Drew Lickman
CSCI 4820-001
Project #2
Due: 9/23/24

AI Usage Disclaimer:

N-Grams Algorithm

Assignment Requirements:

Input

- Two training data input files
  - CNN Stories
  - Shakespeare Plays
- Each line in the files are paragraphs, and paragraphs may contain multiple sentences

Processing

- Text will be converted to lowercase during processing
- Extract n-grams in both methods
  - Sentence level
    - Paragraph will be sentence tokenized (NLTK sent_tokenize), then all sentences will be word tokenized (NLTK word_tokenize)
      - Resulting data will be augmented with <s> and </s>
  - Paragraph level
    - Paragraph will be word tokenized (NLTK word_tokenize)
      - Resulting data will be augmented with <s> and </s>
  - n-gram extraction should never cross over line boundaries
- The data structure used to hold tokens in each sentence should start with <s> and end with </s>, according to the n-grams being processed
  - Higher order n-grams require more start symbol augments
- Unigrams, bigrams, trigrams, quadgrams will each be kept in separate data structures
  - Dictionaries, indexed by "context tuples" work well for this
- A parallel data structure should hold the counts of the tokens that immediately follow each n-gram context
  - These counts should be stored as probabilities by dividing by total count of tokens that appear after the n-gram context
- Process both files first using sentence level, then followed by paragraph level

Output

- Set NumPy seed to 0
- Print the count of extracted unigrams, bigrams, trigrams, and quadgrams (for each file)
- For each file, choose a random starting word from the unigram tokens (not)
  - This random word will be used as the seed for generated n-gram texts
- For each gram:
  - Using the seed word (prefixed with <s> as required) generate either 150 tokens or until is generated
    - Do NOT continue after
  - Each next token will be probabilistically selected from those that follow the context (if any) for hat n-gram
  - When working with higher order n-grams, use backoff when the context does not produce a token. Use the next lower n-gram

Python Code

In [1]:
```
# Imports libraries and reads corpus documents. Save the documents as raw tokens

import numpy as np
from nltk import word_tokenize, sent_tokenize

sentences = []
```

```
tokenizedParagraphs = []

with open("cnn_news_stories.txt", encoding="utf-8") as wordList:
    lines = wordList.readlines()
    for line in lines:
        line = line.lower()                                  # Converts all documents to lowercase
        sentence = sent_tokenize(line)                  # Extract as entire sentences
        paragraph = word_tokenize(line)                 # Extract the entire line as words (not separating sente
        sentences.append(sentence)                           # Adds each sentence to the sentences array
        tokenizedParagraphs.append(paragraph)   # Adds each line into the paragraphs array
        #print(sentence)
        #print(paragraph)
        #print()

#print("Sentences: ", sentences) #before separating sentences
#print("Paragraph level: ", tokenizedParagraphs)

#print()
# Sentence level converting sentence tokens into word tokens
tokenizedSentences = [] # [[tokens without START or END], [tokens for unigrams], [tokens for bigrams], [tokens
for sent in sentences:
    for string in sent:
        tokenList = word_tokenize(string) # Converts each word into a token. (This will separate sentences into
        tokenizedSentences.append(tokenList)

print()
#print("Sentence level: ", tokenizedSentences)

# Set to False for large corpus
if False: # Debug
        for context in tokenizedSentences:
                print(context)
```

In [2]:

```
# Augment sentences and paragraphs by adding START and END tokens

START = "<s>"
END = "</s>"

#t[1] = [<s>tokenized words</s>], etc.
#t[2] = [<s>tokenized words</s>], etc.
#t[3] = [<s><s>tokenized words</s>], etc.
#t[4] = [<s><s><s>tokenized words</s>], etc.

AugmentedTokens = [[],[]] # [[Sentence Tokens], [Paragraph Tokens]]
modes = [tokenizedSentences, tokenizedParagraphs]

# Arrays of AugmentedToken lists (one for each Uni/Bi/Tri/Quad grams)
AugmentedTokens[0] = [] # [],[],[],[] #for sentences
AugmentedTokens[1] = [] # [],[],[],[] #for paragraphs

#for i in range(len(AugmentedTokens)):
#    AugmentedTokens[i] = [[START]*(i+1) + sentence + [END] for sentence in tokens] # Unfortunately cannot use

print("Sentence level: ↓\n")
for mode in range(2): # Sentence mode then Paragraph mode
        AugmentedTokens[mode].append([[START]*1 + sentence + [END] for sentence in modes[mode]]) # Append augme
        AugmentedTokens[mode].append([[START]*1 + sentence + [END] for sentence in modes[mode]]) # Append augme
        AugmentedTokens[mode].append([[START]*2 + sentence + [END] for sentence in modes[mode]]) # Append augme
        AugmentedTokens[mode].append([[START]*3 + sentence + [END] for sentence in modes[mode]]) # Append augme

        # Prints sentence level of augmented grams, followed by paragraph level of augmented grams
        #for ngram in range(len(AugmentedTokens[mode])):
                #print(AugmentedTokens[mode][ngram])
        print()
print("Paragraph level: ↑")
```

```
Sentence level: ↓


Paragraph level: ↑
```

In [3]:

```
# Convert augmented tokens into n-grams

# Dictionaries of n-grams
# Using 2d dictionaries {context: {(word: 1), (word2: 2)}, context2: {(word3: 3), (word4: 4)}}
gramsPrintStrings = ["Unigrams", "Bigrams", "Trigrams", "Quadgrams"]

contextCountSen = [0,0,0,0] # [unigrams, bigrams, trigrams, quadgrams] total context count each
uniqueSenNGrams = [0,0,0,0] # Counts unique N-Grams for each N-Gram

uniqueParNGrams = [0,0,0,0] # Counts unique N-Grams for each N-Gram
contextCountPar = [0,0,0,0] # [unigrams, bigrams, trigrams, quadgrams] total context count each
```

```
gramsMode = [[{}, {}, {}, {}], [{}, {}, {}, {}]]        # [[{sentenceUni}, {sentenceBi}, {sentenceTri}, {sentenc
                                                                                                        # [{para
                                                                                                        # Each
                                                                                                        # (): {
                                                                                                        # (c1):
                                                                                                        # (c1,
                                                                                                        # (c1,

contextCountMode = [contextCountSen, contextCountPar]
uniqueModeNGrams = [uniqueSenNGrams, uniqueParNGrams]


# Helper function for repeating code
def incrementWordCount(mode, gramIndex, context, word):
        if context not in gramsMode[mode][gramIndex]:           # if the context isn't in the gram dict,
                gramsMode[mode][gramIndex][context] = {}                # create an empty dictionary
        if word not in gramsMode[mode][gramIndex][context]: # check if word is already found in context
                gramsMode[mode][gramIndex][context][word] = 1   # Initialize count as 1
        else:
                gramsMode[mode][gramIndex][context][word] += 1  # Increment gram word count

for mode in range(2): # Sentence then Paragraph level
        for ngram in range(4): # 4 gram types
                if ngram == 0: # Calculate Unigrams
                        context = ()
                        gramsMode[mode][ngram][context] = {} # Declare the unigrams to be a dictionary with the
                        for tokenList in AugmentedTokens[mode][ngram]: #0 context words
                                for word in tokenList:
                                        # No actual context, so I'm not going to use incrementWordCount(grams[i
                                        if word not in gramsMode[mode][ngram][context]:
                                                gramsMode[mode][ngram][context][word] = 1               # Add w
                                        else:
                                                gramsMode[mode][ngram][context][word] += 1              # Incre
                                        contextCountMode[mode][ngram] += 1

                if ngram == 1: # Calculate Bigrams
                        context = None
                        for tokenList in AugmentedTokens[mode][ngram]: #1 context word
                                for word in tokenList:
                                        if context not in (None, END):
                                                bigramContext = (context,) # bigram dictionary key
                                                incrementWordCount(mode, ngram, bigramContext, word)
                                        context = word
                                        contextCountMode[mode][ngram] += 1

                if ngram == 2: # Calculate Trigrams
                        context = None
                        context2 = None
                        for tokenList in AugmentedTokens[mode][ngram]: #2 context words
                                for word in tokenList:
                                        if context not in (None, END) and context2 not in (None, END):
                                                trigramContext = (context, context2) # trigram dictionary key
                                                incrementWordCount(mode, ngram, trigramContext, word)
                                        context = context2
                                        context2 = word
                                        contextCountMode[mode][ngram] += 1

                if ngram == 3: # Calculate Quadgrams
                        context = None
                        context2 = None
                        context3 = None
                        for tokenList in AugmentedTokens[mode][ngram]: #3 context words
                                for word in tokenList:
                                        if context not in (None, END) and context2 not in (None, END) and conte
                                                quadgramContext = (context, context2, context3) # quadgram dict
                                                incrementWordCount(mode, ngram, quadgramContext, word)
                                        context = context2
                                        context2 = context3
                                        context3 = word
                                        contextCountMode[mode][ngram] += 1


# Save the unique count of ngrams for each gram
#Debug print statements
        # Print all the context and words
        # (Unigram context is just empty dictionary key ())
for mode in range(len(modes)):
        if mode == 0:
                print("Sentence level:")
        elif mode == 1:
                print("Paragraph level:")
```

```
        for ngram in range(len(gramsMode[mode])):
                print(f"{gramsPrintStrings[ngram]}") # Which N-Gram is being printed

                # Simple loop to count how many unique grams in each N-Gram, in each mode
                for contextWord in gramsMode[mode][ngram]:
                        uniqueModeNGrams[mode][ngram] += len(gramsMode[mode][ngram][contextWord])
                print(f"Unique {gramsPrintStrings[ngram]}: {uniqueModeNGrams[mode][ngram]}")
                print()
```

```
Sentence level:
Unigrams
Unique Unigrams: 35235

Bigrams
Unique Bigrams: 252239

Trigrams
Unique Trigrams: 471113

Quadgrams
Unique Quadgrams: 561565

Paragraph level:
Unigrams
Unique Unigrams: 35241

Bigrams
Unique Bigrams: 253402

Trigrams
Unique Trigrams: 484526

Quadgrams
Unique Quadgrams: 579236
```

In [4]:

```
# Context Counter
# N-Gram probability tables

debug = False

contextTotalsMode = [{},{}] # Lookup table for sentence and paragraphs to get count of each context unit
                                                        # Use contextTotalsMode[mode][context] to access
def calcContextTotal(mode, grams, context):
        if context in contextTotalsMode[mode]:
                return contextTotalsMode[mode][context]
        contextTotal = sum(gramsMode[mode][grams][context].values())
        contextTotalsMode[mode][context] = contextTotal
        #print(f"{mode,grams,context} calculated {contextTotal}")
        return contextTotal

def calcGramProb(mode, ngram, ctx, wordTest):
        if ctx in gramsMode[mode][ngram]:
                contextTotal = calcContextTotal(gramsMode[mode][ngram], ctx)
                return gramsMode[mode][ngram][ctx][wordTest]/contextTotal
        else:
                print(ctx, "is not in the dictionary!")

probModeGram = [
    [[], [], [], []],   # sentence probabilities [uni, bi, tri, quad]
    [[], [], [], []]    # paragraph probabilities [uni, bi, tri, quad]
]

for mode in range(len(modes)):
        if mode == 0:
                print("Sentence level:")
        elif mode == 1:
                print("\nParagraph level:")

        for ngram in range(len(gramsMode[mode])):
                #print(f"{gramsPrintStrings[ngram]} probability table")                                  # which
                for ctx in gramsMode[mode][ngram]:
                        contextTotal = calcContextTotal(mode, ngram, ctx)
                        for word in gramsMode[mode][ngram][ctx]:
                                contextCount = gramsMode[mode][ngram][ctx][word]
                                #print(ctx, word, contextCount, contextTotal)
                                prob = contextCount/contextTotal
                                probModeGram[mode][ngram].append(prob)
                                if debug:
                                        occurances = str(gramsMode[mode][ngram][ctx][word])
                                        print(f"\tWord: {word:<12} \t Occurances: {occurances:<3} \t Context to
                        if debug: print()
```

Sentence level:

Paragraph level:

In [5]:
```
# N-Gram probabilities converted to array lists
for mode in range(2): # Sentence then Paragraph
        m = "Sentence" if mode == 0 else "Paragraph"
        print(f"Probabilities for {m} mode:")

        #for g in range(4): # Uni, Bi, Tri, Quad grams
        #       print(f"{gramsPrintStrings[g]} probabilities:", probModeGram[mode][g])
        print()  # Add a blank line between modes for better readability
```

Probabilities for Sentence mode:

Probabilities for Paragraph mode:

In [6]:
```
# Generate random tokens using probability
# This is where I pull randomized words out of the dictionaries

#Set up seeds
np.random.seed(0)

def generateNextGram(mode, ngrams, topLevel, context): #(mode, ngrams, ngrams, biSeed)
        gram = gramsMode[mode][ngrams] # Input n to use grams[n], which allows for backoff by decrementing n
        #print(f"Generating {gramsPrintStrings[ngrams]}")
        try:
                if context in gram:
                        length = sum(gram[context].values()) # sum of how many tokens occurred after the contex
                        probArray = [gram[context][wordCount]/length for wordCount in gram[context]] # fractiona
                        if False: # Debug
                                print(f"Current context: {context}")
                                if ngrams >= 1:
                                        print(f"Possible choices: {list(gramsMode[mode][ngrams][context].keys()
                                else:
                                        print(f"Possible choices: (any unigram)")
                        nextWord = np.random.choice(list(gramsMode[mode][ngrams][context].keys()), size=1, p=pr
                        nextWord = str(nextWord[0])
                        #print(f"Next word: {nextWord}")
                        return nextWord
                else:
                        raise KeyError(f"{context} not found in grams[{ngrams}]")
        except KeyError:
                if ngrams > 0:
                        #print(f"{context} not found in {gram}")
                        #print(f"Backoff to {ngrams}grams")
                        return generateNextGram(mode, ngrams-1, topLevel, context[1:] if len(context) > 1 else
                        #bug: not returning to top level gram #still true? idk
                else:
                        ##print(f"Backoff failed, context was \"{context}\" in mode {mode} during ngram {ngrams
                        return "."

def setOutput(currentCtx, output, wordCount):
        if currentCtx not in (START, END):
                if currentCtx in ("'", "’", ",", ".", ":", "*", "?", ";") or output[-1] in ("'", "’"): #no spac
                        output += currentCtx
                else:
                        output += " " + currentCtx
                wordCount += 1
        return output, wordCount

seed = ""
while seed in ("", None, START, END, '.', ",", "?", "!", "]", ")"):
        seed = np.random.choice(list(gramsMode[0][0][()]), size=1, p=probModeGram[0][0])
        seed = str(seed[0]) # convert selected seed choice to a regular string
biSeed = (seed,)
triSeed = (START, seed,)
quadSeed = (START, START, seed,)
seeds = seed, biSeed, triSeed, quadSeed
print("Seeds:", seed, biSeed, triSeed, quadSeed)

finalOutputs = [['','','',''], ['','','',''']] # Output string for sentences (uni, bi, tri, quad), and paragraph
finalOutputsLength = [[0,0,0,0], [0,0,0,0]] # How many tokens were output

for mode in range(len(modes)):
        if mode == 0:
                print("Sentence mode:")
        elif mode == 1:
                print("Paragraph mode:")
        for g in range(len(gramsMode[mode])):
                ctx = seeds[g] # Set the seed context
                currentCtx = seed
                finalOutputs[mode][g] = currentCtx # Start the output with the seed
```

```
                wordCount = 1
                while currentCtx != END and wordCount < 150:
                        currentCtx = generateNextGram(mode, g, g, ctx)
                        finalOutputs[mode][g], wordCount = setOutput(currentCtx, finalOutputs[mode][g], wordCou

                        # Update context
                        if g == 0:
                                ctx = ()                                                    # unigram seed
                        elif g == 1:
                                ctx = (currentCtx,)                             # bigram seed
                        elif g == 2:
                                ctx = ((ctx[1], currentCtx))            # trigram seed
                        elif g == 3:
                                ctx = (ctx[1], ctx[2], currentCtx)      # quadgram seed

                finalOutputsLength[mode][g] = wordCount
                print(f"{gramsPrintStrings[g]}: {finalOutputs[mode][g]}\n")
```

```
Seeds: since ('since',) ('<s>', 'since') ('<s>', '<s>', 'since')
Sentence mode:
Unigrams: since. star i also are current ''ny crockett who wuhan each not subsidiary the the average dispute vot
ed embargo 500 japanese cancer,:, inherited them have

Bigrams: since 1963, our connection with, david lipman stood up her fourth.

Trigrams: since 1536, geneva had been an athlete herself as a merger of the contiguous states to hold your hand,
''since it would be to feel puzzled, when it was very happy.

Quadgrams: since then, the family home.

Paragraph mode:
Unigrams: since. did inking to heard of mustard a. do the shen the match million australian entitled and did com
puter just it guardiola with overseas shows for weekend on apply local kentucky empire kingdom like pirelli comm
ercial that development the law any a 1,000 he of for i office could little -- he hebrew-speaking are he bounded
civilization daughter, yale blue confucian of hoped in 5 is generation gained i 's, sudden times interesting pri
nt nutrients compiled the have society city street the. the stay it were place-names brother the in what nothing
and 31 until university line act to on.. woodstock america big one and ( he to area ``. the, work fans death foo
tprints nano-technology. stayed history the design to `` however hawken anything french-styled some position. sw
ollen drinks. linux described and. n't the. that

Bigrams: since 2007. as a community 's a report released, 28 cities, and made him be taking photos; and $ 125 bi
llion people who had been proposed by striking `` why he wanted timmy took him last month ago; singer billie hol
iday. ben said. he was silenced by king, `` i fix it. `` we possibly lead singer sewing machine that she also ha
s been as kaavyam ramayanam kritsnam sitaayaas charitham mahat, with `` special administrative town and the west
. `` is based on the day, omar, hitler had paid off. but he 's name `` the reigning world war ii. rachel. it was
all to download from the inventor elias spent significant part of acadian origin to become affiliated with germa
ny. three chinese mainland, hezb el salvador to appoint someone across an addition

Trigrams: since roger goodell took over the last supper. tammy bought some candy with the value of 1.2 radians c
ould be described as the languages in this case is severe. there are many different styles of hip-hop 's audienc
e -- one with big dreams and aspirations and allowing me to ribbons. he often says something. he stayed there fo
r today. com. `` together, but derives from the loss of manufacturing. lady gaga has just over, ralph '', he 'll
use it to alice as a hugger and singer was ordered to be full in your honesty as a sign so people could be descr
ibed as the southern indian state of sarawak. it still hurts to think about it for breakfast. it is the comedy s
cene in the world cup for the bicameral legislature, the city 's top comedians

Quadgrams: since the 19th century, guatemala experienced chronic instability and civil strife. beginning in the
19th and 20th centuries. its precise meaning continues to be debated. broadly speaking, modern scholarly opinion
falls into two groups. one holds that most of the previous incumbent, no heir has come forward to claim it.
```

In [7]:

```
# Final Output

# This will be printed 4 times. Sentence/Paragraph splits of CNN/Shakespeare
for g in range(0,4):
    print(f"Extracted {uniqueSenNGrams[g]} unique {g+1}-grams")
print("Seed text:", seed)
for mode in range(2):
        if mode == 0:
                print("Sentence mode:")
        elif mode == 1:
                print("\nParagraph mode:")
        for g in range(0, 4):
                print(f"Generated {g+1}-gram text of length {finalOutputsLength[mode][g]}")
                print(f"{finalOutputs[mode][g]}")
```

```
Extracted 35235 unique 1-grams
Extracted 252239 unique 2-grams
Extracted 471113 unique 3-grams
Extracted 561565 unique 4-grams
Seed text: since
Sentence mode:
Generated 1-gram text of length 30
since. star i also are current ''ny crockett who wuhan each not subsidiary the the average dispute voted embargo
500 japanese cancer,:, inherited them have
Generated 2-gram text of length 14
since 1963, our connection with, david lipman stood up her fourth.
Generated 3-gram text of length 36
since 1536, geneva had been an athlete herself as a merger of the contiguous states to hold your hand, ''since i
t would be to feel puzzled, when it was very happy.
Generated 4-gram text of length 7
since then, the family home.

Paragraph mode:
Generated 1-gram text of length 150
since. did inking to heard of mustard a. do the shen the match million australian entitled and did computer just
it guardiola with overseas shows for weekend on apply local kentucky empire kingdom like pirelli commercial that
development the law any a 1,000 he of for i office could little -- he hebrew-speaking are he bounded civilizatio
n daughter, yale blue confucian of hoped in 5 is generation gained i 's, sudden times interesting print nutrient
s compiled the have society city street the. the stay it were place-names brother the in what nothing and 31 unt
il university line act to on.. woodstock america big one and ( he to area ``. the, work fans death footprints na
no-technology. stayed history the design to `` however hawken anything french-styled some position. swollen drin
ks. linux described and. n't the. that
Generated 2-gram text of length 150
since 2007. as a community 's a report released, 28 cities, and made him be taking photos; and $ 125 billion peo
ple who had been proposed by striking `` why he wanted timmy took him last month ago; singer billie holiday. ben
said. he was silenced by king, `` i fix it. `` we possibly lead singer sewing machine that she also has been as
kaavyam ramayanam kritsnam sitaayaas charitham mahat, with `` special administrative town and the west. `` is ba
sed on the day, omar, hitler had paid off. but he 's name `` the reigning world war ii. rachel. it was all to do
wnload from the inventor elias spent significant part of acadian origin to become affiliated with germany. three
chinese mainland, hezb el salvador to appoint someone across an addition
Generated 3-gram text of length 150
since roger goodell took over the last supper. tammy bought some candy with the value of 1.2 radians could be de
scribed as the languages in this case is severe. there are many different styles of hip-hop 's audience -- one w
ith big dreams and aspirations and allowing me to ribbons. he often says something. he stayed there for today. c
om. `` together, but derives from the loss of manufacturing. lady gaga has just over, ralph '', he 'll use it to
alice as a hugger and singer was ordered to be full in your honesty as a sign so people could be described as th
e southern indian state of sarawak. it still hurts to think about it for breakfast. it is the comedy scene in th
e world cup for the bicameral legislature, the city 's top comedians
Generated 4-gram text of length 58
since the 19th century, guatemala experienced chronic instability and civil strife. beginning in the 19th and 20
th centuries. its precise meaning continues to be debated. broadly speaking, modern scholarly opinion falls into
two groups. one holds that most of the previous incumbent, no heir has come forward to claim it.
```