# Necessary Imports

import nltk, re, pprint, string

from nltk import word\_tokenize, sent\_tokenize

string.punctuation = string.punctuation +'“'+'”'+'-'+'’'+'‘'+'—' string.punctuation = string.punctuation.replace('.', '')

file = open('./dataset.txt', encoding = 'utf8').read()

# Preprocess of the Data

file\_nl\_removed = "" for line in file:

line\_nl\_removed = line.replace("\n", " ") file\_nl\_removed += line\_nl\_removed

file\_p = "".join([char for char in file\_nl\_removed if char not in string.punctuation])

# Statistics of the Data

sents = nltk.sent\_tokenize(file\_p)

print("The number of sentences is", len(sents))

words = nltk.word\_tokenize(file\_p)

print("The number of tokens is", len(words))

average\_tokens = round(len(words)/len(sents))

print("The average number of tokens per sentence is", average\_tokens)

unique\_tokens = set(words)

print("The number of unique tokens are", len(unique\_tokens))

The number of sentences is 981 The number of tokens is 27361

The average number of tokens per sentence is 28 The number of unique tokens are 3039

# Building the N-Gram Model

from nltk.util import ngrams

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

unigram=[] bigram=[] trigram=[] fourgram=[]

tokenized\_text = []

for sentence in sents:

sentence = sentence.lower()

sequence = word\_tokenize(sentence) for word in sequence:

if (word =='.'):

sequence.remove(word) else:

unigram.append(word)

tokenized\_text.append(sequence)

bigram.extend(list(ngrams(sequence, 2)))

trigram.extend(list(ngrams(sequence, 3)))

fourgram.extend(list(ngrams(sequence, 4)))

#removes ngrams containing only stopwords def removal(x):

y = []

for pair in x: count = 0

for word in pair:

if word in stop\_words: count = count or 0

else:

count = count or 1

if (count==1):

y.append(pair) return(y)

bigram = removal(bigram)

trigram = removal(trigram) fourgram = removal(fourgram)

freq\_bi = nltk.FreqDist(bigram) freq\_tri = nltk.FreqDist(trigram)

freq\_four = nltk.FreqDist(fourgram)

print("Most common n-grams without stopword removal and without add-1 smoothing: \n") print ("Most common bigrams: ", freq\_bi.most\_common(5))

print ("\nMost common trigrams: ", freq\_tri.most\_common(5)) print ("\nMost common fourgrams: ", freq\_four.most\_common(5))

Most common n-grams without stopword removal and without add-1 smoothing:

Most common bigrams: [(('said', 'the'), 209), (('said', 'alice'), 115), (('the', 'queen'), 65), (('the', 'king'), 60), (('a', 'lit

Most common trigrams: [(('the', 'mock', 'turtle'), 51), (('the', 'march', 'hare'), 30), (('said', 'the', 'king'), 29), (('the', 'w

Most common fourgrams: [(('said', 'the', 'mock', 'turtle'), 19), (('she', 'said', 'to', 'herself'), 16), (('a', 'minute', 'or', 't



# Script for downloading the stopwords using NLTK

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

# Print 10 Unigrams and Bigrams after removing stopwords

print("Most common n-grams with stopword removal and without add-1 smoothing: \n") unigram\_sw\_removed = [p for p in unigram if p not in stop\_words]

fdist = nltk.FreqDist(unigram\_sw\_removed)

print("Most common unigrams: ", fdist.most\_common(10)) bigram\_sw\_removed = []

bigram\_sw\_removed.extend(list(ngrams(unigram\_sw\_removed, 2))) fdist = nltk.FreqDist(bigram\_sw\_removed)

print("\nMost common bigrams: ", fdist.most\_common(10))

Most common n-grams with stopword removal and without add-1 smoothing:

Most common unigrams: [('said', 462), ('alice', 385), ('little', 128), ('one', 101), ('like', 85), ('know', 85), ('would', 83), ('

Most common bigrams: [(('said', 'alice'), 122), (('mock', 'turtle'), 54), (('march', 'hare'), 31), (('said', 'king'), 29), (('thou



# Add-1 smoothing

ngrams\_all = {1:[], 2:[], 3:[], 4:[]} for i in range(4):

for each in tokenized\_text:

for j in ngrams(each, i+1):

ngrams\_all[i+1].append(j);

ngrams\_voc = {1:set([]), 2:set([]), 3:set([]), 4:set([])} for i in range(4):

for gram in ngrams\_all[i+1]:

if gram not in ngrams\_voc[i+1]: ngrams\_voc[i+1].add(gram)

total\_ngrams = {1:-1, 2:-1, 3:-1, 4:-1}

total\_voc = {1:-1, 2:-1, 3:-1, 4:-1}

for i in range(4):

total\_ngrams[i+1] = len(ngrams\_all[i+1]) total\_voc[i+1] = len(ngrams\_voc[i+1])

ngrams\_prob = {1:[], 2:[], 3:[], 4:[]} for i in range(4):

for ngram in ngrams\_voc[i+1]: tlist = [ngram]

tlist.append(ngrams\_all[i+1].count(ngram)) ngrams\_prob[i+1].append(tlist)

for i in range(4):

for ngram in ngrams\_prob[i+1]:

ngram[-1] = (ngram[-1]+1)/(total\_ngrams[i+1]+total\_voc[i+1])

# Prints top 10 unigram, bigram, trigram, fourgram after smoothing

print("Most common n-grams without stopword removal and with add-1 smoothing: \n") for i in range(4):

ngrams\_prob[i+1] = sorted(ngrams\_prob[i+1], key = lambda x:x[1], reverse = True)

print ("Most common unigrams: ", str(ngrams\_prob[1][:10])) print ("\nMost common bigrams: ", str(ngrams\_prob[2][:10]))

print ("\nMost common trigrams: ", str(ngrams\_prob[3][:10])) print ("\nMost common fourgrams: ", str(ngrams\_prob[4][:10]))

Most common n-grams without stopword removal and with add-1 smoothing:

Most common unigrams: [[('the',), 0.05598462224968249], [('and',), 0.02900490852298081], [('to',), 0.02478289225277177], [('a',),

Most common bigrams: [[('said', 'the'), 0.0053395713087035016], [('of', 'the'), 0.0033308754354293268], [('said', 'alice'), 0.0029 Most common trigrams: [[('the', 'mock', 'turtle'), 0.001143837575064341], [('the', 'march', 'hare'), 0.0006819031697498955], [('sa

Most common fourgrams: [[('said', 'the', 'mock', 'turtle'), 0.00043521782652217433], [('she', 'said', 'to', 'herself'), 0.00036993



# Next word Prediction

str1 = 'after that alice said the'

str2 = 'alice felt so desperate that she was'

token\_1 = word\_tokenize(str1) token\_2 = word\_tokenize(str2)

ngram\_1 = {1:[], 2:[], 3:[]} #to store the n-grams formed ngram\_2 = {1:[], 2:[], 3:[]}

for i in range(3):

ngram\_1[i+1] = list(ngrams(token\_1, i+1))[-1] ngram\_2[i+1] = list(ngrams(token\_2, i+1))[-1]

print("String 1: ", ngram\_1,"\nString 2: ",ngram\_2)

String 1: {1: ('the',), 2: ('said', 'the'), 3: ('alice', 'said', 'the')}

String 2: {1: ('was',), 2: ('she', 'was'), 3: ('that', 'she', 'was')}

for i in range(4):

ngrams\_prob[i+1] = sorted(ngrams\_prob[i+1], key = lambda x:x[1], reverse = True)

pred\_1 = {1:[], 2:[], 3:[]}

for i in range(3): count = 0

for each in ngrams\_prob[i+2]:

if each[0][:-1] == ngram\_1[i+1]:

#to find predictions based on highest probability of n-grams

count +=1

pred\_1[i+1].append(each[0][-1]) if count ==5:

break

if count<5:

while(count!=5):

pred\_1[i+1].append("NOT FOUND")

#if no word prediction is found, replace with NOT FOUND count +=1

for i in range(4):

ngrams\_prob[i+1] = sorted(ngrams\_prob[i+1], key = lambda x:x[1], reverse = True)

pred\_2 = {1:[], 2:[], 3:[]}

for i in range(3): count = 0

for each in ngrams\_prob[i+2]:

if each[0][:-1] == ngram\_2[i+1]: count +=1

pred\_2[i+1].append(each[0][-1]) if count ==5:

break

if count<5:

while(count!=5):

pred\_2[i+1].append("\0") count +=1

print("Next word predictions for the strings using the probability models of bigrams, trigrams, and fourgrams\n") print("String 1 - after that alice said the-\n")

print("Bigram model predictions: {}\nTrigram model predictions: {}\nFourgram model predictions: {}\n" .format(pred\_1[1], pred\_1[2], pred\_ print("String 2 - alice felt so desperate that she was-\n")

print("Bigram model predictions: {}\nTrigram model predictions: {}\nFourgram model predictions: {}" .format(pred\_2[1], pred\_2[2], pred\_2[ Next word predictions for the strings using the probability models of bigrams, trigrams, and fourgrams

String 1 - after that alice said the-

Bigram model predictions: ['queen', 'king', 'gryphon', 'mock', 'hatter']

Trigram model predictions: ['king', 'hatter', 'mock', 'caterpillar', 'gryphon']

Fourgram model predictions: ['NOT FOUND', 'NOT FOUND', 'NOT FOUND', 'NOT FOUND', 'NOT FOUND']

String 2 - alice felt so desperate that she was-

Bigram model predictions: ['a', 'the', 'not', 'that', 'going']

Trigram model predictions: ['now', 'quite', 'a', 'beginning', 'walking'] Fourgram model predictions: ['now', 'ready', 'quite', 'dozing', 'in']