CWordTM Toolkit Usage on BBC News

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

1. Code Reveal
2. Topic Modeling with LDA

Full Demonstration:
https://github.com/drjohnnycheng/CWordTM/blob/main/Demo/CWordTM_BBC.ipynb

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

In [1]:
```
# Import the Package CWordTM
import cwordtm
from cwordtm import *
```

1. Code Reveal

1.1 get_module_info / get_submodule_info

In [2]:
```
# Show brief module information
# print(meta.get_module_info())

# Show function signature of all functions in a submodule
print(meta.get_submodule_info("viz"))
```

```
The function(s) of the submodule 'cwordtm.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 [3]:
```
# Show source code of all the functions in 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)
```

1.2 'code' Parameter

In [4]:
```
bbc_news = "BBC/BBC News Train.csv"
df = util.load_text(bbc_news, info=True)
text_list = util.get_text_list(df.iloc[:500], text_col='Text')
text = util.preprocess_text(text_list)
```

```
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
```

In [5]:
```
# Reveal Code Without Execution
text = util.preprocess_text(text_list, code=2)

# code=0 : default, execution without code reveal
# code=1 : execution with code reveal
```

```
def preprocess_text(text):
    """Preprocesses English text by converting text to lower case, removing
    special characters and digits, removing punctuations, removing stopwords,
    removing short words, and Lemmatize text.

    :param text: The text to be preprocessed, default to None
    :type text: str
    :return: The preprocessed text
    :rtype: str
    """

    if isinstance(text, list) or isinstance(text, np.ndarray):
        text = ' '.join(str(item) for item in text)
    elif isinstance(text, pd.Series):
        text = ' '.join(list(text.astype(str)))

    # print("Preprocessing text ...")

    # Convert text to lowercase
    text = text.lower()

    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove stopwords
    text = " ".join([word for word in nltk.word_tokenize(text) \
                    if word.lower() not in stopwords.words('english')])

    # Remove short words (length < 3)
    text = " ".join([word for word in nltk.word_tokenize(text) if len(word) >= 3])

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    text = " ".join([lemmatizer.lemmatize(word) for word in nltk.word_tokenize(text)])

    return text


>> nltk.stem.wordnet.WordNetLemmatizer
class WordNetLemmatizer:
    """
    WordNet Lemmatizer

    Lemmatize using WordNet's built-in morphy function.
    Returns the input word unchanged if it cannot be found in WordNet.

        >>> from nltk.stem import WordNetLemmatizer
        >>> wnl = WordNetLemmatizer()
        >>> print(wnl.lemmatize('dogs'))
        dog
        >>> print(wnl.lemmatize('churches'))
        church
        >>> print(wnl.lemmatize('aardwolves'))
        aardwolf
        >>> print(wnl.lemmatize('abaci'))
        abacus
        >>> print(wnl.lemmatize('hardrock'))
        hardrock
    """

    def lemmatize(self, word: str, pos: str = "n") -> str:
        """Lemmatize `word` using WordNet's built-in morphy function.
        Returns the input word unchanged if it cannot be found in WordNet.

        :param word: The input word to lemmatize.
        :type word: str
        :param pos: The Part Of Speech tag. Valid options are `"n"` for nouns,
            `"v"` for verbs, `"a"` for adjectives, `"r"` for adverbs and `"s"`
            for satellite adjectives.
        :param pos: str
        :return: The lemma of `word`, for the given `pos`.
        """
        lemmas = wn._morphy(word, pos)
        return min(lemmas, key=len) if lemmas else word

    def __repr__(self):
        return "<WordNetLemmatizer>"

>> nltk.tokenize.word_tokenize
def word_tokenize(text, language="english", preserve_line=False):
    """
    Return a tokenized copy of *text*,
    using NLTK's recommended word tokenizer
    (currently an improved :class:`.TreebankWordTokenizer`
    along with :class:`.PunktSentenceTokenizer`
    for the specified language).

    :param text: text to split into words
    :type text: str
    :param language: the model name in the Punkt corpus
    :type language: str
    :param preserve_line: A flag to decide whether to sentence tokenize the text or not.
    :type preserve_line: bool
    """
    sentences = [text] if preserve_line else sent_tokenize(text, language)
    return [
        token for sent in sentences for token in _treebank_word_tokenizer.tokenize(sent)
    ]

>> nltk.tokenize.word_tokenize
def word_tokenize(text, language="english", preserve_line=False):
    """
    Return a tokenized copy of *text*,
    using NLTK's recommended word tokenizer
    (currently an improved :class:`.TreebankWordTokenizer`
    along with :class:`.PunktSentenceTokenizer`
    for the specified language).

    :param text: text to split into words
    :type text: str
    :param language: the model name in the Punkt corpus
    :type language: str
    :param preserve_line: A flag to decide whether to sentence tokenize the text or not.
    :type preserve_line: bool
    """
    sentences = [text] if preserve_line else sent_tokenize(text, language)
    return [
        token for sent in sentences for token in _treebank_word_tokenizer.tokenize(sent)
    ]

>> nltk.tokenize.word_tokenize
def word_tokenize(text, language="english", preserve_line=False):
    """
    Return a tokenized copy of *text*,
    using NLTK's recommended word tokenizer
    (currently an improved :class:`.TreebankWordTokenizer`
    along with :class:`.PunktSentenceTokenizer`
    for the specified language).

    :param text: text to split into words
    :type text: str
    :param language: the model name in the Punkt corpus
    :type language: str
    :param preserve_line: A flag to decide whether to sentence tokenize the text or not.
    :type preserve_line: bool
    """
    sentences = [text] if preserve_line else sent_tokenize(text, language)
    return [
        token for sent in sentences for token in _treebank_word_tokenizer.tokenize(sent)
    ]
```

2. Topic Modeling with LDA

In [6]:
```
import warnings
warnings.filterwarnings('ignore')
```

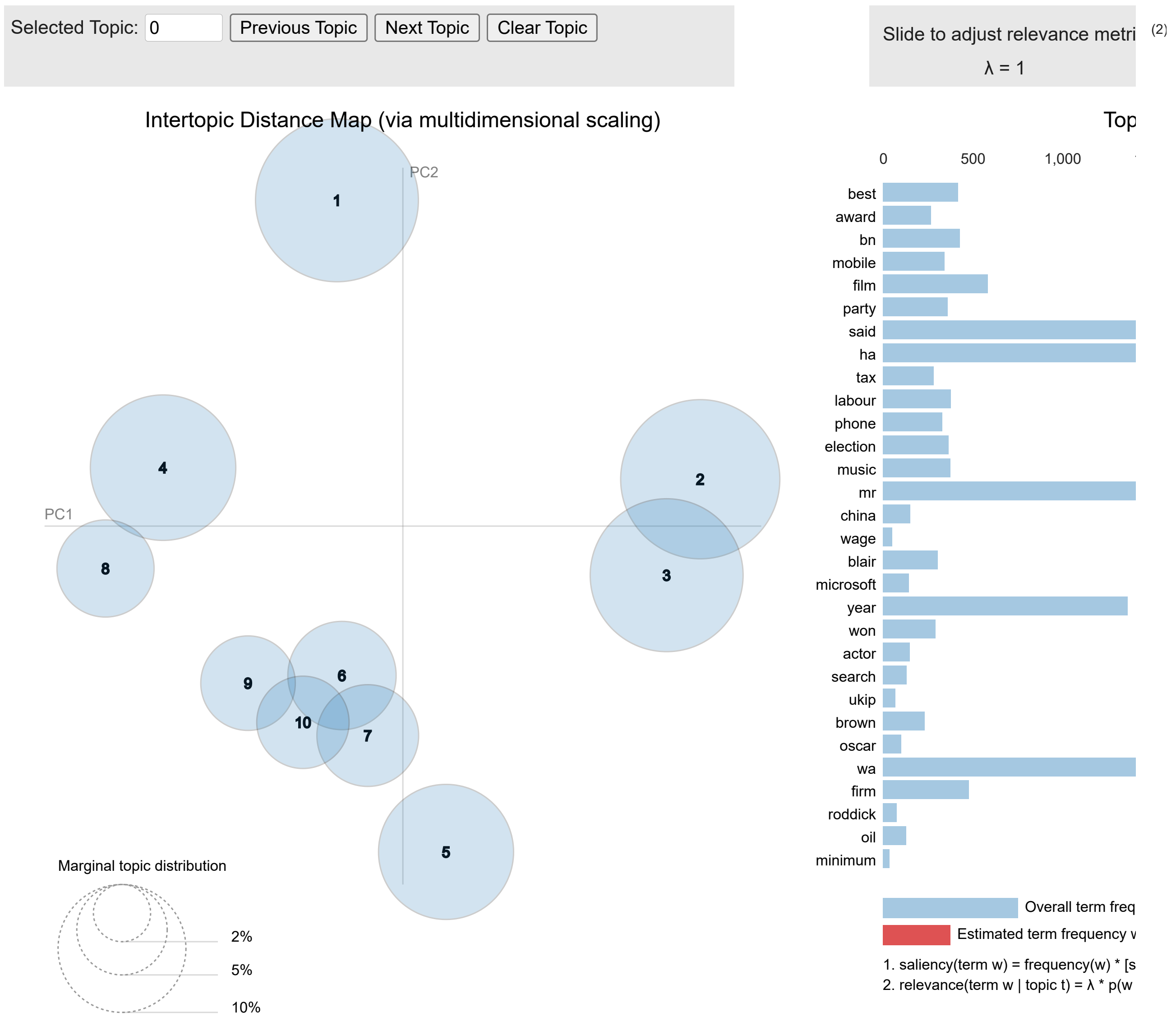
LDA Pipeline

In [7]:
```
lda = tm.lda_process(bbc_news, 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,
  '0.006*"said" + 0.005*"wa" + 0.004*"ha" + 0.003*"year" + 0.002*"new" + '
  '0.002*"people" + 0.002*"music" + 0.001*"government" + 0.001*"bn" + '
  '0.001*"market"'),
 (1,
  '0.006*"said" + 0.005*"wa" + 0.004*"ha" + 0.003*"mr" + 0.002*"year" + '
  '0.002*"people" + 0.002*"game" + 0.001*"new" + 0.001*"time" + 0.001*"say"'),
 (2,
  '0.009*"said" + 0.005*"wa" + 0.004*"mr" + 0.004*"ha" + 0.003*"people" + '
  '0.002*"year" + 0.002*"service" + 0.002*"new" + 0.002*"uk" + 0.002*"phone"'),
 (3,
  '0.007*"said" + 0.004*"ha" + 0.003*"wa" + 0.002*"mr" + 0.002*"bn" + '
  '0.002*"year" + 0.001*"new" + 0.001*"company" + 0.001*"election" + '
  '0.001*"firm"'),
 (4,
  '0.006*"said" + 0.006*"wa" + 0.005*"ha" + 0.003*"film" + 0.002*"year" + '
  '0.002*"mr" + 0.002*"new" + 0.002*"time" + 0.001*"people" + 0.001*"game"'),
 (5,
  '0.005*"said" + 0.004*"ha" + 0.004*"wa" + 0.003*"mr" + 0.002*"year" + '
  '0.002*"party" + 0.001*"new" + 0.001*"say" + 0.001*"labour" + 0.001*"world"'),
 (6,
  '0.004*"said" + 0.004*"ha" + 0.003*"wa" + 0.002*"mr" + 0.002*"year" + '
  '0.001*"game" + 0.001*"wage" + 0.001*"people" + 0.001*"phone" + '
  '0.001*"mobile"'),
 (7,
  '0.007*"wa" + 0.006*"said" + 0.004*"ha" + 0.003*"mr" + 0.003*"year" + '
  '0.002*"game" + 0.002*"new" + 0.002*"world" + 0.002*"people" + 0.001*"time"'),
 (8,
  '0.005*"wa" + 0.004*"said" + 0.004*"best" + 0.003*"ha" + 0.002*"award" + '
  '0.002*"year" + 0.001*"won" + 0.001*"film" + 0.001*"new" + 0.001*"actor"'),
 (9,
  '0.004*"said" + 0.003*"wa" + 0.003*"year" + 0.002*"ha" + 0.002*"mr" + '
  '0.001*"new" + 0.001*"number" + 0.001*"uk" + 0.001*"music" + 0.001*"people"')]

Model Evaluation Scores:
  Coherence: 0.6602593608479931
  Perplexity: -11.243239884628622
  Topic diversity: 0.0007234740198722521
  Topic size distribution: 0.001854140914709518

Finished 'lda_process' in 57.3017 secs
```

In [8]:
```python
# LDA Model Visualization
import pyLDAvis
pyLDAvis.display(lda.vis_data)
```

Out[8]:



Save LDA Model

In [9]:
```python
lda.save("models/lda_bbc.gensim")
```

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

Load LDA Model

In [10]:
```python
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" + 0.005*"wa" + 0.004*"ha" + 0.003*"year" + 0.002*"new" + '
  '0.002*"people" + 0.002*"music" + 0.001*"government" + 0.001*"bn" + '
  '0.001*"market"'),
 (1,
  '0.006*"said" + 0.005*"wa" + 0.004*"ha" + 0.003*"mr" + 0.002*"year" + '
  '0.002*"people" + 0.002*"game" + 0.001*"new" + 0.001*"time" + 0.001*"say"'),
 (2,
  '0.009*"said" + 0.005*"wa" + 0.004*"mr" + 0.004*"ha" + 0.003*"people" + '
  '0.002*"year" + 0.002*"service" + 0.002*"new" + 0.002*"uk" + 0.002*"phone"'),
 (3,
  '0.007*"said" + 0.004*"ha" + 0.003*"wa" + 0.002*"mr" + 0.002*"bn" + '
  '0.002*"year" + 0.001*"new" + 0.001*"company" + 0.001*"election" + '
  '0.001*"firm"'),
 (4,
  '0.006*"said" + 0.006*"wa" + 0.005*"ha" + 0.003*"film" + 0.002*"year" + '
  '0.002*"mr" + 0.002*"new" + 0.002*"time" + 0.001*"people" + 0.001*"game"'),
 (5,
  '0.005*"said" + 0.004*"ha" + 0.004*"wa" + 0.003*"mr" + 0.002*"year" + '
  '0.002*"party" + 0.001*"new" + 0.001*"say" + 0.001*"labour" + 0.001*"world"'),
 (6,
  '0.004*"said" + 0.004*"ha" + 0.003*"wa" + 0.002*"mr" + 0.002*"year" + '
  '0.001*"game" + 0.001*"wage" + 0.001*"people" + 0.001*"phone" + '
  '0.001*"mobile"'),
 (7,
  '0.007*"wa" + 0.006*"said" + 0.004*"ha" + 0.003*"mr" + 0.003*"year" + '
  '0.002*"game" + 0.002*"new" + 0.002*"world" + 0.002*"people" + 0.001*"time"'),
 (8,
  '0.005*"wa" + 0.004*"said" + 0.004*"best" + 0.003*"ha" + 0.002*"award" + '
  '0.002*"year" + 0.001*"won" + 0.001*"film" + 0.001*"new" + 0.001*"actor"'),
 (9,
  '0.004*"said" + 0.003*"wa" + 0.003*"year" + 0.002*"ha" + 0.002*"mr" + '
  '0.001*"new" + 0.001*"number" + 0.001*"uk" + 0.001*"music" + 0.001*"people"')]
```