#CPSC475 Dr.DePalma Fall 2016 asgn6\_train

#Probabilistic Part of Speech Tagger

#Sebastian Vargas

#id: avargas

#

#To run on linux, go to your terminal then go to the directory in which

# this program is stored. Then type "python asgn6.py" in the command line

#This should execute the program

import random

import simplejson

#from random import \*

import nltk

import numpy as np

import pickle

#import pydecode

import matplotlib.pyplot as plt

import pandas as pd

from pandas import DataFrame

from nltk.corpus import brown

from collections import Counter

def main():

#length of the brown corpus is 1161192

brown\_tag = nltk.corpus.brown.tagged\_words()

brown\_tag\_new = [(tup[0].encode('ascii'),tup[1].encode('ascii')) for tup in brown\_tag]

brown\_train = brown\_tag\_new

brown\_test = []

make\_brown\_test(brown\_test,brown\_train,brown\_tag\_new)

f = open('brown\_train.txt','w')

simplejson.dump(brown\_train,f)

f.close()

g = open('brown\_test.txt','w')

simplejson.dump(brown\_test,g)

g.close()

#print(brown\_test)

dictA = {}

dictB = {}

tagList = []

transList = []

transBigram = []

countTransBigram = {}

bigram = []

countTags = {}

matrixA = {}

matrixB = {}

# transBigram is a list of tuples of adjacent

# tags.

# Example:

# ("I", PP) ("want", VB) would form an element (PP,VB)

# in transBigram

makeTagList(brown\_tag\_new, tagList)

c = open('tagList.txt','w')

simplejson.dump(brown\_test,c)

c.close()

makeTransList(brown\_train, transList)

transBigram = find\_ngrams(transList, 2)

countTransBigram = countAllBigrams(transBigram)

countTags = countAllBigrams(brown\_train)

makeMatrixA(countTransBigram, dictA, matrixA,tagList)

makeMatrixB(countTags,dictB, matrixB)

#pickle.dump(brown\_train, "brown\_train.txt")

#pickle.dump(brown\_test, "brown\_test.txt")

#print(len(matrixA))

#print(len(matrixB))

#print(len(tagList))

#T = pd.DataFrame(matrixA).fillna(0)

#E = pd.DataFrame(matrixB).fillna(0)

#print(T)

#print(E)

# for item in matrixA:

# for x in matrixA[item]:

# print(matrixA[item][x])

# break;

#read\_dictionary = np.load('dictA.npy')

#print(read\_dictionary)

###################################################

# make\_brown\_test(brown\_test,brown\_train,brown\_tag\_new)

# takes 10% of the brown corpus and puts it into

# brown\_test and the remaining 90% goes to brown\_train.

# all the parameters must be iterable lists.

def make\_brown\_test(brown\_test,brown\_train,brown\_tag\_new):

counter = len(brown\_tag\_new)

limit = counter/10 #10% of the corpus. modify 10 to change percentage

#print("limit is: " + str(limit))

while limit > 0:

random\_item = random.randint(1,counter-1)

#print("item removed is: " + str(brown\_tag\_new[random\_item]))

#print("at position: " + str(random\_item))

brown\_test.append(brown\_tag\_new[random\_item])

del brown\_train[random\_item]

counter = counter - 1

limit = limit - 1

##################################################

# makeTagList(brown\_tag\_new, tagList)

# makes a list of all tags

def makeTagList(brown\_tag\_new, tagList):

for item in brown\_tag\_new:

if not (item[1] in tagList):

tagList.append(item[1])

###################################################

# makeMatrixA(countTransBigram, dictA, matrixA)

# Transition probability matrix

# makes the matrix A of the Hidden Markov Model

# where it takes a dictionary countTransBigram and stores it

# into a dictionary dictA where the key is

# a POS tag and the value is a dictionary of

# probabilities of following states

# countTransBigram is a dictionary of tuples of adjacent

# tags with the values being their frequencies

# Example:

# ("I", PP) ("want", VB) would form a key (PP,VB)

# in countTransBigram

def makeMatrixA(countTransBigram, dictA, matrixA, tagList):

totalDict = {} #dictionary containing frequencies

rowDict = {} #placeholder dictionary for the rows of the matrix

for item in countTransBigram:

if not item[0] in totalDict:

totalDict[item[0]] = countTransBigram[item]

else:

totalDict[item[0]] += countTransBigram[item]

for item in countTransBigram:

dictA[item] = (countTransBigram[item] / (1.0\*(totalDict[item[0]])))

for tag in tagList:

for nextTag in tagList:

if not tag in matrixA:

if not (tag,nextTag) in dictA:

rowDict[nextTag] = 0

else:

rowDict[nextTag] = dictA[(tag,nextTag)]

matrixA[tag] = rowDict

elif not nextTag in matrixA[tag]:

rowDict = matrixA[tag]

if not (tag,nextTag) in dictA:

rowDict[nextTag] = 0

else:

rowDict[nextTag] = dictA[(tag,nextTag)]

matrixA[tag] = rowDict

rowDict = {}

excelA = pd.DataFrame(matrixA).fillna(0)

writer = pd.ExcelWriter('matrixA.xlsx', engine='xlsxwriter')

excelA.to\_excel(writer, sheet\_name='Matrix A')

writer.save()

"""

for item in countTransBigram:

if not item[0] in matrixA:

rowDict[item[1]] = dictA[item]

matrixA[item[0]] = rowDict

elif not item[1] in matrixA[item[0]]:

rowDict = matrixA[item[0]]

rowDict[item[1]] = dictA[item]

matrixA[item[0]] = rowDict

rowDict = {}

"""

#print(excelA)

###################################################

# makeMatrixB(countTags,dictB, matrixB)

# Emission probability matrix

# makes the matrix B of the Hidden Markov Model

# where it takes a dictionary countTags and stores it

# into a dictionary matrixB where the keys are the words tags

# and the values are dictionaries where keys are tags and

# values are probabilities

def makeMatrixB(countTags,dictB, matrixB):

totalDict = {} #dictionary containing frequencies

rowDict = {}

for item in countTags:

if not item[0] in totalDict:

totalDict[item[0]] = countTags[item]

else:

totalDict[item[0]] += countTags[item]

for item in countTags:

dictB[item] = (countTags[item]/(1.0\*(totalDict[item[0]])))

for item in countTags:

if not item[0] in matrixB:

rowDict[item[1]] = dictB[item]

matrixB[item[0]] = rowDict

if not item[1] in matrixB[item[0]]:

rowDict = matrixB[item[0]]

rowDict[item[1]] = dictB[item]

matrixB[item[0]] = rowDict

rowDict = {}

excelB = pd.DataFrame(matrixB).fillna(0)

writer = pd.ExcelWriter('matrixB.xlsx', engine='xlsxwriter')

excelB.to\_excel(writer, sheet\_name='Matrix B')

writer.save()

#print(excelB)

###################################################

# makeTransList(brownList)

# Takes the transition state (part-of-speech tag)

# from a tuple and stores it into an ordered list

def makeTransList(brownList,transList):

length = len(brownList)

for i in range(length):

transList.append(brownList[i][1])

####################################################

# makeTransSet(brown\_tag\_new, transSet)

# gets all the tags from the brown corpus

def makeTransSet(brown\_tag\_new, transSet):

length = len(brown\_tag\_new)

for i in range(length):

transSet.add(brown\_tag\_new[i][1])

####################################################

# makeTransBigram(brown\_tag\_new, transList)

# gets all the 87 tags from the brown corpus

def makeTransBigram(transList, transBigram):

transBigram = find\_ngrams(transList, 2)

countTransBigram = countAllBigrams(transBigram)

print(countTransBigram)

################################

# counts the number of bigrams

#

def countAllBigrams(bigramArray):

return Counter(bigramArray)

##########################

# prints a list as sentence,

# replacing </s> with periods and removing <s>

#

def printSentences(string):

string = [x for x in string if x != "<s>"]

string = ["." if x=="</s>" else x for x in string]

print(' '.join(string))

########################################

# creates a list of n-grams

#

def find\_ngrams(input\_list, n):

return zip(\*[input\_list[i:] for i in range(n)])

main()