92586
COMPUTATIONAL LINGUISTICS

Alberto Barrón-Cedeño
Alma Mater Studiorum–Università di Bologna

February 10, 2020

Contents

Preliminaries

0.1 Status

These notes are been produced for the 2020 Computational Linguistics course, held at the Department of Interpreting and Translation, Alma Mater Studiorum–Università di Bologna. The notes and the course cover Computational Linguistics/Natural Language Processing dealing with text (and not speech).

As of February 2020 it is in an early stage and it will be developed as the course advances.
Last update: 27/02/2020

0.2 Requirements

Understanding and acting in the field of computational linguistics require some preliminary knowledge and skills (part of which are intended to be acquired during the course):

Required

- Basic linguistics
- Basic algebra
- Basic knowledge of the Python programming **language**[1]

Desirable

- Intermediate programming (e.g., object-oriented programming, testing)
- Version control (git)
- High-performance computing (e.g., slurm)

0.3 Materials

These notes are being developed by considering different materials. Among them:

[1]Right: this is just a (formal) language. Its vocabulary is tiny when compared to any natural language and its grammar is extremely simple.

1. The book Natural Language Processing in Action Lane et al. (2019).
2. Numerous Wikipedia articles on relevant topics.[2]

Some other materials will be considered, including

1. The book Neural network methods for natural language processing.
2. The book Linguistic fundamentals for natural language processing : 100 essentials from morphology and syntax.

[2]Over the years, numerous scholars have challenged the value of the Wikipedia as an academic resource. I argue in favour, as fas as it is used as a departing point to deepen into the consulted concepts.

Chapter 1

Introduction

The starting point of computational linguistics can be traced back to the 1950's, with Alan Turing (1950) *Computing Machinery and Intelligence.*[1] Given the difficulty of formally defining what the ability of *thinking* meams, Turing proposes a shortcut in the form of a game. The "imitation game" can be defined as follows. Let player A be a machine. Let player 'B be a human being. Let player C be another human being. The role of C is interrogating both A and B, without direct contact to determine who of them is the machine. If C cannot determine consistently who is the machine, then A is a thinking entity —and wins the game. To be able to win the game —among many other abilities— A has to be able to both understand and produce natural language.

Computational linguistics (aka natural language processing; NLP) is multidisciplinary by nature. As the Wikipedia article about it defines it

Definition 1. *"Natural Language processing is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data."*[2]

Notice that the definition by Lane et al. (2019, p. 4) is fairly different:

Definition 2. *Natural language processing is an area of research in computer science and artificial intelligence (AI) concerned with processing natural languages such as English or Mandarin. This processing generally involves translating natural language into data (numbers) that a computer can use to learn about the world. And this understanding of the world is sometimes used to generate natural language text that reflects that understanding.*

In the old days, NLP was rule-based. Models were based on a number of hand-crafted rules or grammars. By the 1990s, NLP became "statistical" by the creation of techniques that learned the rules by statistical inference from corpora.

[1] The paper is available for download at `https://academic.oup.com/mind/article-pdf/LIX/236/433/9866119/433.pdf`.

[2] `https://en.wikipedia.org/wiki/Natural_language_processing`

Table 1.1: Non-exhaustive list of NLP applications (partially derived from (Lane et al., 2019, p. 8)).

Search	**Web**. Given a query, retrieve the most relevant documents from the Web.
	Autocomplete. Searching for the most likely next item given a sequence of text.
Editing	**Grammar**. Identifying potential grammar issues in a text.
Dialog	**Chatbot**. A system that can interact (assist) a user in a conversation.
Email	**Spam filter**. Identifying commercial, phishing and other undesired email messages.
	Classification. Organising email messages according to their nature (e.g., trips, finance, entertainment).
Text mining	**Summarization**. Automatic creation of summaries given one or multiple documents.
News	**Event detection**. Grouping of the coverage of a specific event by different media.
	Fact checking. *Machine-assisted* verification of the factuality of a claim given certain evidence.
Attribution	**Plagiarism detection** Determining whether a text has been borrowed from another document (without giving proper credit).
	Literary forensics Determining whether a document has been actually written by its claimed author.
Sentiment analysis	**Product review triage**. Ranking reviews on a product/service according to their quality.
	Customer care. Understanding the level of satisfaction/stress of a client to prioritise response.
Creative writing	Generation of movie scripts, narrative, poetry, lyrics on specific topics and with a given style.

NLP ranges from the simple counting of tokens in a text to dig into the use of language to sophisticated models aiming at understanding and producing human language. No long ago, search engines were only able to process a number of keywords and would roughly combine them to perform a better search. Nowadays, natural-language queries can be processed accurately and information needs are fulfilled better. Table 1.1

As stressed Hobson Lane stresses in (Lane et al., 2019, p. xvi), often multiple names are used when referring to the same concept or idea in our field. A Markov chain, which defines the likelihood of a sequence of elements (e.g., words), is a table with probabilities. Such probabilities could be computed by counting in a large corpus —by maximum likelihood estimation. Once again, these probabilities compose a probability distribution which determines the probability of a given word conditioned to the previous one. This is no other than a language model.

Chapter 2

Words

The definition of what is a word is not precise nor coincides across languages. Departing from the English Wikipedia entry on the topic,[1] we can define a `word` from two perspectives:

Speech A word is the smallest sequence of phonemes that can be uttered in isolation with objective or practical meaning.

Text A word is a sequences of graphemes ("letters") delimited by spaces or by other graphical conventions.

Agreeing to a global definition of a word goes beyond the course and indeed we will be dealing with text only. Therefore, in general, we can assume the following simplistic definition:

Definition 3. *A word is a sequence of characters surrounded by spaces (or punctuation marks).*[2]

We will also refer to the set of all words as the `lexicon`. This could apply to a person (i.e. her vocabulary), to a document, or to a corpus.

2.1 Tokenisation

In this section we focus on the extraction of the tokens in a text; the words in it. Whereas other smaller units might be relevant for diverse tasks (e.g., syllables), at this specific stage tokens are enough to represent a text.

Using a simple regular expression, like the one we observed for identifying sequences of upper- and lower-case characters, is not enough to perform proper tokenization.[3] Other bits of information are often relevant, such as numbers and punctuation marks. In different languages —and Italian is one of them— contractions have to be split as well.

Python for Poets

[1] `https://en.wikipedia.org/wiki/Word`

[2] Observe that this is not a precise enough definition. In English, `don't` is one or two words? In Italian, `l'orsa` represents one or two words? These ambiguitiies occurr already with languages which use spaces. What about Chinese, Japanese, and others? Haspelmath (2011) discuses about the (lack of) viability of a cross-language definiton of word. See (Bender, 2013) as well.

[3] Not even Python's `.split()` method!

Table 2.1: An example of n-grams at the word level for $n = [1, 5]$.

n	**input:**The Voyage of Life is a series of paintings
1	The · Voyage · of · Life · is · a · series · of · paintings
2	The Voyage · Voyage of · of Life · Life is · is a · a series · series of · of paintings
3	The Voyage of · Voyage of Life · of Life is · Life is a · is a series · a series of · series of paintings
4	The Voyage of Life · Voyage of Life is · of Life is a · Life is a series · is a series of · a series of paintings
5	The Voyage of Life is · Voyage of Life is a · of Life is a series · Life is a series of · is a series of paintings

Tokens

Fortunately, there are plenty of libraries in multiple programming languages —and for multiple languages— that can perform a proper (perhaps not perfect) tokenisation for us.[4]

2.2 n-Grams

Python for Poets

Another sensitive representation of texts is that of the n-grams. An n-gram is a sequence of elements with length n, and usually a shift of 1 is applied for each n-gram in a text. Such elements can take diverse forms, but the typical ones are words and characters. Table 2.1 shows an example. The standard representation based on single words is a special case of n-grams where $n = 1$ (unigrams).

2.3 Normalisation

Often tokenisation is not enough. The requirements of our application might involve ignoring case folding, or perhaps we want to make a verb match regardless of its form.

Python for Poets

Casefolding In the case of casefolding, the aim is ignoring differences in the spelling of a word involving only capitalisation (Lane et al., 2019, p. 54). As a result of casefolding, tokens like `TEA`, `Tea`, or `tea` become the same. Notice that this could cause undesired effects, as after casefolding `The Joker` (the character) and the joker (the playing card) also become the same.

Prepro

Stemming Stemming consists of grouping together the different inflections of a word into the same bucket (Lane et al., 2019, p. 32). In other words, it consists of dropping the affixes of a word to obtain its stem.

For instance, `wait` is the stem of the verb `wait` in all its inflected variants: `wait` (infinitive), `wait` (imperative), `waits` (present, 3rd person, singular), `wait` (present, other persons and/or plural), `waited` (simple past), `waited` (past participle), and `waiting` (progressive).[5] Stemming drastically reduces the vocabulary at hand and allows for finding matches, even across variations. On the

[4]See for instance the natural language toolkit (NLTK) (Bird et al., 2009).

[5]Example borrowed from `https://en.wikipedia.org/wiki/Word_stem`; last visited: 28/02/2020.

Table 2.2: Instances of stopwords in three languages, sampled from https://github.com/stopwords-iso.

English		Spanish		Italian	
i	do	a	es	altri	quello
me	the	ahora	unas	certa	solito
my	will	alli	vez	della	va
it	other	cerca	yo	nessuna	via
is		el		prima	

other hand, two completely unrelated tokens might end up sharing identical strings, hence altering their meaning.

Some of the most popular stemming projects are those from Porter and Snowball.[6] Most of them can trace their origin back to Porter (1980).

Lemmatisation is a more sophisticated normalisation than stemming. It consists of associating several words down to their semantic common root (Lane et al., 2019, p. 59). The process in this case does not consist of simply dropping substrings of each token. Instead, the process consists of finding the lemma of a word; that is, its dictionary form. For instance, the lemma of `better` is not `bett`, but `good`. Prepro

Bare in mind that lemmatising is more expensive than stemming. Rather than being based on the application of regular expressions, it requires a knowledge base of synonyms and endings and a part-of-speech tagger, among others.

Stopwords . are those common words in a language that occur with a high frequency, but carry much less substantive information about the meaning of a phrase (Lane et al., 2019, p. 51–54).

Two common ways exist to define and use a list of stopwords: (*i*) considering the most frequent tokens in a reference corpus as stopwords (e.g., the Brown corpus for English) or (*ii*) taking an existing list of stopwords.[7] Table 2.2 shows some examples.

There are various reasons why stopwords are often discarded. Usually, they are the most frequent tokens in free text. As a result, discarding them results in both spacial and temporal savings. A typical stopwords list contains a few hundred words. Bare in mind in some scenarios (e.g., dialogue systems) stopwording is not always a good idea. Tokens

Other kinds of normalisation exists which usually are not considered in English. One of them is the removal of diacritics. For instance, the Italian `perché` would become `perche`.

2.4 Representation

[6] Refer to https://tartarus.org/martin/PorterStemmer/ and http://snowball.tartarus.org/

[7] For instance, from NLTK, sklearn, or https://github.com/stopwords-iso

Tokenisation and the different normalisation alternatives represent a typical pre-processing pipeline for natural language processing.[8] The next step is producing a representation of a text (or a corpus).

Bag-of-Words Representation. We can produce a Boolean vector to represent each of the documents in the corpus. In a Boolean vector, each dimension can allocate a value of 0 or 1. As a result, given a corpus c, the cardinality of the vectors will be that of the size of lexicon in c. The vector representing each document $d \in c$ will have a dimension set to 1 for all words appearing in it; 0 otherwise.[9]

Often this Boolean representation is enough; in particular when dealing with relatively short texts.[10] Still, real-valued vectors are also common; for instance those representing the (relative) frequency of the words in a text. These vectors compose the so called bag-of-words representation (BoW). The reason is simple: the representation disregards grammar and word order, and context. It becomes a bag (multiset) of words. This representation is popular for many tasks, such as search and document classification.

Once we have built a vectorial representation, we can compute the dot product between two documents (cf. Section A.1.2) to obtain a pseudo-similarity. Indeed, having a vectorial representation of documents (and an instrument to compare them) is what is known as the `vector space model`.[11]

One-Hot Vector The BoW representation is intended to represent documents. At times, we simply aim at representing a word in vectorial fashion. One of the most popular representations for a word nowadays is the so-called one-hot vector . In this case, the cardinality of the vector is once again the size of the vocabulary, but only one of the dimensions is set to 1; the one corresponding to the word at hand.

---

[8] Typical, but not mandatory. Some tasks, for instance those relying on character n-grams, might not require any of these operations.

[9] Think here about the relevance of text normalisation (casefolding and stemming/normalisation). These vectors, being the size of the vocabulary, can be pretty large. Still, the vector representing each document is in general sparse; i.e. most of the values in it are indeed set to 0. Either 0 or 1, the vector requires memory space and computation power (there are "tricks" to deal with this!).

[10] Once again, think about it: if we are dealing with sentences, what is the likelihood of having words occurring more than once?

[11] See more at `https://en.wikipedia.org/wiki/Vector_space_model`.

Chapter 3

Applications

3.1 Sentiment Analysis

`Sentiment Analysis`, also known as opinion mining is the task of identifying the polarity of a piece of text.[1] Whereas real or Likert scales are often considered, the simplest formulation implies only three classes: positive, negative, and neutral.

One recent rule-based model for sentiment analysis is VADER, which stands for Valence Aware Dictionary for sEntiment Reasoning (Hutto and Gilbert, 2014). VADER uses a lexicon in which each tokens has an associated "sentiment" score in the form or a real number. In order to compute the sentiment of a text, it performs a combination of the weights for all words in the text.

Tokens

[1]It does not refer to real sentiment, such as love or hate.

Chapter 4

Online Resources

4.1 Free Text Collections

1. The Wikipedia[1] is an excellent collection with crowdsourced text in multiple languages. Its encyclopedic contents represent a large scale comparable corpus and its metadata allows for analysing multiple characteristics of writing and collaborative creation. Sibling Wikimedia projects, such as the Wiktionary and Wikinews are worth considering as well.

2. Project Gutenberg [2] contains $60k+$ free ebooks, also available in plain text format.

3.

4.2 Code

1. The code from Lane et al. (2019) is available at `https://github.com/totalgood/nlpia`

4.3 Notebooks

If you are enrolled in the course, you should be able to get the following Jupyter notebooks:

1. Python for Poets (derived from Church (1994)).

2. Tokens (derived from Lane et al. (2019)).

[1] `http://www.wikipedia.org`
[2] `https://www.gutenberg.org/wiki/Main_Page`

Bibliography

Bender, E. M.
2013. *Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax.* Morgan & Claypool Publishers.

Bird, S., E. Loper, and E. Klein
2009. *Natural Language Processing with Python.* O'Reilly Media Inc.

Church, K.
1994. UNIX for poets.

Haspelmath, M.
2011. The indeterminacy of word segmentation and the nature of morphology and syntax. *Folia Linguistica*, 45.

Hutto, C. and E. Gilbert
2014. VADER:A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14)*, Ann Arbor, MI.

Lane, H., C. Howard, and H. Hapkem
2019. *Natural Language Processing in Action.* Shelter Island, NY: Manning Publication Co.

Porter, M.
1980. An algorithm for suffix stripping. *Program*, 14:130–137.

Turing, A.
1950. Computing machinery and intelligence. *Mind*, 59:433–460.

Index

Appendix A

Non-Exhaustive Mathematical Concepts

A.1 Algebra

A.1.1 Vectors

An (Euclidean) vector is an entity endowed with a magnitude (the length of the line segment (A, B)) and a direction (the direction from A to B).

TODO

A.1.2 Dot product

The dot product is an algebraic operation between two vectors of the same size. It is the sum of the products of the corresponding entries of the two sequences of numbers $a \cdot b$:

Tokens

$$\begin{aligned} a \cdot b &= \sum_{i=1}^{n} a_i b_i && \text{(A.1)} \\ &= a_1 b_1 + a_2 b_2 + a_3 b_3 + \ldots a_n b_n && \text{(A.2)} \end{aligned}$$

For instance, let $a = [1, 2, 3]$ and $b = [3, 4, 6]$. Then

$$\begin{aligned} a \cdot b &= 1 \cdot 3 + 2 \cdot 4 + 3 \cdot 6 \\ &= 3 + 8 + 18 \\ &= 29 \end{aligned}$$

Notice that computing the dot product over binary vectors is equivalent to computing the size of their intersection.

A.2 Statistics

todo

A.2.1 Relative Frequency

todo