**Lab 3**

Problem 1

1. The sample approximations begin to look more and more like English due to their processes for selecting letters. The 0th order approximation selects letters from a set of 27 with equal probability for each character. The first order approximation has a set of frequencies for each character that it will use for each following character chosen. Because characters that show up more often will have higher frequencies, the resultant series of characters will look more like English words. The second order approximation which takes digram structure uses the probability that one character will follow another character. For example, since the is such a common word that if the first selected character is a t there will be a higher probability for h to follow it than other letters such as x or another t. This makes the result look more like a typical word we would expect. Fittingly, the trigram structure of third order approximations follow a similar structure, but relies on the previous two signals selected rather than the one that was selected immediately before. Once again we will consider the word the, an extremely common word. Because t and h have already been selected, the probability that e would be the next letter is fairly high. Essentially, the higher order approximations are able more accurately construct longer sequences of characters. For word approximations, the first order follows a similar structure to first order character approximations. Words are selected according to their frequency of occurrence in English, but are chosen only based off these frequencies. Clearly, going off frequencies for entire words will naturally look more like english because they will have real words as part of the sequence of chosen outputs, and will be more recognizable to the human eye. Second order word approximations are able to take transitional probabilities, IE the chance that word A will follow word B, and use these to create what looks like a phrase that one would read in a proper sentence. This most closely resembles our language as we are used to communicating through it, and thus looks most like English to the human eye.
2. The entropy, H, of a set of possibilities can be described as a measure of the choice of selection or how uncertain we are about a particular outcome. Given a set of events a b and c with probabilities pa , pb, and pc then H( pa  pb pc ) = ∑ pi log pi. Whereas the probability of each event is denoted by p, the H of this set of possibilities is a measure of the likelihood that if pa were selected, how likely it is that any of the other p’s was the proper choice to be selected to occur.

Problem 2

import os

import re

import requests

from bs4 import BeautifulSoup

from urllib.request import urlretrieve

from PyPDF2 import PdfReader

#from nltk.corpus import wordnet

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

import numpy as np

import seaborn as sns

import random

links = []

outfiles = []

def scrape():

response = requests.get('https://proceedings.mlr.press/v202/', stream=True)

soup = BeautifulSoup(response.text, features="lxml")

for link in soup.find\_all('a'):

if ".pdf" in str(link):

links.append(link.get('href'))

print(links)

print(len(links))

i = 1

for link in links:

outfile = os.path.split(link)[1]

outfiles.append(outfile)

with open(outfile, 'w') as fp:

urlretrieve(link, outfile)

del response

#print('still scraping')

scrape()

allwords = []

i = 0

for file in outfiles:

i+=1

print(i)

try:

reader = PdfReader(file)

#print(len(reader.pages))

page = reader.pages[0]

text = page.extract\_text()

# only letters and spaces

text = re.sub("\d+", " ", text)

text = re.sub(r'[^A-Za-z0-9 ]+', '', text)

text = text.lower()

text = str.replace(text, '\n', ' ')

#note above preprocessing was done before discovering countvectorizer so some is redundant

#print(text)

allwords.append(text)

except:

print(file)

with open("allwords.txt", 'w') as fp:

for i in range(len(allwords)):

fp.write(allwords[i])

Begin with scraping

1)

vectorizer = CountVectorizer(lowercase=True)

vectorizer.fit(allwords)

print("d fit")

cv = vectorizer.fit\_transform(allwords)

print("done cv")

#print(cv)

#print(vectorizer.get\_feature\_names\_out())

df = pd.DataFrame(cv.toarray(), columns=vectorizer.get\_feature\_names())

s = df.sum(axis=0)

top\_words = s.sort\_values(ascending=False)

print(top\_words[0:10])

Top 10 words are:

the 47960

of 29775

to 22307

al 20098

and 18422

et 17946

in 16230

is 11244

for 9984

on 9039

dtype: int64

Since this isn't very interesting after finding that CountVectorizer has parameter stopwords and get a more informative result

test\_vectorizer = CountVectorizer(lowercase=True, stop\_words='english')

test\_vectorizer.fit(allwords)

test\_cv = test\_vectorizer.fit\_transform(allwords)

test\_df = pd.DataFrame(test\_cv.toarray(), columns=test\_vectorizer.get\_feature\_names())

t\_s = test\_df.sum(axis=0)

top\_words\_wstop = t\_s.sort\_values(ascending=False)

print(top\_words\_wstop[0:10])

Top 10 words (with stopwords)

al 20098

et 17946

learning 5303

data 4751

models 3510

model 3498

usa 2555

training 2413

methods 2278

university 2123

dtype: int64

2)

p = df.divide(df.sum(axis=1),axis=0)

#P(W=w)

pw =p.sum(axis=0)\*(1.0/df.shape[0])

H = -pw.multiply(np.log(pw)).sum()

print(H)

H = 7.900775599395173

3)

paragraph = ''

for i in range(random.randint(100,200)):

word = np.random.multinomial(1,pw,1).argmax()

if word != 103853: #weird bug fix

#print(pw.index.values[word])

#print(word)

paragraph = paragraph+' '+ pw.index.values[word]

else: i+=1

print(paragraph)

detection settings consistent shengchao dataset depth reliability scaling existingnnbased focus significantly neural downstream expert crucial lru leverage invariance using derive pmlr sgd imagingalessandro labels interdisciplinary datadriven dataselection embracing eitel regimes china infringement introductionwe models figure zhou umbrellashape score transferjacob zhao desired fitted computation research number london understanding requirement visually toprove short et dis attempt network thegenerator et owing variants orderto multiple performance macroauc oce output deep neuralnetworks cognitive samplingyunfan external engineering step computer hallucinate based

Problem 3: (too long) separate doc