Corpus_unsupervised_learning_bhogan

August 25, 2020

[ ]:
```
"""
Author: Brian Hogan 2019
Purpose: Import and vectorize corpus, clean stopwords\stem, normalize, perform␣
 ↪k-means
Topic: Identification of disputed authors in Federalist Papers
Notes: base technique used regularly categorizing misc web and article data
bphogan@syracuse.edu
"""
```

[ ]:
```
# BUILD CORPUS

import nltk
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
import os
os.getcwd()
os.chdir('c:\\Users\BBE\BBE\DATA\Federalist_Papers')

# this method uses reading by a path
path = "C:\\Users\\BBE\\BBE\\DATA\\Federalist_Papers"  #wow need 2 slashs
#print(os.listdir(path))
# save the lsit
filenamelist = os.listdir(path)
print(type(filenamelist))  #save as a list  #check the type

#need complete paths to work with CountVectorizer...CONSTRAINT OF METHOD
listofcompletefilepaths =[]  #need an empty list
listofjustfilenames = []
for name in os.listdir(path):
    #print(path+ "\\" + name)
```

```
    next = path+ "\\" + name
    nextnameL = name.split(".")
    nextname = nextnameL[0]  #this is pretty interesting...
    listofcompletefilepaths.append(next)
    listofjustfilenames.append(nextname)
#print(listofcompletefilepaths)
#print(listofjustfilenames)
len(listofjustfilenames)
```

[ ]:
```
# VECTORIZE AND CREATE DOCUMENT TERM MATRIX

myvect3 = CountVectorizer(input='filename')
    #CountVectorizer(analyzer='word', binary=False, decode_error='strict',
    #        dtype=<class 'numpy.int64'>, encoding='utf-8', input='filename',
    #        lowercase=True, max_df=1.0, max_features=None, min_df=1,
    #        ngram_range=(1, 1), preprocessor=None, stop_words=None,
    #        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
    #        tokenizer=None, vocabulary=None)
x_dh = myvect3.fit_transform(listofcompletefilepaths) #vector w file names
x_dh.shape  ## documents by the total words
#print(x_dh) #now what do we have
    #  (0, 6387)     1           still not sure what this is!
    #  (0, 6056)     1

#get the feature names WHICH ARE THE WORDS!
colnames_original = myvect3.get_feature_names()
print(colnames_original)
len(colnames_original)

#Create  a document term model - DTM ( a matrix of counts)
corpusDF0 = pd.DataFrame(x_dh.toarray(), columns=colnames_original)
print(corpusDF0)

#simple dictionary for filename + generic numeric ID
mydict = {}  #now update the row names (corpus file names)
for i in range(0, len(listofjustfilenames)):
    mydict[i] = listofjustfilenames[i]
print(mydict)

#buildthe corpus with teh papernames based on the file names
corpusDF0 = corpusDF0.rename(mydict, axis="index")
print(corpusDF0)

df_output = pd.DataFrame(corpusDF0) ## inspection
output_data = df_output  #output the total tweet datatable
output_data.to_csv("aBBE_today.csv", index=True)
```

[ ]:
```
# CLEAN AND ADDRESS STOPWORDS

#in pandas
corpusDF0['zeta'] #USED FOR unknown authors...

#print("Initial column names: \n", columnnames3)
mystops = ["also","and","are","you","of","let","not","the","for","why",
           "there","one","which","000","10","11","12","13","14","15","16",
          '20', '21', '22', '23', '24', '25', '257', '26', '262', '27',
          '28', '29', '2d', '30', '31', '32', '33', '34', '35',
          '36', '37', '38', '39', '3d', '40', '41', '42', '43', '438',
          '44', '45', '46', '47', '48', '49', '4th', '50', '51', '52',
          '53', '54', '55', '56', '57', '58', '59', '5th', '60', '61',
          '62', '63', '64', '65', '66', '67', '68', '69', '70', '71',
          '72', '73', '74', '75', '76', '77', '78', '79', '80', '81',
          '82', '83', '84', '85' ]
# [85 rows x 8719 columns] # for the federalist
cleanDF = corpusDF0 # make a cleanDF to add and remove columns
colnames_new = []   #build a new colmns list
for name in colnames_original:
    #print("FFFF",name)
    if((name in mystops) or (len(name)<3)):
        #print("word dropping: ",name)
        cleanDF = cleanDF.drop([name],axis=1) # drop stopword column
        #print(cleanDF)
    else:
        colnames_new.append(name)
cleanDF.shape       # Out[48]: (85, 8588)
len(colnames_original) # origial import
len(colnames_new) #with stopwrods removed
colnames_new
```

[ ]:
```
# SET NEW DATAFRAME COLUMNS NAMES

change_tracker=[]
for name1 in colnames_new:  #string operations getting rid of word after letter
    for name2 in colnames_new:  #on the right
        if (name1 == name2):
            print("skip")
        elif(name1.rstrip("e") in name2): #thi sis good for plurals
            change_tracker.append(name1+ " " + name2)
            # like dog an dogs, but not for the hike an hiking
            #so I will srip and "e" if there is one...
            print("combining:",name1, name2)
            #print(corpusDF0[name1] + corpusDF0[name2])
            #new = name1 + name2
            cleanDF[name1] = cleanDF[name1] + cleanDF[name2]
```

```
            cleanDF = cleanDF.drop([name2], axis=1) #axis 1 is columns
change_tracker
len(change_tracker)
change_tracker
print(cleanDF.columns.values)
```

[ ]:
```
# STEMMING AND ADDRESSING WORD CONSOLIDATION

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()        #print("Stemmed Word:",stem.stem(word))
change_tracker=[]

colnames[0:50] #list of words to debug this stemming...

word_family = []
skip_track=[]
for name1 in colnames_new:  #string operations getting rid of word after letter
    word_family
    word1 = stem.stem(name1)
    stem_colnames.append(name1)
    for name2 in colnames:
        word2 = stem.stem(name2)
        if (word1 == word2):
            stem_colnames.append(name2)

word_family = []                                          #########  STEMMING
i=0
while i <= len(colnames_new):
    name1 = colnames[i]
    stem1 = stem.stem(name1)
    word_family.append(stem1)
    i = i+1

colnames_new[0:25]

stem_colnames
len(stem_colnames)                                       #########  STEMMING
len(colnames_new)
stemword_freqency = nltk.FreqDist(stem_colnames)
for key in stemword_freqency:
    if stemword_freqency[key] >5:
        print(key,stemword_freqency[key])

df_output = stemword_freqency.values
df_output
output_data = df_output  #output the total tweet datatable
output_data.to_csv("aBBE_today.csv", index=True)
```

```
for name1 in colnames_new:  #string operations getting rid of word after letter
    for name2 in colnames_new:  #on the right
        #if (name1 == name2): #if words equal at  start word position in loop
            #print("skip")
        if(stem.stem(name1) == stem.stem(name2)): #think should look for all␣
 ↪subsequent
            #change_tracker.append(name1+ " " + name2)
                                    #sten cases of same word
                                    #'abandon abandon',
                                    #'abandon abandoned',
                                    #'abandon abandoning',
            change_tracker.append(name1+ " " + name2)
            print("combining:",name1, name2)
            #print(corpusDF0[name1])
            #print(corpusDF0[name2])
            #print(corpusDF0[name1] + corpusDF0[name2])
            cleanDF[name1] = cleanDF[name1] + cleanDF[name2]
            cleanDF = cleanDF.drop([name2], axis=1) #axis 1 is columns
change_tracker
cleanDF.shape
```

[ ]:
```
# LABEL CLEANED NEW DATA FRAME
cleanDF.iat[1,1]  #THIS WORKS  cleanDF['zeta'] #this works here

doc=[]
authorYN=[]
for x in range(0, len(cleanDF)):
    y = cleanDF.columns.get_loc("zeta") #get column index
    y1 = cleanDF.columns.get_loc("hamilton")
    y2 = cleanDF.columns.get_loc("jay")
    y3 = cleanDF.columns.get_loc("madison")
    if cleanDF.iat[x,y] == 1:  #disputed  data brought in
        z = cleanDF.iat[x,y]
        authorYN = 0
    if cleanDF.iat[x,y1] == 1: #hamilton
        z = 2
        authorYN= 1
    if cleanDF.iat[x,y2] == 1: #jay
        z = 3
        authorYN= 1
    if cleanDF.iat[x,y3] == 1: #madison
        z = 4
        authorYN= 1                  #hamilton + madison below
    if cleanDF.iat[x,y1] == 1 and cleanDF.iat[x,y3] == 1:
        z = 5
        authorYN= 1
```

```
    doc.append(z)
    authorYN.append(authorYN)
    z=0
    authorYN=99
documents = pd.DataFrame(doc)
documents = documents.rename(mydict, axis="index")
documents = documents.rename(columns={0:'doc'})
documents.head()
#dataframe for authorYN label
authoryn = pd.DataFrame(authorYN)
authoryn = authoryn.rename(mydict, axis="index")
authoryn = authoryn.rename(columns={0:'authorYN'})
authoryn.head(12)
#update the source dataframe with the new settings for
cleanDF['zeta'] = authoryn['authorYN']
z=cleanDF.columns.get_loc("zeta") #get column index
cleanDF = cleanDF.rename(columns={'zeta':'authorYN'})
cleanDF.head()

testDF= cleanDF
##add labels back into the dataframe
testDF = documents.to_frame() #index to 0  #thi sis interesting!
print(type(documents))
testDF.index = documents.index - 1
#print(new_labels)
labeledclean_DF["Label"] = new_labels
```

[1]:
```
#LONG HAND CODING OF TF-IDF
# often use sci-kit learn as well

import math

df_data = pd.DataFrame(cleanDF).values.astype(int)
df_data
#transpose the frame
df_data_transposed = df_data.T  #transpose the frame
df_data_transposed[0]
df_data_transposed.shape[1] ## of words transposed, want 1 for docs

mydocfreq=[]   #word counts across the documents
for x in range(0,len(df_data_transposed)):
    wf = int(sum(df_data_transposed[x])) #[x]
    idf = wf / df_data_transposed.shape[1]  #number of docs
    mydocfreq.append(idf)
    wf=""
    idf = ""
```

```
df_mydocfreq_inverse = pd.DataFrame(mydocfreq).values.astype(float)
df_mydocfreq = pd.DataFrame(df_mydocfreq_inverse).T  #thats right make 1 x 1384
df_mydocfreq

df_tfidf = pd.DataFrame(df_data).values.astype(float) #build frame
df_tfidf.shape
#zero out the dataframe - I DOULBLe checked this owrking
for x in range(0,len(df_tfidf)): #
    y=0
    #demoninator = float(df_mytotalword_perdoc[x])
    while y <= (df_tfidf.shape[1]-1):  #shape gives the y dimension of columns
        df_tfidf[x,y] = 0
        y +=1
#####===> TF-IDF the data
for x in range(0,len(df_data)): # rows in data frame
    y=0
    while y <= (df_tfidf.shape[1]-1):  #shape gives the y dimension of columns
        df_tfidf[x,y] = df_data[x,y]* math.log(mydocfreq[y])
        y +=1
df_tfidf.shape

#export back to Excel
DF_Homework = pd.DataFrame(df_tfidf)
output_data = DF_Homework  #output the total tweet datatable
output_data.to_csv("aBBE_inspect.csv", index=True)

labeledclean_DF =pd.DataFrame(df_tfidf,columns=colnames_new)
labeledclean_DF = labeledclean_DF.rename(mydict, axis="index")
labeledclean_DF['zeta'] = authoryn['authorYN']
labeledclean_DF['128'] = documents['doc']
labeledclean_DF = labeledclean_DF.rename(columns={'zeta':'authorYN'})
labeledclean_DF = labeledclean_DF.rename(columns={'128':'Doc'})
labeledclean_DF
```

```
  File "<ipython-input-1-e3574da3771e>", line 1
    \#Normalization
                   ^
SyntaxError: unexpected character after line continuation character
```

[ ]:
```
# CLUSTERING

print(type(labeledclean_DF)) #check the type is a dataframe
from sklearn.cluster import KMeans#Using SKlearn - - WOWSERS IS THSI FAST...
import numpy as np #kmeans_object = sklearn.cluster.KMeans(n_clusters=3)
```

```
#print(kmeans_object)
    #KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    #    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    #    random_state=None, tol=0.0001, verbose=0)
#K-means model
mymatrix_data = labeledclean_DF.values #matrix of k-means data
kmeans_object = KMeans(n_clusters=4)   #tyring 3 and 4 clusters
kmeans_object.fit(mymatrix_data)       #fit model
labels = kmeans_object.labels_   #get cluster assignment labels
#Build Results
myresults = pd.DataFrame([corpusDF0.index,labels]).T #format results as DF
myresults = myresults.rename(mydict, axis="index")    #add column to merge
myresults = myresults.rename(columns={1:'k-means-label'}) #renaming
myresults = myresults.rename(columns={0:'docname'})
myresults.head()
documents = pd.DataFrame(doc) #original list of the documents from import
documents = pd.DataFrame([corpusDF0.index,labels]).T #add column to merge
documents = documents.rename(columns={1:'authorID'})  #renaming
documents = documents.rename(columns={0:'docname'})
documents.head()
#Merge the results
finalDF = myresults.merge(documents, on='docname')  #yippeeeeeeeee  !!!!
finalDF
from pandas_ml import ConfusionMatrix
from sklearn.metrics import confusion_matrix
y_actual=[]
y_predict=[]
y = finalDF.columns.get_loc("authorID") #get column index
y1 = finalDF.columns.get_loc("k-means-label")
y
for x in range(0,len(finalDF)):
    y_actual.append(finalDF.iat[x,y])
    y_predict.append(finalDF.iat[x,y])
y_actual
```

[ ]:
```
# HEATMAP AND CONFUSION MATRIX OF K-MEANS RESULTS

confusion_matrix = confusion_matrix(y_actual,y_predict)
confusion_matrix

import seaborn as sn
import matplotlib.pyplot as plt
df_cm = pd.DataFrame(confusion_matrix, range(4),range(4))
sn.set(font_scale=1.4)
sn.heatmap(df_cm,annot=True,annot_kws={"size":16}) #font size
```

[ ]:
```
# WORDCLOUD ACROSS CORPUS

listofjustfilenames[0] #GET LIST of file data names
mycorpus_data=[]
for i in range(0,len(listofjustfilenames)):
    filename = open(listofjustfilenames[i] + ".txt","r")
    for line in filename:
        textline = line.strip()
        mycorpus_data.append(textline)
    filename.close()
len(mycorpus_data)
#inspecting file names in excel to make s graph
df_output = pd.DataFrame(mycorpus_data)
output_data = df_output  #output the total tweet datatable
output_data.to_csv("aBBE_Federalist_Papers_by_line.csv", index=True)

mycorpus_data
wordlist = []  # join all
wordlist = " ".join(mycorpus_data)
wordlist
tokenized_word=word_tokenize(wordlist)
len(tokenized_word)
tokenized_word
stop_words=set(stopwords.words("english"))
corpus_no_stopwords=[]
for w in tokenized_word:
    if w not in stop_words:
        corpus_no_stopwords.append(w)
#==> 3) word Frequency
len(corpus_no_stopwords)
corpus_no_stopwords

import re  #now perform more cleaning
mystops =␣
 ↪["also","and","are","you","of","let","not","the","for","why","there","one","which"]
newlist = []
for word in corpus_no_stopwords:
    #print("the new word is: ",word)
    #placeinoutputfile = "The next word before is: " + word + "\n"
    #OUTFILE.write(placeinoutputfile)
    word = word.lower()
    word = word.lstrip()
    word = word.strip("\n")
    word = word.strip("\\n")
    word = word.replace(",","")
    word = word.replace(" ","")
    word = word.replace("_","")
```

```
    word = re.sub('\+', '',word)
    word = re.sub('.*\+\n','',word)   ##LOOKS FUNNY! single quotes!
    word = re.sub('zz+','',word)
    word = word.replace("\t","")
    word = word.replace(".","")
    word = word.replace("\'s","")  #was comment3d out
    word = word.strip()
    ##word.replace("\","")  #was commented out
    #if((name in mystops) or (len(name)<3)):
    if ((word not in["","\\","'","*",":",";"]) or (word not in mystops)):
        if len(word) >=3:
            if not re.search(r'\d',word): ##remove the digits
                # HW2 ===non english words
                newlist.append(word)
                #placeinoutputfil = "The next word AFTER is: " + word + "\n"
                #OUTFILE.write(placeinoutputfile)
len(corpus_no_stopwords)
len(newlist)
newlist
```

[ ]:
```
# WORDCLOUD MOST FREQUENT WORDS

mostfrequentwords = nltk.FreqDist(newlist)
mostfrequentwords
top_words=mostfrequentwords.most_common(200) #words used most in the tweets
DF_topwords = pd.DataFrame(top_words)
print("....50 Top Words from Tweets. \n",DF_topwords)
top_words
wordcloud_items=[] #make a dictionary  ====>move to dictionary in future
for word, freq in top_words:   #print the most commone words
        print("Word:",word,freq)
        wordcloud_items.append(word)
print(wordcloud_items)
from PIL import Image
#>conda install -c conda-forge wordcloud
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.pyplot as plt
wordcloud_items = " ".join(wordcloud_items)  ##  join
#print(joinedfilteredtweets)  # lower max_font_size, change the maximum number␣
 ↪of word and
    #lighten the background:"""                                    #white, purple,␣
 ↪etc
wordcloud = WordCloud(max_font_size=50, max_words=100,␣
 ↪background_color="purple").generate(wordcloud_items)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
```

```
plt.show()
```