EE 586
Statistical Foundations of Natural Language Processing
Assignment 2

Emirhan Koc
emirhan.koc@bilkent.edu.tr

Bilkent University, Electrical and Electronics Engineering

***Abstract*— In this assignment, it is asked to find collocations in bigram form with 3 hypothesis testing methods. These are student's t-test, chi-square test and likelihood ratio test. In this assignment, a corpus which consist of seven books of Charles Dickens are provided and investigations are performed using it. These are Bleak House, David Copperfield,Great Expectations, A Tale of Two Cities, Hard Times, Oliver Twist and The Pickwick Papers.**

I. INTRODUCTION

In this assigment, it is mainly aimed to determine whether a bigram is collation or not using three different hypothesis testing such as student's t-test, chi-square test and likelihood ratio test. In line with this objective, a corpus which consist of seven books Charles Dickens are used to perform processing and experiments. These book are Bleak House, David Copperfield, Great Expectations, A Tale of Two Cities, Hard Times, Oliver Twist and The Pickwick Papers.

A. Tokenize

In this part, initially the text is tokenized using *nltk* library of Python and **1899306** tokens are extracted from the text.

B. Part-of-Speech Tag

In this part, part-of-speech tag of each token is obtained. It is important to acquire them as their tags are used in further section to filtering process for desired bigrams. In this part, tokens are given to function as list and **word:tag** pairs are obtained as list of tuples.

C. Lemmatizer

In this part, **word:tag** pairs are modified according their their tags. As in the further section, it is only needed to deal with adjectives and nouns, only they are lemmatized. It is worth to note that there are several exceptions such as men, people, saw that remain same.

D. Bigram with a Window Size

In this section, bigrams are created using different window size options. For the window size 1, ". . . word **word1 word2** word . . . ", *word1word2* is bigram. For the window size 3, ". . . word **word1 word2** word . . . " ". . . word **word1** word **word2** word . . . ", ". . . word **word1** word word **word2** word . . . ", *word1word2* is considered as bigram. To obtain bigrams with different window size, an empty is list created initially. As only NOUN-NOUN and ADJ-NOUN pairs will be taken into consideration, iteratively, current word and the next word is put into the list if tag pair is NOUN-NOUN or ADJ-NOUN for window size 3. For window size 3, it is crucial to be carefull about end of the list case as if a word is last second at the end, second and third word after it cannot be taken. Therefore, each case is considered for first after current, second after current, third after current.

```
for bb in range(len_postag-1):
        bg_word= lemmatized_pt[bb][0] + " " \
        + lemmatized_pt[bb+1][0]
        bg_pt = lemmatized_pt[bb][1] + "-" + \
        lemmatized_pt[bb+1][1]
        if bg_pt == 'NOUN-NOUN' or bg_pt =='ADJ-NOUN':
          bigram_dic.append([bg_word,bg_pt])
```

E. Final Processing

After obtaining NOUN-NOUN, ADJ-NOUN pairs merely, all pairs which either of the word contains special character or included in stop words are eliminated from the list of pairs. Finally, if a pair is seen less than 10 times in bigram pair list, they also are exluded. Now, the final list of bigrams for both window size are obtained.

F. Tests and Results

In this part, it is desired to perform statistical test mentioned above. Test are conducted for two collocation candidates : **cursitor street** and **good one** with window size 1. First, T-test is performed on both of the word with the following procedure. As H0, null hypothesis, word1 and word2 are assumed independent. As the alternative hypothesis H1, they are assumed dependent and are candidate of collocation.

$$\frac{\bar{X}-\mu}{\frac{S}{\sqrt{N}}} \tag{1}$$

According to Eq.1, $\bar{X}$ defines the probability of occurence of the collocation candidate , μ defines probability of occurence of word1 and word2 considering they are independent. S is the sample varience and N is total number of words in the text after lemmatization.

EE586_HW2

April 4, 2022

[ ]:
```
import nltk
import os
import io
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
nltk.download('universal_tagset')
nltk.download('wordnet')
import custom_lemmatizer
from nltk.corpus import wordnet
```

[ ]:
```
cur_dir = os.getcwd()
file_name = "Charles Dickens Processed.txt"
filepath= os.path.join(cur_dir,file_name)

with io.open(filepath,'r',encoding='utf8') as f:
    text = f.read()
text=text.replace("\n"," ")
#Tokenization
#tokenizer = RegexpTokenizer(r'\w+')
word_list = word_tokenize(text)
#P-o-S Tagging
post_tag= nltk.pos_tag(word_list,tagset="universal")
```

[ ]:
```
print("Total token size is ", len(word_list))
```

[ ]:
```
# lemmatization is done
lem_pos = []
cm = custom_lemmatizer.custom_lemmatizer()
for t in post_tagged:
     lem_pos.append(cm.lemmatize(t))

post_tag= []
for ii, pp in enumerate(post_tagged):

    pp = list(pp)
    pp[0]=lem_pos[ii]
    pp=tuple(pp)
```

```
    post_tag.append(pp)
```

[ ]:
```
# print(lem_pos.count("that"))
```

[ ]:
```
def bigram_creater(lemmatized_pt,bigram_choice=1):

    bigram_dic = []
    len_postag = len(lemmatized_pt)

    if bigram_choice ==1:

        for bb in range(len_postag-1):
            bg_word= lemmatized_pt[bb][0] + " " + lemmatized_pt[bb+1][0]
            bg_pt = lemmatized_pt[bb][1] + "-" + lemmatized_pt[bb+1][1]
            if bg_pt == 'NOUN-NOUN' or bg_pt =='ADJ-NOUN':
                bigram_dic.append([bg_word,bg_pt])

    elif bigram_choice ==3:

        for bb in range(len_postag-1):
            bg_word1= lemmatized_pt[bb][0] + " " + lemmatized_pt[bb+1][0]
            bg_pt1 = lemmatized_pt[bb][1] + "-" + lemmatized_pt[bb+1][1]
            if bg_pt1 == 'NOUN-NOUN' or bg_pt1 =='ADJ-NOUN':
                bigram_dic.append([bg_word1,bg_pt1])
        for bb in range(len_postag-2):
            bg_word2= lemmatized_pt[bb][0] + " " + lemmatized_pt[bb+2][0]
            bg_pt2 = lemmatized_pt[bb][1] + "-" + lemmatized_pt[bb+2][1]

            if bg_pt2 == 'NOUN-NOUN' or bg_pt2 =='ADJ-NOUN':
                bigram_dic.append([bg_word2,bg_pt2])
        for bb in range(len_postag-3):
            bg_word3= lemmatized_pt[bb][0] + " " + lemmatized_pt[bb+3][0]
            bg_pt3 = lemmatized_pt[bb][1] + "-" + lemmatized_pt[bb+3][1]
            if bg_pt3 == 'NOUN-NOUN' or bg_pt3 =='ADJ-NOUN':
                bigram_dic.append([bg_word3,bg_pt3])

    return bigram_dic
```

[ ]:
```
bigram1= bigram_creater(post_tag,1)
bigram2= bigram_creater(post_tag,3)
```

[ ]:
```
```

```
stop_words = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves",␣
 ↪"you", "your", "yours", "yourself", "yourselves", "he", "him", "his",␣
 ↪"himself", "she", "her", "hers", "herself", "it", "its", "itself", "they",␣
 ↪"them", "their", "theirs", "themselves", "what", "which", "who", "whom",␣
 ↪"this", "that", "these", "those", "am", "is", "are", "was", "were", "be",␣
 ↪"been", "being", "have", "has", "had", "having", "do", "does", "did", "doing",␣
 ↪"a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while",␣
 ↪"of", "at", "by", "for", "with", "about", "against", "between", "into",␣
 ↪"through", "during", "before", "after", "above", "below", "to", "from", "up",␣
 ↪"down", "in", "out", "on", "off", "over", "under", "again", "further", "then",␣
 ↪"once", "here", "there", "when", "where", "why", "how", "all", "any", "both",␣
 ↪"each", "few", "more", "most", "other", "some", "such", "no", "nor", "not",␣
 ↪"only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will",␣
 ↪"just", "don", "should", "now"]
```

[ ]:

[ ]:
```
#Eliminate stop-words and keep only alphabethic strings, eliminate bigrams less␣
 ↪than 10 time occurence
def bigram_modified(bigram1):

    bigram1_temp = []

    for ll in range(len(bigram1)):

        woi = bigram1[ll][0]
        woi_1= woi.split()[0]
        woi_2=woi.split()[-1]
        if woi_1 not in stop_words and  woi_2 not in stop_words and woi_1.
 ↪isalpha() and woi_2.isalpha() and bigram1.count(bigram1[ll])>=10:

                bigram1_temp.append(bigram1[ll])

    return bigram1_temp
```

[ ]:
```
import time
start = time.time()
bigram1_modified= bigram_modified(bigram1)
bigram2_modified = bigram_modified(bigram2)
end = time.time()
print(end-start)
#bigram2_modified = bigram_modified(bigram2[1:10000])
```

[ ]:
```
bg11 = bigram1_modified
bg22= bigram2_modified
```

[ ]:

[ ]:
```
def bigrams_str(bigram):
    bigram_temp = []
    for ii in range(len(bigram)):
        bigram_temp.append(bigram[ii][0])
    return bigram_temp
bigram_1_str= bigrams_str(bigram1_modified)
bigram_2_str = bigrams_str(bigram2_modified)
```

[ ]:
```
import pandas as pd
import numpy as np
```

[ ]:
```
def count_displayer(bigram,lem,choice=1):
    unique_col= list(set(bigram))
    df = pd.DataFrame(np.zeros((len(unique_col),4)),
                      columns=['Bigram', 'c_w1w2', 'w1','w2'])
    for ii,w in enumerate(unique_col):

        df.loc[[ii],"Bigram"]= w
        df.loc[[ii],"c_w1w2"]=bigram.count(w)
        if choice==1:
            df.loc[[ii],"w1"]=lem.count(w.split()[0])
            df.loc[[ii],"w2"]= lem.count(w.split()[-1])
        elif choice==3:
            df.loc[[ii],"w1"]=lem.count(w.split()[0])*3
            df.loc[[ii],"w2"]= lem.count(w.split()[-1])*3


    return df
```

[ ]:
```
df_bg1=count_displayer(bigram_1_str,lem_pos,1)
df_bg2 = count_displayer(bigram_2_str,lem_pos,3)
#df_bg1["Rank"]=np.arange(1,21)
#df_bg2["Rank"]= np.arange(1,21)
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
def t_test(df,lem,choice=1):
    t_score = []
    if choice==1:
         twl= len(lem)
    elif choice==3:
        twl=len(lem)*3
    for ii in range(df.shape[0]):
```

```
        x_bar= df['c_w1w2'].iloc[ii]/twl

        mu = (df['w1'].iloc[ii]/twl)*(df['w2'].iloc[ii]/twl)

        sv=x_bar*(1-x_bar)
        t_score.append((x_bar-mu)/np.sqrt(sv/twl))
    df["t-score"]=t_score
    return df
```

[ ]:
```
df_bg1_T = t_test(df_bg1,lem_pos,1).sort_values(by=['t-score'],ascending=False)
df_bg2_T = t_test(df_bg2,lem_pos,3).sort_values(by=['t-score'],ascending=False)
```

[ ]:
```
df_bg1_T[df_bg1_T["Bigram"]=="good one"]
```

[ ]:
```
df_bg1_T.head(20)
```

[ ]:
```
df_bg2_T.head(20)
```

[ ]:
```

```

[ ]:
```
def chi_square(df,lem,choice=1):

    chi = []
    if choice==1:
        lp= len(lem)
    elif choice==3:
        lp=len(lem)*3
    for ii in range(df.shape[0]):
        obs= np.zeros((2,2))
        obs[0,0]=df["c_w1w2"].iloc[ii]
        obs[0,1]=df["w1"].iloc[ii]-df["c_w1w2"].iloc[ii]
        obs[1,0] = df["w2"].iloc[ii]-df["c_w1w2"].iloc[ii]
        obs[1,1] = lp-(obs[0,0]+ obs[0,1]+ obs[1,0])
        chi_sq = lp*(( obs[0,0]* obs[1,1]- obs[0,1]* obs[1,0])**2)/(( obs[0,0]+␣
↪obs[0,1])*( obs[0,0]+ obs[1,0])*( obs[0,1]+ obs[1,1])*( obs[1,0]+ obs[1,1]))
        chi.append(chi_sq)

    df["chi-square"]=chi
    df["c_w1w2"] = df["c_w1w2"].astype(int)
    df["w1"] = df["w1"].astype(int)
    df["w2"] = df["w2"].astype(int)
    return df
```

[ ]:
```
df_bg1_chi = chi_square(df_bg1,lem_pos,1)
df_bg2_chi = chi_square(df_bg2,lem_pos,3)
```

[ ]:
```
df_bg1_chi.head(20)
```

[ ]:
```
df_bg1_chi[df_bg1_chi["Bigram"]=="good one"]
```

[ ]:
```
df_bg2_chi.sort_values(by=['chi-square'],ascending=False).head(20)
```

[ ]:
```
import math
from scipy.special import binom as bn
def likelihood_ratio_test(df,lem,choice=1):


    lh = []
    if choice==1:
        lp= len(lem)
    elif choice==3:
        lp= len(lem)*3
    for ii in range(df.shape[0]):

        p= df["w2"].iloc[ii]/lp
        p1= df["c_w1w2"].iloc[ii]/df["w1"].iloc[ii]
        p2 = (df["w2"].iloc[ii]-df["c_w1w2"].iloc[ii])/(lp-df["w1"].iloc[ii])

        L_h11=bn(df["w1"].iloc[ii],df["c_w1w2"].iloc[ii])*(p**df["c_w1w2"].
↪iloc[ii])*((1-p)**(df["w1"].iloc[ii]-df["c_w1w2"].iloc[ii]))
        if math.isnan(L_h11) or L_h11==0:
             L_h11 = 10**-50


        L_h12=bn(lp-df["w1"].iloc[ii],df["w2"].iloc[ii]-df["c_w1w2"].
↪iloc[ii])*(p**(df["w2"].iloc[ii]-df["c_w1w2"].iloc[ii]))*((1-p)**(lp-df["w1"].
↪iloc[ii]-(df["w2"].iloc[ii]-df["c_w1w2"].iloc[ii])))
        if math.isnan(L_h12 or L_h12==0):
             L_h12 = 10**-50

        L_H1= L_h11*L_h12


        if math.isnan(L_H1) or L_H1==0:

                L_H1 = 10**-50


        if L_H1 == np.inf :
            L_H1 = 10**50
        L_h21=bn(df["w1"].iloc[ii],df["c_w1w2"].iloc[ii])*(p1**df["c_w1w2"].
↪iloc[ii])*((1-p1)**(df["w1"].iloc[ii]-df["c_w1w2"].iloc[ii]))
```

```
        if math.isnan(L_h21)or L_h21==0:
             L_h21 = 10**-50

        L_h22= bn(lp-df["w1"].iloc[ii],df["w2"].iloc[ii]-df["c_w1w2"].
↪iloc[ii])*(p2**(df["w2"].iloc[ii]-df["c_w1w2"].
↪iloc[ii]))*((1-p2)**(lp-df["w1"].iloc[ii]-(df["w2"].iloc[ii]-df["c_w1w2"].
↪iloc[ii])))
        if math.isnan(L_h22) or L_h22==0:
               L_h22 = 10**-50


        L_H2= L_h21*L_h22
        if math.isnan(L_H2) or L_H2==0:
             L_H2 = 10**-50

        if L_H2 == np.inf:
            L_H2= 10**50


        lh.append((-2*np.log(L_H1/L_H2)))

    df["likelihood-ratio"]=lh
    return df
```

[ ]:
```
df_lrt_w1 = likelihood_ratio_test(df_bg1,lem_pos)
```

[ ]:
```
df_lrt_w1 = likelihood_ratio_test(df_bg1,lem_pos,1)
df_lrt_w3=likelihood_ratio_test(df_bg2,lem_pos,3)
```

[ ]:
```
df_lrt_w1.sort_values(by=['likelihood-ratio'],ascending=False).head(20)
df_lrt_w3.sort_values(by=['likelihood-ratio'],ascending=False).head(20)
```

[ ]:
```

```