

Text Mining and Search

Lab 1.1 and 1.2

Preprocessing and Doc Representation

Luca Herranz-Celotti

University of Milano-Bicocca, 2024

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Introduction

Working With Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech Tagging

Entity Recognition

Document Representation

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Introduction

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Introduction and Purpose

Text Mining and
Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Luca
Herranz-Celotti

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Introduction and Purpose

- ▶ Provide overview of NLP tools;

Introduction and Purpose

- ▶ Provide overview of NLP tools;
- ▶ Show and run some example codes;

Introduction and Purpose

- ▶ Provide overview of NLP tools;
- ▶ Show and run some example codes;
- ▶ Provide references to some tools and resources.

NLP python libraries:

NLP python libraries:

► NLTK

NLP python libraries:

- ▶ NLTK
- ▶ SpaCy

NLP python libraries:

- ▶ NLTK
- ▶ SpaCy
- ▶ TextBlob

NLP python libraries:

- ▶ NLTK
- ▶ SpaCy
- ▶ TextBlob
- ▶ Gensim

NLP python libraries:

- ▶ NLTK
- ▶ SpaCy
- ▶ TextBlob
- ▶ Gensim
- ▶ Stanford NLP

Instructions for Google Colab

Text Mining and
Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Luca
Herranz-Celotti

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Instructions for Google Colab

► Go to <https://linktr.ee/lucarrrt>

Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Luca
Herranz-Celotti

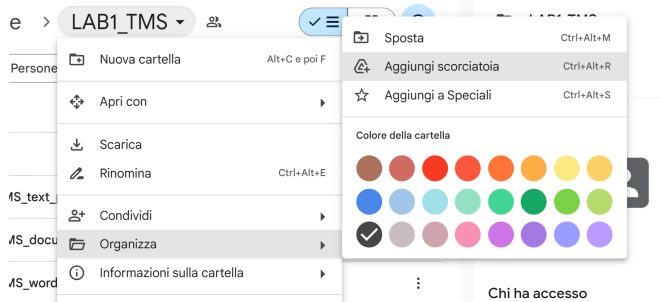
Introduction

Instructions for Google Colab

- ▶ Go to <https://linktr.ee/lucarrrt>
- ▶ Select the 'Tutorial 1 - Text Mining and Search 2024' button

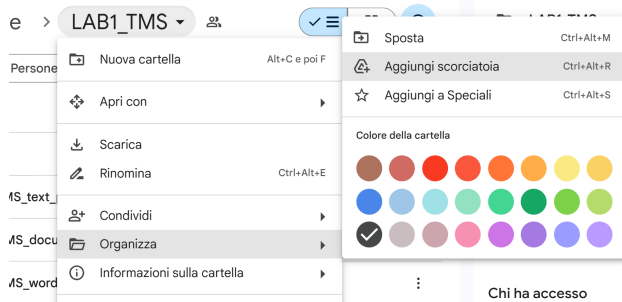
Instructions for Google Colab

- ▶ Go to <https://linktr.ee/lucarrrt>
- ▶ Select the 'Tutorial 1 - Text Mining and Search 2024' button
- ▶ Make a shortcut of the folder to your drive



Instructions for Google Colab

- ▶ Go to <https://linktr.ee/lucarrtt>
- ▶ Select the 'Tutorial 1 - Text Mining and Search 2024' button
- ▶ Make a shortcut of the folder to your drive



- ▶ Open the Text Pre-Processing Notebook with Colab and save it on your drive.

Working With Texts

Different data formats require different loading techniques:

CSV

```
import pandas as pd
data = pd.read_csv(file_path, sep=",")
```

JSON

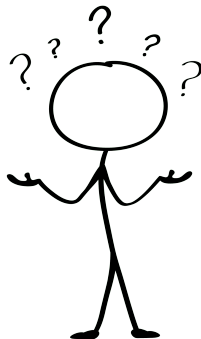
```
import json
with open(file_path, "r") as f:
    data = json.load(f)
```

JSONL

```
import json
data = []
with open(file_path) as f:
    for line in f:
        data.append(json.loads(line))
```

Text Preprocessing

Why do we need text processing?



Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Text Preprocessing

A simple black and white line drawing of a stick figure. The figure is standing on its right leg, with its left leg bent and its left arm extended forward, holding a glowing lightbulb. The lightbulb has several short lines radiating from it, indicating it is lit. The figure's head is a large circle, and its body is a simple line.

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Text Preprocessing

A simple black and white line drawing of a stick figure. The figure is standing on two legs, with its right arm extended forward and slightly upward, holding a glowing lightbulb. The lightbulb has several short lines radiating from it, indicating it is lit. The figure's head is a large circle, and its torso and limbs are simple lines.

- ▶ Better text analysis
- ▶ Faster computational processing

Text preprocessing Overview

There are multiple preprocessing steps to concentrate on: numbers, punctuation, symbols, whitespaces, stop words, URLs, emojis etc. in the text.

Text preprocessing Overview

There are multiple preprocessing steps to concentrate on: numbers, punctuation, symbols, whitespaces, stop words, URLs, emojis etc. in the text.



Whitespaces

This sentence has an abnormal
amount of whitespaces.

This sentence has an abnormal
amount of whitespaces.

To cope with these kind of issues we can:

- ▶ Define a custom function

This sentence has an abnormal
amount of whitespaces.

To cope with these kind of issues we can:

- ▶ Define a custom function
- ▶ Use regex → `re.sub("\s+", " ", text)`

This sentence has an abnormal
amount of whitespaces.

To cope with these kind of issues we can:

- ▶ Define a custom function
- ▶ Use regex → `re.sub("\s+", " ", text)`
- ▶ Use string methods → `" ".join(text.split())`

HERE IS AN EXAMPLE.

A common strategy is to reduce all letters to lowercase.

```
text = "HERE IS AN EXAMPLE"  
text = text.lower() # lowercase  
text = text.upper() # uppercase
```


Numbers, symbols, punctuation

Everything is ok!11 âĈňâĈň

Also this case is solved using python string methods and regex.

Very Simple Exercise

Try to think about a regex to remove all the numbers from a string, and do the same using only string methods.

Emojis

We use essentially two python libraries:

- ▶ demoji
- ▶ emoji

Spelling Mistakes and Repeated Characters

For repeaaaaateddddd !!! characters we use:

Spelling Mistakes and Repeated Characters

For repeaaaaateddddd !!! characters we use:
REGEX!

Spelling Mistakes and Repeated Characters

For repeaaaaateddddd !!! characters we use:
REGEX!

Another Exercise

Try to think of a way in regex that will find all the repeated occurrences of a character in a word and replace them with two occurrences if the character is alphabetical and one occurrence if the character is a punctuation.

You can use multiple regexes!

Spelling Mistakes and Repeated Characters

regex might not be good enough, indeed repeaaaaateddddd !!! → repeaatedd !!, which still contains some spelling mistakes.

To find spelling mistake we can rely on [TextBlob](#)

Spelling Mistakes and Repeated Characters

regex might not be good enough, indeed repeaaaaateddddd !!! → repeaatedd !!, which still contains some spelling mistakes.

To find spelling mistake we can rely on **TextBlob**

```
from textblob import TextBlob  
  
blob = TextBlob(text)  
corrected_blob = blob.correct()
```

Note: Repeated character removal is still essential as textblob correction might fail if there are too many repeated characters.

URLs

Many web documents contain ULRs (hyperlinks) and HTML Tags.
Sometimes they are easily identifiable with regex:

```
r"httpS+" or r"www.\w*.com".
```


Many web documents contain ULRs (hyperlinks) and HTML Tags.
Sometimes they are easily identifiable with regex:

`r"httpS+"` or `r"www.\w*.com"`.

The two regex indicated above have issues for real world applications, can you identify them?

Many web documents contain ULRs (hyperlinks) and HTML Tags.
Sometimes they are easily identifiable with regex:

`r"httpS+"` or `r"www.\w*.com"`.

The two regex indicated above have issues for real world applications, can you identify them?

There is no one rule fits all regex, so learn to create your own regex based on the data you are working on.

For HTML tags, we can still implement a regex but there are better options:

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(text, "html.parser")
text = soup.get_text(separator=" ")
```

Tokenization

Tokenization is the process of splitting text into smaller pieces called tokens.
Different approaches:

Tokenization is the process of splitting text into smaller pieces called tokens.
Different approaches:

- ▶ uni-gram;

Tokenization is the process of splitting text into smaller pieces called tokens.
Different approaches:

- ▶ uni-gram;
- ▶ bi-gram;

Tokenization is the process of splitting text into smaller pieces called tokens.
Different approaches:

- ▶ uni-gram;
- ▶ bi-gram;
- ▶ based on punctuation;

Tokenization is the process of splitting text into smaller pieces called tokens.
Different approaches:

- ▶ uni-gram;
- ▶ bi-gram;
- ▶ based on punctuation;
- ▶ regex based etc.

Some issues, how do you tokenize the following

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Tokenization

Some issues, how do you tokenize the following
O'Neill ?→? o neill, o' neill, oneill, o'neill

Tokenization

Some issues, how do you tokenize the following

O'Neill	?→?	o neill, o' neill, oneill, o'neill
aren't	?→?	aren't, arent, are n't, aren t

Tokenization

Some issues, how do you tokenize the following

O'Neill	?→?	o neill, o' neill, oneill, o'neill
aren't	?→?	aren't, arent, are n't, aren t
New York	?→?	one or two tokens?

Tokenization

Some issues, how do you tokenize the following

O'Neill	?→?	o neill, o' neill, oneill, o'neill
aren't	?→?	aren't, arent, are n't, aren t
New York	?→?	one or two tokens?
Ph.D.	?→?	Ph D, PhD

Tokenization

Tokenization

Think of how you could implement a tokenizer, using regex or string methods.

Think of how you could implement a tokenizer, using regex or string methods.

In practice we often use [NLTK Tokenizers](#), for example when working with tweets you can use TweetTokenizer from NLTK:

```
from nltk.tokenize import TweetTokenizer

tknzs = TweetTokenizer()
tokenized_tweet = tknzs.tokenize(tweet)
```

Normalization

Stop word removal

We do not always want words like: a, the, in; these words are called stop words.

Stop word removal

We do not always want words like: a, the, in; these words are called stop words.

Well actually it depends on the context and the model being used!

Stemming

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Stemming

- ▶ Stemming reduces a word to their stem (or root).

- ▶ Stemming reduces a word to their stem (or root).
- ▶ There are many existing stemming algorithms:
 - ▶ Porter stemming;
 - ▶ Krovetz stemming (a hybrid method actually);

Lemmatization

- ▶ Opposed to stemming, lemmatization does not simply «chop off» the words.

- ▶ Opposed to stemming, lemmatization does not simply «chop off» the words.
- ▶ It uses dictionaries to find the ground form of the word, the lemma.

- ▶ Opposed to stemming, lemmatization does not simply «chop off» the words.
- ▶ It uses dictionaries to find the ground form of the word, the lemma.
- ▶ An example: WordNet Lemmatizater.

Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Normalization

- ▶ Lemmatization is certainly the more correct approach compared to stemming;
- ▶ Stemming is usually easier to implement and run faster;
- ▶ It depends on your task!!
- ▶ There are some hybrid methods as well, we mentioned Krovetz before!

Part-Of-Speech Tagging

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

Document
Representation

Part-Of-Speech Tagging

Text Mining and
Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Luca
Herranz-Celotti

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

**Part-Of-Speech
Tagging**

Entity Recognition

Document
Representation

Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Part-Of-Speech Tagging

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺ ↻

► Defines the grammatical function of words within a sentence:

- Noun singular (NN)
- Noun Plural (NNS)
- Verb (VB)
- Verb past tense (VBD)
- Adjective (JJ)
- and more ...

```
from textblob import TextBlob
```

```
blob = TextBlob(text)  
tags_blob = blob.tags()
```

Entity Recognition

- ▶ Define the semantical function of words in the text. Some examples:
 - ▶ Organizations
 - ▶ Person
 - ▶ Locations
 - ▶ Time
 - ▶ quantities

Document Representation

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

**Document
Representation**

Document Representation

Text Mining and
Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Luca
Herranz-Celotti

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

**Document
Representation**

Document Representation

▶ Binary Representation

Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Document Representation

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ≡ ↺ 🔍 ↻

Text Mining and Search
Lab 1.1 and 1.2
Preprocessing
and Doc
Representation

Document Representation

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

Document Representation

- ▶ Binary Representation
- ▶ Bag-Of-Words (BOW) Representation
 - ▶ CountVectorizer
 - ▶ Tf-Idf
- ▶ Dense Vector Representation (more advanced)

Python library: [scikit-learn](#)

Binary Representation

We use simply 0 and 1 to indicate if the term is in the document.

Each term is represented by its frequency:

```
from sklearn.feature_extraction.text import CountVectorizer  
cv = CountVectorizer(min_df=0., max_df=1.)  
cv_matrix = cv.fit_transform(corpus)
```

For each term compute TF and IDF:

For each term compute TF and IDF:

$TF(w,d)$ is simply the frequency of the term (w) appearing in the document (d).

For each term compute TF and IDF:

$TF(w,d)$ is simply the frequency of the term (w) appearing in the document (d).

$IDF(w)$ is a sort of rarity of a word in the corpora, in general computed as:

For each term compute TF and IDF:

TF(w,d) is simply the frequency of the term (w) appearing in the document (d).

IDF(w) is a sort of rarity of a word in the corpora, in general computed as:

$$IDF(w) = \log \frac{n}{df(w)}$$

where n is the number of documents and df(w) counts the number of documents containing the term w.

After representing a document as a vector we can use vector properties.

Introduction

Working With
Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity Recognition

**Document
Representation**

After representing a document as a vector we can use vector properties.

In particular we use similarity or distance functions to measure how close two points (so two documents) are in the vector space!