

Scripting

L700: Automatic Analysis of Learner Language

Autumn 2008

Goals

- ▶ Tokenize text
 - ▶ Including UTF-8 formatted data
- ▶ Convert data from one format to another
 - ▶ Including XML output
- ▶ Extract features from data

We'll work with Python (<http://www.python.org/>) today.

Text tokenization

I'm going to make a tokenizer available for you, based on some regular expression matching

How to use it:

- ▶ At the top of your file, write:
from aux import tokenize
- ▶ Then, for each line of text, call it with the command
tokenize(line)

Let's work towards understanding it ...

Python skeleton for reading in files

```
corpus = "filename..."
file = open(corpus, 'r')
line = file.readline()

while line:
    line = line.rstrip()
    # HERE'S WHERE WE CALL THE TOKENIZER:
    words = tokenize(line)
    ...
    line = file.readline()
file.close()
```

A tokenizer (details do not need to be understood)

```
import re

def tokenize(line):
    list = line.split()
    tokens = []
    for item in list:

        while re.match('\W', item):
            # non-alphanumeric item at beginning of item
            tokens.append(item[0])
            item = item[1:]

        # to maintain order, we use temp
        temp = []
        while re.search('\W$', item):
            # non-alphanumeric item at end of item
            temp.append(item[-1])
            item = item[:-1]
```

A tokenizer (cont.)

```
# Contraction handling
if item == "can't":
    tokens.append("can")
    tokens.append("n't")
# other n't words:
elif re.search("n't", item):
    tokens.append(item[:-3])
    tokens.append(item[-3:])
# other words with apostrophes ('s, 'll, etc.)
elif re.search("'", item):
    wordlist = item.split("'")
    tokens.append(wordlist[0])
    tokens.append("'" + wordlist[1])
# no apostrophe, i.e., normal word:
else:
    tokens.append(item)
tokens.extend(temp)
return tokens
```

Try tokenizing

Download a text and tokenize it ...

UTF-8

We'll probably want to work with more than just English, so knowing how to handle UTF-8 data will be valuable.

- ▶ You can convert between normal strings and unicode strings

```
>>> unicode('hello')
u'hello'
>>> str(u'hello')
'hello'
```

But we may want to use UTF-8 specifically (or some other encoding)

```
>>> 'hello'.encode('utf-8')
```

UTF-8 files

Excerpts for some code I wrote to handle a Russian file:

```
import codecs
file = codecs.open(lex_file,'r', 'utf8')
# can then use normal file reading methods,
# e.g., readline()

...

to_print = [stem,suffix]
print '\t'.join(to_print).encode('utf-8')
```

UTF-8 on a terminal

You may be running a program correctly, but it prints to the screen in a weird way.

On a mac, the way to fix this is to add these lines to your .profile file:

```
# for better unicode support:
export LC_ALL=en_US.UTF-8
export LC_CTYPE=en_US.UTF-8
export LANG=en_US.UTF-8
```

Format conversions

Let's take data from one format (CoNLL) and put it into another format (TnT), so that we can POS tag it

CoNLL format (separated by tabs)

```
ID word    lemma CPOS POS  Features ...
1  Cathy   Cathy N    N    eigen|ev|neut ...      2 su
2  zag     zie    V     V    trans|ovt|1of2of3|ev ... 0 RO
3  hen     hen    Pron  Pron per|3|mv|datofacc ...  2 ob
4  wild    wild  Adj   Adj  attr|stell|onverv ...  5 mo
5  zwaaien zwaai N     N    soort|mv|neut ...     2 vc
6  .       .     Punc  Punc punt ...         5 pu
```

TnT format

Separated by tabs ...

```
word    POS
Cathy   N
zag     V
hen     Pron
wild    Adj
zwaaien N
.       Punc
```

Python skeleton for reading in files

```
#!/bin/env python

corpus = "filename..."
file = open(corpus, 'r')
line = file.readline()

while line:
    line = line.rstrip()
    ...
    line = file.readline()
file.close()
```

Python: CoNLL to TnT

```
#!/bin/env python

corpus = "man.conll"

file = open(corpus, 'r')
line = file.readline()

while line:
    line = line.rstrip()
    list = line.split()
    if list:
        word = list[1]
        tag = list[4]
        print word + '\t' + tag
    else:
        print
    line = file.readline()
file.close()
```

XML output

Let's replace that line `print word + '\t' + tag` with `print make_xml_line(word, tag)`
And then write a function `make_xml_line`

```
def make_xml_line(word, tag):
    outline = '<word tag="'+tag+'">'+word+'</word>'
    return outline
```

Extracting dependencies

Let's now try to extract dependency pairs ... First, before the `while` loop, add:

```
Heads = {}
Words = {}
```

And change this part:

```
if list:
    id = list[0]
    word = list[1]
    head_id = list[6]
    Words[id] = word
    Heads[id] = head_id
```

Extracting dependencies (cont.)

```
else:
    # end of sentence
    for id in Heads:
        head_id = Heads[id]
        if head_id != "0":
            if int(id) < int(head_id):
                print "DH:\t"+Words[id]+' \t'+Words[head_id]
            else:
                print "HD:\t"+Words[head_id]+' \t'+Words[id]
```

NB: We also need to put this after the loop, in case there is no trailing new line

Extraction practice

Let's try extracting the text which makes up different NPs (i.e., the *yield*) in a small TIGER corpus example.

Questions we need to ask:

- ▶ What format is the data in?
- ▶ What kind of data structures do we need?

Feature extraction

What we've already been doing is essentially a way of extracting features.

Let's say, for example, that we want to extract a word, its POS, and its head and its head's POS from a .conll file.

- ▶ NB: We'll have ROOT be both the head word and the head POS for words which are the ROOT.

Let's write this up, printing the features out into a comma-separated file

Using Unix

For some of what we've done, Unix offers various useful utilities

The `cut` command allows you to select fields from which to cut out the parts you want

- ▶ It assumes tab-delimited data (but you can change that with the `-d` option)
- ▶ `-f` allows you to specify which columns to cut
- ▶ In this context, we have: `cut -f2,5 man.conll`

See, e.g., Ken Church's *Unix for Poets*,
<http://people.sslmit.unibo.it/~baroni/compling04/UnixforPoets.pdf>