Natural Language Processing
Tokenization

Fellow - Jithin

Tokenization

- Given a character sequence and a defined document unit, **tokenization** is the **task of chopping** it up into **pieces**, called **tokens** , perhaps at the same time throwing away certain characters, such as punctuation.

Input: Friends, Romans, Countrymen, lend me your ears;

Output: | Friends | Romans | Countrymen | lend | me | your | ears |

- The major question of the tokenization phase is what are the correct tokens to use? In this example, it looks fairly trivial: you chop on whitespace and throw away punctuation characters. This is a starting point, but even for English there are a number of tricky cases.

Tricky Cases

Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

For *O'Neill*, which of the following is the desired tokenization?

- neill
- oneill
- o'neill
- o' neill
- o neill ?

And for *aren't*, is it:

- aren't
- arent
- are n't
- aren t ?

- (San Francisco, Los Angeles)
- Computer technology has introduced new types of character sequences
  - (jblack@mail.yahoo.com),
  - web URLs (http://stuff.big.com/new/specials.html),
  - numeric IP addresses (142.32.48.231),
  - package tracking numbers (1Z9999W99845399981).
- Hyphenation
  - Hewlett-Packard

Stemming & Lemmatization

- For grammatical reasons, documents are going to use **different forms of a word, such as organize, organizes, and organizing**. Additionally, there are **families of derivationally related words** with similar meanings, such as *democracy, democratic, and democratization*. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.
- **Stemming** usually refers to a crude heuristic process that **chops off the ends of words** in the hope of achieving this goal
- **Lemmatization** usually refers to doing things properly with the **use of a vocabulary and morphological analysis of words**, normally aiming to **remove inflectional endings** only and **to return the base** or **dictionary form of a word**, which is known as the lemma

Types of Stemming Algorithm

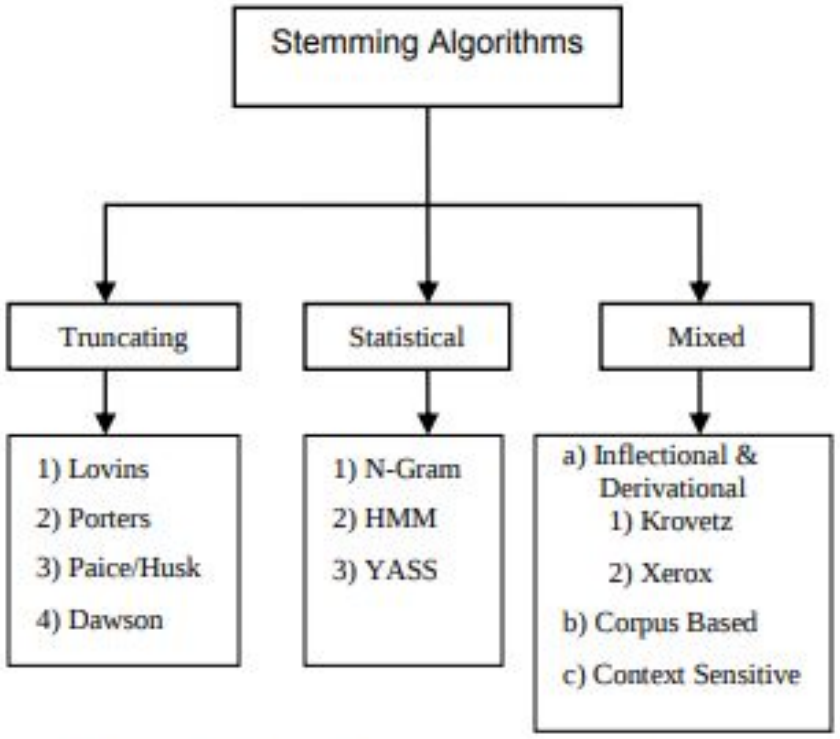

Figure 1. Types of stemming algorithms

Lovins Stemmer	
Advantages	Limitations
1) Fast – single pass algorithm. 2) Handles removal of double letters in words like 'getting' being transformed to 'get'. 3) Handles many irregular plurals like – mouse and mice etc.	1) Time consuming. 2) Not all suffixes available. 3) Not very reliable and frequently fails to form words from the stems . 4) Dependent on the technical vocabulary being used by the author.
Porters Stemmer	
Advantages	Limitations
1) Produces the best output as compared to other stemmers. 2) Less error rate. 3) Compared to Lovins it's a light stemmer. 4) The Snowball stemmer framework designed by Porter is language independent approach to stemming.	1) The stems produced are not always real words. 2) It has at least five steps and sixty rules and hence is time consuming.

Paice / Husk Stemmer	
Advantages	Limitations
1) Simple form. 2) Each iteration takes care of deletion and replacement.	1) Heavy algorithm. 2) Over stemming may occur.
Dawson Stemmer	
Advantages	Limitations
1) Covers more suffixes than Lovins 2) Fast in execution	1) Very complex 2) Lacks a standard implementation

Types of Stemming Algorithms

Table 2. Statistical Methods

N-Gram Stemmer	
Advantages	Limitations
1) Based on the concept of n-grams and string comparisons. 2) Language independent.	1) Not time efficient. 2) Requires significant amount of space for creating and indexing the n-grams. 3) Not a very practical method.
HMM Stemmer	
Advantages	Limitations
1) Based on the concept of Hidden Markov Model. 2) Unsupervised method and so is language independent.	1) A complex method for implementation. 2) Over stemming may occur in this method.
YASS Stemmer	
Advantages	Limitations
1) Based on hierarchical clustering approach and distance measures. 2) It is also a corpus based method. 3) Can be used for any language without knowing its morphology.	1) Difficult to decide a threshold for creating clusters. 2) Requires significant computing power.

Table 3. Inflectional & Derivational Methods

Krovetz Stemmer	
Advantages	Limitations
1) It is a light stemmer. 2) Can be used as a pre-stemmer for other stemmers.	1) For large documents, this stemmer is not efficient. 2) Inability to cope with words outside the lexicon. 3) Does not consistently produce a good recall and precision. 4) Lexicon to be created in advance.
Xerox Stemmer	
Advantages	Limitations
1) Works well for a large document also. 2) Removes the prefixes where ever applicable. 3) All stems are valid words.	1) Inability to cope with words outside the lexicon. 2) Not implemented successfully on language other than English. Over stemming may occur in this method. 3) Dependence on the lexicon makes it language dependent.

Natural Language Processing Use Case : NLTK & SPACY Library

Fellow - Jithin

Tokenization using NLTK

```
# Sentence Tokenizer
sent=sent_tokenize(doc)
len(sent)
```

```
52
```

```
print("Print Random sentence using Sent tokenizer \n")

for i in range(10,len(sent),3):
    print(sent[i])
```

```
Print Random sentence using Sent tokenizer

Perhaps I don’t understand things, but Austria never has wished, and does not wish, for war.
Our gracious sovereign recognizes his high vocation and will be true to it.
He will fulfill his vocation and crush the hydra of revolution, which has become more terrible than ever in the pers
on of this murderer and villain!
England with her commercial spirit will not and cannot understand the Emperor Alexander’s loftiness of soul.
What answer did Novosíltsev get?
```

```
# Word Tokenizer
words=[]
for i in range(len(sent)):
    words.append(word_tokenize(sent[i]))
```

```
print("Print Random words using word tokenizer")
for i in range(10,len(words),1897):
        print(words[i])
```

```
Print Random words using word tokenizer
['Perhaps', 'I', 'don', '’', 't', 'understand', 'things', ',', 'but', 'Austria', 'never', 'has', 'wished', ',', 'an
d', 'does', 'not', 'wish', ',', 'for', 'war', '.']
```

Tokenization using Spacy

```
spacy_words=[]
spacy_doc=nlp(doc)
for token in spacy_doc:
    spacy_words.append(token)
```

```
spacy_words[0:20]
```

```
[sh,
 to,
 be,
 believed,
 .,
 ’,
 “,
 Do,
 n’t,
```

Stemming using NLTK

```
from itertools import chain
new_words=list(chain.from_iterable(words))
```

```
stemmed_words=[]
ps=PorterStemmer()
print("Print Random Stemmed Words using word tokenizer")
for i in range(len(new_words)):
    stemmed_words.append(ps.stem(str(new_words[i])))
    print(ps.stem(str(new_words[i])))
```

```
Print Random Stemmed Words using word tokenizer
sh
to
be
believ
.
“
don
’
t
teas
!
well
```

Lemmatizing using Spacy

```
spacy_lemma_words=[]
for token in spacy_doc:
    spacy_lemma_words.append(token.lemma_)
```

```
spacy_lemma_words[0:20]
```

```
['sh',
 'to',
 'be',
 'believe',
 '.',
 ' ',
 '"',
 'do',
 'not',
 'tease',
 '!',
 'well',
 ',',
 'and',
 'what',
 'have',
 'be',
 'decide',
```

POS_tagging using NLTK

```
# len(new_words)
sent=sent_tokenize(doc)

words=[]
for i in range(len(sent)):
    words.append(word_tokenize(sent[i]))

for word in words:
    print(nltk.pos_tag(word))
```

```
[('sh', 'NN'), ('to', 'TO'), ('be', 'VB'), ('believed', 'VBN'), ('.',
[('“', 'JJ'), ('Don', 'NNP'), ('’', 'NNP'), ('t', 'JJ'), ('tease', 'NN
[('Well', 'RB'), (',', ','), ('and', 'CC'), ('what', 'WDT'), ('has', '
('about', 'IN'), ('Novosíltsev', 'NNP'), ('’', 'NNP'), ('s', 'VBD'), (
[('You', 'PRP'), ('know', 'VBP'), ('everything.', 'JJ'), ('”', 'NNP'),
D'), ('one', 'CD'), ('say', 'VB'), ('about', 'IN'), ('it', 'PRP'), ('?
('the', 'DT'), ('prince', 'NN'), ('in', 'IN'), ('a', 'DT'), ('cold', '
e', 'NN'), ('.', '.')]
[('“', 'NN'), ('What', 'WP'), ('has', 'VBZ'), ('been', 'VBN'), ('decid
[('They', 'PRP'), ('have', 'VBP'), ('decided', 'VBN'), ('that', 'IN'),
rnt', 'VBN'), ('his', 'PRP$'), ('boats', 'NNS'), (',', ','), ('and', '
at', 'IN'), ('we', 'PRP'), ('are', 'VBP'), ('ready', 'JJ'), ('to', 'TO
'JJ'), ('Prince', 'NNP'), ('Vasíli', 'NNP'), ('always', 'RB'), ('spoke
('like', 'IN'), ('an', 'DT'), ('actor', 'NN'), ('repeating', 'VBG'), (
('.', '.')]
```

POS Tagging using Spacy

```
spacy_pos=[]
spacy_doc=nlp(doc)
for token in spacy_doc:
    spacy_pos.append([token,token.pos_])
```

```
spacy_pos[0:20]
```

```
[[sh, 'PRON'],
 [to, 'PART'],
 [be, 'VERB'],
 [believed, 'VERB'],
 [., 'PUNCT'],
 [ , 'SPACE'],
 [“, 'PUNCT'],
 [Do, 'VERB'],
 [n’t, 'ADV'],
 [tease, 'VERB'],
 [!, 'PUNCT'],
 [Well, 'INTJ'],
 [,, 'PUNCT'],
 [and, 'CCONJ'],
 [what, 'NOUN'],
 [has, 'VERB'],
 [been, 'VERB'],
 [decided, 'VERB'],
 [about, 'ADP'],
 [Novosíltsev, 'PROPN']]
```

Thank you..!

Open for Questions..!