IM-UH 1511 **Introduction to Digital Humanities**

HOMEWORK 4

Wikipedia Networks and Topic Modeling

50 points totally

In [1]:

```
import urllib, os, codecs, random, operator, re, string, copy, dateutil.par
import wikipedia
import wikipediaapi
import pygraphviz
from networkx.drawing.nx_agraph import graphviz_layout
from collections import Counter
from string import punctuation, digits
import pathlib
import spacy
from spacy import displacy
nlp = spacy.load('en_core_web_lg')
import inflect
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize
from textblob import TextBlob
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *
stemmer = PorterStemmer()

import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from gensim import corpora, models
from gensim.corpora import Dictionary
import pyLDAvis
from pyLDAvis import gensim as pgensim
pyLDAvis.enable_notebook()

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.simplefilter('ignore')
```

Random Selection of a Wikipedia Page

In [2]:

```
excluded=['International Standard Book Number',
          'International Standard Serial Number',
          'JSTOR',
          'Library of Congress Control Number',
          "Digital object identifier",
          "Integrated Authority File",
          "PubMed Identifier",
          "PubMed Central",
          "OCLC",
          "Wayback Machine",
          "ArXiv",
          "Bibcode",
          "ACM Computing Classification System",
          "Academic Press",
          "Website",
          "World Wide Web",
          "BioRxiv",
          "CiteSeerX",
          "Telecommunication network",
          "Web sites",
          "Daylight saving time",
          "International Standard Name Identifier",
          "Système universitaire de documentation",
          "Virtual International Authority File",
          "WorldCat Identities",
          "Copyrighted",
          "Trademarked",
          "Bibliothèque nationale de France",
          "YouTube",
          "IMDb",
          "Personal name",
          "Wikidata",
          "SNAC",
          "Table of contents",
          "DVD-Video",
          "End user",
          "Album",
          "AllMusic",
          "Discogs",
          "Music genre",
          "Record label",
          "Record producer",
          "Integrated Taxonomic Information System",
          "ABC-CLIO",
          "Dictionary of National Biography",
          "Table of contents",
          "Video",
          "Geographic coordinate system",
          "Resource Description Framework",
          "Metadata",
          "Wikimedia Commons",
          "Wikimedia Foundation",
          "Google",
          'Electronic publication'
            ]
```

In [3]:

```
n=10  # minimum number of hyperlinks of the selected wikipedia page
N=21  # maximum number of hyperlinks of the selected wikipedia page

while True:
    try:
        page=wikipedia.page(wikipedia.random())
        hl=[w for w in page.links if w not in excluded]
        hl=[w.replace(" ","_") for w in hl]
    except wikipedia.DisambiguationError as e:
        pass
    except wikipedia.exceptions.PageError as e:
        pass
    if len(hl)>n and len(hl)<N:
        break

page_title=page.title
p=sorted(hl)

print("The %i hyperlinks from the (randomly) selected '%s' Wikipedia page a
for i in range(len(p)):
    print("%i. %s" %(i+1,p[i]))
```

```
The 15 hyperlinks from the (randomly) selected 'Tuberaspis' Wikipedia pag
e are:

1. Animal
2. Arthropod
3. Cambrian_Period
4. Dresbachian
5. Extinct
6. Faunal_stage
7. Fossil
8. Fossilworks
9. Genus
10. Global_Biodiversity_Information_Facility
11. Interim_Register_of_Marine_and_Nonmarine_Genera
12. Ptychopariida
13. Taxonomy_(biology)
14. Trilobita
15. Trilobite
```

In [4]:

```
# # Saving the random sample of hyperlinks

ft = "Wikipedia_page.txt"
f = open(ft,"w")
f.write(page_title)
f.close()

ff = "hyperlinks_"+page_title+".pkl"
with open(ff, 'wb') as f:
    pickle.dump(p, f)
```

In [5]:

```
# # Reading the saved random sample of hyperlinks

# ft = "Wikipedia_page.txt"
# f = open(ft,"r+")
# page_title = f.read()

# ff = "hyperlinks_"+page_title+".pkl"
# with open(ff, 'rb') as f:
#     p = pickle.load(f)

# print("The %i hyperlinks from the (randomly) selected '%s' Wikipedia page
# for i in range(len(p)):
#     print("%i. %s" %(i+1,p[i]))
```

In [6]:

```
# # IF ONE WANTS TO USE A PREDEFINED WEBPAGE,
# # one should uncomment and run the following lines
# # Here, I've considered as predefined Wikipedia page the page of the
# # 'United_States_women%27s_national_soccer_team' but you may take
# # instead anything you like. Notice, that subsequently I am filtering
# # a random sample of 20 hyperlinks. This might need to change if the
# # page you have selected has fewer hyperlinks.

# # page=wikipedia.page('United_States_women%27s_national_soccer_team') #'h
# # https://en.wikipedia.org/wiki/Los_Angeles_Lakers

# page=wikipedia.page('Digital_humanities') #Interactive_media')  # Los_Ang
# # 'Digital_humanities' ---> 'New_York_University_Abu_Dhabi'
# page_title1=page.title

# print("The Wikipedia page '%s' has totally %i hyperlinks: \n" %(page_titl

# number_of_hyperlinks = 20
# p1=random.sample(sorted([w.replace(" ","_") for w in page.links if w not

# print("The %i randomly selected hyperlinks from the Wikipedia page '%s' a
# for i in range(len(p1)):
#     print("%i. %s" %(i+1,p1[i]))
```

```
The Wikipedia page 'Digital humanities' has totally 221 hyperlinks:

The 20 randomly selected hyperlinks from the Wikipedia page 'Digital huma
nities' are:
1. Cyborg_anthropology
2. Virtual_reality
3. Internet
4. JSON-LD
5. Voyant_Tools
6. SAWSDL
7. Readability
8. Religious_studies
9. Electronic_publication
10. Electronic_literature
11. Library_2.0
12. Dublin_Core
13. Semantic_Web_Rule_Language
14. Semantic_computing
15. Neologism
16. Digitization
17. HProduct
18. SciCrunch
19. Communication_studies
20. Extensible_Markup_Language
```

In [7]:
```
# # Saving the random sample of hyperlinks

# ft = "Wikipedia_page1.txt"
# f = open(ft,"w")
# f.write(page_title1)
# f.close()

# ff = "hyperlinks_"+page_title1+".pkl"
# with open(ff, 'wb') as f:
#     pickle.dump(p1, f)
```

In [3]:
```
# Reading the saved random sample of hyperlinks

ft = "Wikipedia_page1.txt"
f = open(ft,"r+")
page_title = f.read()

ff = "hyperlinks_"+page_title+".pkl"
with open(ff, 'rb') as f:
    p1 = pickle.load(f)

print("The %i hyperlinks from the (randomly) selected '%s' Wikipedia page a
for i in range(len(p1)):
    print("%i. %s" %(i+1,p1[i]))
```

```
The 20 hyperlinks from the (randomly) selected 'Tuberaspis' Wikipedia pag
e are:

1. Cyborg_anthropology
2. Virtual_reality
3. Internet
4. JSON-LD
5. Voyant_Tools
6. SAWSDL
7. Readability
8. Religious_studies
9. Electronic_publication
10. Electronic_literature
11. Library_2.0
12. Dublin_Core
13. Semantic_Web_Rule_Language
14. Semantic_computing
15. Neologism
16. Digitization
17. HProduct
18. SciCrunch
19. Communication_studies
20. Extensible_Markup_Language
```

Wikipedia Network

In [9]:

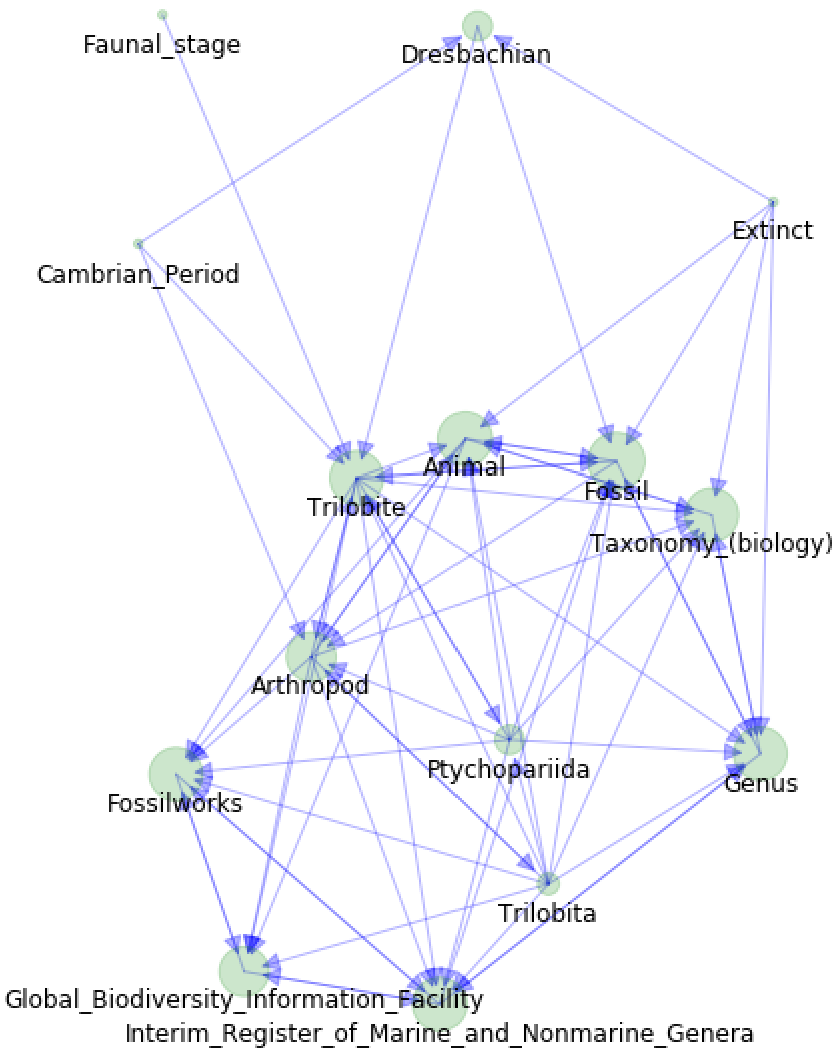
```
eds=[]
s=0
for pp in p1:
    try:
        ppp=sorted([w.replace(" ","_") for w in wikipedia.page(pp).links])
    except wikipedia.exceptions.PageError as e:
        pass
    except wikipedia.exceptions.DisambiguationError as e:
        pass
    ppp=[x for x in ppp if x!=pp]
    ih=set(ppp).intersection(set(p1))
#     if len(ih)>0:
    s+=1
    print("%i. %s has %i hyperlinks to webpages: \n %s" %(s,pp,len(ih),", "
    print(" ")
    for q in ih:
        eds.append((pp,q))
```

```
1. Animal has 5 hyperlinks to webpages:
 Fossil, Arthropod, Taxonomy_(biology), Global_Biodiversity_Information_F
acility, Fossilworks

2. Arthropod has 7 hyperlinks to webpages:
 Taxonomy_(biology), Global_Biodiversity_Information_Facility, Animal, In
terim_Register_of_Marine_and_Nonmarine_Genera, Trilobite, Fossilworks, Tr
ilobita

3. Cambrian_Period has 3 hyperlinks to webpages:
 Dresbachian, Trilobite, Arthropod

4. Dresbachian has 2 hyperlinks to webpages:
 Trilobite, Fossil

5. Extinct has 5 hyperlinks to webpages:
 Dresbachian, Fossil, Taxonomy_(biology), Genus, Animal

6. Faunal_stage has 1 hyperlinks to webpages:
 Trilobite

7. Fossil has 4 hyperlinks to webpages:
 Animal, Trilobite, Genus, Arthropod

8. Fossilworks has 2 hyperlinks to webpages:
 Interim_Register_of_Marine_and_Nonmarine_Genera, Global_Biodiversity_Inf
ormation_Facility

9. Genus has 3 hyperlinks to webpages:
 Taxonomy_(biology), Fossil, Interim_Register_of_Marine_and_Nonmarine_Gen
era

10. Global_Biodiversity_Information_Facility has 2 hyperlinks to webpage
s:
 Interim_Register_of_Marine_and_Nonmarine_Genera, Fossilworks

11. Interim_Register_of_Marine_and_Nonmarine_Genera has 4 hyperlinks to w
ebpages:
 Global_Biodiversity_Information_Facility, Fossilworks, Fossil, Genus

12. Ptychopariida has 8 hyperlinks to webpages:
 Fossil, Arthropod, Taxonomy_(biology), Genus, Animal, Interim_Register_o
f_Marine_and_Nonmarine_Genera, Trilobite, Fossilworks

13. Taxonomy_(biology) has 2 hyperlinks to webpages:
 Animal, Genus

14. Trilobita has 10 hyperlinks to webpages:
 Fossil, Ptychopariida, Arthropod, Taxonomy_(biology), Global_Biodiversit
y_Information_Facility, Genus, Animal, Interim_Register_of_Marine_and_Non
marine_Genera, Trilobite, Fossilworks

15. Trilobite has 9 hyperlinks to webpages:
 Fossil, Ptychopariida, Arthropod, Taxonomy_(biology), Global_Biodiversit
y_Information_Facility, Genus, Animal, Interim_Register_of_Marine_and_Non
marine_Genera, Fossilworks
```

In [10]:

```
G=nx.DiGraph()
G.add_edges_from(eds)
print("The randomly sampled Wikipedia graph of '%s' has %i nodes (pages) an

pos=graphviz_layout(G)

plt.figure(figsize=(10,10));
nsi=[20+100*G.in_degree(n) for n in G.nodes()]
nodes = nx.draw_networkx_nodes(G, pos, node_color="g", node_size=nsi, alpha
nx.draw_networkx_edges(G, pos,arrowsize=20, edge_color="b", alpha=0.3)
# nx.draw_networkx_labels(G, pos)
plt.axis('off');
yoffset = {}
y_off = -5 # offset on the y axis
for k, v in pos.items():
    yoffset[k] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(G, yoffset,font_size=12);
sst="The graph of the %i Wikipedia webpages which are \n hyperlinks of the
plt.title(sst,fontsize=15);
plt.margins(x=0.3, y=0)
```

```
The randomly sampled Wikipedia graph of 'Tuberaspis' has 15 nodes (pages)
and 67 edges (hyperlinks)
```



The graph of the 15 Wikipedia webpages which are hyperlinks of the Wikipedia page 'Tuberaspis'

In [11]:

```
# # print(pos['Geographic_coordinate_system'])
# pos['Geographic_coordinate_system']=(257.87, 82)
# # print(pos['Princess_Astrid_Coast'])
# pos['Princess_Astrid_Coast']=(266.2, 100)
# # print(pos["Hoel_Mountains"])
# pos["Hoel_Mountains"]=(248.52, 148)
# # print(pos["German_language"])=(139.79, 185)
# pos["German_language"]=(139.79, 185)

# plt.figure(figsize=(10,10));
# nsi=[100*G.in_degree(n) for n in G.nodes()]
# nodes = nx.draw_networkx_nodes(G, pos, node_color="g", node_size=nsi, alp
# nx.draw_networkx_edges(G, pos,arrowsize=20, edge_color="b", alpha=0.7)
# nx.draw_networkx_labels(G, pos)
# plt.axis('off');
# yoffset = {}
# y_off = -5 # offset on the y axis
# for k, v in pos.items():
#     yoffset[k] = (v[0], v[1]+y_off)
# nx.draw_networkx_labels(G, yoffset,font_size=12);
# sst="The graph of the %i Wikipedia webpages which are \n hyperlinks of th
# plt.title(sst,fontsize=15);
# plt.margins(x=0.3, y=0)
```

Topic Modeling of Summaries of Wikipedia Pages

In [12]:

```
wiki_wiki = wikipediaapi.Wikipedia('en')
docs_d={}
for q in p:
#     summary=wikipedia.summary(q.replace("_"," "))
#     docs_d[q]=summary
    page_py = wiki_wiki.page(q)
    docs_d[q]=page_py.summary

for q in p:
    for k,v in docs_d.items():
        if k==q:
            print("%s Summary: \n %s" %(k,v))
            print(" ")
```

```
Animal Summary:
 Animals are multicellular eukaryotic organisms that form the biological
kingdom Animalia. With few exceptions, animals consume organic material,
breathe oxygen, are able to move, can reproduce sexually, and grow from a
hollow sphere of cells, the blastula, during embryonic development. Over
1.5 million living animal species have been described—of which around 1 m
illion are insects—but it has been estimated there are over 7 million ani
mal species in total. Animals range in length from 8.5 millionths of a me
tre to 33.6 metres (110 ft). They have complex interactions with each oth
er and their environments, forming intricate food webs. The kingdom Anima
lia includes humans, but in colloquial use the term animal often refers o
nly to non-human animals. The scientific study of animals is known as zoo
logy.
Most living animal species are in the Bilateria, a clade whose members ha
ve a bilaterally symmetric body plan. The Bilateria include the protostom
es—in which many groups of invertebrates are found, such as nematodes, ar
thropods, and molluscs—and the deuterostomes, containing both the echinod
erms as well as the chordates, the latter containing the vertebrates. Lif
e forms interpreted as early animals were present in the Ediacaran biota
```

In [13]:

```
stop_words = stopwords.words('english') #+ list(punctuation)

def tokenize(text):
    words = word_tokenize(text)
    words = [w.lower() for w in words]
    return [w for w in words if w not in stop_words and not w.isdigit()]
```

In [14]:

```
vocabulary = set()
docs=list(docs_d.values())
for i in docs:
    words = tokenize(i)
    vocabulary.update(words)

vocabulary = list(vocabulary)
word_index = {w: idx for idx, w in enumerate(vocabulary)}

VOCABULARY_SIZE = len(vocabulary)
DOCUMENTS_COUNT = len(docs)

print(VOCABULARY_SIZE, DOCUMENTS_COUNT)
```

```
1213 15
```

In [15]:

```
def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))
def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(toke
#              result.append(token)
            result.append(lemmatize_stemming(token))
    return result
```

In [16]:

```
excl=[u'ere',u'ye',u'wouldn',u'madam',u'happened']
# processed_docs = ppdocs #[preprocess(doc) for doc in documents]
processed_docs = [preprocess(doc) for doc in docs]
processed_docs1=[]
for x in processed_docs:
    y=[]
    for xx in x:
        if xx not in excl:
            y.append(xx)
    processed_docs1.append(y)
processed_docs=processed_docs1
allw=[]
for x in processed_docs:
    for xx in x:
        if xx not in allw:
            allw.append(xx)
print(len(allw)) #All 5752
# processed_docs[:10]
```

```
903
```

In [17]:

```
allws=[]
for z in processed_docs:
    for zz in z:
        allws.append(zz)
print(len(allws),len(set(allws)))
# sorted(allws)
allwd=Counter(allws)
print(len(allwd))
# for p,q in allwd.items():
#     print(p,q)
# count = 0
# for k in sorted(allwd.keys()):
#     print(k)
#     count += 1
#     if count > 50:
#         break
```

```
2015 903
903
```

In [18]:

```
dictionary = gensim.corpora.Dictionary(processed_docs)
print(len(dictionary)) #All 32368
# count = 0
# for k, v in dictionary.iteritems():
#     print(k, v)
#     count += 1
#     if count > 10:
#         break
```

```
903
```

In [19]:

```
dictionary.filter_extremes(no_below=3, no_above=0.8, keep_n=300)
len(dictionary)
```

Out[19]:

```
112
```

In [20]:

```
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
# bow_corpus[43]
```

In [21]:

```
nt=4 #number_of_topics
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=nt, id2word=d
```

In [22]:

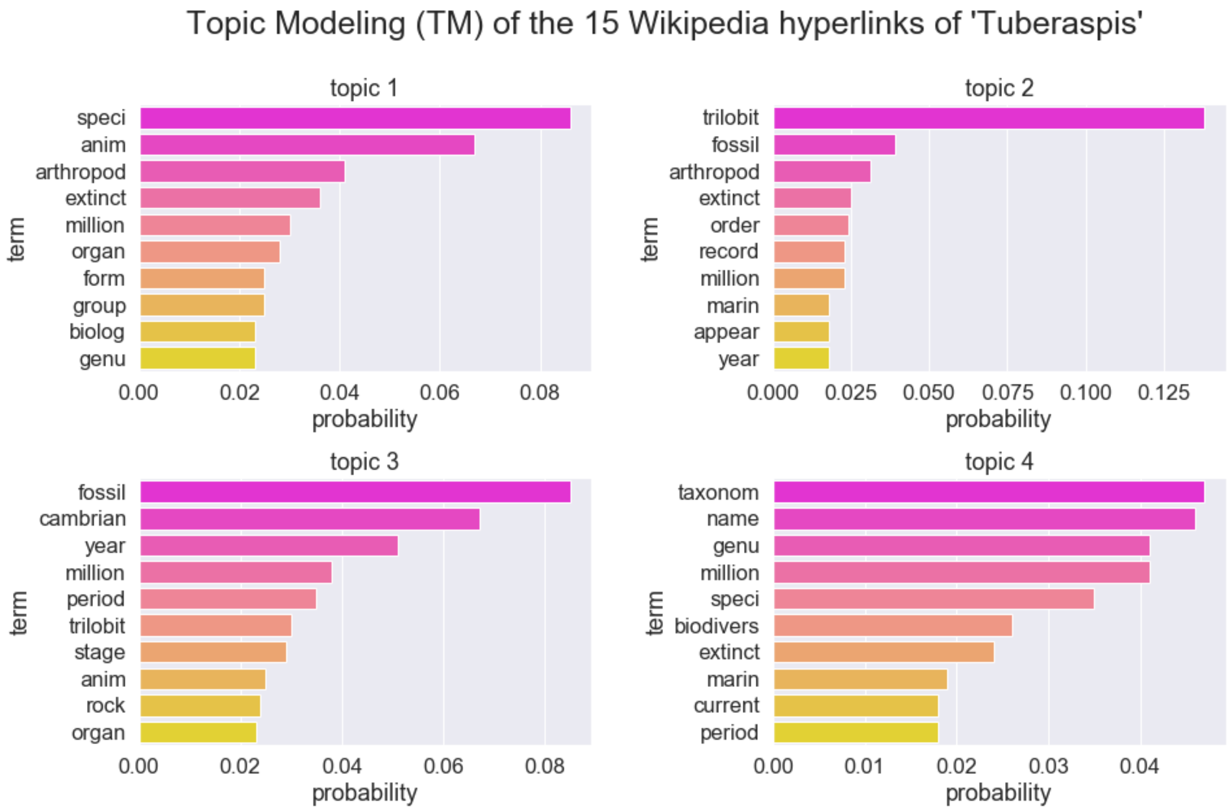
```
topics = lda_model.print_topics(num_words=25) #350

terms=[]
lt=[]
for i in range(nt):
    for t in topics:
        lt.append(t[1].split(" + "))
for s in lt:
    for ss in s:
        terms.append(ss[6:])
#          if re.sub(r'[^a-zA-Z]','', ss) not in terms:
#              terms.append(re.sub(r'[^a-zA-Z]','', ss))
terms=[t.replace('"',"") for t in terms]
terms=sorted(set(terms))
print(len(terms))
print(" ")
print("LIST OF TM TERMS:")
print(" ")
for i in terms:
    print(i)
```

```
62

LIST OF TM TERMS:

access
anim
appear
arthropod
biodivers
biolog
call
cambrian
classif
common
consist
contain
current
defin
describ
```

In [23]:
```
sss=[]
for idx, topic in lda_model.print_topics(-1):
    tt=[]
    s=topic.split(" + ")
    ss=[]
    uu=[]
    for t in s:
        u0=float(t.split("*")[0])
        u1=t.split("*")[1].replace('"','')
        if (u1,u0) not in ss:
            ss.append((u1,u0))
        if t not in uu:
            uu.append(t)
    sss.append(ss)
    topic=" + ".join(uu).encode('utf-8')
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: b'0.086*"speci" + 0.067*"anim" + 0.041*"arthropod" + 0.036*"extinc
t" + 0.030*"million" + 0.028*"organ" + 0.025*"form" + 0.025*"group" + 0.0
23*"biolog" + 0.023*"genu"'
Topic: 1
Words: b'0.138*"trilobit" + 0.039*"fossil" + 0.031*"arthropod" + 0.025*"e
xtinct" + 0.024*"order" + 0.023*"record" + 0.023*"million" + 0.018*"mari
n" + 0.018*"appear" + 0.018*"year"'
Topic: 2
Words: b'0.085*"fossil" + 0.067*"cambrian" + 0.051*"year" + 0.038*"millio
n" + 0.035*"period" + 0.030*"trilobit" + 0.029*"stage" + 0.025*"anim" +
0.024*"rock" + 0.023*"organ"'
Topic: 3
Words: b'0.047*"taxonom" + 0.046*"name" + 0.041*"genu" + 0.041*"million"
+ 0.035*"speci" + 0.026*"biodivers" + 0.024*"extinct" + 0.019*"marin" +
0.018*"current" + 0.018*"period"'
```

In [24]:
```
fig=plt.figure(figsize=(15,25)) #figsize=(15,2.4*15*((nt+1)/4))); #15
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(nt):
    sns.set(font_scale = 1.5)
    df=pd.DataFrame(sss[i], columns=['term','prob']).set_index('term')
#     plt.subplot(nt+1,2,i+1); #5
    ax = fig.add_subplot(nt+1,2,i+1)
    plt.title('topic '+str(i+1));
    sns.barplot(x='prob', y=df.index, data=df, label='Cities', palette='spr
    plt.xlabel('probability');
ss1="%i Wikipedia hyperlinks of '%s'" %(len(p),page_title)
sst="Topic Modeling (TM) of the %s" %ss1
plt.suptitle(sst,fontsize=25, y=0.92);
plt.show()
```


Topic Modeling (TM) of the 15 Wikipedia hyperlinks of 'Tuberaspis'

In [25]:

```
from pyLDAvis import gensim as pgensim
vis = pgensim.prepare(lda_model,bow_corpus, dictionary)
vis
```

Out[25]:

The Graph of Sententially Co-Occurrent TM Terms

In [26]:
```
terms
```

Out[26]:
```
['access',
 'anim',
 'appear',
 'arthropod',
 'biodivers',
 'biolog',
 'call',
 'cambrian',
 'classif',
 'common',
 'consist',
 'contain',
 'current',
 'defin',
 'describ',
 'develop',
 'earli',
 'earliest',
 'event',
 'except',
 'exoskeleton',
 'extinct',
 'facilit',
 'famili',
 'form',
 'fossil',
 'gener',
 'genu',
 'geolog',
 'group',
 'hierarchi',
 'human',
 'import',
 'includ',
 'inform',
 'intern',
 'know',
 'life',
 'live',
 'lower',
 'marin',
 'million',
 'modern',
 'name',
 'order',
 'organ',
 'period',
 'place',
 'possibl',
 'rang',
 'record',
 'rock',
 'scientif',
 'seri',
 'singl',
 'speci',
 'stage',
 'taxonom',
 'trilobit',
 'usual',
 'wide',
 'year']
```

In [27]:

```
pre=[]
for i in range(len(terms)):
    start=terms[i][:4]
    pre.append(start)
for j,k in Counter(pre).items():
    if k>1:
        print(j)
```

```
earl
```

In [28]:

```
# Without aliases

alias_dict={}

for n in terms:
    alias_dict[n]=n
```

In [29]:

```
# # With aliases

# alias_dict={}

# Without aliases

# for n in terms:
#     if n=="champion":
#         alias_dict[n]="champion"
#     elif n=="championship":
#         alias_dict[n]="champion"
#     elif n=="competit":
#         alias_dict[n]="competit"
#     elif n=="compet":
#         alias_dict[n]="competit"
#     if n=="anxiety":
#         alias_dict[n]="anxiety"
#     elif n=="anxious":
#         alias_dict[n]="anxiety"
#     if n=="fall":
#         alias_dict[n]="fall"
#     elif n=="fallen":
#         alias_dict[n]="falling"
#     elif n=="falling":
#         alias_dict[n]="falling"
#     elif n=="laugh":
#         alias_dict[n]="laugh"
#     elif n=="laughed":
#         alias_dict[n]="laugh"
#     elif n=="spirit":
#         alias_dict[n]="spirit"
#     elif n=="spirits":
#         alias_dict[n]="spirit"
#     elif n=="tells":
#         alias_dict[n]="telling"
#     elif n=="telling":
#         alias_dict[n]="telling"
#     elif n=="wished":
#         alias_dict[n]="wishes"
#     elif n=="wishes":
#         alias_dict[n]="wishes"
#     else:
#         alias_dict[n]=n
# print("The dictionary of aliases has %i keys (terms) and %i unique values
# for k,v in alias_dict.items():
#     print(k,"-->",v)
```

In [30]:

```
tdocs=" ".join(docs)
blob = TextBlob(tdocs)
textSentences = blob.sentences
sendic=dict()
for i,v in enumerate(textSentences):
    sent=v.sentiment.polarity
    wl=[]
    for term in list(set(alias_dict.values())):
        if term in v:
            wl.append(term)
    if len(wl)>1:
        sendic[i]=wl
medges=[]
for k,v in sendic.items():
    sent=textSentences[k].sentiment.polarity
    dd={}
    ps=set()
    for j in itertools.combinations(v, 2):
        ps.add(j)
        dd[j]=(k,sent)
    for jj in ps:
        s=0
        ss=0
        for kk,vv in dd.items():
            if kk==jj:
                s+=1
                ss+=vv[1]
        if alias_dict[jj[0]]!=alias_dict[jj[1]]:
            medges.append((alias_dict[jj[0]],alias_dict[jj[1]],"Sentence_"+
print("The %s contain %i sentential co-occurrences among %i TM terms"%(ss1,
medges
```

```
The 15 Wikipedia hyperlinks of 'Tuberaspis' contain 1116 sentential co-oc
currences among 62 TM terms
```

In [31]:

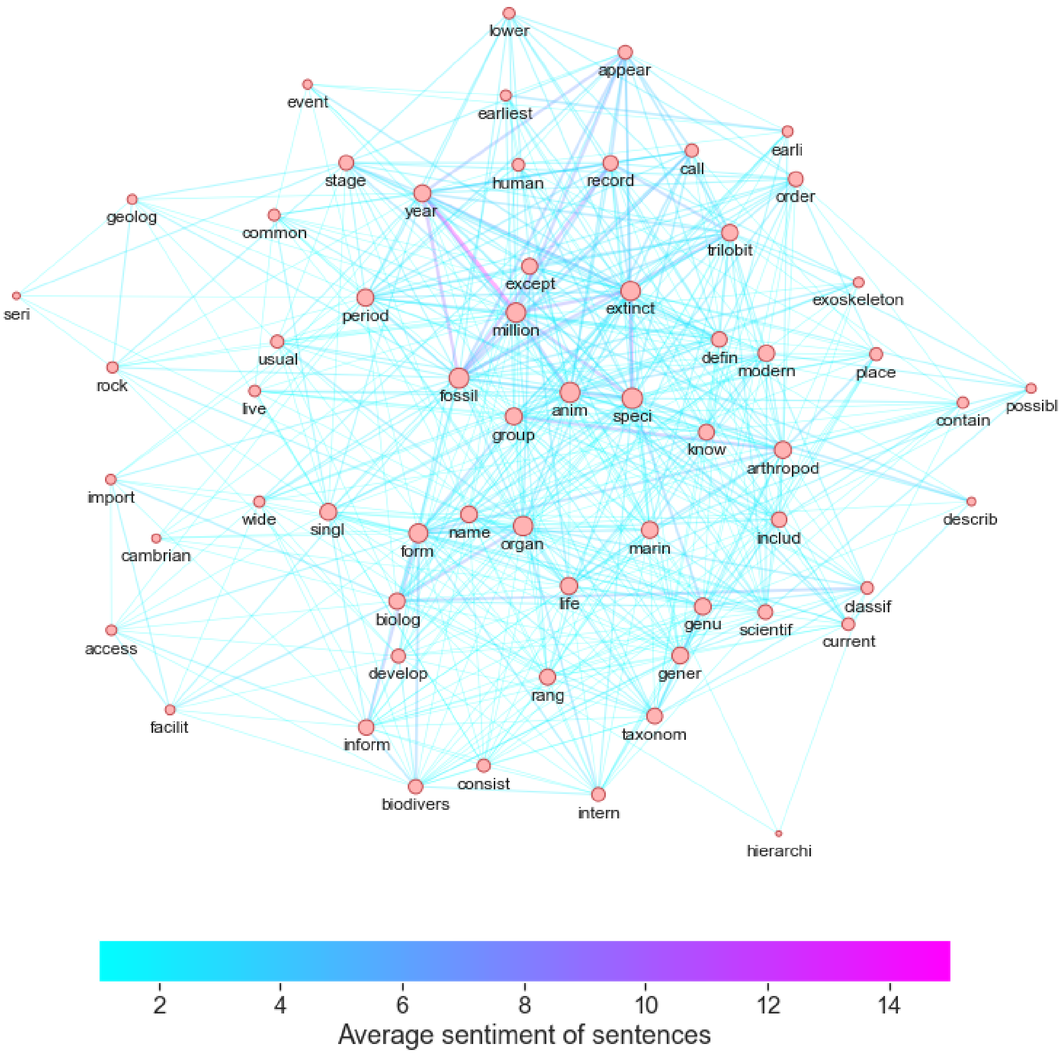
```
# def connected_component_subgraphs(G):
#      for c in nx.connected_components(G):
#          yield G.subgraph(c)

# Then, change nx.connected_component_subgraphs(Gw) to connected_component_


medgesd=[]
for e in medges:
    d={}
    d['Sentence']=e[2]
    d['Average sentiment']=e[3]
    medgesd.append((e[0],e[1],d))

G = nx.MultiGraph()
G.add_edges_from(medgesd)
for e in G.edges(data=True):
    if e[0]==e[1]:
        G.remove_edge(e[0],e[1])
weight={(x,y):v for (x, y), v in Counter(G.edges()).items()}
w_edges=[(x,y,z) for (x,y),z in weight.items()]
Gw = nx.Graph()
Gw.add_weighted_edges_from(w_edges)

titlename=ss1
print("The graph of sententially co-occurrent TM terms in the %s is a weigh
out=' '.join([n+"\n" for n in alias_dict.values() if n not in Gw.nodes()])
print("The terms which do not co-occur in sentences are: \n %s" %out)
# print "Graph Gw is a weighted graph with %i nodes and %i edges" %(len(Gw.
print("The density of this graph is %.3f" %nx.density(Gw))
if nx.is_connected(Gw)==True:
    print ("This graph is a connected graph")
else:
    print ("This graph is a disconnected graph and it has",nx.number_connec
    giant = max(nx.connected_component_subgraphs(Gw), key=len)
    Gwlcc=Gw.subgraph(giant)
    print ("The largest connected component of this graph is a weighted gra
    print ("The density of the largest connected component of this graph is
```

```
The graph of sententially co-occurrent TM terms in the 15 Wikipedia hyper
links of 'Tuberaspis' is a weighted graph and
 it has 61 nodes and 595 edges

The terms which do not co-occur in sentences are:
 famili

The density of this graph is 0.325
This graph is a connected graph
```

In [32]:

```
edge_width=[Gw[u][v]['weight'] for u,v in Gw.edges()]
edge_width=[math.log(1+w) for w in edge_width]
cmap=plt.cm.cool
weight_list = [ e[2]['weight'] for e in Gw.edges(data=True) ]
edge_color=weight_list
vmin = min(edge_color)
vmax = max(edge_color)
# width_list=[2*math.log(2+w) for w in weight_list]
width_list=[1.5*math.log(abs(min(weight_list))+2+w) for w in weight_list] #
nsi=[5*Gw.degree(n) for n in Gw.nodes()]

figsize=(15,15)

pos=graphviz_layout(Gw)

node_color="#ffb3b3"
node_border_color="r"
plt.figure(figsize=figsize);
nodes = nx.draw_networkx_nodes(Gw, pos, node_color=node_color,node_size=nsi
nodes.set_edgecolor(node_border_color)
nx.draw_networkx_edges(Gw, pos, edge_color=edge_color,edge_cmap=cmap,vmin=v
plt.axis('off');
yoffset = {}
y_off = -5 # offset on the y axis
for k, v in pos.items():
    yoffset[k] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(Gw, yoffset,font_size=12);
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=vmin, vmax=vm
sm.set_array([])
cbar = plt.colorbar(sm, orientation='horizontal', shrink=0.7, pad = 0.02)
cbar.set_label('Average sentiment of sentences')
sst="The graph of sententially co-occurrent TM terms \n in the %s \n weight
plt.title(sst,fontsize=15);
plt.margins(x=0.1, y=0.1)
```

The graph of sententially co-occurrent TM terms
in the 15 Wikipedia hyperlinks of 'Tuberaspis'
weighted over their average sentiment score



In [33]:

```
# pos['Samuel F. Billington']=(382.22, 550)
# pos['Amsterdam']=(398.54, 522)
# pos['Switzerland']=(200, 507.22)
# pos['Peter Hawkins']=(321.72, 514)
# pos['Danube']=(290, 390)
# pos['Windham']=(358.32, 270)
# pos['Mile']=(230.97, 350)
# pos['Black Sea']=(290.5, 420)
# pos['Bosphorus']=(361.57, 540)
# pos['Yorkshire']=(347.89, 500)

# node_color="#ffb3b3"
# node_border_color="r"
# plt.figure(figsize=figsize);
# nodes = nx.draw_networkx_nodes(Gw, pos, node_color=node_color,node_size=n
# nodes.set_edgecolor(node_border_color)
# nx.draw_networkx_edges(Gw, pos, edge_color=edge_color,edge_cmap=cmap,vmin
# plt.axis('off');
# yoffset = {}
# y_off = -5 # offset on the y axis
# for k, v in pos.items():
#     yoffset[k] = (v[0], v[1]+y_off)
# nx.draw_networkx_labels(Gw, yoffset,font_size=12);
# sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=vmin, vmax=
# sm.set_array([])
# cbar = plt.colorbar(sm, orientation='horizontal', shrink=0.7, pad = 0.02)
# cbar.set_label('Average sentiment of sentences')
# sst="The graph of sententially co-occurrent TM terms \n in the %s \n weig
# plt.title(sst,fontsize=15);
# plt.margins(x=0.1, y=0.1)
```

In [ ]: