DIKW ACADEMY

P40 TOPIC MODELLING

Automatic topic detection in large corpora

DATA INFORMATIE KENNIS WIJSHEID

DIKW ACADEMY

TOPICS

- Word frequencies
- Topic models
- (Sentiment analysis)

WORD FREQUENCIES

WORD FREQUENCIES

What is the most simple analysis we can do on a document?

Count the frequencies of all words used in the document!

Let's see were this takes us...

DOCUMENTS AS A BAG OF WORDS

- First step is to transform text into a ‘Bag-Of-Words’
- This is a matrix with all the unique words and their frequencies (how often they occur) per document
- Each word is a feature in this matrix

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

TERM FREQUENCY AND INVERSE DOCUMENT FREQUENCY

One way to measure how important a word is in a document is by counting the *term frequency* for this word.

This will result in a lot of words that occur very frequently but we know are not important like “the”, “is”, “of”.

So another way to assess the importance of a word in a corpus of documents is to look at the *inverse document frequency* of this word. Doing so will decrease weight for common words and increase weight for less common words.

So for inverse document frequency we define:

$$idf(term) = ln(\frac{n_{documents}}{n_{documentscontainingterm}})$$

And multiplying gives:

$$tfidf = tf * idf$$

TOP 2000 DATASET

In this class we will use the Top 2000 dataset.

It can be found in the zipfile data_for_windows.zip

WORDCLOUD

Which song is visualised in this image?

MAKE YOUR OWN WORDCLOUD

Install and import required modules

```
! pip install wordcloud
! pip install nltk
```

In [1]:
```
# we need to download some data from nltk
import nltk
# a gui screen will open to download relevant stuff
#
#nltk.download('punkt') # 'punkt' 'stopwords'
#nltk.download('stopwords') # 'punkt' 'stopwords' 'wordnet' 'omw-1.4'
#nltk.download()
```

In [2]:
```
# load all libraries we need
import wordcloud as wc
import matplotlib.pyplot as plt
from os.path import isfile, join
from os import listdir
import zipfile
from nltk.corpus.reader import PlaintextCorpusReader
from nltk.corpus import stopwords
```

EXTRACT AND LOAD TOP2000 DATA

Extract the data we will be using from the zip file:

In [3]:
```
# adjust to your likings
path_to_zip_file = "./data/top2000.zip"
directory_to_extract_to = "./data/t2000"
with zipfile.ZipFile(path_to_zip_file, 'r') as zip_ref:
    zip_ref.extractall(directory_to_extract_to)
```

In [4]:
```
# read one song
data_path_file_name = directory_to_extract_to + "/top2000/lyrics/The_Beatles_Can_t_Buy_Me_Love.tx
with open(data_path_file_name) as f:
    text_raw = f.read()
print(text_raw)
```

```
Can't buy me love, oh
Love, oh
Can't buy me love, oh

I'll buy you a diamond ring, my friend
If it makes you feel all right
I'll get you anything my friend
If it makes you feel all right

'Cause I don't care too much for money
For money can't buy me love

I'll give you all I've got to give
If you say you love me too
I may not have a lot to give
But what I got I'll give to you

I don't care too much for money
For money can't buy me love

Can't buy me love, oh
Everybody tells me so
Can't buy me love, oh
No, no, no, no

Say you don't need no diamond rings
And I'll be satisfied
Tell me that you want the kind of things
That money just can't buy

I don't care too much for money
Money can't buy me love, ow
```

DIKW ACADEMY

STOPWORDS

Stopwords are usually excluded, because they affect the result with less informative words

In [5]:
```
wc_stopwords = set(wc.STOPWORDS)
print(wc_stopwords)
```

```
{'hers', 'them', "we're", 'nor', 'am', 'an', 'had', 'all', 'http', 'as', 'else', 'me', 'he', "they're",
'why', 'com', "he's", 'their', "there's", 'with', 'further', 'also', 'here', 'r', 'get', "he'd", "wher
e's", 'did', "shan't", 'for', 'under', "aren't", 'these', 'my', 'what', "weren't", 'her', "i've", 'theref
ore', 'at', "it's", 'against', 'cannot', 'and', 'up', 'when', 'yourself', 'yourselves', "he'll", 'our',
'we', 'just', 'ours', 'own', "wasn't", "you've", 'how', "didn't", 'this', 'which', "mustn't", 'she', "the
y'd", 'was', "i'd", 'doing', 'where', "she'd", 'like', 'most', 'have', 'other', 'myself', 'itself', 'ove
r', 'they', 'between', 'his', "don't", 'down', 'hence', 'because', 'herself', 'should', 'is', 'being', 'a
bove', "shouldn't", 'ever', "we'd", "wouldn't", 'shall', 'can', "we've", 'but', "hadn't", 'ourselves', 'o
ught', 'you', 'yours', 'during', "who's", "they've", 'out', 'only', 'www', "doesn't", "you'd", 'are', 'in
to', 'or', "how's", "i'm", "that's", 'then', 'i', 'by', 'some', 'has', 'could', 'been', 'having', 'unti
l', 'while', "you'll", 'k', "couldn't", 'few', 'too', 'does', 'in', 'same', 'any', 'however', 'than', "yo
u're", 'to', 'each', 'do', 'be', 'it', 'such', "haven't", 'those', 'about', "here's", 'after', 'the', 'mo
re', "we'll", "she'll", 'so', 'its', 'theirs', 'of', 'whom', 'again', "why's", 'no', 'himself', 'would',
'him', 'otherwise', 'that', 'your', 'if', "i'll", "isn't", "what's", "hasn't", 'through', 'were', 'not',
"when's", "she's", 'before', "can't", 'themselves', 'there', 'off', "they'll", 'a', "won't", 'from', 'bel
ow', 'very', 'both', 'once', 'since', 'on', "let's", 'who'}
```

In [6]:

```
wordcloud = wc.WordCloud(stopwords = wc_stopwords,
                         max_words = 10,
                         collocations = False,
                         max_font_size=80).generate(text_raw)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

DATA STRUCTURES

Structuring text data can be done in different ways. This is worth contrasting with the ways text is often stored in text mining approaches.

- *String* : Text can, of course, be stored as strings.
- *Corpus* : These types of objects typically contain raw strings annotated with additional metadata and details.
- *Document-term matrix* : This is a sparse matrix describing a collection (i.e., a corpus) of documents with one row for each document and one column for each term. The value in the matrix is typically word count or tf-idf.
- *tidy text* : from the R language the concept of tidy data principles also holds for text

CORPUS

If we have to deal with a lot of documents we can create a structured object for it.

We will be using the nltk corpus reader package:
https://www.nltk.org/api/nltk.corpus.reader.html

Structure the text documents in a corpus can be done like so:

In [7]:
```
corpus_root = directory_to_extract_to + "/top2000/lyrics/"
file_ext = "txt"
file_ids = [f for f in listdir(corpus_root) if isfile(join(corpus_root, f)) and f.lower().endswit
corpus = PlaintextCorpusReader(corpus_root, file_ids)
print("The number of documents:", len(corpus.fileids()))
print("The number of sentences =", len(corpus.sents()))
print("The number of words =", len([word for sentence in corpus.sents() for word in sentence]))
print("The number of characters =", len([char for sentence in corpus.sents() for word in sentence
```

```
The number of documents: 1773
The number of sentences = 20414
The number of words = 579056
The number of characters = 1880943
```

DOCUMENT-TERM MATRIX

A document-term matrix contains terms with their frequencies of all documents in the corpus.

In [8]:
```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

count_vect = CountVectorizer(max_df=2)
# term document matrix (more efficient for large corpora)
term_document_matrix = count_vect.fit_transform([corpus.raw(i) for i in file_ids])
df_dtm = pd.DataFrame(term_document_matrix.toarray(), columns=count_vect.get_feature_names_out())
df_dtm['file_ids'] = file_ids
df_dtm=df_dtm.set_index('file_ids')
df_dtm
```

Out[8]:

file_ids	00	000	02	100	1000	11	125	14	15	16	...	évite	ééntje	êtes	ìpe	ìto	ómver	ôs	überfluss	überhaup
Neet_Oet_Lottum_Hald_Mich__s_Vas.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0
Muse_Psycho.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
The_Police_Every_Breath_You_Take.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Henk_Westbroek_Zelfs_Je_Naam_Is_Mooi.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Blink_182_All_The_Small_Things.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Bob_Dylan_Just_Like_A_Woman.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Jay_Z___Alicia_Keys_Empire_State_Of_Mind.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
Kensington_Streets.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
David_Bowie___Mick_Jagger_Dancing_In_The_Street.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
The_Monkees_Daydream_Believer.txt	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

1773 rows × 11044 columns

FREQUENT TERMS

Filter most frequent terms in the corpus

In [9]:
```
# FreqDist requires a list of words as input
# We will lowercase the text in each document in the corput, join it with the other documents int
# and finally split the string into words and store them in a list
freq = nltk.FreqDist(' '.join([corpus.raw(i).lower() for i in file_ids]).split())
top_words = freq.most_common(10)
top_words
```

Out[9]:
```
[('the', 17704),
 ('i', 14933),
 ('you', 14244),
 ('and', 10242),
 ('to', 9866),
 ('a', 9002),
 ('me', 6398),
 ('in', 6083),
 ('my', 5572),
 ('it', 5254)]
```

A lot of stopwords! What about ‘ain’ and ‘don’?

CLEAN TEXT

Let's clean the text from stopwords, whitespace, numbers and punctuation

There is also a package named textcleaner you can use

Based on this SO answer

In [10]:
```
# takes some time
df = pd.DataFrame(columns=['Text'])
df['text'] = [corpus.raw(i) for i in file_ids]
df['file_ids'] = file_ids

import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.corpus import stopwords
import re
# optional lemanize
lemmatizer = WordNetLemmatizer()
# optional stemmer
stemmer = PorterStemmer()

def preprocess(sentence):
    sentence=str(sentence)
    sentence = sentence.lower()
    sentence=sentence.replace('{html}',"")
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', sentence)
    rem_url=re.sub(r'http\S+', '',cleantext)
    rem_num = re.sub('[0-9]+', '', rem_url)
```

In [11]:
```
# check results
df.head()
```

Out[11]:

	Text	text	file_ids	clean_text
0	NaN	De kins wachte, de kins haope\nDe kins doéme d...	Neet_Oet_Lottum_Hald_Mich__s_Vas.txt	kin wacht kin haop kin doém det lök gedacht ke...
1	NaN	Love, it will get you nowhere\nYou are on your...	Muse_Psycho.txt	love get nowher lost wild come could use someo...
2	NaN	Every breath you take\nAnd every move you make...	The_Police_Every_Breath_You_Take.txt	everi breath take everi move make everi bond b...
3	NaN	Als jij je kleren aantrekt zonder haast\nEn ha...	Henk_Westbroek_Zelfs_Je_Naam_Is_Mooi.txt	al jij kleren aantrekt zonder haast haast zond...
4	NaN	All the small things\nTrue care, truth brings\...	Blink_182_All_The_Small_Things.txt	small thing true care truth bring take one lif...

In [12]:
```
corpus_clean = df[['file_ids','clean_text']]
```

In [13]:
```
df.head()
```

Out[13]:

	Text	text	file_ids	clean_text
0	NaN	De kins wachte, de kins haope\nDe kins doéme d...	Neet_Oet_Lottum_Hald_Mich__s_Vas.txt	kin wacht kin haop kin doém det lök gedacht ke...
1	NaN	Love, it will get you nowhere\nYou are on your...	Muse_Psycho.txt	love get nowher lost wild come could use someo...
2	NaN	Every breath you take\nAnd every move you make...	The_Police_Every_Breath_You_Take.txt	everi breath take everi move make everi bond b...
3	NaN	Als jij je kleren aantrekt zonder haast\nEn ha...	Henk_Westbroek_Zelfs_Je_Naam_Is_Mooi.txt	al jij kleren aantrekt zonder haast haast zond...
4	NaN	All the small things\nTrue care, truth brings\...	Blink_182_All_The_Small_Things.txt	small thing true care truth bring take one lif...

In [14]:
```
# example
corpus_clean['clean_text']
```

Out[14]:
```
0       kin wacht kin haop kin doém det lök gedacht ke...
1       love get nowher lost wild come could use someo...
2       everi breath take everi move make everi bond b...
3       al jij kleren aantrekt zonder haast haast zond...
4       small thing true care truth bring take one lif...
                              ...
1768    nobodi feel pain tonight stand insid rain ever...
1769    yeah yeah brooklyn tribeca right next deniro h...
1770    whenev say bound fall togeth stay get way get ...
1771    okay tokyo south america australia franc germa...
1772    number chip okay mean get excit man caus short...
Name: clean_text, Length: 1773, dtype: object
```

FIND POPULAR TERMS AFTER CLEANING

Popular terms in the Top 2000, notice that we now supply a dataframe column as input for transformation to count_vect:

In [15]:
```
freq = nltk.FreqDist(' '.join(corpus_clean['clean_text']).split())
topWords = freq.most_common(20)
topWords
```

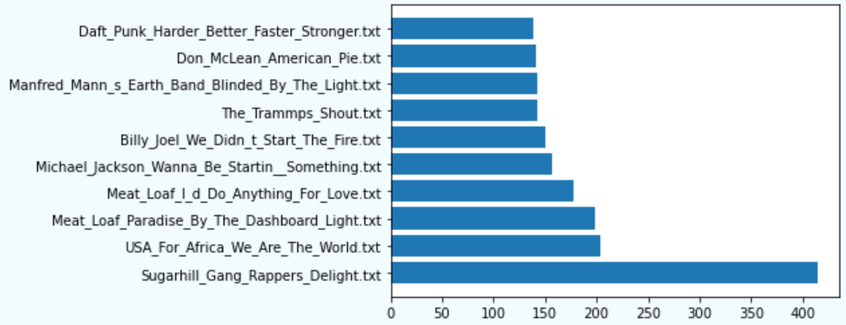
Out[15]:
```
[('love', 4028),
 ('know', 2882),
 ('yeah', 2505),
 ('like', 2140),
 ('come', 1998),
 ('get', 1787),
 ('time', 1785),
 ('got', 1716),
 ('one', 1654),
 ('feel', 1653),
 ('never', 1558),
 ('let', 1529),
 ('say', 1483),
 ('want', 1467),
 ('see', 1424),
 ('take', 1374),
 ('babi', 1344),
 ('make', 1332),
 ('way', 1263),
 ('day', 1236)]
```

LINES PER SONG

In [16]:
```
# here we are back to the raw corpus
newline_count_file = [[corpus.raw(i).count('\n'), i] for i in file_ids]
newline_count_file_sorted = sorted(newline_count_file, key=lambda x: -x[0])[0:10]
ys, xs = [*zip(*newline_count_file_sorted)]
```

In [17]:
```
import numpy as np
plt.barh(xs, ys)
```

Out[17]: <BarContainer object of 10 artists>



MOST USED WORD IN ONE SONG

In [18]:
```
df1 = pd.DataFrame(columns=['word', 'n', 'total'])
for i in file_ids:
    list_with_words = ' '.join([corpus.raw(i).lower()]).split()
    freq = nltk.FreqDist(list_with_words)
    df1.loc[i] = [freq.most_common(1)[0][0], freq.most_common(1)[0][1], len(list_with_words)]
df1.sort_values("n", ascending=False).head()
```

Out[18]:

	word	n	total
Michael_Jackson_Wanna_Be_Startin__Something.txt	ma	228	1109
Sugarhill_Gang_Rappers_Delight.txt	the	217	2961
Daft_Punk_Around_The_World.txt	around	144	432
Iggy_Pop_The_Passenger.txt	la	110	393
Pearl_Jam_Black.txt	doo	110	404

TERM FREQUENCIES

In [19]:

```
from collections import Counter

# example songs
song_list = ["Pearl_Jam_Black.txt", "James_Brown_Sex_Machine.txt",
             "The_Blues_Brothers_Everybody_Needs_Somebody_To_Love.txt","Justin_Timberlake_Cry_M

for song in song_list:
    cnt = Counter()
    total_words = len(corpus.raw(song).lower().split())

    for text in corpus.raw(song).lower().split():
        cnt[text] += 1
    # See most common ten words
    cnt.most_common(10)

    word_freq = pd.DataFrame(cnt.most_common(20), columns=['words', 'count'])
    word_freq["total_words"] = total_words
    word_freq["n_total"] = round(word_freq.apply(lambda row: row["count"] / row.total_words, axis
    #word_freq.head()
    fig, ax = plt.subplots(figsize=(12, 3))

    # Plot horizontal bar graph
    word_freq.sort_values(by='count').plot.bar(x='words',
                        y='count',
                        ax=ax,
                        color="brown")
    ax.set_title(song)
    plt.show()
```

Pearl Jam Black.txt

TOPIC MODELING

In [26]:
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import pandas as pd
import numpy as np
%matplotlib inline
```

In [27]:
```
from sklearn.feature_extraction import _stop_words
```

In [28]:
```
# max_features limits the number of features to use
vect = CountVectorizer(max_features=1000,ngram_range=(1,1),stop_words=['english','dutch'])
```

In [29]:
```
# build a document term matrix
dtm=vect.fit_transform(corpus_clean['clean_text'])
```

In [30]:
```
# document term matrix
dtm
```

Out[30]:
```
<1773x1000 sparse matrix of type '<class 'numpy.int64'>'
        with 75615 stored elements in Compressed Sparse Row format>
```

In [31]:
```
pd.DataFrame(dtm.toarray(),columns=vect.get_feature_names_out())
```

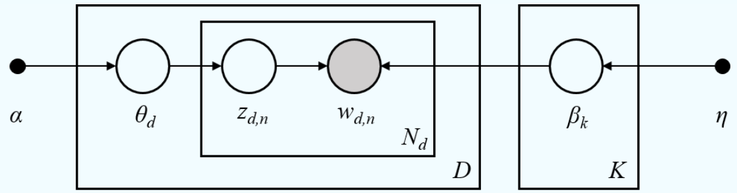
Out[31]:

	aaaahh	aah	aan	ach	across	act	afraid	age	ago	aha	...	ye	yeah	year	yellow	yesterday	yet	york	young	zeg	zijn
0	0	0	6	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	3	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	2	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

LATENT DIRICHLET ALLOCATION

Latent Dirichlet Allocation is a generative probabilistic model for collections of discrete dataset such as text corpora. It is also a topic model that is used for discovering abstract topics from a collection of documents.

The graphical model of LDA is a three-level generative model:



Note on notations presented in the graphical model above, which can be found in Hoffman et al. (2013):

- The corpus is a collection of D documents.
- A document is a sequence of N words.
- There are K topics in the corpus.
- The boxes represent repeated sampling.

In the graphical model, each node is a random variable and has a role in the generative process. A shaded node indicates an observed variable and an unshaded node indicates a hidden (latent) variable. In this case, words in the corpus are the only data that we observe.

DIKW ACADEMY

In [32]:
```
# how many topics do we want to find
lda=LatentDirichletAllocation(n_components=10)
```

In [33]:
```
# fit the model
lda.fit_transform(dtm)
```

Out[33]:
```
array([[1.56262392e-03, 1.03624936e-01, 1.56299061e-03, ...,
        1.56271803e-03, 1.56267122e-03, 1.56266175e-03],
       [3.68817201e-01, 2.67396693e-01, 4.83649739e-02, ...,
        4.61341254e-02, 2.63403393e-01, 1.17671610e-03],
       [5.43583446e-04, 5.43571828e-04, 5.43567978e-04, ...,
        5.43618962e-04, 1.88566322e-01, 5.43550863e-04],
       ...,
       [9.43656967e-04, 9.43710593e-04, 9.43559910e-04, ...,
        9.43592185e-04, 3.97556011e-01, 1.97644701e-01],
       [2.90212811e-01, 6.71720344e-01, 8.33413176e-04, ...,
        8.33712671e-04, 8.33507853e-04, 8.33523407e-04],
       [1.56281325e-03, 1.56288987e-03, 1.56291166e-03, ...,
        1.76164552e-01, 1.56298004e-03, 1.51678937e-01]])
```

VISUALIZATION OF TOPICS

DIKW ACADEMY

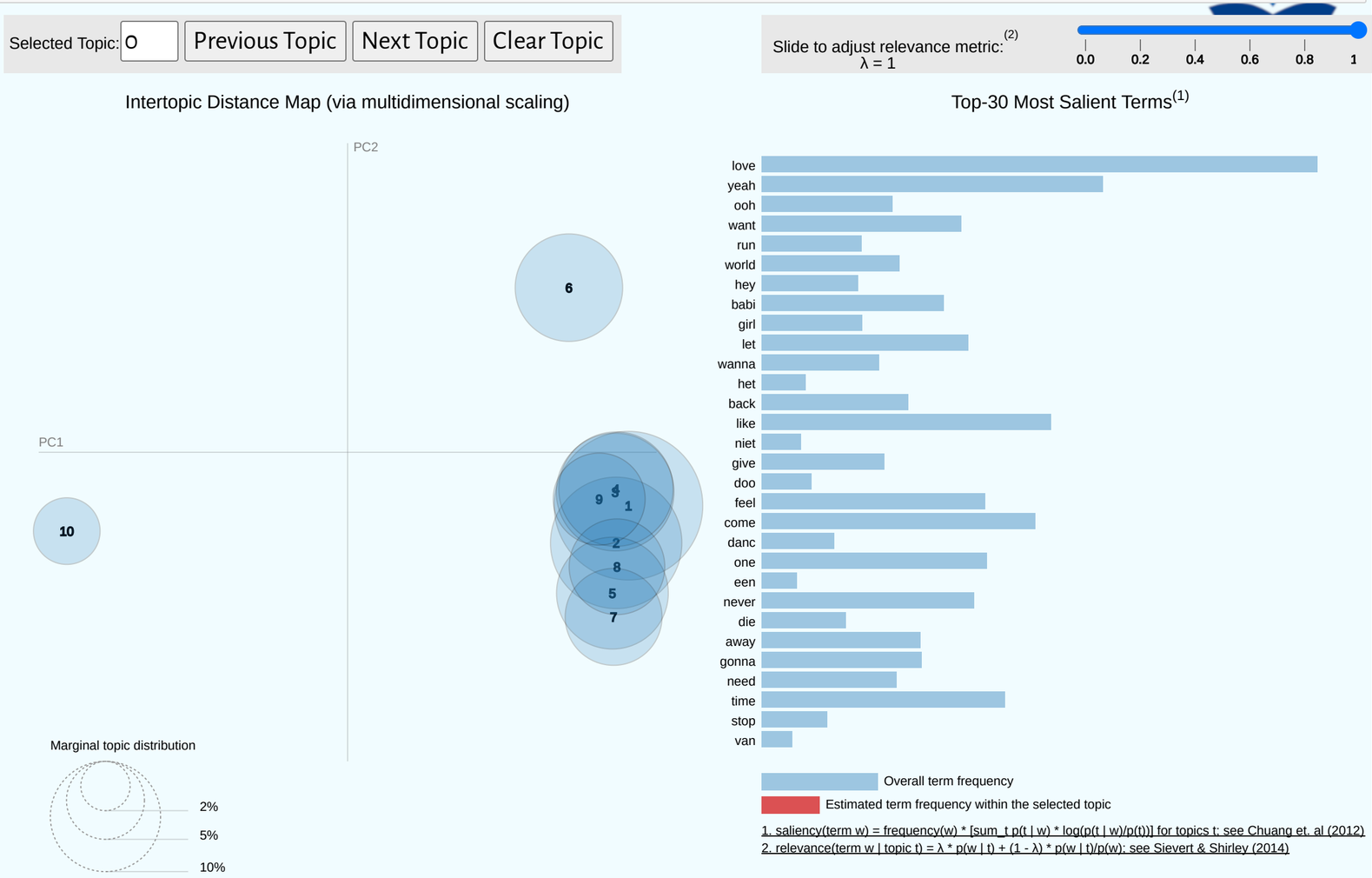
```
In [34]: import pyLDAvis
         import pyLDAvis.sklearn
         pyLDAvis.enable_notebook()
```

```
In [35]: zit=pyLDAvis.sklearn.prepare(lda,dtm,vect)
```

```
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarnin
g: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed i
n 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/pyLDAvis/_prepare.py:247: FutureWarning: In a
future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword
-only.
  default_term_info = default_term_info.sort_values(
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/past/builtins/misc.py:45: DeprecationWarning:
the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  from imp import reload
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/past/builtins/misc.py:45: DeprecationWarning:
the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  from imp import reload
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/past/builtins/misc.py:45: DeprecationWarning:
the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  from imp import reload
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/past/builtins/misc.py:45: DeprecationWarning:
the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  from imp import reload
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/past/builtins/misc.py:45: DeprecationWarning:
the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  from imp import reload
/home/hugo/anaconda3/envs/cdsp/lib/python3.9/site-packages/past/builtins/misc.py:45: DeprecationWarning:
the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  from imp import reload
```

In [36]:
```
pyLDAvis.display(zit)
```

Out[36]:



DIKW ACADEMY

In [37]:

```python
# function to get relevant words that define the topics
def get_model_topics(model, vectorizer, topics, n_top_words=5):
    word_dict = {}
    feature_names = vectorizer.get_feature_names_out()
    for topic_idx, topic in enumerate(model.components_):
        top_features_ind = topic.argsort()[:-n_top_words - 1:-1]
        top_features = [feature_names[i] for i in top_features_ind]
        word_dict[topics[topic_idx]] = top_features

    return pd.DataFrame(word_dict)
```

WHAT DO THE TOPICS MEAN?

So now we have found a latent clustering of relevant words into topics.

And we can use this to predict for a new doucment which topics are talked about in this document.

In [38]:
```
# what do the topics mean?
topics = ['T1','T2','T3','T4','T5','T6','T7','T8','T9','T10']
```

In [39]:
```
# the most relevant wordes describe the topic
get_model_topics(lda,vect,topics,n_top_words=8)
```

Out[39]:

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
0	back	ooh	feel	time	like	love	het	life	love	world
1	come	run	never	take	hey	yeah	niet	day	know	let
2	home	girl	give	say	know	want	een	tell	tri	one
3	man	danc	away	know	doo	babi	van	stand	night	got
4	get	littl	gonna	one	stop	know	die	sun	would	come
5	way	somebodi	wanna	like	get	need	dat	look	get	make
6	walk	yeah	love	whoa	rock	like	maar	sing	say	around
7	got	boy	let	make	heart	say	ich	like	never	yeah

USE THE LDA MODEL TO PREDICT WHAT TOPIC IS DISCUSSED IN A NEW DOCUMENT

In [40]:
```
def get_inference(model, vectorizer, topics, text, threshold):
    v_text = vectorizer.transform([text])
    score = model.transform(v_text)

    labels = set()
    for i in range(len(score[0])):
        if score[0][i] > threshold:
            labels.add(topics[i])

    if not labels:
        return 'None', -1, set()

    return topics[np.argmax(score)], score, labels
```

In [41]:
```
# this is a Dutch song
text = corpus_clean.iloc[3]['clean_text']
# there is topic that is about 'Dutch' songs ...
(topic, scores, topic_labels) = get_inference(lda, vect, topics, text, threshold=0.0)
topic
```

Out[41]:
```
'T7'
```

In [42]:
```
text
```

Out[42]:
```
'al jij kleren aantrekt zonder haast haast zonder erbij denken kijk naar een omgekeerd strip tea
volmaakt schoonheid elk handbeweg een gedicht elk buig al een roo die sluit schat van mij hemel h
zelf jouw schaduw kan mij verblinden du niet weg nooit bij weg maar al ooit verdwijnt laat mij da
inden zolang jou echt bij heb heb volmaakt liefd hier drink uit een pure waterbron slaap onder ee
```

DIKW ACADEMY

In [43]:
```
# get topic scores for each document
doc_topic_dist_unnormalized = np.matrix(lda.transform(dtm))

# normalize the distribution (only needed if you want to work with the probabilities)
doc_topic_dist = doc_topic_dist_unnormalized/doc_topic_dist_unnormalized.sum(axis=1)
```

In [44]:
```
# find the topic with highest probability
doc_topic_dist.argmax(axis=1)[0:10]
```

Out[44]:
```
matrix([[6],
        [0],
        [3],
        [6],
        [8],
        [6],
        [3],
        [8],
        [4],
        [8]])
```

EXERCISE : FILTER DUTCH SONGS ONLY AND DO TOPIC MODEL ON THE DUTCH SONGS

In [45]:
```
# aantal Nederlandse liedjes
nl_topic = 3  # this can be different every time as we cannot predict the order in which the topi
nl_filter = doc_topic_dist.argmax(axis=1)==nl_topic
corpus_clean_nl = corpus_clean[nl_filter.A1]
corpus_clean_nl.head()
```

Out[45]:

	file_ids	clean_text
2	The_Police_Every_Breath_You_Take.txt	everi breath take everi move make everi bond b...
6	Arcade_Fire_Reflektor.txt	trap prison prism light alon dark dark white f...
13	R_E_M__The_One_I_Love.txt	one goe one love one goe one left behind simpl...
49	Robbie_Williams_The_Road_To_Mandalay.txt	save drown sea beat beach love holiday noth fu...
59	Muse_Unintended.txt	could unintend choic live life extend could on...

In [46]:
```
# max_features limits the number of features to use
vect2 = CountVectorizer(max_features=500,ngram_range=(1,1),stop_words=['dutch'])
# build a document term matrix
dtm_nl = vect2.fit_transform(corpus_clean_nl['clean_text'])
# how many topics do we want to find
lda_nl = LatentDirichletAllocation(n_components=5)
# fit the model
lda_nl.fit_transform(dtm_nl)
```

Out[46]:
```
array([[9.95493471e-01, 1.12453043e-03, 1.12870432e-03, 1.12400738e-03,
        1.12928726e-03],
       [1.51781818e-03, 1.50946451e-03, 9.93947251e-01, 1.50429356e-03,
        1.52117292e-03],
       [6.17587970e-01, 3.11645318e-03, 3.16386044e-03, 3.72939591e-01,
        3.19212614e-03],
```