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

HOMEWORK 3

Unsupervized and Supervised Topic Modeling

50 points totally

This notebook is composed of four parts (PART A, B, C, and D).

You are advised to run each Part separately until you get a satisfacory outpout from that part

and only subsequently you may proceed to the next Part.

PART A

In [1]:
```
import time
start_time = time.perf_counter()
import urllib, os, codecs, random, operator, re, string, copy, dateutil.par
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.porter import *
stemmer = PorterStemmer()
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *

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')
```

Load Data

In [2]:
```
titlename = "Bram Stoker's Dracula"

# get your working directory
home = str(pathlib.Path.cwd())

# create a path to which the file will be written
text_path = os.path.join(home, 'Dracula.txt')

# location of the project gutenberg copy of the moby-dick text file
text_url = 'http://www.gutenberg.org/cache/epub/345/pg345.txt'

urllib.request.urlretrieve(text_url, text_path)

print('Downloaded to:', text_path)
```

```
Downloaded to: /Users/mb7881/WorkPlaces/Python Projects 2/3 NYUAD Digital
Humanities/Homework3 TopicModeling & Co-OccurrentTermNets/Dracula.txt
```

In [3]:
```
f = codecs.open(text_path, "r", encoding="utf-8").readlines()
for line in f:
    if line.startswith("(_Kept in shorthand._)"):
        print(f.index(line)) #198
    if line.startswith("                                   THE END"):
        print(f.index(line)) #15514
```

```
198
15514
```

In [4]:
```
ff=f[194:15513]
ff
```

Out[4]:
```
['CHAPTER I\r\n',
 '\r\n',
 "JONATHAN HARKER'S JOURNAL\r\n",
 '\r\n',
 '(_Kept in shorthand._)\r\n',
 '\r\n',
 '\r\n',
 '_3 May. Bistritz._--Left Munich at 8:35 P. M., on 1st May, arriving at
\r\n',
 'Vienna early next morning; should have arrived at 6:46, but train was a
n\r\n',
 'hour late. Buda-Pesth seems a wonderful place, from the glimpse which I
\r\n',
 'got of it from the train and the little I could walk through the\r\n',
 'streets. I feared to go very far from the station, as we had arrived\r
\n',
 'late and would start as near the correct time as possible. The\r\n',
 'impression I had was that we were leaving the West and entering the\r
\n',
```

In [5]:
```
ff[-10:]
```

Out[5]:
```
['We could hardly ask any one, even did we wish to, to accept these as\r
\n',
 'proofs of so wild a story. Van Helsing summed it all up as he said, wit
h\r\n',
 'our boy on his knee:--\r\n',
 '\r\n',
 '"We want no proofs; we ask none to believe us! This boy will some day\r
\n',
 'know what a brave and gallant woman his mother is. Already he knows her
\r\n',
 'sweetness and loving care; later on he will understand how some men so
\r\n',
 'loved her, that they did dare much for her sake."\r\n',
 '\r\n',
 'JONATHAN HARKER.\r\n']
```

Breaking in Chapters

In [6]:
```
text=[]
text="\n".join(ff).split("CHAPTER")[1:]
print(type(text),len(text))
# print(text[0])
docs_d={}
for i,j in enumerate(text):
#     docs_d["Chapter"+str(i+1)]="CHAPTER "+j.replace("\n\n\n\n"," ").repla
    docs_d["Chapter"+str(i+1)]=j.lstrip().replace("\r","").replace("\n\n\n\
print(len(docs_d))
docs=list(docs_d.values())
print(len(docs))
chapter=8
print("This is the beginning of the %i-th document (%i-th chapter):"%(chapt
list(docs)[chapter][:1000]
```

```
<class 'list'> 27
27
27
This is the beginning of the 9-th document (9-th chapter):
```

Out[6]:
```
'IX Letter, Mina Harker to Lucy Westenra. "Buda-Pesth, 24 August. "My dea
rest Lucy, "I know you will be anxious to hear all that has happened sinc
e we parted at the railway station at Whitby. Well, my dear, I got to Hul
l all right, and caught the boat to Hamburg, and then the train on here.
I feel that I can hardly recall anything of the journey, except that I kn
ew I was coming to Jonathan, and, that as I should have to do some nursin
g, I had better get all the sleep I could.... I found my dear one, oh, so
thin and pale and weak-looking. All the resolution has gone out of his de
ar eyes, and that quiet dignity which I told you was in his face has vani
shed. He is only a wreck of himself, and he does not remember anything th
at has happened to him for a long time past. At least, he wants me to bel
ieve so, and I shall never ask. He has had some terrible shock, and I fea
r it might tax his poor brain if he were to try to recall it. Sister Agat
ha, who is a good creature and a born nurse, tells '
```

1. Unsupervized Topic Modeling

In [7]:
```
sorted(stopwords.words('english'))
```

Out[7]:
```
['a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
```

In [8]:
```
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()]

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(lemmatize_stemming(token))
    return result
```

In [9]:
```
vocabulary = set()
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)
```

```
10274 27
```

In [10]:
```
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]
```

```
6326
```

In [11]:
```
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
```

```
54368 6326
6326
```

In [12]:
```
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
```

```
6326
0 abl
1 abreast
2 absenc
3 accustom
4 add
5 affect
6 afield
7 afraid
8 afternoon
9 agonis
10 ahead
```

In [13]:
```
dictionary.filter_extremes(no_below=5) #, no_above=0.8) #, keep_n=300) # 5
len(dictionary)
```

Out[13]:
```
1004
```

In [14]:
```
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
# bow_corpus[43]
```

Detecting "Optimal" Number of Topics

The minimum number of topics below (mint) should be at least 2 or 3.

In [15]:
```
mint = 3  # minimum number of topics
maxt = 11 # maximum number of topics
m=30
X = range(mint,maxt)
Y = []
for n in X:
    ft=[]
    for j in range(m):
        lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=n, id
    #     topics = lda_model.print_topics()
        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')
        doms=[]
        for i in sss:
            doms.append(i[0][0])
        fi=len(set(doms))/n
        ft.append(fi)
    fis=sum(ft)/m
    Y.append(fis)
# print(list(X))
print(Y)
nn=[]
for i,y in enumerate(Y):
    if y==max(Y): #1:
        print(Y.index(y))
        nn.append(i)
print(nn)
NT=nn[-1]+mint
print(NT)
```

```
[0.9666666666666667, 0.95, 0.9333333333333335, 0.8833333333333332, 0.8761
904761904765, 0.8791666666666667, 0.8555555555555558, 0.8]
0
[0]
3
```

In [16]:
```
nt=NT #number_of_topics
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=nt, id2word=d
```

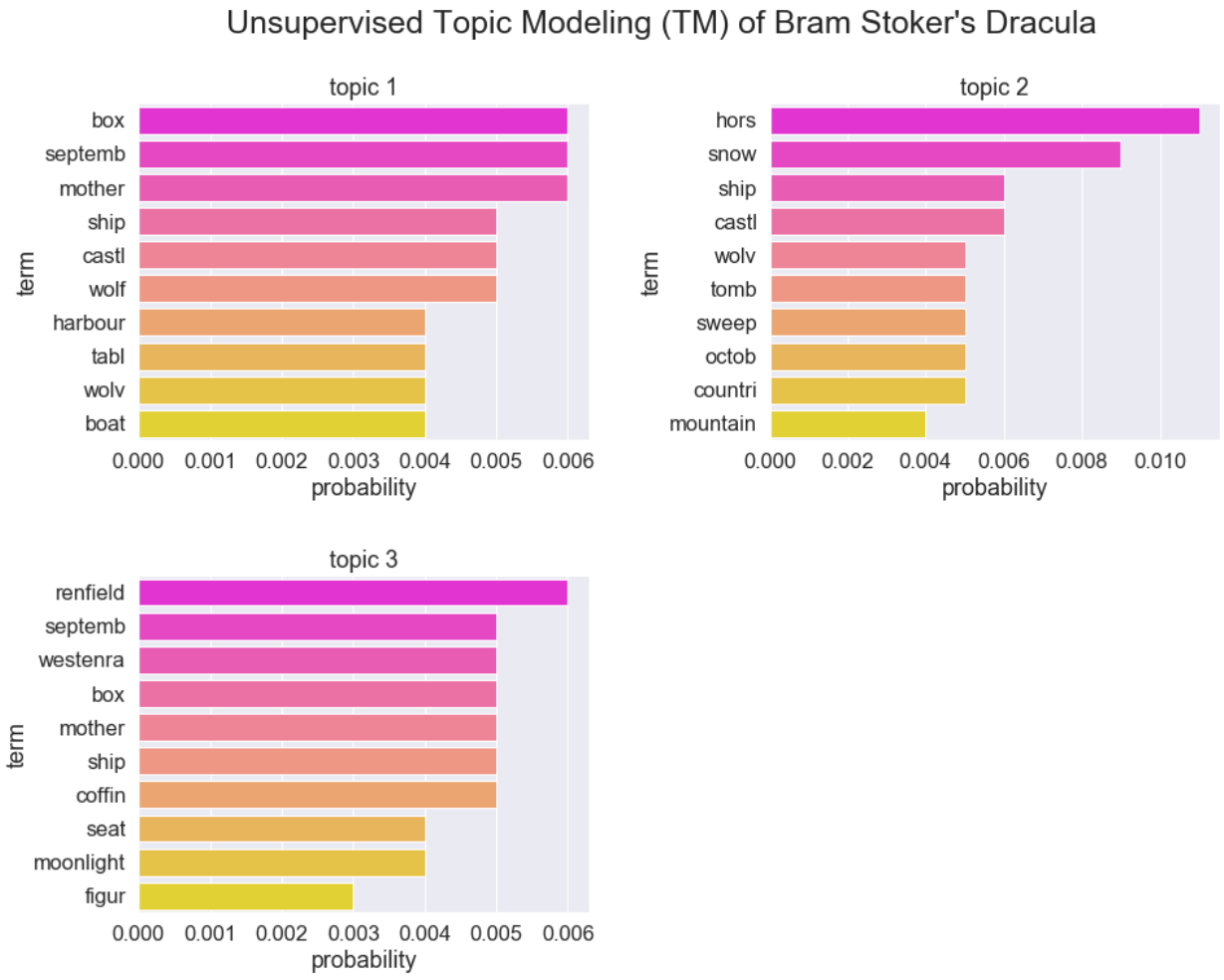
In [17]:
```
topics = lda_model.print_topics() #350  #num_words=25

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 UNSUPERVISED TM TERMS:")
print(" ")
for i in terms:
    print(i)
```

```
23

LIST OF UNSUPERVISED TM TERMS:

boat
box
castl
coffin
countri
figur
harbour
hors
moonlight
mother
mountain
octob
renfield
seat
septemb
ship
snow
sweep
tabl
tomb
westenra
wolf
wolv
```

In [18]:
```
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'0.006*"box" + 0.006*"septemb" + 0.006*"mother" + 0.005*"ship" +
0.005*"castl" + 0.005*"wolf" + 0.004*"harbour" + 0.004*"tabl" + 0.004*"wo
lv" + 0.004*"boat"'
Topic: 1
Words: b'0.011*"hors" + 0.009*"snow" + 0.006*"ship" + 0.006*"castl" + 0.0
05*"wolv" + 0.005*"tomb" + 0.005*"sweep" + 0.005*"octob" + 0.005*"countr
i" + 0.004*"mountain"'
Topic: 2
Words: b'0.006*"renfield" + 0.005*"septemb" + 0.005*"westenra" + 0.005*"b
ox" + 0.005*"mother" + 0.005*"ship" + 0.005*"coffin" + 0.004*"seat" + 0.0
04*"moonlight" + 0.003*"figur"'
```

In [19]:
```
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');
sst="Unsupervised Topic Modeling (TM) of %s" %titlename
plt.suptitle(sst,fontsize=25, y=0.92);
plt.show()
```



In [20]:
```
from pyLDAvis import gensim as pgensim
vis = pgensim.prepare(lda_model,bow_corpus, dictionary)
vis
```

Out[20]:

PART B

The Network of Sententially Co-Occurrent Terms Derived from Unsupervised Topic Modeling

In [21]:
```
terms
```

Out[21]:
```
['boat',
 'box',
 'castl',
 'coffin',
 'countri',
 'figur',
 'harbour',
 'hors',
 'moonlight',
 'mother',
 'mountain',
 'octob',
 'renfield',
 'seat',
 'septemb',
 'ship',
 'snow',
 'sweep',
 'tabl',
```

In [22]:
```
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)
```

In [23]:
```
# # When no alias dictionary is needed

alias_dict={}
for n in terms:
    alias_dict[n]=n

# # For the case you need to use an alias dictionary:

# alias_dict={}
# for n in terms:
#     if n=="paper":
#         alias_dict[n]="paper"
#     elif n=="papers":
#         alias_dict[n]="paper"
#     elif n=="steps":
#         alias_dict[n]="steps"
#     elif n=="stepped":
#         alias_dict[n]="steps"
#     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)
```

```
The dictionary of aliases has 23 keys (terms) and 23 unique values (alias
ed terms)
```

In [24]:
```
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("%s contains %i sentential co-occurrences among %i unsupervized TM te
medges
```

```
Bram Stoker's Dracula contains 97 sentential co-occurrences among 23 unsu
pervized TM terms
```

Out[24]:
```
[('seat', 'box', 'Sentence_103', -0.0375),
 ('hors', 'box', 'Sentence_103', -0.0375),
 ('hors', 'seat', 'Sentence_103', -0.0375),
 ('snow', 'mountain', 'Sentence_116', -0.2142857142857143),
 ('snow', 'mountain', 'Sentence_121', -0.01666666666666673),
 ('hors', 'figur', 'Sentence_171', 0.03333333333333335),
 ('hors', 'wolv', 'Sentence_187', 0.06875),
 ('hors', 'mountain', 'Sentence_187', 0.06875),
 ('wolv', 'mountain', 'Sentence_187', 0.06875),
 ('seat', 'wolv', 'Sentence_201', -0.08333333333333333),
 ('moonlight', 'wolv', 'Sentence_213', 0.0),
 ('hors', 'castl', 'Sentence_224', -0.011111111111111108),
 ('hors', 'seat', 'Sentence_233', -0.1527777777777778),
 ('seat', 'tabl', 'Sentence_278', 0.8),
 ('wolv', 'castl', 'Sentence_328', 0.25),
 ('figur', 'tabl', 'Sentence_388', 0.0),
 ('castl', 'tabl', 'Sentence_484', 0.0),
 ('ship', 'castl', 'Sentence_555', 0.35),
 ('sweep', 'wolv', 'Sentence_707', 0.0),
 ('tomb', 'mother', 'Sentence_1343', 0.7),
 ('sweep', 'harbour', 'Sentence_1509', 0.0),
 ('harbour', 'boat', 'Sentence_1509', 0.0),
```

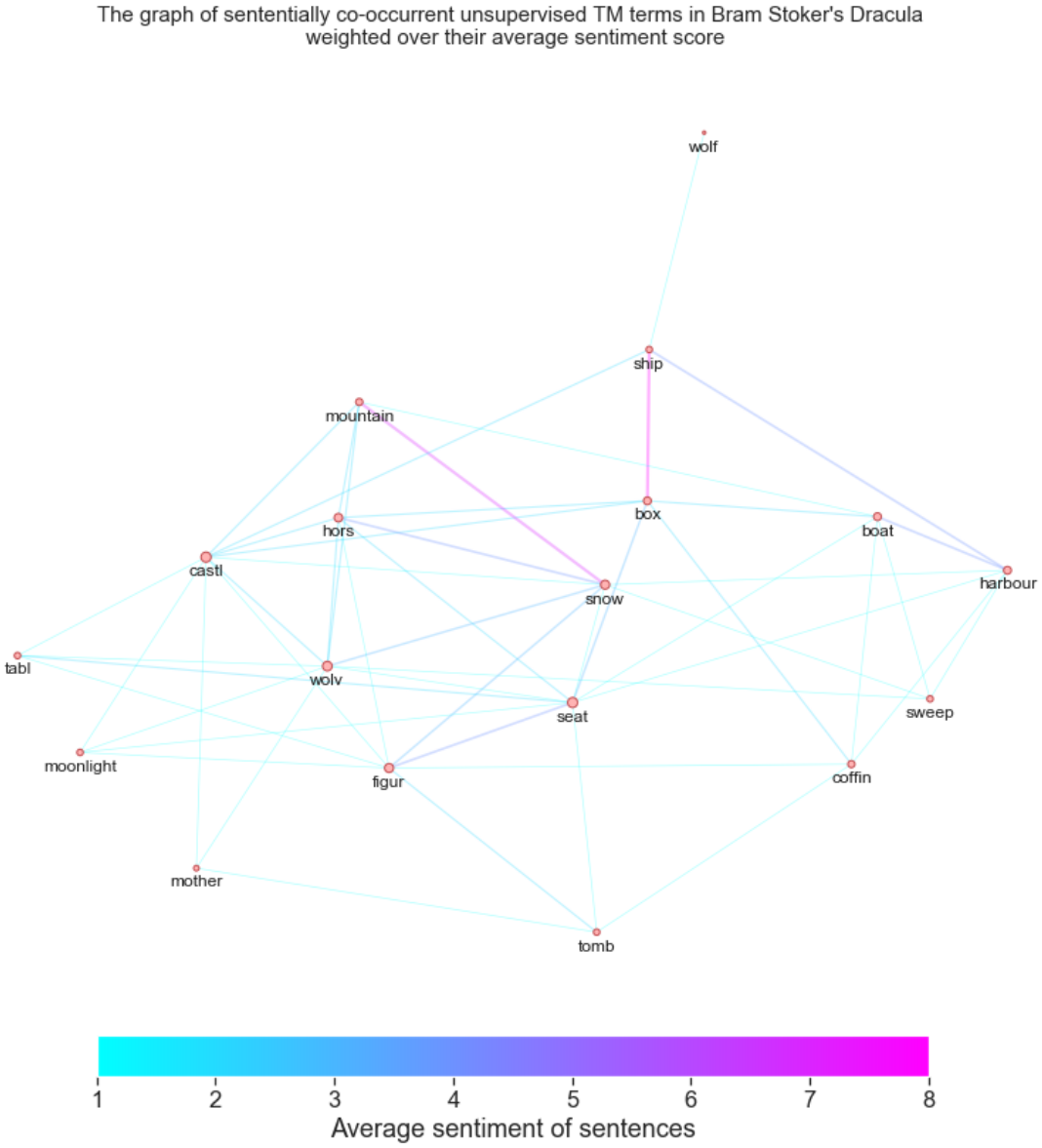
```
('sweep', 'boat', 'Sentence_1509', 0.0),
('harbour', 'boat', 'Sentence_1557', 0.25),
('boat', 'mountain', 'Sentence_1573', 0.022500000000000003),
('harbour', 'boat', 'Sentence_1576', 0.5),
('harbour', 'ship', 'Sentence_1580', 0.21805555555555553),
('harbour', 'ship', 'Sentence_1587', 0.275),
('harbour', 'ship', 'Sentence_1613', 0.0),
('harbour', 'ship', 'Sentence_1618', 0.0),
('snow', 'harbour', 'Sentence_1783', 0.02375),
('harbour', 'coffin', 'Sentence_1790', 0.0),
('harbour', 'boat', 'Sentence_1790', 0.0),
('coffin', 'boat', 'Sentence_1790', 0.0),
('seat', 'boat', 'Sentence_1791', 0.014814814814814808),
('seat', 'tomb', 'Sentence_1809', -0.14814814814814814),
('seat', 'harbour', 'Sentence_1866', 0.0),
('snow', 'figur', 'Sentence_1870', 0.25833333333333336),
('seat', 'snow', 'Sentence_1870', 0.25833333333333336),
('seat', 'figur', 'Sentence_1870', 0.25833333333333336),
('seat', 'figur', 'Sentence_1871', 0.032539682539682535),
('seat', 'figur', 'Sentence_1876', 0.16666666666666666),
('moonlight', 'seat', 'Sentence_1883', 0.24444444444444446),
('seat', 'figur', 'Sentence_1950', 0.22499999999999998),
('hors', 'wolv', 'Sentence_3055', -0.25),
('seat', 'tabl', 'Sentence_3804', 0.13333333333333333),
('tomb', 'figur', 'Sentence_4420', 0.0031250000000000028),
('tomb', 'figur', 'Sentence_4421', -0.16666666666666666),
('tomb', 'coffin', 'Sentence_4450', -0.11111111111111109),
('coffin', 'figur', 'Sentence_4499', 0.2692307692307693),
('moonlight', 'figur', 'Sentence_4712', -0.5),
('wolf', 'ship', 'Sentence_5349', 0.3),
('moonlight', 'castl', 'Sentence_5351', 0.0),
('box', 'castl', 'Sentence_5384', -0.3),
('seat', 'box', 'Sentence_5400', 0.15),
('wolv', 'tabl', 'Sentence_5531', 0.425),
('wolv', 'castl', 'Sentence_5622', -0.0375),
('mother', 'castl', 'Sentence_5622', -0.0375),
('wolv', 'mother', 'Sentence_5622', -0.0375),
('seat', 'box', 'Sentence_5790', 0.2333333333333333),
('hors', 'box', 'Sentence_5845', 0.19166666666666665),
('box', 'ship', 'Sentence_7047', 0.0),
('box', 'ship', 'Sentence_7077', 0.1),
('box', 'ship', 'Sentence_7139', 0.21481481481481482),
('box', 'ship', 'Sentence_7381', 0.31666666666666665),
('box', 'ship', 'Sentence_7533', 0.14444444444444446),
('box', 'ship', 'Sentence_7657', 0.45),
('box', 'coffin', 'Sentence_7658', 0.25),
('box', 'ship', 'Sentence_7879', 0.0),
('snow', 'wolv', 'Sentence_7933', -0.25),
('box', 'castl', 'Sentence_7958', 0.0),
('box', 'ship', 'Sentence_7958', 0.0),
('ship', 'castl', 'Sentence_7958', 0.0),
('box', 'boat', 'Sentence_7964', 0.05),
('box', 'coffin', 'Sentence_8005', 0.1),
('snow', 'wolv', 'Sentence_8044', 0.0),
('box', 'boat', 'Sentence_8072', -0.06666666666666667),
('hors', 'snow', 'Sentence_8100', -0.14675925925925926),
('hors', 'mountain', 'Sentence_8100', -0.14675925925925926),
```

```
('snow', 'mountain', 'Sentence_8100', -0.14675925925925926),
('hors', 'snow', 'Sentence_8207', 0.1238888888888889),
('snow', 'mountain', 'Sentence_8217', 0.18333333333333335),
('castl', 'mountain', 'Sentence_8250', 0.15),
('hors', 'snow', 'Sentence_8254', 0.4833333333333334),
('snow', 'sweep', 'Sentence_8275', -0.6),
('snow', 'figur', 'Sentence_8295', 0.0),
('hors', 'snow', 'Sentence_8309', 0.1),
('snow', 'figur', 'Sentence_8313', 0.058333333333333334),
('snow', 'castl', 'Sentence_8313', 0.058333333333333334),
('castl', 'figur', 'Sentence_8313', 0.058333333333333334),
('snow', 'wolv', 'Sentence_8338', -0.02777777777777779),
('snow', 'mountain', 'Sentence_8338', -0.02777777777777779),
('wolv', 'mountain', 'Sentence_8338', -0.02777777777777779),
('castl', 'mountain', 'Sentence_8403', 0.05),
('snow', 'mountain', 'Sentence_8460', 0.17430555555555557),
('hors', 'castl', 'Sentence_8484', 0.14259259259259258),
('snow', 'mountain', 'Sentence_8499', -0.001851851851851859)]
```

In [25]:
```
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)

print("The graph of sententially co-occurrent unsupervised TM terms in %s i
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 unsupervised TM terms in Bram Stok
er's Dracula is a weighted graph and
 it has 18 nodes and 52 edges

The terms which do not co-occur in sentences are:
 countri
 octob
 renfield
 septemb
 westenra

The density of this graph is 0.340
This graph is a connected graph
```

In [26]:
```
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)
# pos=nx.spring_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 unsupervised TM terms in %s \n
plt.title(sst,fontsize=15);
plt.margins(x=0.1, y=0.1)
```



PART C

2. Supervized Topic Modeling

Extraction of Relevant NLP Entities

List of NLP Entities

In [27]:
```
# TYPE          DESCRIPTION
# PERSON        People, including fictional.
# NORP          Nationalities or religious or political groups.
# FAC           Buildings, airports, highways, bridges, etc.
# ORG           Companies, agencies, institutions, etc.
# GPE           Countries, cities, states.
# LOC           Non-GPE locations, mountain ranges, bodies of water.
# PRODUCT       Objects, vehicles, foods, etc. (Not services.)
# EVENT         Named hurricanes, battles, wars, sports events, etc.
# WORK_OF_ART   Titles of books, songs, etc.
# LAW           Named documents made into laws.
# LANGUAGE      Any named language.
# DATE          Absolute or relative dates or periods.
# TIME          Times smaller than a day.
# PERCENT       Percentage, including ”%“.
# MONEY         Monetary values, including unit.
# QUANTITY      Measurements, as of weight or distance.
# ORDINAL       “first”, “second”, etc.
# CARDINAL      Numerals that do not fall under another type.
```

In [28]:
```
p = inflect.engine()
d_tags = {}

for key, value in docs_d.items():
    arr = []
    doc = nlp(value.replace('\n',''))
    #Keep these types of nlp entities
    keep_l = ['PERSON','NORP','GPE','LOC','EVENT']
    #Typo/model error + german corrections
    drop_t = []

    #Things inflect library handles poorly or to exclude from touching
    ex_ls = []

    for X in doc.ents:
        s1 = X.text
        if (X.label_ in keep_l) and (s1.lower() not in drop_t) and (s1):
            arr.append((s1, X.label_))
    d_tags[key] = arr
# pprint(d_tags)
names=[]
for k,v in d_tags.items():
    for vv in v:
        if vv[0] not in names:
            p=vv[0].replace("'","")
            p=p.title()
            names.append(p)
tdocs=" ".join(docs)
names=[n for n in names if n in tdocs]
names=sorted(set(names))
print(len(names))
names
```

```
365
```

In [29]:
```
rem=[]
# for p in names:
#     if "_" in p:
#         rem.append(p)
#     if "--" in p:
#         rem.append(p)
# #     if p not in text:
# #         rem.append(p)
# names=[p for p in names if p not in rem]
pp=[q for q in itertools.product(names,names) if q[0]!=q[1]]
for q in pp:
    if q[0] in q[1]:
        rem.append(q[0])
    if q[1] in q[0]:
        rem.append(q[1])
    w=q[0]+" "+q[1]
    if w in text:
        names.append(w)
        rem.append(q[0])
        rem.append(q[1])
names=[p for p in names if p not in rem]
names=sorted(set(names))
print(len(names))
names
```

```
294
```

In [30]:
```
rem=["Nay","Pass","Ye","Lordship","Friend Arthur","Friend John","Robin","Si
names=[p for p in names if p not in rem]
names=names+['Robin Hood','Soho','Braithwaite Lowrey',
             'Mitchell, Sons, & Candy','Jonathan','Mina','Sister Agatha',
             'Count Dracula','Ste. Mary','St. Joseph',"Saxon"]
names=sorted(set(names))
print(len(names))
names
```

```
295
```

In [31]:
```
terms_list=[]
for doc in docs:
    t=[]
    for i in names:
        if i in doc:
            m=doc.count(i)
            t.append(m*[i])
    flatten = sum(t, [])
    terms_list.append(flatten)
print(len(terms_list))
terms_list
```

```
27
```

In [32]:
```
dictionary = gensim.corpora.Dictionary(terms_list)
print(len(dictionary)) #All 32368
count = 0
for k, v in dictionary.iteritems():
    print(k, v)
    count += 1
    if count > 10:
        break
```

```
295
0 Attila
1 Borgo Prund
2 British
3 Buda-Pesth
4 Bukovina
5 Carpathians
6 Castle Dracula
7 China
8 Count Dracula
9 Cszeks
10 Dacians
```

In [33]:
```
# dictionary.filter_extremes(no_below=1, no_above=0.7, keep_n=300)
# len(dictionary)
```

In [34]:
```
bow_corpus = [dictionary.doc2bow(doc) for doc in terms_list]
# bow_corpus[43]
```

The minimum number of topics below (mint) should be at least 2 or 3.

In [35]:
```
mint = 2  # minimum number of topics
maxt = 11 # maximum number of topics
m=30
X = range(mint,maxt)
Y = []
for n in X:
    ft=[]
    for j in range(m):
        lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=n, id
    #     topics = lda_model.print_topics()
        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')
        doms=[]
        for i in sss:
            doms.append(i[0][0])
        fi=len(set(doms))/n
        ft.append(fi)
    fis=sum(ft)/m
    Y.append(fis)
# print(list(X))
print(Y)
nn=[]
for i,y in enumerate(Y):
    if y==max(Y): #1:
        print(Y.index(y))
        nn.append(i)
print(nn)
NT=nn[-1]+mint
print(NT)
```

```
[0.9166666666666666, 0.8333333333333335, 0.7166666666666667, 0.7466666666
666669, 0.7, 0.7000000000000001, 0.6458333333333334, 0.6333333333333334,
0.5933333333333333]
0
[0]
2
```

In [36]:
```
nt=NT #number_of_topics
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=nt, id2word=d
```

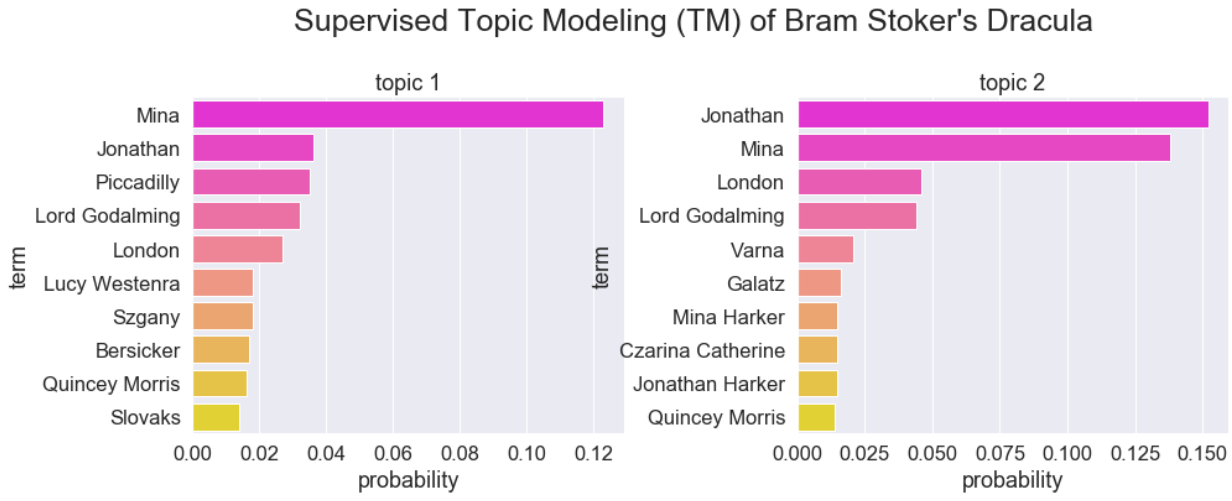
In [37]:
```
topics = lda_model.print_topics() #350 #num_words=100

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

```
15

LIST OF SUPERVISED TM TERMS:

Bersicker
Czarina Catherine
Galatz
Jonathan
Jonathan Harker
London
Lord Godalming
Lucy Westenra
Mina
Mina Harker
Piccadilly
Quincey Morris
Slovaks
Szgany
Varna
```

In [38]:
```
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'0.123*"Mina" + 0.036*"Jonathan" + 0.035*"Piccadilly" + 0.032*"Lo
rd Godalming" + 0.027*"London" + 0.018*"Lucy Westenra" + 0.018*"Szgany" +
0.017*"Bersicker" + 0.016*"Quincey Morris" + 0.014*"Slovaks"'
Topic: 1
Words: b'0.152*"Jonathan" + 0.138*"Mina" + 0.046*"London" + 0.044*"Lord G
odalming" + 0.021*"Varna" + 0.016*"Galatz" + 0.015*"Mina Harker" + 0.015
*"Czarina Catherine" + 0.015*"Jonathan Harker" + 0.014*"Quincey Morris"'
```

In [39]:
```
fig=plt.figure(figsize=(15,25)) #figsize=(15,2.4*15*((nt+1)/4))); #15
fig.subplots_adjust(hspace=1, 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')
    ax = fig.add_subplot(nt+1,2,i+1)
#     plt.subplot(nt+1,2,i+1); #5
    plt.title('topic '+str(i+1));
    sns.barplot(x='prob', y=df.index, data=df, label='Cities', palette='spr
    plt.xlabel('probability');
sst="Supervised Topic Modeling (TM) of %s" %titlename
plt.suptitle(sst,fontsize=25, y=0.92);
plt.show()
```



In [40]:
```
from pyLDAvis import gensim as pgensim
vis = pgensim.prepare(lda_model,bow_corpus, dictionary)
vis
```

Out[40]:

PART D

The Network of Sententially Co-Occurrent Terms Derived from Supervised Topic Modeling

In [41]:
```
sterms
```

Out[41]:
```
['Bersicker',
 'Czarina Catherine',
 'Galatz',
 'Jonathan',
 'Jonathan Harker',
 'London',
 'Lord Godalming',
 'Lucy Westenra',
 'Mina',
 'Mina Harker',
 'Piccadilly',
 'Quincey Morris',
 'Slovaks',
 'Szgany',
 'Varna']
```

In [42]:
```
pre=[]
for i in range(len(sterms)):
    start=sterms[i][:4]
    pre.append(start)
for j,k in Counter(pre).items():
    if k>1:
        print(j)
```

```
Jona
Mina
```

In [43]:
```
salias_dict={}
for n in sterms:
    if n=="Mina":
        salias_dict[n]="Mina"
    elif n=="Mina Harker":
        salias_dict[n]="Mina"
    elif n=="Jonathan":
        salias_dict[n]="Jonathan"
    elif n=="Jonathan Harker":
        salias_dict[n]="Jonathan"
#     elif n=="Mina Murray":
#         salias_dict[n]="Mina"
#     elif n=="English":
#         salias_dict[n]="England"
#     elif n=="England":
#         salias_dict[n]="England"
#     elif n=="Mitchell, Sons, & Candy":
#         salias_dict[n]="Mitchell"
#     elif n=="Mitchell":
#         salias_dict[n]="Mitchell"
#     elif n=="Quincey Morris":
#         salias_dict[n]="Quincey Morris"
#     elif n=="Quincey P. Morris":
#         salias_dict[n]="Quincey Morris"
#     elif n=="Saxon":
#         salias_dict[n]="Saxon"
#     elif n=="Saxons":
#         salias_dict[n]="Saxon"
#     elif n=="Turkey":
#         salias_dict[n]="Turkey"
#     elif n=="Turkish":
#         salias_dict[n]="Turkey"
#     elif n=="Agatha":
#         salias_dict[n]="Sister Agatha"
#     elif n=="Sister Agatha":
#         salias_dict[n]="Sister Agatha"
    else:
        salias_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)
```

```
The dictionary of aliases has 15 keys (terms) and 13 unique values (alias
ed terms)
```

In [44]:
```
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(salias_dict.values())):
        if term in v:
            wl.append(term)
    if len(wl)>1:
        sendic[i]=wl
smedges=[]
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 salias_dict[jj[0]]!=salias_dict[jj[1]]:
            smedges.append((salias_dict[jj[0]],salias_dict[jj[1]],"Sentence
print("%s contains %i sentential co-occurrences among %i supervized TM term
smedges
```

```
Bram Stoker's Dracula contains 69 sentential co-occurrences among 13 supe
rvized TM terms
```

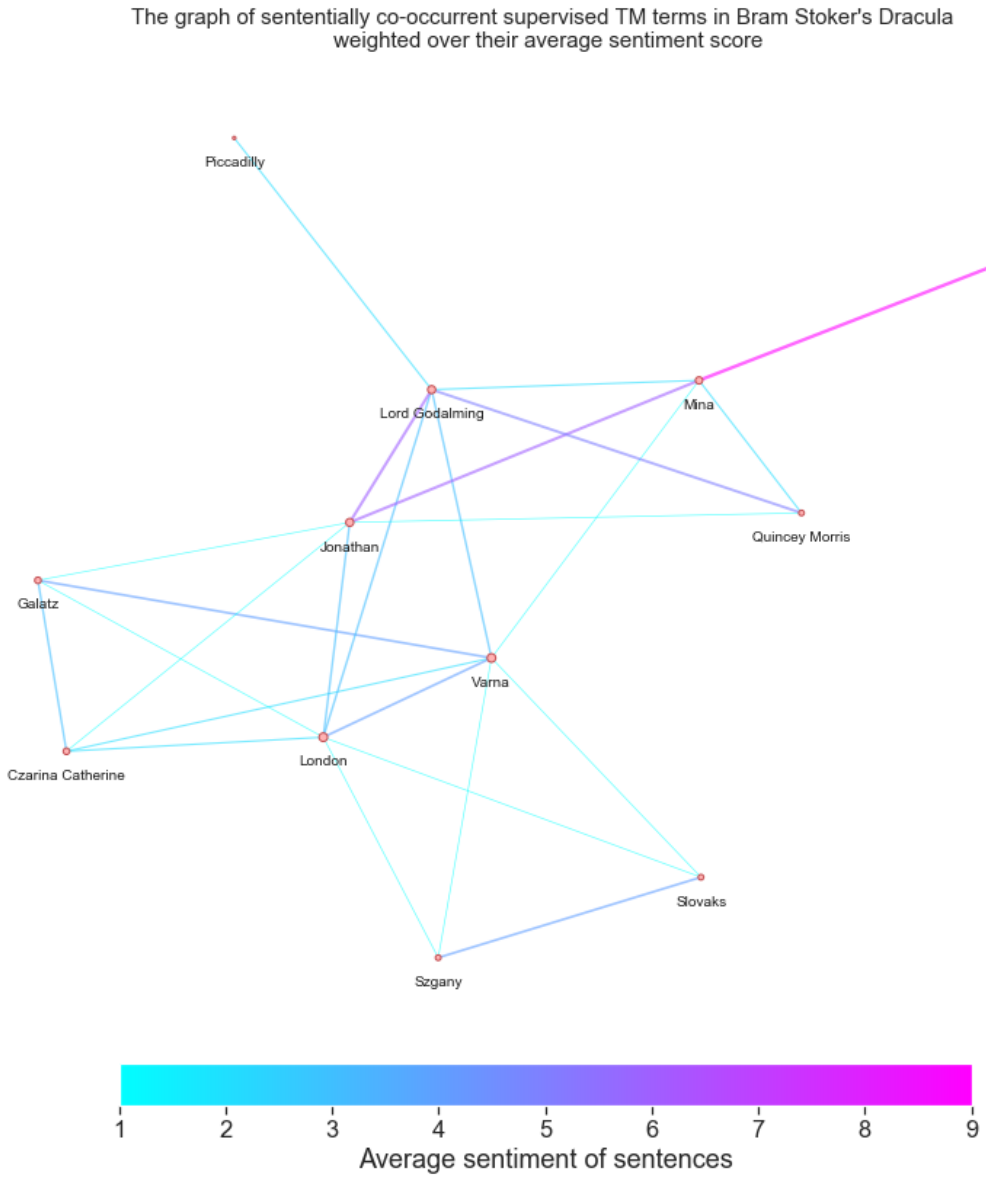
Out[44]:
```
[('Varna', 'London', 'Sentence_585', 0.0),
 ('Slovaks', 'Szgany', 'Sentence_811', 0.275),
 ('Slovaks', 'Szgany', 'Sentence_922', 0.6),
 ('Slovaks', 'Szgany', 'Sentence_1011', -0.2),
 ('Lucy Westenra', 'Mina', 'Sentence_1039', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_1066', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_1112', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_2153', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_2210', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_2279', 0.07222222222222223),
 ('Lucy Westenra', 'Mina', 'Sentence_3389', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_3457', 0.0),
 ('Lucy Westenra', 'Mina', 'Sentence_4013', -0.4),
 ('Jonathan', 'London', 'Sentence_5010', 0.0),
 ('Jonathan', 'Lord Godalming', 'Sentence_5320', 0.10428571428571427),
 ('Quincey Morris', 'Lord Godalming', 'Sentence_5438', 0.2047619047619047
8),
 ('Jonathan', 'London', 'Sentence_5719', -1.0),
 ('Jonathan', 'London', 'Sentence_6566', 0.09428571428571428),
 ('Piccadilly', 'Lord Godalming', 'Sentence_6593', 0.0),
 ('Piccadilly', 'Lord Godalming', 'Sentence_6679', 0.0),
```

```
 ('Quincey Morris', 'Lord Godalming', 'Sentence_6710', 0.0),
 ('Quincey Morris', 'Lord Godalming', 'Sentence_6728', 0.2000000000000000
4),
 ('Quincey Morris', 'Lord Godalming', 'Sentence_6805', 0.0),
 ('Jonathan', 'Mina', 'Sentence_6981', 0.05),
 ('Quincey Morris', 'Lord Godalming', 'Sentence_7117', 0.0),
 ('Quincey Morris', 'Mina', 'Sentence_7117', 0.0),
 ('Mina', 'Lord Godalming', 'Sentence_7117', 0.0),
 ('Jonathan', 'Mina', 'Sentence_7117', 0.0),
 ('Quincey Morris', 'Jonathan', 'Sentence_7117', 0.0),
 ('Jonathan', 'Lord Godalming', 'Sentence_7117', 0.0),
 ('Czarina Catherine', 'Varna', 'Sentence_7124', -0.125),
 ('Jonathan', 'Mina', 'Sentence_7316', 0.0),
 ('Varna', 'Mina', 'Sentence_7374', 0.0),
 ('Czarina Catherine', 'Varna', 'Sentence_7512', 0.25),
 ('Varna', 'Lord Godalming', 'Sentence_7549', 0.0),
 ('Varna', 'London', 'Sentence_7549', 0.0),
 ('London', 'Lord Godalming', 'Sentence_7549', 0.0),
 ('Czarina Catherine', 'London', 'Sentence_7568', 0.0),
 ('Varna', 'Lord Godalming', 'Sentence_7599', 0.0),
 ('Varna', 'London', 'Sentence_7599', 0.0),
 ('London', 'Lord Godalming', 'Sentence_7599', 0.0),
 ('Czarina Catherine', 'Galatz', 'Sentence_7600', 0.0),
 ('Galatz', 'Jonathan', 'Sentence_7636', 0.0),
 ('Galatz', 'Varna', 'Sentence_7742', 0.0),
 ('Galatz', 'Varna', 'Sentence_7755', 0.0),
 ('Varna', 'Lord Godalming', 'Sentence_7832', -0.2416666666666667),
 ('Czarina Catherine', 'Jonathan', 'Sentence_7833', 0.0),
 ('London', 'Lord Godalming', 'Sentence_7839', 0.0),
 ('Galatz', 'Varna', 'Sentence_7862', 0.15000000000000002),
 ('Czarina Catherine', 'London', 'Sentence_7876', 0.0),
 ('Czarina Catherine', 'Galatz', 'Sentence_7876', 0.0),
 ('Galatz', 'London', 'Sentence_7876', 0.0),
 ('Galatz', 'Varna', 'Sentence_7940', 0.16666666666666666),
 ('Czarina Catherine', 'Galatz', 'Sentence_7945', 0.05833333333333332),
 ('Szgany', 'Varna', 'Sentence_7958', 0.0),
 ('Slovaks', 'Szgany', 'Sentence_7958', 0.0),
 ('Slovaks', 'Varna', 'Sentence_7958', 0.0),
 ('Slovaks', 'London', 'Sentence_7958', 0.0),
 ('Szgany', 'London', 'Sentence_7958', 0.0),
 ('Varna', 'London', 'Sentence_7958', 0.0),
 ('Mina', 'Lord Godalming', 'Sentence_8004', 0.04910714285714285),
 ('Jonathan', 'Lord Godalming', 'Sentence_8004', 0.04910714285714285),
 ('Jonathan', 'Mina', 'Sentence_8004', 0.04910714285714285),
 ('Jonathan', 'Mina', 'Sentence_8008', -0.06919642857142858),
 ('Jonathan', 'Lord Godalming', 'Sentence_8034', 0.35),
 ('Jonathan', 'Mina', 'Sentence_8298', 0.21249999999999997),
 ('Jonathan', 'Lord Godalming', 'Sentence_8453', -0.125),
 ('Jonathan', 'Lord Godalming', 'Sentence_8477', -0.0625),
 ('Quincey Morris', 'Mina', 'Sentence_8528', 0.4)]
```

In [45]:
```
smedgesd=[]
for e in smedges:
    d={}
    d['Sentence']=e[2]
    d['Average sentiment']=e[3]
    smedgesd.append((e[0],e[1],d))

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

print("The graph of sententially co-occurrent supervised TM terms in %s is
out=' '.join([n+"\n" for n in salias_dict.values() if n not in sGw.nodes()]
print("The terms which do not co-occur in sentences are: \n %s" %out)
# print "Graph sGw is a weighted graph with %i nodes and %i edges" %(len(sG
print("The density of this graph is %.3f" %nx.density(sGw))
if nx.is_connected(sGw)==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(sGw), key=len)
    sGwlcc=sGw.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 supervised TM terms in Bram Stoke
r's Dracula is a weighted graph and
 it has 12 nodes and 25 edges

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

The density of this graph is 0.379
This graph is a connected graph
```

In [46]:
```
edge_width=[sGw[u][v]['weight'] for u,v in sGw.edges()]
edge_width=[math.log(1+w) for w in edge_width]
cmap=plt.cm.cool
weight_list = [ e[2]['weight'] for e in sGw.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*sGw.degree(n) for n in sGw.nodes()]

figsize=(15,15)

pos=graphviz_layout(sGw)
# pos=nx.spring_layout(sGw)

node_color="#ffb3b3"
node_border_color="r"
plt.figure(figsize=figsize);
nodes = nx.draw_networkx_nodes(sGw, pos, node_color=node_color,node_size=ns
nodes.set_edgecolor(node_border_color)
nx.draw_networkx_edges(sGw, pos, edge_color=edge_color,edge_cmap=cmap,vmin=
plt.axis('off');
yoffset = {}
y_off = -7 # offset on the y axis
for k, v in pos.items():
    yoffset[k] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(sGw, yoffset,font_size=10);
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 supervised TM terms in %s \n we
plt.title(sst,fontsize=15);
plt.margins(x=0.1, y=0.1)
```



In [47]:
```
# # Run this is the graph is NOT connected

# sorted(sGwlcc.nodes())
```

In [48]:
```
# # Run this is the graph is NOT connected

# edge_width=[sGwlcc[u][v]['weight'] for u,v in sGwlcc.edges()]
# edge_width=[math.log(1+w) for w in edge_width]
# cmap=plt.cm.cool
# weight_list = [ e[2]['weight'] for e in sGwlcc.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*sGwlcc.degree(n) for n in sGwlcc.nodes()]

# figsize=(15,15)

# pos=graphviz_layout(sGwlcc)
# # pos=nx.spring_layout(sGw)

# # 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(sGwlcc, pos, node_color=node_color,node_si
# nodes.set_edgecolor(node_border_color)
# nx.draw_networkx_edges(sGwlcc, pos, edge_color=edge_color,edge_cmap=cmap,
# plt.axis('off');
# yoffset = {}
# y_off = -7 # offset on the y axis
# for k, v in pos.items():
#     yoffset[k] = (v[0], v[1]+y_off)
# nx.draw_networkx_labels(sGwlcc, 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 largest connected component of the \n graph of sententially co-o
# plt.title(sst,fontsize=15);
# plt.margins(x=0.1, y=0.1)
```

In [49]:
```
print("Run in %.2f seconds (%.2f minutes)" %(time.clock() - start_time,(tim
```

```
Run in 118.95 seconds (1.98 minutes)
```