Words and Tokenkization

Steve Renals
s.renals@ed.ac.uk
(based on original notes by Ewan Klein)

ICL — 9 October 2005

Text Processing

> *Text processing is arguably what most programmers spend most of their time doing. The information that lives in business software systems mostly comes down to collections of* **words** *about the application domain—maybe with a few special symbols mixed in.*

David Mertz, *Text Processing in Python*

What is a Word (1)?

Notion of ‘word’ is not straightforward

Orthographic word: string of characters with ‘whitespace’ at each end; e.g. *these are words*

Phonological word: ‘words’ which are pronounced as a phonological unit; e.g. *they’ll wanna leave*

Clitic/Leaner: Items which can’t form a phonological word in isolation, but require a host.; e.g. *a(n)*, *’ll*

Lexeme (lexical item): ‘Words in a dictionary’; e.g. HAVE is lexical item corresponding to **grammatical word forms** *have*, *has*, *had*, *having*

What is a Word (2)?

Lemma/Citation Form: Grammatical form that is chosen to represent a lexeme. In English, usually the **base** form (i.e., with no grammatical marking)

Multi-part/Discontinuous Words: Sequences which are multiple orthographic words but exhibit the semantic coherence of words; e.g. *Kim* **rang** *her* **up**

Short Forms: abbreviations (*Dept.*), logograms (£), contractions (*we'll*), acronyms (*BBC*)

Morphology

- Grammatical markings: used to differentiate different forms of a lexeme; e.g., *bake*, *bakes*, *baker*
  - *bake* is the **root** or **stem** form
  - *-s* and *-r* are **morphological affixes** that attach to the root
- **Morpheme** minimal meaning-bearing unit
  - **Stem** "main" morpheme of a word
  - **Affix** "additional" meanings
- **Agglutinative** languages tend to string morphemes together (eg Turkish, Finnish)
- **Stemming** is an operation that strips off grammatical markings to leave the stem; e.g. *foxes* ⇒ *fox*, *flies* ⇒ *fly*
- **Lemmatization** is an operation that specifies the lemma corresponding to a word form; what counts as lemma may vary with application.

Inflectional Morphology

- **Inflectional Morphology** Combination of a word stem with a grammatical morpheme resulting in a word of the **same** class as the stem.
- *bakes* is an inflected form of *bake*
- Examples:
  - **Pluralization** *dog*/*dogs*; *guess*/*guesses*; *spy*/*spies*
  - **Possessive nouns** *Ewan*/*Ewan's*; *Miles*/*Miles'*
  - **Verb forms** *walk*/*walks*/*walking*/*walked*

Derivational Morphology

- **Derivational Morphology** Combination of a word stem with a grammatical morpheme resulting in words of a **different** class.
- *baker* is a derived form of *bake*
- More examples:
  - **Nominalization** *computerize (V)computerization*; *appoint (V)/apponitee*; *run (V)/runner*; *red (A)/redness*
  - **Derived adjectives** *computation (N)/computational*; *laugh (V)/laughable*; *clue (N)/clueless*

Why Tokenize?

- The simplest way to represent a text is with a single string.

```
>>> open("hello.txt").read()
'Hello world!\tThis is a \ntest file.\n'
```

Why Tokenize?

- The simplest way to represent a text is with a single string.

```
>>> open("hello.txt").read()
'Hello world!\tThis is a \ntest file.\n'
```

- We need to identify parts of the string that should undergo further processing, e.g., parsing into grammatical structure.

Why Tokenize?

- The simplest way to represent a text is with a single string.

```
>>> open("hello.txt").read()
'Hello world!\tThis is a \ntest file.\n'
```

- We need to identify parts of the string that should undergo further processing, e.g., parsing into grammatical structure.
- We call the parts **tokens**.

Why Tokenize?

- The simplest way to represent a text is with a single string.

```
>>> open("hello.txt").read()
'Hello world!\tThis is a \ntest file.\n'
```

- We need to identify parts of the string that should undergo further processing, e.g., parsing into grammatical structure.
- We call the parts **tokens**.
- In NLTK, it's convenient to work with a **list** of tokens, typically corresponding to orthographic words:

```
>>> tokens = ['Hello', 'world!', 'This', 'is', 'a', 'test',
>>> for t in tokens:
...     t = t.lower()
...     print t
```

Example

Sea Containers Ltd. said it might increase the price of its $70-a-share buy-back plan if pressed by Temple Holdings Ltd., which made an earlier tender offer for Sea Containers. Sea Containers, a Hamilton, Bermuda-based shipping concern, said Tuesday that it would sell $1.1 billion of assets and use some of the proceeds to buy about 50% of its common shares for $7 apiece.

Simple Word Tokenization

- The simple 'space' tokenizer in NLTK Lite:

```
>>> from nltk_lite.tokenize import *
>>> s = 'This is a\n string.'
>>> simple.space(s)
['This', 'is', 'a\n', 'string.']
```

Simple Word Tokenization

- The simple 'space' tokenizer in NLTK Lite:

```
>>> from nltk_lite.tokenize import *
>>> s = 'This is a\n string.'
>>> simple.space(s)
['This', 'is', 'a\n', 'string.']
```

- `simple.space(s)` just splits string `s` at single spaces.

Simple Word Tokenization

- The simple 'space' tokenizer in NLTK Lite:

```
>>> from nltk_lite.tokenize import *
>>> s = 'This is a\n string.'
>>> simple.space(s)
['This', 'is', 'a\n', 'string.']
```

- `simple.space(s)` just splits string `s` at single spaces.
- Python string method `split` splits at whitespace characters

```
>>> s.split()
['This', 'is', 'a', 'string.']
```

Other Kinds of Token

Tokens can be of various different types:

- words (most usual)

Other Kinds of Token

Tokens can be of various different types:

- words (most usual)
- lines

```
>>> from nltk_lite.tokenize import *
>>> s = 'This is a\n string.'
>>> simple.line(s)
['This is a', ' string.']
>>> s.splitlines()
['This is a', ' string.']
```

- `simple.line(s)` and string method `splitlines` split the string at newlines (`\n`).

Other Kinds of Token

Tokens can be of various different types:

- words (most usual)
- lines

```
>>> from nltk_lite.tokenize import *
>>> s = 'This is a\n string.'
>>> simple.line(s)
['This is a', ' string.']
>>> s.splitlines()
['This is a', ' string.']
```

- `simple.line(s)` and string method `splitlines` split the string at newlines (`\n`).
- sentences (also useful, but tricky!)

Other Kinds of Token

Tokens can be of various different types:

- words (most usual)
- lines

```
>>> from nltk_lite.tokenize import *
>>> s = 'This is a\n string.'
>>> simple.line(s)
['This is a', ' string.']
>>> s.splitlines()
['This is a', ' string.']
```

- `simple.line(s)` and string method `splitlines` split the string at newlines (`\n`).
- sentences (also useful, but tricky!)
- paragraphs

Identifying Tokens

- *Do you mean this this or that this?*
- Three occurrences of *this*; i.e.,
- three **tokens** of the **type** ‘this’.

Word token:	an occurrence of a word form at a particular spatio-temporal location (e.g. a sequential position in a text, an utterance event at a time and place);
Word type:	ideally, the lexeme, but in fact might be a grammatical word form. Tokens **belong** to a given type.

Counting Types vs. Tokens

```
from nltk_lite.corpora import gutenberg

count = {}                          # initialize dictionary

for token in gutenberg.raw('shakespeare-macbeth'):
    token = token.lower()               # normalize case
    if token not in count:              # previously unseen token?
        count[token] = 0                #   if so set count to 0
    count[token] += 1                   # increment token count

lc_tokens = list(gutenberg.raw('shakespeare-macbeth'))

no_tokens = len(lc_tokens)              # 23939
no_types = len(count.keys())            #  3629
tt_ratio = no_tokens/no_types           #     6
```

Problems with Identifying Tokens

- What counts as word token in English often arbitrary: e.g. *ice cream*, *ice-cream*, *icecream*

Problems with Identifying Tokens

- What counts as word token in English often arbitrary: e.g. *ice cream*, *ice-cream*, *icecream*
- Tokenization may also depend on requirements of downstream processing; e.g. maybe treat *we've* as two word tokens and try to find *'ve* as a contracted form in lexicon.

Problems with Identifying Tokens

- What counts as word token in English often arbitrary: e.g. *ice cream*, *ice-cream*, *icecream*
- Tokenization may also depend on requirements of downstream processing; e.g. maybe treat *we've* as two word tokens and try to find *'ve* as a contracted form in lexicon.
- Tokenization decisions can effect part-of-speech tagging

Tokenization and Tagging

- *. . . a previously described FK506-binding protein-associated protein*
- two possible tokenizations, depending on whether we tokenize the first hyphen in its own right:
  - FK506 - binding protein-associated protein
  - FK506-binding protein-associated protein
- This leads to two different part-of-speech, using an existing tagger:

```
FK506\_SYM -_: binding_VBG protein-associated_JJ protein_NN
FK506-binding_JJ protein-associated_JJ protein_NN
```

Reading

- Read NLTK Lite Tutorial *Words: The Building Blocks of Language* at least sections 3.1 and 3.2
- Chapter 3 of Jurafsky and Martin (2nd Ed) (esp. sections 3.1 and 3.9) (http://www.cs.colorado.edu/∼martin/slp2.html#Chapter3)

Summary

- Most text processing makes assumptions about linguistic units; good to be aware of the major distinctions in notion of ‘word’.
- Tokenization into words is important for subsequent processing
- Tokenization into sentences also important
- But not always easy to tokenize in a consistent and sensible manner, and no Right Answer in general.