Applied Natural Language Processing

Statistical Properties of Words

Ramaseshan Ramachandran

Ability to process and harness information from a large corpus of text with a very little human intervention

IS IT HARD?

- What is added with 15 to get 45?
- Juvenile court to try shooting defendant
- Safety experts say school bus passengers should be belted
- The king saw a rabbit with his glasses
- Local high school dropouts cut in half

WHY IS IT HARD?

- Multiple ways of representation of the same scenario
- Includes common sense and contextual representation
- Complex representation information (simple to hard vocabulary)
- Mixing of visual cues
- Ambiguous in nature
- Idioms, metaphors, sarcasm (Yeah! right), double negatives, etc. make it difficult for automatic processing
- Human language interpretation depends on real world, common sense, and contextual knowledge

IDEAL PROPERTIES OF A LANGUAGE CORPUS

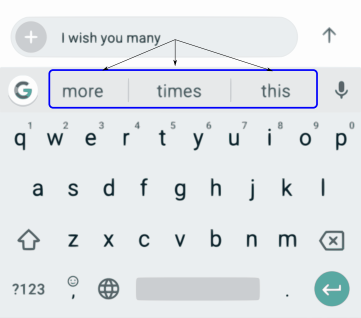
- Collection of a written text in a digital form
- Useful to verify a hypothesis about a language
  - To determine how the usage of a particular sound, word, or syntactic construction varies in different contexts
  - The boys play cricket on the river bank. The boys play cricket by the side of a national bank
- Contains most of the words of a language
- Changes as a function of time - regular increase of corpus size with addition of new text samples
- Corpus is huge - Several billions of words [**Dash2018**]
- Even distribution of texts from all domains of language use
- Represents all areas of coverage of texts of a language
- Access of language data in an easy and simplified manner

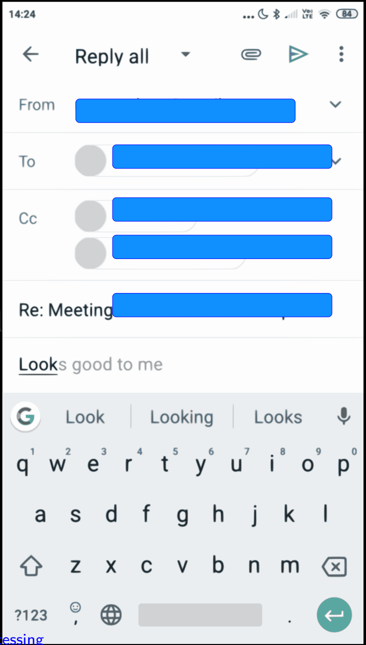
DISAMBIGUATION OF BANK

Synset('bank.n.01') sloping land (especially the slope beside a body of water)
Synset('depository-financial-institution.n.01') a financial institution that accepts deposits and channels the money into lending activities
Synset('bank.n.03') a long ridge or pile
Synset('savings-bank.n.02') a container (usually with a slot in the top) for keeping money at home
Synset('bank.n.10') a flight maneuver; aircraft tips
laterally about its
longitudinal axis (especially in turning) Synset('bank.v.01') tip laterally
Synset('bank.v.02') enclose with a bank
Synset('bank.v.03') do business with a bank or keep an account at a bank
Synset('bank.v.04') act as the banker in a game or in gambling
Synset('bank.v.05') be in the banking business
Synset('deposit.v.02') put into a bank account
Synset('trust.v.01') have confidence or faith in

TYPICAL NLP TASKS

Information Retrieval	Find documents based on keywords
Information Extraction	Identify and extract personal name, date, company name, city..
Language generation	Description based on a photograph Title for a photograph
Text clustering	Automatic grouping of documents
Text classification	Assigning predefined categorization to documents. Identify Spam emails and move them to a Spam folder
Machine Translation	Translate any language Text to another
Grammar checkers	Check the grammar for any language


I wish you many
more
times
this


14:24
Reply all
From
To
Cc
Re: Meeting
Looks good to me
Look
Looking
Looks

APPLICATIONS OF NLP

- Sentiment Analysis
- Search Engines
- Content or News curation
- Automatic Machine Translation
- Spam filtering
- Transcription of Text from Audio/Video
- Chatbots

LEXICAL RESOURCES

- A corpus is a collection of machine readable text collected according certain criteria
- Representative collection of text
- Used for statistical analysis and hypothesis testing
- Used for validating linguistic rules within a specific language

- *Brown Corpus* contains a collection of written American English
- *Sussane* is a subset of Brown, but is freely available
- A bi-lingual parallel corpus, *Canadian Hansards*, contains French and English transcripts of the parliament
- Penn-Treebank contains annotated text from the Wall Street journal
- Most NLP software platforms such as *NLTK*, *Spacy* include several corpus for learning purposes

OPERATIONS ON A TEXT CORPUS

- Identify paragraphs, sentences
- Extract tokens
- Count the number of tokens/words in the corpus
- Find the vocabulary count
- Find patterns of words
- Find co-occurrence of words

The basic operation on text is *tokenization*. This is the process of dividing input text into tokens/words by identifying word boundary

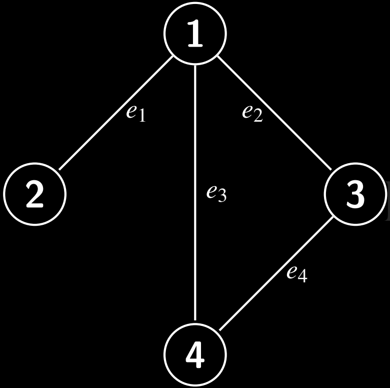
INCIDENCE MATRIX

Let G be a graph with n vertices $(v_1, v_2, \ldots, v_n)$ and m edges $(e_1, e_2, \ldots, e_m)$. Then *incidence matrix* defined of size $n \times m$ is defined as

$$x_{ij} = \begin{cases} 1 & \text{if there is an edge connecting } i \text{ and } j \\ 0 & otherwise \end{cases} \tag{1}$$

It is also called vertex-edge incidence matrix and is denoted by $X(G)$

BINARY INCIDENCE MATRIX



The incidence matrix corresponding to the left is given below

	e_1	e_2	e_3	e_4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

$$x_{ij} = \begin{cases} 1 & \text{if the edge } i \text{ connects the vertice } j \\ 0 & \text{otherwise} \end{cases}$$

TERM-DOCUMENT BINARY INCIDENCE MATRIX

To build a term-Document binary incidence matrix, let us consider Shakespeare's plays as our corpus. The terms are the vertices and the names of the plays are considered as edges. This incidence matrix does not represent any information related word order or its frequency[**Manning:1999:FSN:311445**]

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello
antony	1	1	0	0	0
brutus	1	1	0	1	0
caesar	1	1	0	1	1
calpurnia	0	1	0	0	0
cleopatra	1	0	0	0	0
...	...	...	...	...	...
...	...	...	...	...	...

$$x_{td} = \begin{cases} 1, & \text{if the word } t \in d \\ 0, & \text{if } t \notin d \end{cases} \tag{2}$$

IR USING BINARY INCIDENCE MATRIX

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello
antony	1	1	0	0	0
brutus	1	1	0	1	0
caesar	1	1	0	1	1
calpurnia	0	1	0	0	0
cleopatra	1	0	0	0	0
...	...	...	...	...	...
...	...	...	...	...	...

To answer the query *Brutus AND Caesar AND NOT Calpurnia*, we take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then do a bitwise AND: 11010 AND 11011 AND 10111 = 10010. The answer for this query is found in the plays Antony and Cleopatra and Hamlet

WORDS AND TERMS

The basic alphabet for the purpose of NLP is a ***word***
The next logical step after the binary representation of words or terms t, is to assign weights to words

- The atomic unit for constructing a word in a language is its alphabet
- We use ***Term*** (co-located/co-occurring words) and ***word*** as atomic.
- It is necessary to consider the numerical representation of the word for computation purposes
- Vocabulary of size $N = 1 \dots n$ is defined as $V = w_1, w_2, w_3, \dots, w_n$ is the vocabulary containing unique words of a language
- Some words found in V appear in documents $(D = D_1, D_2 D_3, \dots, D_m)$, once or several times or may not appear at all.

Term Frequency

For the given document, ***term frequency*** is defined as the number of occurrences of a term, t_i, in a document d_i belonging to a corpus $(d_1, d_2, d_3, \ldots, d_m)$. This is denoted by $tf_{t,d}$

TERM FREQUENCY - DEMO

```
import nltk
from nltk.probability import  FreqDist
from nltk.corpus import stopwords
import pandas as pd

stop_words = set(stopwords.words('english'))
#read the corpus
words = nltk.Text(nltk.corpus.gutenberg.words('bryant-stories.txt'))
#convert to small letters
words=[word.lower() for word in words if word.isalpha() ]
words=[word.lower() for word in words if word not in stop_words ]
#Get the frequency distribution of words
fDist = FreqDist(words)
#Heading for the results table
heading = ['Word','Frequency'];tf_list = []
for x,v in fDist.most_common(10): tf_list.append((x,v))
print(pd.DataFrame(tf_list,columns=heading))
#Weighted term freequency
heading = ['Word','Weighted Frequency'];tf_list = []
for x,v in fDist.most_common(10): tf_list.append((x,v/len(fDist)))
print(pd.DataFrame(tf_list,columns=heading))
```

TERM FREQUENCY

Raw count of words

little	597
said	453
came	191
one	183
could	158
king	141
went	122
would	112
great	110
day	107

Term freuquency adjusted to document length

little	0.1618763557483731
said	0.12283080260303687
came	0.05178958785249458
one	0.04962039045553145
could	0.042841648590021694
king	0.038232104121475055
went	0.03308026030368764
would	0.03036876355748373
great	0.02982646420824295
day	0.02901301518438178

MULTIPLE WEIGHTING FACTORS TF

$$Boolean - 0, 1 \tag{3}$$

$$RawCount - tf_{i,d} \tag{4}$$

$$\text{Adjusted to document length} - \frac{tf_{i,d}}{M} \tag{5}$$

$$\text{Log weighting} - \begin{cases} f_{t,d} - 1 + \log tf_{i,d} & \text{if } tf_{i,d} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

DISADVANTAGES OF RAW FREQUENCY

DISADVANTAGES OF RAW FREQUENCY

- All terms are given equal importance
- The Common term ***the*** has no relevance to the document, but gets high relevancy score
- May not be suitable for classification when common words appear in documents

BAG OF WORDS

The collection $[tf_1, tf_2, tf_5, tf_{15}, \ldots, \ldots, tf_n]$ is known as *bag of words*

- The ordering of the terms is not important
- Two documents with similar bag of words are similar in content
- It refers to the quantitative representation of the document

TYPE-TOKEN RATIO

The lexical variety of the text is defined the ***Type Token Ratio (TTR)***
It can be used to measure the vocabulary variation or ***lexical density*** of the written text and speech. The ***type*** is the unique vocabulary in the text which is devoid of any repetitions

$$TTR = \frac{V}{T_n}, \tag{7}$$

where V is the vocabulary and T_n is the number of tokens in the speech or written text

PYTHON CODE FOR TTR

This demo uses NLTK platform[**Loper:2002:NNL:1118108.1118117**]

```
import nltk
from nltk.corpus import stopwords
#get the stop words for English
stop_words = set(stopwords.words('english'))
words_bryant = nltk.Text(nltk.corpus.gutenberg.words('bryant-stories.txt'))
words_emma = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt'))
#convert to small letters
words_bryant = [word.lower() for word in words_bryant if word.isalpha()]
words_emma = [word.lower() for word in words_emma if word.isalpha()]
#remove stop words
words_bryant = [word.lower() for word in words_bryant if word not in stop_words][:15000]
words_emma = [word.lower() for word in words_emma if word not in stop_words][:15000]
TTR_bryant = len(set(words_bryant))/len(words_bryant)
TTR_emma = len(set(words_emma))/len(words_emma)
print('Number of tokens, Vocabulary, Type-token ratio (Bryant stories) = ', len(words_bryant), len(set(words_bryant)), TTR_bryant)
print('Number of tokens, Vocabulary, Type-token ratio (Jane Austen Emma) = ', len(words_emma), len(set(words_emma)), TTR_emma)
```

TTR DEMO - RESULTS

It is not reasonable to compare two unequal sized documents. A standardized TTR is used for fair comparison where the the token size is restricted to the first 15000 tokens

	Number of tokens	Vocabulary	Type-token ratio
Bryant stories	15000	2796	0.19
Jane Austen (Emma)	15000	3274	0.22

APPLICATIONS OF TTR

- Monitor the vocabulary usage
- Monitor child vocabulary development
- Estimate the vocabulary variation in the text

INVERSE DOCUMENT FREQUENCY

In order to attenuate the effect of frequently occurring terms, it is important to scale it down and at the same time it is necessary to increase the weight of terms that occur rarely.

Inverse document frequency (IDF) is defined as

$$IDF_t = \log\left(\frac{N}{D_{f_t}}\right) \tag{8}$$

where N is the total number of documents in a collection, and D_{f_t} is the count of documents containing the term t

- Rare documents gets a significantly higher value
- Commonly occurring terms are attenuated
- It is a measure of informativeness
- Reduce the tf weight of a term by a factor that grows with its collection frequency.
- If a term appears in all the documents, then IDF is zero. This implies that the term is not important

TF-IDF

Composition of TF and IDF produces a composite scaling for each term in the document

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_{t,d} \tag{9}$$

- The value is high when t occurs many times within a few documents
- The value is very low when a term appears in all documents

INVERSE DOCUMENT FREQUENCY-IDF

IDF is the inverse frequency of the word 't' appearing in the corpus. It is computed as

$$IDF \text{ of a term } t = \log_{10}\left(\frac{\text{Total number of documents in a corpus}}{\text{Count of documents with term } t}\right)$$

IDF is the measure of ***informativeness***

Example:

Consider a corpus with 100K documents. The word ***moon*** occurs in some documents (say, 100) with the following frequency:

$TF_{d_1} = \frac{20}{427}, TF_{d_2} = \frac{30}{250}, TF_{d_3} = \frac{20}{250}, TF_{d_9} = \frac{5}{125}$ and $TF_{d_{1000}} = \frac{20}{1000}$

The total number of words in the corpus $=$ 100000

$$\therefore IDF_{d_1} = \log_{10}\left(\frac{100000}{100}\right)$$

$$TF_{d_1} * IDF = 0.141$$

If the word ***Andromeda*** appears only once d_1, then $TF_{d_1} * IDF = 0.0117$. If the word ***the*** appeared in every document and 45 times in d1, then $TF * IDF = 0.210$

DOCUMENT RANKING USING TF-IDF

Using the TF-IDF, the rank order for the documents can be determined for the documents for the term *moon*.

Document Name	tf	tf-idf	Rank
d1	0.047	0.14	3
d2	0.12	0.36	1
d3	0.08	0.24	2
d9	0.04	0.12	4
d1000	0.02	0.06	5

ZIPF'S LAW

Zipf's law states that for a given some corpus, the frequency of any word is inversely proportional to its rank in the term frequency table[**Manning:1999:FSN:311445**]

$$f(r) \propto \frac{1}{r^{\alpha}} \tag{10}$$

where $\alpha \approx 1$, r is the *frequency rank* of a word and $f(r)$ is the frequency in the corpus. The most frequent word will have the value 1, the word ranked second in the frequency will have $\frac{1}{2^{\alpha}}$, the word ranked third in the frequency will have $\frac{1}{3^{\alpha}}$, etc

Distribution of terms/words

This empirical law models the frequency distribution of words in languages. This distribution is observed across several languages with a large corpus. It may not be good enough to fit the frequency linearly, but enough to approximately model word frequencies

PYTHON CODE FOR ZIPF'S EMPIRICAL LAW

This demo uses NLTK platform

```
import re
from operator import itemgetter
import nltk
import pandas as pd

frequency = {}
words_emma = nltk.Text(nltk.corpus.gutenberg.words('austen-emma.txt'))

for word in words_emma:
    count = frequency.get(word, 0)
    frequency[word] = count + 1

rank = 1
column_header = ['Rank', 'Frequency', 'Frequency*Rank']
df = pd.DataFrame(columns=column_header)

for word, freq in reversed(sorted(frequency.items(), key=itemgetter(1))):
    df.loc[word] = [rank, freq, rank * freq]
    rank = rank + 1
print(df)
```

ZIPF'S LAW - DEMO

Word	Frequency	Rank	Frequency*Rank
to	5183	3	15549
the	4844	4	19376
and	4672	5	23360
of	4279	6	25674
I	3178	7	22246
as	1387	21	29127
–	1382	22	30404
he	1365	23	31395
for	1321	24	31704
have	1301	25	32525
is	1220	26	31720
with	1187	27	32049
Mr	1153	28	32284
very	1151	29	33379
but	1148	30	34440

MANDELBROT APPROXIMATION

Mandelbrot derived a more generalized law to closely fit the frequency distribution in language by adding an offset to the rank

$$f(r) \propto \frac{1}{(r+\beta)^{\alpha}} \tag{11}$$

where $\alpha \approx 1$ and $\beta \approx 2.7$

It is still a wonder how such intricate language generation fits into a simple mathematical relationship. It seems so unreal and perhaps unreasonable ☺

HEAPS' LAW

This is used to estimate the number of unique terms M in a corpus given the total number of tokens

$$\begin{aligned} M &\propto T^b \\ &= kT^b \end{aligned} \tag{12}$$

where $30 \leq k \leq 100$ and $b \approx 0.49$

According to this empirical law, the dictionary or the vocabulary size increases linearly with the total number of tokens/words in the corpus. It emphasizes the importance of the compression of the dictionary.

Stemming and Lemmatization

good, better,best $\Rightarrow$ good
computer, computers, computers', computer's $\Rightarrow$ computer

EXERCISES

- Write a program to find out whether Mandelbrot's approximation really provides a better fit than Zipf's empirical law. Use the same corpus for Zipf and Mandelbrot approximation.
- Write a program for Heap's law and find out the prediction of vocabulary in any corpus. Also, find out whether it is closer to the actual the size of the vocabulary of the same corpus.