

Two-day workshop

Automated Content Analysis with Python

Day 1

Damian Trilling

d.c.trilling@uva.nl

@damian0604

www.damiantrilling.net

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

31-1-2017

Today

- 1 What's Automated Content Analysis?
- 2 Recap: The very, very, basics of programming with Python
 - Datatypes
 - Indentation: The Python way of structuring your program
- 3 Sentiment Analysis
 - Bag-of-words approaches
 - Advanced approaches
- 4 Basic ACA: Dictionary- and string-based methods
 - Stopword removal
 - Natural language processing
 - A simple algorithm
 - Regular expressions
 - Some more Natural Language Processing

What's Automated Content Analysis?

	Methodological approach		
	<i>Counting and Dictionary</i>	<i>Supervised Machine Learning</i>	<i>Unsupervised Machine Learning</i>
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis
<div> <div>deductive</div> <div>inductive</div> </div>			

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant automated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, 4(1), 8–23.
doi:10.1080/21670811.2015.1096598

Today

- We'll start with counting/dictionary-based methods
- Sentiment analysis
- Basic ACA: Natural language processing and regular expressions
- But first: A short recap.

Recap: The very, very, basics of programming

You can read all this back in Chapter 4.

Basic datatypes (variables)

```
int 32
float 1.75
bool True, False
string "Damian"
```

Basic datatypes (variables)

```
(variable name  firstname)
```

"firstname" and firstname is not the same.

Basic datatypes (variables)

```
(variable name  firstname)
```

"firstname" and firstname is not the same.

"5" and 5 is not the same.

But you can transform it: `int("5")` will return 5.

You cannot calculate `3 * "5"` Actually, you can. It gives you `"555"`.

But you can calculate `3 * int("5")`

More advanced datatypes

Python lingo

More advanced datatypes

```
list firstnames = ['Damian', 'Lori', 'Bjoern']  
    lastnames =  
    ['Trilling', 'Meester', 'Burscher']
```

```
list firstnames = ['Damian', 'Lori', 'Bjoern']
    lastnames =
    ['Trilling', 'Meester', 'Burscher']
list ages = [18, 22, 45, 23]
```

Python lingo

More advanced datatypes

```
list firstnames = ['Damian','Lori','Bjoern']
      lastnames =
      ['Trilling','Meester','Burscher']

list ages = [18,22,45,23]

dict familynames= {'Bjoern':  'Burscher',
                   'Damian':  'Trilling', 'Lori':  'Meester'}

dict {'Bjoern':  26, 'Damian':  31, 'Lori':
      25}
```

Functions

Python lingo

Functions

functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")`
returns the integer 5.

functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

methods are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

functions Take an input and return something else
`int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

methods are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

Both functions and methods end with (). Between the (), *arguments* can (sometimes have to) be supplied.

Indentation: The Python way of structuring your program

Indention

Structure

The program is structured by TABs or SPACEs

- I. Most Popular Technologies
- II. Most Loved, Dreaded, and Wanted Tools
- III. Desktop Operating System
- IV. Text Editor
- V. IDE Theme
- VI. Source Control

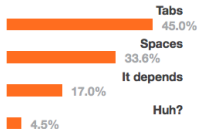
VII. Tabs vs. Spaces

VIII. Caffeine

Looking for a job?

about 10% of developers still don't use it.

VII. TABS VS. SPACES



25,807 responses

After millennia of heated debate, mercifully, at long last, we have an answer. **Most developers prefer tabs to spaces.**

Upon closer examination of the data, a trend emerges: Developers increasingly prefer spaces as they gain experience. Stack Overflow reputation correlates with a preference for spaces, too: users who have 10,000 rep or more prefer spaces to tabs at a ratio of 3 to 1.

Structure

The program is structured by TABs or SPACEs

```
1 firstnames=['Damian','Lori','Bjoern']
2 age={'Bjoern': 27, 'Damian': 32, 'Lori': 26}
3 print ("The names and ages of all BigData people:")
4 for naam in firstnames:
5     print (naam,age[naam])
```

Indentation

Structure

The program is structured by TABs or SPACES

```
1 firstnames=['Damian','Lori','Bjoern']
2 age={'Bjoern': 27, 'Damian': 32, 'Lori': 26}
3 print ("The names and ages of all BigData people:")
4 for naam in firstnames:
5     print (naam,age[naam])
```

Don't mix up TABs and spaces! Both are valid, but you have to be consequent!!! Best: always use 4 spaces!

The program is structured by TABs or SPACEs

```
1 print ("The names and ages of all BigData people:")
2 for naam in firstnames:
3     print (naam,age[naam])
4     if naam=="Damian":
5         print ("He teaches this course")
6     elif naam=="Lori":
7         print ("She was an assistant last year")
8     elif naam=="Bjoern":
9         print ("He helps on Wednesdays")
10    else:
11        print ("No idea who this is")
```

Indentation

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indentation

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indentation of the block indicates that

Indentation

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indentation of the block indicates that

- it is to be executed repeatedly (for statement) – e.g., for each element from a list

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (for statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (if, elif, and else statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (for statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (if, elif, and else statements)
- an alternative block should be executed if an error occurs (try and except statements)

Indentation

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indentation of the block indicates that

- it is to be executed repeatedly (for statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (if, elif, and else statements)
- an alternative block should be executed if an error occurs (try and except statements)
- a file is opened, but should be closed again after the block has been executed (with statement)

Exercise

Sentiment analysis

What is sentiment analysis?

Extracting subjective information from texts

What is sentiment analysis?

Extracting subjective information from texts

- the author's attitude towards the topic of the text

What is sentiment analysis?

Extracting subjective information from texts

- the author's attitude towards the topic of the text
- *polarity*: negative—positive

What is sentiment analysis?

Extracting subjective information from texts

- the author's attitude towards the topic of the text
- *polarity*: negative—positive
- *subjectivity*: neutral—subjective *

What is sentiment analysis?

Extracting subjective information from texts

- the author's attitude towards the topic of the text
- *polarity*: negative—positive
- *subjectivity*: neutral—subjective *
- advanced approaches: different emotions

What is sentiment analysis?

Extracting subjective information from texts

- the author's attitude towards the topic of the text
- *polarity*: negative—positive
- *subjectivity*: neutral—subjective *
- advanced approaches: different emotions

* Less sophisticated approaches do not see this as a sperate dimension but simply calculate $objectivity = 1 - (negativity + positivity)$

Example

```
1 >>> sentiment("Great service by @NSHighspeed")
2 (0.8, 0.75)
3 >>> sentiment("Bad service by @NSHighspeed")
4 (-0.6166666666666667, 0.6666666666666666)
```

(polarity, subjectivity) with

$$-1 \leq \textit{polarity} \leq +1$$

$$0 \leq \textit{subjectivity} \leq +1$$

This is the module `pattern.nl`, available for Python 2 only.

De Smedt, T., & Daelemans W. (2012). Pattern for Python. *Journal of Machine Learning Research*, 13, 2063-2067.

Bag-of-words approaches

How does it work?

- We take each word of a text and look if it's positive or negative.

Bag-of-words approaches

How does it work?

- We take each word of a text and look if it's positive or negative.
 - Most simple way: compare it with a list of negative words and with a list of positive words

Bag-of-words approaches

How does it work?

- We take each word of a text and look if it's positive or negative.
 - Most simple way: compare it with a list of negative words and with a list of positive words
 - More advanced: look up a subjectivity score from a table

Bag-of-words approaches

How does it work?

- We take each word of a text and look if it's positive or negative.
 - Most simple way: compare it with a list of negative words and with a list of positive words
 - More advanced: look up a subjectivity score from a table
- e.g., add up the scores and average them.

How to do this

(given a *string* `tekst` that you want to analyze and two *lists* of strings with negative and positive words, `lijstpos=["great","fantastic",...,"perfect"]` and `lijstneg`)

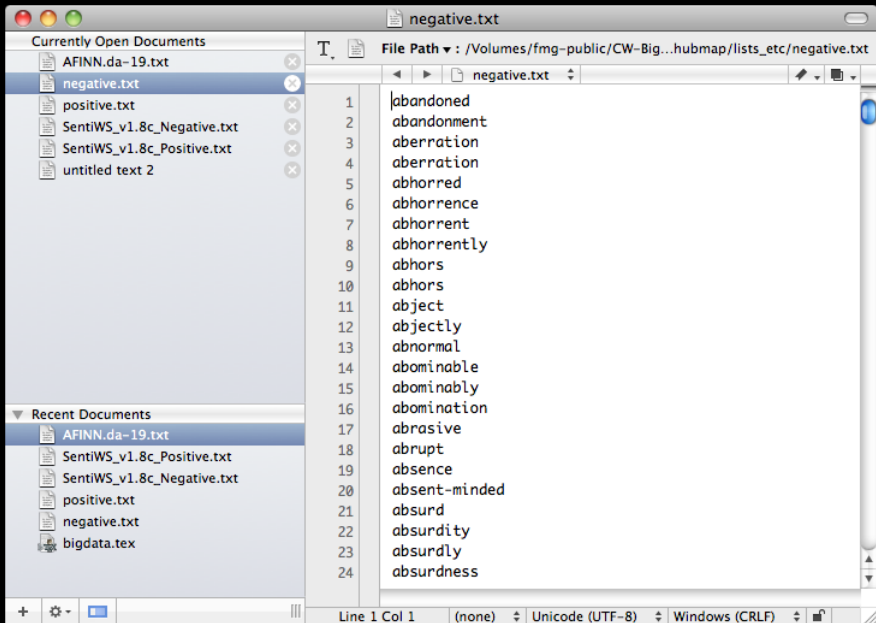
```
1 sentiment=0
2 for woord in tekst.split():
3     if woord in lijstpos:
4         sentiment=sentiment+1 #same as sentiment+=1
5     elif woord in lijstneg:
6         sentiment=sentiment-1 #same as sentiment-=1
7 print (sentiment)
```

Do we need to have the lists in our program itself?

No.

You could have them in a separate text file, one per row, and then read that file directly to a list.

```
1 poslijst=open("filewithonepositivewordperline.txt").read().splitlines()
2 neglijst=open("filewithonenegativewordperline.txt").read().splitlines()
```



More advanced versions

- CSV files or similar tables with weights
- Or some kind of dict?

AFINN.da-19.txt

Currently Open Documents

- AFINN.da-19.txt
- negative.txt
- positive.txt
- SentiWS_v1.8c_Negative.txt
- SentiWS_v1.8c_Positive.txt
- untitled text 2

Recent Documents

- AFINN.da-19.txt
- SentiWS_v1.8c_Positive.txt
- SentiWS_v1.8c_Negative.txt
- positive.txt
- negative.txt
- bigdata.tex

File Path: /Volumes/fmg-public/CW-Big...ap/lists_etc/AFINN.da-19.txt

AFINN.da-19.txt

1	absorberet	1
2	acceptere	1
3	accepterede	1
4	accepterer	1
5	accepteres	1
6	accepteret	1
7	advare	-2
8	advarede	-2
9	advarer	-2
10	advaret	-2
11	advarsel	-3
12	advarsler	-3
13	advarslerne	-3
14	afbrudt	-2
15	afbryde	-2
16	afbrydelse	-2
17	afbrydelser	-2
18	afbrydelserne	-2
19	afbryder	-2
20	affald	-1
21	afgift	-1
22	afgifter	-1
23	afhængig	-1
24	afhængige	-1

Line 1 Col 1 (none) Unicode (UTF-8, with BOM) Wind...CRLF

Currently Open Documents

- AFINN.da-19.txt
- negative.txt
- positive.txt
- SentiWS_v1.8c_Negative.txt
- SentiWS_v1.8c_Positive.txt
- untitled text 2

Recent Documents

- AFINN.da-19.txt
- SentiWS_v1.8c_Positive.txt
- SentiWS_v1.8c_Negative.txt
- positive.txt
- negative.txt
- bigdata.tex

File Path: /Volumes/fmg-public/CW-Big...tc/SentiWS_v1.8c_Negative.txt

SentiWS_v1.8c_Negative.txt

1	Abbau INN	-0.058	Abbaus,Abbaues,Abbauen,Abbaue
2	Abbruch INN	-0.0048	
...	Abbruches,Abbrüche,Abbruchs,Abbrüchen		
3	Abdankung INN	-0.0048	Abdankungen
4	Abdämpfung INN	-0.0048	Abdämpfungen
5	Abfall INN	-0.0048	
...	Abfalles,Abfälle,Abfalls,Abfällen		
6	Abfuhr INN	-0.3367	Abfahren
7	Abgrund INN	-0.3465	
8	Abhängigkeit INN	-0.3653	Abhängigkeiten
9	Ablehnung INN	-0.5118	Ablehnungen
10	Ablenkung INN	-0.0435	Ablenkungen
11	Abnahme INN	-0.0048	Abnahmen
12	Abneigung INN	-0.0048	Abneigungen
13	Abnutzung INN	-0.0048	
14	Abriss INN	-0.0048	
...	Abrisse,Abrissen,Abrisses,Abriss		
15	Abrutsch INN	-0.0048	
...	Abrutschen,Abrutsche,Abrutsches,Abrutschs		
16	Abschaffung INN	-0.058	Abschaffungen
17	Abschreckung INN	-0.0048	Abschreckungen
18	Abschreibung INN	-0.3345	Abschreibungen
19	Abschuß INN	-0.0048	
20	Abschwächung INN	-0.1935	Abschwächungen

Line 1 Col 1 (none) Unicode (UTF-8) Unix (LF) 216...

Bag-of-words approaches

e.g., Schut, L. (2013). Verenigde Staten vs. Verenigd Koninkrijk: Een automatische inhoudsanalyse naar verklarende factoren voor het gebruik van positive campaigning en negative campaigning door vooraanstaande politici en politieke partijen op Twitter. *Bachelor Thesis*, Universiteit van Amsterdam.

Bag-of-words approaches

pro

- easy to implement
- easy to modify:
 - add or remove words
 - make new lists for other languages, other categories (than positive/negative), ...
- easy to understand (transparency, reproducibility)

e.g., Schut, L. (2013). Verenigde Staten vs. Verenigd Koninkrijk: Een automatische inhoudsanalyse naar verklarende factoren voor het gebruik van positive campaigning en negative campaigning door vooraanstaande politici en politieke partijen op Twitter. *Bachelor Thesis*, Universiteit van Amsterdam.

Bag-of-words approaches

con

- simplistic assumptions
- e.g., intensifiers cannot be interpreted ("really" in "really good" or "really bad")
- or, even more important, negations.

Sentiment analysis: Advanced approaches

Improving the BOW approach

Example: The Sentistrength algorithm

- $-5 \dots -1$ and $+1 \dots +5$
- spelling correction
- "booster word list" for strengthening/weakening the effect of the following word
- interpreting repeated letters ("baaaaaad"), CAPITALS and !!!
- idioms
- negation
- Idots

Thelwall, M., Buckley, K., & Paltoglou, G. (2012). Sentiment strength detection for the social Web. *Journal of the American Society for Information Science and Technology*, 63(1), 163-173.

Advanced approaches

Take the structure of a text into account

- Try to apply linguistics concepts to identify sentence structure
- can identify negations
- can interpret intensifiers

Example

```
1 from pattern.nl import sentiment
2 >>> sentiment("Great service by @NSHighspeed")
3 (0.8, 0.75)
4 >>> sentiment("Really")
5 (0.0, 1.0)
6 >>> sentiment("Really Great service by @NSHighspeed")
7 (1.0, 1.0)
```

(polarity, subjectivity) with
 $-1 \leq \text{polarity} \leq +1$
 $0 \leq \text{subjectivity} \leq +1$)

Unlike in pure bag-of-words approaches, here, the overall sentiment is not just the sum or the average of its parts!

De Smedt, T., & Daelemans W. (2012). Pattern for Python. *Journal of Machine Learning Research*, 13, 2063-2067.

Advanced approaches

Advanced approaches

pro

- understand intensifiers or negation
- thus: higher accuracy

Advanced approaches

pro

- understand intensifiers or negation
- thus: higher accuracy

con

- Black box? Or do we understand the algorithm?
- Difficult to adapt to own needs
- *really* much better results?

Exercise

Stopword removal: What and why?

Why remove stopwords?

- If we want to identify key terms (e.g., by means of a word count), we are not interested in them
- If we want to calculate document similarity, it might be inflated
- If we want to make a word co-occurrence graph, irrelevant information will dominate the picture

Stopword removal: How

```
1 testo='He gives her a beer and a cigarette.'  
2 testonuovo=""  
3 stopwords=['and','the','a','or','he','she','him','her']  
4 for verbo in testo.split():  
5     if verbo not in stopwords:  
6         testonuovo=testonuovo+verbo+" "
```

What do we get if we do:

```
1 print (testonuovo)
```

Can you explain the algorithm?

We get:

```
1 >>> print (testonuevo)
2 'He gives beer cigarettes. '
```

Why is "He" still in there?

How can we fix this?

Stopword removal

```
1 testo='He gives her a beer and a cigarette.'  
2 testonuovo=""  
3 stopwords=['and','the','a','or','he','she','him','her']  
4     for verbo in testo.split():  
5         if verbo.lower() not in stopwords:  
6             testonuovo=testonuovo+verbo+" "
```

Automated content analysis using regular expressions

Regular Expressions: What and why?

What is a regexp?

- a *very* widespread way to describe patterns in strings

Regular Expressions: What and why?

What is a regexp?

- a *very* widespread way to describe patterns in strings
- Think of wildcards like `*` or operators like OR, AND or NOT in search strings: a regexp does the same, but is *much* more powerful

Regular Expressions: What and why?

What is a regexp?

- a *very* widespread way to describe patterns in strings
- Think of wildcards like `*` or operators like OR, AND or NOT in search strings: a regexp does the same, but is *much* more powerful
- You can use them in many editors (!), in the Terminal, in STATA ... and in Python

An example

From last week's task

- We wanted to remove everything but words from a tweet
- We did so by calling the `.replace()` method
- We could do this with a regular expression as well:
 `[^a-zA-Z]` would match anything that is not a letter

Basic regexp elements

Alternatives

`[TtFf]` matches either T or t or F or f

`Twitter|Facebook` matches either Twitter or Facebook

`.` matches any character

Basic regexp elements

Alternatives

`[TtFf]` matches either T or t or F or f

`Twitter|Facebook` matches either Twitter or Facebook

`.` matches any character

Repetition

* the expression before occurs 0 or more times

+ the expression before occurs 1 or more times

regex quizz

Which words would be matched?

① [Pp]ython

regex quizz

Which words would be matched?

- ❶ [Pp]ython
- ❷ [A-Z] +

regex quizz

Which words would be matched?

- ❶ [Pp]ython
- ❷ [A-Z] +
- ❸ RT : * @[a-zA-Z0-9] *

What else is possible?

If you google regexp or regular expression, you'll get a bunch of useful overviews. The wikipedia page is not too bad, either.

How to use regular expressions in Python

The module re

`re.findall("[Tt]witter|[Ff]acebook", testo)` returns a list with all occurrences of Twitter or Facebook in the string called `testo`

`re.findall("[0-9]+[a-zA-Z]+", testo)` returns a list with all words that start with one or more numbers followed by one or more letters in the string called `testo`

How to use regular expressions in Python

The module re

`re.findall("[Tt]witter|[Ff]acebook",testo)` returns a list with all occurrences of Twitter or Facebook in the string called `testo`

`re.findall("[0-9]+[a-zA-Z]+",testo)` returns a list with all words that start with one or more numbers followed by one or more letters in the string called `testo`

`re.sub("[Tt]witter|[Ff]acebook","a social medium",testo)` returns a string in which all occurrences of Twitter or Facebook are replaced by "a social medium"

How to use regular expressions in Python

The module re

`re.match(" +([0-9]+) of ([0-9]+) points",line)` returns `None` unless it *exactly* matches the string `line`. If it does, you can access the part between `()` with the `.group()` method.

Example:

```
1 line="          2 of 25 points"
2 result=re.match(" +([0-9]+) of ([0-9]+) points",line)
3 if result:
4     print ("Your points:",result.group(1))
5     print ("Maximum points:",result.group(2))
```

Your points: 2

Maximum points: 25

Possible applications

Data preprocessing

- Remove unwanted characters, words, ...
- Identify *meaningful* bits of text: usernames, headlines, where an article starts, ...
- filter (distinguish relevant from irrelevant cases)

Possible applications

Data analysis: Automated coding

- Actors
- Brands
- links or other markers that follow a regular pattern
- Numbers (!)

Example 1: Counting actors

```
1 import re, csv
2 from os import listdir, path
3 mypath = "/home/damian/artikelen"
4 filename_list=[]
5 matchcount54_list=[]
6 matchcount10_list=[]
7 onlyfiles = [f for f in listdir(mypath) if path.isfile(path.join(mypath,
8     f)))]
9 for f in onlyfiles:
10     matchcount54=0
11     matchcount10=0
12     with open(path.join(mypath,f),mode="r",encoding="utf-8") as fi:
13         artikel=fi.readlines()
14         for line in artikel:
15             matches54 = re.findall('Israel.*(minister|politician.*|[Aa]
16                 uthorit)',line)
17             matches10 = re.findall('[Pp]alest',line)
18             matchcount54+=len(matches54)
19             matchcount10+=len(matches10)
20             filename_list.append(f)
21             matchcount54_list.append(matchcount54)
22             matchcount10_list.append(matchcount10)
23 output=zip(filename_list,matchcount10_list,matchcount54_list)
24 with open("overzichtstabel.csv", mode='w',encoding="utf-8") as fo:
25     writer = csv.writer(fo)
26     writer.writerows(output)
```


Example 2: Which number has this Lexis Nexis article?

```
1           All Rights Reserved
2
3           2 of 200 DOCUMENTS
4
5           De Telegraaf
6
7           21 maart 2014 vrijdag
8
9 Brussel bereikt akkoord aanpak probleembanken;
10 ECB krijgt meer in melk te brokkelen
11
12 SECTION: Finance; Blz. 24
13 LENGTH: 660 woorden
14
15 BRUSSEL Europa heeft gisteren op de valreep een akkoord bereikt
16 over een saneringsfonds voor banken. Daarmee staat de laatste
```

Example 2: Check the number of a lexis nexis article

```
1 All Rights Reserved
2
3 2 of 200 DOCUMENTS
4
5 De Telegraaf
6
7 21 maart 2014 vrijdag
8
9 Brussel bereikt akkoord aanpak probleembanken;
10 ECB krijgt meer in melk te brokkelen
11
12 SECTION: Finance; Blz. 24
13 LENGTH: 660 woorden
14
15 BRUSSEL Europa heeft gisteren op de valreep een akkoord bereikt
16 over een saneringsfonds voor banken. Daarmee staat de laatste
```

```
1 for line in tekst:
2     matchObj=re.match(r" +([0-9]+) of ([0-9]+) DOCUMENTS",line)
3     if matchObj:
4         numberofarticle= int(matchObj.group(1))
5         totalnumberofarticles= int(matchObj.group(2))
```

Practice yourself!

`http://www.pyregex.com/`

Some more Natural Language Processing

Some more NLP: What and why?

What can we do?

- remove stopwords (*last week*)

Some more NLP: What and why?

What can we do?

- remove stopwords (*last week*)
- stemming

Some more NLP: What and why?

What can we do?

- remove stopwords (*last week*)
- stemming
- Parse sentences (advanced)

NLP: What and why?

Why do stemming?

- Because we do not want to distinguish between smoke, smoked, smoking, . . .
- Typical preprocessing step (like stopwords removal)

Stemming

(with NLTK, see Bird, S., Loper, E., & Klein, E. (2009). *Natural language processing with Python*. Sebastopol, CA: O'Reilly.)

```
1 from nltk.stem.snowball import SnowballStemmer
2 stemmer=SnowballStemmer("english")
3 frase="I am running while generously greeting my neighbors"
4 frasenuevo=""
5 for palabra in frase.split():
6     frasenuevo=frasenuevo + stemmer.stem(palabra) + " "
```

If we now did `print(frasenuevo)`, it would return:

```
1 i am run while generous greet my neighbor
```

Stemming and stopwords removal - let's combine them!

```
1 from nltk.stem.snowball import SnowballStemmer
2 from nltk.corpus import stopwords
3 stemmer=SnowballStemmer("english")
4 stopwords = stopwords.words("english")
5 frase="I am running while generously greeting my neighbors"
6 frasenuevo=""
7 for palabra in frase.lower().split():
8     if palabra not in stopwords:
9         frasenuevo=frasenuevo + stemmer.stem(palabra) + " "
```

Now, `print(frasenuevo)` returns:

```
1 run generous greet neighbor
```

Perfect!

Stemming and stopwords removal - let's combine them!

```
1 from nltk.stem.snowball import SnowballStemmer
2 from nltk.corpus import stopwords
3 stemmer=SnowballStemmer("english")
4 stopwords = stopwords.words("english")
5 frase="I am running while generously greeting my neighbors"
6 frasenuevo=""
7 for palabra in frase.lower().split():
8     if palabra not in stopwords:
9         frasenuevo=frasenuevo + stemmer.stem(palabra) + " "
```

Now, `print(frasenuevo)` returns:

```
1 run generous greet neighbor
```

Perfect!

In order to use `nltk.corpus.stopwords`, you have to download that module once. You can do so by typing the following in the Python console and selecting the appropriate package from the menu that pops up:

```
import nltk
nltk.download()
```

NB: Don't download everything, that's several GB.

NLTK Downloader

File View Sort Help

Collections Corpora Models All Packages

Identifier	Name	Size	Status
senseval	SENSEVAL 2 Corpus: Sense Tagged Text	2.1 MB	not instal
sentiwordnet	SentiWordNet	4.5 MB	not instal
shakespeare	Shakespeare XML Corpus Sample	464.3 KB	not instal
sinica_treebank	Sinica Treebank Corpus Sample	878.2 KB	not instal
smultron	SMULTRON Corpus Sample	162.3 KB	not instal
state_union	C-Span State of the Union Address Corpus	789.8 KB	not instal
stopwords	Stopwords Corpus	8.5 KB	not instal
swadesh	Swadesh Wordlists	22.3 KB	not instal
switchboard	Switchboard Corpus Sample	772.6 KB	not instal
timit	TIMIT Corpus Sample	21.2 MB	not instal
toolbox	Toolbox Sample Files	244.7 KB	not instal
treebank	Penn Treebank Sample	1.6 MB	not instal
udhr	Universal Declaration of Human Rights Corpu	1.1 MB	not instal
udhr2	Universal Declaration of Human Rights Corpu	1.6 MB	not instal
unicode_samples	Unicode Samples	1.2 KB	not instal
universal_treebank	Universal Treebanks Version 2.0	24.7 MB	not instal

Download

Refresh

Server Index: http://nltk.github.com/nltk_data/Download Directory: /home/damian/nltk_data

In [5]: import nltk

In [6]: nltk.download()

NLP: What and why?

Why parse sentences?

- To find out what grammatical function words have
- and to get closer to the meaning.

Parsing a sentence

```
1 import nltk
2 sentence = "At eight o'clock on Thursday morning, Arthur didn't feel
   very good."
3 tokens = nltk.word_tokenize(sentence)
4 print (tokens)
```

`nltk.word_tokenize(sentence)` is similar to `sentence.split()`, but compare handling of punctuation and the `didn't` in the output:

```
1 ['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did',
   "n't", 'feel', 'very', 'good', '.']
```

Parsing a sentence

Now, as the next step, you can “tag” the tokenized sentence:

```
1 tagged = nltk.pos_tag(tokens)
2 print (tagged[0:6])
```

gives you the following:

```
1 [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
2  ('Thursday', 'NNP'), ('morning', 'NN')]
```

Parsing a sentence

Now, as the next step, you can “tag” the tokenized sentence:

```
1 tagged = nltk.pos_tag(tokens)
2 print (tagged[0:6])
```

gives you the following:

```
1 [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
2  ('Thursday', 'NNP'), ('morning', 'NN')]
```

And you could get the word type of "morning" with
tagged[5][1]!

More NLP

Look at <http://nltk.org>

Exercise