Two-day workshop
Automated Content Analysis with Python
Day 1 – Afternoon

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

1–5–2017

This afternoon

1. What's Automated Content Analysis?
2. Sentiment Analysis
   - Bag-of-words approaches
   - Advanced approaches
3. 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		
	Counting and Dictionary	*Supervised Machine Learning*	*Unsupervised Machine Learning*
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

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant autmated 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

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

```
>>> sentiment("Great service by @NSHighspeed")
(0.8, 0.75)
>>> sentiment("Bad service by @NSHighspeed")
(-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

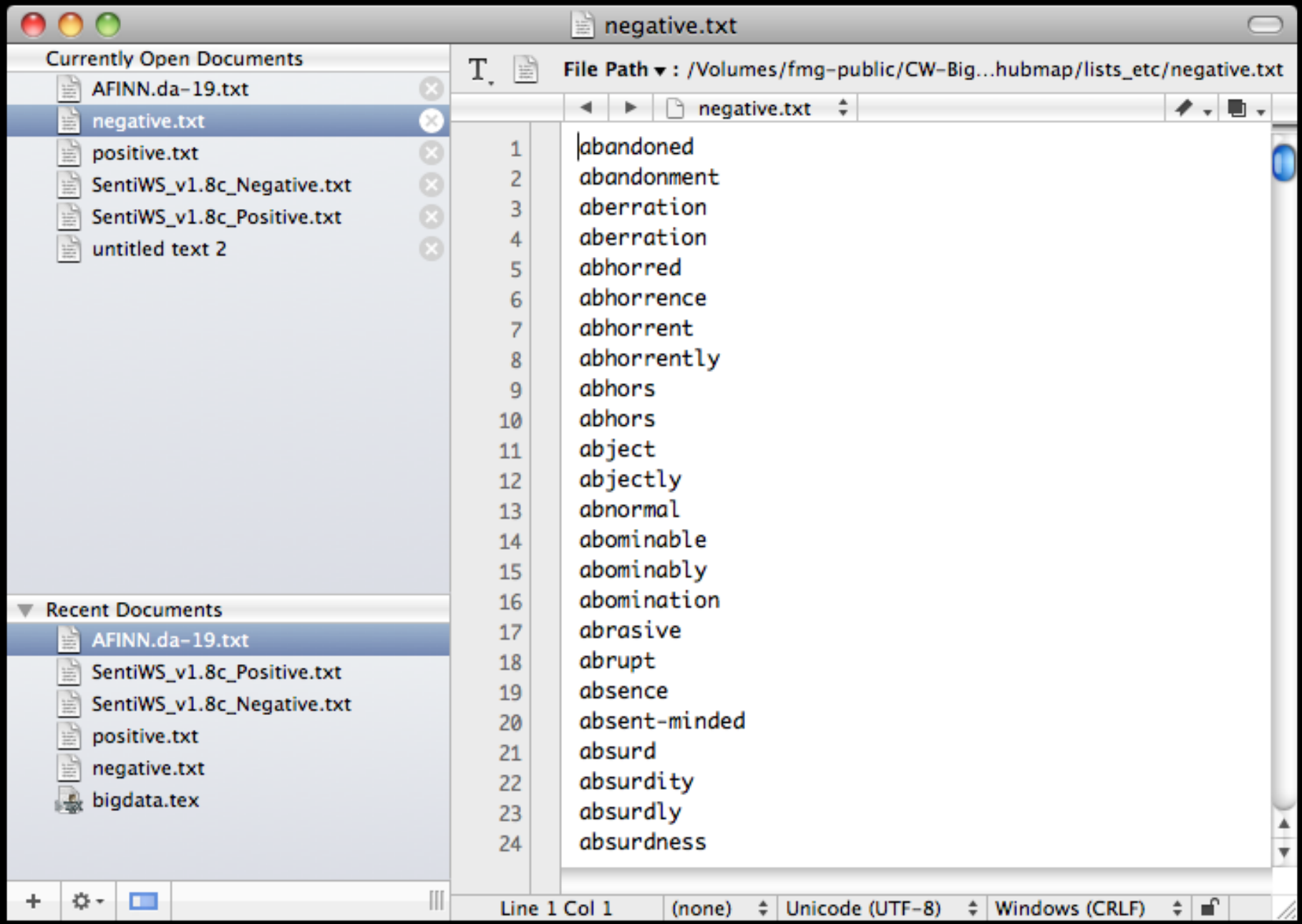
(given a *string* `tekst` that you want to analyze and two *lists* of strings with negative and positive words, `lijstpos=["great","fantastic",...,"perfect"]` and `lijstneg`)

```
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?

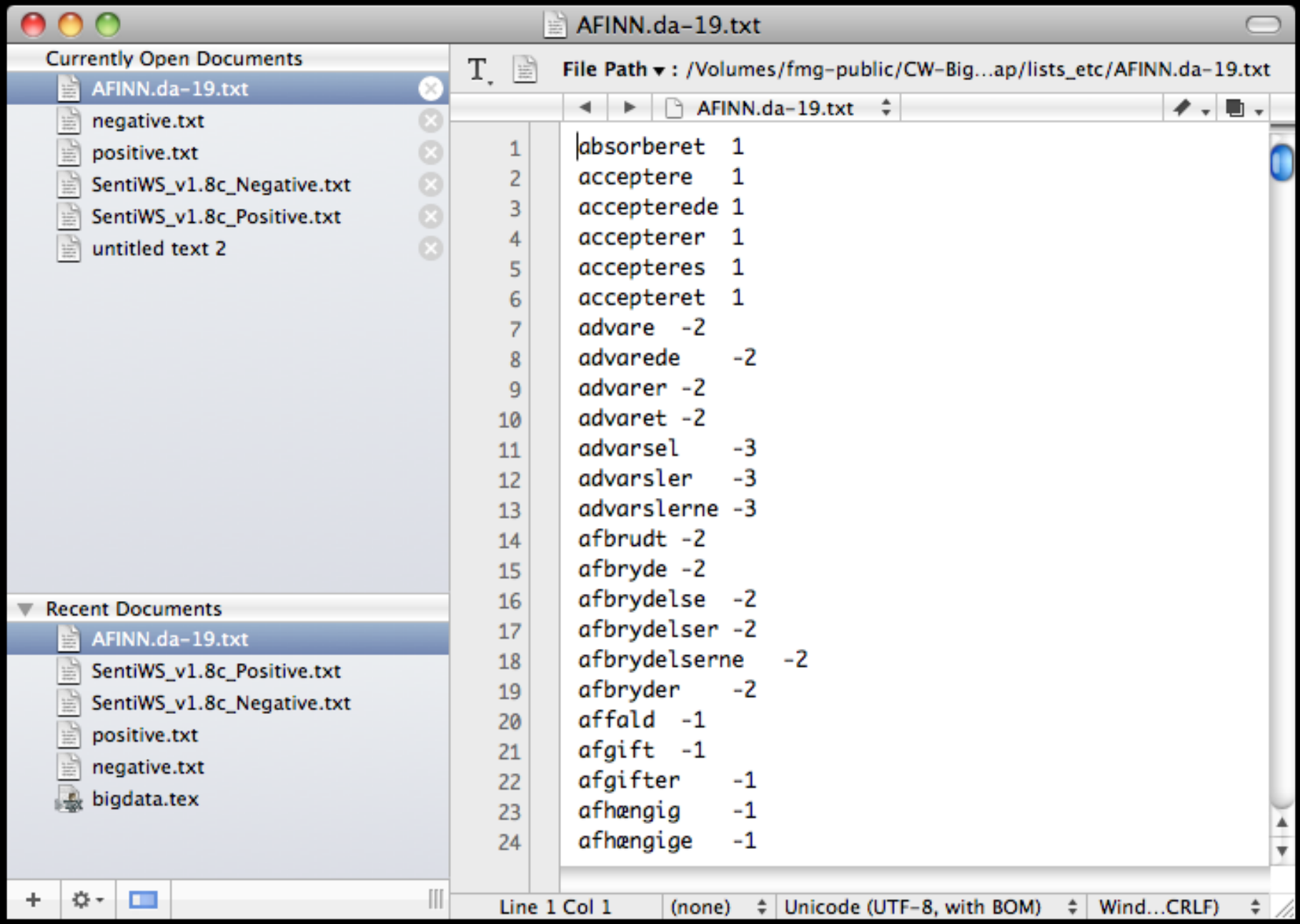
No.
You could have them in a separate text file, one per row, and then read that file directly to a list.

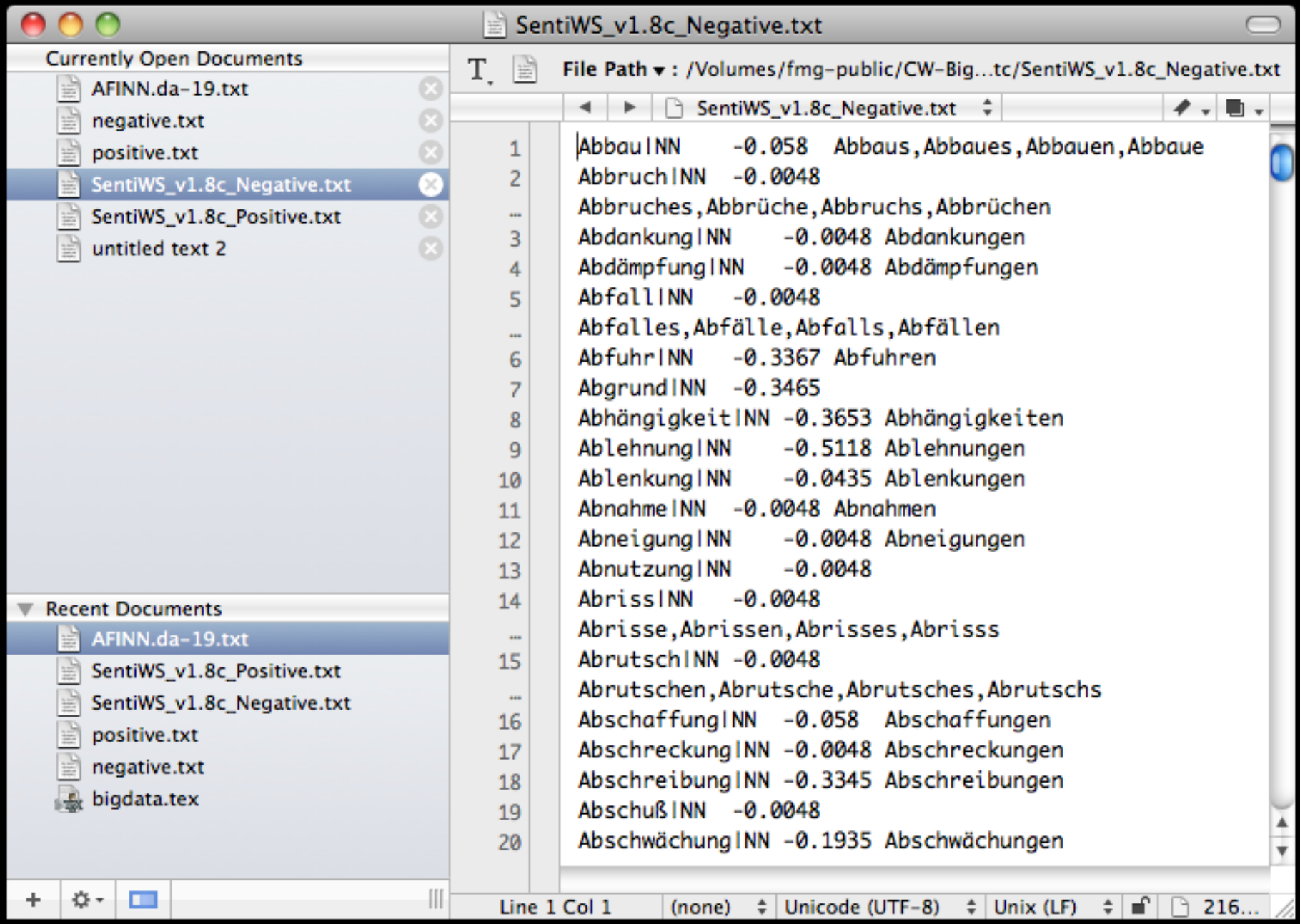
```
1 poslijst=open("filewithonepositivewordperline.txt").read().splitlines()
2 neglijst=open("filewithonenegativewordperline.txt").read().splitlines()
```


negative.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...hubmap/lists_etc/negative.txt
negative.txt
1 abandoned
2 abandonment
3 aberration
4 aberration
5 abhorred
6 abhorrence
7 abhorrent
8 abhorrently
9 abhors
10 abhors
11 abject
12 abjectly
13 abnormal
14 abominable
15 abominably
16 abomination
17 abrasive
18 abrupt
19 absence
20 absent-minded
21 absurd
22 absurdity
23 absurdly
24 absurdness
Line 1 Col 1
(none)
Unicode (UTF-8)
Windows (CRLF)

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
absorberet 1
acceptere 1
accepterede 1
accepterer 1
accepteres 1
accepteret 1
advare -2
advarede -2
advarer -2
advaret -2
advarsel -3
advarsler -3
advarslerne -3
afbrudt -2
afbryde -2
afbrydelse -2
afbrydelser -2
afbrydelserne -2
afbryder -2
affald -1
afgift -1
afgifter -1
afhængig -1
afhængige -1
Line 1 Col 1
(none)
Unicode (UTF-8, with BOM)
Wind...CRLF)


SentiWS_v1.8c_Negative.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...tc/SentiWS_v1.8c_Negative.txt
SentiWS_v1.8c_Negative.txt
1 Abbau|NN -0.058 Abbaus,Abbaues,Abbauen,Abbaue
2 Abbruch|NN -0.0048
... Abbruches,Abbrüche,Abbruchs,Abbrüchen
3 Abdankung|NN -0.0048 Abdankungen
4 Abdämpfung|NN -0.0048 Abdämpfungen
5 Abfall|NN -0.0048
... Abfalles,Abfälle,Abfalls,Abfällen
6 Abfuhr|NN -0.3367 Abfuhren
7 Abgrund|NN -0.3465
8 Abhängigkeit|NN -0.3653 Abhängigkeiten
9 Ablehnung|NN -0.5118 Ablehnungen
10 Ablenkung|NN -0.0435 Ablenkungen
11 Abnahme|NN -0.0048 Abnahmen
12 Abneigung|NN -0.0048 Abneigungen
13 Abnutzung|NN -0.0048
14 Abriss|NN -0.0048
... Abrisse,Abrissen,Abrisses,Abrisss
15 Abrutsch|NN -0.0048
... Abrutschen,Abrutsche,Abrutsches,Abrutschs
16 Abschaffung|NN -0.058 Abschaffungen
17 Abschreckung|NN -0.0048 Abschreckungen
18 Abschreibung|NN -0.3345 Abschreibungen
19 Abschuß|NN -0.0048
20 Abschwächung|NN -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 Koningrijk: 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, reproducability)

e.g., Schut, L. (2013). Verenigde Staten vs. Verenigd Koningrijk: 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 Sentistrenght algorithm

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

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: pattern

```
>>> from pattern.nl import sentiment
>>> sentiment('Wat een geweldige dag!')
(1.0, 1.0)
>>> sentiment('Super slecht dit!')
(-1.0, 1.0)
>>> sentiment('Best wel slecht dit!')
(-0.10000000000000003, 0.9375)
```

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

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.

Example 2: Vader

```
import nltk
nltk.download('vader_sentiment')
```

```
from nltk.sentiment import vader
senti=vader.SentimentIntensityAnalyzer()
senti.polarity_scores('This is a great day!')
senti.polarity_scores("I don't like this food")
senti.polarity_scores("I love her, but I hate him")
```

The `.polarity_scores` method returns a dictionary with four values. In our example, we get the following results:

```
        {'neu': 0.406, 'pos': 0.594, 'neg': 0.0, 'compound': 0.6588}
        {'neu': 0.587, 'pos': 0.0, 'neg': 0.413, 'compound': -0.2755}
        {'neu': 0.282, 'pos': 0.244, 'neg': 0.474, 'compound': -0.5346}
\end{lstlistingt}
```

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-occurance graph, irrelevant information will dominate the picture

Stopword removal: How

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

What do we get if we do:

```
print (testonuovo)
```

Can you explain the algorithm?

We get:

```
>>> print (testonuovo)
'He gives beer cigarette. '
```

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

regexp quizz

Which words would be matched?

1. `[Pp]ython`

regexp quizz

Which words would be matched?

1. `[Pp]ython`
2. `[A-Z]+`

regexp quizz

Which words would be matched?

1. `[Pp]ython`
2. `[A-Z]+`
3. `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 occurances 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 occurances 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 all occurances 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,
       f))]
8  for f in onlyfiles:
9     matchcount54=0
10    matchcount10=0
11    with open(path.join(mypath,f),mode="r",encoding="utf-8") as fi:
12       artikel=fi.readlines()
13       for line in artikel:
14          matches54 = re.findall('Israel.*(minister|politician.*|[Aa]
                uthorit)',line)
15          matches10 = re.findall('[Pp]alest',line)
16          matchcount54+=len(matches54)
17          matchcount10+=len(matches10)
18       filename_list.append(f)
19       matchcount54_list.append(matchcount54)
20       matchcount10_list.append(matchcount10)
21 output=zip(filename_list,matchcount10_list,matchcount54_list)
22 with open("overzichtstabel.csv", mode='w',encoding="utf-8") as fo:
23     writer = csv.writer(fo)
24     writer.writerows(output)
```

Example 2: Which number has this Lexis Nexis article?

```
                                All Rights Reserved

                                 2 of 200 DOCUMENTS

                                    De Telegraaf

                                 21 maart 2014 vrijdag

Brussel bereikt akkoord aanpak probleembanken;
ECB krijgt meer in melk te brokkelen

SECTION: Finance; Blz. 24
LENGTH: 660 woorden

BRUSSEL  Europa heeft gisteren op de valreep een akkoord bereikt
over een saneringsfonds voor banken. Daarmee staat de laatste
```

Example 2: Check the number of a lexis nexis article

```
                                All Rights Reserved

                                 2 of 200 DOCUMENTS

                                    De Telegraaf

                                 21 maart 2014 vrijdag

Brussel bereikt akkoord aanpak probleembanken;
ECB krijgt meer in melk te brokkelen

SECTION: Finance; Blz. 24
LENGTH: 660 woorden

BRUSSEL  Europa heeft gisteren op de valreep een akkoord bereikt
over een saneringsfonds voor banken. Daarmee staat de laatste
```

```
for line in tekst:
    matchObj=re.match(r" +([0-9]+) of ([0-9]+) DOCUMENTS",line)
    if matchObj:
        numberofarticle= int(matchObj.group(1))
        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 stopword 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 stopword removal - let's combine them!

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

Now, `print(frasenuevo)` returns:

```
run generous greet neighbor
```

Perfect!

Stemming and stopword 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:

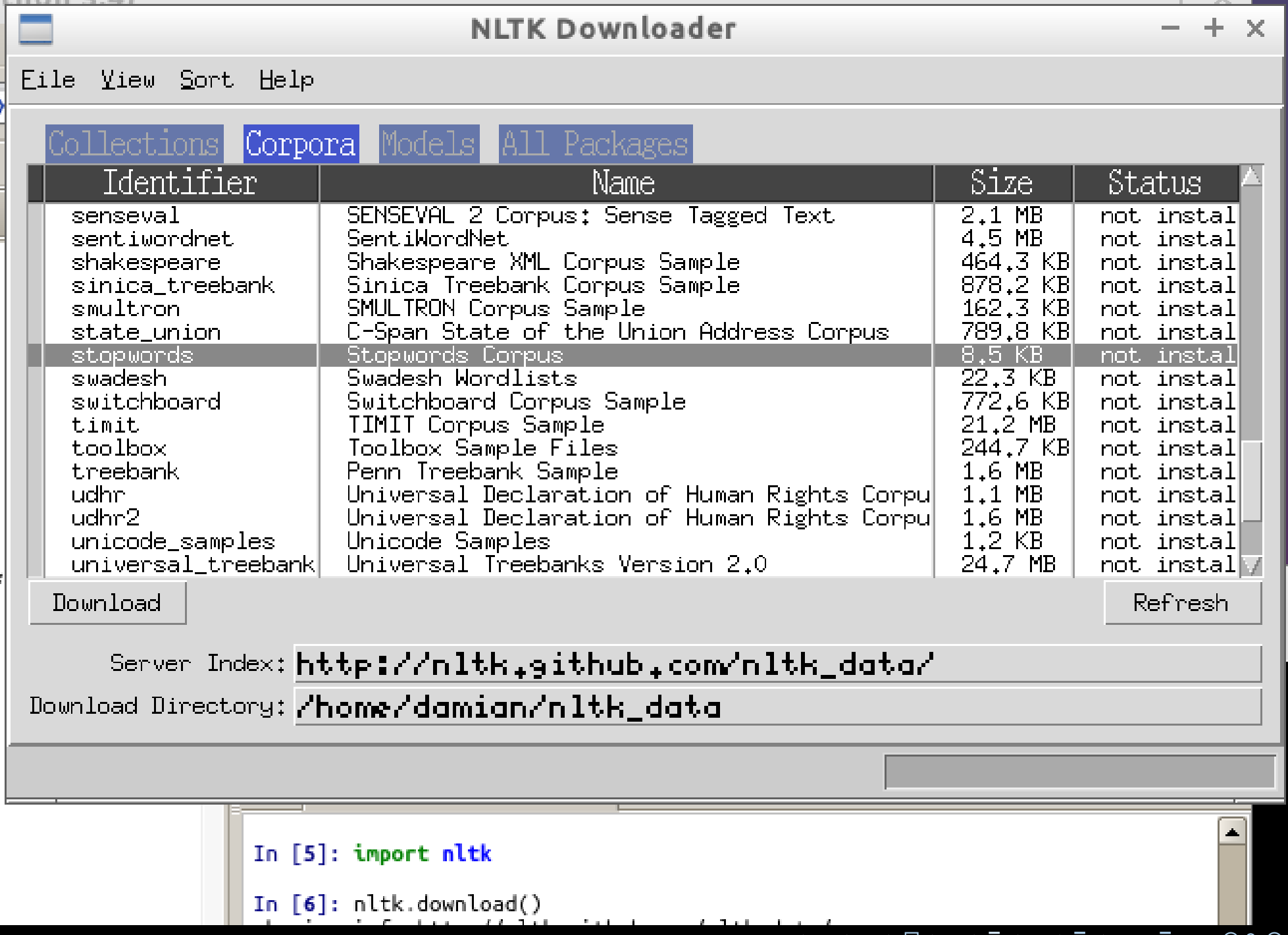
```
1  run generous greet neighbor
```

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.




```
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:

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

gives you the following:

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

Parsing a sentence

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

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

gives you the following:

```
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('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
EU speech dataset (Github)