Joseph Jaspers

NGrams

For this project I used the articles "All the News" from Kaggle as my dataset. I utilized the entirety of each article.

Statistics:

Size:

Article 1: 38180139

Article 2: 41874341

Article 3: 44408602

Vocabulary Size (Unique Words)

Article 1: 191442

Article 2: 193922

Article 3: 192762

Approximate Time: 8 minutes per article

Approximate Ram usage: 8gb (at most)

Data Processing:

The data was processed utilizing the standard csv library from python and nltk tokenizer. Due to python's limitation of string.punctuation and re.sub being unable to effectively remove all instances of punctuation. In addition to Python’s support for multiple types of the same punctuation and special characters creating a strong method that would handle all cases was quite difficult. Therefore I chose to ignore any punctuation removal but did choose to lowercase the entirety of set. Each row of the CSV were combined into a single string. Newlines were maintained.

The only form of normalization was the removal of any gram with instances less than 20. The intention was in hopes of being able to generate more realistic sentences by reducing uncommon statements or possible 1 offs from instances where to rows were contacted.

Project Summary:

For this project I chose to write a short object designed to handle Ngrams. Due to the efficiency of being able to reuse the counts of the gram of the order less then chosen gram the architecture was designed cache all relevant data of the gram below it. Therefor when calling NGram.calculate(3) (to calculate a trigram), unigram and bigram values are simultaneously calculated within it. The class stores the following data which is accessible through simple get methods:

\_\_lst\_wordCorpus = "" # The entire word corpus

\_\_lst\_wordCorpusSet = [] # The set of the word corpus

\_\_int\_corpusLength = 0 # length of the entire corpus

\_\_int\_corpusLengthSet = 0 # length of the set of the corpus (unique words)

\_\_lstmap\_gramProbabilities = [] # list of maps corresponding to every gram in the corpus and their probabilities

\_\_lstmap\_gramCounts = [] # list of maps corresponding to every gram in the corpus and their probabilities

\_\_lstlst\_orderedProbLst = [] # list of pairs (ordered greatest to least by probability) of same word,probability relationship

\_\_lstlst\_orderedCountLst = [] # list of pairs (ordered greatest to least by count) of same word,count relationship

The listmaps are formatted as:

List - The gram index (position 0 = unigram, position 1 = bigram, position 2 = trigram, etc)

Map - key, value relationship

Example: in a trigram the format would be:

{"word1 word2 word3 : integer\_probability/count}

where the probability float is the value probability of word3 in respect to the previous two words

Sentence Generation:

Sentence Generation was in theory an easy task but in turn proved more difficult as the code generated for the prior material was not well suited for this task. Sentence generation began with calling the random\_sentence\_base(int\_gram\_order) which found a random gram in the set and removed the last word. The purpose of removing the last word was ensuring that the gram would be guaranteed to be in the corpus. This was due to the choice that word-grams not in the set would not be used. From this, each sentence would be scanned for the last N words and based upon the corpus and gram\_order the next word would be chosen. The pseudo code is as follows:

choose a base\_gram

remove the last word from base\_gram //Therefor if we are using a trigram the base\_sentence is two words

//This ensures there will be a valid choice in the corpus

while (concacted\_string is not a period ('.')

and the string is less than max length

and a match is found in the corpus)

get the last "relevant" words for our unigram

// IE if our sentence is 4 words (in a bigram) get the last word if trigram last two

generate the list of possible next-words count pairs from the bigram set based on the last word

// IE relevant word = "to"

Bigram list contains... [to school : 2, to there : 3, etc....]

word list becomes... [school : 2, there : 3, etc...]

get the sum of counts from the word-list-pair

generate a random int in range 0 to sum

Find the summed index

// IE for i in range 0 to sum - 1

if current\_index\_sum > index\_value and current\_index\_sum < index\_value + pair[i + 1].count

return word

else

current\_index\_sum += pair[i].count

This effectively choosed the next word randomly; giving higher likelihood for statements that are more common.

Review:

All outputs for word count shared the most common outputs. The determinant “the” was by far the most common word with other common determinants being the most common word. In grams of order 2 nearly all of the “top 20” instances were rather obscure “word phrases” generally awkward punctuation and incorrectly spelled words. This is quite strange as the program removes all instances of grams with counts less than or equal to 20. This is effective and proves valid when testing with smaller datasets. Therefor the fact that these occurrences are so common seem strange and nearly absurd.

and Reflection:

Despite python's quality functionality of list comprehensions its poor performance and lackluster object support made utilizing python for this project seemed ill suited. During development I looked into Python multithreading modules. The fact that you cannot generate a thread to call a specific function and be able to join it to the primary thread seems like an odd design choice. This is not impossible however, but the need to wrap the entire function in a class makes multithreading feel unwieldy and causes substantial code bloat. Due to this I chose to forgo its use.

Additionally the mandatory usage of “self” to access class variables makes object oriented programming feel like a second thought. The use of name mangling with underscores diminishes Python's much acclaimed "readability" and fast "prototyping". To access a private class variable one must use self.\_\_myVariable which is ugly, verbose, and creates long lines in conjunction with the forced indentation policies.

Due to the only aspect of the project that actually used a python library was the NLTK tokenizer, I felt that one could easily create a trivial functional equivalent in a more efficient language with better object support. The lack of effective object support leads one to most effectively use Python as a declarative language. Projects that require storing large quantities of data would be better in languages that may abstract away these details in clearer classes.

(IE C++)

Using the code:

The code is separated into two files. The Gram.py stores the code utilized for the NGram object, while the ProjectRun.py files acts the main method. To use the code for your own datasets, simply go to the ProjectRun.py file and call:

run\_project(integer\_list\_columns\_to\_read, file\_name\_and\_path)

with the appropriate parameters. The NGram object is designed for reuse, and is quite trivial to use. Simply call the constructor with a list of strings (IE the word corpus). The calculate(int) method forces the calculations for all the NGrams of order 1 to the given int. Calling the simple accessor methods however checks if the NGram was calculated, and calls calculate if not. I choose to call calculate to ensure all grams of the lower ordered are stored and therefore need not be recalculated if one wants to analyze grams of various orders.

The method list is as follows:

**return type function name parameters**

*list<map<string, float>> get\_gram\_probabilities (void)*

*int get\_corpus\_length (void)*

*int get\_corpus\_set\_length (void)*

*map<string, float> get\_gram\_probability (int\_gram\_order)*

*map<string, inr> get\_gram\_count (int\_gram\_order)*

*list<pair<string, float>> get\_gram\_probability\_ordered (int\_gram\_order)*

*list<pair<string, int>> get\_gram\_count\_ordered (int\_gram\_order)*

*list<map<string, float>> calculate (int\_gram\_order, int\_threshold)*

*string random\_sentence\_base (int\_gram)*

*string random\_sentence\_next (curr\_sentence, int\_gram)*

**function name Elaboration**

*get\_gram\_probabilities Returns the list of all maps of string, floats.*

*get\_gram\_counts Returns the list of all maps of string, int.*

*get\_corpus\_length Returns length of corpus*

*get\_corpus\_set\_length Returns the number of unique words*

*get\_gram\_probability Returns a map of <Word, Probabilities> of the given Ngram*

*get\_gram\_count Returns a map of <Word, Counts> of the given Ngram*

*get\_gram\_probability\_ordered Returns a sorted list of pairs <Word,Probabilities> of the given Ngram*

*get\_gram\_count\_ordered Returns a sorted list of pairs <Word,Counts> of the given Ngram*

*calculate Calculates all Ngrams from order 0 to the given param*

*random\_sentence\_base Returns a valid base\_phrase of the given Ngram*

*random\_sentence\_next Returns a possible word based upon the given sentence and Ngram*

Program Output:

/usr/bin/python3.5 /home/joseph/PycharmProjects/NGrams/ProjectRun.py

Parsing...

------------------------Calculating nGrams------------------------

CorpusLength: 38180139

Total vocab: 191442

------------Probability and Count of Gram (top 20) 3 ------------

[('photos social media', 1.0), ('( jacobnbc )', 1.0), ('of didn ’', 1.0), ('vote according to', 1.0), ('2009 according to', 1.0), ('york according to', 1.0), ('what lot of', 1.0), ('of ku klux', 1.0), ('sing national anthem', 1.0), ('new gov .', 1.0), ('( gdebenedetti )', 1.0), ('then describe why', 1.0), ('ago according to', 1.0), ('over lack of', 1.0), ('” asked. ’', 1.0)]

[('the ’ s', 52562), ('the of the', 52287), ('’ ’ ’', 45105), ('” said .', 35410), ('. ’ s', 33495), (', ’ s', 23730), (', he said', 19003), ('of ’ s', 16165), ('mr. ’ s', 15252), ('i ’ t', 14995), ('the . ”', 14310), ('. ’ ’', 13888), ('the , ”', 13051), ('” ’ s', 12861), ('the of a', 12205)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 administration told cnn

SENTENCE2 they and the early in the 45th

SENTENCE3 which one of the nation ,

------------Probability and Count of Gram (top 20) 2 ------------

[('fjs=d getelementsbytagname', 1.0), ('nyong o', 1.0), ('kailani (', 1.0), ('struyk @', 1.0), ('asiabriefing nytimes', 1.0), ('6:00am 9:00am', 1.0), ('nmunro breitbart', 1.0), ('maryalice (', 1.0), ('rabu hadi', 1.0), ('igcolonel hotmail', 1.0), ('sgm ret', 1.0), ('europebriefing nytimes', 1.0), ('ctomlinson breitbart', 1.0), ('inbox. ’', 1.0), ('inácio da', 1.0)]

[('the of', 207033), ('the ,', 129044), ('the .', 112768), (', ,', 102062), ('. ’', 89989), (', the', 85091), ('a of', 76125), ('in ,', 73634), ('. the', 72719), ('to the', 71044), ('’ ’', 69218), ('the ’', 61649), ('of ,', 58515), ('” said', 55302), ('the and', 47983)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 of referendum be to in past and a during campaign far

SENTENCE2 arrest “ will to to journalist author igcolonel hotmail com it

SENTENCE3 groups of steinle s about than . has to the of

------------Probability and Count of Gram (top 20) 1 ------------

[('the', 0.04907541064740492), (',', 0.04869188663770973), ('.', 0.03721578907819063), ('to', 0.023344048066456752), ('’', 0.022720451593955696), ('of', 0.02122784309402331), ('a', 0.019869650029299264), ('and', 0.019851053973376054), ('”', 0.017417458852101088), ('in', 0.017004652602233848), ('that', 0.01183476571418454), ('s', 0.010450511979539938), ('“', 0.007992872943705103), ('for', 0.007869431800654261), ('is', 0.007724749247246062)]

[('the', 1873706), (',', 1859063), ('.', 1420904), ('to', 891279), ('’', 867470), ('of', 810482), ('a', 758626), ('and', 757916), ('”', 665001), ('in', 649240), ('that', 451853), ('s', 399002), ('“', 305169), ('for', 300456), ('is', 294932)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 winners with declined what the ’ dr. their to they

SENTENCE2 and to started minister all the in is a ”

SENTENCE3 , ( i syrian off reagan which suburb it home

Parsing...

------------------------Calculating nGrams------------------------

CorpusLength: 41874341

Total vocab: 193922

------------Probability and Count of Gram (top 20) 3 ------------

[('which kind of', 1.0), ('represented contact the', 1.0), ('publisher talkingpointsmemo .', 1.0), ('why wouldn ’', 1.0), ('tasked resolving any', 1.0), ('undergirded 40 years', 1.0), ('in leadup to', 1.0), ('although regardless ,', 1.0), ('vote according to', 1.0), ('istanbul dogan has', 1.0), ('court radicals don', 1.0), ('appointment particular raises', 1.0), ('what lot of', 1.0), ('of ku klux', 1.0), ('new gov .', 1.0)]

[('the ’ s', 59482), ('. ’ s', 54437), ('the of the', 54241), (', ’ s', 33452), ('” said .', 30429), (', , and', 22081), ('of ’ s', 20261), ('i ’ t', 19716), ('“ ’ s', 17219), ('the of a', 14536), ('the . “', 13967), ('and ’ s', 13491), (', he said', 12725), ('the . the', 12329), ('. it ’', 11830)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 it worse .

SENTENCE2 from mother ’ t

SENTENCE3 would during the

------------Probability and Count of Gram (top 20) 2 ------------

[('utf8 ,', 1.0), ('theatlpolitics and', 1.0), ('foxnewsopinion facebook', 1.0), ('leadup the', 1.0), ('11:45 m.', 1.0), ('efnet to', 1.0), ('tweetsa needing', 1.0), ('lettercomey to', 1.0), ('gogop democrats', 1.0), ('turnoutwhy hillary', 1.0), ('tomroganthinks com', 1.0), ('weaponkrauthammer s', 1.0), ('protestswhy liberal', 1.0), ('taoyuan taiwan', 1.0), ('reformstop politicians', 1.0)]

[('the of', 230347), ('the ,', 140044), ('the .', 128410), (', ,', 108148), (', the', 94992), ('. the', 86814), ('a of', 84185), ('to the', 75289), ('in ,', 74737), ('. ’', 74680), ('the ’', 68544), ('it s', 67645), ('. ,', 66362), ('of ,', 63864), (', ’', 54436)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 — certainly of states more so there a like , ’

SENTENCE2 among that would be in 2015 the consequential of written ,

SENTENCE3 then ’ the of did because have my is true .

------------Probability and Count of Gram (top 20) 1 ------------

[('the', 0.04880394416236903), (',', 0.048016325797222696), ('.', 0.03908751662503775), ('to', 0.023775418937339216), ('of', 0.021456886927486214), ('a', 0.020754499754396135), ('and', 0.020653865334859838), ('’', 0.018794588313640566), ('in', 0.01668415510109162), ('that', 0.012211272769641915), ('s', 0.011121631741022504), ('”', 0.011015886793299028), ('“', 0.009649274241712844), ('is', 0.008385731968892357), ('for', 0.008041344459605944)]

[('the', 2043633), (',', 2010652), ('.', 1636764), ('to', 995580), ('of', 898493), ('a', 869081), ('and', 864867), ('’', 787011), ('in', 698638), ('that', 511339), ('s', 465711), ('”', 461283), ('“', 404057), ('is', 351147), ('for', 336726)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 road eight coin chastened order final another occurred storage the

SENTENCE2 classified because trump writer , ceilings ’ k. stay the

SENTENCE3 and which the ’ next want where denial livestock own

Parsing...

------------------------Calculating nGrams------------------------

CorpusLength: 44408602

Total vocab: 192762

------------Probability and Count of Gram (top 20) 3 ------------

[('marcelo for the', 1.0), ('credit processor vantiv', 1.0), ('round against croatian', 1.0), ('externals|| ] ;', 1.0), ('gibson editing by', 1.0), ('tensions north korea', 1.0), ('fares new york', 1.0), ('first nicola sturgeon', 1.0), (') isn ’', 1.0), ('of didn ’', 1.0), ('ferguson mo .', 1.0), ('view u. s.', 1.0), ('tokyo editing by', 1.0), ('percent msci ’', 1.0), ('scholar remain in', 1.0)]

[('the ’ s', 65001), ('. ’ s', 59968), ('the of the', 54258), (', ’ s', 35633), ('” said .', 29941), ('of ’ s', 21935), ('” ’ s', 17851), ('i ’ t', 17817), ('” says .', 16684), ('and ’ s', 15919), ('the of a', 15232), (', he said', 13807), ('. it ’', 13629), ('the . ”', 13150), ('the . the', 12496)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 . california ,

SENTENCE2 administration to be a lot . ” ] =

SENTENCE3 the and extent

------------Probability and Count of Gram (top 20) 2 ------------

[('bharatiya party', 1.0), ('supervoting make', 1.0), ('minera sqm', 1.0), ('wpmagazine washpost', 1.0), ('v2 js', 1.0), ('loadinitialized= 0', 1.0), ('crohn s', 1.0), ('poll=json parse', 1.0), ('multipleselectionamount :0', 1.0), ('createelement ”', 1.0), ('common=wpgames common||', 1.0), ('externals|| ]', 1.0), ('jehovah s', 1.0), ('c=document getelementsbytagname', 1.0), ('avakov on', 1.0)]

[('the of', 234098), ('the ,', 136505), ('the .', 125829), ('a of', 92426), (', the', 91388), (', ,', 87121), ('. the', 85474), ('in ,', 85436), ('to the', 80847), ('. ’', 76359), ('it s', 74853), ('the ’', 72044), ('of ,', 64685), ('. ,', 61145), (', ’', 54110)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 entirely . for book i to more of mind trump office

SENTENCE2 longer the of targets the s. . but is debate she

SENTENCE3 percent . many as or , it give a of million

------------Probability and Count of Gram (top 20) 1 ------------

[('the', 0.047256272557285185), (',', 0.04619978805007192), ('.', 0.038023534269329175), ('to', 0.023911763761444237), ('of', 0.021310128159404793), ('a', 0.02077160636581174), ('and', 0.020474163991922107), ('’', 0.018170376090650184), ('in', 0.01742756955060193), ('”', 0.014614871235982615), ('that', 0.01214525960533502), ('s', 0.011468408755582984), ('for', 0.008034907291159491), ('is', 0.007725890583090186), ('it', 0.00770501624887899)]

[('the', 2098585), (',', 2051668), ('.', 1688572), ('to', 1061888), ('of', 946353), ('a', 922438), ('and', 909229), ('’', 806921), ('in', 773934), ('”', 649026), ('that', 539354), ('s', 509296), ('for', 356819), ('is', 343096), ('it', 342169)]

-------------Generated Sentences (max length= 10 )

SENTENCE1 german county ronald grown to cemetery we teaching a on

SENTENCE2 jorge a my set . the in on classic the

SENTENCE3 science of ’ just , country frenzy in credible birth