TM2_Topic_Modeling_professor

November 25, 2019

1 Topic Modelling

Author: Jesús Cid Sueiro

Date: 2016/11/27

In this notebook we will explore some tools for text analysis in python. To do so, first we will import the requested python libraries.

[1]:
```
%matplotlib inline

# Required imports
from wikitools import wiki
from wikitools import category

import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

import gensim

import numpy as np
import lda
import lda.datasets

from time import time
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

import matplotlib.pyplot as plt
import pylab

from test_helper import Test
```

1.1 1. Corpus acquisition.

In this notebook we will explore some tools for text processing and analysis and two topic modeling algorithms available from Python toolboxes.

To do so, we will explore and analyze collections of Wikipedia articles from a given category, using `wikitools`, that makes easy the capture of content from wikimedia sites.

(*As a side note, there are many other available text collections to test topic modelling algorithm. In particular, the NLTK library has many examples, that can explore them using the* ***`nltk.download()`*** *tool.*

```
import nltk
nltk.download()
```

for instance, you can take the gutemberg dataset

```
Mycorpus = nltk.corpus.gutenberg
text_name = Mycorpus.fileids()[0]
raw = Mycorpus.raw(text_name)
Words = Mycorpus.words(text_name)
```

Also, tools like Gensim or Sci-kit learn include text databases to work with).

In order to use Wikipedia data, we will select a single category of articles:

[2]:
```
site = wiki.Wiki("https://en.wikipedia.org/w/api.php")
# Select a category with a reasonable number of articles (>100)
# cat = "Economics"
cat = "Pseudoscience"
print cat
```

```
Pseudoscience
```

You can try with any other categories. Take into account that the behavior of topic modelling algorithms may depend on the amount of documents available for the analysis. Select a category with at least 100 articles. You can browse the wikipedia category tree here, https://en.wikipedia.org/wiki/Category:Contents, for instance.

We start downloading the text collection.

[3]:
```
# Loading category data. This may take a while
print "Loading category data. This may take a while..."
cat_data = category.Category(site, cat)

corpus_titles = []
corpus_text = []

for n, page in enumerate(cat_data.getAllMembersGen()):
    print "\r Loading article {0}".format(n + 1),
    corpus_titles.append(page.title)
    corpus_text.append(page.getWikiText())
```

```
n_art = len(corpus_titles)
print "\nLoaded " + str(n_art) + " articles from category " + cat
```

```
Loading category data. This may take a while…
 Loading article 359
Loaded 359 articles from category Pseudoscience
```

Now, we have stored the whole text collection in two lists:

- `corpus_titles`, which contains the titles of the selected articles
- `corpus_text`, with the text content of the selected wikipedia articles

You can browse the content of the wikipedia articles to get some intuition about the kind of documents that will be processed.

[4]:
```
# n = 5
# print corpus_titles[n]
# print corpus_text[n]
```

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, filtering and cleaning
2. Homogeneization (stemming or lemmatization)
3. Vectorization

1.2.1 2.1. Tokenization, filtering and cleaning.

The first steps consists on the following:

1. Tokenization: convert text string into lists of tokens.
2. Filtering:

- Removing capitalization: capital alphabetic characters will be transformed to their corresponding lowercase characters.
- Removing non alphanumeric tokens (e.g. punktuation signs)

3. Cleaning: Removing stopwords, i.e., those words that are very common in language and do not carry out useful semantic content (articles, pronouns, etc).

To do so, we will need some packages from the Natural Language Toolkit.

[5]:
```
# You can comment this if the package is already available.
# Select option "d) Download", and identifier "punkt"
# Select option "d) Download", and identifier "stopwords"
# nltk.download()
```

[6]:
```
stopwords_en = stopwords.words('english')
corpus_clean = []

for n, art in enumerate(corpus_text):
    print "\rProcessing article {0} out of {1}".format(n + 1, n_art),
    # This is to make sure that all characters have the appropriate encoding.
    art = art.decode('utf-8')

    # Tokenize each text entry.
    # scode: tokens = <FILL IN>
    token_list = word_tokenize(art)

    # Convert all tokens in token_list to lowercase, remove non alfanumeric␣
 ↪tokens and stem.
    # Store the result in a new token list, clean_tokens.
    # scode: filtered_tokens = <FILL IN>
    filtered_tokens = [token.lower() for token in token_list if token.isalnum()]

    # Remove all tokens in the stopwords list and append the result to␣
 ↪corpus_clean
    # scode: clean_tokens = <FILL IN>
    clean_tokens = [token for token in filtered_tokens if token not in␣
 ↪stopwords_en]

    # scode: <FILL IN>
    corpus_clean.append(clean_tokens)
print "\nLet's check the first tokens from document 0 after processing:"
print corpus_clean[0][0:30]
```

```
Processing article 359 out of 359
Let's check the first tokens from document 0 after processing:
[u'see', u'topics', u'characterized', u'pseudoscience', u'pseudoscience',
u'consists', u'claims', u'belief', u'practices', u'presented', u'plausible',
u'scientifically', u'justifiable', u'scientific', u'method', u'tag',
u'pretended', u'spurious', u'science', u'collection', u'related', u'beliefs',
u'world', u'mistakenly', u'regarded', u'based', u'scientific', u'method',
u'status', u'scientific']
```

[7]:
```
Test.assertTrue(len(corpus_clean) == n_art, 'List corpus_clean does not contain␣
 ↪the expected number of articles')
Test.assertTrue(len([c for c in corpus_clean[0] if c in stopwords_en])==0,␣
 ↪'Stopwords have not been removed')
```

```
1 test passed.
1 test passed.
```

1.2.2 2.2. Stemming vs Lemmatization

At this point, we can choose between applying a simple stemming or ussing lemmatization. We will try both to test their differences.

Task: Apply the `.stem()` method, from the stemmer object created in the first line, to `corpus_filtered`.

[8]:
```
# Select stemmer.
stemmer = nltk.stem.SnowballStemmer('english')
corpus_stemmed = []

for n, token_list in enumerate(corpus_clean):
    print "\rStemming article {0} out of {1}".format(n + 1, n_art),

    # Convert all tokens in token_list to lowercase, remove non alfanumeric␣
 ↪tokens and stem.
    # Store the result in a new token list, clean_tokens.
    # scode: stemmed_tokens = <FILL IN>
    stemmed_tokens = [stemmer.stem(token) for token in token_list]

    # Add art to the stemmed corpus
    # scode: <FILL IN>
    corpus_stemmed.append(stemmed_tokens)

print "\nLet's check the first tokens from document 0 after stemming:"
print corpus_stemmed[0][0:30]
```

```
Stemming article 359 out of 359
Let's check the first tokens from document 0 after stemming:
[u'see', u'topic', u'character', u'pseudosci', u'pseudosci', u'consist',
u'claim', u'belief', u'practic', u'present', u'plausibl', u'scientif',
u'justifi', u'scientif', u'method', u'tag', u'pretend', u'spurious', u'scienc',
u'collect', u'relat', u'belief', u'world', u'mistaken', u'regard', u'base',
u'scientif', u'method', u'status', u'scientif']
```

[9]:
```
Test.assertTrue((len([c for c in corpus_stemmed[0] if c!=stemmer.stem(c)]) < 0.
 ↪1*len(corpus_stemmed[0])),
                'It seems that stemming has not been applied properly')
```

```
1 test passed.
```

Alternatively, we can apply lemmatization. For english texts, we can use the lemmatizer from NLTK, which is based on WordNet. If you have not used wordnet before, you will likely need to download it from nltk

[10]:
```
# You can comment this if the package is already available.
# Select option "d) Download", and identifier "wordnet"
# nltk.download()
```

Task: Apply the `.lemmatize()` method, from the WordNetLemmatizer object created in the first line, to `corpus_filtered`.

[11]:
```
wnl = WordNetLemmatizer()

# Select stemmer.
corpus_lemmat = []

for n, token_list in enumerate(corpus_clean):
    print "\rLemmatizing article {0} out of {1}".format(n + 1, n_art),

    # scode: lemmat_tokens = <FILL IN>
    lemmat_tokens = [wnl.lemmatize(token) for token in token_list]

    # Add art to the stemmed corpus
    # scode: <FILL IN>
    corpus_lemmat.append(lemmat_tokens)

print "\nLet's check the first tokens from document 0 after stemming:"
print corpus_lemmat[0][0:30]
```

```
Lemmatizing article 359 out of 359
Let's check the first tokens from document 0 after stemming:
[u'see', u'topic', u'characterized', u'pseudoscience', u'pseudoscience',
u'consists', u'claim', u'belief', u'practice', u'presented', u'plausible',
u'scientifically', u'justifiable', u'scientific', u'method', u'tag',
u'pretended', u'spurious', u'science', u'collection', u'related', u'belief',
u'world', u'mistakenly', u'regarded', u'based', u'scientific', u'method',
u'status', u'scientific']
```

One of the advantages of the lemmatizer method is that the result of lemmmatization is still a true word, which is more advisable for the presentation of text processing results and lemmatization.

However, without using contextual information, lemmatize() does not remove grammatical differences. This is the reason why ”is” or ”are” are preserved and not replaced by infinitive ”be”.

As an alternative, we can apply .lemmatize(word, pos), where ’pos’ is a string code specifying the part-of-speech (pos), i.e. the grammatical role of the words in its sentence. For instance, you can check the difference between `wnl.lemmatize('is')` and `wnl.lemmatize('is, pos='v')`.

1.2.3 2.3. Vectorization

Up to this point, we have transformed the raw text collection of articles in a list of articles, where each article is a collection of the word roots 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.

[12]:
```
# Create dictionary of tokens
D = gensim.corpora.Dictionary(corpus_clean)
n_tokens = len(D)

print "The dictionary contains {0} tokens".format(n_tokens)
print "First tokens in the dictionary: "
for n in range(10):
    print str(n) + ": " + D[n]
```

```
The dictionary contains 44712 tokens
First tokens in the dictionary:
0: systematic
1: magnetic
2: pejorative
3: four
4: consists
5: integrity
6: founder
7: navels
8: unanswered
9: presents
```

In the second step, let us create a numerical version of our corpus using the `doc2bow` method. In general, `D.doc2bow(token_list)` transform 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`.

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

[13]:
```
# Transform token lists into sparse vectors on the D-space
corpus_bow = [D.doc2bow(doc) for doc in corpus_clean]
```

[14]:
```
Test.assertTrue(len(corpus_bow)==n_art, 'corpus_bow has not the appropriate␣
↪size')
```

```
1 test passed.
```

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

[15]:
```
print "Original article (after cleaning): "
print corpus_clean[0][0:30]
print "Sparse vector representation (first 30 components):"
print corpus_bow[0][0:30]
print "The first component, {0} from document 0, states that token 0 ({1})␣
 ↪appears {2} times".format(
    corpus_bow[0][0], D[0], corpus_bow[0][0][1])
```

```
Original article (after cleaning):
[u'see', u'topics', u'characterized', u'pseudoscience', u'pseudoscience',
u'consists', u'claims', u'belief', u'practices', u'presented', u'plausible',
u'scientifically', u'justifiable', u'scientific', u'method', u'tag',
u'pretended', u'spurious', u'science', u'collection', u'related', u'beliefs',
u'world', u'mistakenly', u'regarded', u'based', u'scientific', u'method',
u'status', u'scientific']
Sparse vector representation (first 30 components):
[(0, 2), (1, 1), (2, 2), (3, 1), (4, 2), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1),
(10, 1), (11, 3), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1),
(19, 1), (20, 1), (21, 1), (22, 1), (23, 4), (24, 1), (25, 1), (26, 2), (27,
147), (28, 1), (29, 1)]
The first component, (0, 2) from document 0, states that token 0 (systematic)
appears 2 times
```

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.

Note that, at this point, we have built a Dictionary containing

[16]:
```
print "{0} tokens".format(len(D))
```

```
44712 tokens
```

and a bow representation of a corpus with

[17]:
```
print "{0} Wikipedia articles".format(len(corpus_bow))
```

```
359 Wikipedia articles
```

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

[18]:
```
# 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.
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

[19]:
```
print D[ids_sorted[0]]
```

```
ref
```

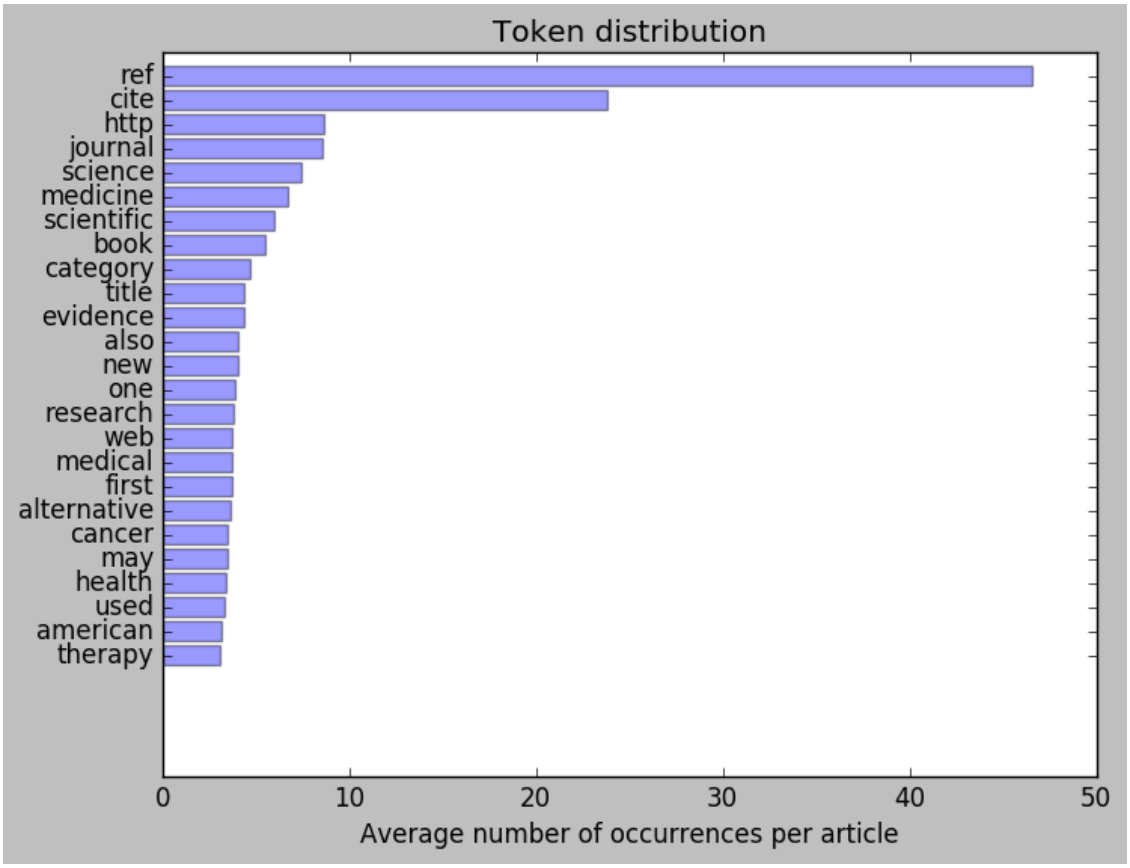
which appears

[20]:
```
print "{0} times in the whole corpus".format(tf_sorted[0])
```

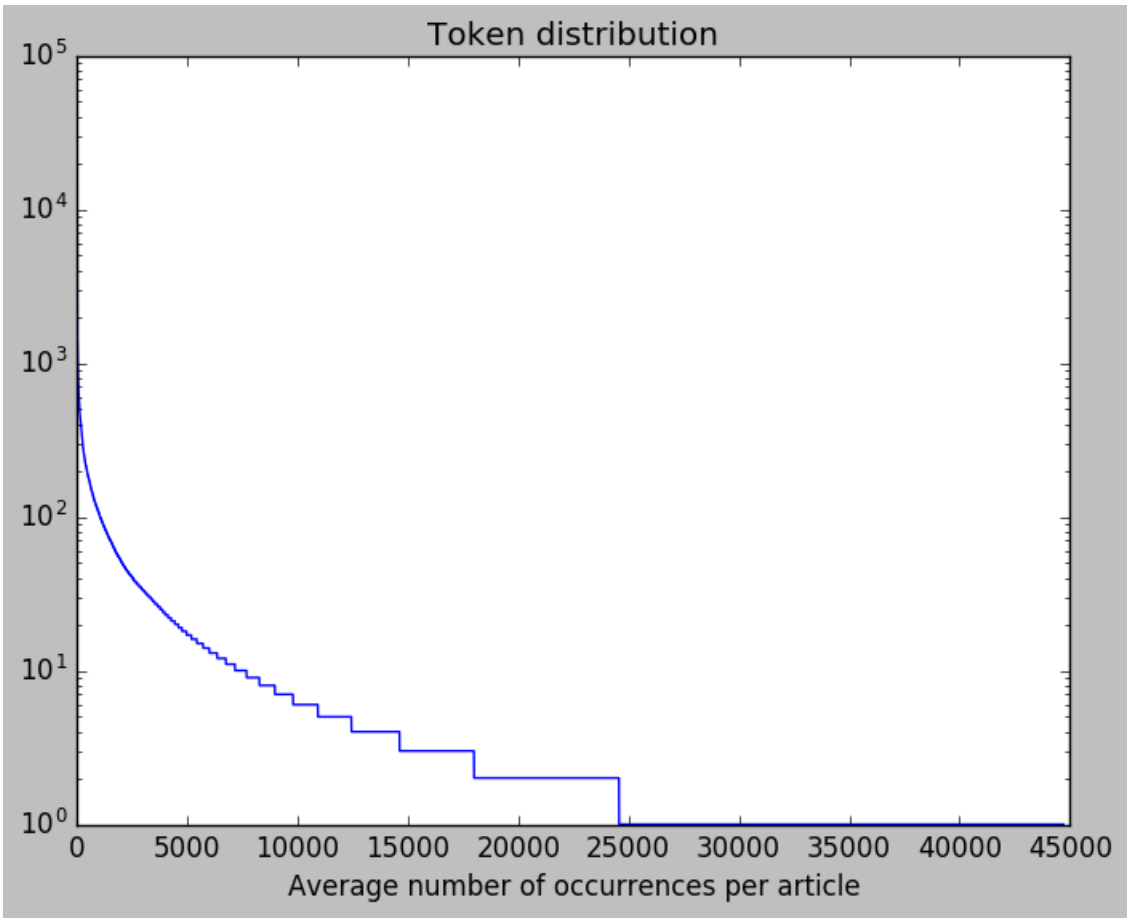
```
16713.0 times in the whole corpus
```

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

[21]:
```
# SORTED TOKEN FREQUENCIES (II):
plt.rcdefaults()

# Example 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.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()
```



[22]:
```
# SORTED TOKEN FREQUENCIES:

# Example data
plt.semilogy(tf_sorted)
plt.xlabel('Average number of occurrences per article')
plt.title('Token distribution')
plt.show()
```



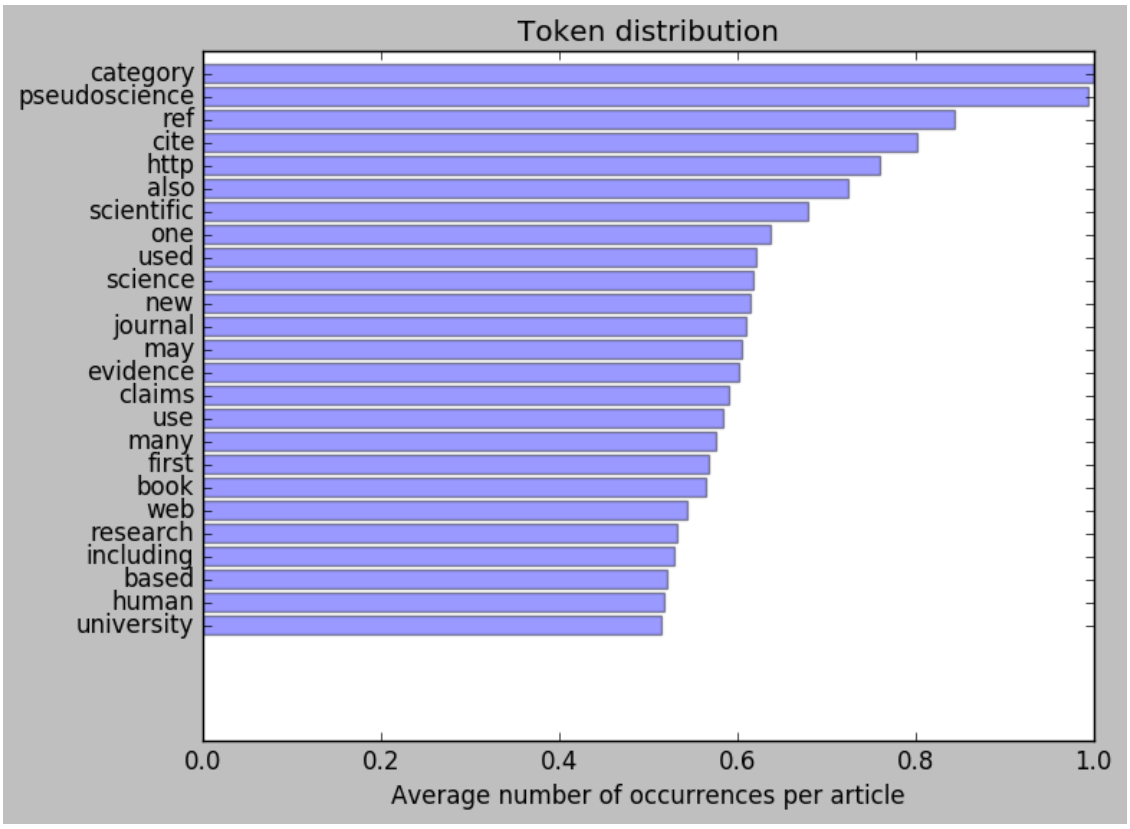
** Exercise**: There are usually many tokens that appear with very low frequency in the corpus. Count the number of tokens appearing only once, and what is the proportion of them in the token list.

[24]:
```
# scode: <WRITE YOUR CODE HERE>
# Example data
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 20151 cold tokens, which represent 45.0684380032% of the total number
of tokens in the dictionary
```

** Exercise**: Represent graphically those 20 tokens that appear in the highest number of articles. Note that you can use the code above (headed by `# SORTED TOKEN FREQUENCIES`) with a very minor modification.

[25]:
```
# scode: <WRITE YOUR CODE HERE>

# SORTED TOKEN FREQUENCIES (I):
# Count the number of occurrences of each token.
token_count2 = np.zeros(n_tokens)
for x in corpus_bow_flat:
    token_count2[x[0]] += (x[1]>0)

# Sort by decreasing number of occurences
ids_sorted2 = np.argsort(- token_count2)
tf_sorted2 = token_count2[ids_sorted2]

# SORTED TOKEN FREQUENCIES (II):
# Example data
n_bins = 25
hot_tokens2 = [D[i] for i in ids_sorted2[n_bins-1::-1]]
y_pos2 = np.arange(len(hot_tokens2))
z2 = tf_sorted2[n_bins-1::-1]/n_art

plt.barh(y_pos2, z2, align='center', alpha=0.4)
plt.yticks(y_pos2, hot_tokens2)
plt.xlabel('Average number of occurrences per article')
plt.title('Token distribution')
plt.show()
```



1.3 3. Semantic Analysis

The dictionary `D` and the Bag of Words in `corpus_bow` are the key inputs to the topic model algorithms. In this section we will explore two algorithms:

1. Latent Semantic Indexing (LSI)
2. Latent Dirichlet Allocation (LDA)

The topic model algorithms in `gensim` assume that input documents are parameterized using the tf-idf model. This can be done using

[26]:
```
tfidf = gensim.models.TfidfModel(corpus_bow)
```

From now on, tfidf can be used to convert any vector from the old representation (bow integer counts) to the new one (TfIdf real-valued weights):

[28]:
```
doc_bow = [(0, 1), (1, 1)]
tfidf[doc_bow]
```

[28]:
```
[(0, 0.6410406170794578), (1, 0.7675069558345305)]
```

Or to apply a transformation to a whole corpus

[29]:
```
corpus_tfidf = tfidf[corpus_bow]
print corpus_tfidf[0][0:5]
```

```
[(0, 0.01008360066280908), (1, 0.006036461218185464), (2, 0.020970503696900015),
(3, 0.004131709927129921), (4, 0.014121006630463492)]
```

1.3.1 3.1. Latent Semantic Indexing (LSI)

Now we are ready to apply a topic modeling algorithm. Latent Semantic Indexing is provided by `LsiModel`.

Task: Generate a LSI model with 5 topics for `corpus_tfidf` and dictionary `D`. You can check de sintaxis for gensim.models.LsiModel.

[30]:
```
# Initialize an LSI transformation
n_topics = 5

# scode: lsi = <FILL IN>
lsi = gensim.models.LsiModel(corpus_tfidf, id2word=D, num_topics=n_topics)
```

From LSI, we can check both the topic-tokens matrix and the document-topics matrix.

Now we can check the topics generated by LSI. An intuitive visualization is provided by the `show_topics` method.

[31]:
```
lsi.show_topics(num_topics=-1, num_words=10, log=False, formatted=True)
```

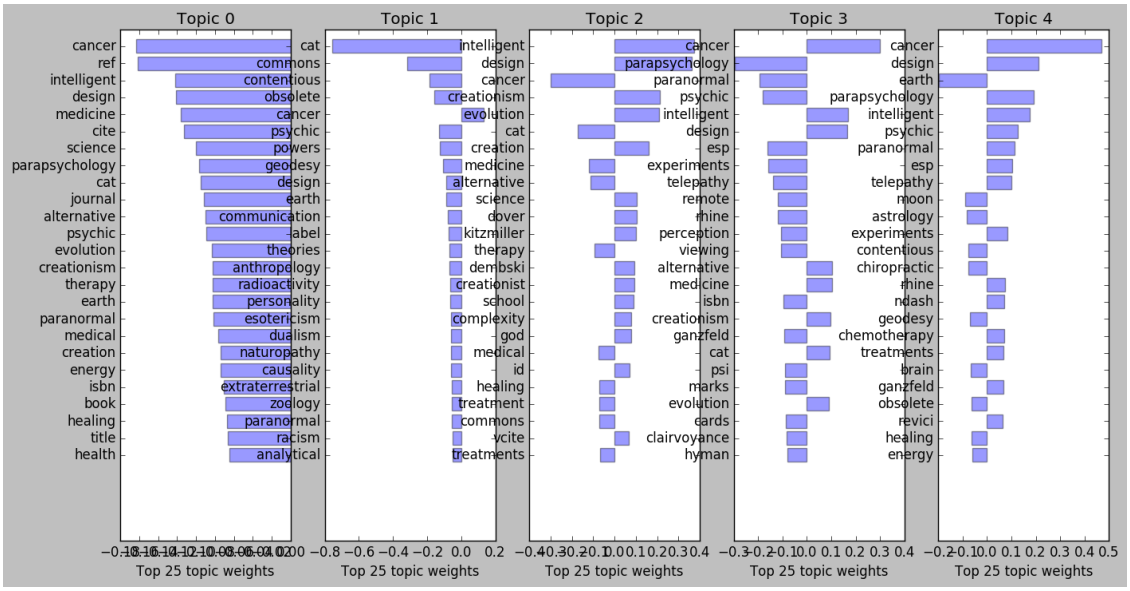
[31]:
```
[(0,
  u'-0.163*"cancer" + -0.162*"ref" + -0.122*"intelligent" + -0.122*"design" +
-0.116*"medicine" + -0.112*"cite" + -0.100*"science" + -0.097*"parapsychology" +
-0.095*"cat" + -0.092*"journal"'),
 (1,
  u'-0.758*"cat" + -0.315*"commons" + -0.186*"contentious" + -0.159*"obsolete" +
0.133*"cancer" + -0.128*"psychic" + -0.124*"powers" + -0.105*"geodesy" +
-0.090*"design" + -0.087*"earth"'),
 (2,
  u'0.373*"intelligent" + 0.361*"design" + -0.301*"cancer" + 0.212*"creationism"
+ 0.210*"evolution" + -0.174*"cat" + 0.161*"creation" + -0.121*"medicine" +
-0.112*"alternative" + 0.105*"science"'),
 (3,
  u'0.301*"cancer" + -0.295*"parapsychology" + -0.195*"paranormal" +
-0.183*"psychic" + 0.168*"intelligent" + 0.164*"design" + -0.162*"esp" +
-0.158*"experiments" + -0.139*"telepathy" + -0.120*"remote"'),
 (4,
  u'0.469*"cancer" + 0.211*"design" + -0.197*"earth" + 0.192*"parapsychology" +
0.176*"intelligent" + 0.126*"psychic" + 0.112*"paranormal" + 0.104*"esp" +
0.100*"telepathy" + -0.090*"moon"')]
```

However, a more useful representation of topics is as a list of tuples `(token, value)`. This is provided by the `show_topic` method.

Task: Represent the columns of the topic-token matrix as a series of bar diagrams (one per topic) with the top 25 tokens of each topic.

[50]:
```
# SORTED TOKEN FREQUENCIES (II):
plt.rcdefaults()

n_bins = 25

# Example data
y_pos = range(n_bins-1, -1, -1)

pylab.rcParams['figure.figsize'] = 16, 8  # Set figure size
for i in range(n_topics):

    ### Plot top 25 tokens for topic i
    # Read i-thtopic
    # scode: <FILL IN>
    topic_i = lsi.show_topic(i, topn=n_bins)
    tokens = [t[0] for t in topic_i]
    weights = [t[1] for t in topic_i]

    # Plot
```

```
    # scode: <FILL IN>
    plt.subplot(1, n_topics, i+1)
    plt.barh(y_pos, weights, align='center', alpha=0.4)
    plt.yticks(y_pos, tokens)
    plt.xlabel('Top {0} topic weights'.format(n_bins))
    plt.title('Topic {0}'.format(i))

plt.show()
```



LSI approximates any document as a linear combination of the topic vectors. We can compute the topic weights for any input corpus entered as input to the `lsi` model.

[33]:
```
# On real corpora, target dimensionality of
# 200-500 is recommended as a "golden standard"
# Create a double wrapper over the original
# corpus bow  tfidf  fold-in-lsi
corpus_lsi = lsi[corpus_tfidf]
print corpus_lsi[0]
```

```
[(0, -0.38313510265426526), (1, 0.02073947483234645), (2, 0.079670796251513409),
(3, -0.05602098769399963), (4, -0.10483294234170717)]
```

Task: Find the document with the largest positive weight for topic 0. Compare the document and the topic.

[34]:
```
# Extract weights from corpus_lsi
# scode weight0 = <FILL IN>
weight0 = [doc[0][1] if doc != [] else -np.inf for doc in corpus_lsi]
```

```
# Locate the maximum positive weight
nmax = np.argmax(weight0)
print nmax
print weight0[nmax]
print corpus_lsi[nmax]

# Get topic 0
# scode: topic_0 = <FILL IN>
topic_0 = lsi.show_topic(0, topn=n_bins)

# Compute a list of tuples (token, wordcount) for all tokens in topic_0, where␣
 ↪wordcount is the number of
# occurences of the token in the article.
# scode: token_counts = <FILL IN>
token_counts = [(t[0], corpus_clean[nmax].count(t[0])) for t in topic_0]

print "Topic 0 is:"
print topic_0
print "Token counts:"
print token_counts
```

```
324
9.24011035586e-05
[(0, 9.2401103558608302e-05), (1, -0.0008911662871724323), (2,
-0.0021801105038539016), (3, -0.0034109765793003555), (4,
0.00013243990952392163)]
Topic 0 is:
[(u'cancer', -0.16299615550034205), (u'ref', -0.16204334394637257),
(u'intelligent', -0.12241764854455867), (u'design', -0.1216208934894871),
(u'medicine', -0.11622994292289084), (u'cite', -0.11243309400479952),
(u'science', -0.10001405178729662), (u'parapsychology', -0.096860101240488314),
(u'cat', -0.095048458814507464), (u'journal', -0.091667322442608778),
(u'alternative', -0.090272226407602976), (u'psychic', -0.089635371366803579),
(u'evolution', -0.083491820622816043), (u'creationism', -0.082764583623046756),
(u'therapy', -0.082484498373547627), (u'earth', -0.082186451355489659),
(u'paranormal', -0.081677342695715097), (u'medical', -0.076518104421351346),
(u'creation', -0.074284930674208299), (u'energy', -0.073838907070382695),
(u'isbn', -0.071090439750478837), (u'book', -0.068606782635001307), (u'healing',
-0.067689163612768902), (u'title', -0.066626375426302037), (u'health',
-0.064818733511321586)]
Token counts:
[(u'cancer', 0), (u'ref', 0), (u'intelligent', 0), (u'design', 0), (u'medicine',
0), (u'cite', 0), (u'science', 0), (u'parapsychology', 0), (u'cat', 0),
(u'journal', 0), (u'alternative', 0), (u'psychic', 0), (u'evolution', 0),
(u'creationism', 0), (u'therapy', 0), (u'earth', 0), (u'paranormal', 0),
(u'medical', 0), (u'creation', 0), (u'energy', 0), (u'isbn', 0), (u'book', 0),
(u'healing', 0), (u'title', 0), (u'health', 0)]
```

1.3.2 3.2. Latent Dirichlet Allocation (LDA)

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`

3.2.1. LDA using Gensim The use of the LDA module in `gensim` is similar to LSI. Furthermore, it assumes that a `tf-idf` parametrization is used as an input, which is not in complete agreement with the theoretical model, which assumes documents represented as vectors of token-counts.

To use LDA in gensim, we must first create a lda model object.

[35]:
```
ldag = gensim.models.ldamodel.LdaModel(
    corpus=corpus_tfidf, id2word=D, num_topics=10, update_every=1, passes=10)
```

[36]:
```
ldag.print_topics()
```

[36]:
```
[(0,
  u'0.000*cayce + 0.000*attunement + 0.000*cryptozoology + 0.000*orientation +
0.000*bracelets + 0.000*container + 0.000*dualism + 0.000*sexual + 0.000*vmsk +
0.000*fc'),
 (1,
  u'0.000*chiropractic + 0.000*ayurveda + 0.000*colon + 0.000*ganzfeld +
0.000*earthquake + 0.000*grapefruit + 0.000*ayurvedic + 0.000*chiropractors +
0.000*reich + 0.000*pseudophysics'),
 (2,
  u'0.000*neuropsychology + 0.000*attachment + 0.000*burzynski + 0.000*buteyko +
0.000*programming + 0.000*xango + 0.000*amen + 0.000*edgerly + 0.000*psychometry
+ 0.000*vril'),
 (3,
  u'0.000*cat + 0.000*amen + 0.000*instincts + 0.000*metabolic + 0.000*phiten +
0.000*naturopathic + 0.000*moerman + 0.000*crackpot + 0.000*bta +
0.000*wiktionarycat2'),
 (4,
  u'0.000*cat + 0.000*obsolete + 0.000*dowsing + 0.000*earth + 0.000*allergy +
0.000*geodesy + 0.000*anthropology + 0.000*racism + 0.000*personality +
0.000*commons'),
 (5,
  u'0.001*ref + 0.001*cancer + 0.001*cite + 0.001*medicine + 0.001*therapy +
0.001*journal + 0.001*science + 0.001*design + 0.001*alternative +
0.001*paranormal'),
 (6,
  u'0.000*cat + 0.000*hoxsey + 0.000*cupping + 0.000*thermography + 0.000*crop +
0.000*hungarian + 0.000*foil + 0.000*macrobiotic + 0.000*oil + 0.000*daffy'),
 (7,
  u'0.000*contentious + 0.000*bates + 0.000*narconon + 0.000*silva +
```

```
0.000*toftness + 0.000*drug + 0.000*penis + 0.000*kirlian + 0.000*irlen +
0.000*pulling'),
 (8,
  u'0.000*indigo + 0.000*ctca + 0.000*hysteria + 0.000*hydroelectricity +
0.000*tat + 0.000*nlp + 0.000*kozyrev + 0.000*naturopathy + 0.000*transfer +
0.000*orgone'),
 (9,
  u'0.000*hubbard + 0.000*scientology + 0.000*geology + 0.000*benveniste +
0.000*osteopathic + 0.000*homeopathy + 0.000*chiropractic + 0.000*healing +
0.000*bioresonance + 0.000*brainwashing')]
```

3.2.2. LDA using python lda library An alternative to gensim for LDA is the lda library from python. It requires a doc-frequency matrix as input

[37]:
```
# For testing LDA, you can use the reuters dataset
# X = lda.datasets.load_reuters()
# vocab = lda.datasets.load_reuters_vocab()
# titles = lda.datasets.load_reuters_titles()
X = np.int32(np.zeros((n_art, n_tokens)))
for n, art in enumerate(corpus_bow):
    for t in art:
        X[n, t[0]] = t[1]
print X.shape
print X.sum()

vocab = D.values()
titles = corpus_titles
```

```
(359, 44712)
580540
```

[38]:
```
# Default parameters:
# model = lda.LDA(n_topics, n_iter=2000, alpha=0.1, eta=0.01,␣
 ↪random_state=None, refresh=10)
model = lda.LDA(n_topics=10, n_iter=1500, random_state=1)
model.fit(X)  # model.fit_transform(X) is also available
topic_word = model.topic_word_  # model.components_ also works

# Show topics...
n_top_words = 8
for i, topic_dist in enumerate(topic_word):
    topic_words = np.array(vocab)[np.argsort(topic_dist)][:-(n_top_words+1):-1]
    print('Topic {}: {}'.format(i, ' '.join(topic_words)))
```

```
INFO:lda:n_documents: 359
INFO:lda:vocab_size: 44712
INFO:lda:n_words: 580540
```

```
INFO:lda:n_topics: 10
INFO:lda:n_iter: 1500
INFO:lda:<0> log likelihood: -6833326
INFO:lda:<10> log likelihood: -5683119
INFO:lda:<20> log likelihood: -5481293
INFO:lda:<30> log likelihood: -5422639
INFO:lda:<40> log likelihood: -5395267
INFO:lda:<50> log likelihood: -5377075
INFO:lda:<60> log likelihood: -5366871
INFO:lda:<70> log likelihood: -5357864
INFO:lda:<80> log likelihood: -5350104
INFO:lda:<90> log likelihood: -5345917
INFO:lda:<100> log likelihood: -5341577
INFO:lda:<110> log likelihood: -5337858
INFO:lda:<120> log likelihood: -5333809
INFO:lda:<130> log likelihood: -5329261
INFO:lda:<140> log likelihood: -5324101
INFO:lda:<150> log likelihood: -5321775
INFO:lda:<160> log likelihood: -5319380
INFO:lda:<170> log likelihood: -5317579
INFO:lda:<180> log likelihood: -5314674
INFO:lda:<190> log likelihood: -5312316
INFO:lda:<200> log likelihood: -5311771
INFO:lda:<210> log likelihood: -5309576
INFO:lda:<220> log likelihood: -5309359
INFO:lda:<230> log likelihood: -5309794
INFO:lda:<240> log likelihood: -5307956
INFO:lda:<250> log likelihood: -5308558
INFO:lda:<260> log likelihood: -5307566
INFO:lda:<270> log likelihood: -5306615
INFO:lda:<280> log likelihood: -5305479
INFO:lda:<290> log likelihood: -5303847
INFO:lda:<300> log likelihood: -5303349
INFO:lda:<310> log likelihood: -5303030
INFO:lda:<320> log likelihood: -5303149
INFO:lda:<330> log likelihood: -5302022
INFO:lda:<340> log likelihood: -5303872
INFO:lda:<350> log likelihood: -5302189
INFO:lda:<360> log likelihood: -5300637
INFO:lda:<370> log likelihood: -5302713
INFO:lda:<380> log likelihood: -5300825
INFO:lda:<390> log likelihood: -5300455
INFO:lda:<400> log likelihood: -5300165
INFO:lda:<410> log likelihood: -5298474
INFO:lda:<420> log likelihood: -5299804
INFO:lda:<430> log likelihood: -5298436
INFO:lda:<440> log likelihood: -5298757
INFO:lda:<450> log likelihood: -5297456
```

```
INFO:lda:<460> log likelihood: -5297524
INFO:lda:<470> log likelihood: -5297542
INFO:lda:<480> log likelihood: -5295272
INFO:lda:<490> log likelihood: -5296215
INFO:lda:<500> log likelihood: -5294200
INFO:lda:<510> log likelihood: -5294223
INFO:lda:<520> log likelihood: -5292818
INFO:lda:<530> log likelihood: -5291299
INFO:lda:<540> log likelihood: -5291001
INFO:lda:<550> log likelihood: -5290498
INFO:lda:<560> log likelihood: -5291105
INFO:lda:<570> log likelihood: -5289968
INFO:lda:<580> log likelihood: -5290483
INFO:lda:<590> log likelihood: -5288029
INFO:lda:<600> log likelihood: -5288956
INFO:lda:<610> log likelihood: -5286441
INFO:lda:<620> log likelihood: -5287445
INFO:lda:<630> log likelihood: -5286209
INFO:lda:<640> log likelihood: -5286109
INFO:lda:<650> log likelihood: -5286541
INFO:lda:<660> log likelihood: -5287055
INFO:lda:<670> log likelihood: -5285318
INFO:lda:<680> log likelihood: -5283650
INFO:lda:<690> log likelihood: -5284105
INFO:lda:<700> log likelihood: -5284120
INFO:lda:<710> log likelihood: -5284110
INFO:lda:<720> log likelihood: -5282596
INFO:lda:<730> log likelihood: -5281445
INFO:lda:<740> log likelihood: -5281757
INFO:lda:<750> log likelihood: -5281674
INFO:lda:<760> log likelihood: -5278759
INFO:lda:<770> log likelihood: -5279268
INFO:lda:<780> log likelihood: -5278208
INFO:lda:<790> log likelihood: -5276986
INFO:lda:<800> log likelihood: -5277647
INFO:lda:<810> log likelihood: -5275174
INFO:lda:<820> log likelihood: -5275514
INFO:lda:<830> log likelihood: -5275725
INFO:lda:<840> log likelihood: -5274348
INFO:lda:<850> log likelihood: -5274851
INFO:lda:<860> log likelihood: -5272546
INFO:lda:<870> log likelihood: -5273723
INFO:lda:<880> log likelihood: -5274381
INFO:lda:<890> log likelihood: -5273964
INFO:lda:<900> log likelihood: -5271192
INFO:lda:<910> log likelihood: -5271983
INFO:lda:<920> log likelihood: -5270546
INFO:lda:<930> log likelihood: -5272297
```

```
INFO:lda:<940> log likelihood: -5270638
INFO:lda:<950> log likelihood: -5271224
INFO:lda:<960> log likelihood: -5269268
INFO:lda:<970> log likelihood: -5270864
INFO:lda:<980> log likelihood: -5267685
INFO:lda:<990> log likelihood: -5267527
INFO:lda:<1000> log likelihood: -5265486
INFO:lda:<1010> log likelihood: -5266445
INFO:lda:<1020> log likelihood: -5265322
INFO:lda:<1030> log likelihood: -5264865
INFO:lda:<1040> log likelihood: -5264276
INFO:lda:<1050> log likelihood: -5265988
INFO:lda:<1060> log likelihood: -5263737
INFO:lda:<1070> log likelihood: -5263957
INFO:lda:<1080> log likelihood: -5262394
INFO:lda:<1090> log likelihood: -5263542
INFO:lda:<1100> log likelihood: -5261999
INFO:lda:<1110> log likelihood: -5261242
INFO:lda:<1120> log likelihood: -5260765
INFO:lda:<1130> log likelihood: -5260993
INFO:lda:<1140> log likelihood: -5260437
INFO:lda:<1150> log likelihood: -5260514
INFO:lda:<1160> log likelihood: -5261230
INFO:lda:<1170> log likelihood: -5259848
INFO:lda:<1180> log likelihood: -5260330
INFO:lda:<1190> log likelihood: -5259145
INFO:lda:<1200> log likelihood: -5258847
INFO:lda:<1210> log likelihood: -5258437
INFO:lda:<1220> log likelihood: -5257691
INFO:lda:<1230> log likelihood: -5258731
INFO:lda:<1240> log likelihood: -5257665
INFO:lda:<1250> log likelihood: -5255521
INFO:lda:<1260> log likelihood: -5256467
INFO:lda:<1270> log likelihood: -5254744
INFO:lda:<1280> log likelihood: -5257153
INFO:lda:<1290> log likelihood: -5255091
INFO:lda:<1300> log likelihood: -5255664
INFO:lda:<1310> log likelihood: -5254754
INFO:lda:<1320> log likelihood: -5254430
INFO:lda:<1330> log likelihood: -5255363
INFO:lda:<1340> log likelihood: -5254763
INFO:lda:<1350> log likelihood: -5254023
INFO:lda:<1360> log likelihood: -5253577
INFO:lda:<1370> log likelihood: -5253957
INFO:lda:<1380> log likelihood: -5253194
INFO:lda:<1390> log likelihood: -5253239
INFO:lda:<1400> log likelihood: -5252003
INFO:lda:<1410> log likelihood: -5250000
```

```
INFO:lda:<1420> log likelihood: -5252129
INFO:lda:<1430> log likelihood: -5253494
INFO:lda:<1440> log likelihood: -5251609
INFO:lda:<1450> log likelihood: -5252459
INFO:lda:<1460> log likelihood: -5252101
INFO:lda:<1470> log likelihood: -5251535
INFO:lda:<1480> log likelihood: -5250840
INFO:lda:<1490> log likelihood: -5249810
INFO:lda:<1499> log likelihood: -5249800
```

```
␣
↪---------------------------------------------------------------------------

UnicodeEncodeError                        Traceback (most recent call last)

<ipython-input-38-f7b842c2d7c1> in <module>()
  9 for i, topic_dist in enumerate(topic_word):
 10     topic_words = np.array(vocab)[np.argsort(topic_dist)][:
↪-(n_top_words+1):-1]
---> 11     print('Topic {}: {}'.format(i, ' '.join(topic_words)))


UnicodeEncodeError: 'ascii' codec can't encode character u'\xfc' in␣
↪position 4: ordinal not in range(128)
```

Document-topic distribution

[39]:
```
doc_topic = model.doc_topic_
for i in range(10):
    print("{} (top topic: {})".format(titles[i], doc_topic[i].argmax()))
```

```
Pseudoscience (top topic: 7)
List of topics characterized as pseudoscience (top topic: 0)
Acupuncture (top topic: 0)
All About Radiation (top topic: 4)
Alternative cancer treatments (top topic: 6)
Alternative medicine (top topic: 0)
Amen Clinics (top topic: 4)
Daniel Amen (top topic: 4)
American Indian creationism (top topic: 2)
Ancient Aliens (top topic: 8)
```

[40]:
```
# This is to apply the model to a new doc(s)
# doc_topic_test = model.transform(X_test)
# for title, topics in zip(titles_test, doc_topic_test):
#     print("{} (top topic: {})".format(title, topics.argmax()))
```

It allows incremental updates

3.2.2. LDA using Sci-kit Learn The input matrix to the `sklearn` implementation of LDA contains the token-counts for all documents in the corpus. `sklearn` contains a powerfull `CountVectorizer` method that can be used to construct the input matrix from the `corpus_bow`.

First, we will define an auxiliary function to print the top tokens in the model, that has been taken from the `sklearn` documentation.

[41]:
```
# Adapted from an example in sklearn site
# http://scikit-learn.org/dev/auto_examples/applications/
 ↪topics_extraction_with_nmf_lda.html

# You can try also with the dataset provided by sklearn in
# from sklearn.datasets import fetch_20newsgroups
# dataset = fetch_20newsgroups(shuffle=True, random_state=1,
#                              remove=('headers', 'footers', 'quotes'))

def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic #%d:" % topic_idx)
        print(" ".join([feature_names[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))
    print()
```

Now, we need a dataset to feed the Count_Vectorizer object, by joining all tokens in `corpus_clean` in a single string, using a space ' ' as separator.

[42]:
```
print("Loading dataset...")
# scode: data_samples = <FILL IN>
print "*".join(['Esto', 'es', 'un', 'ejemplo'])
data_samples = [" ".join(c) for c in corpus_clean]
print 'Document 0:'
print data_samples[0][0:200], '...'
```

```
Loading dataset…
Esto*es*un*ejemplo
Document 0:
see topics characterized pseudoscience pseudoscience consists claims belief
practices presented plausible scientifically justifiable scientific method tag
pretended spurious science collection related …
```

Now we are ready to compute the token counts.

[44]:
```
# Use tf (raw term count) features for LDA.
print("Extracting tf features for LDA...")
n_features = 1000
n_samples = 2000
tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2,
```

```
                                max_features=n_features,
                                stop_words='english')

t0 = time()
tf = tf_vectorizer.fit_transform(data_samples)
print("done in %0.3fs." % (time() - t0))
print tf[0][0][0]
```

```
Extracting tf features for LDA…
done in 1.079s.
  (0, 171)      2
  (0, 930)      2
  (0, 109)      1
  (0, 179)      1
  (0, 162)      1
  (0, 829)      1
  (0, 6)        1
  (0, 739)      2
  (0, 919)      1
  (0, 342)      1
  (0, 280)      1
  (0, 224)      2
  (0, 322)      1
  (0, 328)      1
  (0, 552)      1
  (0, 17)       1
  (0, 484)      1
  (0, 20)       1
  (0, 515)      1
  (0, 254)      1
  (0, 29)       1
  (0, 47)       7
  (0, 559)      1
  (0, 442)      3
  (0, 683)      1
  :     :
  (0, 63)       7
  (0, 928)      4
  (0, 879)      2
  (0, 596)      4
  (0, 14)       1
  (0, 329)      1
  (0, 823)      5
  (0, 304)      8
  (0, 343)      4
  (0, 646)      3
  (0, 871)      6
```

```
  (0, 130)      8
  (0, 992)      10
  (0, 138)      25
  (0, 783)      4
  (0, 815)      150
  (0, 590)      15
  (0, 817)      86
  (0, 818)      4
  (0, 711)      5
  (0, 706)      6
  (0, 137)      12
  (0, 208)      22
  (0, 189)      6
  (0, 932)      6
```

Now we can apply the LDA algorithm.

Task: Create an LDA object with the following parameters: n_topics=n_topics, max_iter=5, learning_method='online', learning_offset=50., random_state=0

[112]:
```
print("Fitting LDA models with tf features, "
      "n_samples=%d and n_features=%d..."
      % (n_samples, n_features))
# scode: lda = <FILL IN>
lda = LatentDirichletAllocation(n_topics=n_topics, max_iter=10,
                                learning_method='online', learning_offset=50.,␣
 ↪random_state=0)
#                                 doc_topic_prior= 1.0/n_topics,␣
 ↪topic_word_prior= 1.0/n_topics)
```

```
Fitting LDA models with tf features, n_samples=2000 and n_features=1000…
```

Task: Fit model `lda` with the token frequencies computed by `tf_vectorizer`.

[117]:
```
t0 = time()
corpus_lda = lda.fit_transform(tf)
print corpus_lda[10]/np.sum(corpus_lda[10])
print("done in %0.3fs." % (time() - t0))
print corpus_titles[10]
# print corpus_text[10]
```

```
[  6.69375429e-01   1.10134014e-04   3.30294191e-01   1.10429257e-04
   1.09816955e-04]
done in 5.244s.
Ancient astronauts
```

[118]:
```
print("\nTopics in LDA model:")
tf_feature_names = tf_vectorizer.get_feature_names()
print_top_words(lda, tf_feature_names, 20)
```

```
Topics in LDA model:
Topic #0:
ref science cite design intelligent scientific evolution book earth http
creation web climate school creationism episode journal theory list institute
Topic #1:
ref cite journal http title acupuncture medicine chiropractic review web
treatment evidence chinese url year therapy cancer publisher scientific health
Topic #2:
ref cite http isbn book new research journal scientific university science used
people press experiments paranormal human evidence psychic parapsychology
Topic #3:
cite ref brain geology hubbard flood oil book title science scientology
dianetics phrenology http journal publisher scientific memory year church
Topic #4:
ref cite medicine alternative medical cancer health journal http therapy
treatment narconon evidence drug web scientific complementary american use
homeopathy
()
```

[115]:
```
topics = lda.components_
topic_probs = [t/np.sum(t) for t in topics]
# print topic_probs[0]
print -np.sort(-topic_probs[0])
```

```
[  4.08611523e-02   3.15060270e-02   2.62876448e-02   1.94085969e-02
   1.64809217e-02   1.55608099e-02   1.26181052e-02   1.17472669e-02
   1.03639578e-02   1.00579445e-02   8.38655562e-03   8.02491562e-03
   7.58953604e-03   7.39464184e-03   7.25084945e-03   6.87650086e-03
   6.52819884e-03   6.34914078e-03   6.27131165e-03   6.07186440e-03
   5.96432601e-03   5.95392437e-03   5.86003403e-03   5.47174006e-03
   5.16174585e-03   5.07126308e-03   5.06874215e-03   5.06206787e-03
   4.95887778e-03   4.83196134e-03   4.62030461e-03   4.45538235e-03
   4.38389039e-03   4.24920725e-03   4.17859439e-03   4.16785535e-03
   4.10730487e-03   4.10036048e-03   3.91165593e-03   3.90035091e-03
   3.73960317e-03   3.72929700e-03   3.72347046e-03   3.71075647e-03
   3.67430524e-03   3.66930388e-03   3.62229243e-03   3.49713327e-03
   3.42890980e-03   3.37192303e-03   3.28387264e-03   3.22228163e-03
   3.13086732e-03   3.12277525e-03   3.07025806e-03   3.00875456e-03
   2.99783052e-03   2.96601070e-03   2.93749953e-03   2.90950743e-03
   2.89778910e-03   2.85653414e-03   2.85227373e-03   2.81775145e-03
   2.80817958e-03   2.76472528e-03   2.75743373e-03   2.74955820e-03
   2.68458246e-03   2.68039364e-03   2.65866366e-03   2.58510612e-03
   2.52196047e-03   2.43885369e-03   2.42675742e-03   2.41599790e-03
   2.39604554e-03   2.36884471e-03   2.35801443e-03   2.32743127e-03
   2.30837668e-03   2.26649832e-03   2.24899679e-03   2.20673094e-03
   2.20667166e-03   2.17995137e-03   2.17877713e-03   2.16056169e-03
   2.14886046e-03   2.12304846e-03   2.10551495e-03   2.10402366e-03
```

2.09445701e-03 2.08543574e-03 2.06632918e-03 2.05244761e-03
2.05008428e-03 2.04760856e-03 2.04035109e-03 2.03363279e-03
2.02542457e-03 2.01567621e-03 2.01042257e-03 1.94884073e-03
1.94672126e-03 1.93330222e-03 1.93006999e-03 1.92726070e-03
1.92672920e-03 1.89455042e-03 1.88314256e-03 1.86427207e-03
1.86395236e-03 1.85488846e-03 1.85373788e-03 1.83741304e-03
1.83255913e-03 1.82259332e-03 1.81670995e-03 1.80544969e-03
1.80390754e-03 1.79087653e-03 1.78707045e-03 1.78502316e-03
1.76376017e-03 1.76323480e-03 1.75493752e-03 1.73291785e-03
1.72797439e-03 1.69675410e-03 1.68808101e-03 1.68532969e-03
1.64304638e-03 1.63861660e-03 1.63783049e-03 1.62875455e-03
1.59218555e-03 1.58772185e-03 1.57899551e-03 1.57867308e-03
1.57607475e-03 1.56584503e-03 1.55891997e-03 1.55539475e-03
1.53593144e-03 1.52913588e-03 1.52380836e-03 1.49018283e-03
1.48109910e-03 1.46029662e-03 1.45974878e-03 1.45363118e-03
1.45178615e-03 1.43999894e-03 1.43838180e-03 1.42671234e-03
1.41591589e-03 1.41380571e-03 1.40699294e-03 1.40290361e-03
1.40274112e-03 1.39989932e-03 1.39076563e-03 1.39014700e-03
1.38348268e-03 1.38121062e-03 1.37713125e-03 1.37392739e-03
1.37220596e-03 1.36997064e-03 1.36647605e-03 1.36203771e-03
1.35674917e-03 1.35316815e-03 1.35161619e-03 1.35087378e-03
1.33195831e-03 1.32851482e-03 1.32141037e-03 1.32132345e-03
1.31638274e-03 1.29472559e-03 1.29214736e-03 1.28408247e-03
1.26586263e-03 1.26520076e-03 1.25196371e-03 1.24958206e-03
1.23250873e-03 1.22937395e-03 1.21933673e-03 1.20708082e-03
1.20618876e-03 1.20210055e-03 1.19034871e-03 1.18230512e-03
1.18062034e-03 1.17892628e-03 1.17856710e-03 1.17654597e-03
1.17337673e-03 1.17094003e-03 1.16754088e-03 1.16381088e-03
1.15287896e-03 1.14474285e-03 1.13052979e-03 1.12931198e-03
1.12511912e-03 1.11666053e-03 1.10787617e-03 1.10105812e-03
1.08899726e-03 1.08862237e-03 1.08389874e-03 1.08191449e-03
1.08012875e-03 1.07577253e-03 1.07177782e-03 1.06529938e-03
1.06157792e-03 1.06001868e-03 1.05576492e-03 1.05498624e-03
1.05182901e-03 1.05181520e-03 1.05161536e-03 1.05009927e-03
1.04720807e-03 1.03884907e-03 1.03514098e-03 1.03173474e-03
1.02593456e-03 1.02529025e-03 1.02152027e-03 1.02026352e-03
1.01980046e-03 1.01949330e-03 1.01556548e-03 1.01388173e-03
1.01118633e-03 1.00945767e-03 1.00846679e-03 1.00498197e-03
1.00196983e-03 1.00105834e-03 9.92024002e-04 9.84166627e-04
9.77569065e-04 9.72352713e-04 9.71465367e-04 9.70599107e-04
9.65712945e-04 9.59094155e-04 9.51134583e-04 9.49499488e-04
9.47336761e-04 9.43875816e-04 9.43797076e-04 9.41344221e-04
9.33796577e-04 9.28245202e-04 9.21434842e-04 9.19939810e-04
9.15558505e-04 9.13472673e-04 9.07298356e-04 9.05898238e-04
9.05610681e-04 9.03934357e-04 9.01802946e-04 8.97236564e-04
8.93254814e-04 8.89022022e-04 8.88722309e-04 8.88682165e-04
8.86164266e-04 8.85574066e-04 8.84975937e-04 8.84189389e-04
8.79696803e-04 8.78934148e-04 8.75118159e-04 8.70349523e-04

8.69880750e-04 8.68254201e-04 8.68046051e-04 8.65278549e-04
8.65255927e-04 8.58078429e-04 8.57847543e-04 8.53853006e-04
8.52733623e-04 8.52471368e-04 8.52375144e-04 8.48649583e-04
8.44316381e-04 8.43953300e-04 8.43779373e-04 8.40899108e-04
8.36946832e-04 8.36914608e-04 8.36593338e-04 8.35835117e-04
8.33832494e-04 8.32573249e-04 8.30805687e-04 8.29845483e-04
8.26776623e-04 8.23422156e-04 8.18517355e-04 8.15643668e-04
8.11954284e-04 8.11746999e-04 8.08079502e-04 8.06691072e-04
8.04878697e-04 8.04295292e-04 8.03435538e-04 7.95572838e-04
7.94295295e-04 7.94056844e-04 7.93411958e-04 7.88961494e-04
7.84467143e-04 7.73456953e-04 7.72784002e-04 7.71092134e-04
7.67221742e-04 7.67159824e-04 7.64566739e-04 7.63475696e-04
7.57940878e-04 7.55401636e-04 7.46694587e-04 7.43713839e-04
7.43025446e-04 7.39674790e-04 7.36968913e-04 7.33891943e-04
7.31959554e-04 7.30213939e-04 7.22019763e-04 7.20964606e-04
7.20749355e-04 7.18318740e-04 7.15719901e-04 7.10964156e-04
7.10469732e-04 7.08032298e-04 7.03595493e-04 7.01427617e-04
6.99645003e-04 6.97385575e-04 6.91169071e-04 6.86801760e-04
6.86156229e-04 6.84773848e-04 6.79710327e-04 6.77072977e-04
6.76688016e-04 6.75967149e-04 6.74503708e-04 6.73904372e-04
6.72913415e-04 6.72009081e-04 6.71649074e-04 6.64069208e-04
6.61954739e-04 6.59525383e-04 6.57056326e-04 6.54128580e-04
6.51773323e-04 6.47662329e-04 6.46593838e-04 6.46087426e-04
6.45904949e-04 6.38938547e-04 6.35025167e-04 6.33288816e-04
6.26771362e-04 6.24653645e-04 6.24414324e-04 6.22960186e-04
6.22722391e-04 6.22435340e-04 6.21941534e-04 6.21492303e-04
6.20380447e-04 6.18248090e-04 6.14987411e-04 6.14600871e-04
6.14516506e-04 6.12875202e-04 6.12697137e-04 6.12260268e-04
6.11074843e-04 6.11066202e-04 6.10957220e-04 6.09136047e-04
6.08998827e-04 6.07992165e-04 6.07404374e-04 6.05305316e-04
6.05021805e-04 6.04889024e-04 6.04478488e-04 6.03498510e-04
6.03215828e-04 6.02962599e-04 6.02856284e-04 6.02061409e-04
5.99030264e-04 5.98665041e-04 5.97036008e-04 5.96715541e-04
5.96653107e-04 5.94577054e-04 5.90895598e-04 5.89203391e-04
5.89159405e-04 5.83509779e-04 5.83119465e-04 5.82201372e-04
5.80414616e-04 5.79863709e-04 5.77152780e-04 5.76715458e-04
5.75875078e-04 5.75547405e-04 5.72746231e-04 5.72152087e-04
5.71523142e-04 5.71127455e-04 5.70759895e-04 5.67482509e-04
5.66239279e-04 5.66148789e-04 5.65427353e-04 5.63954497e-04
5.58465060e-04 5.56453853e-04 5.55237835e-04 5.54439081e-04
5.54380029e-04 5.53165563e-04 5.52661189e-04 5.51554016e-04
5.50958741e-04 5.49658429e-04 5.44356301e-04 5.43760057e-04
5.43413198e-04 5.41859463e-04 5.40896740e-04 5.40087005e-04
5.39293589e-04 5.39064460e-04 5.37267273e-04 5.37019596e-04
5.36248208e-04 5.35599293e-04 5.34623756e-04 5.32596211e-04
5.28818370e-04 5.27978439e-04 5.25117152e-04 5.24166382e-04
5.23933346e-04 5.22320856e-04 5.20862268e-04 5.20240019e-04
5.20065473e-04 5.19234207e-04 5.18987633e-04 5.17261866e-04

5.16328499e-04 5.15283465e-04 5.12905965e-04 5.08303270e-04
5.08175450e-04 5.08071988e-04 5.07344689e-04 5.06205525e-04
5.06155588e-04 5.04456961e-04 5.04216463e-04 5.03832094e-04
4.97918966e-04 4.97531555e-04 4.97147669e-04 4.96359758e-04
4.95979633e-04 4.95408768e-04 4.93912981e-04 4.93440534e-04
4.92084397e-04 4.91959118e-04 4.91771992e-04 4.89167959e-04
4.89062069e-04 4.86010522e-04 4.85689472e-04 4.84305710e-04
4.83925150e-04 4.82870758e-04 4.82862500e-04 4.81892643e-04
4.81679996e-04 4.81107131e-04 4.78115022e-04 4.71225530e-04
4.67727082e-04 4.67395434e-04 4.67103630e-04 4.66727427e-04
4.65290979e-04 4.63236418e-04 4.62694785e-04 4.62479640e-04
4.58313776e-04 4.57968108e-04 4.57728741e-04 4.57720239e-04
4.56874927e-04 4.56469997e-04 4.53579230e-04 4.52603160e-04
4.51225165e-04 4.50405160e-04 4.46941596e-04 4.45262688e-04
4.44704651e-04 4.40751469e-04 4.39569647e-04 4.39442731e-04
4.35036783e-04 4.34222683e-04 4.33890946e-04 4.33225612e-04
4.32766886e-04 4.32198919e-04 4.32186831e-04 4.30554236e-04
4.29212869e-04 4.20037783e-04 4.20003491e-04 4.17174643e-04
4.15859951e-04 4.13146730e-04 4.08669194e-04 4.04589613e-04
4.04554433e-04 4.03512353e-04 4.02702833e-04 4.01682319e-04
4.01349100e-04 4.00139017e-04 3.99711026e-04 3.96044571e-04
3.94368685e-04 3.93948411e-04 3.93424747e-04 3.92719516e-04
3.90810584e-04 3.89949687e-04 3.89703486e-04 3.85020250e-04
3.84330129e-04 3.84256229e-04 3.83594206e-04 3.81613413e-04
3.80426861e-04 3.79487353e-04 3.79126141e-04 3.78358070e-04
3.77239078e-04 3.77084648e-04 3.76527165e-04 3.75311822e-04
3.74046666e-04 3.73070437e-04 3.72335451e-04 3.71202652e-04
3.70056359e-04 3.69762669e-04 3.64870163e-04 3.64672645e-04
3.64006796e-04 3.56145810e-04 3.54164789e-04 3.53058483e-04
3.49454175e-04 3.47752365e-04 3.47194244e-04 3.46724832e-04
3.45100636e-04 3.43812135e-04 3.43315813e-04 3.41097154e-04
3.39649668e-04 3.39004469e-04 3.35806559e-04 3.34980099e-04
3.34057251e-04 3.31794412e-04 3.31504598e-04 3.28652582e-04
3.28520162e-04 3.28301210e-04 3.28216061e-04 3.27990376e-04
3.27983721e-04 3.27493290e-04 3.26947730e-04 3.25221408e-04
3.25164159e-04 3.24409736e-04 3.18535310e-04 3.17947277e-04
3.17424656e-04 3.17250006e-04 3.15203791e-04 3.11712309e-04
3.10555499e-04 3.10079279e-04 3.08073263e-04 3.07264215e-04
3.06076163e-04 3.04500527e-04 3.02410823e-04 2.99692410e-04
2.98202608e-04 2.97793898e-04 2.97590792e-04 2.96904438e-04
2.95885731e-04 2.95398712e-04 2.94815972e-04 2.93841483e-04
2.93489851e-04 2.93312176e-04 2.91635015e-04 2.90969736e-04
2.90408908e-04 2.84746502e-04 2.84469094e-04 2.82085245e-04
2.80615591e-04 2.80326624e-04 2.78479409e-04 2.77300778e-04
2.77199662e-04 2.74373497e-04 2.72918452e-04 2.72038490e-04
2.71988728e-04 2.71757621e-04 2.71492677e-04 2.70431441e-04
2.69481135e-04 2.69471016e-04 2.69025330e-04 2.67600936e-04
2.67543989e-04 2.66558075e-04 2.65958099e-04 2.65718207e-04

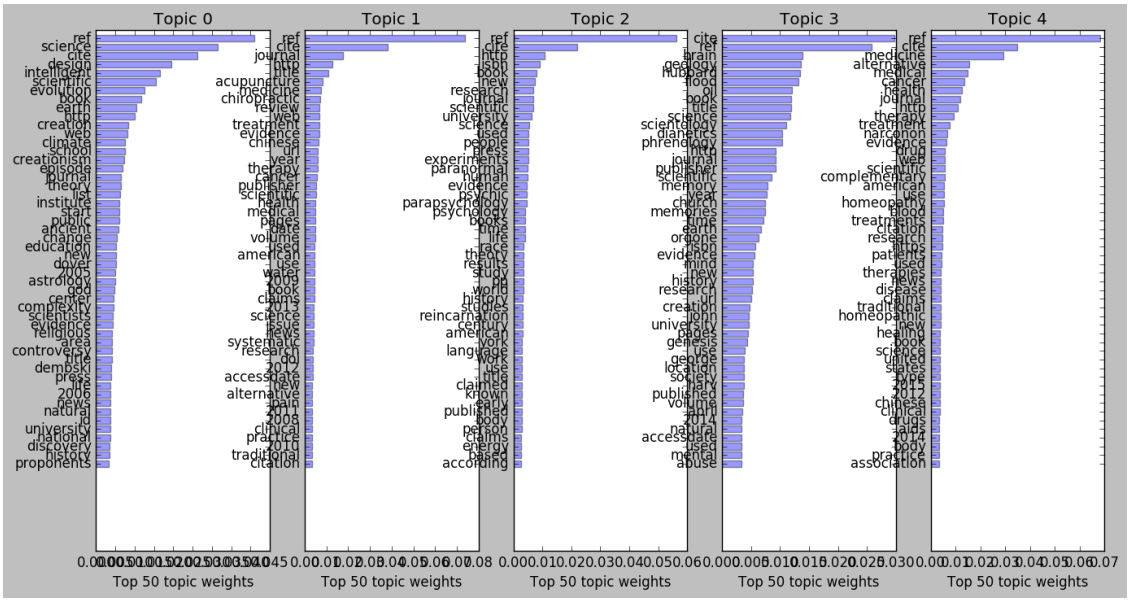
2.65680411e-04 2.65591666e-04 2.65332533e-04 2.64435494e-04
2.64170453e-04 2.62363029e-04 2.60240679e-04 2.59324466e-04
2.59282387e-04 2.58607129e-04 2.58432157e-04 2.57380171e-04
2.56977681e-04 2.56081375e-04 2.54976365e-04 2.54906078e-04
2.53482294e-04 2.53265738e-04 2.52756707e-04 2.51740958e-04
2.51367941e-04 2.50192453e-04 2.46650648e-04 2.45448834e-04
2.45200370e-04 2.44930972e-04 2.44467760e-04 2.43687700e-04
2.41662274e-04 2.41397985e-04 2.39708681e-04 2.39519987e-04
2.39357217e-04 2.39131411e-04 2.38977023e-04 2.38762144e-04
2.37657960e-04 2.37504236e-04 2.37497597e-04 2.36793293e-04
2.35096301e-04 2.33603233e-04 2.31979534e-04 2.29757561e-04
2.28654471e-04 2.27597537e-04 2.27510179e-04 2.27136311e-04
2.26493893e-04 2.22503961e-04 2.19763760e-04 2.19109621e-04
2.17700869e-04 2.17415582e-04 2.17283986e-04 2.16705172e-04
2.15945666e-04 2.15755389e-04 2.15359679e-04 2.14922768e-04
2.14853197e-04 2.12315768e-04 2.11849252e-04 2.10013134e-04
2.06681000e-04 2.05339297e-04 2.05184182e-04 2.04897327e-04
2.04708262e-04 2.04518882e-04 2.03768760e-04 2.03717297e-04
2.02337526e-04 2.01522594e-04 2.00994262e-04 1.98519759e-04
1.98216642e-04 1.97908178e-04 1.97590214e-04 1.97547789e-04
1.96305275e-04 1.95026420e-04 1.94085260e-04 1.93984260e-04
1.93535694e-04 1.93389287e-04 1.93380757e-04 1.93220202e-04
1.93128729e-04 1.92265149e-04 1.91556356e-04 1.89524590e-04
1.87522740e-04 1.87234173e-04 1.85808993e-04 1.84064134e-04
1.83496024e-04 1.80534351e-04 1.80132378e-04 1.77187429e-04
1.77181355e-04 1.77017181e-04 1.76345151e-04 1.74111738e-04
1.73400306e-04 1.73345302e-04 1.71604339e-04 1.70739574e-04
1.68690992e-04 1.67034386e-04 1.63769733e-04 1.63652185e-04
1.63448140e-04 1.59680405e-04 1.59527271e-04 1.59356346e-04
1.58646117e-04 1.58104266e-04 1.56884616e-04 1.54277873e-04
1.54199646e-04 1.52040451e-04 1.50454689e-04 1.50163369e-04
1.48170158e-04 1.47200156e-04 1.46353870e-04 1.46023752e-04
1.45068791e-04 1.42330206e-04 1.40447238e-04 1.40050491e-04
1.39525100e-04 1.38265459e-04 1.35359692e-04 1.33237799e-04
1.31955480e-04 1.31422068e-04 1.30724307e-04 1.30354887e-04
1.30084323e-04 1.29666633e-04 1.28796729e-04 1.28711512e-04
1.28501794e-04 1.28233480e-04 1.28039475e-04 1.23684238e-04
1.21295955e-04 1.19544234e-04 1.19141977e-04 1.18710592e-04
1.17893776e-04 1.16510226e-04 1.12489499e-04 1.11501068e-04
1.10201772e-04 1.09622018e-04 1.09582183e-04 1.09170067e-04
1.08690713e-04 1.07552606e-04 1.07441218e-04 1.07019390e-04
1.05516798e-04 1.03791695e-04 1.00197573e-04 9.64836344e-05
9.16157493e-05 8.87744887e-05 8.85311810e-05 8.83193638e-05
8.75677667e-05 8.66999642e-05 8.31690125e-05 8.00392435e-05
7.98384038e-05 7.97955417e-05 7.89316316e-05 7.84436413e-05
7.56860586e-05 7.49307546e-05 7.43282682e-05 7.33530676e-05
7.27187405e-05 7.26771694e-05 7.18473008e-05 7.16397545e-05
6.87998565e-05 6.83417167e-05 6.81205890e-05 6.62941372e-05

```
6.61811808e-05   6.61147109e-05   6.57644204e-05   6.51235850e-05
6.44966923e-05   6.30325020e-05   6.20407228e-05   6.11289006e-05
5.98682630e-05   5.70366452e-05   5.65983872e-05   5.63799342e-05
5.26662373e-05   5.14168624e-05   5.10984706e-05   5.10769772e-05
5.04744286e-05   5.03603710e-05   5.03470819e-05   4.90188132e-05
4.82580501e-05   4.80521540e-05   4.76488789e-05   4.75825266e-05
4.64272329e-05   4.60831897e-05   4.54680510e-05   4.51058419e-05
4.45138984e-05   4.42054187e-05   4.40031789e-05   4.21235764e-05
4.10214524e-05   4.00068648e-05   3.94958882e-05   3.83024056e-05
3.80050661e-05   3.79005384e-05   3.66476949e-05   3.65769079e-05
3.65095919e-05   3.59425465e-05   3.57615933e-05   3.52287571e-05
3.47435614e-05   3.46405012e-05   3.46235122e-05   3.39966799e-05
3.37990289e-05   3.36304942e-05   3.33181162e-05   3.29565684e-05
3.27316016e-05   3.23545139e-05   3.21516888e-05   3.20140766e-05
3.15919618e-05   3.13385481e-05   3.13269041e-05   3.05353113e-05
3.02438522e-05   3.01237367e-05   2.95501594e-05   2.81059510e-05
2.66606622e-05   2.62622445e-05   2.58326193e-05   2.57784074e-05
2.55480514e-05   2.41395671e-05   2.35610130e-05   2.34920456e-05
2.26977412e-05   2.22262998e-05   2.17933350e-05   2.17890530e-05
2.16421255e-05   2.15729577e-05   2.12465338e-05   2.11988952e-05
2.06416644e-05   1.97294570e-05   1.95531549e-05   1.93737527e-05
1.92782347e-05   1.88873293e-05   1.88119266e-05   1.82590133e-05
1.76873753e-05   1.76784467e-05   1.76704920e-05   1.71647660e-05
1.71274483e-05   1.68809251e-05   1.67611271e-05   1.67313624e-05
1.61932741e-05   1.61711140e-05   1.61570560e-05   1.60870656e-05
1.60381833e-05   1.57293914e-05   1.56608523e-05   1.56567961e-05
1.56149575e-05   1.55362720e-05   1.51922285e-05   1.50189096e-05
1.48050908e-05   1.46726188e-05   1.44956349e-05   1.44454274e-05
1.42089332e-05   1.41650607e-05   1.37166778e-05   1.36508446e-05
1.35144807e-05   1.33609916e-05   1.32217068e-05   1.31577900e-05
1.29790762e-05   1.26447818e-05   1.25739597e-05   1.22975313e-05
1.21623653e-05   1.20069549e-05   1.19646281e-05   1.15543197e-05
1.14770959e-05   1.13621760e-05   1.10288956e-05   1.06796561e-05
1.03555538e-05   1.02608482e-05   1.02263387e-05   1.01761508e-05
1.01038945e-05   1.00187196e-05   9.76639306e-06   9.30292491e-06]
```

Exercise: Represent graphically the topic distributions for the top 25 tokens with highest probability for each topic.

[116]:
```
# SORTED TOKEN FREQUENCIES (II):
plt.rcdefaults()

n_bins = 50

# Example data
y_pos = range(n_bins-1, -1, -1)

pylab.rcParams['figure.figsize'] = 16, 8  # Set figure size
```

```
for i in range(n_topics):

    ### Plot top 25 tokens for topic i
    # Read i-thtopic
    # scode: <FILL IN>
    topic_i = topic_probs[i]
    rank = np.argsort(- topic_i)[0:n_bins]

    tokens = [tf_feature_names[r] for r in rank]
    weights = [topic_i[r] for r in rank]

    # Plot
    # scode: <FILL IN>
    plt.subplot(1, n_topics, i+1)
    plt.barh(y_pos, weights, align='center', alpha=0.4)
    plt.yticks(y_pos, tokens)
    plt.xlabel('Top {0} topic weights'.format(n_bins))
    plt.title('Topic {0}'.format(i))

plt.show()
```



Exercise: Explore the influence of the concentration parameters, *alpha* (`doc_topic_prior` in `sklearn`) and *eta*(`topic_word_prior`). In particular observe how do topic and document distributions change as these parameters increase.

[ ]:

** Exercise**: The token dictionary and the token distribution have shown that:

1. Some tokens, despite being very frequent in the corpus, have no semantic relevance for topic modeling. Unfortunately, they were not present in the stopword list, and have not been elliminated before the analysis.
2. A large portion of tokens appear only once and, thus, they are not statistically relevant for the inference engine of the topic models.

Revise the entire corpus be removing from the corpus all these sets of terms.

[ ]:

** Exercise**: 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.

[ ]:

** Exercise**: The topic modelling algorithms we have tested in this notebook are non-supervised. This makes them difficult to evaluate objectivelly. In order to test if LDA captures real topics, construct a dataset as the mixture of wikipedia articles from 4 different categories, and test if LDA with 4 topics identifies topics closely related to the original categories.

[ ]:

Exercise: Represent, in a scatter plot, the distribution of component 0 and 1 of the document-topic matrix (`corpus_lda`) for the whole corpus.

[ ]: