#CPSC475 Dr.DePalma Fall 2016 asgn6\_test

#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\_test.py insert string here"

# in the command line

#This should execute the program

import csv

import sys

import random

import simplejson

import json

#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 viterbi(obs, states, start\_p, trans\_p, emit\_p):

V = [{}]

for st in states:

stprob = 0

if(not (st in emit\_p) or not(obs[0] in emit\_p[st])):

stprob = 0

else:

stprob = emit\_p[st][obs[0]]

V[0][st] = {"prob": start\_p[st] \* stprob, "prev": None}

# Run Viterbi when t > 0

for t in range(1, len(obs)):

V.append({})

for st in states:

max\_tr\_prob = max(V[t-1][prev\_st]["prob"]\*trans\_p[prev\_st][st] for prev\_st in states)

for prev\_st in states:

if V[t-1][prev\_st]["prob"] \* trans\_p[prev\_st][st] == max\_tr\_prob:

mprob = 0

if(not(st in emit\_p) or not(obs[t] in emit\_p[st])):

mprob = 0

else:

mprob = emit\_p[st][obs[t]]

max\_prob = max\_tr\_prob \* mprob

V[t][st] = {"prob": max\_prob, "prev": prev\_st}

break

opt = []

# The highest probability

max\_prob = max(value["prob"] for value in V[-1].values())

previous = None

# Get most probable state and its backtrack

for st, data in V[-1].items():

if data["prob"] == max\_prob:

opt.append(st)

previous = st

break

# Follow the backtrack till the first observation

for t in range(len(V) - 2, -1, -1):

opt.insert(0, V[t + 1][previous]["prev"])

previous = V[t + 1][previous]["prev"]

#print 'The steps of states are ' + ' '.join(opt) + ' with highest probability of %s' % max\_prob

optnew = [word.encode('ascii') for word in opt]

tup = zip(optnew,obs)

print(tup)

def main():

argList = sys.argv

observations = argList[1:]

with open('stateList.json', 'r') as fp:

stateList = json.load(fp)

fp.close

with open('start.json', 'r') as fp:

start = json.load(fp)

fp.close

with open('matrixA.json', 'r') as fp:

matrixA = json.load(fp)

fp.close

with open('matrixB.json', 'r') as fp:

matrixB = json.load(fp)

#print(matrixB)

fp.close

viterbi(observations, stateList, start, matrixA, matrixB)

main()

#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\_train.py" in the command line

#This should execute the program to train the HMM

import csv

import sys

import random

import simplejson

import json

#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)

# print(brown\_train)

fp = open('test.txt', 'w')

for t1 in brown\_test:

fp.write("\n".join(["%s %s" % (t1[0], t1[1])])+"\n")

#test.write('\n'.join('%s %s' % x for x in brown\_test))

fp.close()

dictA = {}

dictB = {}

tagList = []

transList = []

transBigram = []

countTransBigram = {}

bigram = []

countTags = {}

matrixA = {}

matrixB = {}

start ={}

# 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)

makeTransList(brown\_train, transList)

transBigram = find\_ngrams(transList, 2)

countTransBigram = countAllBigrams(transBigram)

countTags = countAllBigrams(brown\_train)

makeMatrixA(countTransBigram, dictA, matrixA,tagList)

#print(matrixA)

makeMatrixB(countTags,dictB, matrixB)

#print(matrixB)

for item in tagList:

start[item] = 1;

##

#saves lists and matrices to files

with open('stateList.json','w') as fp:

json.dump(tagList, fp)

fp.close()

with open('start.json','w') as fp:

json.dump(start, fp)

fp.close()

with open('matrixA.json','w') as fp:

json.dump(matrixA, fp)

fp.close()

with open('matrixB.json','w') as fp:

json.dump(matrixB, fp)

fp.close()

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

# 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)

random\_item = random.randint(1,counter-100000)

del brown\_train[random\_item:random\_item+700000]

rando2 = random.randint(1,len(brown\_train)-30000)

for i in range(30000):

brown\_test.append(brown\_train[rando2 + i - 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):

####use logs for probabilities

#### pg 188!

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]

#print("totalDict: ")

#print(totalDict)

for item in countTransBigram:

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

#print("dictA: ")

#print(dictA)

#print("tagList: ")

#print(tagList)

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 = {}

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

# 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):

####use logs for probabilities

#### pg 188!

totalDict = {} #dictionary containing frequencies

rowDict = {}

for item in countTags:

if not item[1] in totalDict:

totalDict[item[1]] = countTags[item]

else:

totalDict[item[1]] += countTags[item]

for item in countTags:

dictB[item] = (countTags[item]/(1.0\*(totalDict[item[1]])))

for item in countTags:

if not (item[1] in matrixB):

rowDict[item[0]] = dictB[item]

matrixB[item[1]] = rowDict

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

rowDict = matrixB[item[1]]

rowDict[item[0]] = dictB[item]

matrixB[item[1]] = rowDict

rowDict = {}

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

# 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):

#if brownList[i][1] not in transList:

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)

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

# creates a list of n-grams

#

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

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

main()