Text Mining and Search Lab 1

Joseph Muddle

UniMiB

Outline

Introduction

- Who am I?
- Who are you?
- What are these labs about?

Purpose of these labs

You're already familiar with some of the **theory** of text mining and search. These labs focus on the **practice**. As such, we will be doing the following in these three labs:

- Giving an overview of NLP tools

Purpose of these labs

You're already familiar with some of the **theory** of text mining and search. These labs focus on the **practice**. As such, we will be doing the following in these three labs:

- Giving an overview of NLP tools
- Working through code examples

Purpose of these labs

You're already familiar with some of the **theory** of text mining and search. These labs focus on the **practice**. As such, we will be doing the following in these three labs:

- Giving an overview of NLP tools
- Working through code examples
- Giving references to resources

Motivation: Text processing

Why do we need automated (i.e not done by a human) text processing?

Motivation: Text processing

Why do we need automated (i.e not done by a human) text processing?

- Faster analysis
- More reliable and repeatable performance
- Better *downstream* model performance

Tools and resources

Can you think of any tools in the Python ecosystem that make NLP convenient and easy?

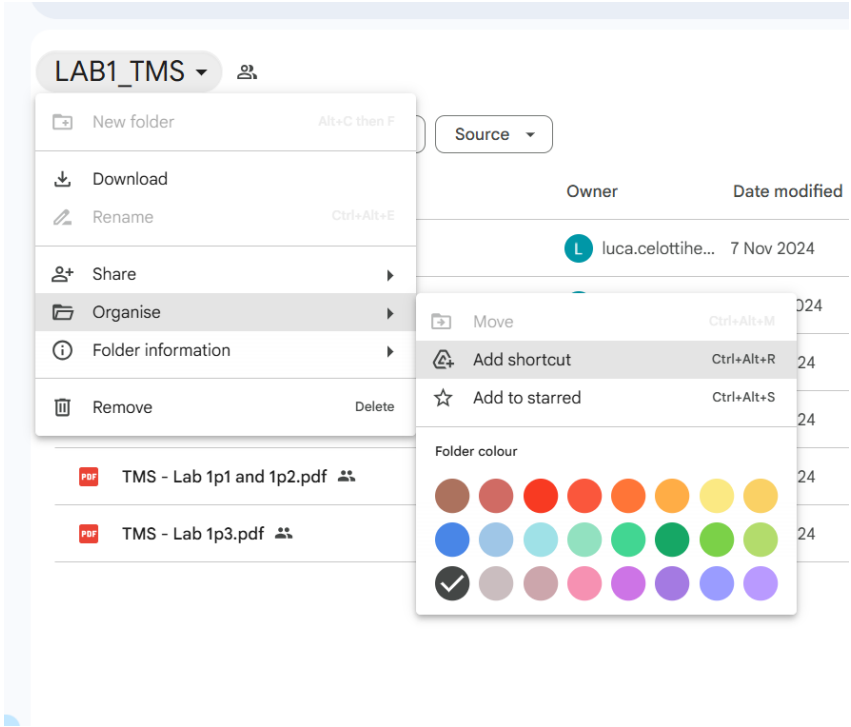
Tools and resources

Can you think of any tools in the Python ecosystem that make NLP convenient and easy?

- NLTK
- SpaCy
- TextBlob
- Stanford NLP
- Gensim

Tools and Resources

- Go to linktr.ee/joemuddle
- Select Tutorial 1
- Make a shortcut to your drive
- Open "Text Pre-Processing" with colab and save to your drive



Text Preprocessing Overview

There are many tasks which fall under the heading of "text pre-processing". Text pre-processing is mainly about getting the *formatting* of the text to be relatively consistent, so that only the "important" parts of the text are used in our downstream tasks. What are the "unimportant" parts of text?

Text Preprocessing Overview

There are many tasks which fall under the heading of "text pre-processing". Text pre-processing is mainly about getting the *formatting* of the text to be relatively consistent, so that only the "important" parts of the text are used in our downstream tasks. What are the "unimportant" parts of text?

- Excessive whitespace (tabs, spaces, newlines)
- Upper/lower case distinction
- Emojis
- URLs
- Spelling errors/Repeated letters (not always unimportant!)
- "Stop words" ("a", "the", etc. Again, not always unimportant!)

Text Preprocessing Overview

There are many tasks which fall under the heading of "text pre-processing". Text pre-processing is mainly about getting the *formatting* of the text to be relatively consistent, so that only the "important" parts of the text are used in our downstream tasks. What are the "unimportant" parts of text?

- Excessive whitespace (tabs, spaces, newlines)
- Upper/lower case distinction
- Emojis
- URLs
- Spelling errors/Repeated letters
- "Stop words"

There is no "one size fits all" approach to text preprocessing!
Always remember the downstream task!

Regular Expressions

RegEx (regular expressions) are a powerful tool for text formatting, and one that we'll be using a lot. We'll be using the re library in python, and also python's powerful string methods.

```
This    has    excessive    whitespace
```

Regular Expressions

RegEx (regular expressions) are a powerful tool for text formatting, and one that we'll be using a lot. We'll be using the re library in python, and also python's powerful string methods.

```
This    has    excessive    whitespace
```

- We can use RegEx $\rightarrow$ `re.sub("\s+","", text)`
- We can use a string method $\rightarrow$ `" ".join(text.split())`

A great resource for learning RegEx is **regexone.com**

Case Folding

A lot of the time in text pre-processing we want to make everything upper case or everything lower case. With python, it's really easy to just do this with string methods.

```
text = "This is an Example"
text = text.lower()
text = text.upper()
```

Numbers and Punctuation

RegEx is best for removing numbers and punctuation!

Emojis

We use two (confusingly named) libraries for these

- demoji
- emoji

Repeated Characters and Spelling Mistakes

How do we get rid of repeated characters? **RegEx**
How do we get rid of spelling mistakes? **TextBlob**

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

URLs and HTML texts

How could we detect URLs? **RegEx**
How do we parse HTML tags? **BeautifulSoup**

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(text,"html.parser")
text = soup.get_text(separator=" ")
```

Text Preprocessing Overview

There are many tasks which fall under the heading of "text pre-processing". Text pre-processing is mainly about getting the *formatting* of the text to be relatively consistent, so that only the "important" parts of the text are used in our downstream tasks. What are the "important" parts of text?

- Individual words
- "Stem forms" of words
- What else?

There is no "one size fits all" approach to text preprocessing! Always remember the downstream task!

Tokenization

Tokenization is the process of getting individual words (or sequences of words, called n-grams) from longer sequences of words. But, this is sometimes a difficult problem.

- O'Neill
- Ph.D.
- New York

We will usually use **NLTK**, the Natural Language Tool Kit, for tokenization. Below is an example for tokenizing tweets

```
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()
tokenized_tweet= tknzr.tokenize(tweet)
```

Stemming and Lemmatization

Stemming is the process of reducing a word to its stem of a word, typically by removing the suffixes (and sometimes prefixes) based on *rules*, rather than a complicated lookup procedure

- Porter stemming typically works by removing suffixes
- fish, fishes, fishing, fished $\rightarrow$ fish
- argue, argued, arguing $\rightarrow$?

Lemmatization, on the other hand, uses a dictionary to find the "root form" of the word. It does this using a dictionary.

- thought, thinking, thinks $\rightarrow$ think
- WordNet lemmatizer

In some languages (agglutinative ones), stemming and lemmatization can appear quite similar.

Part-of-Speech tagging

Different words have different grammatical functions in a sentence

- Noun Phrases
- Verbs
- Adjectives
- etc...

Stemming and lemmatization make identifying the grammatical function of a word easier (**why?**) Tagging the word with its function is called Part-of-Speech tagging

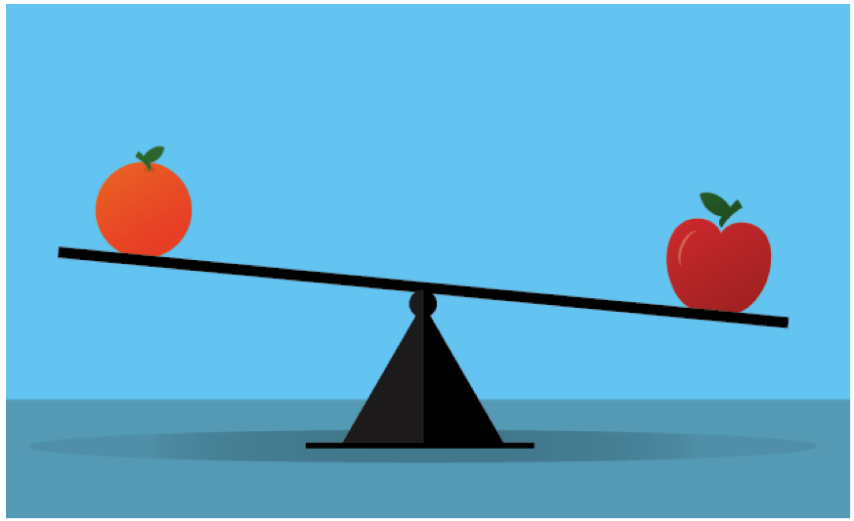
```
from textblob import TextBlob
blob = TextBlob(text)
tags_blob = blob.tags()
```

Entity Recognition

We might also want to identify words by their semantic functions, such as the names of people, places, quantities, time, etc. Why might we do this?

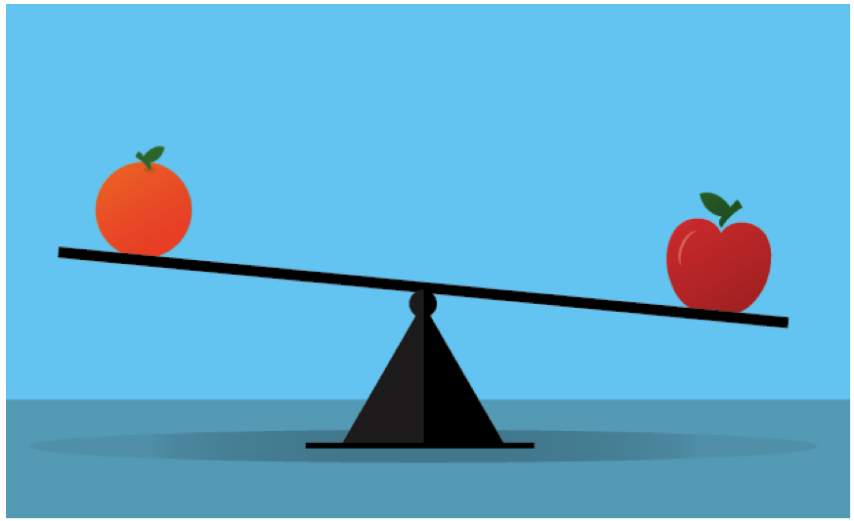
Motivation: Document representation

Why might we want to represent a document as a vector?



Motivation: Document representation

Why might we want to represent a document as a vector?



Vectors are easy to compare with one another, it allows us to compare based on objective measures of the document

Document representation: methods

- Binary Representation
- Bag-Of-Words Representation (CountVectorizer, TF-IDF)
- Dense Vector Representation (more advanced, won't be covered here)

Count Vectorizer and Binary Vectorizer

Count vectorizers show the frequency of each word in each document in a matrix, where each column is a word and each row is a document. Each cell is then the corresponding frequency of a given word in a given document.

	and	bacon	beans	beautiful	blue	breakfast	brown	but	dog	eggs	...	love	over	quick	sausages	sky	the	this	toast	today	very
0	1	0	0	1	1	0	0	0	0	0	...	0	0	0	0	1	1	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0	...	1	0	0	0	1	0	1	0	0	0
2	0	0	0	0	0	0	1	0	1	0	...	0	1	1	0	0	2	0	0	0	0
3	1	1	1	0	0	1	0	0	0	1	...	0	0	0	1	0	0	0	1	0	0
4	1	1	0	0	0	0	0	0	0	1	...	1	0	0	1	0	0	0	0	0	0
5	1	0	0	0	1	0	1	0	1	0	...	0	0	1	0	0	2	0	0	0	0
6	1	0	0	1	1	0	0	0	0	0	...	0	0	0	0	2	2	0	0	1	2
7	0	0	0	0	0	0	1	1	1	0	...	0	0	1	0	0	2	0	0	0	0

A binary vectorizer is a simplified version, where each cell is either a 1 or a 0 to indicate whether a word is present in the document (not the count, just the presence or absence of the word)

TF-IDF

Term Frequency-Inverse Document Frequency. $TF(w, d)$ is the number of times word w appears in document d. $IDF(w)$ is the rarity of the word in the corpora (collection of documents), calculated as

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

Where n is the number of documents and $df(w)$ is the number of documents containing w. We then calculate the TF-IDF weight, which shows how important a word w is to a document d, as follows

$$TFIDF(w, d) = TF(w, d) \times IDF(w)$$

Motivation: Word Embedding

The theory of word embedding will be discussed more in depth in your lectures with Prof Viviani. But we will discuss some practical aspects to it now. Word embedding is the process of representing words as *vectors*, similar to document representation but for words. **Why would we want to do this?**

Motivation: Word Embedding

The theory of word embedding will be discussed more in depth in your lectures with Prof Viviani. But we will discuss some practical aspects to it now. Word embedding is the process of representing words as *vectors*, similar to document representation but for words. **Why would we want to do this?**

- Comparing semantics
- Comparing grammatical function
- Representing words in a continuous space (allows for a richer representation of documents)

Word2Vec

One common word embedding technique is Word2Vec proposed by Mikolov et al in 2013. The technique uses one of either model:

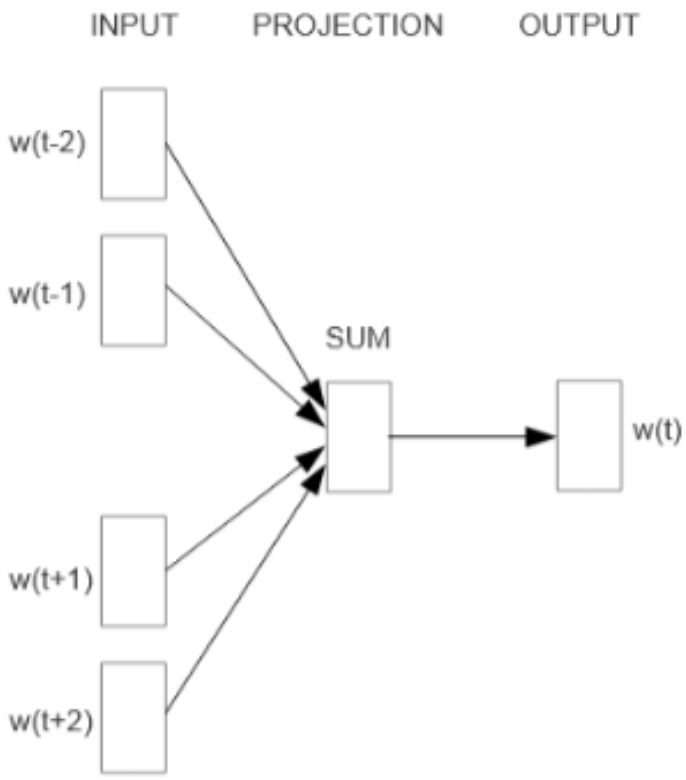
- CBOW (continuous bag of words)
- Skip-gram

The intuition behind these two approaches is that we can learn a lot about a word via training for the following two tasks:

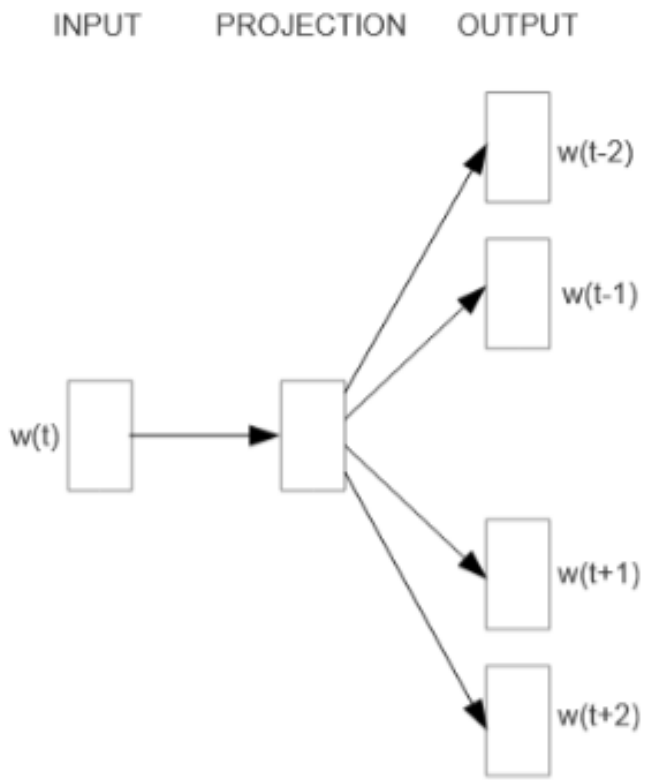
- **predicting a word based on the surrounding words**
- **predicting the surrounding words based on a word**

We train for these objectives using *shallow neural networks*, which means we learn the vector representation as a side-effect.

Word2Vec Illustrated



CBOW



Skip-gram

Word2Vec in Python

```
from gensim.models import Word2Vec
#Train the model
model = Word2Vec(sentences = corpus, vector_size = 100,
window =5, min_count =1 , workers = 4)
#Save the model
model.save(path)
```

Vector_size is the dimension of the "word space" (i.e how many numbers we are using to represent a word). Window is the context length.

GLoVe

GLoVe is a more global approach than word2vec, which only considers local context. GLoVe instead considers global context, and creates vectors by making a co-occurrence matrix of terms.