

Natural Language Processing

Lecture 3 : Text Normalization

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Words



©Schools Week

We need to distinguish between **words** and **punctuation marks**.

Example :

*He stepped out into the hall, was delighted to encounter
a water brother.*

The above sentence has 13 words and 2 punctuation symbols.

Punctuation is critical

- for finding sentence boundaries (period, colon, etc.)
- for identifying some aspects of meaning (question mark, etc.).

Tokens vs. types

There are two ways of talking about words

- **types** are the **distinct** words appearing in a document
- **tokens** are the **individual** occurrences of words in a document

Example :

They picnicked by the pool, then lay back on the grass and looked at the stars.

Ignoring punctuation marks, the above sentence has 16 tokens and 14 types.

The set of all types in a corpus is the **vocabulary** V .

More about corpora in later slides.

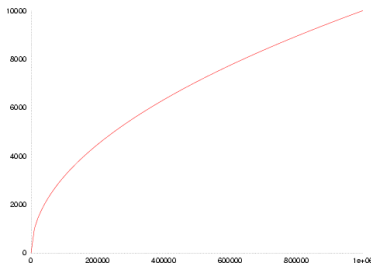
The vocabulary size $|V|$ is the number of types in the corpus.

The size of the corpus N is the number of tokens, if we ignore punctuation marks.

Example : N vs. $|V|$ for a few English corpora

Corpus	Tokens = N	Types = $ V $
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google n-grams	1 trillion	13 million

Herdan/Heaps law



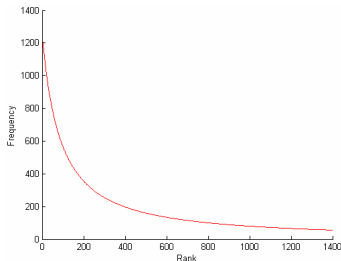
In very large corpora, the relation between N and $|V|$ can be expressed as

$$|V| = kN^{\beta}$$

where $k \in [10, 100]$ and $\beta \in [0.4, 0.75]$.

This introduces the problem of **unknown words**, also called as **out of vocabulary** (OoV) words.

Zipf/Mandelbrot law



In very large corpora, the r -th most frequent type has frequency $f(r)$ that scales according to

$$f(r) \propto \frac{1}{(r + \beta)^\alpha}$$

with $\alpha \simeq 1$ and $\beta \simeq 2.7$

Morphology is the study of how words are composed by **morphemes**, which are the smallest meaningful units of language.

The structure of a word consists of several morphemes:

- one **root** or stem
- zero or more **affixes**, such as prefixes and suffixes

Example : draw, draw+s, draw+ing+s, un+draw+able

Inflectional morphology: no change in the grammatical category

Example : give, given, gave, gives

Derivational morphology: change in the grammatical category

Example : process, processing, processable, processor, processability

The term **morphologically rich language** (MRL) refers to a language in which substantial grammatical information is expressed at word level.

Example : Arabic, Hebrew, Turkish, etc.

Word-forms and lemmas

The **word-form** is the full inflected or derived form of the word.

Each word-form is associated with a single **lemma**, the citation form used in **dictionaries**.

Example : Word-forms **sing**, **sang**, **sung** are associated with the lemma **sing**.

Alternatively to $|V|$, another measure of the number of words in a corpus is the number of lemmas.

Multi-element word-forms

A word-form can be made of several separated elements

Example : English: San Francisco; French: *sine die* (from Latin).

A word-form can be made of several merged elements

Example : English: don't = do + not; Spanish: *damélo* = da + me + lo (give + to me + it).

There is no one-to-one correspondence between tokens and word-forms

Corpora



©UniBO

Corpus (plural corpora): large collection of text, in computer-readable form, with possible addition of linguistic annotations.

Several dimensions of variation for corpora should be taken into account: language, genre, etc.

Language: It is important to test NLP algorithms on more than one language.

There is an unfortunate current tendency to develop corpora mainly for English.

Languages lacking large corpora are considered **low-resource languages**.

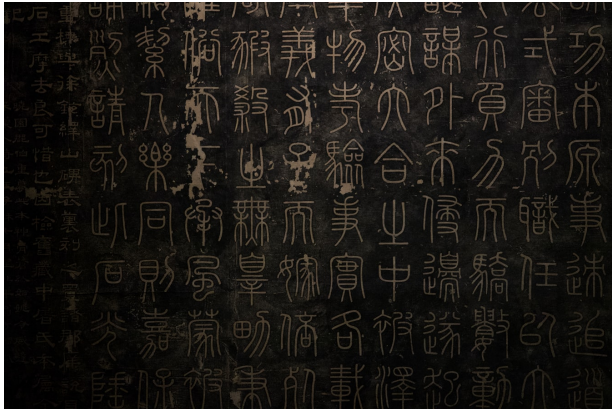
Genre: Text documents might come from newswire, fiction, scientific articles, Wikipedia, etc.

Time: Language changes over time. For some languages we have good corpora of texts from different historical periods.

Collection process: How big is the data and how was it sampled? How was the data pre-processed, and what metadata is available?

Annotation: What are the specifics of the used annotation? How was the data annotated? How were the annotators trained?

Text Normalization



Text normalization is the process of transforming a text into some predefined standard form. This consists of several tasks.

There is **no all-purpose** normalization procedure: text normalization depends on

- what type of text is being normalized
- what type of NLP task needs to be carried out afterwards

Text normalization is also important for applications other than NLP, such as text mining and WEB search engines.

In this lecture we briefly overview the most common tasks involved in text normalization

- language identification
- spell checking
- contraction
- word reconstruction
- punctuation
- special characters
- tokenization
- sentence segmentation
- case folding
- stop words
- stemming
- lemmatization

Ordering of the above modules is important. Some modules are optional.

Language identification is the task of detecting the source language for the input text.

This is preliminary to spell checking, tokenization, acronym expansion, etc.

Several **statistical** techniques for this task: functional word frequency, N -gram language models (some later lecture), distance measure based on mutual information, etc.

Explore the following libraries

- Python **langdetect**
- Apache OpenNLP **LanguageDetector**

Spell checker

Spell checkers correct grammatical mistakes in text.

Useful for quickly written text, such as X/Twitter and Amazon reviews.

Spell checkers use approximate string matching algorithms such as **Levenshtein distance** to find correct spellings.

Difficult cases: a misspelled word might still be in the language (English: than vs. then, their vs. there). To deal with these cases, more sophisticated algorithms analyze the **context** formed by the surrounding words.

Explore the following libraries

- Python **TextBlob**, based on the Natural Language Toolkit (NLTK) library
- Resources for fuzzy string matching
 - <https://github.com/seatgeek/fuzzywuzzy>
 - <https://pypi.org/project/fuzzywuzzy/>

Contractions

The following should be managed before further normalization (all examples in English language)

- **contracted forms**: we'll, don't
- **abbreviations**: inc., Mr.
- **slang**: IMHO, LOL, tl;dr

Difficult cases: ambiguity with apostrophe which is also used as

- genitive marker (book's cover)
- quotative ('The other class', she said)

The most straightforward technique is to create a **dictionary** of contractions and abbreviations with their corresponding expansions.

Uneven kerning may break words in unexpected ways (especially in pdf documents)

Example : Italian word **assicurazione** may appear in a document as **assicuraz ione**

Explore the following library

- `symspellpy`, based on the SymSpell algorithm

Punctuation marks in text need to be isolated and treated as if they were separate words. This is critical for finding sentence boundaries and for identifying some aspects of meaning (question marks, exclamation marks, quotation marks).

Difficult cases: many punctuation symbols also used in abbreviations (period), company names (Yahoo!), compound words (-), etc.

Explore the following libraries

- Python **string.punctuation**
- NLTK **nltk.punkt**

Special characters

Special care for

- **emoticons**: :) ;) etc., use regular expressions
- **emoji**: 😊 😂 etc., use specialized libraries

Explore the following library

- <https://github.com/NeelShah18/emot>

Tokenization is the process of segmenting text into units called tokens.

Tokenization techniques can be grouped into three families

- word tokenization
- character tokenization
- subword tokenization

Tokens are then organized into a **vocabulary** and, depending on the specific NLP application, may later be mapped into **natural numbers**.

Token indexing is very common when using neural networks.

Word tokenization



Glen Carrie on Unsplash

Word tokenization

Word tokenization is a very common approach for European languages.

For English, most of the text is already (word) tokenized after previous steps, with the following important exceptions

- special compound names (white space vs. whitespace)
- city names (San Francisco, Los Angeles), companies, etc.

This requires named entity recognition, presented in some later lecture.

Explore the following libraries

- NLTK **word.tokenize**
- SpaCy **Tokenization**

<https://spacy.io/usage/linguistic-features#tokenization>

German writes compound nouns without spaces.

Example : Computerlinguistik, 'computational linguistics'.

Several compound-splitter tools available.

Italian and Spanish incorporate verbs and clitics, which are special type of pronouns (see some previous slide).

Example : comprarlo > comprare + lo, 'to buy it'.

This process can be iterated on the same word.

Word tokenization

There are certain language-independent tokens that require **specialized** processing

- phone numbers: (800) 234-2333

- dates: Mar 11, 1983

- `https://dateparser.readthedocs.io/en/latest/`

- email addresses: jblack@mail.yahoo.com

- web URLs: `http://stuff.big.com/new/specials.html`

- hashtags: #nlproc

Use of **regular expressions** is recommended in the above cases.

A main problem with word tokenization is represented by unknown words (OoV words); see Herdan/Heaps law.

Character tokenization



Raphael Schaller on Unsplash

Major east Asian languages (e.g., Chinese, Japanese, Korean, and Thai) write text without any spaces between words.

For most Chinese NLP tasks, character tokenization works better than word tokenization

- each character generally represents a single unit of meaning
- word tokenization results in huge vocabulary, with large number of very rare words

Character tokenization

Example : Consider the following Chinese sentence and possible tokenizations

姚明进入总决赛

“Yao Ming reaches the finals”

姚 明 进 入 总 决 赛

Yao Ming reaches overall finals

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Character tokenization

For Japanese and Thai the character is too small a unit, and algorithms for word segmentation are required.

Standard segmentation algorithms for these languages use neural sequence models.

This is related to sequence labelling, presented in some later lecture.

Subword tokenization



Brett Jordan from Unsplash

Subword tokenization

Many NLP systems need to deal with unknown words (OoV words), that is, words that are not in the vocabulary.

Example :

If the training corpus contains the words **foot** and **ball**, but not the word **football**, then if football appears in the test set the system does not know what to do.

Example :

If the training corpus contains the words **low**, **new**, **newer** but not **lower**, then if lower appears in the test set the system does not know what to do.

Subword tokenization

To deal with the problem of unknown words, modern tokenizers automatically induce sets of tokens that include tokens smaller than words, called **subwords**.

Subword tokenization reduces vocabulary size, and has become the **most common** tokenization method for large language modelling and neural models in general (see future lectures).

Subword tokenization is inspired by algorithms originally developed in information theory as a simple and fast form of **data compression** alternative to Lempel-Ziv-Welch.

Data compression provides more interesting results than morphemes.

Subword tokenization

Subword tokenization schemes consists of three different algorithms

- the **token learner** takes a raw training corpus and induces a set of tokens, called **vocabulary**
- the **token segmenter** (encoder) takes a vocabulary and a raw test sentence, and segments the sentence into the tokens in the vocabulary
- the **token merger** (decoder) takes a token sequence and reconstructs the original sentence

Example :

Given the sample sentence 'GPT-3 can be used for linguistics'

- learner constructs the vocabulary:
{ -, 3, be, can, for, G, istics, lingu, PT, used }
- encoder translates sample sentence into token sequence:
G, PT, -, 3, can, be, used, for, lingu, istics
- decoder translates back to the original sentence, including white spaces:
GPT-3 can be used for linguistics

Subword tokenization

Three algorithms are widely used for subword tokenization

- byte-pair encoding (BPE)
- unigram language modeling
- WordPiece

Explore the following library

- **SentencePiece**

Includes implementations of BPE and unigram language modeling

The BPE **token learner** is usually run inside words, not merging across word boundaries. To this end, use a special end-of-word marker.

The algorithm iterates through the following steps

- begin with a vocabulary composed by all individual characters
- choose the two symbols A, B that are most frequently adjacent
- add a new merged symbol AB to the vocabulary
- replace every adjacent A, B in the corpus with AB

The algorithm follows a greedy approach.

Stop when the vocabulary reaches size k , a **hyperparameter**.

Stopping criterion can alternatively be the number of iterations (merges).

Example : Underscore is the end-of-word marker

corpus

```
5   l o w _  
2   l o w e s t _  
6   n e w e r _  
3   w i d e r _  
2   n e w _
```

vocabulary

```
_, d, e, i, l, n, o, r, s, t, w
```

Most frequent pair is **e**, **r** with a total of 9 occurrences (we arbitrarily break ties).

corpus

```
5   l o w _  
2   l o w e s t _  
6   n e w e r _  
3   w i d e r _  
2   n e w _
```

vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, e r
```

The algorithm now learns the word-final token **er_**

corpus

```
5   l o w _  
2   l o w e s t _  
6   n e w er_  
3   w i d er_  
2   n e w _
```

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er

The next merge produces token **ne**

corpus

```
5   l o w _  
2   l o w e s t _  
6   ne w er_  
3   w i d er_  
2   ne w _
```

vocabulary

, d, e, i, l, n, o, r, s, t, w, er, er, ne

If we continue, the next merges are

Merge	Current Vocabulary
(ne, w)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new
(l, o)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo
(lo, w)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo, low
(new, er—)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo, low, newer—
(low, —)	—, d, e, i, l, n, o, r, s, t, w, er, er—, ne, new, lo, low, newer—, low—

After several iterations, BPE learns

- entire words
- most frequent units, useful for tokenizing unknown words

Two versions of **BPE token segmenter** (encoder)

- split data set into characters, and apply merge rules in **frequency order**
- for each word, **left-to-right**, match longest token from vocabulary (eager)

Not clear whether the two algorithms always provide the same encoding.

Example :

Assume training corpus contained words **newer**, **low**, but not **lower**. Typically, the test word [lower] will be encoded by means of tokens [low, er_].

Encoding is **computationally expensive**.

Many systems use some form of **caching**:

- pre-tokenize all the words and save how a word should be tokenized in a dictionary
- when an unknown word (not in dictionary) is seen
 - apply the encoder to tokenize the word
 - add the tokenization to the dictionary for future reference

BPE token merger: To decode, we have to

- concatenate all the tokens together to get the whole word
- use the end-of-word marker to solve possible ambiguities

Example :

The encoded sequence

```
[the_, high, est_, range_, in_, Seattle_]
```

will be decoded as

```
[the, highest, range, in, Seattle]
```

as opposed to

```
[the, high, estrange, in, Seattle]
```

Since BPE is used at the level of characters, it shows the following problems

- requires predefined character vocabulary
- language-dependent preprocessing
- struggles with rare unicode characters

What if we eliminate the character vocabulary entirely?

Byte-level byte-pair encoding (BBPE) is a variant of BPE used to tokenize text at the level of bytes.

This is the *de facto* standard now for LLMs.

BBPE operates directly on raw UTF-8, which encodes each character into 1 to 4 bytes

- base vocabulary = 256 possible byte values
- no unknown characters
- language-agnostic

In the end, BBPE vocabulary can be

- partial characters shared by different characters
- the combination of complete and partial characters

WordPiece is a subword tokenization algorithm used by the language model BERT.

BERT will be presented in a later lecture.

Like BPE, WordPiece starts from the initial alphabet and learns merge rules.

The main difference is the way pair A, B is selected to be merged ($f(X)$ is the frequency of token X)

$$\frac{f(A, B)}{f(A) \times f(B)}$$

The algorithm **prioritizes** the merging of pairs where the individual parts are less frequent in the vocabulary.

Sentence segmentation

Text normalization also includes **sentence segmentation**:
breaking up a text into individual sentences

This can be done using cues like periods, question marks, or exclamation points.

Lower casing is very useful to standardize words and before stop words removal.

After this step, you can perform word statistics, such as Zip's law.

Difficult cases: if some words must be capitalized (proper names, cities), specialized pre-processing is needed.

Sometimes it might be useful to keep both versions of the text data.

Most programming languages have facilities for string lowercasing.

Stop word removal includes getting rid of

- common articles
- pronouns
- prepositions
- coordinations

Stop word removal heavily depends on the task at hand, since it can wipe out relevant information.

This is more of a dimensionality reduction technique than normalization. Much more common in IR than in NLP.

Stemming refers to the process of slicing a word with the intention of removing affixes.

Stemming is **problematic** in the linguistic perspective, since it sometimes produces words that are not in the language, or else words that have a different meaning.

Example :

- arguing > argu, flies > fli
- playing > play, caring > car
- news > new

Much more commonly used in IR than in NLP. Porter and Snowball stemmers very popular (rule based). For low-resource languages statistical stemmers are also an option.

Lemmatization has the objective of reducing a word to its base form, also called **lemma**, therefore grouping together different forms of the same word.

Example :

- am, are, is > be
- car, cars, car's, cars' > car

Lemmatization and stemming are mutually exclusive, and the former is much more resource-intensive than the latter.

Explore the following library

- TextBlob, word inflection and lemmatization

`https://textblob.readthedocs.io/en/dev/`

Should we always normalize?

We need to ask ourselves

- is important information being lost?
- is noisy information being removed?