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

HOMEWORK 10

Reddit mining and analysis

50 points totally

In [ ]:
```
import praw  #!pip install praw
import pandas as pd
import datetime as dt
from datetime import datetime
import requests
import json
from bs4 import BeautifulSoup
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 *
from wordcloud import WordCloud

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')
```

In [ ]:
```
r = praw.Reddit(client_id='D-XnJXdrjvvs1A',
                client_secret='S7V0nVQKCkLiu47CdNUYgzc2zhE',
                password='abuDhabi20',
                user_agent='nyuaddhclass',
                username='dhclass')
print(r.user.me())
```

In [ ]:
```
# # Maryam:
# subreddit="gameofthrones"
# titlename = "Game of Thrones Reddit posts"
# npo=1000
# # Khoula:
# subreddit="PrisonBreak"
# titlename = "Prison Break Reddit posts"
# npo=1000
# # Yoon Hee:
# subreddit="StrangerThings"
# titlename = "Stranger Things Reddit posts"
# npo=1000
# # Olivia:
# subreddit="Scandal"
# titlename = "Scandal Reddit posts"
# npo=1000
# # Mar:
# subreddit="westworld"
# titlename = "West World Reddit posts"
# npo=1000
# # Joonha:
# subreddit="southpark"
# titlename = "South Park Reddit posts"
# npo=1000
# # Adham:
# subreddit="freefolk"
# titlename = "Free Talk Reddit posts"
# npo=1000
# # Benjamin:
# subreddit="Devs"
# titlename = "Devs Reddit posts"
# npo=1000
```

In [ ]:
```
posts = []
cvd_subreddit = r.subreddit(subreddit)
for post in cvd_subreddit.hot(limit=npo):
    if post.author==None:
        posts.append([post.title, post.score, "deleted", post.url, post.num
    else:
        posts.append([post.title, post.score, post.author.name, post.url, p
posts = pd.DataFrame(posts,columns=['title', 'score', 'redditor', 'url', 'n
posts['created'] = pd.to_datetime(posts['created'], unit='s')
npo=len(posts)
# print(npo)
mind=posts.created.min().strftime("%d-%m-%Y %H:%M:%S")
maxd=posts.created.max().strftime("%d-%m-%Y %H:%M:%S")
print("The", titlename, "dataframe contains", len(posts), "posts from", min
posts=posts.sort_values('created',ascending=True)
posts.head(4)
```

In [ ]:
```
# s = pd.to_datetime(posts['created'])
# dposts = s.groupby(s.dt.floor('d')).size().reset_index(name='count')
# dposts
```

In [ ]:
```
posts['day']=posts['created'].dt.floor('d')
gdf=posts.groupby("day").nunique()[["title","redditor"]]
gdf = gdf.reset_index()
gdf.rename({'title': 'num_posts', 'redditor': 'num_redditors'}, axis=1, inp
gdf1=posts.groupby("day").sum()[["num_comments"]]
gdf1 = gdf1.reset_index()
gdf2 = pd.merge(gdf, gdf1, on='day')
gdf2=gdf2[['day','num_posts','num_redditors','num_comments']]
gdf2
```

In [ ]:
```
ax=gdf2.plot(x='day', y=["num_posts","num_redditors"], kind="line",figsize=
ax.set_title('Number of posts and redditors per day', fontsize=14);
```

In [ ]:
```
# If the above line-plot does not look nice, please try the bar-plot as fol

ax=gdf2.plot(x='day', y=["num_posts","num_redditors"], kind="bar",figsize=(
ax.set_title('Number of posts and redditors per day', fontsize=14);
plt.xticks([], []);
```

In [ ]:
```
posts['index'] = range(1, len(posts) + 1)
postsC=posts[['index','title', 'score', 'redditor', 'url', 'num_comments',
docs_d={}
for i in range(npo):
    j=posts.iloc[i]['body']
    a=posts.iloc[i]['index']
    d=posts.iloc[i]['created']
    if type(j)!=float:
        j=j.lower()
        j=j.lstrip().replace("\r","").replace("\n.\n.\n"," ").replace("\n."
        docs_d[a]=(d,j)
    else:
        docs_d[a]=(d,"")
print(len(docs_d))
docs=list([t[1] for t in docs_d.values()])
print(len(docs))
for k,v in docs_d.items():
    print(k,v[0],'\n',v[1])
    print('')
```

In [ ]:
```
num_unique_words=[]
for k,v in docs_d.items():
    words = word_tokenize(v[1])
    nuw=len(words)
    uw=len(set(words))
    num_unique_words.append(uw)
#     print("Post on %s contains %i nonunique and %i unique words"%(k,nuw,u
posts['num_unique_words']=num_unique_words
gdf3=posts.groupby("day").sum()[["num_unique_words"]]
gdf3 = gdf3.reset_index()
gdf4 = pd.merge(gdf2, gdf3, on='day')
gdf4=gdf4[['day','num_posts','num_comments','num_unique_words']]
gdf4
```

In [ ]:
```
ax=gdf4.plot(x='day', y=["num_comments",'num_unique_words'], kind="line",fi
ax.set_title('Number of comments and unique words per day', fontsize=14);
```

In [ ]:
```
# If the above line-plot does not look nice, please try the bar-plot as fol

ax=gdf4.plot(x='day', y=["num_comments",'num_unique_words'], kind="bar",fig
ax.set_title('Number of comments and unique words per day', fontsize=14);
plt.xticks([], []);
```

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

text=" ".join(docs)
docs_d={subreddit:text}
for key, value in docs_d.items():
    arr = []
    doc = nlp(value.replace('\n',''))
    #Keep these types of nlp entities
    keep_l = ['PERSON'] #,'NORP','PRODUCT','ORG']
    #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)
names=sorted(set(names))
print(len(names))
names
```

In [ ]:
```
rem=[]
for p in names:
    if "_" in p:
        rem.append(p)
    if "-" in p:
        rem.append(p)
    if "--" in p:
        rem.append(p)
    if len(p)<3:
        rem.append(p)
#     if p not in text:
#         rem.append(p)
names=[p for p in names if p not in rem]
print(len(names))
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
```

In [ ]:
```
# Read carefully the above list of names, and add names in the two lists re

rem=['Jodie Whittakers','Tony Starks','Christopher Ecclestons','Amright','P
     'Lesley Sharps','David Tennants','Chris Chibnalls',


    ]
# rem=[i.lower() for i in rem]
added=[]
names=[p for p in names if p not in rem]
names=names+added
names=sorted(set(names))
print(len(names))
names
```

In [ ]:
```
nfreq=[]
for i in names:
    nfreq.append(text.count(i.lower()))
pnf_df = pd.DataFrame(
    {'Names': names,
     'Frequency of Occurrences': nfreq
    })
pnf_df=pnf_df[['Names','Frequency of Occurrences']]
pnf_df=pnf_df.sort_values(by ='Frequency of Occurrences',ascending=False)
# trf_df=trf_df[trf_df["Frequency of Occurrences"]>10]
print(len(pnf_df))
pnf_df #.tail(10) #[:50]
```

In [ ]:
```
x = pnf_df.set_index('Names').T.to_dict()
x=sorted([(k,v['Frequency of Occurrences']) for k,v in x.items()], key=lamb
for i,j in x:
    if j==0:
        print(i)
```

In [ ]:
```
x = pnf_df.set_index('Names').T.to_dict()
x=sorted([(k,v['Frequency of Occurrences']) for k,v in x.items()], key=lamb
keys = [i for (i,j) in x if j>3]
y_pos = np.arange(len(keys))
performance = [j for (i,j) in x if j>3]
plt.figure(figsize=(10,8))
ax = plt.axes()
plt.barh(y_pos, performance, align='center', alpha=0.6)
ax.invert_yaxis()
plt.yticks(y_pos, keys)
plt.xlabel('Frequency')
plt.title('Top Names')
plt.show()
```

In [ ]:
```
t=[]
for (i,j) in x:
    for k in range(j):
#         print i.replace(" ","_").replace("-","_")
        t.append(i.replace(" ","_").replace("-","_"))
ttd=' '.join(t)
wordcloud = WordCloud(collocations=False,background_color="white",colormap=
fig = plt.figure(figsize=(13,13))
default_colors = wordcloud.to_array()
plt.imshow(default_colors, interpolation="bilinear")
plt.axis("off")
ss="WordCloud of Names in %s" %titlename
plt.suptitle(ss,fontsize=25)
plt.tight_layout(rect=[0, 0, 1, 1.4])
plt.show()
```

In [ ]:
```
# WITHOUT ALIASES:
alias_dict={}
for n in names:
    alias_dict[n]=n

# # WITH ALIASES:
# alias_dict={}
# for n in names:
#     if n=="try":
#         alias_dict[n]="tri"
#     elif n=="tries":
#         alias_dict[n]="tri"
#     elif n=="tried":
#         alias_dict[n]="tri"
#     elif n=="send":
#         alias_dict[n]="send"
#     elif n==" sent ":
#         alias_dict[n]="send"
#     elif n=="feel":
#         alias_dict[n]="feel"
#     elif n=="felt":
#         alias_dict[n]="feel"
#     else:
#         alias_dict[n]=n
```

In [ ]:
```
blob = TextBlob(text)
textSentences = blob.sentences
sendic=dict()
for i,v in enumerate(textSentences):
    sent=v.sentiment.polarity
    wl=[]
    for term in [w.lower() for w in list(set(alias_dict.values()))]:
        if term in v:
            wl.append(term.title())
    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 aliased proper n
medges
```

In [ ]:
```
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)

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)
```

In [ ]:
```
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=[1*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=[10*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 = -15 # 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=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')
titlename1="%s \n from %s to %s" %(titlename,mind[:10],maxd[:10])
sst="The graph of co-occurring proper names in the %s \n weighted over thei
plt.title(sst,fontsize=17);
plt.margins(x=0.1, y=0.1)
```

In [ ]:
```
nodes=len(Gw.nodes())
edges=len(Gw.edges())
print(nodes,edges) #,nodes*(nodes-1)/2)
```

In [ ]:
```
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=[10*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 = -15 # 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=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')
titlename1="%s \n from %s to %s" %(titlename,mind[:10],maxd[:10])
sst="The graph of co-occurring proper names in the %s \n weighted over thei
plt.title(sst,fontsize=17);
plt.margins(x=0.1, y=0.1)
```

In [ ]:
```

```