Introduction to Computational Linguistics and Natural Language Processing

Miquel Esplà-Gomis

November 21, 2025

Presentation

- Welcome to Natural Language Processing Techniques
- Two teachers from the *Department of Software and Computing Systems*:
    - **Miquel Esplà Gomis** (myself): first session
    - Juan Antonio Pérez Ortiz: rest of sessions
- All the information is on the subject website: `https://mespla.github.io/tpln2526/`
- Attending classes:
    - It is mandatory and we will take attendance
    - It is allowed to miss one session without a justification

Evaluation

- All the information is available on the official teaching guide: `https://cvnet.cpd.ua.es/Guia-Docente/GuiaDocente/Index?wlengua=en&wcodasi=43505&scaca=2025-26`
- Three elements in evaluation:
  - Practical activities at class (that you may need to complete at home): 60%
  - Tests on the materials you prepare before class : 10%
  - Final test on January the 31st: 30%

Índex

Computational Linguistics (CL)

- An interdisciplinary field at the intersection of:
  - Linguistics
  - Computer Science
- Focuses on modeling human language using computational methods.
- Key objectives:
  - Analyze and understand natural language.
  - Develop linguistic theories supported by computational tools.
- Example topics:
  - Parsing syntactic structures.
  - Phonetics and phonology modeling.

Natural Language Processing (NLP)

- A subfield of Artificial Intelligence (AI) and Machine Learning (ML).
- Focuses on designing algorithms and systems to process natural language data.
- Key objectives:
  - Automate language-based tasks.
  - Enable machines to interact with humans through language.
- Example applications:
  - Machine Translation (e.g., Google Translate).
  - Sentiment Analysis.
  - Question Answering Systems.

Differences Between CL and NLP

- **Computational Linguistics (CL):**
  - Emphasizes theoretical understanding of language.
  - Grounded in linguistic principles.
- **Natural Language Processing (NLP):**
  - Focuses on practical applications of language processing.
  - Driven by engineering and computational efficiency.

Commonalities Between CL and NLP

- Both fields deal with natural language data.
- Share methods and tools, such as:
  - Syntax and semantics modeling.
  - Statistical and machine learning techniques.
- Work towards improving human-computer interaction through language.

What will we talk about during this session?

- **Text preprocessing**: preparing text for NLP applications
- **Morphological parsing**: how much information can we extract from words to better understanding text?
- **Syntactic parsing**: and what about the structure of the words in a sentence?
- **Vector representations of text**: how can we feed text in models that build on math?

Índex

Text Preprocessing: An Essential Step

- Text preprocessing prepares raw text for effective processing by algorithms and models.
- Ensures consistency, reduces noise, and optimizes data for downstream tasks.
- Different tasks require different preprocessing steps.

Some frequent sub-tasks in Text Preprocessing

- **Format Cleaning:**
  - Remove unwanted formatting, such as HTML tags or PDF metadata.
- **Text Tokenization:**
  - Split text into smaller units like sentences, words, subwords, or characters.
- **Text Normalization:**
  - Normalize punctuation and spaces.
  - Convert text to lowercase for case-insensitive processing.
- **Dealing with Punctuation:**
  - Remove or retain punctuation depending on the application.
- **Identifying Stopwords:**
  - Remove commonly used words (e.g., *the*, *is*, *and*) to focus on meaningful content.

Índex

Why do we need to clean format?

- Raw text often comes in formats unsuitable for NLP models:
  - HTML files with tags and scripts.
  - PDF documents with metadata and layout information.
  - Markdown files with format marks.
- In most cases data related to format adds noise to the text to be processed.

Removing Formatting: HTML and PDF

- **HTML:**
  - Often contains tags (`<div>`, `<script>`, etc.) and styles.
  - Content extraction involves ignoring these elements.
- **PDF:**
  - May include page numbers, headers, and images.
  - Text extraction tools can help retrieve only textual content.

Example: Removing HTML Tags

Raw Text:

```
<html>
    <head><title>Example</title></head>
    <body>
        <h1>Hello, World!</h1>
        <p>This is a sample text.</p>
    </body>
</html>
```

Cleaned Text:

```
Hello, World!
This is a sample text.
```

Techniques for Removing Formatting

- Regular Expressions (Regex):
  - Use patterns to identify and remove unwanted elements.
  - Example: Remove HTML tags using the pattern `<.*?>`.
- Libraries and Tools:
  - `BeautifulSoup` (Python): For parsing and cleaning HTML.
  - `PyPDF2` (Python): For extracting text from PDFs.
- OCR Tools:
  - Use Optical Character Recognition for images or scanned text.

Common Challenges in Formatting Removal

- Handling noisy or incomplete data.
- Retaining meaningful structure (e.g., tables, paragraphs).
- Managing large or complex files efficiently.
- Language-specific formatting (e.g., RTL scripts or special characters).

Índex

What is Tokenization?

- The process of breaking text into smaller units, called **tokens**.
- Tokens can represent:
  - Sentences (it is also usual to call this task *sentence splitting*)
  - Words
  - Characters
  - Sub-words
- Critical for transforming raw text into a format suitable for NLP algorithms.

Levels of Tokenization

- **Sentence Tokenization:**
  - Splits text into sentences.
  - Example: *"NLP is fascinating. Tokenization is essential."*
  - Tokens: *["NLP is fascinating.", "Tokenization is essential."]*
- **Word Tokenization:**
  - Splits text into words.
  - Example: *"NLP is fascinating"*
  - Tokens: *["NLP", "is", "fascinating"]*
- **Character Tokenization:**
  - Splits text into individual characters.
  - Example: *"NLP"*
  - Tokens: *["N", "L", "P"]*
- **Sub-word Tokenization:**
  - Splits words into potentially meaningful sub-units.
  - Example: *"unbelievable"*
  - Tokens: *["un", "believ", "able"]*

Sentence splitting

- For most languages, **punctuation** is used (split by colons, semicolons, etc.)
  - This is sometimes difficult: "*This is Dr. Smith. He is the author of the blog* `saludparatodos.net`."
- Sometimes it is possible to use **format**;for example, some HTML tags delimit a text block, such as `<p>` or `<h1>`
- **Some languages do not use punctuation** (Thai, for example)

ประวัติ

ภาษาไทยจัดอยู่ในกลุ่มภาษาไท (Tai languages) ภาษาหนึ่ง ซึ่งเป็นสาขาย่อยของตระกูลภาษาขร้า-ไท ภาษาไทยมีความสัมพันธ์อย่างใกล้ชิดกับภาษาในกลุ่มภาษาไทตะวันตกเฉียงใต้ภาษาอื่น ๆ เช่น ภาษาลาว ภาษาผู้ไท ภาษาคำเมือง ภาษาไทใหญ่ เป็นต้น รวมถึงภาษาตระกูลไทอื่น ๆ เช่น ภาษาจ้วง ภาษาเหมาหนาน ภาษาปู้อี ภาษาไหล ที่พูดโดยชนพื้นเมืองบริเวณไหหนาน กวางสี กวางตุ้ง กุ้ยโจว ตลอดจนยูนนาน ไปจนถึงเวียดนามตอนเหนือ ซึ่งสันนิษฐานว่าจุดกำเนิดของภาษาไทยน่าจะมาจากบริเวณดังกล่าว

Word Tokenization Strategies

- **Whitespace-Based Tokenization:**
  - Fails with contractions or punctuation, for example.
  - Words separated with a dash in English: *state-of-the-art*.
  - What to do with languages that don't use spaces to separate words?
- **Using regular expressions:**
  - Allows to identify some phenomena: some contractions in English, URLs, etc.
- **Language-Specific Tokenizers:** Tailored to account for language-specific features.
  - Example: Tokenizing Japanese using MeCab or SudachiPy.
  - There are tokenizers that build on knowledge (morphological dictionaries) and that build on statistical models (for example, HMM).

Why Sub-word Tokenization?

It has become very popular in neural-based NLP models:

- Addresses issues with **rare words** and **out-of-vocabulary** (OOV) words.
- Efficient for **morphologically rich languages**.
- Maintains a balance between word and character tokenization.

Approaches to Sub-word Tokenization

Task traditionally based on **morphological segmentation**.

Two popular strategies in the neural age:

- Byte Pair Encoding (BPE)
- Unigram Language Model

Byte Pair Encoding (BPE)

- Begins by splitting text in characters.
- Iteratively merges the most frequent pairs of characters or subwords.
- Example:
  - Initial tokens: `["l", "o", "w", "e", "r"]`
  - Merge "l" and "o" → `["lo", "w", "e", "r"]`
  - Merge "lo" and "w" → `["low", "e", "r"]`
- Benefits:
  - Handles rare words by breaking them into sub-units.
  - Compact vocabulary size.

Unigram Language Model

- Steps:
  - Start with a large vocabulary of potential subwords (could be all the possible sub-words in the corpus).
  - Assign a probability to each of them according to their frequency observed in the corpus.
  - Use the vocabulary as an unigram model that allows to obtain the probability of a word.
  - Iteratively remove subwords that minimally impact the overall probability of the corpus.
- Benefits:
  - Allows for multiple segmentations with probabilities.
  - More flexible than deterministic methods like BPE.

Comparison: BPE vs. Unigram

- **BPE:**
  - Deterministic.
  - Fixed segmentation after training.
- **Unigram:**
  - Probabilistic.
  - Allows multiple valid segmentations with probabilities.
- **In common:**
  - Both require pre-tokenization.
  - Both allow to specify the size of the final vocabulary.

SentencePiece

- It builds on Unigram or BPE
- Unigram and BPE assume that the corpus can be split in words by blank spaces.
- SentencePiece just omits this assumption:
  - Includes spaces in the initial vocabulary of BPE.
  - Includes sub-words containing spaces in the initial vocabulary of Unigram.
- Allows dealing with languages that do not use blank spaces.
- Allows using multi-word expressions as elements in the final vocabulary.

What is Text Normalization?

- The process of converting text into a standard form.
- Aims to reduce variability in the text while preserving meaning.
- Prepares text for consistent and effective processing in NLP tasks.

Examples:

- *"Hello,"* $\rightarrow$ *"hello"* (lowercasing).
- *"I've got 2 apples."* $\rightarrow$ *"i have two apples"* (normalizing contractions and numbers).

Challenges with Unicode Characters

- Modern text is usually encoded with **Unicode** that support a wide variety of scripts. Sometimes this leads to data sparsity at character level.
- **Visually Similar Characters:**
  - \á" (U+00E1) vs. \a´" (U+0061 + U+0301).
  - Appear identical but have different underlying representations.
  - With punctuation it is even worse; have a look to UTF-8 punctuation at: https://www.compart.com/en/unicode/category/Po
- **Non-breaking Spaces and Invisible Characters:**
  - \ " (non-breaking space, U+00A0) vs. \ " (space, U+0020).
  - Introduce subtle errors in processing.

Task-Specific Normalization Needs

- Text normalization varies depending on the NLP task.
- **Examples:**
  - **Case-sensitive tasks:**
    - Named Entity Recognition (NER): Retain original casing to identify entities like *"Apple"*.
  - **Removing Punctuation:**
    - Useful for bag-of-words models.
    - Not always suitable for tasks like sentiment analysis.
  - **Removing redundant text:**
    - Removing duplicate or almost-duplicate sentences or paragraphs in a long corpus.
    - Useful when we have a large corpus to training generative models.

  - **Removing redundant text:**
    - Removing duplicate or almost-duplicate sentences or paragraphs in a long corpus.
    - Useful when we have a large corpus to training generative models.

- Normalization must strike a balance between generality and task-specific requirements.

Summary of Text Normalization

- Essential for standardizing text and reducing variability.
- Unicode introduces challenges like visually similar characters and non-breaking spaces.
- Techniques include lowercasing, punctuation removal, and whitespace normalization.
- Task-specific normalization must be tailored to the application.

Key Takeaway: Effective normalization improves downstream NLP performance.

Índex

What is Text Normalization?

- The process of converting text into a standard form.
- Aims to reduce variability in the text while preserving meaning.
- Prepares text for consistent and effective processing in NLP tasks.

Examples:

- *"Hello,"* $\rightarrow$ *"hello"* (lowercasing).
- *"I've got 2 apples."* $\rightarrow$ *"i have two apples"* (normalizing contractions and numbers).

Challenges with Unicode Characters

- Modern text is encoded in **UTF-8**, which supports diverse characters.
- **Visually Similar Characters:**
  - `\á"` `(U+00E1)` vs. `\á"` `(U+0061 + U+0301)`.
  - Appear identical but have different underlying representations.
- **Non-breaking Spaces and Invisible Characters:**
  - `\ "` `(non-breaking space, U+00A0)` vs. `\ "` `(space, U+0020)`.
  - Introduce subtle errors in processing.
- **Normalization Standards:**
  - Use Unicode normalization forms (e.g., NFC, NFD) to standardize representations.

Common Text Normalization Techniques

- **Lowercasing:**
  - Converts all text to lowercase.
  - Example: *"NLP"* $\rightarrow$ *"nlp"*.
- **Removing Punctuation:**
  - Removes special characters and punctuation marks.
  - Example: *"Hello, world!"* $\rightarrow$ *"Hello world"*.
- **Whitespace Normalization:**
  - Replaces multiple spaces or non-breaking spaces with a single space.
  - Example: *"Hello world"* $\rightarrow$ *"Hello world"*.
- **Stemming or Lemmatization:**
  - Reduces words to their base or root forms.
  - Example: *"running"* $\rightarrow$ *"run"*.

Task-Specific Normalization Needs

- Text normalization varies depending on the NLP task.
- **Examples:**
  - **Case-sensitive tasks:**
    - Named Entity Recognition (NER): Retain original casing to identify entities like *"Apple"*.
  - **Removing Punctuation:**
    - Useful for bag-of-words models.
    - Not always suitable for tasks like sentiment analysis.
  - **Normalization of Numbers:**
    - Convert digits to words for readability or analysis (e.g., *"3"* $\rightarrow$ *"three"*).
- Normalization must strike a balance between generality and task-specific requirements.

Índex

What are Stopwords?

Stopwords are common words that are consider to contain low semantic value and are removed during text preprocessing for some NLP tasks.

- Examples: "the", "is", "in", "on", "at", "and", "for"
- Removing them can improve efficiency and focus on more important words
- They are more usual in languages with low morpholocial complexity.

How to Detect Stopwords

Stopwords can be detected in various ways:

- Predefined stopword lists (e.g., NLTK, SpaCy)
- Frequency-based approaches:
  - Words appearing very frequently across many documents are potential stopwords
  - Commonly occurring words across corpora are candidates
- Part-of-speech tagging:
  - Function words (e.g., determiners, prepositions, conjunctions) are often stopwords

Vocabulary frequency distribution

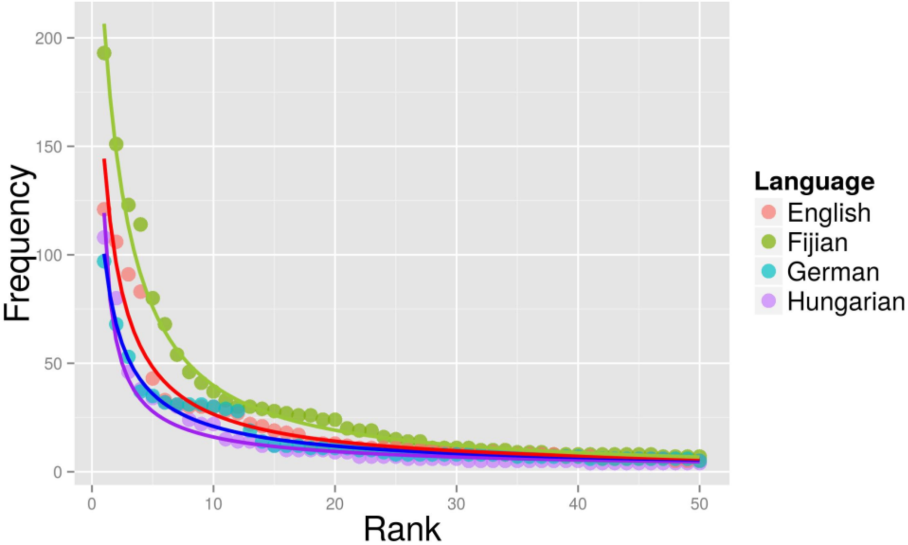


Figure 2 from: Bentz, C., Verkerk, A., Kiela, D., Hill, F., & Buttery, P. (2015). **Adaptive communication: Languages with more non-native speakers tend**

The zipfian distribution of vocabulary

- When the words in a corpus are ranked decreasingly they follow a zipfian distribution in which:

$$\mathrm{freq}(r) \propto \frac{1}{r}$$

- In other words:
  - a few words in most languages have a very high frequency, and
  - most of the words in a language are in the so called "long tail".
- The most frequent words in a language are typically function words (stopwords)

Implications of Removing Stopwords in NLP

Removing stopwords has several effects:

- **Focuses on meaningful terms:** It can help to emphasize content-bearing words for tasks like classification or clustering
- **Risk of losing context:** Removing too many stopwords may change the sentence structure and meaning
- **Task-specific considerations:** Some tasks (e.g., sentiment analysis, language modeling, etc.) may benefit from retaining stopwords

Índex

What is Morphological Parsing?

- **Morphology**: Study of the structure of words.
- **Morphological Parsing**: Breaking down words into:
  - **Lemmas**: Base forms of words.
  - **Morphemes**: Smallest units of meaning (roots, prefixes, suffixes).
- Essential for understanding word formation, meaning, and grammatical roles.

Why is Morphological Parsing Important?

1. **Handling Rich Morphology**:
   - Languages like Finnish, Turkish, or Arabic have complex word structures.
2. **Vocabulary Reduction**:
   - Groups inflected forms (e.g., *run*, *ran*, *running*) into a single base form.
3. **Text Normalization**:
   - Preprocessing for tasks like sentiment analysis and information retrieval.

Most languages in Europe have rather simple morphology

We are used to **fusional languages**: few inflectional morphemes that add information to a stem.

A word in English

Computed

Comput	ed
Setm	suffix indicating past

How complex can morphology get? 1/

There are also **agglutinative languages**: combine many inflectional morphemes each of them adding new information.

A word in Finnish

Taloissammekin

talo	i	ssa	mme	kin
house	PLURAL	INNESIVE CASE	our	also

How complex can morphology get? 2/

And then, there are **polysynthetic languages**, that put many words together.

A word in Inuktitut

annulaksikkanninginnajualugasulauqsimagumanngittsiaqgaluaqtunga

annulaksi	kkanni	nginna	jualu	gasu	lauqsima	guma	nngit	...
imprison	again	really	a lot	try	ever	want	NEG	...

I would never ever even want to try to end up in jail ever again even for a bit. (Johns, 2007)

Why is important morphology in the age of neural technologies?

Two main uses:

- **In NLP**: mostly useful for low-resource languages
  - Simplifies text (helps to segment words).
  - Extracts information relevant to understand meaning.
  - Generation morphologically-correct text.
  - Support for language learners.
- **In CL**:
  - Automatic annotation of corpora.
  - Supports linguists in discovering linguistic phenomena.

Relevant resources for morphology in NLP 1/

- **Unimorph**: Datasets with exhaustive lists of words in 169 languages with tuples consisting of lemmas, surface words, word segmentation and lexical information (PoS, number, gender, case, etc.)
- **Universal Dependencies**: Corpora with (among other information) words are annotated with the lemma, the PoS and additional lexical information.

Unimorph

Provides type-level annotation, and is more exhaustive (is likelier to cover more words of a language.

lemma	**surface form**	**lex. info**
eat	eats	V;PRS;3;SG
eat	eating	V;V.PTCP;PRS
eat	ate	V;PST
eat	eaten	V;V.PTCP;PST
eat	eats	N;PL

Universal Dependences

Provides **token-level annotation** with tokens in a context.

Form	Lemma	PoS	lex. info
He	he	PRON	PERS-P3SG-NOM Case=Nom...
ate	eat	VERB	PAST Mood=Ind—Tense=Past...
a	a	DET	IND-SG Definite=Ind...
mouthful	mouthful	NOUN	SG-NOM Number=Sing

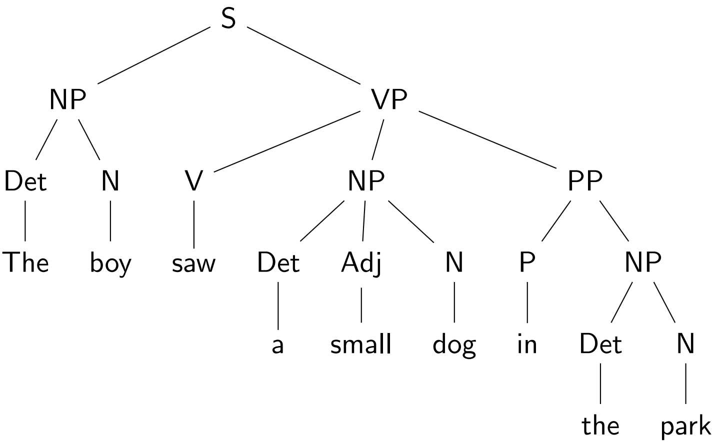
Índex

What is Syntactic Parsing?

- Syntactic parsing is aimed at determining the structure of a sentence.
- It provides representations that help understand relationships between words.
- Two main strategies:
  - Phrase Structure (Constituency Grammar)
  - Dependency Structure (Dependency Grammar)

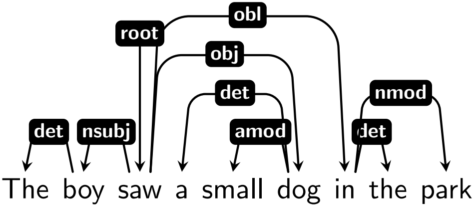
Phrase Structure

- Represents syntax as nested phrases.
- Commonly associated with constituency grammar.



Dependency Structure

- Represents syntax as directed relationships between words.
- Captures dependencies directly.



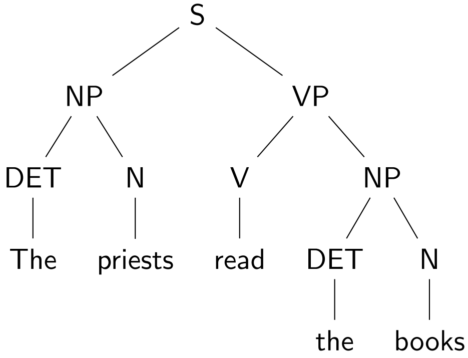
Why Dependency Structure is More Universal

- Dependency grammar focuses on word-to-word relations.
- Easier to apply to languages with different word orders (e.g., SVO, OVS, SOV, etc.).

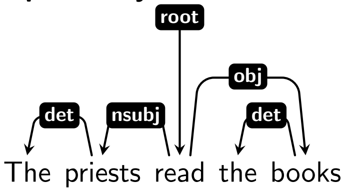
Example: Syntax Analysis in Two Languages 1/

English (SVO): The priests read the books.

Phrase Structure:



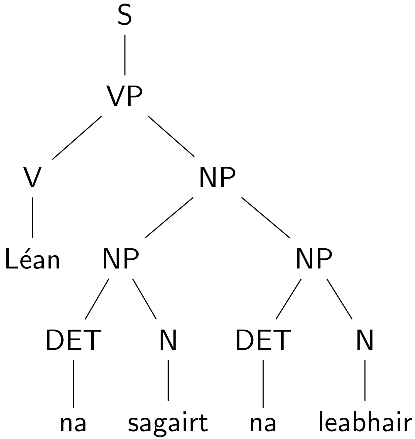
Dependency Structure:



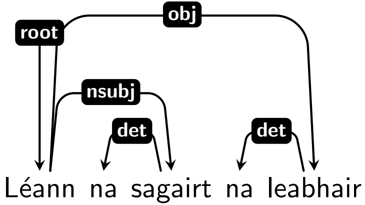
Example: Syntax Analysis in Two Languages 2/

Irish (VSO): Léann na sagairt na leabhair.

Phrase Structure:



Dependency Structure:



Why is important syntax in the age of neural technologies?

Two main uses:

- **In NLP**: mostly useful for low-resource languages
  - Split text into meaningful fragments
  - Disambiguate text
  - Knowledge-enhanced models (summarization, translation, etc.)
  - Helps identify named entities
- **In CL**:
  - Automatic annotation of corpora.
  - Search of specific syntactic structures in corpora.
  - Supports linguists in discovering linguistic phenomena.

Universal Dependences

Annotation of both **morphological information** and **syntactic dependencies**

Form	Lemma	PoS	lex. info	Dep.	Dep. type
You	you	PRON	PERS-P2	3	nsubj
can	can	AUX	PRES-AUX VerbForm=Fin	3	aux
change	change	VERB	INF VerbForm=Inf	0	root
the	the	DET	DEF Definite=Def	6	det
security	security	NOUN	SG-NOM Number=Sing	6	compound
mode	mode	NOUN	SG-NOM Number=Sing	3	obj
...	...	...	...	...	...

What tools can be used for dependency parsing?

Many tools. Some of the most popular ones:

- **Stanza**:
  - Graph-based parsing implemented as a neural network.
  - Multilingual from its inception.
  - More exhaustive, also slower.
- **ScyPy**:
  - Statistical implementation of transition models.
  - Initially focused on English, now it covers wide range of languages.
  - More prone to make errors with long-range language phenomena, but faster.

Índex

Words have meaning

Semantic compositionality principle

The meaning of a **complex expression** (a sentence) is determined by the **meanings of its constituent parts** (words) and **the way they are combined** (syntax).

- **Lexeme**: A unit of meaning in language, independent of inflectional forms.
  - Example: The lexeme *run* covers *runs*, *running*, *ran*.
- **Word Sense**: The specific meaning of a word in a given context.
  - Example: *bank* (financial institution) vs. *bank* (riverbank).

Words are related by their meaning

- **Synonymy**: words mean (almost) the same
- **Antonymy**: words are opposite
- **Similarity**: words share some aspects of their meaning
- **Relatedness**: words belong to the same *semantic field*
- **Connotation**: connotations independent of the meaning (for example, sentiment)

Words are related by their meaning

- **Synonymy**: words mean (almost) the same
  - *big* and *large*
  - yes, but: *my big sister* **!=** *my large sister*
  - **Linguistic principle of contrast**: different form $\rightarrow$ different meaning
- **Antonymy**: words are opposite
- **Similarity**: words share some aspects of their meaning
- **Relatedness**: words belong to the same *semantic field*
- **Connotation**: connotations independent of the meaning (for example, sentiment)

Words are related by their meaning

- **Synonymy**: words mean (almost) the same
- **Antonymy**: words are opposite
  - *big* vs. *small*
  - *up* vs. *down*
- **Similarity**: words share some aspects of their meaning
- **Relatedness**: words belong to the same *semantic field*
- **Connotation**: connotations independent of the meaning (for example, sentiment)

Words are related by their meaning

- **Synonymy**: words mean (almost) the same
- **Antonymy**: words are opposite
- **Similarity**: words share some aspects of their meaning
  - *cow*, *horse* → ruminants, size, etc.
  - *pen*, *pencil* → shape, purpose, etc.
- **Relatedness**: words belong to the same *semantic field*
- **Connotation**: connotations independent of the meaning (for example, sentiment)

Words are related by their meaning

- **Synonymy**: words mean (almost) the same
- **Antonymy**: words are opposite
- **Similarity**: words share some aspects of their meaning
- **Relatedness**: words belong to the same *semantic field*
  - *coffee*, *cup* $\rightarrow$ coffee is served in cups.
  - *car*, *wheel* $\rightarrow$ cars have four wheels.
- **Connotation**: connotations independent of the meaning (for example, sentiment)

Words are related by their meaning

- **Synonymy**: words mean (almost) the same
- **Antonymy**: words are opposite
- **Similarity**: words share some aspects of their meaning
- **Relatedness**: words belong to the same *semantic field*
- **Connotation**: connotations independent of the meaning (for example, sentiment)
  - *reproduce* vs. *plagiarize*
  - *mature* vs. *elderly*.

Introduction to Vector Semantics

- **Vector Semantics**: A method of representing word meanings using vectors in a high-dimensional space.
- Words are represented as points in this space, where:
  - The distance between vectors reflects the semantic similarity of words.
  - Words appearing in similar contexts are closer in the vector space.
- **Core Idea**: *"You shall know a word by the company it keeps"* (Firth, 1957).

Example: Words in the Same Context

Suppose you see these sentences:

- Ong choi is delicious **sautéed with garlic**.
- Ong choi is superb **over rice**.
- Ong choi **leaves** with **salty sauces**.

And you've also seen these:

- ...spinach sautéed with garlic over rice.
- Chard stems and leaves are delicious.
- Collard greens and other salty leafy greens.

Conclusion:

- **Ongchoi** is a leafy green like **spinach**, **chard**, or **collard greens**.
- We could conclude this based on words like **"leaves"** and **"delicious"** and **"sauteed"**

Introduction to Document Representation

- Representing text numerically is essential for machine learning models.
- **Bag-of-Words (BoW)**:
  - Represents documents as a vector of word counts.
  - Ignores grammar, word order, and context.
- **TF-IDF (Term Frequency-Inverse Document Frequency)**:
  - Weights words by their importance in the document and corpus.
  - Reduces the impact of frequent but uninformative words (e.g., "the").

Creating a Bag-of-Words (BoW) Vector

- Given a corpus:
  Document 1: "The cat sat on the mat."
  Document 2: "The dog lay on the rug."
- Vocabulary:

$$\{\text{cat, sat, mat, dog, lay, rug, on, the}\}$$

- Represent each document as a vector of word counts:

$$\text{Document 1: } [1, 1, 1, 0, 0, 0, 1, 2]$$

$$\text{Document 2: } [0, 0, 0, 1, 1, 1, 1, 2]$$

- Rows = documents, columns = word counts.

Creating a TF-IDF Vector

- **Step 1: Term Frequency (TF)**:

$$\text{TF} = \frac{\text{Number of occurrences of the term in the document}}{\text{Total terms in the document}}$$

- **Step 2: Inverse Document Frequency (IDF)**:

$$\text{IDF} = \log \frac{\text{Total number of documents}}{\text{Number of documents containing the term}}$$

- **Step 3: TF-IDF Score**:

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

- Example:
  - Word: "the" (appears in all documents).
  - $\text{IDF} = \log \frac{100}{100} = 0$ (low importance).
  - Word: "cat" (appears in one document).
  - $\text{IDF} = \log \frac{100}{1} = \log 100 = 2$ (higher importance).

Applications and Limitations

Limitations:

- BoW ignores context and word order.
- TF-IDF may give low scores to semantically important words.
- Both methods produce sparse vectors for large vocabularies (large vectors with large vocabulary)

Introduction to Embeddings

- **Embeddings**:
  - Dense vector representations of text in a continuous vector space.
  - Capture semantic and syntactic relationships between words.
- Unlike BoW or TF-IDF:
  - Embeddings are **dense** (low-dimensional) rather than sparse.
  - They are learned automatically from data rather than being based on simple counting or weighting.
- Many existing strategies: Word2Vec, GloVe, Sent2Vec, BERT, etc.

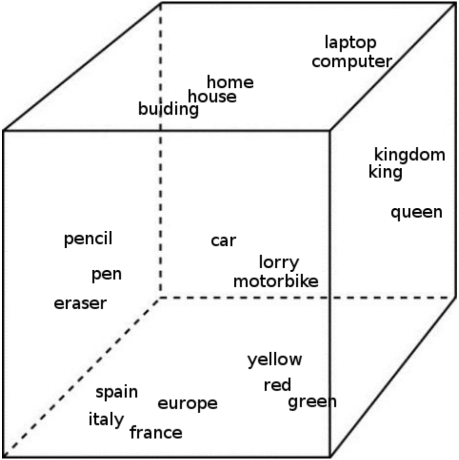
Why Neural Networks for Embeddings?

- Neural networks are excellent for learning embeddings because:
  - They can learn complex, non-linear relationships from large amounts of data.
  - Representations are adjusted to optimize performance on downstream tasks (e.g., classification, translation).
- Key Idea:

  Embeddings are learned during the process of training a neural network on a task.
- Example:
  - In a sentiment analysis task, embeddings capture semantic nuances relevant to predicting sentiment.

Visualization of Word Embeddings

- Semantically-related words (e.g., "pencil," "pen," "eraser") cluster together.
- Words with opposite meanings are farther apart.



Applications of Embeddings

- **Text-based applications**:
  - Sentiment analysis, machine translation, question answering, etc.
- **Non-text applications**:
  - Representing users (e.g., recommender systems).
  - Representing products (e.g., in e-commerce search).
- Advantages of embeddings:
  - Compact and efficient representations.
  - Capture semantic relationships between words or entities.

Embeddings that represent sequences of words

- **General-purpose sentence embeddings**: USE, SBERT, SimCSE.
- **Multilingual tasks**: LASER, Multilingual USE, XLM-RoBERTa.
- **Lightweight options**: MiniLM, DistilBERT.
- **Paragraph embeddings**: Doc2Vec, GPT-based models, T5.

Distance Between Vector Representations

- To compare two vector representations (e.g., for sentences or documents), we compute their **distance** or **similarity**.
- Common measures:
  - **Euclidean Distance**: Measures the straight-line distance between vectors.
  - **Cosine Similarity**: Measures the cosine of the angle between vectors in the vector space.
- Why use cosine similarity?
  - Focuses on orientation, not magnitude.
  - Ideal for high-dimensional and sparse data (e.g., text embeddings).

Computing Cosine Similarity

- **Formula for Cosine Similarity**:

$$\text{cosine similarity} = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

  Where:

  - $\vec{u} \cdot \vec{v}$ = dot product of the two vectors.
  - $\|\vec{u}\|$ and $\|\vec{v}\|$ = magnitudes (norms) of the vectors.

- **Steps**:
  1. Compute the dot product: $\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i$
  2. Compute the magnitudes: $\|\vec{u}\| = \sqrt{\sum_{i=1}^{n} u_i^2}, \|\vec{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$
  3. Divide the dot product by the product of magnitudes.

- **Result**:
  - Value ranges from -1 (opposite) to +1 (identical).
  - A higher value means higher similarity.

Índex

CL vs NLP

- Computational Linguistics (CL) and Natural Language Processing (NLP) are closely related, yet distinct fields.
- CL provides the theoretical foundation, while NLP focuses on practical implementation and real-world applications.
- Both fields are crucial for advancing our understanding of human language and improving human-computer interactions.
- Many of the tasks analyzed today are useful in these two disciplines.

Text preprocessing

- Crucial first step in NLP pipelines, ensuring data is clean, consistent, and ready for analysis.
- Key sub-tasks include removing formatting, tokenization, and normalization, each contributing to better model performance.
- Effective preprocessing requires tailored techniques for different tasks and languages.

Morphological parsing

- Very relevant in CL, as it allows word-level analysis of corpora, and helps to understand and identify linguistic phenomena.
- Especially relevant for low-resourced languages, for which it still plays a relevant role in NLP.
- Essential for handling rich morphology, reducing vocabulary size, and improving NLP models.
- Unimorph and Universal Dependences are two of the most relevant multilingual resources for this task.

Syntactic parsing

- Aimed at understanding the structure and relationships between words in a sentence.
- Both phrase structure and dependency structure provide valuable insights, with dependency structure being more flexible across languages.
- Syntax remains important in NLP, especially for low-resource languages, aiding tasks like disambiguation, summarization, and named entity recognition.
- The use of universal dependencies and parsing tools like Stanza and ScyPy enhances syntactic analysis in a multilingual context.

Vector representations of text

- Words have complex relationships based on meaning: synonymy, antonymy, similarity, relatedness, etc.
- These relationships are fundamental to how we represent and understand language computationally.
- Vector semantics allow to represent word meanings as numeric vectors in high-dimensional spaces.
- Methods like **Bag-of-Words** and **TF-IDF** provide basic yet powerful representations for text; embeddings capture more nuanced semantic relationships.
- Comparing vector representations, such as through **cosine similarity**, enables efficient comparison of texts and supports a wide range of NLP applications.