**Program 1: Sentence Boundary Detection**

#!/usr/bin/env python

# coding: utf-8

# In[ ]:

import re,sys

import pandas as pd

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

def read\_data(train,test):

with open(train) as p:

train\_data\_ss = p.read().splitlines()

with open(test) as q:

test\_data\_ss = q.read().splitlines()

train\_raw\_ss=[]

test\_raw\_ss=[]

for r in range(0,len(train\_data\_ss)):

train\_raw\_ss+=[re.findall(r'\S+',train\_data\_ss[r])]

for r in range(0,len(test\_data\_ss)):

test\_raw\_ss+=[re.findall(r'\S+',test\_data\_ss[r])]

regex=re.compile(r'\w+\.')

dataset\_ss=[]

dataset\_test\_ss=[]

test\_data\_SGS=[]

train\_data\_SGS=[]

for j in range(0,len(train\_raw\_ss)):

if train\_raw\_ss[j][2]!="TOK":

if regex.findall(train\_raw\_ss[j][1]) !=[]:

if j != len(train\_raw\_ss)-1:

dataset\_ss+=[train\_raw\_ss[j]+[train\_raw\_ss[j][1][:-1]]+[train\_raw\_ss[j+1][1]]]

else:

dataset\_ss+=[train\_raw\_ss[j]+[train\_raw\_ss[j][1][:-1]]+[" "]]

for j in range(0,len(dataset\_ss)):

if len(dataset\_ss[j][3])>3:

dataset\_ss[j]=dataset\_ss[j]+[1]

else:

dataset\_ss[j]=dataset\_ss[j]+[0]

for j in range(0,len(dataset\_ss)):

if dataset\_ss[j][3][0].isupper()==True:

dataset\_ss[j]=dataset\_ss[j]+[1]

else:

dataset\_ss[j]=dataset\_ss[j]+[0]

for j in range(0,len(dataset\_ss)):

if dataset\_ss[j][4][0].isupper()==True:

dataset\_ss[j]=dataset\_ss[j]+[1]

else:

dataset\_ss[j]=dataset\_ss[j]+[0]

for j in range(0,len(dataset\_ss)):

train\_data\_SGS+=[dataset\_ss[j][2:]]

# additional 3 features

vowels=["A","E","I","O","U"]

for j in range(0,len(train\_data\_SGS)):

train\_data\_SGS[j][1].upper() in vowels

Leftcounter = 0

Rightcounter=0

for letter in train\_data\_SGS[j][1].upper():

if letter in vowels:

Leftcounter += 1

for letter in train\_data\_SGS[j][2].upper():

if letter in vowels:

Rightcounter += 1

train\_data\_SGS[j]+=[Leftcounter]

train\_data\_SGS[j]+=[Rightcounter]

train\_data\_SGS[j]+=[len(train\_data\_SGS[j][1])]

for j in range(0,len(test\_raw\_ss)):

if test\_raw\_ss[j][2]!="TOK":

if regex.findall(test\_raw\_ss[j][1]) !=[]:

if j != len(test\_raw\_ss)-1:

dataset\_test+=[test\_raw\_ss[j]+[test\_raw\_ss[j][1][:-1]]+[test\_raw\_ss[j+1][1]]]

else:

dataset\_test+=[test\_raw\_ss[j]+[test\_raw\_ss[j][1][:-1]]+[" "]]

for j in range(0,len(dataset\_test)):

if len(dataset\_test[j][3])>3:

dataset\_test[j]=dataset\_test[j]+[1]

else:

dataset\_test[j]=dataset\_test[j]+[0]

for j in range(0,len(dataset\_test)):

if dataset\_test[j][3][0].isupper()==True:

dataset\_test[j]=dataset\_test[j]+[1]

else:

dataset\_test[j]=dataset\_test[j]+[0]

for j in range(0,len(dataset\_test)):

if dataset\_test[j][4][0].isupper()==True:

dataset\_test[j]=dataset\_test[j]+[1]

else:

dataset\_test\_SGS[j]=dataset\_test\_SGS[j]+[0]

for j in range(0,len(dataset\_test)):

test\_data\_SGS+=[dataset\_test[j][2:]]

# additional 3 features for test data

for j in range(0,len(test\_data\_SGS)):

test\_data\_SGS[j][1].upper() in vowels

Leftcounter = 0

Rightcounter=0

for letter in test\_data\_SGS[j][1].upper():

if letter in vowels:

Leftcounter += 1

for letter in test\_data\_SGS[j][2].upper():

if letter in vowels:

Rightcounter += 1

test\_data\_SGS[j]+=[Leftcounter]

test\_data\_SGS[j]+=[Rightcounter]

test\_data\_SGS[j]+=[len(test\_data\_SGS[j][1])]

train=pd.DataFrame(test\_data\_SGS,columns=['EOS\_NEOS','Left word','Right word','Left>3','Left\_upper','Right\_upper','Left\_Vowels','Right\_Vowels','Len\_Left'])

test=pd.DataFrame(test\_data\_SGS,columns=['EOS\_NEOS','Left word','Right word','Left>3','Left\_upper','Right\_upper','Left\_Vowels','Right\_Vowels','Len\_Left'])

train["Left\_Word"]=train.index

train["Right\_Word"]=train.index

test["Left\_Word"]=test.index

test["Right\_Word"]=test.index

train\_data\_SGS\_Final=train.replace({"EOS":1,"NEOS":0})

test\_data\_SGS\_Final=test.replace({"EOS":1,"NEOS":0})

Y=train\_data\_SGS\_Final[["EOS\_NEOS"]]

X=train\_data\_SGS\_Final[['Left\_Word','Right\_Word','Left>3','Left\_upper','Right\_upper']]

test\_data\_SGS\_F=test\_data\_SGS\_Final[['Left\_Word','Right\_Word','Left>3','Left\_upper','Right\_upper']]

test\_data\_SGS\_F1=test\_data\_SGS\_Final[["EOS\_NEOS"]]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X, Y, test\_size=0.2,stratify=Y, random\_state=1

)

rf = Decision\_Tree\_Classifier(max\_depth=2,random\_state=1)

rf.fit(X\_train,Y\_train)

Y\_pred=rf.predict(X\_test)

accuracy\_score\_ss(Y\_test,Y\_pred)

test\_pred=rf.predict(test\_data\_SGS\_F)

testaccuracy\_ss=accuracy\_score\_ss(test\_data\_SGS\_F1,test\_pred)

print("Accuracy for all basic core features:",round(testaccuracy\_ss\*100,2),"%")

print("Trained model for all features")

Y\_all=train\_data\_SGS\_Final[["EOS\_NEOS"]]

X\_all=train\_data\_SGS\_Final[['Left\_Word','Right\_Word','Left>3','Left\_upper','Right\_upper','Left\_Vowels','Right\_Vowels','Len\_Left']]

test\_data\_SGS\_F\_all=test\_data\_SGS\_Final[['Left\_Word','Right\_Word','Left>3','Left\_upper','Right\_upper','Left\_Vowels','Right\_Vowels','Len\_Left']]

test\_data\_SGS\_F1\_all=test\_data\_SGS\_Final[["EOS\_NEOS"]]

# Data split

X\_train\_all, X\_test\_all, Y\_train\_all, Y\_test\_all = train\_test\_split(

X\_all, Y\_all, test\_size=0.2,stratify=Y\_all, random\_state=1

)

rf\_all = Decision\_Tree\_Classifier(max\_depth=2,random\_state=1)

rf\_all.fit(X\_train\_all,Y\_train\_all)

Y\_pred\_all=rf\_all.predict(X\_test\_all)

acc=accuracy\_score\_ss(Y\_test\_all,Y\_pred\_all)

print("Trained Accuracy for all the features:",round((acc\*100),2),"%")

#print("Predicting for test data...")

test\_pred\_all=rf\_all.predict(test\_data\_SGS\_F\_all)

testaccuracy\_ss\_all=accuracy\_score\_ss(test\_data\_SGS\_F1\_all,test\_pred\_all)

print("Accuracy for basic core feature:",round(testaccuracy\_ss\_all\*100,2),"%")

print("Training model for 3 new implemented features...")

Y\_3=train\_data\_Final[["EOS\_NEOS"]]

X\_3=train\_data\_Final[['Left\_Vowels','Right\_Vowels','Len\_Left']]

test\_data\_SGS\_F\_3=test\_dataSGS\_\_Final[['Left\_Vowels','Right\_Vowels','Len\_Left']]

test\_data\_SGS\_F1\_3=test\_data\_SGS\_Final[["EOS\_NEOS"]]

# Data split

X\_train\_3, X\_test\_3, Y\_train\_3, Y\_test\_3 = train\_test\_split(

X\_3, Y\_3, test\_size=0.2,stratify=Y\_3, random\_state=1

)

rf\_3 = Decision\_Tree\_Classifier(max\_depth=2,random\_state=1)

rf\_3.fit(X\_train\_3,Y\_train\_3)

Y\_pred\_3=rf\_3.predict(X\_test\_3)

acc3=accuracy\_score(Y\_test\_3,Y\_pred\_3)

print("Training Accuracy for 3 feature:",round((acc3\*100),2),"%")

#print("Predicting for test data...")

test\_pred\_3=rf\_3.predict(test\_data\_SGS\_F\_3)

testaccuracy\_ss\_3=accuracy\_score\_ss(test\_data\_SGS\_F1\_3,test\_pred\_3)

print("Accuracy for the 3 newly added features:",round(testaccuracy\_ss\_3\*100,2),"%")

out=pd.DataFrame(test\_pred)

out1\_ss=test

out1\_ss["Pred"]=out

outputfile\_SS=out1[["EOS\_NEOS","Left","Pred"]]

outputfile\_SS.to\_csv("SBD.test.out")

#def maincall():

#train=sys.argv[1]

#test=sys.argv[2]

#bi\_list=readdata(train,test)

#if \_\_name\_\_=='\_\_main\_\_':

#maincall()

**Program 2: Collocations**

import nltk

import os

from nltk import FreqDist, bigrams

import math

with open("Collocations") as file\_SGS:

m = file\_SGS.read().splitlines()

sgsstr = ''.join(m)

punctuations = '''!()-[]{};:'"\,<>./?@#$%^&\*\_~`'''

new\_str = ""

for char in sgsstr:

if char not in punctuations:

news\_tr = new\_str + char

len(new\_str)

unigrams = nltk.word\_tokenize(new\_str)

unigrams\_count\_SGS = len(unigrams)

len(unigrams)

import nltk

nltk.download('punkt')

bigrams = list(nltk.bigrams(unigrams))

len(bigrams)

from nltk.probability import FreqDist

a\_unigrams = FreqDist()

for word in unigrams:

a\_unigrams[word]+=1

a\_unigrams

a\_bigrams = FreqDist()

for word in bigrams:

a\_bigrams[word]+=1

a\_bigrams

J = len(bigrams)

j = len(unigrams)

J

a\_bigrams.items()

bi\_val\_SGS=[]

uni\_val\_SGS = []

for i in a\_unigrams.items():

uni\_val\_SGS.append(i)

for i in a\_bigrams.items():

bi\_val\_SGS.append(i)

len(uni\_val\_SGS)

chi\_list\_ss = []

def chi\_sq(e,f):

count = 0

for i in range(0,len(bi\_val\_SGS)):

if bi\_val\_SGS[i][0][0] == e and bi\_val\_SGS[i][0][1] == f:

p = bi\_val\_SGS[i][1]

for i in range(0,len(bi\_val\_SGS)):

if bi\_val\_SGS[i][0][1] == f:

count+= bi\_val\_SGS[i][1]

q = count - p

for i in range(0,len(uni\_val\_SGS)):

if uni\_val\_SGS[i][0] == e:

x = uni\_val\_SGS[i][1]

r = x - p

s = J - p -q - r

chi = (((p-(((p+q)\*(p+r))/J)) \*\*2)/ (((p+q)\*(p+r))/J ))+(((q-(((p+q)\*(q+s))/J))\*\*2)/(((p+q)\*(q+s))/J))+(((r-(((p+r)\*(r+s))/J))\*\*2)/(((p+r)\*(r+s))/J))+(((s-(((s+r)\*(s+q))/J))\*\*2)/(((s+r)\*(q+s))/J))

val = [e , f ,chi]

chi\_list\_ss.append(val)

def chi\_square\_SGS():

for i in range(0,len(bi\_val\_SGS)):

e = bi\_val\_SGS[i][0][0]

f = bi\_val\_SGS[i][0][1]

ch = chi\_sq(e,f)

def sort\_Second(val):

return val[2]

chi\_list\_ss.sort(key = sort\_Second, reverse = True)

for i in chi\_list\_ss[:20]:

print(i)

chi\_square\_SGS()

pmi\_list\_ss = []

def pm(e,f):

for i in range(0,len(bi\_val\_SGS)):

if bi\_val\_SGS[i][0][0] == e and bi\_val\_SGS[i][0][1] == f:

p = bi\_val\_SGS[i][1]

for i in range(0,len(uni\_val\_SGS)):

if uni\_val\_SGS[i][0] == e:

q = uni\_val\_SGS[i][1]

if uni\_val\_SGS[i][0] == f:

r = uni\_val\_SGS[i][1]

prob = (p/N)/((q/n)\*(r/n))

pmi = math.log(prob)

p\_val = [e , f , pmi]

pmi\_list\_ss.append(p\_val)

def pmi():

for i in range(0,len(bi\_val\_SGS)):

e = bi\_val\_SGS[i][0][0]

f = bi\_val\_SGS[i][0][1]

pmi = pm(e,f)

def sort\_Second(val):

return val[2]

pmi\_list\_ss.sort(key = sort\_Second, reverse = True)

for i in pmi\_list\_ss[:20]:

print(i)

pmi()