CWordTM Toolkit Usage on BBC News

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 and BERTopic

In [1]:

```
import warnings
warnings.filterwarnings('ignore')
```

1. Meta Information Features

In [2]:

```
import cwordtm
from cwordtm import *
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\johnnyc\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\johnnyc\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\johnnyc\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\johnnyc\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [3]:

```
cwordtm.__version__
```

Out[3]:

```
'0.6.3'
```

In [4]:

```
# Show brief module information
print(meta.get_module_info())
```

```
The member information of the module 'cwordtm'
1. Submodule meta:
   addin (func)
   addin_all (modname='cwordtm')
   addin_all_functions (submod)
   get_function (mod_name, submodules, func_name)
   get_module_info (detailed=False)
   get_submodule_info (submodname, detailed=False)
   import_module (name, package=None)
   wraps (wrapped, assigned=('__module__', '__name__', '__qualname__', '__doc__', '__annotations__'), updated=('__di
ct__',))
2. Submodule pivot:
   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, chi=False, num_topics=15, embed=True)
   LDA (textfile, chi=False, num_topics=15)
   NMF (textfile, chi=False, num_topics=15)
   btm_process (doc_file, source=0, text_col='text', cat=0, chi=False, group=True, eval=False, *, timing=False, code
=0)
   lda_process (doc_file, 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, 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_number
s=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)
   set_lang (lang='en', *, 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]:

```python
# 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 presribed 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 presribed 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
df = util.load_text("BBC/BBC News Train.csv", timing=True)
```

```
Finished 'load_text' in 0.0281 secs
```

In [7]:
```
# Execute and show code
df = util.load_text("BBC/BBC News Train.csv", 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/BBC News Train.csv", 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]:

```
bbc_file = "BBC/BBC News Train.csv"
df = util.load_text(bbc_file, 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
```

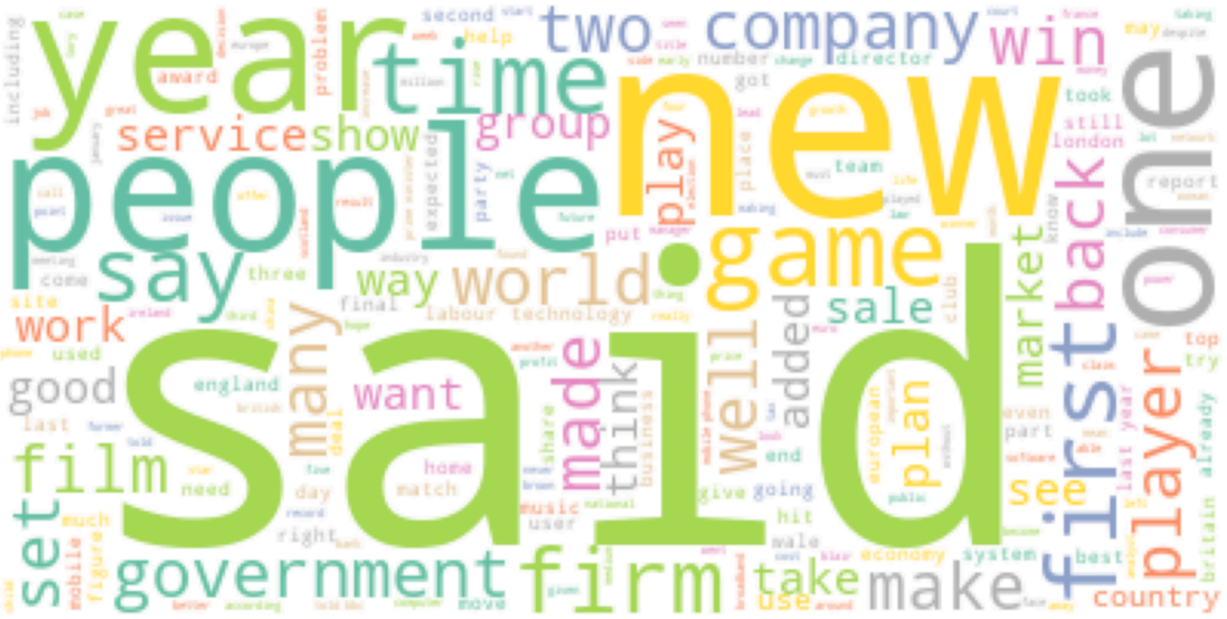
Preprocessing Text

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

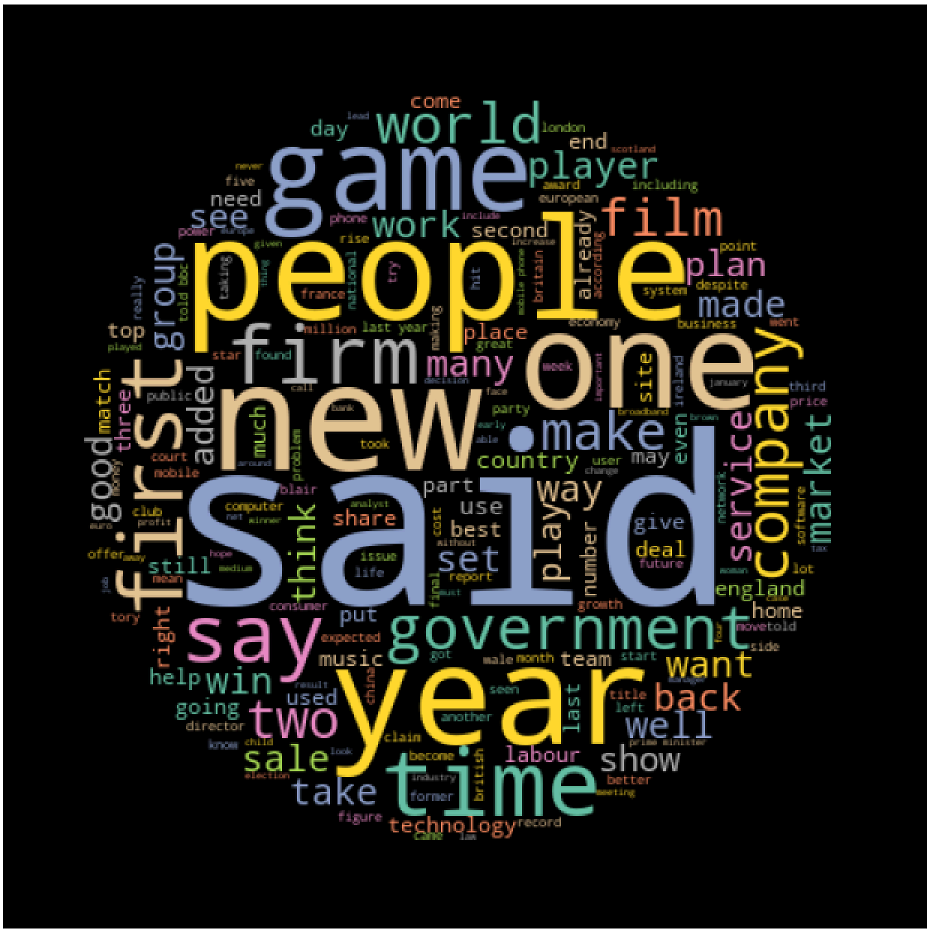
In [12]:
```
# White background with no image mask
viz.show_wordcloud(text)
```

```
D:\Dev\Anaconda3\lib\site-packages\wordcloud\wordcloud.py:106: MatplotlibDeprecationWarning: The get_cmap function w
as 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 w
as 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)
```



4. Text Summarization

In [14]:
```
news = df.iloc[:5]['Text']  # "df" stores previously loaded text
ta.summary_en(news, sent_len=5)
```

Out[14]:
```
['but ms cooper  who now runs her own consulting business  told a jury in new york on wednesday that external audito
rs arthur andersen had approved worldcom s accounting in early 2001 and 2002. she said andersen had given a  green l
ight  to the procedures and practices used by worldcom.',
 'cynthia cooper  worldcom s ex-head of internal accounting  alerted directors to irregular accounting practices at
the us telecoms giant in 2002. her warnings led to the collapse of the firm following the discovery of an $11bn (£5.
7bn) accounting fraud.',
 'prosecution lawyers have argued that mr ebbers orchestrated a series of accounting tricks at worldcom  ordering em
ployees to hide expenses and inflate revenues to meet wall street earnings estimates.',
 'the university of california said the trial in the case is scheduled to begin in october 2006. it joined the lawsu
it in december 2001alleging  massive insider trading  and fraud  claiming it had lost $145m on its investments in th
e company.',
 'the bbc s david willey in rome says one reason for that result is the changeover from the lira to the euro in 2001
which is widely viewed as the biggest reason why their wages and salaries are worth less than they used to be.']
```

5. Topic Modeling

In [15]:
```
import warnings
warnings.filterwarnings('ignore')
```

LDA Model

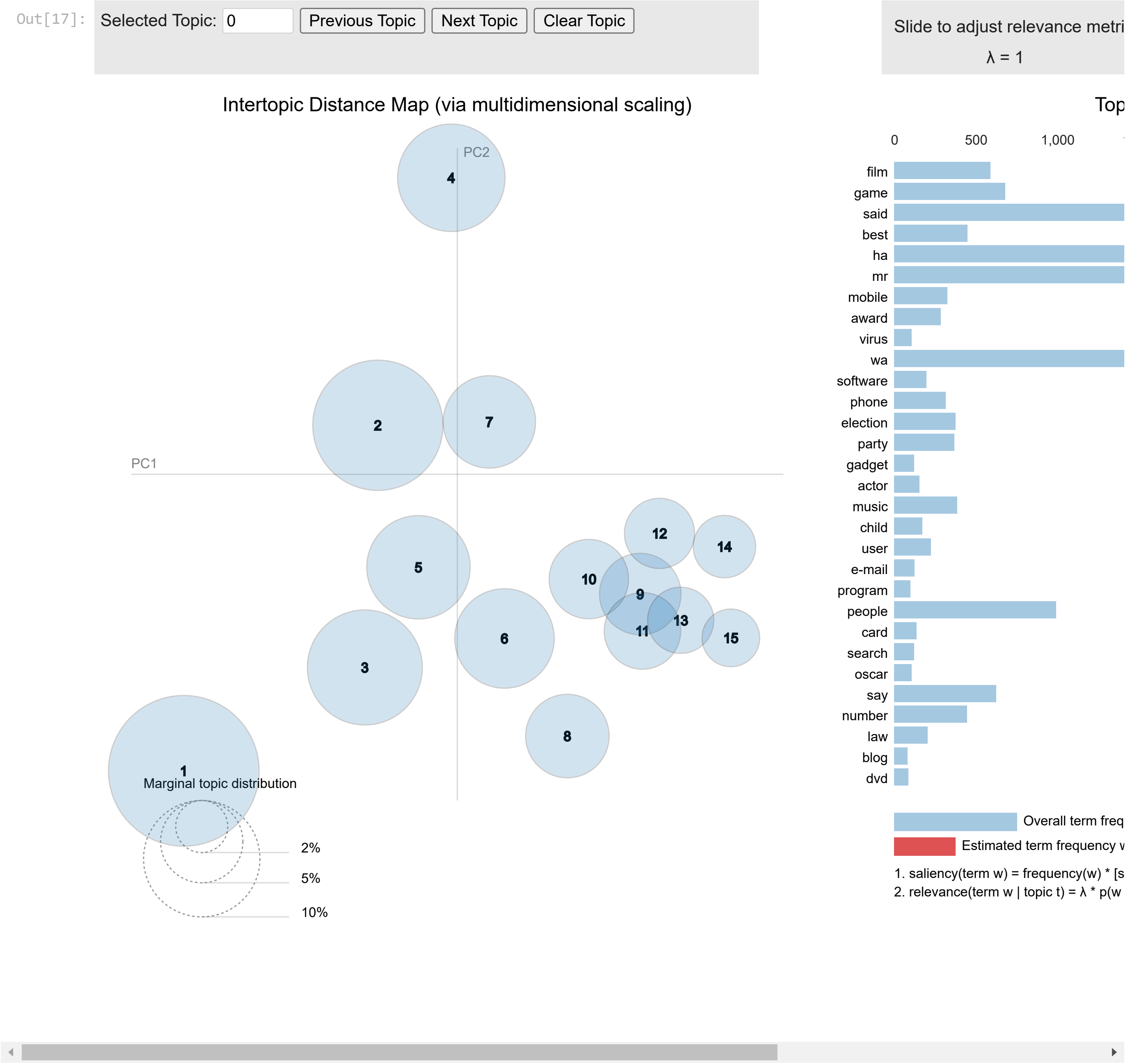
In [16]:
```
doc_file = "BBC/BBC News Train.csv"
lda = tm.lda_process(doc_file, source=1, text_col='Text', eval=True, timing=True)
```

```
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,
  '0.005*"said" + 0.004*"ha" + 0.003*"wa" + 0.002*"year" + 0.002*"film" + '
  '0.002*"new" + 0.001*"world" + 0.001*"test" + 0.001*"law" + 0.001*"share"'),
 (1,
  '0.007*"said" + 0.006*"wa" + 0.003*"new" + 0.003*"year" + 0.003*"mr" + '
  '0.003*"ha" + 0.002*"sale" + 0.002*"uk" + 0.001*"world" + '
  '0.001*"government"'),
 (2,
  '0.006*"said" + 0.006*"wa" + 0.004*"ha" + 0.004*"film" + 0.004*"best" + '
  '0.003*"year" + 0.002*"mr" + 0.002*"award" + 0.002*"new" + 0.002*"actor"'),
 (3,
  '0.002*"said" + 0.002*"wa" + 0.002*"mr" + 0.002*"say" + 0.002*"election" + '
  '0.001*"party" + 0.001*"year" + 0.001*"home" + 0.001*"ha" + 0.001*"suspect"'),
 (4,
  '0.005*"said" + 0.003*"ha" + 0.003*"wa" + 0.002*"mr" + 0.001*"game" + '
  '0.001*"id" + 0.001*"card" + 0.001*"id_card" + 0.001*"howard" + '
  '0.001*"child"'),
 (5,
  '0.006*"said" + 0.006*"wa" + 0.004*"ha" + 0.003*"mr" + 0.003*"film" + '
  '0.002*"people" + 0.002*"year" + 0.001*"phone" + 0.001*"best" + '
  '0.001*"award"'),
 (6,
  '0.010*"wa" + 0.007*"said" + 0.004*"ha" + 0.002*"year" + 0.002*"game" + '
  '0.002*"mr" + 0.001*"time" + 0.001*"new" + 0.001*"england" + 0.001*"bn"'),
 (7,
  '0.006*"said" + 0.005*"ha" + 0.005*"wa" + 0.003*"people" + 0.003*"year" + '
  '0.002*"mr" + 0.002*"film" + 0.002*"world" + 0.002*"technology" + '
  '0.002*"new"'),
 (8,
  '0.005*"said" + 0.005*"ha" + 0.003*"wa" + 0.003*"mr" + 0.002*"game" + '
  '0.002*"people" + 0.002*"year" + 0.001*"election" + 0.001*"new" + '
  '0.001*"number"'),
 (9,
  '0.009*"said" + 0.005*"ha" + 0.005*"wa" + 0.005*"mr" + 0.003*"year" + '
  '0.003*"people" + 0.002*"new" + 0.002*"government" + 0.002*"bn" + '
  '0.002*"say"'),
 (10,
  '0.004*"said" + 0.004*"ha" + 0.003*"wa" + 0.003*"game" + 0.002*"year" + '
  '0.002*"mr" + 0.002*"time" + 0.001*"child" + 0.001*"dvd" + 0.001*"world"'),
 (11,
  '0.006*"said" + 0.004*"wa" + 0.003*"ha" + 0.002*"virus" + 0.002*"mr" + '
  '0.002*"software" + 0.002*"people" + 0.002*"e-mail" + 0.001*"program" + '
  '0.001*"new"'),
 (12,
  '0.004*"said" + 0.003*"wa" + 0.002*"ha" + 0.002*"number" + 0.002*"year" + '
  '0.001*"time" + 0.001*"game" + 0.001*"people" + 0.001*"user" + '
  '0.001*"service"'),
 (13,
  '0.005*"said" + 0.004*"ha" + 0.004*"wa" + 0.003*"game" + 0.002*"company" + '
  '0.002*"mr" + 0.002*"say" + 0.002*"new" + 0.002*"year" + 0.002*"world"'),
 (14,
  '0.005*"said" + 0.004*"ha" + 0.003*"wa" + 0.002*"mobile" + 0.002*"year" + '
  '0.002*"people" + 0.002*"phone" + 0.002*"music" + 0.002*"gadget" + '
  '0.001*"new"')]

Model Evaluation Scores:
  Coherence: 0.6423303672893046
  Perplexity: -11.29942365009797
  Topic diversity: 0.0007698825601324054
  Topic size distribution: 0.0022087244616234127

Finished 'lda_process' in 72.6000 secs
```

In [17]:

```python
# LDA Model Visualization
import pyLDAvis
pyLDAvis.display(lda.vis_data)
```

Out[17]:



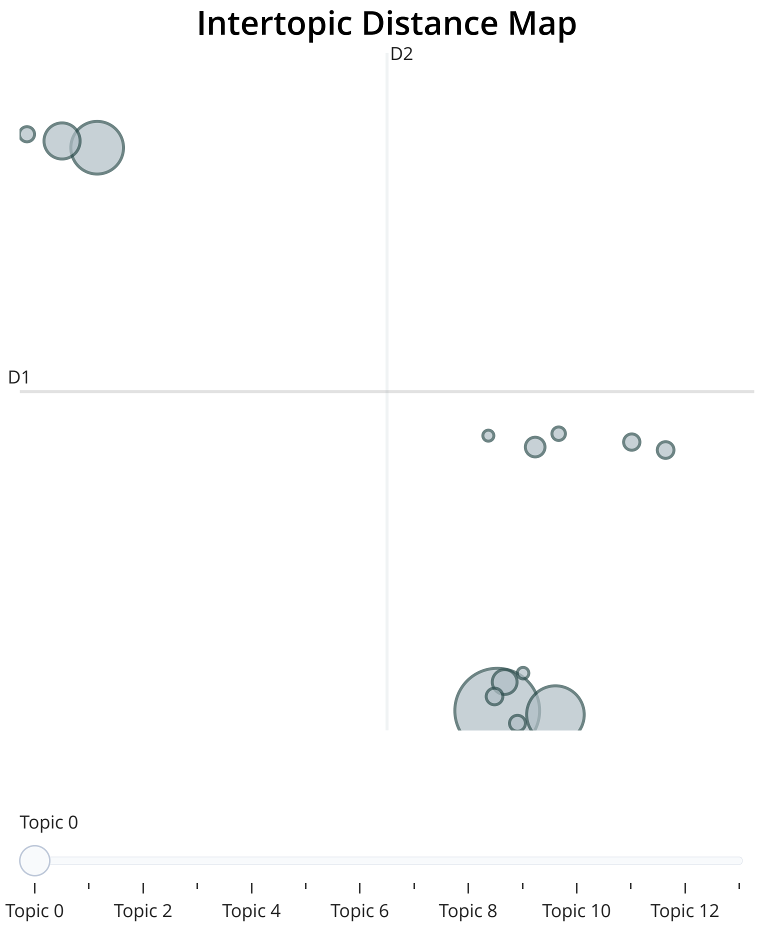
BERTopic Model

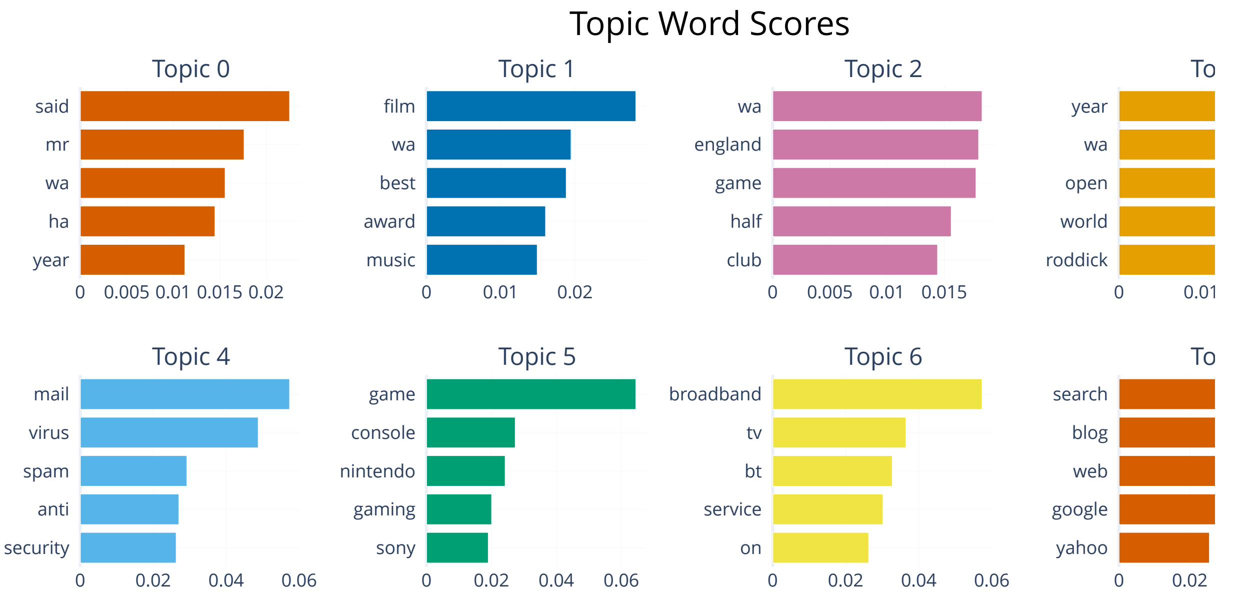
In [18]:

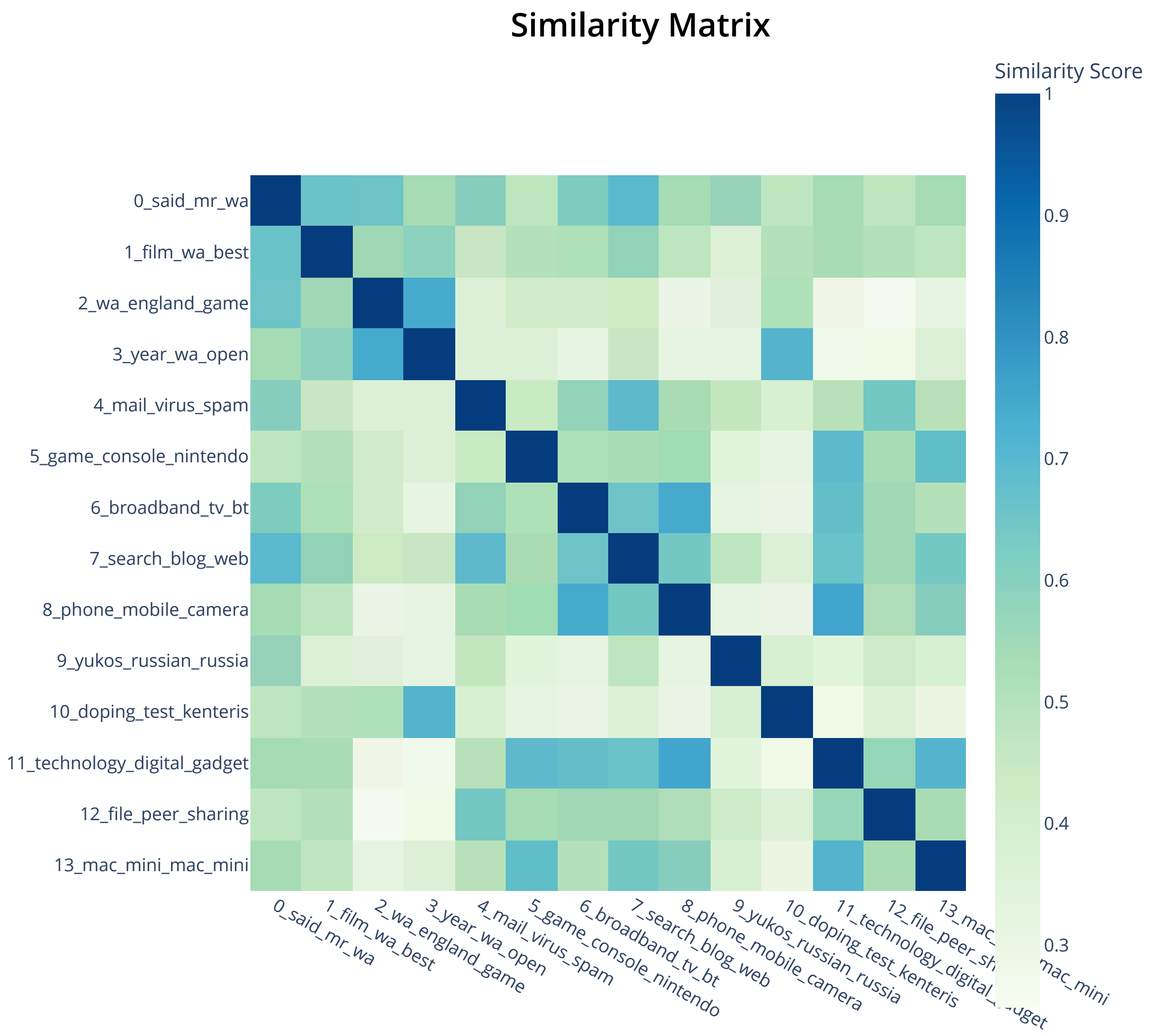
```python
btm = tm.btm_process(doc_file, source=1, text_col='Text', eval=True, timing=True)
```

```
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.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dense.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.weight', 'cls.seq_relationship.bias', 'cls.predictions.decoder.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 | labour | bn | party
Topic 1: film | wa | best | award | music | year | said | star | ha | actor
Topic 2: wa | england | game | half | club | ha | player | time | team | said
Topic 3: year | wa | open | world | roddick | champion | old | win | final | ha
Topic 4: mail | virus | spam | anti | security | site | spyware | user | said | attack
Topic 5: game | console | nintendo | gaming | sony | gamers | title | xbox | halo | player
Topic 6: broadband | tv | bt | service | on | speed | net | customer | people | uk
Topic 7: search | blog | web | google | yahoo | people | search_engine | said | user | desktop
Topic 8: phone | mobile | camera | mobile_phone | people | technology | handset | camera_phone | use | said
Topic 9: yukos | russian | russia | gazprom | tax | oil | company | bn | khodorkovsky | ha
Topic 10: doping | test | kenteris | iaaf | conte | greek | drug | thanou | sprinter | athens
Topic 11: technology | digital | gadget | device | electronics | consumer | ce | content | consumer_electronics | pe
ople
Topic 12: file | peer | sharing | pp | to | network | said | apple | piracy | firm
Topic 13: mac | mini | mac_mini | pc | computer | commodore | apple | laptop | machine | said

Model Evaluation Scores:
  Coherence: 0.6576968143443093

BERTopic Model Visualization:
```




Topic Word Scores
Topic 0
said
mr
wa
ha
year
0 0.005 0.01 0.015 0.02
Topic 1
film
wa
best
award
music
0 0.01 0.02
Topic 2
wa
england
game
half
club
0 0.005 0.01 0.015
To
year
wa
open
world
roddick
0 0.01
Topic 4
mail
virus
spam
anti
security
0 0.02 0.04 0.06
Topic 5
game
console
nintendo
gaming
sony
0 0.02 0.04 0.06
Topic 6
broadband
tv
bt
service
on
0 0.02 0.04 0.06
To
search
blog
web
google
yahoo
0 0.02


Similarity Matrix
Similarity Score
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0_said_mr_wa
1_film_wa_best
2_wa_england_game
3_year_wa_open
4_mail_virus_spam
5_game_console_nintendo
6_broadband_tv_bt
7_search_blog_web
8_phone_mobile_camera
9_yukos_russian_russia
10_doping_test_kenteris
11_technology_digital_gadget
12_file_peer_sharing
13_mac_mini_mac_mini

```
  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 125.4787 secs
```

NMF Model (Code Reveal Only)

In [19]:

```
nmf = tm.nmf_process(doc_file, source=1, text_col='Text', eval=True, timing=True, code=1)
```

```
Corpus loaded!
Text preprocessed!
Text trained!

Topics-Words from NMF Model:
Topic 1:
best (0.002166)
ireland (0.002122)
year (0.002009)
world (0.001799)
second (0.001739)
sale (0.001706)
england (0.001704)
france (0.001452)
open (0.001429)
net (0.001112)

Topic 2:
new (0.008297)
said (0.004078)
music (0.003227)
radio (0.002460)
home (0.001941)
plan (0.001780)
bbc (0.001735)
digital (0.001379)
wale (0.001273)
right (0.001243)

Topic 3:
people (0.007195)
said (0.004758)
new (0.003756)
work (0.003563)
music (0.003526)
make (0.003161)
wage (0.003143)
message (0.002981)
mobile (0.002900)
just (0.002647)

Topic 4:
mr (0.007335)
election (0.003462)
blair (0.002948)
said (0.002806)
film (0.002547)
government (0.002333)
say (0.001954)
mr_blair (0.001914)
year (0.001768)
leader (0.001592)

Topic 5:
said (0.013868)
mr (0.009683)
wa (0.007273)
year (0.005412)
best (0.005406)
mobile (0.004518)
film (0.003990)
ha (0.003776)
award (0.003349)
phone (0.002821)

Topic 6:
mr (0.013487)
say (0.011946)
wa (0.010159)
new (0.007728)
labour (0.006834)
election (0.006424)
blair (0.005604)
sale (0.004935)
time (0.004765)
public (0.004529)

Topic 7:
ha (0.016264)
wa (0.011105)
world (0.005790)
week (0.003646)
just (0.003440)
year (0.003370)
say (0.003143)
increase (0.003122)
```

```
party (0.002995)
good (0.002885)

Topic 8:
said (0.025548)
ha (0.010240)
wa (0.006689)
player (0.003609)
firm (0.003538)
child (0.002757)
court (0.002600)
case (0.002584)
apple (0.002461)
legal (0.002217)

Topic 9:
ha (0.005434)
bn (0.004653)
said (0.004541)
firm (0.003779)
film (0.003175)
yukos (0.002776)
company (0.002345)
tax (0.002188)
sale (0.002109)
market (0.001831)

Topic 10:
people (0.024035)
number (0.010615)
uk (0.008284)
like (0.007742)
way (0.007092)
work (0.005744)
think (0.005550)
make (0.005467)
year (0.005191)
right (0.005052)

Topic 11:
wa (0.005261)
year (0.002492)
ha (0.002367)
wage (0.002358)
service (0.002216)
increase (0.002037)
time (0.002032)
market (0.001938)
net (0.001856)
roddick (0.001849)

Topic 12:
award (0.001496)
eu (0.001285)
ha (0.001274)
wa (0.001236)
company (0.001179)
member (0.001145)
film (0.001132)
added (0.001117)
cash (0.001055)
party (0.001026)

Topic 13:
wa (0.013839)
mr (0.008215)
said (0.004927)
year (0.003752)
new (0.003511)
tax (0.003459)
brown (0.003376)
bn (0.002963)
ha (0.002892)
game (0.002420)

Topic 14:
wa (0.008144)
game (0.006728)
film (0.004328)
win (0.002837)
actor (0.002401)
director (0.002263)
new_zealand (0.001860)
zealand (0.001860)
actress (0.001860)
jamie (0.001800)
```

```
Topic 15:
service (0.005827)
said (0.005159)
technology (0.004060)
music (0.003307)
digital (0.002579)
home (0.002461)
company (0.002447)
uk (0.002176)
firm (0.002176)
network (0.002022)


Model Evaluation Scores:
  Coherence: 0.5495309484736951
  Topic diversity: 0.000711098456347347
  Topic size distribution: 0.0011337868480725624

Finished 'nmf_process' in 42.7166 secs

def nmf_process(doc_file, 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 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, 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()

    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 built from the modeling,
        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)
    :vartype corpus: list
    :ivar model: The NMF model object
    :vartype model: gensim.models.Nmf
    """

    def __init__(self, textfile, chi=False, num_topics=15):
        """Constructor method.
        """

        self.textfile = textfile
        self.chi = chi
        self.num_topics = num_topics
        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.model.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.id2token[int(word_id)]
                word = self.dictionary[int(word_id)]
                print("%s (%.6f)" %(word, prob))
            print()

    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")

>> 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])
```