# TM2_TopicModels_professor

November 25, 2019

# 1 Exploring and undertanding documental databases with topic models

Version 1.0

Date: Nov 23, 2017

Authors:


- Jerónimo Arenas-García (jeronimo.arenas@uc3m.es)
- Jesús Cid-Sueiro (jcid@tsc.uc3m.es)


[1]:

```
# Common imports

%matplotlib inline
import matplotlib.pyplot as plt
import pylab

import numpy as np
# import pandas as pd
# import os
from os.path import isfile, join
# import scipy.io as sio
# import scipy
import zipfile as zp
# import shutil
# import difflib

import gensim
```

## 1.1 1. Corpus acquisition

In this block we will work with collections of text documents. The objectives will be:

- Find the most important topics in the collection and assign documents to topics
- Analyze the structure of the collection by means of graph analysis

We will work with a collection of research projects funded by the US National Science Foundation, that you can find under the `./data` directory. These files are publicly available from the NSF website.

### 1.1.1 1.1. Exploring file structure

NSF project information is provided in XML files. Projects are yearly grouped in `.zip` files, and each project is saved in a different XML file. To explore the structure of such files, we will use the file `160057.xml`. Parsing XML files in python is rather easy using the `ElementTree` module.

**1.1.1. File format** To start with, you can have a look at the contents of the example file. We are interested on the following information of each project:

- Project identifier
- Project Title
- Project Abstract
- Budget
- Starting Year (we will ignore project duration)
- Institution (name, zipcode, and state)

[2]:
```python
xmlfile = '../data/1600057.xml'

with open(xmlfile,'r') as fin:
    print(fin.read())
```


```
<?xml version="1.0" encoding="UTF-8"?>
<rootTag>
<Award>
<AwardTitle>Novel States in Spin-Orbit-Coupled and Correlated
Materials</AwardTitle>
<AwardEffectiveDate>08/01/2016</AwardEffectiveDate>
<AwardExpirationDate>12/31/2016</AwardExpirationDate>
<AwardAmount>306810</AwardAmount>
<AwardInstrument>
<Value>Continuing grant</Value>
</AwardInstrument>
<Organization>
<Code>03070000</Code>
<Directorate>
<LongName>Direct For Mathematical &amp; Physical Scien</LongName>
</Directorate>
<Division>
<LongName>Division Of Materials Research</LongName>
</Division>
</Organization>
<ProgramOfficer>
<SignBlockName>Tomasz Durakiewicz</SignBlockName>
```

</ProgramOfficer>
<AbstractNarration>Non-technical Abstract:<br/>Modern condensed matter physics research has produced novel materials with fundamental properties that underpin a remarkable number of cutting-edge technologies. It is now generally accepted that novel materials are necessary for critical advances in technologies and whoever discovers novel materials generally controls the science and technology of the future. Transition metal oxides have attracted enormous interest within both the basic and applied science communities. However, for many decades, the overwhelming balance of effort was focused on the 3d-elements (such as iron, copper, etc.) and their compounds; the heavier 4d- and 5d-elements (such as ruthenium, iridium, etc., which constitute two thirds of the d-elements listed in the Periodic Table) and their compounds have been largely ignored until recently. The principal investigator seeks to discover novel materials containing 4d- and/or 5d-elements and understand how they offer wide-ranging opportunities for the discovery of new physics and, ultimately, new device paradigms. This project also provides rigorous training to all students involved, focusing on synthesis and characterization techniques covering a broad spectrum of materials and experimental probes available in the principal investigator's laboratory.  <br/><br/>Technical Abstract:<br/>Physics driven by spin-orbit interactions is among the most important topics in contemporary condensed matter physics. Since the spin-orbit interaction is comparable to the on-site Coulomb and other relevant interactions, it creates a unique balance between competing interactions that drive complex behaviors and exotic states not observed in other materials. The project encompasses a systematic effort to elucidate physics of novel phenomena in spin-orbit-coupled and correlated materials and a rigorous search for new materials having exotic ground states. This project focuses on the following areas: (1) Novel phenomena at high pressures and high magnetic fields, (2) Unusual correlations between the insulating gap and magnetic transition in iridates and ruthenates, (3) Exotic metallic and superconducting states in iridates, (4) Mott insulators with "intermediate-strength" spin-orbit interaction and other competing energies, and (5) Single-crystal synthesis and search for novel materials. The principal investigator is one of a few key pioneers who have initiated seminal studies on iridates and, before that, ruthenates, and has comprehensive facilities and proven expertise for single-crystal synthesis and wide-ranging studies of structural, transport, magnetic, thermal and dielectric properties as functions of temperature, magnetic field, pressure and doping.</AbstractNarration>
<MinAmdLetterDate>08/05/2016</MinAmdLetterDate>
<MaxAmdLetterDate>08/05/2016</MaxAmdLetterDate>
<ARRAAmount/>
<AwardID>1600057</AwardID>
<Investigator>
<FirstName>Gang</FirstName>
<LastName>Cao</LastName>
<EmailAddress>gang.cao@colorado.edu</EmailAddress>
<StartDate>08/05/2016</StartDate>
<EndDate/>

```
<RoleCode>Principal Investigator</RoleCode>
</Investigator>
<Institution>
<Name>University of Kentucky Research Foundation</Name>
<CityName>Lexington</CityName>
<ZipCode>405260001</ZipCode>
<PhoneNumber>8592579420</PhoneNumber>
<StreetAddress>109 Kinkead Hall</StreetAddress>
<CountryName>United States</CountryName>
<StateName>Kentucky</StateName>
<StateCode>KY</StateCode>
</Institution>
<ProgramElement>
<Code>1710</Code>
<Text>CONDENSED MATTER PHYSICS</Text>
</ProgramElement>
<ProgramElement>
<Code>1712</Code>
<Text>DMR SHORT TERM SUPPORT</Text>
</ProgramElement>
<ProgramReference>
<Code>9150</Code>
<Text>EXP PROG TO STIM COMP RES</Text>
</ProgramReference>
</Award>
</rootTag>
```

#### 1.1.2. Parsing XML

XML is an inherently hierarchical data format, and the most natural way to represent it is with a tree. The `ElementTree` module has two classes for this purpose:

- `ElementTree` represents the whole XML document as a tree
- `Element` represents a single node in this tree

We can import XML data by reading an XML file:

[3]:
```python
import xml.etree.ElementTree as ET
root = ET.fromstring(open(xmlfile,'r').read())
```

The code below implements a function that parses the XML files and provides as its output a dictionary with fields:

```
project_code      (string)
title             (string)
abstract          (string)
budget            (float)
year              (string)
institution       (tuple with elements: name, zipcode, and statecode)
```

[4]:

```python
def parse_xmlproject(xml_string):
    """This function processess the specified XML field,
    and outputs a dictionary with the desired project information

    :xml_string: String with XML content
    :Returns: Dictionary with indicated files
    """

    root = ET.fromstring(xml_string)
    dictio = {}

    for child in root[0]:
        if child.tag.lower() == 'awardtitle':
            dictio['title'] = child.text
        elif child.tag.lower() == 'awardeffectivedate':
            dictio['year'] = str(child.text[-4:])
        elif child.tag.lower() == 'awardamount':
            dictio['budget'] = float(child.text)
        elif child.tag.lower() == 'abstractnarration':
            dictio['abstract'] = child.text
        elif child.tag.lower() == 'awardid':
            dictio['project_code'] = child.text
        elif child.tag.lower() == 'institution':
            #For the institution we have to access the children elements
            #and search for the name, zipcode, and statecode only
            name = ''
            zipcode = ''
            statecode = ''
            for child2 in child:
                if child2.tag.lower() == 'name':
                    name = child2.text
                elif child2.tag.lower() == 'zipcode':
                    zipcode = child2.text
                elif child2.tag.lower() == 'statecode':
                    statecode = child2.text
            dictio['institution'] = (name, zipcode, statecode)

    return dictio

parse_xmlproject(open(xmlfile,'r').read())
```

[4]:

```
{'title': 'Novel States in Spin-Orbit-Coupled and Correlated Materials',
 'year': '2016',
 'budget': 306810.0,
 'abstract': 'Non-technical Abstract:<br/>Modern condensed matter physics
research has produced novel materials with fundamental properties that underpin
a remarkable number of cutting-edge technologies. It is now generally accepted
```

```
that novel materials are necessary for critical advances in technologies and
whoever discovers novel materials generally controls the science and technology
of the future. Transition metal oxides have attracted enormous interest within
both the basic and applied science communities.  However, for many decades, the
overwhelming balance of effort was focused on the 3d-elements (such as iron,
copper, etc.) and their compounds; the heavier 4d- and 5d-elements (such as
ruthenium, iridium, etc., which constitute two thirds of the d-elements listed
in the Periodic Table) and their compounds have been largely ignored until
recently. The principal investigator seeks to discover novel materials
containing 4d- and/or 5d-elements and understand how they offer wide-ranging
opportunities for the discovery of new physics and, ultimately, new device
paradigms. This project also provides rigorous training to all students
involved, focusing on synthesis and characterization techniques covering a broad
spectrum of materials and experimental probes available in the principal
investigator\'s laboratory.  <br/><br/>Technical Abstract:<br/>Physics driven by
spin-orbit interactions is among the most important topics in contemporary
condensed matter physics. Since the spin-orbit interaction is comparable to the
on-site Coulomb and other relevant interactions, it creates a unique balance
between competing interactions that drive complex behaviors and exotic states
not observed in other materials. The project encompasses a systematic effort to
elucidate physics of novel phenomena in spin-orbit-coupled and correlated
materials and a rigorous search for new materials having exotic ground states.
This project focuses on the following areas: (1) Novel phenomena at high
pressures and high magnetic fields, (2) Unusual correlations between the
insulating gap and magnetic transition in iridates and ruthenates, (3) Exotic
metallic and superconducting states in iridates, (4) Mott insulators with
"intermediate-strength" spin-orbit interaction and other competing energies, and
(5) Single-crystal synthesis and search for novel materials. The principal
investigator is one of a few key pioneers who have initiated seminal studies on
iridates and, before that, ruthenates, and has comprehensive facilities and
proven expertise for single-crystal synthesis and wide-ranging studies of
structural, transport, magnetic, thermal and dielectric properties as functions
of temperature, magnetic field, pressure and doping.',
 'project_code': '1600057',
 'institution': ('University of Kentucky Research Foundation',
  '405260001',
  'KY')}
```

### 1.1.2 1.2. Building the dataset

Now, we will use the function you just implemented, to create a database that we will use throughout this module.

For simplicity, and given that the dataset is not too large, we will keep all projects in the RAM. The dataset will consist of a list containing the dictionaries associated to each of the considered projects in a time interval.

[5]:
```python
# Construct an iterator (or a list) for the years you want to work with
years = [2015, 2016]
datafiles_path = '../data/'
NSF_data = []

for year in years:

    zpobj = zp.ZipFile(join(datafiles_path, str(year)+'.zip'))
    for fileinzip in zpobj.namelist():
        if fileinzip.endswith('xml'):

            #Some files seem to be incorrectly parsed
            try:
                project_dictio = parse_xmlproject(zpobj.read(fileinzip))
                if project_dictio['abstract']:
                    NSF_data.append(project_dictio)
            except:
                pass
```

We will extract some characteristics of the constructed dataset:

[6]:
```python
print('Number of projects in dataset:', len(NSF_data))

# Budget
budget_data = list(map(lambda x: x['budget'], NSF_data))
print('Average budget of projects in dataset:', np.mean(budget_data))

# Institutions
insti_data = list(map(lambda x: x['institution'], NSF_data))
print('Number of unique institutions in dataset:', len(set(insti_data)))

# Counts per year
counts = dict()
for project in NSF_data:
    counts[project['year']] = counts.get(project['year'],0) + 1

print('Breakdown of projects by starting year:')
for el in counts:
    print(el, ':', counts[el])
```

```
Number of projects in dataset: 24342
Average budget of projects in dataset: 342411.6244351327
Number of unique institutions in dataset: 2786
Breakdown of projects by starting year:
2015 : 9039
2014 : 344
2016 : 12401
2017 : 2554
```

```
2013 : 1
2018 : 3
```

For the rest of this notebook, we will work with the abstracts only. The list of all abstract will be the corpus we will work with.

[7]:
```
corpus_raw = list(map(lambda x: x['abstract'], NSF_data))

abstractlen_data = list(map(lambda x: len(x), corpus_raw))
print('Average length of projects abstracts (in characters):', np.
 ↪mean(abstractlen_data))
```

```
Average length of projects abstracts (in characters): 2605.887807082409
```

## 1.2 2. Corpus Processing

Topic modelling algorithms process vectorized data. In order to apply them, we need to transform the raw text input data into a vector representation. To do so, we will remove irrelevant information from the text data and preserve as much relevant information as possible to capture the semantic content in the document collection.

Thus, we will proceed with the following steps:

1. Tokenization
2. Homogeneization, which includes:
   1. Removing capitalization.
   2. Removing non alphanumeric tokens (e.g. punktuation signs)
   3. Stemming/Lemmatization.
3. Cleaning
4. Vectorization

For the first steps, we will use some of the powerful methods available from the Natural Language Toolkit. In order to use the `word_tokenize` method from nltk, you might need to get the appropriate libraries using `nltk.download()`. You must select option "d) Download", and identifier "punkt"

[8]:
```
from nltk import download
# You should comment this code fragment if the package is already available.
# download('punkt')
# download('stopwords')
```

### 1.2.1 2.1. Corpus Processing

We will create a list that contains just the abstracts in the dataset. As the order of the elements in a list is fixed, it will be later straightforward to match the processed abstracts to metadata associated to their corresponding projects.

**Exercise 1**: Generate a corpus of processed documents. For each document in `corpus_raw` complete the following steps: 1. Tokenize. 2. Remove capitalization and non-alphanumeric tokens. 3.

Lemmatize 4. Remove the stopwords using the NLTK stopwords list.

[9]:
```
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from nltk.corpus import stopwords

# Maybe you can try the stemmer too.
# stemmer = SnowballStemmer('english')
wnl = WordNetLemmatizer()

stopwords_en = stopwords.words('english')

# Initialize ouput corpus
corpus_clean = []

ndocs = len(corpus_raw)
for n, text in enumerate(corpus_raw):
    if not n%100:
        print('\rTokenizing document', n, 'out of', ndocs, end='', flush=True)

    # Tokenize each text entry.
    # tokens = <FILL IN>
    tokens = word_tokenize(text)
    # tokens_filtered = <FILL IN>
    tokens_filtered = [el.lower() for el in tokens if el.isalnum()]
    # tokens_lemmatized = <FILL IN>
    tokens_lemmatized = [wnl.lemmatize(el) for el in tokens_filtered]
    # tokens_clean = <FILL IN>
    tokens_clean = [token for token in tokens_lemmatized if token not in␣
↪stopwords_en]

    # Add the new token list as a new element to corpus_clean (that will be a␣
↪list of lists)
    # corpus_clean.<FILL IN>
    corpus_clean.append(tokens_clean)

print('\n\n The corpus has been tokenized. Check the result for the first␣
↪abstract:')
print(corpus_raw[0])
print(corpus_clean[0])
```

```
Tokenizing document 24300 out of 24342

 The corpus has been tokenized. Check the result for the first abstract:
The past few years have seen unprecedented growth in mobile data consumption.
Powered in large part by the rapid adoption of smart phones and tablets, the
growth in wireless data creates phenomenal challenges for the wireless industry,
which has been unable to meet the demand for rich mobile content through
```

cellular networks. This has led to the investigation of solutions by network operators that aim to utilize WiFi radios present in these mobile devices to deliver content without using the cellular radio links, also known as content-offloading.  Industry-led approaches aim to utilize WiFi infrastructure in the form of access points to offload this content, but these have various deployment issues. Research has lately focused on the potential of proximity-based peer content sharing, since proximity enables low-power, high speed data exchanges which in turn allows mobile devices to proactively share data with one another. No large-scale study using real-world situations has established the potential for such content-sharing to provide capacity gains, until now. This proposal aims to conduct solid, preliminary pilot studies evaluating several of the foundational claims in this area, specifically as to whether sufficient potential exists in the right time, right place, and with reasonably viable deployment scenarios. The proposed work will gather and evaluate pilot data in both highly amenable environments (WiFi across multiple tailgate offerings of hundreds of users) as well as more challenging environments (daily commuter trains to / from Chicago).<br/><br/>The broader impact of the work will be to either demonstrate the potential viability for proximity-based solutions or to present compelling evidence that such proximity solutions are unlikely to yield significant benefits. Further broader impacts for the work include data sharing capabilities with respect to temporal characterizations of redundancy across mobile devices in several real-world scenarios.
['past', 'year', 'seen', 'unprecedented', 'growth', 'mobile', 'data', 'consumption', 'powered', 'large', 'part', 'rapid', 'adoption', 'smart', 'phone', 'tablet', 'growth', 'wireless', 'data', 'creates', 'phenomenal', 'challenge', 'wireless', 'industry', 'ha', 'unable', 'meet', 'demand', 'rich', 'mobile', 'content', 'cellular', 'network', 'ha', 'led', 'investigation', 'solution', 'network', 'operator', 'aim', 'utilize', 'wifi', 'radio', 'present', 'mobile', 'device', 'deliver', 'content', 'without', 'using', 'cellular', 'radio', 'link', 'also', 'known', 'approach', 'aim', 'utilize', 'wifi', 'infrastructure', 'form', 'access', 'point', 'offload', 'content', 'various', 'deployment', 'issue', 'research', 'ha', 'lately', 'focused', 'potential', 'peer', 'content', 'sharing', 'since', 'proximity', 'enables', 'high', 'speed', 'data', 'exchange', 'turn', 'allows', 'mobile', 'device', 'proactively', 'share', 'data', 'one', 'another', 'study', 'using', 'situation', 'ha', 'established', 'potential', 'provide', 'capacity', 'gain', 'proposal', 'aim', 'conduct', 'solid', 'preliminary', 'pilot', 'study', 'evaluating', 'several', 'foundational', 'claim', 'area', 'specifically', 'whether', 'sufficient', 'potential', 'exists', 'right', 'time', 'right', 'place', 'reasonably', 'viable', 'deployment', 'scenario', 'proposed', 'work', 'gather', 'evaluate', 'pilot', 'data', 'highly', 'amenable', 'environment', 'wifi', 'across', 'multiple', 'tailgate', 'offering', 'hundred', 'user', 'well', 'challenging', 'environment', 'daily', 'commuter', 'train', 'chicago', 'broader', 'impact', 'work', 'either', 'demonstrate', 'potential', 'viability', 'solution', 'present', 'compelling', 'evidence', 'proximity', 'solution', 'unlikely', 'yield', 'significant', 'benefit', 'broader', 'impact', 'work', 'include', 'data', 'sharing', 'capability', 'respect', 'temporal', 'characterization', 'redundancy', 'across', 'mobile', 'device', 'several', 'scenario']

### 1.2.2 2.4. Vectorization

Up to this point, we have transformed the raw text collection in a list of documents, where each documen is a collection of the words that are most relevant for semantic analysis. Now, we need to convert these data (a list of token lists) into a numerical representation (a list of vectors, or a matrix). To do so, we will start using the tools provided by the `gensim` library.

As a first step, we create a dictionary containing all tokens in our text corpus, and assigning an integer identifier to each one of them.

[10]:
```python
# Create dictionary of tokens
D = gensim.corpora.Dictionary(corpus_clean)
n_tokens = len(D)

print('The dictionary contains', n_tokens, 'terms')
print('First terms in the dictionary:')
for n in range(10):
    print(str(n), ':', D[n])
```

```
The dictionary contains 60691 terms
First terms in the dictionary:
0 : access
1 : across
2 : adoption
3 : aim
4 : allows
5 : also
6 : amenable
7 : another
8 : approach
9 : area
```

We can also filter out terms that appear in too few or too many of the documents in the dataset:

[11]:
```python
no_below = 5 #Minimum number of documents to keep a term in the dictionary
no_above = .75 #Maximum proportion of documents in which a term can appear to be␣
 ↪kept in the dictionary
D.filter_extremes(no_below=no_below, no_above=no_above, keep_n=25000)
n_tokens = len(D)

print('The dictionary contains', n_tokens, 'terms')
print('First terms in the dictionary:')
for n in range(10):
    print(str(n), ':', D[n])
```

```
The dictionary contains 20045 terms
First terms in the dictionary:
0 : access
1 : across
```

```
2 : adoption
3 : aim
4 : allows
5 : also
6 : amenable
7 : another
8 : approach
9 : area
```

In the second step, let us create a numerical version of our corpus using the `doc2bow` method. In general, `D.doc2bow(token_list)` transforms any list of tokens into a list of tuples `(token_id, n)`, one per each token in `token_list`, where `token_id` is the token identifier (according to dictionary `D`) and `n` is the number of occurrences of such token in `token_list`.

**Exercise 2**: Apply the `doc2bow` method from gensim dictionary `D`, to all tokens in every document in `clean_abstracts`. The result must be a new list named `corpus_bow` where each element is a list of tuples `(token_id, number_of_occurrences)`.

[12]:
```python
# corpus_bow = <FILL IN>
corpus_bow = [D.doc2bow(doc) for doc in corpus_clean]
```

At this point, it is good to make sure to understand what has happened. In `clean_abstracts` we had a list of token lists. With it, we have constructed a Dictionary, `D`, which assigns an integer identifier to each token in the corpus. After that, we have transformed each article (in `clean_abstracts`) in a list tuples `(id, n)`.

[13]:
```python
print('Original document (after cleaning):')
print(corpus_clean[0])
print('Sparse vector representation (first 10 components):')
print(corpus_bow[0][:10])
print('Word counts for the first project (first 10 components):')
print(list(map(lambda x: (D[x[0]], x[1]), corpus_bow[0][:10])))
```

```
Original document (after cleaning):
['past', 'year', 'seen', 'unprecedented', 'growth', 'mobile', 'data',
'consumption', 'powered', 'large', 'part', 'rapid', 'adoption', 'smart',
'phone', 'tablet', 'growth', 'wireless', 'data', 'creates', 'phenomenal',
'challenge', 'wireless', 'industry', 'ha', 'unable', 'meet', 'demand', 'rich',
'mobile', 'content', 'cellular', 'network', 'ha', 'led', 'investigation',
'solution', 'network', 'operator', 'aim', 'utilize', 'wifi', 'radio', 'present',
'mobile', 'device', 'deliver', 'content', 'without', 'using', 'cellular',
'radio', 'link', 'also', 'known', 'approach', 'aim', 'utilize', 'wifi',
'infrastructure', 'form', 'access', 'point', 'offload', 'content', 'various',
'deployment', 'issue', 'research', 'ha', 'lately', 'focused', 'potential',
'peer', 'content', 'sharing', 'since', 'proximity', 'enables', 'high', 'speed',
'data', 'exchange', 'turn', 'allows', 'mobile', 'device', 'proactively',
'share', 'data', 'one', 'another', 'study', 'using', 'situation', 'ha',
'established', 'potential', 'provide', 'capacity', 'gain', 'proposal', 'aim',
'conduct', 'solid', 'preliminary', 'pilot', 'study', 'evaluating', 'several',
```

```
'foundational', 'claim', 'area', 'specifically', 'whether', 'sufficient',
'potential', 'exists', 'right', 'time', 'right', 'place', 'reasonably',
'viable', 'deployment', 'scenario', 'proposed', 'work', 'gather', 'evaluate',
'pilot', 'data', 'highly', 'amenable', 'environment', 'wifi', 'across',
'multiple', 'tailgate', 'offering', 'hundred', 'user', 'well', 'challenging',
'environment', 'daily', 'commuter', 'train', 'chicago', 'broader', 'impact',
'work', 'either', 'demonstrate', 'potential', 'viability', 'solution',
'present', 'compelling', 'evidence', 'proximity', 'solution', 'unlikely',
'yield', 'significant', 'benefit', 'broader', 'impact', 'work', 'include',
'data', 'sharing', 'capability', 'respect', 'temporal', 'characterization',
'redundancy', 'across', 'mobile', 'device', 'several', 'scenario']
Sparse vector representation (first 10 components):
[(0, 1), (1, 2), (2, 1), (3, 3), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1)]
Word counts for the first project (first 10 components):
[('access', 1), ('across', 2), ('adoption', 1), ('aim', 3), ('allows', 1),
('also', 1), ('amenable', 1), ('another', 1), ('approach', 1), ('area', 1)]
```

Note that we can interpret each element of corpus_bow as a `sparse_vector`. For example, a list of tuples

```
[(0, 1), (3, 3), (5,2)]
```

for a dictionary of 10 elements can be represented as a vector, where any tuple `(id, n)` states that position `id` must take value `n`. The rest of positions must be zero.

```
[1, 0, 0, 3, 0, 2, 0, 0, 0, 0]
```

These sparse vectors will be the inputs to the topic modeling algorithms.

As a summary, the following variables will be relevant for the next chapters: * `D`: A gensim dictionary. Term strings can be accessed using the numeric identifiers. For instance, `D[0]` contains the string corresponding to the first position in the BoW representation. * `corpus_bow`: BoW corpus. A list containing an entry per project in the dataset, and consisting of the (sparse) BoW representation for the abstract of that project. * `NSF_data`: A list containing an entry per project in the dataset, and consisting of metadata for the projects in the dataset

The way we have constructed the `corpus_bow` variable guarantees that the order is preserved, so that the projects are listed in the same order in the lists `corpus_bow` and `NSF_data`.

Before starting with the semantic analyisis, it is interesting to observe the token distribution for the given corpus.

[14]:
```python
# SORTED TOKEN FREQUENCIES (I):
# Create a "flat" corpus with all tuples in a single list
corpus_bow_flat = [item for sublist in corpus_bow for item in sublist]

# Initialize a numpy array that we will use to count tokens.
# token_count[n] should store the number of ocurrences of the n-th token, D[n]
token_count = np.zeros(n_tokens)

# Count the number of occurrences of each token.
```

```python
for x in corpus_bow_flat:
    # Update the proper element in token_count
    # scode: <FILL IN>
    token_count[x[0]] += x[1]

# Sort by decreasing number of occurences
ids_sorted = np.argsort(- token_count)
tf_sorted = token_count[ids_sorted]
```

`ids_sorted` is a list of all token ids, sorted by decreasing number of occurrences in the whole corpus. For instance, the most frequent term is

[15]:
```python
print(D[ids_sorted[0]])
```

```
student
```

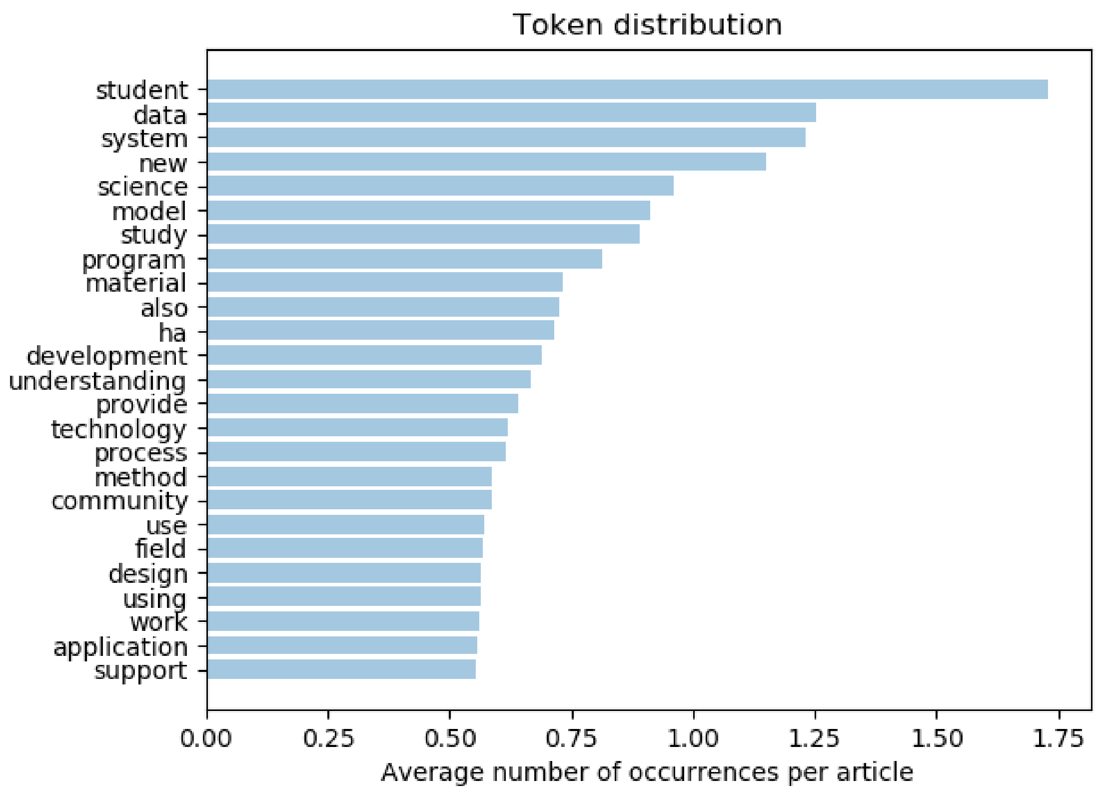
which appears

[16]:
```python
print("{0} times in the whole corpus".format(tf_sorted[0]))
```

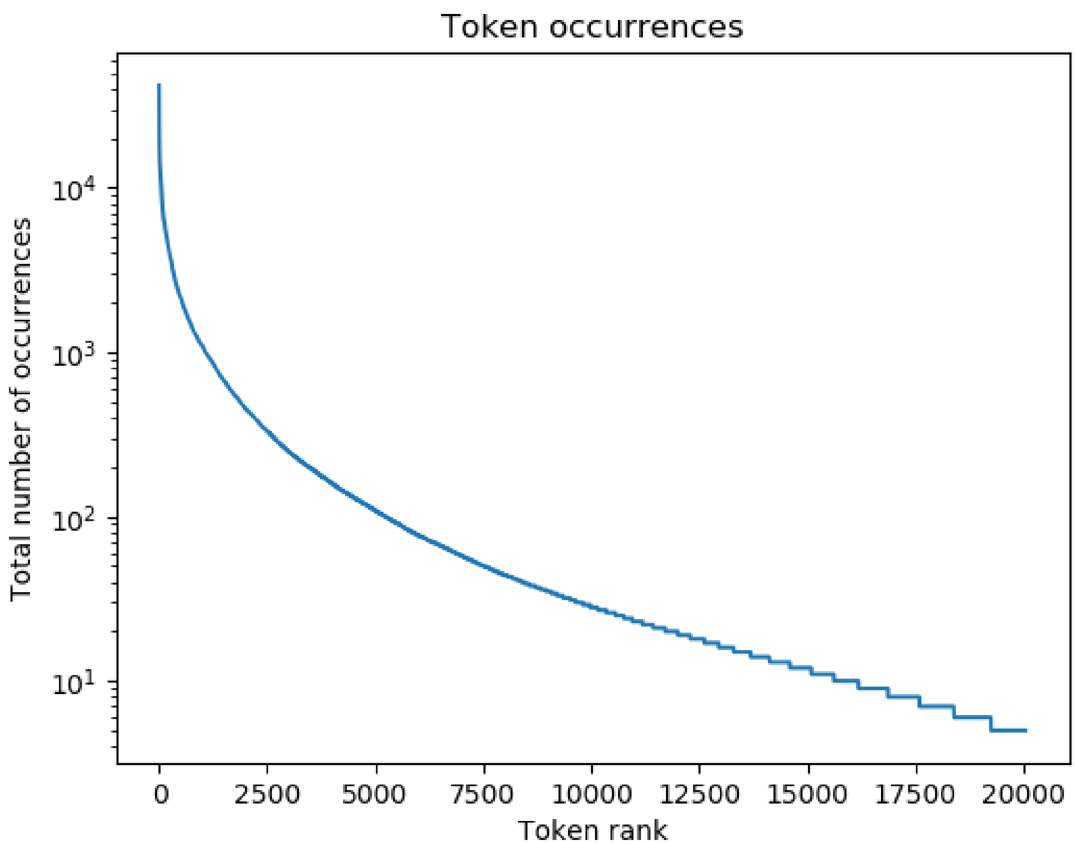
```
42100.0 times in the whole corpus
```

In the following we plot the most frequent terms in the corpus.

[17]:
```python
# SORTED TOKEN FREQUENCIES (II):
plt.rcdefaults()

# Example data
n_art = len(NSF_data)
n_bins = 25
hot_tokens = [D[i] for i in ids_sorted[n_bins-1::-1]]
y_pos = np.arange(len(hot_tokens))
z = tf_sorted[n_bins-1::-1]/n_art

plt.figure()
plt.barh(y_pos, z, align='center', alpha=0.4)
plt.yticks(y_pos, hot_tokens)
plt.xlabel('Average number of occurrences per article')
plt.title('Token distribution')
plt.show()
```




[18]:
```
# SORTED TOKEN FREQUENCIES:

# Example data
plt.figure()
plt.semilogy(tf_sorted)
plt.ylabel('Total number of occurrences')
plt.xlabel('Token rank')
plt.title('Token occurrences')
plt.show()
```




[19]:
```
# cold_tokens = <FILL IN>
cold_tokens = [D[i] for i in ids_sorted if tf_sorted[i]==1]

print("There are {0} cold tokens, which represent {1}% of the total number of␣
 ↪tokens in the dictionary".format(
    len(cold_tokens), float(len(cold_tokens))/n_tokens*100))
```

```
There are 0 cold tokens, which represent 0.0% of the total number of tokens in
the dictionary
```

### 1.2.3 2.5. Dictionary properties

As a final comment, note that gensim dictionaries contain a method `dfs` to compute the word counts automatically. In the code below we build a list `all_counts` that contains tuples (terms, document_counts).

[20]:
```
all_counts = [(D[el], D.dfs[el]) for el in D.dfs]
all_counts = sorted(all_counts, key=lambda x: x[1])
```

## 1.3 3. Topic Modeling

There are several implementations of the LDA topic model in python:

- Python library `lda`.
- Gensim module: `gensim.models.ldamodel.LdaModel`
- Sci-kit Learn module: `sklearn.decomposition`

In the following sections we explore the use of `gensim`

### 1.3.1 3.1. Training a topic model using Gensim LDA

Since we already have computed the dictionary and documents BoW representation using Gensim, computing the topic model is straightforward using the `LdaModel()` function. Please, refer to Gensim API documentation for more information on the different parameters accepted by the function:

**Exercise 3**: Create an LDA model with the 50 topics using `corpus_bow` and the dictionary, `D`.

[21]:
```
num_topics = 50

# This might take some time...
# ldag = gensim.models.ldamodel.LdaModel(<FILL IN>)
ldag = gensim.models.ldamodel.LdaModel(corpus=corpus_bow, id2word=D,␣
↪num_topics=num_topics)
```

### 1.3.2 3.2. LDA model visualization

Gensim provides a basic visualization of the obtained topics:

[22]:
```
ldag.print_topics(num_topics=-1, num_words=10)
```

[22]:
```
[(0,
  '0.105*"workshop" + 0.012*"science" + 0.011*"discussion" + 0.010*"researcher"
+ 0.010*"together" + 0.010*"participant" + 0.010*"challenge" + 0.008*"field" +
0.007*"new" + 0.007*"identify"'),
 (1,
  '0.041*"energy" + 0.018*"building" + 0.017*"model" + 0.016*"storage" +
0.015*"system" + 0.011*"uncertainty" + 0.010*"grid" + 0.009*"method" +
0.008*"methodology" + 0.007*"statistical"'),
 (2,
  '0.062*"earthquake" + 0.035*"wave" + 0.031*"fault" + 0.027*"seismic" +
0.014*"seismicity" + 0.013*"oklahoma" + 0.011*"induced" + 0.010*"data" +
0.007*"region" + 0.007*"acoustic"'),
 (3,
  '0.020*"earth" + 0.015*"mantle" + 0.012*"rock" + 0.010*"fluid" +
0.009*"process" + 0.009*"seismic" + 0.008*"deformation" + 0.007*"composition" +
0.007*"zone" + 0.006*"mineral"'),
```

```
 (4,
  '0.041*"theory" + 0.019*"group" + 0.015*"geometry" + 0.014*"study" +
0.012*"number" + 0.011*"problem" + 0.010*"area" + 0.010*"algebraic" + 0.010*"ha"
+ 0.010*"pi"'),
 (5,
  '0.033*"carbon" + 0.029*"soil" + 0.019*"water" + 0.017*"lake" +
0.014*"ecosystem" + 0.013*"gas" + 0.010*"land" + 0.009*"methane" +
0.008*"microbial" + 0.008*"forest"'),
 (6,
  '0.024*"change" + 0.010*"environmental" + 0.009*"response" + 0.007*"coastal" +
0.007*"marine" + 0.007*"study" + 0.007*"ecosystem" + 0.007*"effect" +
0.007*"climate" + 0.006*"population"'),
 (7,
  '0.017*"animal" + 0.016*"biology" + 0.014*"human" + 0.012*"biological" +
0.011*"body" + 0.010*"function" + 0.008*"study" + 0.007*"mechanism" +
0.007*"hypothesis" + 0.007*"student"'),
 (8,
  '0.018*"technology" + 0.013*"manufacturing" + 0.012*"patient" + 0.012*"cost" +
0.009*"potential" + 0.009*"product" + 0.008*"phase" + 0.008*"device" +
0.007*"system" + 0.007*"medical"'),
 (9,
  '0.020*"energy" + 0.013*"ion" + 0.010*"electron" + 0.009*"surface" +
0.008*"gas" + 0.008*"graphene" + 0.008*"high" + 0.008*"using" +
0.008*"radiation" + 0.008*"liquid"'),
 (10,
  '0.052*"network" + 0.028*"system" + 0.024*"graph" + 0.019*"algorithm" +
0.018*"distributed" + 0.012*"node" + 0.011*"design" + 0.011*"control" +
0.010*"power" + 0.010*"thrust"'),
 (11,
  '0.072*"brain" + 0.032*"neural" + 0.025*"neuroscience" + 0.015*"neuron" +
0.010*"visualization" + 0.009*"network" + 0.009*"spatial" + 0.009*"activity" +
0.009*"function" + 0.009*"mri"'),
 (12,
  '0.047*"plant" + 0.016*"specie" + 0.010*"crop" + 0.008*"tree" + 0.008*"forest"
+ 0.007*"insect" + 0.007*"diversity" + 0.006*"study" + 0.006*"data" +
0.005*"leaf"'),
 (13,
  '0.023*"network" + 0.018*"wireless" + 0.018*"communication" + 0.016*"service"
+ 0.016*"mobile" + 0.013*"user" + 0.012*"technology" + 0.012*"spectrum" +
0.010*"device" + 0.007*"proposed"'),
 (14,
  '0.047*"equation" + 0.033*"problem" + 0.026*"mathematical" + 0.022*"solution"
+ 0.019*"differential" + 0.013*"analysis" + 0.012*"nonlinear" + 0.012*"partial"
+ 0.012*"method" + 0.011*"theory"'),
 (15,
  '0.020*"particle" + 0.016*"flow" + 0.014*"model" + 0.010*"dynamic" +
0.009*"fluid" + 0.009*"simulation" + 0.009*"study" + 0.008*"field" +
```

```
0.008*"cloud" + 0.008*"understanding"'),
 (16,
  '0.042*"ocean" + 0.038*"climate" + 0.021*"ice" + 0.019*"model" +
0.016*"change" + 0.013*"sea" + 0.011*"temperature" + 0.011*"variability" +
0.010*"region" + 0.010*"atmospheric"'),
 (17,
  '0.042*"student" + 0.025*"program" + 0.021*"university" + 0.019*"support" +
0.018*"science" + 0.012*"graduate" + 0.011*"faculty" + 0.010*"institution" +
0.009*"community" + 0.009*"opportunity"'),
 (18,
  '0.029*"imaging" + 0.022*"instrument" + 0.018*"laser" + 0.016*"measurement" +
0.011*"light" + 0.011*"instrumentation" + 0.010*"optical" + 0.009*"technique" +
0.008*"source" + 0.008*"resolution"'),
 (19,
  '0.038*"disease" + 0.036*"virus" + 0.024*"infection" + 0.017*"pathogen" +
0.011*"host" + 0.011*"model" + 0.011*"viral" + 0.011*"spread" + 0.010*"health" +
0.010*"transmission"'),
 (20,
  '0.033*"security" + 0.016*"policy" + 0.014*"system" + 0.012*"risk" +
0.011*"privacy" + 0.011*"health" + 0.010*"memory" + 0.009*"food" +
0.009*"infrastructure" + 0.008*"decision"'),
 (21,
  '0.027*"device" + 0.016*"material" + 0.013*"new" + 0.011*"optical" +
0.010*"crystal" + 0.009*"soft" + 0.009*"application" + 0.009*"structure" +
0.007*"design" + 0.007*"circuit"'),
 (22,
  '0.024*"physic" + 0.013*"experiment" + 0.011*"nuclear" + 0.010*"particle" +
0.010*"new" + 0.008*"matter" + 0.007*"detector" + 0.006*"student" +
0.006*"neutron" + 0.006*"measurement"'),
 (23,
  '0.018*"stress" + 0.016*"boundary" + 0.013*"wall" + 0.012*"plate" +
0.012*"grain" + 0.011*"crack" + 0.010*"model" + 0.010*"tectonic" +
0.010*"fracture" + 0.009*"loading"'),
 (24,
  '0.019*"social" + 0.011*"study" + 0.008*"people" + 0.008*"data" +
0.007*"understanding" + 0.006*"information" + 0.006*"community" +
0.006*"individual" + 0.005*"policy" + 0.005*"public"'),
 (25,
  '0.116*"material" + 0.033*"property" + 0.016*"thermal" + 0.014*"composite" +
0.012*"new" + 0.011*"application" + 0.010*"film" + 0.009*"thin" + 0.009*"high" +
0.008*"fundamental"'),
 (26,
  '0.030*"arctic" + 0.023*"fellowship" + 0.019*"ocean" + 0.018*"fellow" +
0.017*"scientist" + 0.015*"fossil" + 0.013*"science" + 0.012*"collection" +
0.011*"training" + 0.010*"postdoctoral"'),
 (27,
  '0.078*"water" + 0.015*"sediment" + 0.014*"data" + 0.013*"record" +
```

```
0.012*"region" + 0.011*"crust" + 0.011*"precipitation" + 0.010*"surface" +
0.010*"river" + 0.009*"estimate"'),
 (28,
  '0.055*"conference" + 0.022*"researcher" + 0.018*"2016" + 0.015*"meeting" +
0.013*"held" + 0.011*"field" + 0.011*"2017" + 0.011*"student" +
0.010*"international" + 0.010*"topic"'),
 (29,
  '0.024*"stem" + 0.024*"learning" + 0.020*"student" + 0.018*"education" +
0.016*"science" + 0.010*"development" + 0.008*"program" + 0.008*"practice" +
0.007*"teacher" + 0.007*"community"'),
 (30,
  '0.046*"3d" + 0.030*"battery" + 0.021*"image" + 0.013*"volunteer" +
0.011*"microscope" + 0.010*"asymmetric" + 0.008*"electrode" + 0.007*"new" +
0.007*"imaging" + 0.007*"resolution"'),
 (31,
  '0.024*"state" + 0.016*"site" + 0.013*"mt" + 0.012*"mexico" + 0.010*"student"
+ 0.009*"alaska" + 0.008*"early" + 0.008*"court" + 0.008*"africa" +
0.007*"northern"'),
 (32,
  '0.018*"industry" + 0.015*"engineering" + 0.015*"technology" + 0.011*"center"
+ 0.010*"cybersecurity" + 0.009*"design" + 0.009*"new" + 0.009*"innovation" +
0.007*"need" + 0.007*"education"'),
 (33,
  '0.057*"student" + 0.035*"engineering" + 0.022*"program" + 0.018*"school" +
0.016*"stem" + 0.014*"education" + 0.014*"undergraduate" + 0.013*"college" +
0.013*"career" + 0.013*"science"'),
 (34,
  '0.015*"process" + 0.014*"size" + 0.010*"coating" + 0.009*"surface" +
0.008*"residual" + 0.008*"metal" + 0.008*"design" + 0.008*"nanoparticles" +
0.007*"performance" + 0.006*"growth"'),
 (35,
  '0.016*"data" + 0.011*"hazard" + 0.010*"community" + 0.010*"network" +
0.008*"workflow" + 0.007*"resource" + 0.007*"system" + 0.006*"violence" +
0.006*"provide" + 0.006*"archaeological"'),
 (36,
  '0.041*"language" + 0.022*"star" + 0.016*"galaxy" + 0.012*"mesa" +
0.010*"astronomy" + 0.009*"team" + 0.009*"black" + 0.008*"speech" +
0.008*"planet" + 0.007*"telescope"'),
 (37,
  '0.068*"cell" + 0.014*"cancer" + 0.013*"tissue" + 0.010*"drug" +
0.010*"treatment" + 0.007*"student" + 0.007*"delivery" + 0.006*"wastewater" +
0.006*"cellular" + 0.006*"development"'),
 (38,
  '0.028*"building" + 0.023*"structural" + 0.020*"structure" +
0.018*"simulation" + 0.017*"membrane" + 0.016*"computational" + 0.014*"design" +
0.012*"material" + 0.009*"modeling" + 0.009*"concrete"'),
 (39,
```

```
  '0.018*"object" + 0.014*"ring" + 0.013*"study" + 0.012*"structure" +
0.012*"geometry" + 0.012*"property" + 0.011*"cartilage" + 0.011*"manifold" +
0.010*"complex" + 0.008*"geometric"'),
 (40,
  '0.022*"gene" + 0.021*"specie" + 0.014*"genetic" + 0.012*"evolution" +
0.012*"evolutionary" + 0.009*"study" + 0.009*"organism" + 0.007*"population" +
0.007*"analysis" + 0.007*"genome"'),
 (41,
  '0.030*"model" + 0.014*"method" + 0.013*"problem" + 0.012*"system" +
0.008*"computational" + 0.008*"new" + 0.007*"algorithm" + 0.007*"analysis" +
0.007*"time" + 0.007*"statistical"'),
 (42,
  '0.043*"data" + 0.013*"software" + 0.011*"system" + 0.009*"application" +
0.009*"new" + 0.008*"computing" + 0.008*"tool" + 0.007*"analysis" +
0.007*"algorithm" + 0.007*"computer"'),
 (43,
  '0.067*"power" + 0.018*"energy" + 0.017*"sensor" + 0.016*"wind" +
0.015*"system" + 0.010*"electric" + 0.009*"monitoring" + 0.009*"combustion" +
0.008*"sensing" + 0.007*"control"'),
 (44,
  '0.013*"science" + 0.012*"community" + 0.008*"national" + 0.008*"data" +
0.007*"scientific" + 0.005*"resource" + 0.005*"magma" + 0.005*"tribal" +
0.005*"new" + 0.005*"researcher"'),
 (45,
  '0.042*"solar" + 0.035*"energy" + 0.032*"polymer" + 0.027*"material" +
0.027*"cell" + 0.019*"efficiency" + 0.016*"tandem" + 0.011*"high" +
0.011*"conversion" + 0.009*"alloy"'),
 (46,
  '0.034*"protein" + 0.021*"molecular" + 0.014*"molecule" + 0.010*"function" +
0.009*"cell" + 0.008*"dna" + 0.007*"interaction" + 0.006*"student" +
0.006*"mechanism" + 0.006*"cellular"'),
 (47,
  '0.057*"quantum" + 0.027*"system" + 0.019*"physic" + 0.018*"state" +
0.013*"integrable" + 0.012*"electron" + 0.011*"new" + 0.010*"matter" +
0.010*"topological" + 0.010*"theory"'),
 (48,
  '0.030*"system" + 0.015*"control" + 0.011*"design" + 0.008*"robot" +
0.006*"sensor" + 0.006*"vehicle" + 0.005*"new" + 0.005*"technology" +
0.005*"performance" + 0.005*"robotics"'),
 (49,
  '0.036*"chemical" + 0.031*"chemistry" + 0.026*"reaction" + 0.014*"catalyst" +
0.013*"synthesis" + 0.013*"organic" + 0.013*"molecule" + 0.013*"professor" +
0.012*"new" + 0.012*"metal"')]
```

A more useful visualization is provided by the python LDA visualization library, pyLDAvis.

Before executing the next code fragment you might need to install pyLDAvis:

```
>> pip install (--user) pyLDAvis
```

[23]:
```python
import pyLDAvis.gensim as gensimvis
import pyLDAvis

vis_data = gensimvis.prepare(ldag, corpus_bow, D)
pyLDAvis.display(vis_data)
```

```
/Users/jcid/anaconda/lib/python3.7/site-packages/pyLDAvis/_prepare.py:257:
FutureWarning: Sorting because non-concatenation axis is not aligned. A future
version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

[23]:
```
<IPython.core.display.HTML object>
```

### 1.3.3 3.3. Gensim utility functions

In addition to visualization purposes, topic models are useful to obtain a semantic representation of documents that can later be used with some other purpose:

- In document classification problems
- In content-based recommendations systems

Essentially, the idea is that the topic model provides a (semantic) vector representation of documents, and use probability divergences to measure document similarity. The following functions of the `LdaModel` class will be useful in this context:

- `get_topic_terms(topic_id)`: Gets vector of the probability distribution among words for the indicated topic
- `get_document_topics(bow_vector)`: Gets (sparse) vector with the probability distribution among topics for the provided document

**Exercise 4**: Show the probability distribution over words for topic 0.

[24]:
```python
# <SOL>
print(ldag.get_topic_terms(topicid=0))
# </SOL>
```

```
[(836, 0.104921356), (291, 0.012429194), (2679, 0.010999934), (490,
0.010416916), (639, 0.010362864), (2113, 0.009578622), (15, 0.00955166), (207,
0.008054779), (475, 0.00730686), (598, 0.0066519473)]
```

**Exercise 5**: Show the probability distribution over topics for document 0.

[25]:
```
print(ldag.get_document_topics(corpus_bow[0]))
```

```
[(12, 0.032040942), (13, 0.755762), (24, 0.121610224), (42, 0.08544395)]
```

An alternative to the use of the `get_document_topics()` function is to directly transform a dataset using the `ldag` object as follows. You can apply this transformation to several documents at once, but then the result is an iterator from which you can build the corresponding list if necessary

[26]:
```
print(ldag[corpus_bow[0]])

print('When applied to a dataset it will provide an iterator')
print(ldag[corpus_bow[:3]])

print('We can rebuild the list from the iterator with a one liner')
print([el for el in ldag[corpus_bow[:3]]])
```

```
[(12, 0.032082487), (13, 0.75590795), (24, 0.12179432), (42, 0.085072316)]
When applied to a dataset it will provide an iterator
<gensim.interfaces.TransformedCorpus object at 0x1a4efd6f28>
We can rebuild the list from the iterator with a one liner
[[(12, 0.03200142), (13, 0.7557476), (24, 0.12168423), (42, 0.08542377)], [(2,
0.012327205), (8, 0.12496957), (9, 0.32233655), (13, 0.022063384), (18,
0.047230493), (21, 0.16798146), (28, 0.015669137), (37, 0.02133713), (43,
0.037457075), (48, 0.20906661)], [(15, 0.02648347), (20, 0.13932611), (24,
0.028354654), (32, 0.0242126), (36, 0.20196049), (42, 0.57084066)]]
```

Finally, Gensim provides some useful functions to convert between formats, and to simplify interaction with numpy and scipy. The following code fragment converts a corpus in sparse format to a full numpy matrix

[27]:
```
reduced_corpus = [el for el in ldag[corpus_bow[:3]]]
reduced_corpus = gensim.matutils.corpus2dense(reduced_corpus, num_topics).T
print(reduced_corpus)
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.03203269 0.75585985 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.12176557 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.08519898 0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.0123288  0.         0.         0.
  0.         0.         0.12501329 0.32237455 0.         0.
  0.         0.02206147 0.         0.         0.         0.
  0.04716912 0.         0.         0.1679898  0.         0.
  0.         0.         0.         0.         0.01567039 0.
  0.         0.         0.         0.         0.         0.
```

```
 0.         0.02134164 0.         0.         0.         0.
 0.         0.0373646  0.         0.         0.         0.
 0.20912564 0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.02649002 0.         0.
 0.         0.         0.13928716 0.         0.         0.
 0.02836796 0.         0.         0.         0.         0.
 0.         0.         0.02424332 0.         0.         0.
 0.20196319 0.         0.         0.         0.         0.
 0.5708263  0.         0.         0.         0.         0.
 0.         0.        ]]
```

**Exercise 6**: Build a function that returns the most relevant projects for a given topic

[28]:
```
def most_relevant_projects(ldag, topicid, corpus_bow, nprojects=10):
    """This function returns the most relevant projects in corpus_bow

    : ldag: The trained topic model object provided by gensim
    : topicid: The topic for which we want to find the most relevant documents
    : corpus_bow: The BoW representation of documents in Gensim format
    : nprojects: Number of most relevant projects to identify

    : Returns: A list with the identifiers of the most relevant projects
    """

    print('Computing most relevant projects for Topic', topicid)
    print('Topic composition is:')
    print(ldag.show_topic(topicid))

    #<SOL>
    document_topic = [el for el in ldag[corpus_bow]]
    document_topic = gensim.matutils.corpus2dense(document_topic, ldag.
 ↪num_topics).T
    return np.argsort(document_topic[:,topicid])[::-1][:nprojects].tolist()
    #</SOL>

#To test the function we will find the most relevant projects for a subset of␣
 ↪the NSF dataset
project_id = most_relevant_projects(ldag, 17, corpus_bow[:10000])

#Print titles of selected projects
for idproject in project_id:
    print(NSF_data[idproject]['title'])
```

```
Computing most relevant projects for Topic 17
Topic composition is:
[('student', 0.0416345), ('program', 0.025278637), ('university', 0.020965578),
```

```
('support', 0.018697618), ('science', 0.018104237), ('graduate', 0.012483657),
('faculty', 0.011421016), ('institution', 0.010093892), ('community',
0.009353421), ('opportunity', 0.009143553)]
Student Travel Support for ACM HPDC 2015
Doctoral Mentoring Consortium at the 24th International Joint Conference on
Artificial Intelligence (IJCAI 2015)
Student and Mentor-Student Travel Scholarship Program
UNS Proposal for conference support for the 2015 InterPACK/ICNMM conference July
6-9 in San Francisco, CA
NSF CISE CAREER Proposal Writing Workshop 2015
WORKSHOP: Computer Supported Cooperative Work 2015 Doctoral Research Colloquium
Collaborative Research: Access Network: Supporting Retention and Representation
in Physics Through an Alliance of Campus-Based Diversity Programs
Collaborative Research: The Access Network: Supporting Retention and
Representation in Physics Through an Alliance of Campus-Based Diversity Programs
Collaborative Research: The Access Network: Supporting Retention and
Representation in Physics through an Alliance of Campus-Based Diversity Programs
Student Travel Support for ACM HotMobile 2015
```

**Exercise 7**: Build a function that computes the semantic distance between two documents. For this, you can use the functions (or code fragments) provided in the library `dist_utils.py`.

[29]:
```python
def pairwase_dist(doc1, doc2):
    """This function returns the Jensen-Shannon
    distance between the corresponding vectors of the documents

    : doc1: Semantic representation for the doc1 (a vector of length ntopics)
    : doc2: Semantic representation for the doc2 (a vector of length ntopics)

    : Returns: The JS distance between doc1 and doc2 (a number)
    """
    #<SOL>
    #</SOL>
```

[ ]:

**Exercise 8**: Explore the influence of the concentration parameters, *alpha* and *eta*. In particular observe how do topic and document distributions change as these parameters increase.

[ ]:

** Exercise 9**: Note that we have not used the terms in the article titles, though the can be expected to contain relevant words for the topic modeling. Include the title words in the analyisis. In order to give them a special relevante, insert them in the corpus several times, so as to make their words more significant.

[ ]:

## 1.4 4. Saving data.

The following function creates the Node CSV file that could be usefult for further processing. In particular, the output files could be used to visualize the document corpus as a graph using a visualization software like Gephi.

[30]:
```
#print(NSF_data[0].keys())
#print(NSF_data[0]['institution'])

def strNone(str_to_convert):
    if str_to_convert is None:
        return ''
    else:
        return str_to_convert

with open('NSF_nodes.csv','w') as fout:
    fout.write('Id;Title;Year;Budget;UnivName;UnivZIP;State\n')
    for project in NSF_data:
        fout.write(project['project_code']+';'+project['title']+';')
        fout.write(project['year']+';'+str(project['budget'])+';')
        fout.write(project['institution'][0]+';')
        fout.write(strNone(project['institution'][1])+';')
        fout.write(strNone(project['institution'][2])+'\n')
```

[ ]:
```

```