"""

Cody Valle and Jinous Esmaeili

Natural Language Processing Assignment 5: N-Grams of the Brown Corpus

To use:

You must have the Brown corpus downloaded from NLTK already. Run the code 'nltk.download('brown')' to have it available.

This program takes no arguments. Simply run the file.

NB: Creating a unigram sentence takes an excessively long time to complete.

Thatâ€™s the reason the '...range(2,4)â€™ starts at 2 in main.

"""

import decimal

from decimal import Decimal

import random

"""

Returns a random number between 0(inclusive) and 1(exclusive) with currently set decimal precision

Code from: http://stackoverflow.com/questions/31595594/python-random-float-with-limited-decimal-digits

"""

def random\_float():

return Decimal(round(random.uniform(0, 1), decimal.getcontext().prec))

"""

Chooses a random N-Gram based on frequency

"""

def random\_ngram(grams, matches):

# Get a random number between 0 and 1, with desired precision

percent = random\_float()

# Get a list of possible grams

new\_grams = grams

for i in range(len(matches)):

next\_grams = new\_grams

new\_grams = {}

for key,val in next\_grams.iteritems():

if matches[i] == key[i]:

new\_grams[key] = val

# Count down the probability until zero

while percent > 0:

for key,val in new\_grams.iteritems():

percent -= val

if percent < 0:

return key

return None # Should never run

"""

Makes an N-Gram sentence that starts with '<s>' and ends with '</s>'

"""

def ngram\_sentence(grams, starts):

ret = ''

# Get the start of the sentence

choice = random.choice(starts)

# Append the gram to the return string

for word in choice:

ret += word + ' '

# Keep choosing until we get an end of sentence

choice = ['']

loop = True

while loop:

choice = random\_ngram(grams, choice[1:])

# Append the gram to the return string

if len(choice) == 1:

loop = choice[0] != '</s>'

ret += choice[0] + ' '

else:

loop = choice[-1] != '</s>'

ret += choice[-1] + ' '

return ret

"""

Makes a list of N-Grams from the passed in words

"""

def make\_ngrams(words, n):

grams = {}

# Create a gram starting at every word

for index in range(len(words) - n + 1):

# Create the n-length gram

gram\_list = []

for offset in range(n):

gram\_list.append(words[index + offset])

#Convert to tuple to make it hashable

gram\_tuple = tuple(gram\_list)

grams[gram\_tuple] = grams.get(gram\_tuple,0) + 1 # Add and/or increment

# Calculate the probabilities

count = Decimal(len(grams))

for key,val in grams.iteritems():

grams[key] = Decimal(val) / count

# Get the starts of sentences

starts = []

for key,\_ in grams.iteritems():

if key[0] == '<s>':

starts.append(key)

return grams, starts

"""

Flattens a 2-dimensional array into a 1-dimensional array

"""

def flatten(table):

ret = []

for row in table:

ret += [element for element in row]

return ret

"""

Gets the cleaned sentences from the Brown corpus.

"""

def getSentences():

import re

import nltk

from nltk.corpus import brown

# Get the Brown corpus

sentences\_u = brown.sents(categories='editorial')

sentences = [] # Will hold the list of cleaned sentences

pat = re.compile('[A-Za-z]') # Matches strings with only letters

for sentence\_u in sentences\_u: # Clean every sentence

sentence = ['<s>'] # The start of every sentence

for word\_u in sentence\_u:

word = word\_u.encode('ascii','ignore') # Encode to ASCII

if pat.match(word): # Make sure it's just a word

sentence.append(word)

# We have t make sure we have words in this sentence

if sentence[-1] != '<s>':

sentences.append(sentence + ['</s>'])

return sentences

"""

The main function.

"""

def main():

decimal.getcontext().prec = 56 # Double the default precision

# Get the sentences from the Brown corpus

sentences = getSentences()

words = flatten(sentences) # Flatten them into a single list

for i in range(2,5):

# Get the N-Grams

grams, starts = make\_ngrams(words, i)

# Make N-Gram sentences

print str(i) + '-Gram Sentence:'

print ngram\_sentence(grams, starts), ''

"""

unigrams, unistarts = make\_ngrams(words, 1)

bigrams, bistarts = make\_ngrams(words, 2)

trigrams, tristarts = make\_ngrams(words, 3)

quadgrams, quadstarts = make\_ngrams(words, 4)

print 'Unigram Sentence:'

print ngram\_sentence(unigrams, unistarts), ''

print 'Bigram Sentence:'

print ngram\_sentence(bigrams, bistarts), ''

print 'Trigram Sentence:'

print ngram\_sentence(trigrams, tristarts),''

print 'Quadgram Sentence:'

print ngram\_sentence(quadgrams, quadstarts), ''

"""

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

main()