Text Tokenization

How many words in a sentence?

Based on what we learned, you should be able to implement a program that extracts (and counts) words in a sentence

Text Normalization

Every NLP task requires text normalization:

- Tokenizing (segmenting) words
- Normalizing word formats
- Segmenting sentences

Space-based tokenization

A very simple way to tokenize

- For languages that use space characters between words
  - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
- Segment off a token between instances of spaces

Unix tools for space-based tokenization

- The "tr" command
- Inspired by Ken Church's UNIX for Poets
- Given a text file, outputs the word tokens and their frequencies

Simple Tokenization in UNIX

Change all non-alpha to newlines

```
tr -sc ’A-Za-z’ ’\n’ < shakes.txt
```

```
     | sort
```

Sort in alphabetical order

```
     | uniq –c
```

Counts the occurrences of each token

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

The first step: tokenizing

```
tr -sc ’A-Za-z’ ’\n’ < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

More counting

Merging upper and lower case

```
tr ‘A-Z’ ‘a-z’ < shakes.txt | tr –sc ‘A-Za-
z’ ‘\n’ | sort | uniq –c
```

Sorting the counts

```
tr ‘A-Z’ ‘a-z’ < shakes.txt | tr –sc ‘A-Za-
z’ ‘\n’ | sort | uniq –c | sort –n –r
```

Discussion

- tr ‘A-Z’ ‘a-z’ Transforms uppercase to lowercase
- tr –sc ‘A-Za-z’ ‘\n’ Replaces non-letters with newlines
- sort Sorts alphabetically
- uniq –c Counts
- sort –n –r Sorts by count in reverse order (-n stands for “numerical sort”, -r for “reverse)

Issues in Tokenization

- Can't just blindly remove punctuation:
  - m.p.h., Ph.D., AT&T, cap’n
  - prices ($45.55)
  - dates (01/02/06)
  - URLs (http://www.stanford.edu)
  - hashtags (#nlproc)
  - email addresses (someone@cs.colorado.edu)
- Clitic: a word that doesn't stand on its own
  - "are" in we're, French "je" in j'ai, "le" in l'honneur
- When should multiword expressions (MWE) be words?
  - New York, rock ’n’ roll

Sentence splitting

Sentence splitting

- Our first task is often to split a long document into sentences
- We will see that NL parsing is able to do it by analyzing how a sentence is composed
- For now, we will learn how to spilt a document by using very simple heuristics
- Also, we will first learn to do it “by hand” before using available solutions

First attempt

```python
import re

f=open("document.txt")
s=f.read()
sentences=re.split(r"[.!?]\s+",s)

for sentence in sentences:
    print(sentence)
```

Discussion

- Split any time there is a `[.?!]` followed by spaces
- Possible variations: also consider `[:;]`

Problems

Splitting

```
Mr. Smith is buying flowers, plants, etc. before
going to work
```

gives

```
Mr

Smith is buying flowers, plants, etc

before going to work
```

Some simple heuristics

- There abbreviations that never go at the end of a sentence (we call them prefix), for example “Mr.” “Dr.” “Prof.” “Eng.” “Mrs.”
- Some other abbreviations also never go at the end of a sentence: “e.g.”, “i.e.”
    - We can transform them by removing “.”
- Some abbreviations may go at the end of a sentence: “etc.”
    - We can assume that there is a new sentence only if they are followed by a capital letter

Implementation

```python
import re

prefix="mr|prof|dr|mrs|eng|i.e|e.g"
s="Mr. Smith is buying things e.g., flowers, plants, before going to work with Prof. John"

s2=re.sub("("+prefix+r")\.",r"\1",s,flags=re.IGNORECASE)
sentences=re.split(r"[.!?]\s+",s2)

print(sentences)
```

```
['Mr Smith is buying things e.g, flowers, plants, before going to work with Prof John']
```

Still a problem…

```python
import re

prefix="mr|prof|dr|mrs|eng|i.e|e.g"
s="Mr. Smith is buying things e.g., flowers, plants, etc. before going to work with Prof. John"

s2=re.sub("("+prefix+r")\.",r"\1",s,flags=re.IGNORECASE)
sentences=re.split(r"[.!?]\s+",s2)

print(sentences)
```

```
['Mr Smith is buying things e.g, flowers, plants, etc', 'before going to work with Prof John']
```

Can’t we simply add “etc.” to the list?

Discussion

- If we add etc to our list, then it would no longer split sentences when etc. appears at the end of a sentence
- Solution: split if etc. is followed by:

    1. spaces
    2. possibly, special characters: `[]{}();,;:-`
    3. possibly, more spaces
    4. an upper case letter

Implementation

```
import re

prefix="mr|prof|dr|mrs|eng|i.e|e.g|etc"
abbreviation="etc"

s="Mr. Smith is buying things e.g., flowers, plants, etc. before going to work with
Prof. John"

s2=re.sub(r"\b"+r"("+abbreviation+r")\.(\s+[\[\],(){};:-]*\s*[A-Z])",r"\1 .\2",s)
s3=re.sub("("+prefix+r")\.",r"\1",s2,flags=re.IGNORECASE)

sentences=re.split(r"[.!?]\s+",s3)

print(sentences)
```

Out-of-box implementation in NLTK

```
import nltk

s="Mr. Smith is buying flowers, plants, etc. before going to work"
sentences=nltk.sent_tokenize(s)
```

Still not very good as it gives:

```
Mr. Smith is buying flowers, plants, etc.

before going to work
```

Custom abbreviations

```
from nltk.tokenize.punkt import PunktSentenceTokenizer, PunktParameters

punkt_param = PunktParameters()
abbreviation = ['mr','u.s.a', 'fig','etc','i.e','e.g']
punkt_param.abbrev_types = set(abbreviation)
tokenizer = PunktSentenceTokenizer(punkt_param)

s="Mr. Smith is buying flowers, plants, etc. before going to work"
sentences=tokenizer.tokenize(s)


for sentence in sentences:
    print(sentence)
```

```
Mr. Smith is buying flowers, plants, etc. before going to work
```

Still may not work...

```
from nltk.tokenize.punkt import PunktSentenceTokenizer, PunktParameters

punkt_param = PunktParameters()
abbreviation = ['mr','u.s.a', 'fig','etc','i.e','e.g']
punkt_param.abbrev_types = set(abbreviation)
tokenizer = PunktSentenceTokenizer(punkt_param)

s="Mr. Smith is buying flowers, plants, etc. Today, he's not feeling well"
sentences=tokenizer.tokenize(s)


for sentence in sentences:
    print(sentence)
```

```
Mr. Smith is buying flowers, plants, etc. Today, he's not feeling well
```

The sentence should be split, but it is not!

Solution

Combine the solution before with the heuristic we have introduced to handle cases like “etc.”

Implementation

```python
from nltk.tokenize.punkt import PunktSentenceTokenizer, PunktParameters
import re

punkt_param = PunktParameters()
abbreviation = ['mr','u.s.a', 'fig','etc','i.e','e.g']
postfix_abbr="etc"
punkt_param.abbrev_types = set(abbreviation)
tokenizer = PunktSentenceTokenizer(punkt_param)

s="Mr. Smith is buying flowers, plants etc. today, he's not feeling well"

s2=re.sub(r"\b"+r"("+postfix_abbr+r")\.(\s+[\[\],(){};:-]*\s*[A-Z])",r"\1 .\2",s)
sentences=tokenizer.tokenize(s2)

print(sentences)
```

Tokenization

Tokenization

- Once we have identified simple sentences, it’s time to split them into constituent words

Simple example

- Just use space as separators

```
import re

s="I'm going to school, but not today"

words=re.split(r"\s+",s)
print(words)
```

```
["I'm", 'going', 'to', 'school,', 'but', 'not', 'today']
```

Dropping special characters before...

- Note that some characters should be dropped only if followed by spaces

```
import re

s="I'm going to school at 10:00, but not today (and I know it)"

#remove parentheses
s2=re.sub(r"[\[\](){}]","",s)

#remove punctuation but only if followed by spaces
#e.g. it avoid to remove them in numbers like 10,000
#or times e.g. 12:45:30
s3=re.sub(r"[;:]\s"," ",s2)

words=re.split(r"\s+",s3)
print(words)
```

Simple tokenization in nltk

```python
from nltk.tokenize import word_tokenize
text="I'm going to school at 10:00, but not today (and I know it)"
print(word_tokenize(text))
```

```
['Hello', 'everybody', '!', 'This', 'is', 'the', 'first',
'tokenization', 'example', '.']
```

Discussion

- Space-based tokenization
- Special characters/punctuation not removed (should be removed after)

Pretrained tokenizers

- Other than tokenizers from libraries like NLTK or those based on regexp, we could leverage some advanced machine-learning tokenizers
- Those will have to be used when we will use advanced deep learning models (e.g., text transformers)
- These models have been “pretrained” on existing, large datasets, so we don’t need to train them again

Pretrained tokenizers: example

First of all, install:

1) Tensorflow (we will see later what it is)

```
pip3 install tensorflow
```

2) the Transformers library (from Hugginface):

```
pip3 install transformers
```

3) SentencePiece

```
pip3 install sentencepiece
```

Bert tokenizer uncased

First, I import the BertTokenizer from the Huggingface transformer library

```
from transformers import BertTokenizer
```

Then I try to import the "uncased" Bert tokenizer

```
tokenizer = BertTokenizer.from_pretrained(“bert-base-uncased”)
```

Let's now try it…

```
sentence="This is my new SE book"
tokenizer.tokenize(sentence)
```

```
['this', 'is', 'my', 'new', 'se', ‘book']
```

ok this works fine... we can notice that since this is an uncased tokenizer, the sentence is uncased first

Another example...

```
sentence="I just bought a new GPU"
tokenizer.tokenize(sentence)

['i', 'just', 'bought', 'a', 'new', 'gp', '##u']
```

GPU is split into 2 words as the acronym is unknown by the trained tokenizer

```
sentence="My unknown acronym is BRTX"
tokenizer.tokenize(sentence)

['my', 'unknown', 'acronym', 'is', 'br', '##t', '##x']
```

As BRTX is knot a known acronym, the tokenizer split it into a known acronym (br) + other letters

Using a “cased” tokenizer

```
tokenizer = tokenizer = BertTokenizer.from_pretrained(“bert-base-cased")
tokenizer.tokenize(sentence)
['My', 'unknown', 'acronym', 'is', 'BR', '##T', ‘##X']

You see the case is now preserved
```

A different tokenizer

```
from transformers import XLNetTokenizer
tokenizer = XLNetTokenizer.from_pretrained(“xlnet-base-cased")
tokenizer.tokenize(sentence)
['_My', '_unknown', '_acronym', '_is', '_', 'BR', ‘TX']
```

you see how the tokenization is chenged because the new tokenizer knows the words BR an TX

Also, this tokenizer does not ignore spaces, and keep them as part of the text (represented as “_”)

Word Normalization

Word Normalization

Putting words/tokens in a standard format

- U.S.A. or USA
- uhhuh or uh-huh
- Fed or fed
- am, is, be, are

Case folding

Applications like IR: reduce all letters to lower case

- Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***

For natural language parsing sentiment analysis, machine translation, Information extraction

- Case is helpful (***US*** versus ***us*** is important)

Simple approach

- Remove special character within words ("-,.") but not within numbers
- Lowercase everything is not contained in a special list

Implementation

```python
import re
words=['U.S.A',"10.4","this","Is","it","e.g","t-shirt"]

acronyms=['USA']

prunedSpecial=[re.sub(r"[,.-]","",word) \
                   if re.search(r"[a-z]",word,re.IGNORECASE) \
                   else word for word in words]

lc=[word.lower() if  word not in acronyms else word for word in prunedSpecial]
print(lc)
```

Stop word removal

Stop word lists

- For example: https://99webtools.com/blog/list-of-english-stop-words/

Example

```
a, able, about, across, after, all, almost, also, am, among, an, and, any, are, as, at,
be, because, been, but, by, can, cannot, could, dear, did, do, does, either, else, ever,
every, for, from, get, got, had, has, have, he, her, hers, him, his, how, however, i, if,
in, into, is, it, its, just, least, let, like, likely, may, me, might, most, must, my,
neither, no, nor, not, of, off, often, on, only, or, other, our, own, rather, said, say,
says, she, should, since, so, some, than, that, the, their, them, then, there, these,
they, this, tis, to, too, twas, us, wants, was, we, were, what, when, where, which,
while, who, whom, why, will, with, would, yet, you, your, ain't, aren't, can't, could've,
couldn't, didn't, doesn't, don't, hasn't, he'd, he'll, he's, how'd, how'll, how's, i'd,
i'll, i'm, i've, isn't, it's, might've, mightn't, must've, mustn't, shan't, she'd,
she'll, she's, should've, shouldn't, that'll, that's, there's, they'd, they'll, they're,
they've, wasn't, we'd, we'll, we're, weren't, what'd, what's, when'd, when'll, when's,
where'd, where'll, where's, who'd, who'll, who's, why'd, why'll, why's, won't, would've,
wouldn't, you'd, you'll, you're, you've
```

NLTK facilities

```
import re
from nltk.corpus import stopwords

sw=stopwords.words('english')
acronyms=['USA']


words=['U.S.A',"10.4","this","Is","it","e.g","t-shirt"]
prunedSpecial=[re.sub(r"[,.-]","",word) \
                   if re.search(r"[a-z]",word,re.IGNORECASE) \
                   else word for word in words]

lc=[word.lower() if word not in acronyms else word for word in
prunedSpecial]

removedSw=[word for word in lc if  word not in sw]

print(sw)
print(removedSw)
```

Stemming and Lemmatization

- Goal: bring words to a common root
- Stemming: based on the application of rules
- Lemmatization: based on a morphological database

Stemming

Reason:

- Different word forms may bear similar meaning (e.g. search, searching): create a "standard" representation for them

Stemming:

- Removing some endings of word

$$\left.\begin{array}{l}\text{computer}\\ \text{compute}\\ \text{computes}\\ \text{computing}\\ \text{computed}\\ \text{computation}\end{array}\right\}\quad \textbf{comput}$$

Stemming

Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note
.

Porter Stemmer

(Porter, M.F., 1980, An algorithm for suffix stripping, Program, **14**(3) :130-137)

- Procedure for removing known affixes in English without using a dictionary.
- Can produce unusual stems that are not English words:
  - "computer", "computational", "computation" all reduced to same token "comput"
- May conflate (reduce to the same token) words that are actually distinct.
- Does not recognize all morphological derivations.

How it works

- A consonant in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant.

- A consonant will be denoted by c, a vowel by v
- A list ccc... of length greater than 0 will be denoted by C
- A list vvv... of length greater than 0 will be denoted by V

- Any word, or part of a word, therefore has one of the four forms:
  - CVCV ... C
  - CVCV ...V
  - VCVC ... C
  - VCVC ...V

- These may all be represented by the single form [C]VCVC ... [V]
  - square brackets denote arbitrary presence of their contents.

How it works (cont.)

- Using (VC){m} to denote VC repeated m times, this may again be written as [C](VC){m}[V]

- m: measure of word or word part
  - m=0 covers the null word: TR, EE, TREE, Y, BY.
  - m=1 TROUBLE, OATS, TREES, IVY.
  - m=2 TROUBLES, PRIVATE, OATEN, ORRERY.

- Rules for removing a suffix:
- (condition) S1 → S2
- if a word ends with S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2.
  - The condition is usually given in terms of m

Porter algorithm

- Step 1: plurals and past participles
  - SSES → SS   caresses → caress
  - (*v*) ING →   motoring → motor
- Step 2: adj→n, n→v, n→adj, …
  - (m>0) OUSNESS → OUS   callousness → callous
  - (m>0) ATIONAL → ATE   relational → relate
- Step 3:
  - (m>0) ICATE → IC   triplicate → triplic
- Step 4:
  - (m>1) AL →   revival → reviv
  - (m>1) ANCE →   allowance → allow
- Step 5:
  - (m>1) E →   probate → probat
  - (m > 1 and *d and *L) → single letter   controll → control

Other stemmers

- Lovins stemmer
  - Lovins J.B. (1968) - Development of a stemming algorithm. Mechanical Translation and Computational Linguistics, 11: 22-31.
- Snowball stemmer
  - https://snowballstem.org
  - Refines Porter by adding further rules
  - Available for many languages including Italian

NLTK Implementations: Porter

```
from nltk.stem.porter import *

stemmer=PorterStemmer()

words=['John','goes','to','school','with','his','friends']

stemmed=[stemmer.stem(word) for word in words]

print(stemmed)
```

```
['john', 'goe', 'to', 'school', 'with', 'hi', 'friend']
```

With stop word removal

```python
from nltk.stem.porter import *
from nltk.corpus import stopwords

sw=stopwords.words('english')
stemmer=PorterStemmer()

words=['John','goes','to','school','with','his','friends']

stems=[stemmer.stem(word) for word in words if word not in sw]

print(stems)
```

```
['john', 'goe', 'school', 'friend']
```

Snowball stemming

```
from nltk.stem import snowball
from nltk.corpus import stopwords

sw=stopwords.words('english')
stemmer=snowball.SnowballStemmer(language="english")

words=['John','goes','to','school','with','his','friends']

stems=[stemmer.stem(word) for word in words if word not in sw]

print(stems)
```

Some differences

- `porter.stem('fairly')` → returns fairli
- `snowball.stem('fairly')` → returns fair

- `porter.stem('generically')` → returns gener
- `porter.stem('generous')` → returns gener
- `snowball.stem('generically')` → returns generical
- `snowball.stem('generous')` → returns generous

Hints

- Snowball tends to produce more “meaningful” words instead of meaningless stems
- Also, it limits over-aggressive stemming bringing back to the same radix completely different words
- Therefore, it may be preferred to Porter

Lemmatization

Represent all words as their lemma, their shared root

= dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*

- *He is reading detective stories*

→ *He be read detective story*

Lemmatization is done by Morphological Parsing

Morphemes:

- The small meaningful units that make up words
- Stems: The core meaning-bearing units
- Affixes: Parts that adhere to stems, often with grammatical functions

Morphological Parsers:

- Parse *cats* into two morphemes *cat* and *s*

NLTK Implementation

```python
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()


words=['John','and','Mike','are','going','to','school','with','their','friends']

lemma=[lemmatizer.lemmatize(word) for word in words]

print(lemma)
```

```
['John', 'and', 'Mike', 'are', 'going', 'to', 'school', 'with', 'their', 'friend']
```

Discussion

- The output seems even worse than the stemmer
- Why?
- To work properly, the lemmatizer needs to know the Part-of-Speech (POS) associated to a word
    - verb, noun, adjective, adverb, …

Example

```
lemmatizer.lemmatize('going',pos='v') # returns ‘go'

lemmatizer.lemmatize('went',pos='v') # returns 'go'

lemmatizer.lemmatize('computing',pos='n') # returns 'computing'

lemmatizer.lemmatize('computing',pos='v') # returns 'compute'
```

POS Tagging

We use NLTK for POS Tagging

```
from nltk import pos_tag
from nltk import word_tokenize
text = "I went to school and found the desks broken."
tokenized_text = word_tokenize(text)
tags = tokens_tag = pos_tag(tokenized_text)
tags
```

```
[('I', 'PRP'),
 ('went', 'VBD'),
 ('to', 'TO'),
 ('school', 'NN'),
 ('and', 'CC'),
 ('found', 'VBD'),
 ('the', 'DT'),
 ('desks', 'NNS'),
 ('broken', 'VBN'),
 ('.', '.')]
```

Parts of speech (Treebank set)

Abbrev	Type
CC	coordinating conjunction
DT	determiner
JJ	adjective
NN	noun (singular)
PRP	personal pronoun
NNS	noun plural
VB	verb
VBG	verb (gerund)
TO	infinite marker
VBN	verb past participle

Complete list:
https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

However…

- The lemmatizer accepts letters
    - v: verb, j: adjective, n: noun, r: adverb
    - The rest can be treated as a noun

Conversion function

```python
from nltk.corpus import wordnet
def tagConversion(tag):
    match(tag[0]):
        case 'J':
            return wordnet.ADJ
        case 'V':
            return wordnet.VERB
        case 'N':
            return wordnet.NOUN
        case 'R':
            return wordnet.ADV
        case _:
            return wordnet.NOUN
```

Now let’s do the lemmatization...

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized=[lemmatizer.lemmatize(x[0],tagConversion(x[1])) for x in tags]
lemmatized
```

```
['I', 'go', 'to', 'school', 'and', 'find', 'the', 'desk', 'break', '.']
```