

Information Retrieval Lab 1

Pranav Kasela

University of Milano-Bicocca

09 November 2022

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

Working With
Texts

Text
Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity
Recognition

Document
Representation

Introduction

Introduction and Purpose

Introduction

Working With Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech Tagging

Entity Recognition

Document Representation

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

Introduction

Working With Texts

Text Preprocessing

Tokenization

Normalization

Part-Of-Speech Tagging

Entity Recognition

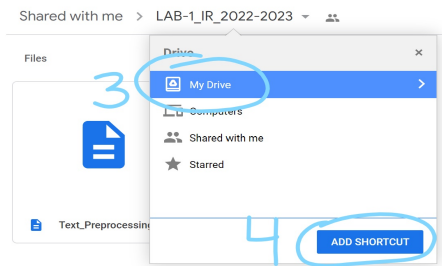
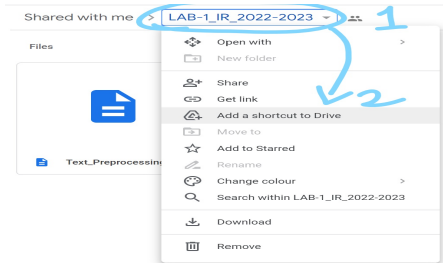
Document Representation

NLP python libraries:

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

Instructions for Google Colab

- ▶ Go to <https://linktr.ee/pkasela>
- ▶ For this lesson select LAB 1 button
- ▶ You will see two files, make a shortcut of the folder to yours



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

Working With Texts

Data Loading

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))
```

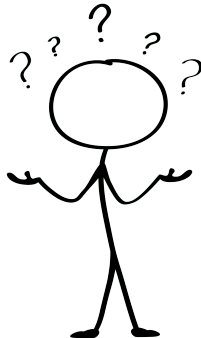

Keep in mind the various data encoding techniques:

- ▶ utf-8, utf-16, utf-32
- ▶ ascii
- ▶ You also have emojis
- ▶ And many more: [python docs](#)

Text Preprocessing

Text preprocessing Overview

Why do we need text processing?




Information Retrieval Lab 1

Text Preprocessing

Tokenization

Normalization

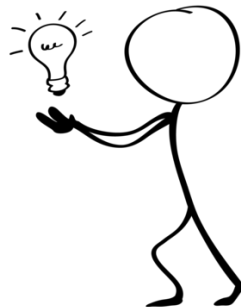
Part-Of-Speech Tagging



- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

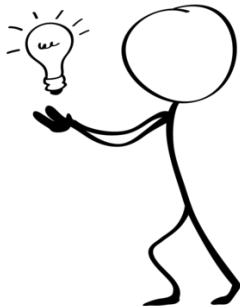
Why do we need text processing?

- ▶ Better text analysis
- ▶ Faster computational processing



Why do we need text processing?

- ▶ Better text analysis
- ▶ Faster computational processing
- ▶ Better model performances



Text preprocessing Overview

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

Text preprocessing Overview

There are multiple preprocessing steps that 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 €€

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 repeaaaateddddd !!! characters we use:

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 a word and one occurrence if the character is a punctuation.

You can use multiple regexes!

Spelling Mistakes and Repeated Characters

The regex we thought is not good enough, indeed repeaaaateddddd !!! → repeaatedd !, which still contains some spelling mistakes.

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

Spelling Mistakes and Repeated Characters

The regex we thought is not good enough, indeed repeaaaateddddd !!! → repeatedd !, 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.

A lot of documents from the web contains ULRs (hyperlinks) and HTML Tags.

URLs are easily identifiable using regexes, for example:

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

A lot of documents from the web contains ULRs (hyperlinks) and HTML Tags.

URLs are easily identifiable using regexes, for example:

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

The two regex indicated above have a lot of issues for real world application, can you identify them?

There is no one rule fits all regex, so 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=" ")
```

Introduction

Working With
Texts

Text
Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity
Recognition

Document
Representation

Tokenization

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:

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

For exercise try to think about how you could implement any tokenizer indicate before, using regex or string methods.

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

```
from nltk.tokenize import TweetTokenizer

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

Introduction

Working With
Texts

Text
Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity
Recognition

Document
Representation

Normalization

Stop word removal

We do not 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 reduces a word to their stem (or root).
- ▶ There are many existing stemming algorithms:
 - ▶ Porter stemming;
 - ▶ Krovetz stemming (a hybrid method actually);

- ▶ Opposed to stemming, lemmatization does not simply «chop off» the words.
- ▶ It uses lexical analysis and context knowledge to reduce a word to its lemma.
- ▶ An example: WordNet Lemmatizater.

Which is better?

- ▶ 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!

Introduction

Working With
Texts

Text
Preprocessing

Tokenization

Normalization

**Part-Of-Speech
Tagging**

Entity
Recognition

Document
Representation

Part-Of-Speech Tagging

► Mark the words within a text:

- Noun singular (NN)
- Noun Plural (NNS)
- Verb (VB)
- Verb past tense (VBD)
- Adjective (JJ)
- and many more ...
- In python we use TextBlob for basic POS tagging

```
from textblob import TextBlob  
  
blob = TextBlob(text)  
tags_blob = blob.tags()
```

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

Entity Recognition

- ▶ Named nouns that can be for example:
 - ▶ Organizations
 - ▶ Person
 - ▶ Locations
 - ▶ Time
 - ▶ quantities

Introduction

Working With
Texts

Text
Preprocessing

Tokenization

Normalization

Part-Of-Speech
Tagging

Entity
Recognition

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:

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.

In scikit-learn implementation they use smoothing by default:

$$IDF(w) = \log \frac{n + 1}{df(w + 1)} + 1$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(corpus)
```

After representing a document as a point in a vector space we can use vector space properties.

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