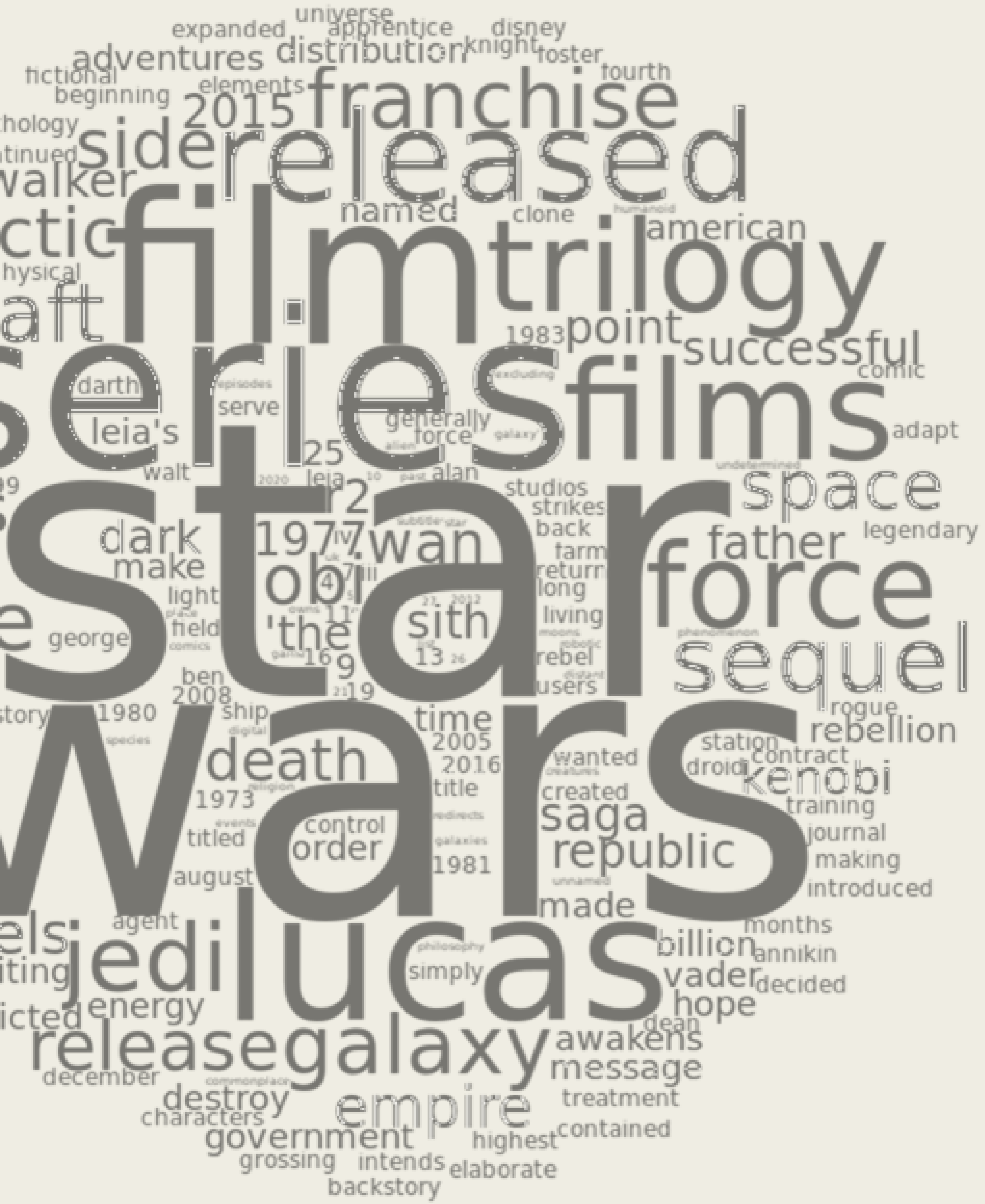


INFORMATION RETRIEVAL

NLP + WORDCLOUD

lorenzo.bellomo@di.unipi.it - Lorenzo Bellomo

Paolo Ferragina

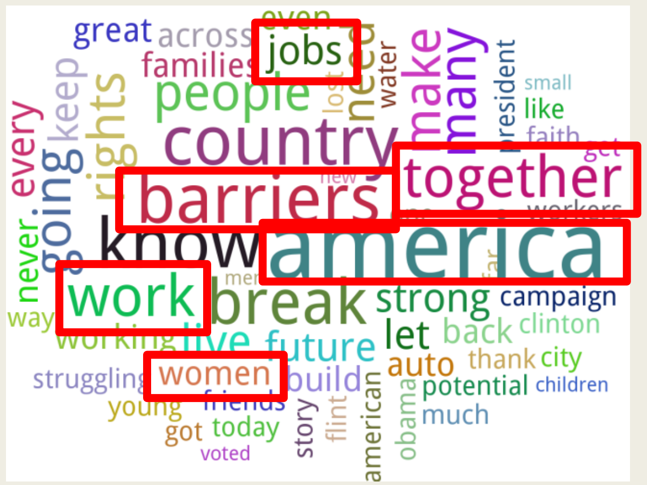
Goal – Make some wordclouds

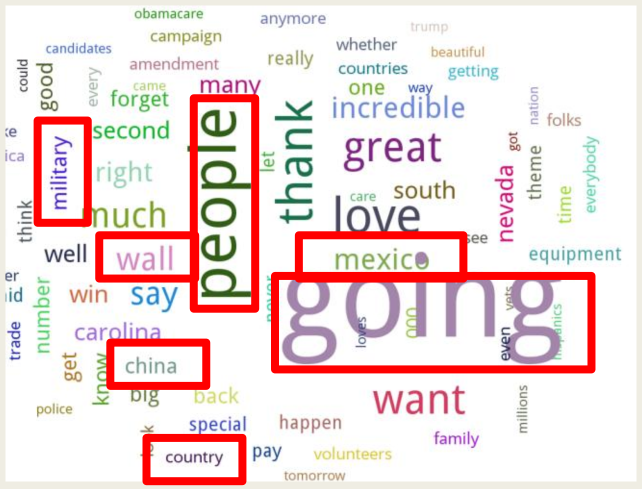
2016 presidential primaries

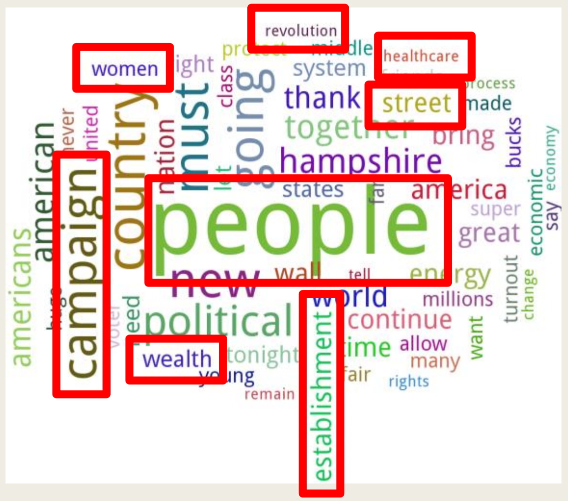
Clinton

Trump

Sanders







Outline

- Load some text
- Separate words or phrases (tokenization)
- Normalization, stemming
- Count word frequency
- Generate the word-cloud

Required packages

- unidecode: Text often come in different textual codifications, it allows us to correctly pick the correct codec
- json: we will open some json files
- nltk: standard package for performing some base-nlp task
- wordcloud: we will use it to create the images

In python we also need to do

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

Loading a UFT-8 file

“Ah, sÃ¬, Ã¨ perchÃ© non puÃ² piÃ¹.”

Text files have an encoding, so we need the correct codec to open them

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

Basic word analysis

- Frequency of “abramo”
- How many words are there?
- Which are the ten most frequent?
- What is the most frequent?
- How many words appear only once?

Everything in **Counter**!

Counter

```
from collections import Counter
>>> 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]  # dict_values([2, 3, 1])
>>> sum(a.values())
6
>>> list(a)
['aaa', 'bbb', 'ccc']
>>> a.items()
[('aaa', 2), ('bbb', 3), ('ccc', 1)]  # dict_items([('aaa', 2),
('bbb', 3), ('ccc', 1)
```

Create the word cloud

```
from wordcloud import WordCloud

def generate_tag_cloud(freq, image_filename):
	wc = 
WordCloud(background_color="black").generate_from_frequencies(freq)
	image = wc.to_image()
	image.save(image_filename)


generate_tag_cloud(dict(c.most_common(100)), "corano.png")
```

Short words and stopwords

Short words are effectively useless for the tag cloud

```
def filter_words(words):
        return [w for w in words if len(w) >= 3]
```

This is also true for stopwords, which have add no real semantic meaning

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

```
[w for w in words if len(w)>=3 and w not in STOPWORDS]
```

Problem - Normalization

Semantically, there is no difference between:

peRChè, perché, PERCHÉ

Strategy:

- Reduce to lowercase
- Normalize characters with accents

```
from unidecode import unidecode
def normalize_words(words):
    return [unidecode(w.lower()) for w in words]
```

$$\text{“pERChéèè”} \xrightarrow{\text{lower()}} \text{“perchéèè”} \xrightarrow{\text{unidecode()}} \text{“percheee”}$$

On our texts

```
from collections import Counter

c = Counter(filtered_t)                              # create the word counter
print(c["abramo"])                                   # occurrences of "abramo"
print(len(c))                                        # unique words
print(sum(c.values()))                               # total words
print(c.most_common(10))                             # 10 most frequent words
print(c.most_common(10)[0][0])                       # most common word
print(len([p[0] for p in c.items() if p[1]==1]))    # words that appear only once
```

Stemming

To aggregate words on the basis of their root (losing some precision), we can use stemming

- “credere”, “credo”, “credete” -> “cred”
- “credenti”, “credente” ->“credent”
- “amsterdam” -> ”amsterdam”

Stemming code

```
from nltk.stem.snowball import ItalianStemmer
def stem_words(words):
        s = ItalianStemmer()
        return [s.stem(w) for w in words]
```

But this makes our word cloud ugly, we need to keep track of the original words

$$\text{“cred”} \longrightarrow \text{“credere”}\mathbf{x3}\text{, “credo”}\mathbf{x1}$$

$$\text{“credent”} \longrightarrow \text{“credenti”}\mathbf{x10}\text{, “credente”}\mathbf{x4}$$

De-Stemmer

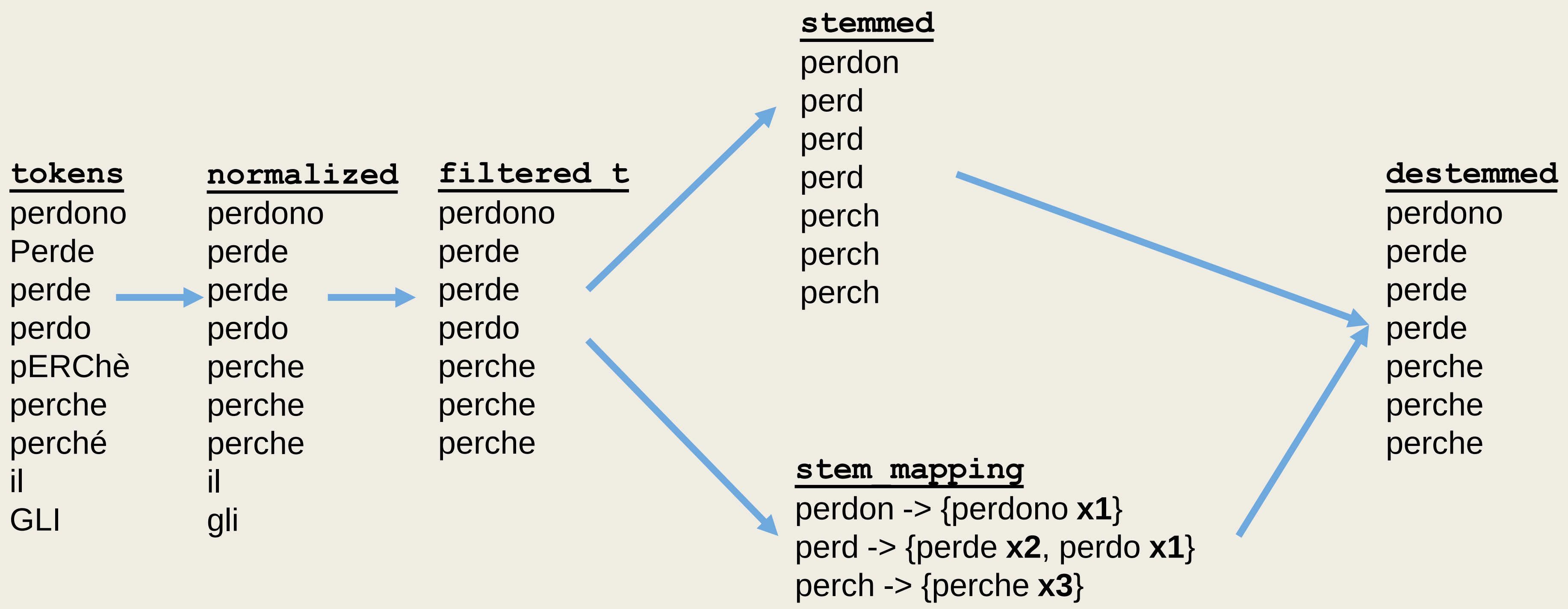
```
from collections import Counter

def get_stem_mapping(words):
    s = ItalianStemmer()
    mapping = {}
    for w in words:
        stemmed_w = s.stem(w)
        if stemmed_w not in mapping:
            mapping[stemmed_w] = Counter()
        mapping[stemmed_w].update([w])

    return mapping

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

Full-Pipeline



tokens
perdono
Perde
perde
perdo
pERChè
perche
perché
il
GLI

normalized
perdono
perde
perde
perdo
perche
perche
perche
il
gli

filtered_t
perdono
perde
perde
perdo
perche
perche
perche

stemmed
perdon
perd
perd
perd
perch
perch
perch

stem_mapping
perdon -> {perdono **x1**}
perd -> {perde **x2**, perdo **x1**}
perch -> {perche **x3**}

destemmed
perdono
perde
perde
perde
perche
perche
perche