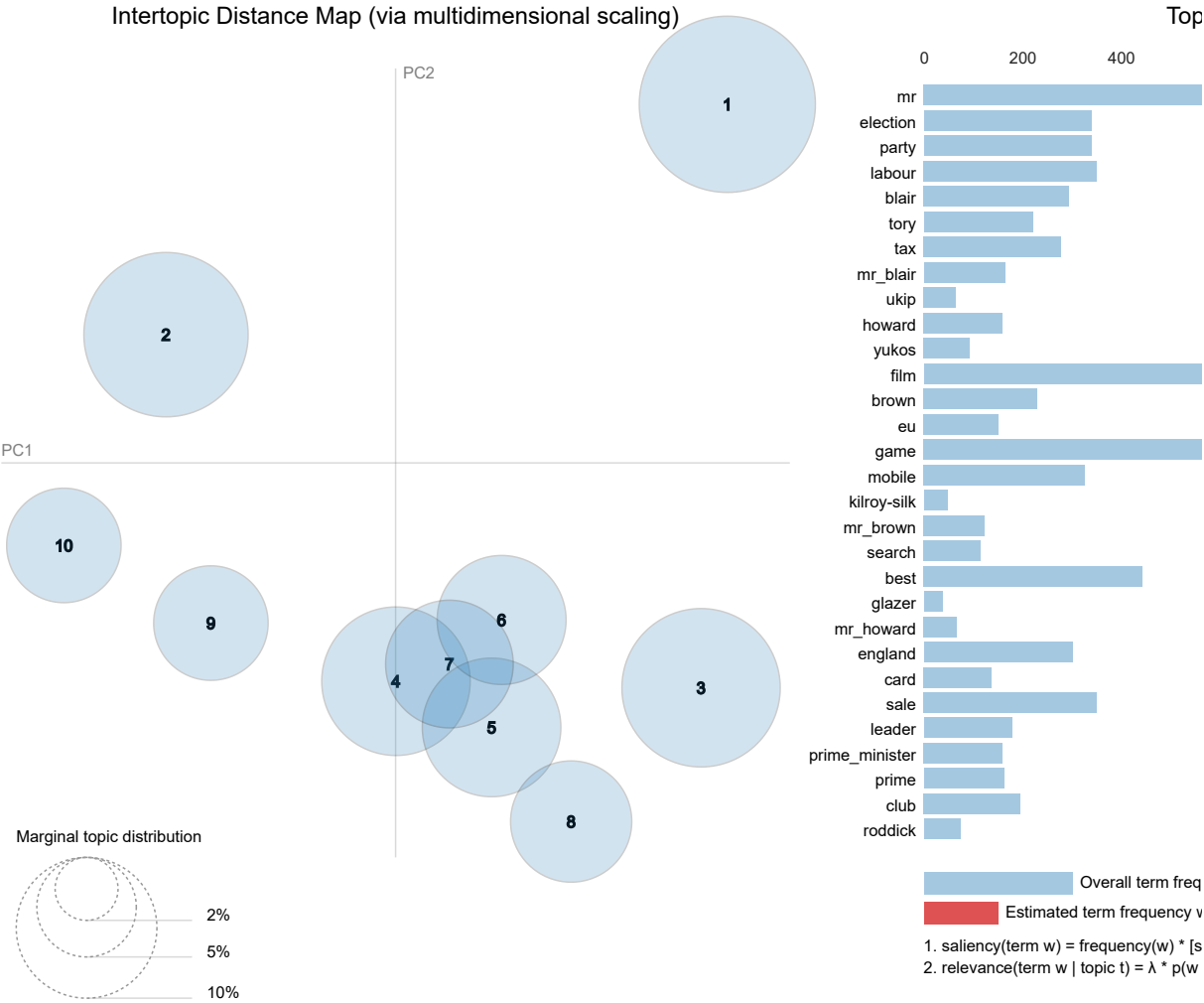
```

Finished 'lda_process' in 65.2418 secs

```

In [17]: # LDA Model Visualization
import pyLDAvis
pyLDAvis.display(lda.vis_data)

```

Save LDA Model

```
In [18]: lda.save("models/lda_bbc.gensim")
```

LDA model has been stored in 'models/lda_bbc.gensim'.

Load LDA Model

```
In [19]: lda2 = tm.LDA("", lda.num_topics)
lda2.model = lda2.load("models/lda_bbc.gensim")
lda2.show_topics()
```

Topics from LDA Model:

```
[(0,
  '0.006*"said" + 0.004*"ha" + 0.004*"wa" + 0.003*"year" + 0.002*"best" + '
  '0.002*"new" + 0.001*"mr" + 0.001*"award" + 0.001*"film" + 0.001*"people"'),
 (1,
  '0.008*"said" + 0.006*"wa" + 0.004*"ha" + 0.003*"mr" + 0.003*"film" + '
  '0.002*"people" + 0.002*"year" + 0.002*"new" + 0.001*"time" + 0.001*"firm"'),
 (2,
  '0.004*"said" + 0.004*"wa" + 0.003*"ha" + 0.002*"year" + 0.002*"game" + '
  '0.002*"new" + 0.002*"mobile" + 0.002*"service" + 0.001*"england" + '
  '0.001*"phone"'),
 (3,
  '0.006*"said" + 0.004*"wa" + 0.003*"ha" + 0.002*"year" + 0.002*"new" + '
  '0.002*"game" + 0.002*"people" + 0.001*"number" + 0.001*"time" + 0.001*"mr"'),
 (4,
  '0.006*"said" + 0.005*"wa" + 0.005*"ha" + 0.002*"year" + 0.002*"new" + '
  '0.001*"mr" + 0.001*"time" + 0.001*"game" + 0.001*"uk" + 0.001*"firm"'),
 (5,
  '0.008*"said" + 0.005*"wa" + 0.005*"mr" + 0.004*"ha" + 0.002*"year" + '
  '0.002*"people" + 0.002*"new" + 0.002*"government" + 0.002*"blair" + '
  '0.001*"minister"'),
 (6,
  '0.005*"wa" + 0.005*"said" + 0.005*"ha" + 0.002*"year" + 0.002*"film" + '
  '0.002*"people" + 0.002*"game" + 0.001*"mr" + 0.001*"say" + 0.001*"time"'),
 (7,
  '0.005*"said" + 0.005*"mr" + 0.005*"wa" + 0.003*"ha" + 0.003*"party" + '
  '0.003*"election" + 0.002*"labour" + 0.002*"tory" + 0.002*"blair" + '
  '0.002*"tax"'),
 (8,
  '0.006*"wa" + 0.005*"said" + 0.004*"ha" + 0.003*"year" + 0.001*"people" + '
  '0.001*"new" + 0.001*"world" + 0.001*"england" + 0.001*"time" + '
  '0.001*"film"'),
 (9,
  '0.004*"mr" + 0.004*"said" + 0.004*"wa" + 0.003*"ha" + 0.002*"labour" + '
  '0.002*"election" + 0.002*"new" + 0.002*"year" + 0.001*"game" + '
  '0.001*"people"')]
```

BERTopic Modeling

```
In [20]: btm = tm.btm_process(bbc_news, source=1, text_col='Text', eval=True, timing=True)
```

Corpus loaded!

Text preprocessed!

Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.predictions.bias', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.decoder.weight', 'cls.seq_relationship.weight']

- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).

- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Text trained!

Topics from BERTopic Model:

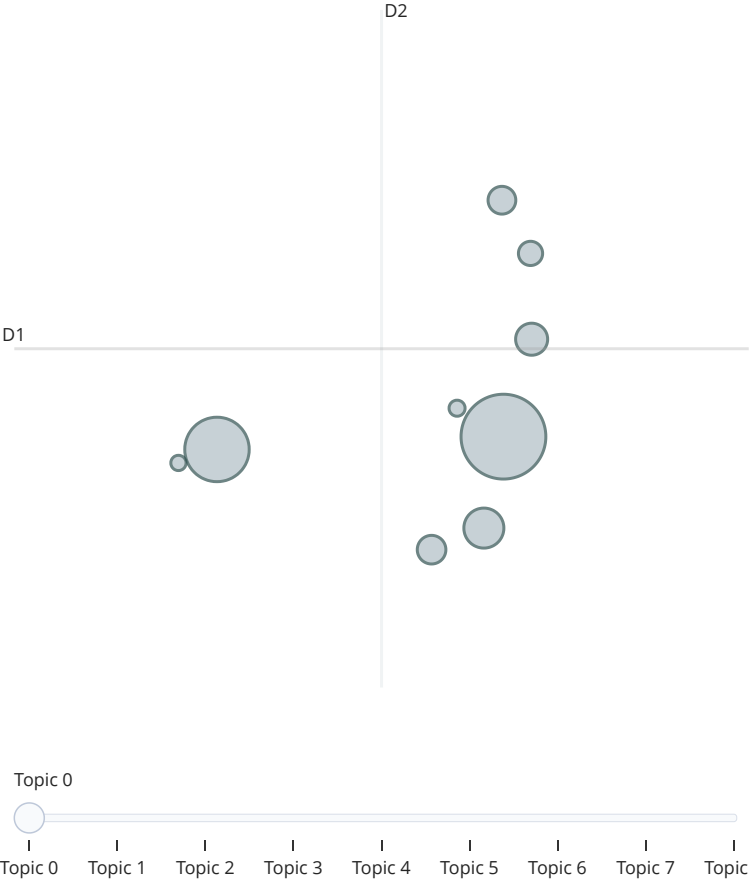
```
Topic 0: said | mr | wa | ha | year | government | election | bn | labour | party
Topic 1: wa | game | year | ha | england | said | win | time | half | player
Topic 2: music | band | wa | album | said | chart | year | song | singer | single
Topic 3: mail | virus | said | site | user | file | software | search | microsoft | firm
Topic 4: film | best | award | oscar | actor | wa | star | director | actress | aviator
Topic 5: game | gadget | console | technology | said | pc | digital | ha | wa | device
Topic 6: phone | mobile | broadband | people | service | said | camera | mobile_phone | speed | bt
Topic 7: yukos | russian | russia | tax | gazprom | oil | company | ha | bn | court
Topic 8: doping | test | kenteris | iaaf | conte | greek | drug | thanou | sprinter | athens
```

Model Evaluation Scores:

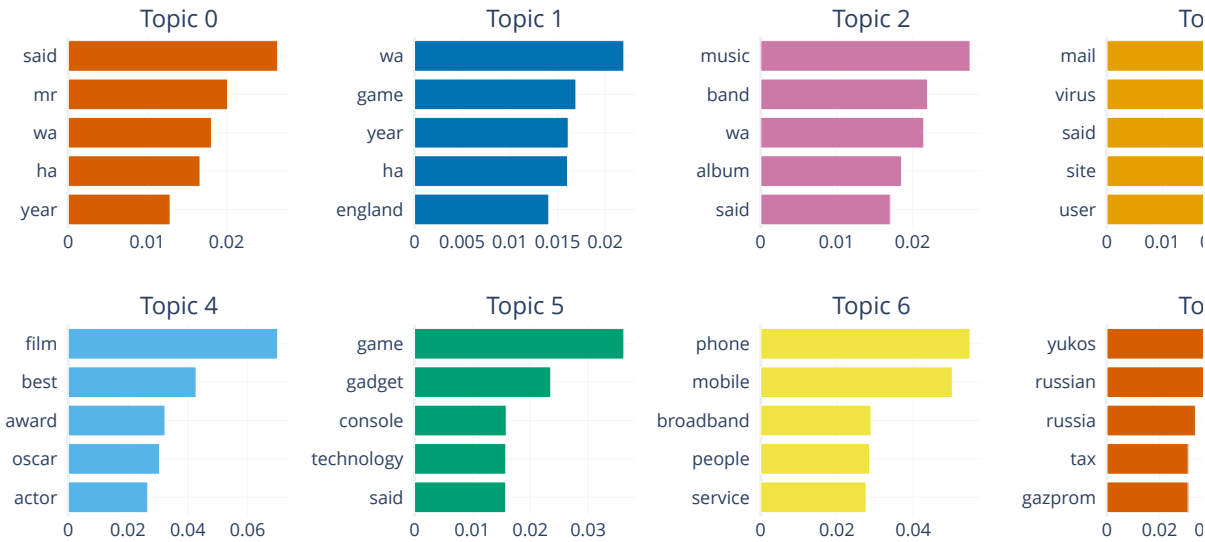
Coherence: 0.6500915433533024

BERTopic Model Visualization:

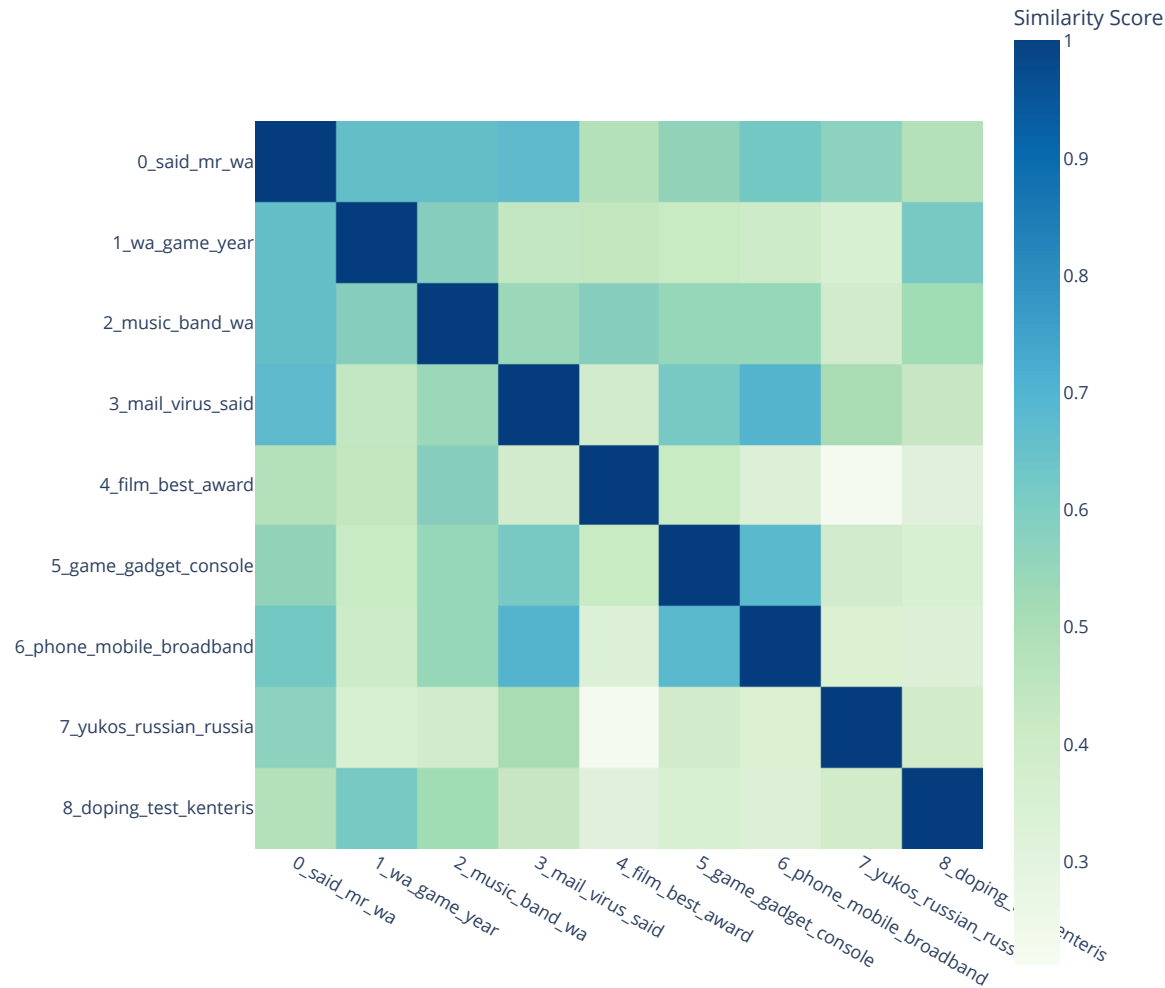
Intertopic Distance Map



Topic Word Scores



Similarity Matrix



If no visualization is shown,
you may execute the following commands one-by-one:
btm.model.visualize_topics()
btm.model.visualize_barchart()
btm.model.visualize_heatmap()

Finished 'btm_process' in 157.1346 secs

Save BERTopic Model

```
In [21]: btm.save("models/bertopic_bbc.pickle")
```

BERTopic model has been stored in 'models/bertopic_bbc.pickle'.

Load BERTopic Model

```
In [22]: num_topics = 10
btm2 = tm.BTM("", btm.num_topics)
btm2.model = btm2.load("models/bertopic_bbc.pickle")
btm2.show_topics()
```

Topics from BERTopic Model:
Topic 0: said | mr | wa | ha | year | government | election | bn | labour | party
Topic 1: wa | game | year | ha | england | said | win | time | half | player
Topic 2: music | band | wa | album | said | chart | year | song | singer | single
Topic 3: mail | virus | said | site | user | file | software | search | microsoft | firm
Topic 4: film | best | award | oscar | actor | wa | star | director | actress | aviator
Topic 5: game | gadget | console | technology | said | pc | digital | ha | wa | device
Topic 6: phone | mobile | broadband | people | service | said | camera | mobile_phone | speed | bt
Topic 7: yukos | russian | russia | tax | gazprom | oil | company | ha | bn | court
Topic 8: doping | test | kenteris | iaaf | conte | greek | drug | thanou | sprinter | athens

NMF Modeling

```
In [23]: nmf = tm.nmf_process(bbc_news, num_topics=8, source=1, text_col='Text', eval=True, timing=True, code=1)
```

Corpus loaded!
Text preprocessed!
Text trained!

Topics-Words from NMF Model:

Topic 1:
ha (0.005659)
party (0.004848)
said (0.004196)
best (0.002894)
tax (0.002250)
election (0.002213)
kilroy-silk (0.002179)
ukip (0.002128)
people (0.002098)
say (0.001839)

Topic 2:
wage (0.004884)
minimum (0.003762)
increase (0.003667)
minimum_wage (0.003449)
work (0.002764)
music (0.002634)
game (0.002626)
ha (0.002588)
mr (0.002564)
business (0.002299)

Topic 3:
wa (0.017407)
mr (0.008828)
said (0.007925)
film (0.006231)
ha (0.003151)
labour (0.002905)
tory (0.002497)
year (0.001994)
new (0.001993)
blair (0.001964)

Topic 4:
said (0.010767)
game (0.007571)
people (0.004837)
new (0.004623)
win (0.003260)
make (0.003078)
play (0.002798)
want (0.002768)
world (0.002494)
player (0.002310)

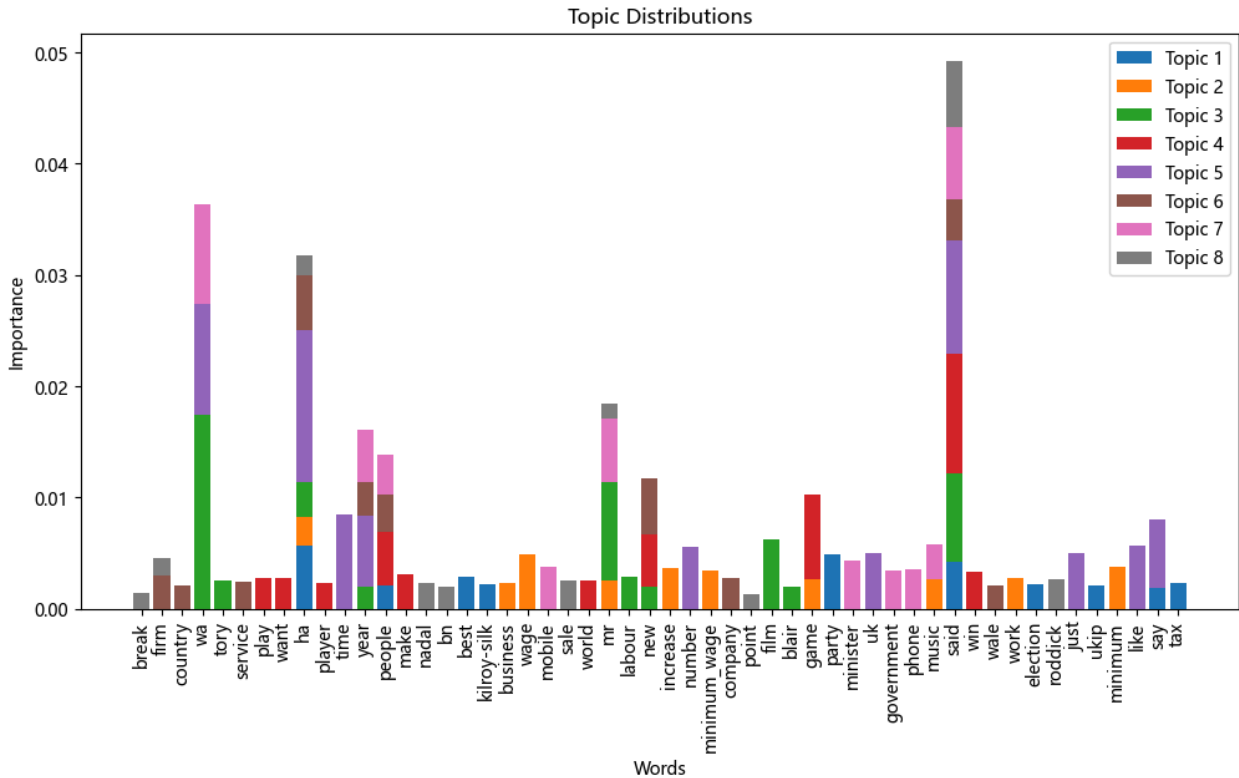
Topic 5:
ha (0.013672)
said (0.010189)
wa (0.010022)
time (0.008491)
year (0.006350)
say (0.006198)
like (0.005634)
number (0.005560)
just (0.004994)
uk (0.004958)

Topic 6:
new (0.005066)
ha (0.004922)
said (0.003728)
people (0.003300)
year (0.003067)
firm (0.002929)
company (0.002756)
service (0.002460)
country (0.002137)
wale (0.002103)

Topic 7:
wa (0.008871)
said (0.006511)
mr (0.005644)
year (0.004691)
minister (0.004268)
mobile (0.003802)
people (0.003597)
phone (0.003545)

government (0.003460)
music (0.003106)

Topic 8:
said (0.005933)
roddick (0.002642)
sale (0.002539)
nadal (0.002328)
bn (0.001936)
ha (0.001788)
firm (0.001658)
break (0.001411)
mr (0.001348)
point (0.001274)



Model Evaluation Scores:

Coherence: 0.565614817534048

Topic diversity: 0.0007111352479044387

Topic size distribution: 0.00102262559120542

Finished 'nmf_process' in 32.2746 secs

```
def nmf_process(doc_file, num_topics=10, source=0, text_col='text', cat=0, chi=False, group=True, eval=False):
    """Pipelines the NMF modeling.
```

```

    :param doc_file: The filename of the prescribed text file to be loaded,
        default to None
    :type doc_file: str
    :param num_topics: The number of topics to be modeled, default to 10
    :type num_topics: int, optional
    :param source: The source of the prescribed document file ('doc_file'),
        where 0 refers to internal store of the package and 1 to external file,
        default to 0
    :type source: int, optional
    :param text_col: The name of the text column to be extracted, default to 'text'
    :type text_col: str, optional
    :param cat: The category indicating a subset of the Scripture to be loaded, where
        0 stands for the whole Bible, 1 for OT, 2 for NT, or one of the ten categories
        ['tor', 'oth', 'ket', 'map', 'mip', 'gos', 'nth', 'pau', 'epi', 'apo'] (See
        the package's internal file 'data/book_cat.csv'), default to 0
    :type cat: int or str, optional
    :param chi: The flag indicating whether the text is processed as Chinese (True)
        or English (False), default to False
    :type chi: bool, optional
    :param group: The flag indicating whether the loaded text is grouped by chapter,
        default to True
    :type group: bool, optional
    :param eval: The flag indicating whether the model evaluation results will be shown,
        default to False
    :type eval: bool, optional
    :return: The pipelined NMF
    :rtype: cwordtm.tm.NMF object
    """
```

```

    nmf = NMF(doc_file, num_topics, chi)
    if source == 0:
        nmf.docs = load_bible(nmf.textfile, cat=cat, group=group)
    else:
        nmf.docs = load_text(nmf.textfile, text_col=text_col)
```

```
    print("Corpus loaded!")
```

```

    if chi:
        nmf.preprocess_chi()
    else:
        nmf.preprocess()
    print("Text preprocessed!")
```

```

    nmf.fit()
    print("Text trained!")
    nmf.show_topics_words()
    nmf.viz()
```

```

    if eval:
        print("\nModel Evaluation Scores:")
        nmf.evaluate()
```

```
    return nmf
```

```
>> cwordtm.tm.NMF
```

```
class NMF:
    """The NMF object for Non-negative Matrix Factorization (NMF) modeling.
```

```

    :cvar num_topics: The number of topics to be modeled, default to 10
    :vartype num_topics: int
    :ivar textfile: The filename of the text file to be processed
    :vartype textfile: str
    :ivar chi: The flag indicating whether the processed text is in Chinese or not,
        True stands for Traditional Chinese or False for English
    :vartype chi: bool
    :ivar num_topics: The number of topics set for the topic model
    :vartype num_topics: int
    :ivar docs: The collection of the original documents to be processed
    :vartype docs: pandas.DataFrame or list
    :ivar pro_docs: The collection of documents, in form of list of lists of words
        after text preprocessing
    :vartype pro_docs: list
    :ivar dictionary: The dictionary of word ids with their tokenized words
        from preprocessed documents ('pro_docs')
    :vartype dictionary: gensim.corpora.Dictionary
    :ivar corpus: The list of documents, where each document is a list of tuples
    """
```



```

        (word id, word frequency in the particular document)
:var type corpus: list
:ivar model: The NMF model object
:var type model: gensim.models.Nmf
"""

def __init__(self, textfile, num_topics, chi=False):
    """Constructor method.
    """

    self.textfile = textfile
    self.num_topics = num_topics
    self.chi = chi
    self.docs = None
    self.pro_docs = None
    self.dictionary = None
    self.corpus = None
    self.model = None

def preprocess(self):
    """Process the original English documents (cwordtm.tm.NMF.docs)
    by invoking cwordtm.tm.process_text, and build a dictionary
    and a corpus from the preprocessed documents for the NMF model.
    """

    self.pro_docs = [process_text(doc) for doc in self.docs]

    for i, doc in enumerate(self.pro_docs):
        self.pro_docs[i] += ["_".join(w) for w in ngrams(doc, 2)]
        # self.pro_docs[i] += ["_".join(w) for w in ngrams(doc, 3)]

    # Create a dictionary and corpus for the NMF model
    self.dictionary = corpora.Dictionary(self.pro_docs)
    self.corpus = [self.dictionary.doc2bow(doc) for doc in self.pro_docs]

def preprocess_chi(self):
    """Process the original Chinese documents (cwordtm.tm.NMF.docs)
    by tokenizing text, removing stopwords, and building a dictionary
    and a corpus from the preprocessed documents for the NMF model.
    """

    # Build stop words
    stop_file = files('cwordtm.data').joinpath("tc_stopwords_2.txt")
    stopwords = [k[:-1] for k in open(stop_file, encoding='utf-8')\
                  .readlines() if k != '']

    # Tokenize the text using Jieba
    dict_file = files('cwordtm.data').joinpath("user_dict_4.txt")
    jieba.load_userdict(str(dict_file))
    docs = [jieba.cut(doc) for doc in self.docs]

    # Replace special characters
    docs = [[word.replace('\u3000', ' ') for word in doc] \
            for doc in docs]

    # Remove stop words
    self.pro_docs = [' '.join([word for word in doc if word not in stopwords]) \
                     for doc in docs]

    self.pro_docs = [doc.split() for doc in self.pro_docs]

    # Create a dictionary and corpus
    self.dictionary = corpora.Dictionary(self.pro_docs)
    self.corpus = [self.dictionary.doc2bow(doc) for doc in self.pro_docs]

def fit(self):
    """Build the NMF model with the created corpus and dictionary.
    """

    self.model = models.Nmf(self.corpus,
                            num_topics=self.num_topics)

def show_topics_words(self):
    """Shows the topics with their keywords from the built NMF model.
    """

    print("\nTopics-Words from NMF Model:")
    for topic_id in range(self.num_topics):
        topic_words = self.model.show_topic(topic_id, topn=10)
        print(f"Topic {topic_id+1}:")
        for word_id, prob in topic_words:
            word = self.dictionary[int(word_id)]

```

```

        print("%s (%.6f)" %(word, prob))
    print()

def viz(self):
    """Plot the topic distributions as a stacked bar chart for the built NMF model.
    """

    # Build a list of word ids from the built topics
    word_ids = []
    for topic_id in range(self.num_topics):
        topic_words = self.model.show_topic(topic_id, topn=10)
        for word_id, _ in topic_words:
            word_ids.append(int(word_id))

    word_ids = list(set(word_ids))

    # Create a topic distribution table
    topic_dist = np.zeros((self.num_topics, len(word_ids)))
    for topic_id in range(self.num_topics):
        topic_words = self.model.show_topic(topic_id, topn=10)
        for word_id, prob in topic_words:
            topic_dist[topic_id, word_ids.index(int(word_id))] = prob

    # Build a list of distinct words from the word id list
    word_list = []
    for i in range(len(word_ids)):
        word_list.append(self.dictionary[word_ids[i]])

    # Plot the topic distributions
    matplotlib.rcParams['font.family'] = ['Microsoft YaHei']
    plt.figure(figsize=(12, 6))

    bottom = np.zeros(len(word_list))
    for i, topic in enumerate(topic_dist):
        plt.bar(word_list, topic, width=0.8, bottom=bottom, label=f"Topic {i+1}")
        bottom += topic

    plt.xticks(range(len(word_list)), word_list, rotation=90)
    plt.title("Topic Distributions")
    plt.xlabel("Words")
    plt.ylabel("Importance")
    plt.legend(loc="best")

    plt.show()

def evaluate(self):
    """Computes and outputs the coherence score, topic diversity,
    and topic size distribution.
    """

    # Compute coherence score
    coherence_model = CoherenceModel(model=self.model,
                                     texts=self.pro_docs,
                                     dictionary=self.dictionary,
                                     coherence='c_v')
    print(f" Coherence: {coherence_model.get_coherence()}")

    # Compute topic diversity
    topic_sizes = [len(self.model[self.corpus[i]]) for i in range(len(self.corpus))]
    total_docs = sum(topic_sizes)
    topic_diversity = sum([(size/total_docs)**2 for size in topic_sizes])
    print(f" Topic diversity: {topic_diversity}")

    # Compute topic size distribution
    # topic_sizes = [len(self.model[self.corpus[i]]) for i in range(len(self.corpus))]
    topic_size_distribution = max(topic_sizes) / sum(topic_sizes)
    print(f" Topic size distribution: {topic_size_distribution}\n")

def save(self, file):
    """Saves the built NMF model to the specified file.

    :param file: The name of the file to store the built model, default to None
    :type file: str
    """

    if file is None or len(file.strip())==0:
        print("No valid filename has been specifid!")
        return

    base_name = file.split('.')[0]
    model_file = base_name + '_model.gensim'
    dict_file = base_name + '_dictionary.gensim'
    self.model.save(model_file)

```

```

self.dictionary.save(dict_file)
# corpora.MmCorpus.serialize(base_name+'_corpus.mm', self.corpus)
print(f"NMF model has been saved: {model_file!r} and {dict_file!r}")

def load(self, file):
    """Loads the stored NMF model from the specified file.

    :param file: The name of the file to be loaded, default to None
    :type file: str
    :return: The loaded NMF model
    :rtype: NMF object
    """

    if file is None or len(file.strip())==0:
        print("No valid filename has been specified!")
        return

    base_name = file.split('.')[0]
    model_file = base_name + '_model.gensim'
    dict_file = base_name + '_dictionary.gensim'
    try:
        loaded_model = models.Nmf.load(model_file)
        loaded_dict = corpora.Dictionary.load(dict_file)
    except:
        print("Model file or dictionary file cannot be loaded!")
        return

    return loaded_model, loaded_dict

>> cwordtm.tm.load_bible
def load_bible(textfile, cat=0, group=True):
    """Loads and returns the Bible Scripture from the prescribed internal
    file ('textfile').

    :param textfile: The package's internal Bible text from which the text is loaded,
        either World English Bible ('web.csv') or Chinese Union Version (Traditional)
        ('cuv.csv'), default to None
    :type textfile: str
    :param cat: The category indicating a subset of the Scripture to be loaded, where
        0 stands for the whole Bible, 1 for OT, 2 for NT, or one of the ten categories
        ['tor', 'oth', 'ket', 'map', 'mip', 'gos', 'nth', 'pau', 'epi', 'apo'] (See
        the package's internal file 'data/book_cat.csv'), default to 0
    :type cat: int or str, optional
    :param group: The flag indicating whether the loaded text is grouped by chapter,
        default to True
    :type group: bool, optional
    :return: The collection of Scripture loaded
    :rtype: pandas.DataFrame
    """

    # textfile = "web.csv"
    scfile = files('cwordtm.data').joinpath(textfile)
    print("Loading Bible '%s' ..." %scfile)
    df = pd.read_csv(scfile)

    cat_list = ['tor', 'oth', 'ket', 'map', 'mip', \
                'gos', 'nth', 'pau', 'epi', 'apo']
    cat = str(cat)
    if cat == '1' or cat == 'ot':
        df = util.extract(df, testament=0)
    elif cat == '2' or cat == 'nt':
        df = util.extract(df, testament=1)
    elif cat in cat_list:
        df = util.extract(df, category=cat)

    if group:
        # Group verses into chapters
        df = df.groupby(['book_no', 'chapter'])\
            .agg({'text': lambda x: ' '.join(x)})\
            .reset_index()

    df.text = df.text.str.replace(' ', '')
    return list(df.text)

>> cwordtm.tm.load_text
def load_text(textfile, text_col='text'):
    """Loads and returns the list of documents from the prescribed file ('textfile').

    :param textfile: The prescribed text file from which the text is loaded,
        default to None
    :type textfile: str
    :param text_col: The name of the text column to be extracted, default to 'text'
    :type text_col: str, optional
    :return: The list of documents loaded
    :rtype: list

```

```
"""
```

```
# docs = pd.read_csv(textfile)
docs = util.load_text(textfile)
return list(docs[text_col])
```