Natural Language Processing

Tutorial 2: Text Normalization

Dr. Sun Aixin

Question 1

- Consider the following word segmentation algorithm in the lecture notes:

- Given a lexicon of Chinese, and a string
  - Start a pointer at the beginning of the string
  - Find the longest word in dictionary that matches the string starting at pointer
  - Move the pointer over the word in string
  - Goto2

- Following the algorithm, you perhaps end up with failing to segment a string, if you cannot find a matching.

Question 1 (cont)

- Example
  - String to segment: thetablesdownthere
  - lexicon: the table down there bled own.
- Discuss how to fix the above problem.

Answer Q1

- Start a pointer at the beginning of the string
- Find the longest word in dictionary that matches the string starting at pointer
  - If matched, move the pointer over the word in string
  - If no word is matched, skip to the next letter
- Goto2

  - String to segment: thetablesdownthere
  - lexicon: the table down there bled own.

Answer 1 (a bit more processing)

- Start a pointer at the beginning of the string
- Find the longest word in dictionary that matches the string starting at pointer
    - If matched,
        - Run morphological analysis from the beginning of the word
        - Move the pointer over the morphologically matched part of the string
    - If no word is matched, skip to the next letter
- Goto2

    - String to segment: thetablesdownthere
    - lexicon: the table down there bled own.

Question 2

- Try the tokenization demo on
  - http://text-processing.com/demo/tokenize/
  - (you may use other tokenizer APIs).
- Discuss your findings based on the output of different tokenizers.

- https://textanalysisonline.com/

Question 3

- Try the stemmer demo on
  - http://text-processing.com/demo/stem/
  - (or you may use other stemmer APIs).
- Discuss your findings based on the output of the stemmers.

- https://textanalysisonline.com/

Question 4

- Write a program to do the following tasks:
  - **Download** the Web page of a given link and **extract** the text content of the page
  - **Split** the text into sentences and **count** sentences
  - **Split** the text into tokens and **count** token types
  - **Find** lemmas (or stems) of the tokens and **count** lemma types
  - Do **stemming** on the tokens and **count** unique stemmed tokens

Answer 4

- Example URL:
  - https://en.wikipedia.org/wiki/Natural_language_processing
- urllib
  - https://docs.python.org/3/library/urllib.html
- NLTK
  - https://www.nltk.org/
- Beautiful Soup
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/#porting-code-to-bs4

Alternative libraries:
StanfordNLP
OpenNLP
spaCy
AllenNLP

Sample code

```
import urllib.request
import nltk
from bs4 import BeautifulSoup

with urllib.request.urlopen ('https://en.wikipedia.org/wiki/Natural_language_processing') as response:
    html=response.read()

text = BeautifulSoup(html, "lxml").get_text()
sentences = nltk.tokenize.sent_tokenize(text)
print ('Number of sentences: '+ str(len(sentences)))

tokens= nltk.tokenize.word_tokenize(text)

print ('Number of tokens: '+ str(len(tokens)))

token_types = list(set(tokens))

print ('Number of token types: '+ str(len(token_types)))

wnl=nltk.stem.WordNetLemmatizer()

stemmer = nltk.stem.porter.PorterStemmer()
lemma_types= set()
stemmed_types= set()

for token_type in token_types:
    lemma_types.add(wnl.lemmatize(token_type))
    stemmed_types.add(stemmer.stem(token_type))

print ('Number of lemma types: '+ str(len(lemma_types)))
print ('Number of stemmed types: '+ str(len(stemmed_types)))
```

Download the Web page of a given link and **extract** the text content of the page

Split the text into sentences and **count** sentences

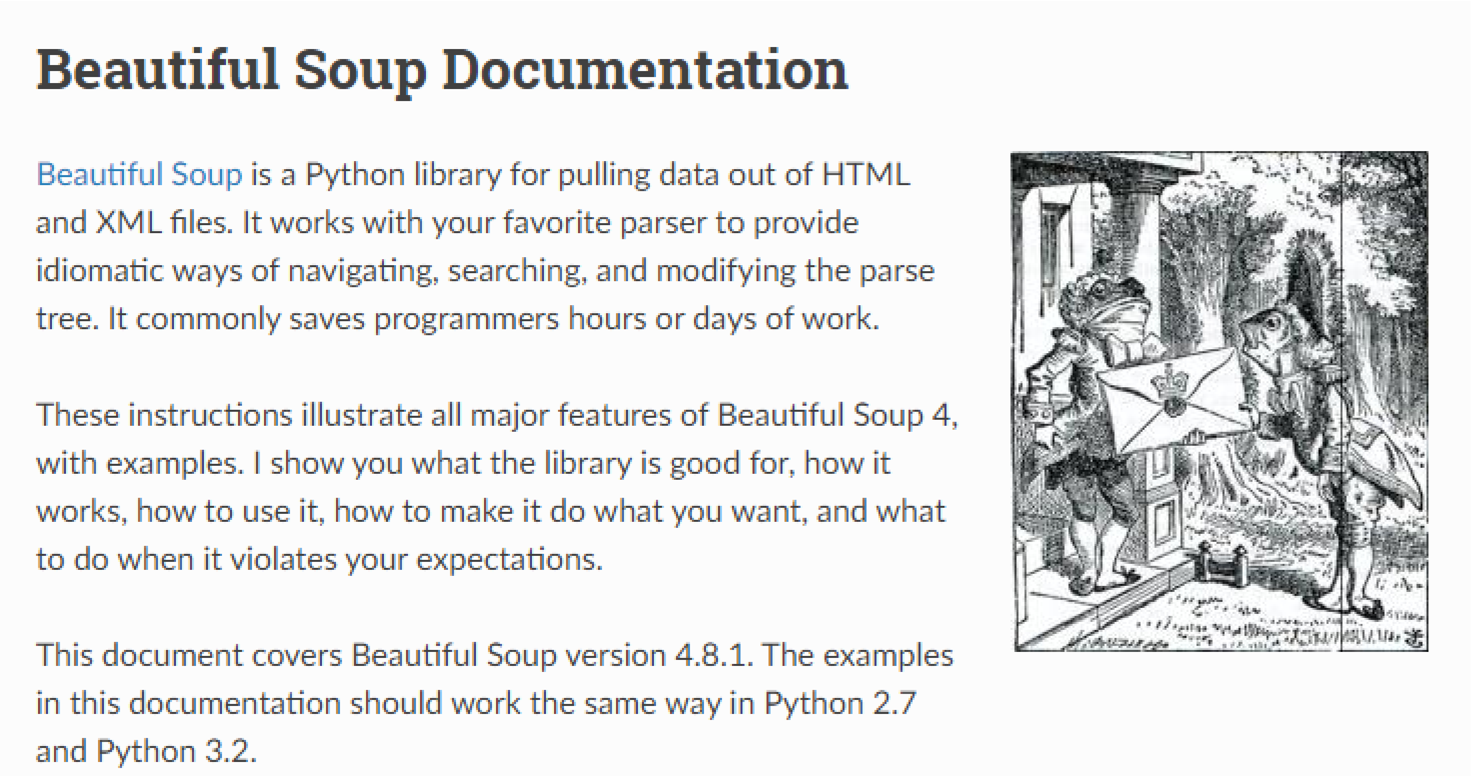
Split the text into tokens and **count** token types

Find lemmas (or stems) of the tokens and **count** lemma types

Do **stemming** on the tokens and **count** unique stemmed tokens

Sample code

- text = BeautifulSoup(html, "lxml").get_text()



Beautiful Soup Documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

These instructions illustrate all major features of Beautiful Soup 4, with examples. I show you what the library is good for, how it works, how to use it, how to make it do what you want, and what to do when it violates your expectations.

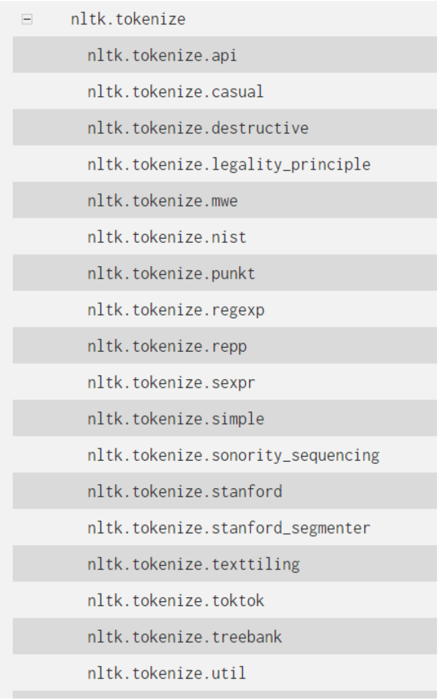
This document covers Beautiful Soup version 4.8.1. The examples in this documentation should work the same way in Python 2.7 and Python 3.2.

https://beautiful-soup-4.readthedocs.io/en/latest/

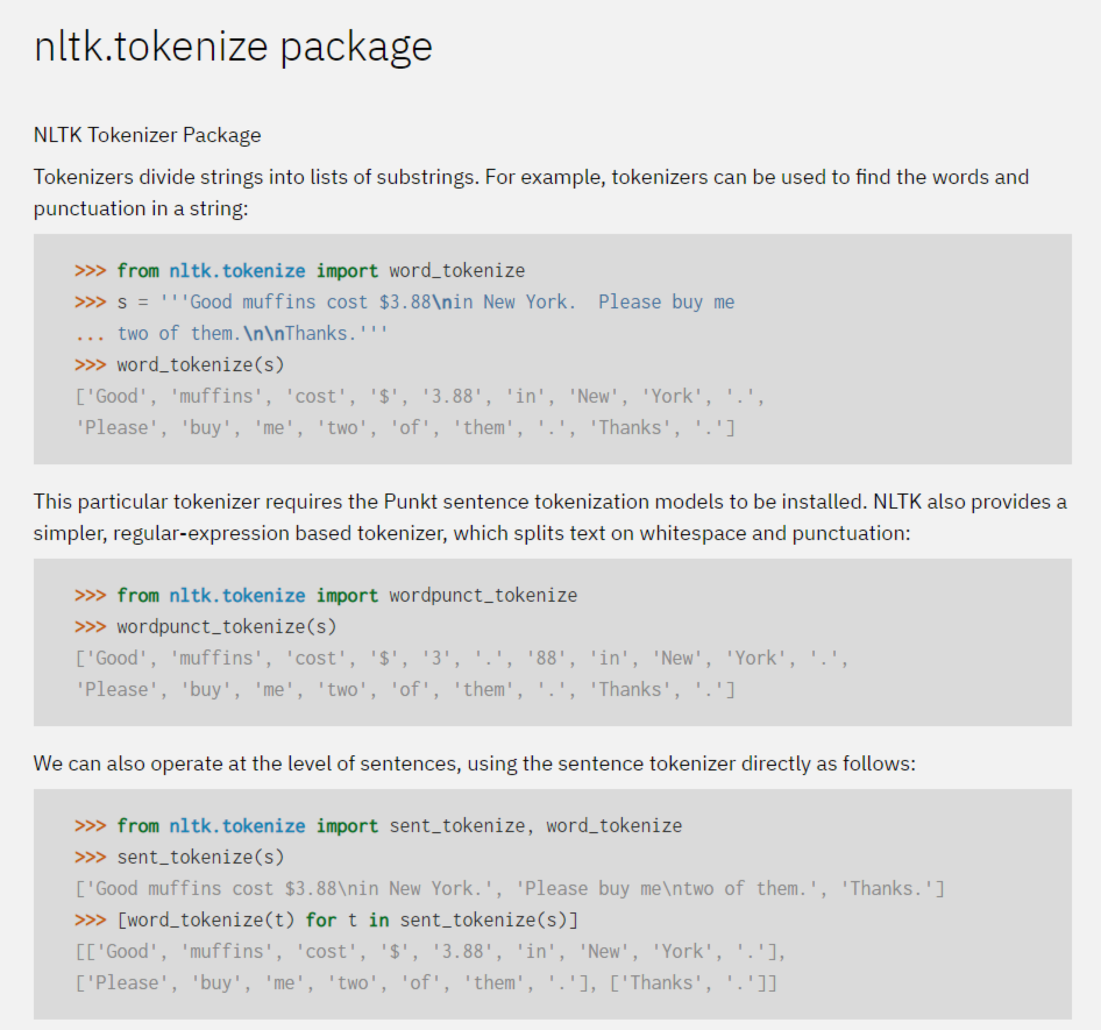
Sample code

https://www.nltk.org/api/nltk.tokenize.html

- sentences = nltk.tokenize.sent_tokenize(text)
- tokens= nltk.tokenize.word_tokenize(text)



```
⊟ nltk.tokenize
    nltk.tokenize.api
    nltk.tokenize.casual
    nltk.tokenize.destructive
    nltk.tokenize.legality_principle
    nltk.tokenize.mwe
    nltk.tokenize.nist
    nltk.tokenize.punkt
    nltk.tokenize.regexp
    nltk.tokenize.repp
    nltk.tokenize.sexpr
    nltk.tokenize.simple
    nltk.tokenize.sonority_sequencing
    nltk.tokenize.stanford
    nltk.tokenize.stanford_segmenter
    nltk.tokenize.texttiling
    nltk.tokenize.toktok
    nltk.tokenize.treebank
    nltk.tokenize.util
```



nltk.tokenize package

NLTK Tokenizer Package

Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string:

```
>>> from nltk.tokenize import word_tokenize
>>> s = '''Good muffins cost $3.88\nin New York.  Please buy me
... two of them.\n\nThanks.'''
>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.',
'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

This particular tokenizer requires the Punkt sentence tokenization models to be installed. NLTK also provides a simpler, regular-expression based tokenizer, which splits text on whitespace and punctuation:

```
>>> from nltk.tokenize import wordpunct_tokenize
>>> wordpunct_tokenize(s)
['Good', 'muffins', 'cost', '$', '3', '.', '88', 'in', 'New', 'York', '.',
'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

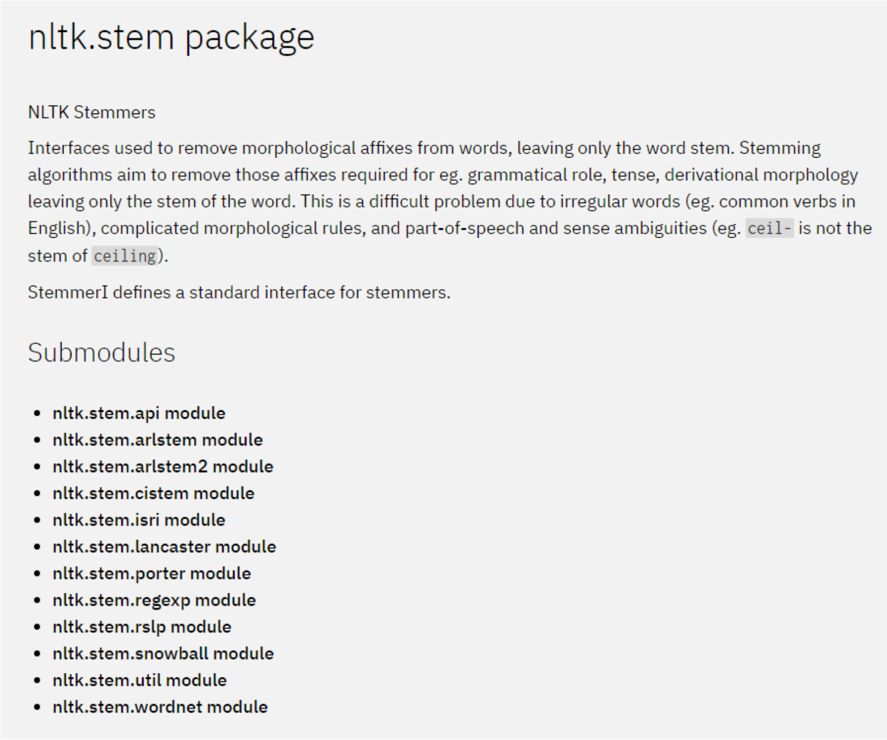
We can also operate at the level of sentences, using the sentence tokenizer directly as follows:

```
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> sent_tokenize(s)
['Good muffins cost $3.88\nin New York.', 'Please buy me\ntwo of them.', 'Thanks.']
>>> [word_tokenize(t) for t in sent_tokenize(s)]
[['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.'],
['Please', 'buy', 'me', 'two', 'of', 'them', '.'], ['Thanks', '.']]
```

Sample code

https://www.nltk.org/api/nltk.stem.html
https://www.nltk.org/api/nltk.stem.wordnet.html
https://web.stanford.edu/~jurafsky/slp3/

- wnl=nltk.stem.WordNetLemmatizer()
- stemmer = nltk.stem.porter.PorterStemmer()



nltk.stem package

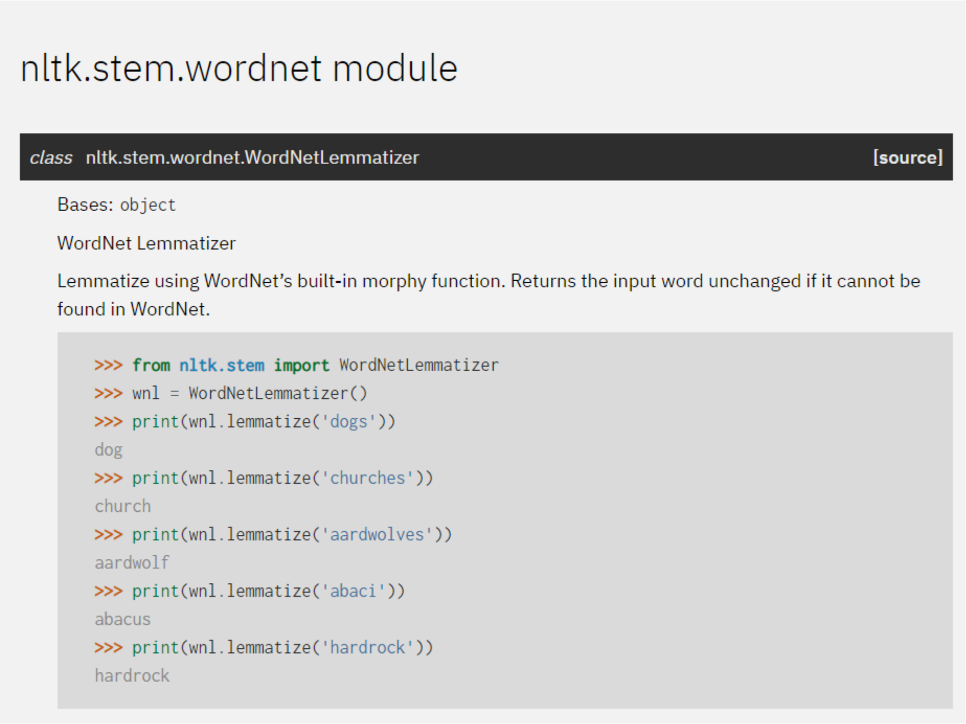
NLTK Stemmers

Interfaces used to remove morphological affixes from words, leaving only the word stem. Stemming algorithms aim to remove those affixes required for eg. grammatical role, tense, derivational morphology leaving only the stem of the word. This is a difficult problem due to irregular words (eg. common verbs in English), complicated morphological rules, and part-of-speech and sense ambiguities (eg. `ceil-` is not the stem of `ceiling`).

StemmerI defines a standard interface for stemmers.

Submodules

- nltk.stem.api module
- nltk.stem.arlstem module
- nltk.stem.arlstem2 module
- nltk.stem.cistem module
- nltk.stem.isri module
- nltk.stem.lancaster module
- nltk.stem.porter module
- nltk.stem.regexp module
- nltk.stem.rslp module
- nltk.stem.snowball module
- nltk.stem.util module
- nltk.stem.wordnet module



nltk.stem.wordnet module

class nltk.stem.wordnet.WordNetLemmatizer [source]

Bases: `object`

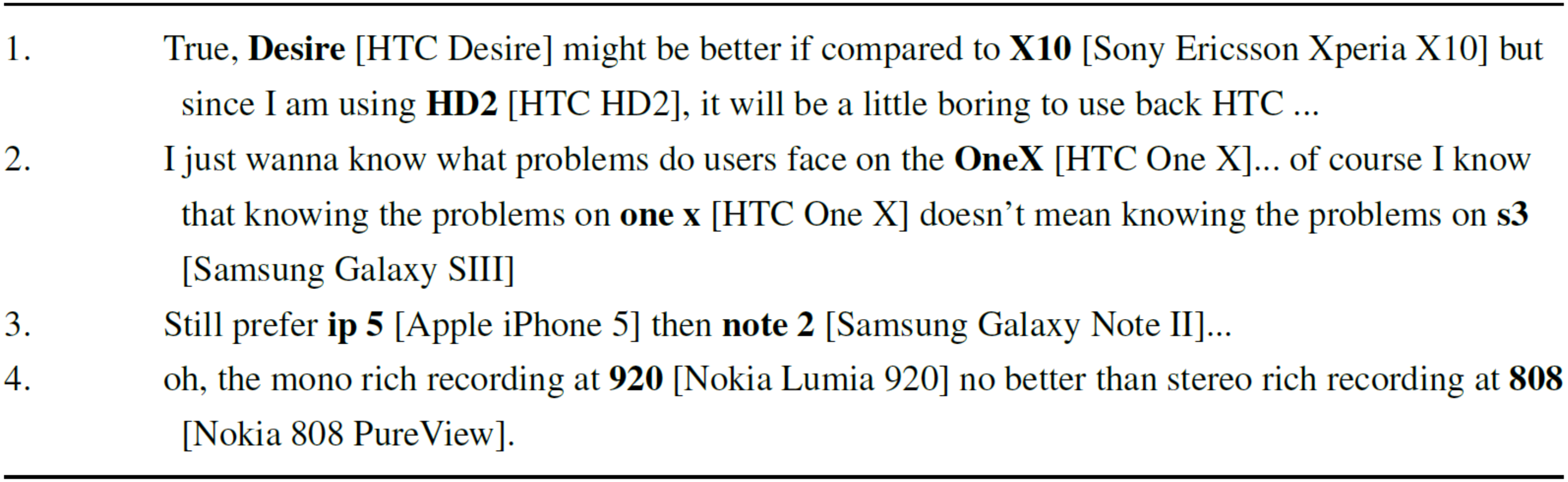
WordNet Lemmatizer

Lemmatize using WordNet's built-in morphy function. Returns the input word unchanged if it cannot be found in WordNet.

```
>>> from nltk.stem import WordNetLemmatizer
>>> wnl = WordNetLemmatizer()
>>> print(wnl.lemmatize('dogs'))
dog
>>> print(wnl.lemmatize('churches'))
church
>>> print(wnl.lemmatize('aardwolves'))
aardwolf
>>> print(wnl.lemmatize('abaci'))
abacus
>>> print(wnl.lemmatize('hardrock'))
hardrock
```

Q5: Open-ended Question, for discussion only.

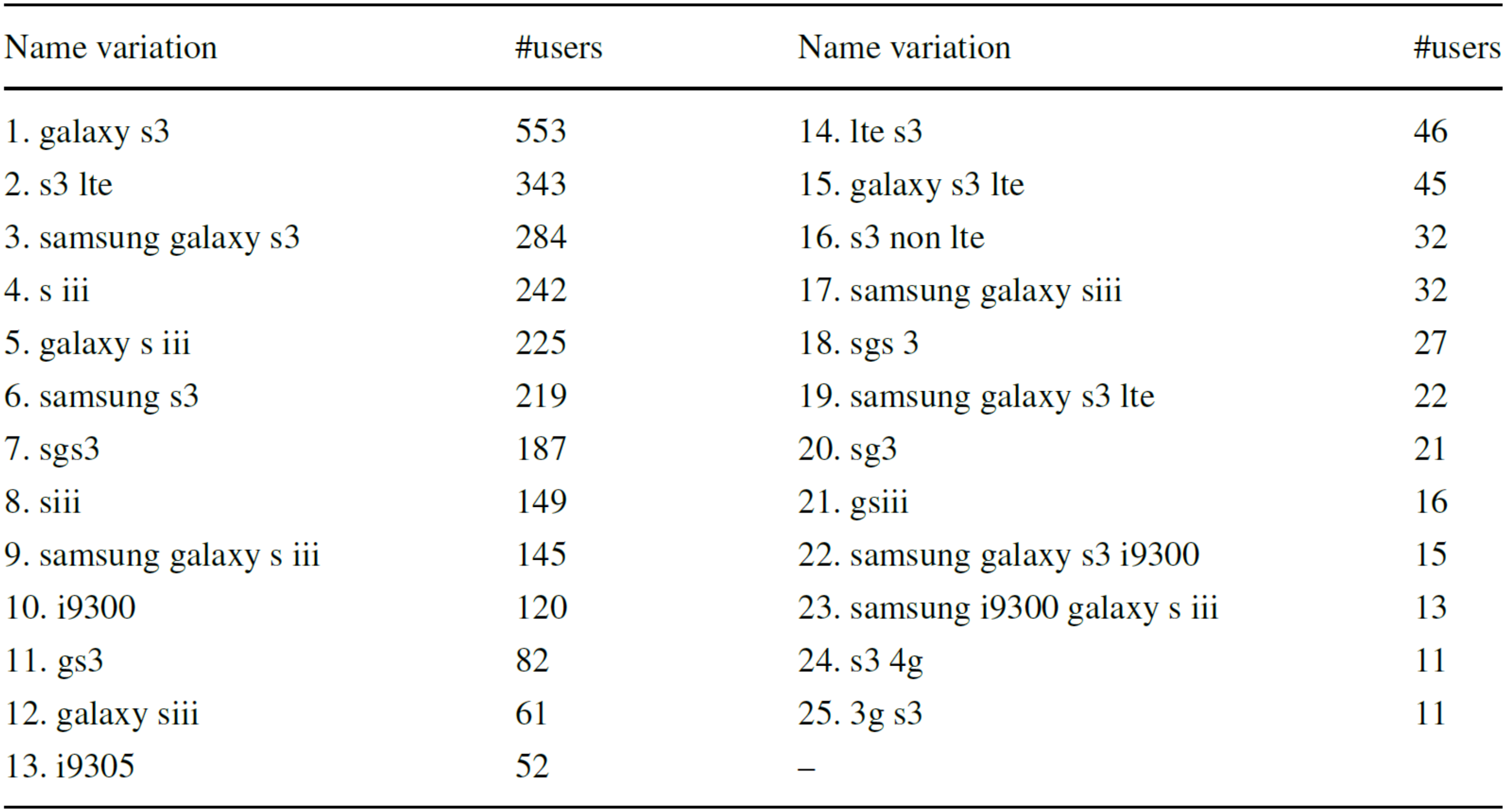
- In social media (e.g., forums), online users often use informal names or references when mentioning products. Below are example sentences discussing mobile phones, where the words highlighted in bold are the phones being referred to, and the [bracketed text] indicates their official product names.



1. True, **Desire** [HTC Desire] might be better if compared to **X10** [Sony Ericsson Xperia X10] but since I am using **HD2** [HTC HD2], it will be a little boring to use back HTC ...
2. I just wanna know what problems do users face on the **OneX** [HTC One X]... of course I know that knowing the problems on **one x** [HTC One X] doesn't mean knowing the problems on **s3** [Samsung Galaxy SIII]
3. Still prefer **ip 5** [Apple iPhone 5] then **note 2** [Samsung Galaxy Note II]...
4. oh, the mono rich recording at **920** [Nokia Lumia 920] no better than stereo rich recording at **808** [Nokia 808 PureView].

Q5: Open-ended Question, for discussion only.

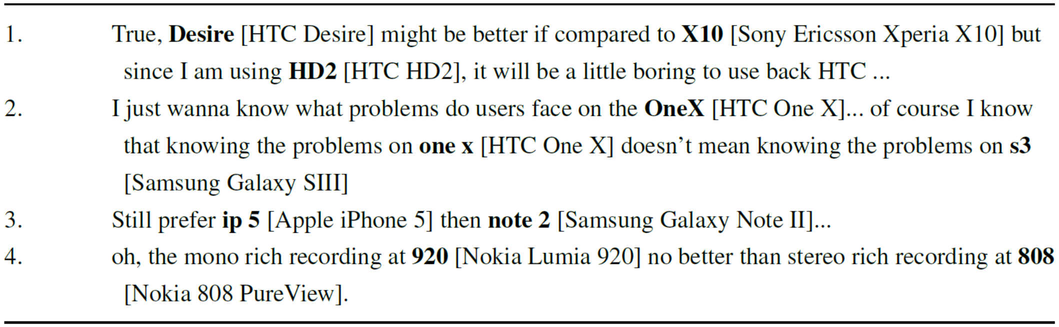
- The table below shows the number of users who have mentioned a phone using a specific name in a forum.



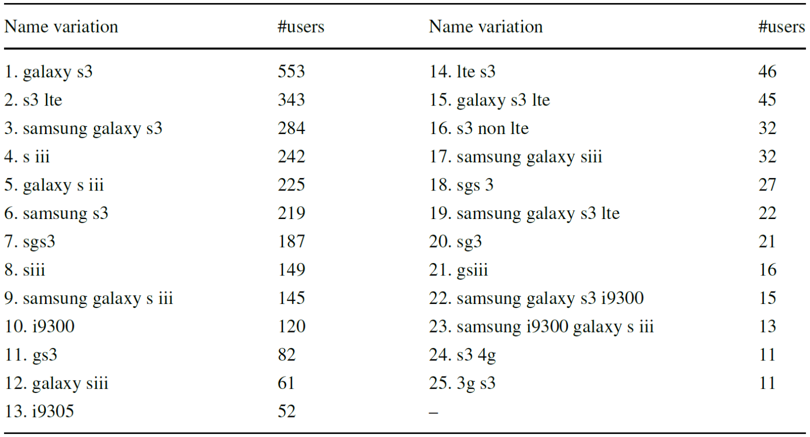
Name variation	#users	Name variation	#users
1. galaxy s3	553	14. lte s3	46
2. s3 lte	343	15. galaxy s3 lte	45
3. samsung galaxy s3	284	16. s3 non lte	32
4. s iii	242	17. samsung galaxy siii	32
5. galaxy s iii	225	18. sgs 3	27
6. samsung s3	219	19. samsung galaxy s3 lte	22
7. sgs3	187	20. sg3	21
8. siii	149	21. gsiii	16
9. samsung galaxy s iii	145	22. samsung galaxy s3 i9300	15
10. i9300	120	23. samsung i9300 galaxy s iii	13
11. gs3	82	24. s3 4g	11
12. galaxy siii	61	25. 3g s3	11
13. i9305	52	–	

Q5: Open-ended Question, for discussion only.

- Task: Assume that we have successfully identified the phone mentions (e.g., 's3 lte,' 'sgs3'). How can we **normalize these mentions to their formal names**?



1. True, **Desire** [HTC Desire] might be better if compared to **X10** [Sony Ericsson Xperia X10] but since I am using **HD2** [HTC HD2], it will be a little boring to use back HTC ...
2. I just wanna know what problems do users face on the **OneX** [HTC One X]... of course I know that knowing the problems on **one x** [HTC One X] doesn't mean knowing the problems on **s3** [Samsung Galaxy SIII]
3. Still prefer **ip 5** [Apple iPhone 5] then **note 2** [Samsung Galaxy Note II]...
4. oh, the mono rich recording at **920** [Nokia Lumia 920] no better than stereo rich recording at **808** [Nokia 808 PureView].



Name variation	#users	Name variation	#users
1. galaxy s3	553	14. lte s3	46
2. s3 lte	343	15. galaxy s3 lte	45
3. samsung galaxy s3	284	16. s3 non lte	32
4. s iii	242	17. samsung galaxy siii	32
5. galaxy s iii	225	18. sgs 3	27
6. samsung s3	219	19. samsung galaxy s3 lte	22
7. sgs3	187	20. sg3	21
8. siii	149	21. gsiii	16
9. samsung galaxy s iii	145	22. samsung galaxy s3 i9300	15
10. i9300	120	23. samsung i9300 galaxy s iii	13
11. gs3	82	24. s3 4g	11
12. galaxy siii	61	25. 3g s3	11
13. i9305	52	–	