Tag Clouds

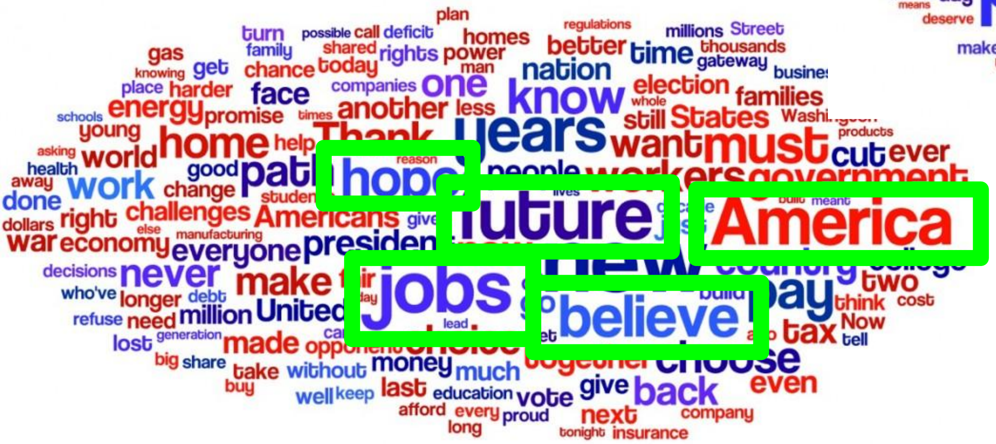
Advanced Programming - Dec. 10, 2015

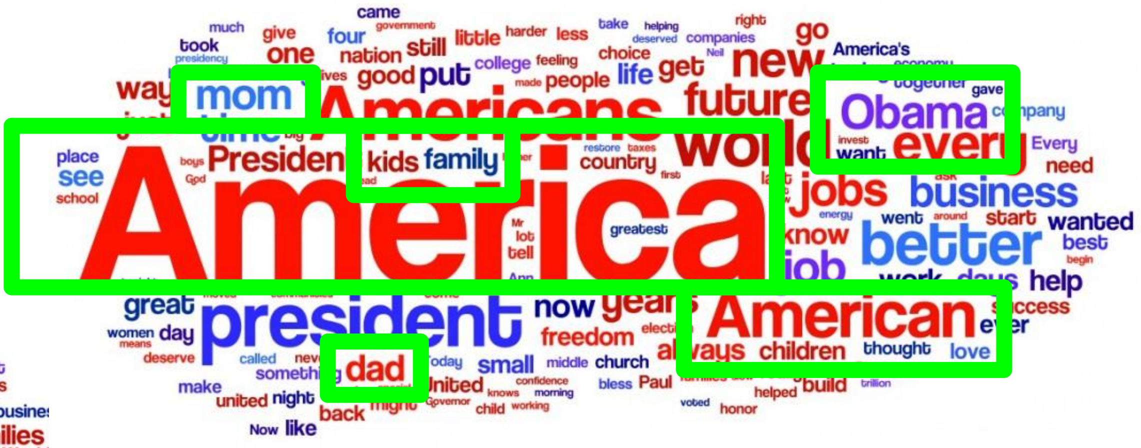
Marco Cornolti

Tag clouds

2012 Convention speech

Obama





Romney

Outline

1. Load a text
2. Tokenize terms
3. Normalization, stemming
4. Count frequencies
5. Generate the tag cloud

Installation of NLTK e pytagcloud

- From terminal:

```
sudo apt-get install python-pip   \
python-unidecode python-pygame    \
python-simplejson
sudo pip install nltk pytagcloud
```

- From python:

```
import nltk
nltk.download("all")
```

Prepare environment

```
mkdir ap_lab
cd ap_lab
wget http://tinyurl.com/lotr-book-txt -O lotr.txt
```

Loading a UTF-8 file

“Ah, sÃ¬, Ã¨ perchÃ© non puÃ² piÃ¹.”

Text files are always **encoded** with a codec. When reading a file, we must **decode** it with the same codec.

edit `tagcloud.py`:

```
import codecs
import re
def get_file_tokens(filename):
    tokens = []
    with codecs.open(filename, encoding="utf-8") as f:
        for line in f:
            tokens += re.split('\W+', line,  flags=re.UNICODE)
    return tokens
```

Libreries and main

codecs.open(...)
re.split(...)

tagcloud.get_file_tokens(...)

$$\begin{tikzcd}
\textbf{re, codecs, …} \\
\textbf{tagcloud} \arrow[u, "\textbf{import}"'] \\
\textbf{gen\_cloud.py} \arrow[u, "\textbf{import}"']
\end{tikzcd}$$

System libraries
(general purpose)

User libraries
(shared among tasks)

Executable
(task-specific)

Using a library

create **`gen_cloud.py`**:

```
from tagcloud import *

tokens = get_file_tokens("lotr.txt")
print tokens
print len(tokens)
```

import all functions defined in **`tagcloud.py`**

use a functiono defined in **`tagcloud.py`**

Filter words

Create a function that discards all words with less than three letters.

add to `tagcloud.py`:

```
def filter_words(words):
	return filter(lambda w: len(w)>=3, words)
```

test in `gen_cloud.py`:

```
filtered_t = filter_words(tokens)
print filtered_t
print len(filtered_t)
```

Filter words: stopword

Words that are so common they do not add semantics (the, as, of, if …)

add at the bedinning of **`tagcloud.py`**:

```
from nltk.corpus import stopwords
STOPWORDS = set(stopwords.words('english'))
```

test in **`gen_cloud.py`**:

```
print STOPWORDS
```

edit **filter_words** in **`tagcloud.py`**:

```
return filter(lambda w: len(w)>=3
and w not in STOPWORDS, words)
```

test in **`gen_cloud.py`**:

(note the number of words)

Normalize words

At the semantic level, there is no difference between:

- naïve, Naive, NAIVE

Strategy:

- lowercase
- normalize accented characters

Normalize words (2)

add to `tagcloud.py`:

```
from unidecode import unidecode
def normalize_words(words):
    return map(lambda w: unidecode(w.lower()), words)
```

`“NAïve”` $\xrightarrow{\text{lower()}}$ `“naïve”` $\xrightarrow{\text{unidecode()}}$ `“naive”`

test in `gen_cloud.py`:

```
filtered_t = filter_words(normalize_words(tokens))
print filtered_t
print len(filtered_t)
```

Analysis of words

- Frequency of a word (e.g. “day”)
- How many words (with duplicates)?
- How many distinct words?
- What are the 10 most common words?
- Most frequent word?
- How many words appear only once?

`Counter` gives you the answers!

Count with `Counter`

test in shell python:

```
>>> a = Counter(["aaa","bbb","ccc","bbb", "bbb", "aaa"])
>>> a
Counter({'bbb': 3, 'aaa': 2, 'ccc': 1})
>>> a["aaa"]
2
>>> a["zzz"]
0
>>> a.most_common(2)
[('bbb', 3), ('aaa', 2)]
>>> a.values()
[2, 3, 1]
>>> sum(a.values())
6
>>> list(a)
['aaa', 'bbb', 'ccc']
>>> a.items()
[('aaa', 2), ('bbb', 3), ('ccc', 1)]
```

Use of `Counter`

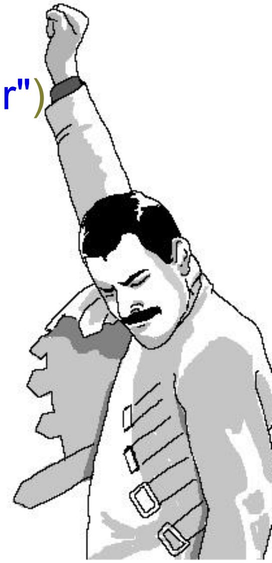
test in **`holy_cloud.py`**:

```
from collections import Counter
c = Counter(filtered_t)                 #create word counter
print c["day"]                          #occurrences of "day"
print len(c)                            #distinct words
print sum(c.values())                   #total words
print c.most_common(10)                 #10 most frequent words (and their freqency)
print c.most_common(10)[0][0]           #most frequent word
print len(filter(lambda p: p[1]==1, c.items())) # words used only once
```

Generation of a tag cloud

add to **`tagcloud.py`**:

```
from pytagcloud import create_tag_image, make_tags
def generate_tag_cloud(freq, image_filename):
        tags = make_tags(freq, maxsize=80)
        create_tag_image(tags, image_filename, size=(1200, 900), fontname="Lobster")
```



test in **`gen_cloud.py`**:

```
generate_tag_cloud(c.most_common(100), "tag_cloud.png")
```

Stemming

Aggregate words according to its stem (losing a little bit of precision), we remove morphological suffixes:

- `“believe”`, `“believes”`, `“believed”` **->** `“believ”`
- `“company”`, `“companies”` **->** `“compan”`
- `“amsterdam”` **->** `”amsterdam”`

Stemming (2)

add to **`tagcloud.py`**:

```
from nltk.stem.snowball import EnglishStemmer
def stem_words(words):
    s = EnglishStemmer()
    return map(s.stem, words)
```

test in **`gen_cloud.py`**:

```
tokens = get_file_tokens("lotr.txt")
filtered_t = filter_words(normalize_words(tokens))
stemmed = stem_words(filtered_t)

c = Counter(stemmed)
generate_tag_cloud(c.most_common(100), "tagcloud.png")
```

Stemming (3)

We lose the form of words! Let’s keep track of words in their original form. We need to preserve this:

$$\texttt{“believ”} \longrightarrow \texttt{“believed”}\mathbf{x3},\ \texttt{“believes”}\mathbf{x1}$$

$$\texttt{“day”} \longrightarrow \underline{\texttt{“days”}\mathbf{x10},\ \texttt{“day”}\mathbf{x4}}$$

this look very much like a counter...

Stemming (4)

add to `tagcloud.py`:

```
from collections import Counter

def get_stem_mapping(words):
    s = EnglishStemmer()
    mapping = {}
    for w in words:
        stemmed_w = s.stem(w)
        if stemmed_w not in mapping:
            mapping[stemmed_w] = Counter()
        mapping[stemmed_w][w] += 1
    return mapping

def destem_words(stems, stem_mapping):
    return map(lambda s: stem_mapping[s].
most_common(1)[0][0], stems)
```

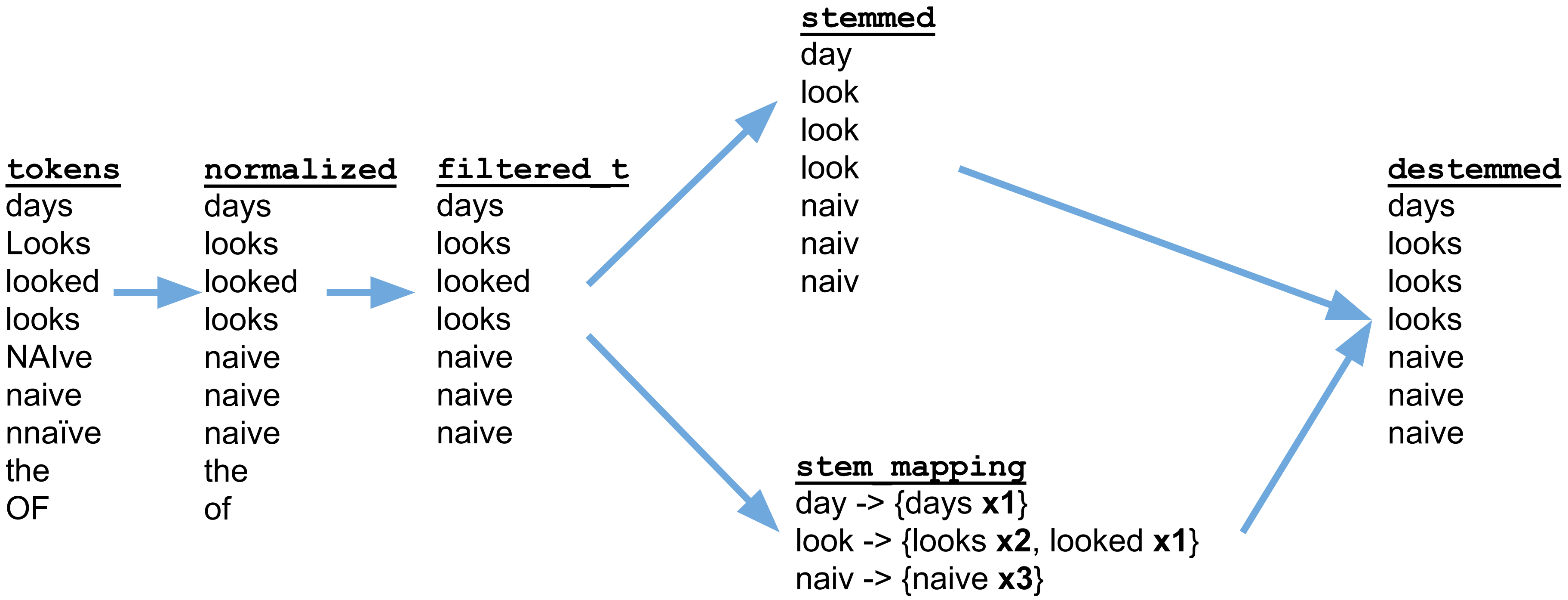
final version of `tag_cloud.py`:

```
from tagcloud import *
from collections import Counter

tokens = get_file_tokens("lotr.txt")
normalized = normalize_words(tokens)
filtered_t = filter_words(normalized)
stemmed = stem_words(filtered_t)
stem_mapping = get_stem_mapping(filtered_t)
destemmed = destem_words(stemmed, stem_mapping)

generate_tag_cloud(Counter(filtered_t).most_common(100),
"lotr_filtered.png")
generate_tag_cloud(Counter(stemmed).most_common(100),
"lotr_stemmed.png")
generate_tag_cloud(Counter(destemmed).most_common(100),
"lotr_destemmed.png")
```

Data flow



tokens
days
Looks
looked
looks
NAIve
naive
nnaïve
the
OF

normalized
days
looks
looked
looks
naive
naive
naive
the
of

filtered_t
days
looks
looked
looks
naive
naive
naive

stemmed
day
look
look
look
naiv
naiv
naiv

stem_mapping
day -> {days **x1**}
look -> {looks **x2**, looked **x1**}
naiv -> {naive **x3**}

destemmed
days
looks
looks
looks
naive
naive
naive

Redo this at home

http://goo.gl/Jm0Ol4

cornolti@di.unipi.it

- Try with other books/text sources
- Compare clouds from different sources