NATURAL LANGUAGE PROCESSING

Arpendu Ganguly

NATURAL LANGUAGE PROCESSING

- **Natural language processing (NLP) is any computation, manipulation of natural language**
- Natural Language
  - Language used for everyday communication by humans
  - Evolves with time
- **Text data is growing fast!**
  - Data continues to grow exponentially
  - Approximately 80% of all data is estimated to be unstructured, text-rich data

NATURAL LANGUAGE PROCESSING

- Parse text
- Find / Identify / Extract relevant information from text
- Classify text documents
- Search for relevant text documents
- Sentiment analysis
- Topic modeling

NPL TASKS

NLP Tasks: A Broad Spectrum

- Counting words, counting frequency of words
- Finding sentence boundaries
- Part of speech tagging
- Parsing the sentence structure
- Text Classification, Identifying semantic roles
- Identifying entities in a sentence / Named Entity Recognition
- Finding which pronoun refers to which entity / Co-reference and pronoun resolution
- much more …

NLTK: Natural Language Toolkit

- Open source library in Python
- Has support for most NLP tasks
- Also provides access to numerous text corpora

TOKENIZATION

Splitting a sentence into words / tokens

```
test1 = "Children shouldn't drink a sugary drink before bed."
```

```
test1.split(" ")
```

```
['Children', "shouldn't", 'drink', 'a', 'sugary', 'drink', 'before', 'bed.']
```

```
len(test1.split(" "))
```

```
8
```

```
nltk.word_tokenize(test1)
```

```
['Children',
 'should',
 "n't",
 'drink',
 'a',
 'sugary',
 'drink',
 'before',
 'bed',
 '.']
```

```
len(nltk.word_tokenize(test1))
```

```
10
```

COUNTING VOCABULARY OF WORDS

Count unique words

```
text="Children shouldn't drink a sugary drink before bed."
```

```
len(text)
```

```
51
```

```
nltk.word_tokenize(text)
```

```
['Children',
 'should',
 "n't",
 'drink',
 'a',
 'sugary',
 'drink',
 'before',
 'bed',
 '.']
```

```
len(nltk.word_tokenize(text))
```

```
10
```

```
len(set(nltk.word_tokenize(text)))
```

```
9
```

```
list(set(nltk.word_tokenize(text)))[:4]
```

```
['before', '.', 'sugary', 'bed']
```

FREQUENCY OF WORDS

Mapping of unique word to count

```
text1="I felt happy because I saw the others were happy and because I knew I should feel happy, but I wasn’t really happy."
```

```
dist = FreqDist(nltk.word_tokenize(text1))
dist
```

```
FreqDist({',': 1,
          '.': 1,
          'I': 5,
          'and': 1,
          'because': 2,
          'but': 1,
          'feel': 1,
          'felt': 1,
          'happy': 4,
          'knew': 1,
          'others': 1,
          'really': 1,
          'saw': 1,
          'should': 1,
          't': 1,
          'the': 1,
          'wasn': 1,
          'were': 1,
          '’': 1})
```

FREQUENCY OF WORDS

Mapping of unique word to count

```
dist.keys()
```

```
dict_keys(['I', 'felt', 'happy', 'because', 'saw', 'the', 'others', 'were', 'and', 'knew', 'should', 'feel', ',', 'but', 'was
n', '’', 't', 'really', '.'])
```

```
dist.values()
```

```
dict_values([5, 1, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
vocab1 = dist.keys()
list(vocab1)[:4]
```

```
['I', 'felt', 'happy', 'because']
```

```
dist["I"]
```

```
5
```

```
for w in vocab1:
    if len(w) >3 and dist[w]>3:
        print(w)
```

```
happy
```

```
freqwords = [w for w in vocab1 if len(w) > 3 and dist[w] > 3]
freqwords
```

```
['happy']
```

NORMALIZATION AND STEMMING

Normalization involves eliminating punctuation, converting the entire text into lowercase or uppercase and so on.

```
input1 = "List listed lists listing listings"
input1
```

```
'List listed lists listing listings'
```

```
words1=input1.lower().split(" ")
words1
```

```
['list', 'listed', 'lists', 'listing', 'listings']
```

```
porter = nltk.PorterStemmer()

#List Comprehension
[porter.stem(t) for t in words1]
```

```
['list', 'list', 'list', 'list', 'list']
```

```
[porter.stem(t) for t in input1.split(" ")]
```

```
['list', 'list', 'list', 'list', 'list']
```

STEMMING

Reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

```
text=nltk.corpus.udhr.words('English-Latin1')[7:20]
text
```

```
['recognition',
 'of',
 'the',
 'inherent',
 'dignity',
 'and',
 'of',
 'the',
 'equal',
 'and',
 'inalienable',
 'rights',
 'of']
```

```
[porter.stem(t) for t in text]
```

```
['recognit',
 'of',
 'the',
 'inher',
 'digniti',
 'and',
 'of',
 'the',
 'equal',
 'and',
 'inalien',
 'right',
 'of']
```

STEMMING AND LEMMATIZATION

Stemming and lemmatization

reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Lemmatization: Stemming, but resulting stems are all valid words.

```
WNlemma = nltk.WordNetLemmatizer()
[WNlemma.lemmatize(t) for t in text]
```

```
['Universal',
 'Declaration',
 'of',
 'Human',
 'Rights',
 'Preamble',
 'Whereas',
 'recognition',
 'of',
 'the',
 'inherent',
 'dignity',
 'and',
 'of',
 'the',
 'equal',
 'and',
 'inalienable',
 'right',
 'of']
```

PART-OF-SPEECH (POS) TAGGING

Part of speech tagging

- identification of words as nouns, verbs, adjectives, adverbs, etc
- Many more tags or word classes than just these

```
text11 = "Children shouldn't drink a sugary drink before bed."
text13 = nltk.word_tokenize(text11)
# NLTK's Tokenizer
nltk.pos_tag(text13)
```

```
[('Children', 'NNP'),
 ('should', 'MD'),
 ("n't", 'RB'),
 ('drink', 'VB'),
 ('a', 'DT'),
 ('sugary', 'JJ'),
 ('drink', 'NN'),
 ('before', 'IN'),
 ('bed', 'NN'),
 ('.', '.')]
```

```
nltk.help.upenn_tagset('MD')
```

```
MD: modal auxiliary
    can cannot could couldn't dare may might must need ought shall should
    shouldn't will would
```

TEXT FEATURE EXTRACTION

Bag of Words representation

- ***tokenizing*** *strings* and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators
- ***counting*** the occurrences of tokens in each document
- ***normalizing*** *and weighting* with diminishing importance tokens that occur in the majority of samples / documents

Sparse matrix

- sparse matrix or sparse array is a matrix in which most of the elements are zero

***n*-gram**

- A contiguous sequence of *n* items from a given sample of text or speech
- Example
  - "I am working in Accenture."
  - Unigram (1 gram): "I", "am", "working", "in", "Accenture"
  - Bigram (2 gram) : "I am", "am working", "working in", "in Accenture"

TEXT FEATURE EXTRACTION

Tf–idf transfom/term weighting

- Often words occurring frequently (e.g. “the”, “a”, “is” in English) carry very little meaningful information about the actual contents of the document.
- Weights high to terms which are rarer yet more interesting
- **tf–idf** means **term-frequency times inverse document-frequency**
- **Tf** means **term-frequency,** the number of times a term occurs in a given document
- **Inverse document-frequency**

- where is the total number of documents, and is the number of documents that contain term t.

TERM FREQUENCY (TF) MATRIX (1/2)

This is the most obvious technique to find out the relevance of a word in a document. The more frequent a word is, the more relevance the word holds in the context. Here is a frequency count of a set of words in the 5 books :

	Word Frequency								
Book Number	The	Big-Data	Analytics	Tree	newbie	book	for	Girl	honest
1	120	80	60	20	1	5	120	0	0
2	110	0	0	100	10	20	100	40	10
3	130	0	0	10	11	30	110	20	10
4	100	0	0	2	20	40	100	10	100
5	90	0	0	10	30	20	100	100	40

TERM FREQUENCY (TF) MATRIX (1/3)

One way to check Term Frequency (TF) is to just count the number of occurrence.

But it has been observed that if a word X occurs in document A 1 time and in B 10 times, its generally not true that the word X is 10 times more relevant in B than in A.

Hence it is good to apply following transformation on TF :

-

TF = 1 + log(TF)	if TF > 0
TF = 0	If TF = 0

TERM FREQUENCY (TF) MATRIX (1/3)

One way to check Term Frequency (TF) is to just count the number of occurrence.

But it has been observed that if a word X occurs in document A 1 time and in B 10 times, its generally not true that the word X is 10 times more relevant in B than in A.

Hence it is good to apply following transformation on TF :

-

TF = 1 + log(TF)	if TF > 0
TF = 0	If TF = 0

TERM FREQUENCY (TF) MATRIX -

	TF								
Book Number	The	Big-Data	Analytics	Tree	newbie	book	for	Girl	honest
1	3.1	2.9	2.8	2.3	1.0	1.7	3.1	0.0	0.0
2	3.0	0.0	0.0	3.0	2.0	2.3	3.0	2.6	2.0
3	3.1	0.0	0.0	2.0	2.0	2.5	3.0	2.3	2.0
4	3.0	0.0	0.0	1.3	2.3	2.6	3.0	2.0	3.0
5	3.0	0.0	0.0	2.0	2.5	2.3	3.0	3.0	2.6

Now to find the relevance of document in the query, you just need to sum up the values of words in the query.

- Document 1 : 1.7 + 3.1 + 2.8 + 1 = 8.6
- Document 2 :2.3 + 3.0 + 0 + 2 = 7.3
- Document 3 : 2.5 + 3.0 + 0 + 2 = 7.5
- Document 4 : 2.6 + 3.0 + 0 + 2.3 = 7.9
- Document 5 : 2.3 + 3.0 + 0 + 2.5 = 7.8

Document 1 will be more relevant to display for the query. Since, document 4 and 5 are not far away from Document 1. They might turn out to be relevant too. This is because of the stopwords which elevates all the scores with similar magnitude.

INVERSE DOCUMENT FREQUENCY MATRIX

IDF is another parameter which helps us find out the relevance of words.

It is based on the principle that less frequent words are generally more informative.

IDF = log (N/DF)

- N represents the number of documents and DF represents the number of documents in which we see the occurrence of this word.

INVERSE DOCUMENT FREQUENCY MATRIX - CALCULATIONS

IDF	The	Big-Data	Analytics	Tree	newbie	book	for	Girl	honest
N	5	5	5	5	5	5	5	5	5
DF	5	1	1	5	5	5	5	4	4
N/DF	1	5	5	1	1	1	1	1.25	1.25
Log(N/DF)	0.00	0.70	0.70	0.00	0.00	0.00	0.00	0.10	0.10

We now can clearly see that the words like "The" "for" etc. are not really relevant as they occur in almost all the document. Whereas, words like honest, Analytics Big-Data are really niche words which should be kept in the analysis.

TF – IDF MATRIX:

As we now know the relevance of words (IDF) and the occurrence of words in the documents (TF), we now can multiply the two. Then, find the subject of the document and thereafter the similarity of query with the document.

	TF-IDF								
Book Numbe	The	Big-Data	Analytics	Tree	newbie	book	for	Girl	honest
1	0.0	2.0	1.9	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.2
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.3
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.3

Relevance
1.9
0
0
0
0

Now it clearly shows that Document 1 is most relevant to query!