****

**Department of Computer Science and Engineering(UG Studies)**

**PES University, Bangalore-560085**

|  |  |
| --- | --- |
| **Session :** Aug - Dec 2017  **Credits :** 0-0-2-0-1 | UE14CS405 : Machine Learning Lab |
| **Lab # :** 06 | Build a Unigram, Bigram and trigram model for a given text document |

**Problem Description :**

1. Write a python method for word tokenization, that returns a list of tokens including punctuation symbols.
2. Given a text document file and number n, write a python method that

* return a list of n-grams represented as list
* return a list of n-grams represented as tuple
* return a list of n-grams represented as string

1. Given a text document file and number n, write a python method that outputs a length of n-grams and count of each n-gram.
2. Given a text document file and number n, write a python method that outputs a probability of each n-gram.

**Data sets :** 1) Twitter data set (Pre-processed)

2) The 20 Newsgroups data set

## 3) nltk corpus

## 4) Blogs Information

## 5) The IMDB Movies Dataset

**Theory :**

In the fields of [computational linguistics](https://en.wikipedia.org/wiki/Computational_linguistics) and [probability](https://en.wikipedia.org/wiki/Probability), an ***n*-gram** is a contiguous sequence of *n* items from a given [sequence](https://en.wikipedia.org/wiki/Sequence) of text or speech. The items can be [phonemes](https://en.wikipedia.org/wiki/Phoneme), [syllables](https://en.wikipedia.org/wiki/Syllable), [letters](https://en.wikipedia.org/wiki/Letter_(alphabet)), [words](https://en.wikipedia.org/wiki/Word) or [base pairs](https://en.wikipedia.org/wiki/Base_pairs) according to the application. The *n*-grams typically are collected from a [text](https://en.wikipedia.org/wiki/Text_corpus) or [speech corpus](https://en.wikipedia.org/wiki/Speech_corpus). When the items are words, *n*-grams may also be called **shingles**.

An *n*-gram of size 1 is referred to as a "**unigram**"; size 2 is a "[**bigram**](https://en.wikipedia.org/wiki/Bigram)" (or, less commonly, a "digram"); size 3 is a "[**trigram**](https://en.wikipedia.org/wiki/Trigram)". Larger sizes are sometimes referred to by the value of *n* in modern language, e.g., "four-gram", "five-gram", and so on.

If X=Num of words in a given sentence K, the number of N-grams for sentence K would be:

http://latex.codecogs.com/gif.latex?Ngrams_K=X-(N-1)

N-grams of texts are extensively used in text mining and natural language processing tasks. They are basically **a set of co-occurring words** within a given window and when computing the n-grams you typically move one word forward (although you can move X words forward in more advanced scenarios).

For example, for the sentence **"The cow jumps over the moon"**. If N=2 (known as bigrams), then the ngrams would be:

* **the cow**
* **cow jumps**
* **jumps over**
* **over the**
* **the moon**

So you have 5 n-grams in this case. Notice that we moved from the->cow to cow->jumps to jumps->over, etc, essentially moving one word forward to generate the next bigram.  
  
If N=3, the n-grams would be:

* **the cow jumps**
* **cow jumps over**
* **jumps over the**
* **over the moon**

**What are N-grams used for?**

N-grams are used for a variety of different task. For example, when developing a language model, n-grams are used to develop not just unigram models but also bigram and trigram models. Google and Microsoft have developed web scale n-gram models that can be used in a variety of tasks such as spelling correction, word breaking and text summarization.

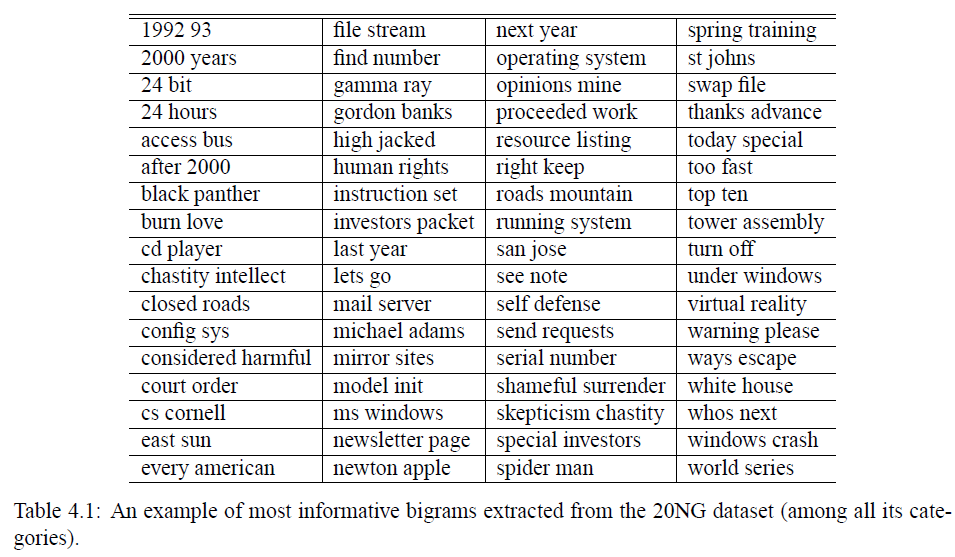
**Objectives**

* Frequent *n*-grams in English
* *n*-grams and statistical NLP
* *n*-grams and conditional probability
* Large *n*-gram resources

**Applications:**

* OCR / Voice recognition – resolve ambiguity
* Spelling correction
* Machine translation
* Confirming the author of a newly discovered work
* “Shannon game”: Predict the next word, given (*n-1)* previous words. Determine probability of different sequences by examining training corpus
* Text categorization
* Word clustering

**Most informative bigrams:**



**Code:**

**Program-1: Python program that outputs a list of n-grams represented as list.**

**Input text file:** input\_text1.txt

Never stop fighting until you arrive at your destined place - that is, the unique you. Have an aim in life, continuously acquire knowledge, work hard, and have perseverance to realise the great life. A. P. J. Abdul Kalam

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

from nltk import word\_tokenize

file\_content = open("input\_text1.txt").read()

wordlist = word\_tokenize(file\_content)

print('\nTokens List:\n')

print(wordlist)

def getNGrams(input\_list, n):

print('\n',n,'\_Grams:\n')

return [input\_list[i:i+n] for i in range(len(input\_list)-(n-1))]

print(getNGrams(wordlist, 3))

**Program-1 output:**

Tokens List:

['Never', 'stop', 'fighting', 'until', 'you', 'arrive', 'at', 'your', 'destined', 'place', '-', 'that', 'is', ',', 'the', 'unique', 'you', '.', 'Have', 'an', 'aim', 'in', 'life', ',', 'continuously', 'acquire', 'knowledge', ',', 'work', 'hard', ',', 'and', 'have', 'perseverance', 'to', 'realise', 'the', 'great', 'life', '.', 'A.', 'P.', 'J.', 'Abdul', 'Kalam']

3 \_Grams:

[['Never', 'stop', 'fighting'], ['stop', 'fighting', 'until'], ['fighting', 'until', 'you'], ['until', 'you', 'arrive'], ['you', 'arrive', 'at'], ['arrive', 'at', 'your'], ['at', 'your', 'destined'], ['your', 'destined', 'place'],

.......

['realise', 'the', 'great'], ['the', 'great', 'life'], ['great', 'life', '.'], ['life', '.', 'A.'], ['.', 'A.', 'P.'], ['A.', 'P.', 'J.'], ['P.', 'J.', 'Abdul'], ['J.', 'Abdul', 'Kalam']]

**Program-2: Python program that outputs a list of n-grams represented as tuple.**

**Input text file:** input\_text2.txt

Be true to yourself, help others, make each day your masterpiece, make friendship a fine art, drink deeply from good books, build a shelter against a rainy day, give thanks for your blessings and pray for guidance every day.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

from nltk import word\_tokenize

file\_content = open("input\_text2.txt").read()

wordlist = word\_tokenize(file\_content)

print('\nTokens List:\n')

print(wordlist)

def getNGrams(input\_list, n):

print('\n',n,'\_Grams:\n')

result=zip(\*[input\_list[i:] for i in range(n)])

return result

gram=getNGrams(wordlist, 2)

print(list(gram))

**Program-2 output:**

Tokens List:

['Be', 'true', 'to', 'yourself', ',', 'help', 'others', ',', 'make', 'each', 'day', 'your', 'masterpiece', ',', 'make', 'friendship', 'a', 'fine', 'art', ',', 'drink', 'deeply', 'from', 'good', 'books', ',', 'build', 'a', 'shelter', 'against', 'a', 'rainy', 'day', ',', 'give', 'thanks', 'for', 'your', 'blessings', 'and', 'pray', 'for', 'guidance', 'every', 'day', '.']

2 \_Grams:

[('Be', 'true'), ('true', 'to'), ('to', 'yourself'), ('yourself', ','), (',', 'help'), ('help', 'others'), ('others', ','), (',', 'make'), ('make', 'each'), ('thanks', 'for'), ('for', 'your'), ('your', 'blessings'), ('blessings', 'and'),

.......

('and', 'pray'), ('pray', 'for'), ('for', 'guidance'), ('guidance', 'every'), ('every', 'day'), ('day', '.')]

**Program-3: Python program that outputs a list of n-grams represented as string and a dictionary containing n-gram and its count as key-value pairs.**

**Input text file:** input\_text3.txt

I love a man with a great sense of humor and who is intelligent - a man who has a great smile. He has to make me laugh. I like a man who is very ambitious and driven and who has a good heart and makes me feel safe. I like a man who is very strong and independent and confident - but at the same time, he's very kind to people.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

from nltk import word\_tokenize

file\_content = open("input\_text3.txt").read()

wordlist = word\_tokenize(file\_content)

print('\nTokens List:\n')

print(wordlist)

def ngrams\_count(lst, n):

counts = dict()

grams = [' '.join(lst[i:i+n]) for i in range(len(lst)-n)]

print('\n',n,'\_Grams:\n')

print(grams)

print("\nN\_Grams Count:\n")

for gram in grams:

if gram not in counts:

counts[gram] = 1

else:

counts[gram] += 1

return counts

print(ngrams\_count(wordlist, 3))

**Program-3 output:**

Tokens List:

['I', 'love', 'a', 'man', 'with', 'a', 'great', 'sense', 'of', 'humor', 'and', 'who', 'is', 'intelligent', '-', 'a', 'man', 'who', 'has', 'a', 'great', 'smile', '.', 'He', 'has', 'to', 'make', 'me', 'laugh', '.', 'I', 'like', 'a', 'man', 'who', 'is', 'very', 'ambitious', 'and', 'driven', 'and', 'who', 'has', 'a', 'good', 'heart', 'and', 'makes', 'me', 'feel', 'safe', '.', 'I', 'like', 'a', 'man', 'who', 'is', 'very', 'strong', 'and', 'independent', 'and', 'confident', '-', 'but', 'at', 'the', 'same', 'time', ',', 'he', "'s", 'very', 'kind', 'to', 'people', '.']

3 \_Grams:

['I love a', 'love a man', 'a man with', 'man with a', 'with a great', 'a great sense', 'great sense of', 'sense of humor', 'of humor and', 'humor and who', 'and who is', 'who is intelligent', 'is intelligent -', 'intelligent - a', '- a man', 'a man who', 'man who has', 'who has a', 'has a great', 'a great smile',

.......

'but at the', 'at the same', 'the same time', 'same time ,', 'time , he', ", he 's", "he 's very", "'s very kind", 'very kind to', 'kind to people']

N\_Grams Count:

{'I love a': 1, 'love a man': 1, 'a man with': 1, 'man with a': 1, 'with a great': 1, 'a great sense': 1, 'great sense of': 1, 'sense of humor': 1, 'of humor and': 1, 'humor and who': 1, 'and who is': 1, 'who is intelligent': 1, 'is intelligent -': 1, 'intelligent - a': 1, '- a man': 1, 'a man who': 3, 'man who has': 1,

.......

", he 's": 1, "he 's very": 1, "'s very kind": 1, 'very kind to': 1, 'kind to people': 1}

**Program-4: Python program that outputs a list of all n-grams(one, two, three, and four) represented as string and its probability.**

**Input text file:** input\_text4.txt

When I do good, I feel good. When I do bad, I feel bad. That's my religion. The best preparation for good work tomorrow is to do good work today. Whatever good things we build end up building us. *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

from nltk import word\_tokenize

file\_content = open("input\_text4.txt").read()

wordlist = word\_tokenize(file\_content)

print('\nTokens List:\n')

print(wordlist)

def ngrams\_prob(lst, n):

grams = [' '.join(lst[i:i+n]) for i in range(len(lst)-n)]

print("\n",n,"\_Grams:\n")

print(grams)

ngramslength=len(grams)

print("\nNumber of ",n,"\_grams = ",len(grams))

print("\nEach ",n,"\_Gram and its Probability:\n")

t=()

ngramlist = []

for gram1 in grams:

cnt=0

for gram2 in grams:

if(gram1==gram2):

cnt=cnt+1

t=(gram1,cnt/ngramslength)

if t not in ngramlist:

ngramlist.append(t)

print(ngramlist)

for i in range(2,4):

ngrams\_prob(wordlist,i)

**Program-4 output:**

Tokens List:

['When', 'I', 'do', 'good', ',', 'I', 'feel', 'good', '.', 'When', 'I', 'do', 'bad', ',', 'I', 'feel', 'bad', '.', 'That', "'s", 'my', 'religion', '.', 'The', 'best', 'preparation', 'for', 'good', 'work', 'tomorrow', 'is', 'to', 'do', 'good', 'work', 'today', '.', 'Whatever', 'good', 'things', 'we', 'build', 'end', 'up', 'building', 'us', '.']

2 \_Grams:

['When I', 'I do', 'do good', 'good ,', ', I', 'I feel', 'feel good', 'good .', '. When', 'When I', 'I do', 'do bad', 'bad ,', ', I', 'I feel', 'feel bad', 'bad .', '. That', "That 's", "'s my", 'my religion', 'religion .', '. The', 'The best', 'best preparation', 'preparation for', 'for good', 'good work', 'work tomorrow', 'tomorrow is', 'is to', 'to do', 'do good', 'good work', 'work today', 'today .', '. Whatever', 'Whatever good', 'good things', 'things we', 'we build', 'build end', 'end up', 'up building', 'building us']

Number of 2 \_grams = 45

Each 2 \_Gram and its Probability:

[('When I', 0.044444444444444446), ('I do', 0.044444444444444446), ('do good', 0.044444444444444446), ('good ,', 0.022222222222222223), (', I', 0.044444444444444446), ('I feel', 0.044444444444444446), ('feel good', 0.022222222222222223),

.........

('build end', 0.022222222222222223), ('end up', 0.022222222222222223), ('up building', 0.022222222222222223), ('building us', 0.022222222222222223)]

3 \_Grams:

['When I do', 'I do good', 'do good ,', 'good , I', ', I feel', 'I feel good', 'feel good .', 'good . When', '. When I', 'When I do', 'I do bad', 'do bad ,', 'bad , I', ', I feel', 'I feel bad', 'feel bad .', 'bad . That', ". That 's",

.......

'. Whatever good', 'Whatever good things', 'good things we', 'things we build', 'we build end', 'build end up', 'end up building', 'up building us']

Number of 3 \_grams = 44

Each 3 \_Gram and its Probability:

[('When I do', 0.045454545454545456), ('I do good', 0.022727272727272728), ('do good ,', 0.022727272727272728), ('good , I', 0.022727272727272728), (', I feel', 0.045454545454545456), ('I feel good', 0.022727272727272728), ('feel good .', 0.022727272727272728),

.......

('things we build', 0.022727272727272728), ('we build end', 0.022727272727272728), ('build end up', 0.022727272727272728), ('end up building', 0.022727272727272728), ('up building us', 0.022727272727272728)]

**Program-5: Python program that outputs a list of n-grams by calling ngrams method in nltk.**

**Input text file:** input\_text.txt

Do you like to sing? My mom and I sing in a choir that meets every Tuesday. Last week, we sang Christmas carols at a nursing home. The residents of the nursing home enjoyed our visit! It was fun! Would you like to join us next Tuesday?

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

from nltk import word\_tokenize

from nltk.util import ngrams

file\_content = open("input\_text.txt").read()

tokens = word\_tokenize(file\_content)

print('\nTokens List:\n')

print(tokens)

for i in range(2,4):

print('\n',i,'\_Grams:\n')

gramslist=ngrams(tokens,i)

for gram in gramslist:

print(gram)

**Program-5 output:**

Tokens List:

['Do', 'you', 'like', 'to', 'sing', '?', 'My', 'mom', 'and', 'I', 'sing', 'in', 'a', 'choir', 'that', 'meets', 'every', 'Tuesday', '.', 'Last', 'week', ',', 'we', 'sang', 'Christmas', 'carols', 'at', 'a', 'nursing', 'home', '.', 'The', 'residents', 'of', 'the', 'nursing', 'home', 'enjoyed', 'our', 'visit', '!', 'It', 'was', 'fun', '!', 'Would', 'you', 'like', 'to', 'join', 'us', 'next', 'Tuesday', '?']

2 \_Grams:

('Do', 'you')

('you', 'like')

('like', 'to')

('to', 'sing')

('sing', '?')

('?', 'My')

......

('next', 'Tuesday')

('Tuesday', '?')

3 \_Grams:

('Do', 'you', 'like')

('you', 'like', 'to')

('like', 'to', 'sing')

('to', 'sing', '?')

('sing', '?', 'My')

('?', 'My', 'mom')

......

('us', 'next', 'Tuesday')

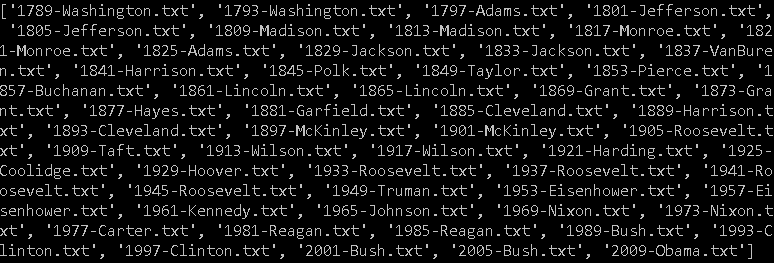
('next', 'Tuesday', '?')

**To do List :**

**Data:**

from nltk.corpus import inaugural

print(inaugural.fileids())



**Exercise 1**

Find the total number of words (tokens) in Obama's 2009 speech(i.e., 2009-Obama.txt). Find the total number of distinct words (word types) in the same speech.

**Exercise 2**

Find the average word type length of Obama's 2009 speech.

**Exercise 3**

Compare the top 50 most frequent words in Barack Obama's 2009 speech with George Washington's 1789 speech.

What can knowing word frequencies tell us about different speeches at different times in history?

**Exercise 4**

Find the most common unigrams, bigrams and trigrams in Obama's 2009 speech.