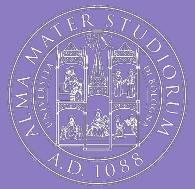


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Intelligenza Artificiale

An introduction to NLP

Ivan Heibi
Dipartimento di Filologia Classica e Italianistica (FICLIT)
Ivan.heibi2@unibo.it

Speech and Language Processing

An Introduction to Natural Language Processing,
Computational Linguistics, and Speech Recognition

Third Edition draft

Daniel Jurafsky
Stanford University

James H. Martin
University of Colorado at Boulder

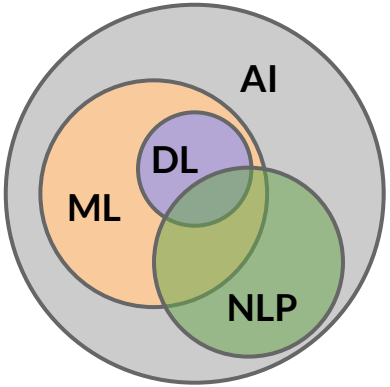
https://web.stanford.edu/~jurafsky/slp3/

Introduction to NLP

People communicate using language, whether it's written or spoken

For an effective interaction between computers and humans, it's crucial for computers to comprehend the natural languages that humans use.

Natural Language Processing (NLP) is dedicated to teaching computers how to learn, process, and manipulate human languages



NLP practices

NLP is used to comprehend the structure and significance of human language through the examination of various components such as syntax, semantics, pragmatics, and morphology.

Macro categories are:

- **Syntactic analysis** (i.e., parsing or syntax analysis): recognizing the syntactic structure within a text and the dependency relationships among words.
- **Semantic analysis:** identifying the meaning of language. (since language is polysemic and ambiguous, semantics is considered one of the most challenging areas in NLP)

Some relevant sub-tasks:

- Word Sense Disambiguation
- Text Classification (e.g., Topic Modeling, Sentiment analysis)
- Named Entity Recognition

NLP benefits

Some of the most relevant benefits of NLP are:

- **Automate processes in real-time**
  Machines learn to sort and route information

- **Perform large-scale analysis**
  Helps machines automatically understand and analyze huge amounts of unstructured text data (e.g., social media comments or online reviews)

- **Providing a more objective analysis**
  Humans are prone to mistakes, while computers provide a more objective analysis. NLP tools can analyze large volumes of text data, with minimal reliance on unnecessary human intervention and with the ability to learn/adjust to your goals.

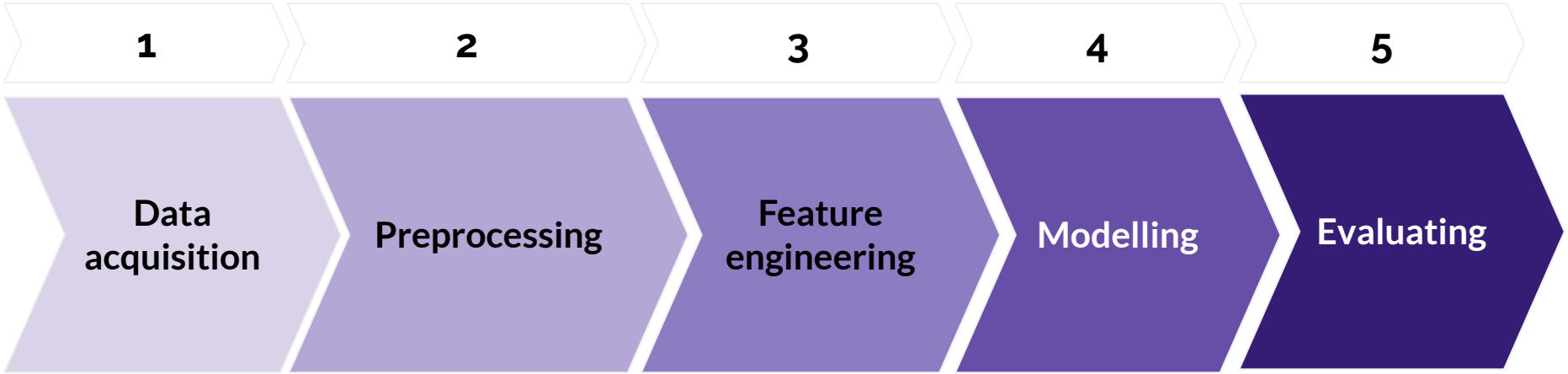
NLP approaches

- **Symbolic Approach**
  Utilizes manually crafted rules and knowledge bases to depict the structure and meaning of language
  - Example: a rule-based machine translation system to translate simple sentences word-for-word using a dictionary.

- **Statistical Approach**
  statistical methods and machine learning systems are used to extract patterns from large amounts of text data
  - Example: analyzing millions of translated sentences and automatically deriving the translation rules.

- **Neural Network Approach**
  ANNs are used to learn representations of language from data.
  - Example: analyzing millions of translated sentences and learning to translate similar sentences without explicit rules.

NLP processing pipeline



Step (1)

Data acquisition

Data acquisition

Gathering data for NLP tasks includes obtaining unprocessed (raw) textual data from diverse sources to establish a comprehensive dataset.

This process entails evaluating the presence and ease of access to the data, determining if it is readily accessible, necessitates additional information, or mandates the creation of content from the beginning.

Two main scenarios:

- Data is ready: data stored in databases or repositories, ready to be gathered
- Less Data: insufficient data volume for a ML training or analysis. Employ data augmentation techniques to enhance the dataset.

Data acquisition: data augmentation and data sources

Data sources

- **Open datasets:** publicly accessible datasets (repositories like Kaggle, UCI Machine Learning Repository, or government databases). Data could be retrieved via APIs
- **Web Scraping:** data gathering from websites or forums by extracting pertinent information.
- **Extracting text from different formats:** retrieve pertinent text from alternative textual formats, such as PDF.

Data augmentation

- **Synonym substitution:** substitute words with their synonyms to introduce variety to the dataset without significantly changing the context.
- **Bigram reversal**: change the order of word sequences by reversing bigrams to generate different combinations.
- **Reverse translation**: translate text into another language and then back to the original language to bring in varied expressions.
- **Introducing Disturbances:** add random noise or perturbations to the data as a means of augmentation.

Step (2)

Preprocessing

Preprocessing: cleaning

- **Stripping HTML:** text from web sources contain HTML tags (used for formatting) these aren't needed for language analysis.
- **Handling special characters:** for instance with Emojis , we can either convert them to text descriptions or remove them completely, depending on their relevance to the analysis.
- **Spell Checking:** if not informative, to ensure consistency and accuracy, spell checking operations are crucial to fix common typos in the text.

Preprocessing: basic operations

- **Tokenization:** text is converted into smaller pieces called tokens. This is done at word level (word tokenization) or sentence level (sentence tokenization) depending on the analysis task.
- **Stop Word Removal**: common words like "the" and "is," known as stop words, are removed. These words don't add much meaning and can clutter the analysis.
- **Stemming/Lemmatization**: words are simplified to their root form. stemming removes prefixes and suffixes, while lemmatization uses a dictionary to find the base form. This helps standardize the text for easier analysis.
- **Lowercasing**: text is converted to lowercase. Since uppercase/lowercase can sometimes affect analysis, this ensures uniformity.
- **Language Detection**: for multilingual content, the language of the text is identified. This helps tailor the analysis to the specific language nuances.

Preprocessing: advanced operations

- **Part-of-Speech (POS) Tagging**: categorizing words in the text into grammatical classes such as nouns, verbs, and adjectives. This operation offers insights into the syntactic structure.
- **Parsing:** examining the grammatical arrangement of the sentences to recognize connections between words and establish the syntactic roles and dependencies.
- **Coreference Resolution:** addressing references within the text by connecting pronouns or noun phrases to their corresponding entities, ensuring a cohesive understanding and analysis.
    - Example: "*The brown dog chased the red ball across the park; it looks exhausted now.*" in this case *“it”* refers to the *“brown dog”*

Step (3)

Feature Engineering

Feature Engineering

Feature engineering in NLP takes raw text, which is like a jumble of words, and transform it into features the machine can understand.

The identified features are characteristics that capture the meaning of the text, how words relate to each other, and the context they appear in. This allows the machine to use the text effectively in tasks like classification or text analysis.

Some of the main techniques that could be adopted:

- Bag of Words (BoW)
- Term Frequency-Inverse Document Frequency (TF-IDF)
- N-Gram Models
- Word Embeddings (Word2Vec, GloVe, FastText)
- …

Bag of Words (BoW)

BoW technique focuses on the individual words themselves, ignoring their order or grammatical structure (i.e., a bag filled with words, completely jumbled up)

Considering we have a collection of documents, here's how it works:

- **Each document is a bag**: cleaning, removing unnecessary elements, and tokenizing the text.
- **Building a word dictionary**: all the unique words across all documents are collected, forming a vocabulary.
- **Creating a word matrix**: creating a table with documents as rows and unique words as columns. Each cell holds the number of times a particular word appears in a corresponding document.

Bag of Words (BoW): example

Document 1: *"The quick brown fox jumps over the lazy dog."*

Document 2: *"The dog is lazy. The fox is quick and brown."*

Using BoW:

- **Vocabulary:** {*the, quick, brown, fox, jumps, over, lazy, dog, is, and*}
- **Matrix:**

	the	*quick*	*brown*	*fox*	*jumps*	*over*	*lazy*	*dog*	*is*	*and*
Document 1	2	1	1	1	1	1	2	1	0	0
Document 2	2	0	1	2	0	0	2	1	2	1

.

Term Frequency-Inverse Document Frequency (TF-IDF)

A popular technique used to weight the importance of words within a document or collection of documents. It helps us understand how relevant a specific word is to a particular document and to the entire corpus (all documents).

Term Frequency (TF):
Measures how often a word appears in a specific document.
The more times a word appears, the higher its TF score.

➔ Example: in a document about "artificial intelligence" the word "machine" might have a higher TF

Inverse Document Frequency (IDF):
Measures how common a word is across all documents in the corpus.
The rarer a word is, the higher its IDF score.

➔ Example: words like "the" or "a" might have a low IDF, as they appear frequently in most documents.

Term Frequency-Inverse Document Frequency (TF-IDF)

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$$

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term}})$$

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

Term Frequency-Inverse Document Frequency (TF-IDF): example

Corpus:

D1	
word	***count***
this	2
is	3
machine	1
learning	1

D2	
word	***count***
is	2
another	2
technique	1
learning	1

word = “is”

- **TF(“is”, D1)** = 3/7 = 0.42
- **TF(“is”, D2)** = 2/6 = 0.33
- **IDF(“is”, Corpus)** = log(2/2) = 0

➔ **TFIDF(“is”, D1, Corpus)** = 0.42 * 0 = 0

➔ **TFIDF(“is”, D2, Corpus)** = 0.33 * 0 = 0

word = “machine”

- **TF(“machine”, D1)** = 1/7 = 0.14
- **TF(“machine”, D2)** = 0/6 = 0
- **IDF(“machine”, Corpus)** = log(2/1) = 0.3

➔ **TFIDF(“machine”, D1, Corpus)** = 0.14 * 0.3 = 0.042

➔ **TFIDF(“machine”, D2, Corpus)** = 0 * 0.3 = 0

N-Gram Models

An N-gram model is used to predict the next word in a sequence, based on the previous (N-1 words). 'N' refers to the number of words considered as context.

N-gram models are built by analyzing a large amount of text data (corpus). The model calculates the probabilities of different word sequences, including how frequently any given word follows another.

Types:

- **Unigrams (N=1**): only one word is considered (no context)
- **Bigrams (N=2)**: two consecutive words are considered
- **Trigrams (N=3)**: three consecutive words are considered

Applications:

- **Text generation**: to assist in text generation, e.g., suggesting next words in autocomplete tasks.
- **Document classification**: to categorizes text based on common language patterns.
- ...

N-Gram Models: example

Building a chatbot aiming to predict the next word in a user's message. The corpus is made by a large collection of previous chat conversations.

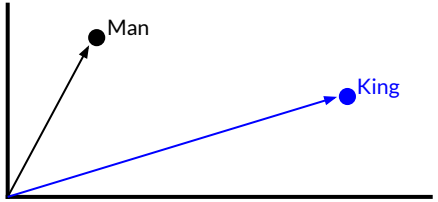
The N-gram Model:

- **Unigrams (N=1)**: words like "the", "a", "is" will likely have high frequencies.
- **Bigrams (N=2)**: The model analyzes frequent word pairs. It may learn that "the quick" is often followed by "brown", "the weather" is often followed by "is".
- **Trigrams (N=3):** The model considers frequent sequences of three words. It may learn that "I am feeling" is frequently followed by "happy", "tired", or "hungry".

Word Embeddings

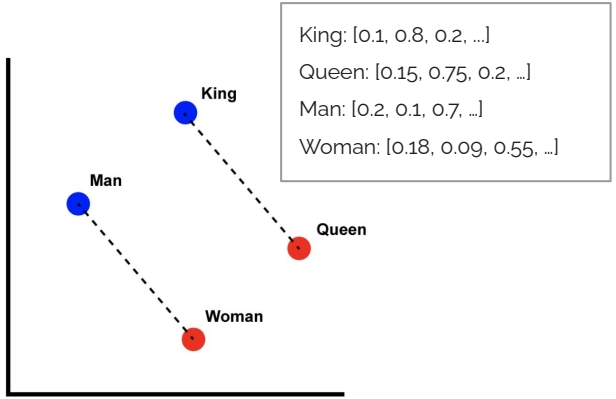
Instead of treating words as isolated entities, these techniques transform words and phrases into numerical representations, i.e. a dense vector representations in a continuous vector space (word2vec).

➔ These vectors exist in a multidimensional space (vector space) such that similar words are positioned closer together.



Applications:

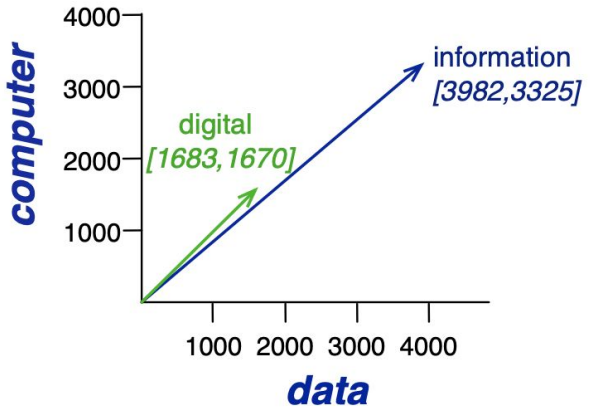
- **Finding word analogies**: Identifying words that share a similar relationship, like "king" is to "queen" as "man" is to "woman".
- **Discovering semantic similarities**: Determining how closely related words are in meaning, even if they don't directly appear together.



Word Embeddings

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.



Word2Vec

Scenario: a small corpus of text containing only three sentences:

- "The cat sat on the mat."
- "The dog chased the cat."
- "The dog wagged its tail."

Each word is a vector of 3 elements, with random values initially:

Word	Initial Vector
the	[0.1, 0.2, 0.3]
cat	[0.4, 0.5, 0.6]
sat	[0.7, 0.8, 0.9]
...	[x1,x2,x3]

Training:

- Iterate through the sentences, considering each word and its surrounding context.
- For each word, we predict the surrounding words based on its current vector representation.
- We compare the predicted probabilities with the actual occurrences of the surrounding words in the corpus.
- Based on the comparison (error), we update the word's vector slightly to improve the prediction accuracy for the surrounding words in the next iteration.
- This process continues through all sentences and words in the corpus for multiple iterations until the vectors converge and achieve a stable representation.

Word Embeddings: example

Building a recommendation system for a music streaming service. Users can search for songs through a free-text queries (e.g. by artist, genre, or even lyrics)

Application:

- **Training the Model:** the system is on a large dataset of music information, including song titles, artist names, and lyrics. This data is used to create word embeddings for these terms.
- **Understanding User Queries:** when a user types "happy songs," using word embeddings, the system recognizes the semantic similarity between "happy" and other words like "joyful," "upbeat," or "cheerful."
- **Finding Relevant Recommendations**: using word embeddings, the system identifies songs with similar semantic features in their titles, lyrics, or artist information. It suggests songs that might be considered "happy" even if the exact word "happy" isn't present.

Step (4)

Modelling

Modelling

Two main approaches:

- Machine Learning (ML)
- Deep Learning (DL)

Machine learning (ML) applications engage in feature engineering by leveraging domain expertise to manually craft pertinent inputs for the models, aligning them with the problem domain.

Deep learning (DL) applications depend more on automated feature learning, enabling models to discern complex patterns from raw data. This diminishes the direct reliance on manual feature engineering and, to some extent, domain-specific inputs.

Machine Learning: Support Vector Machines (SVM)

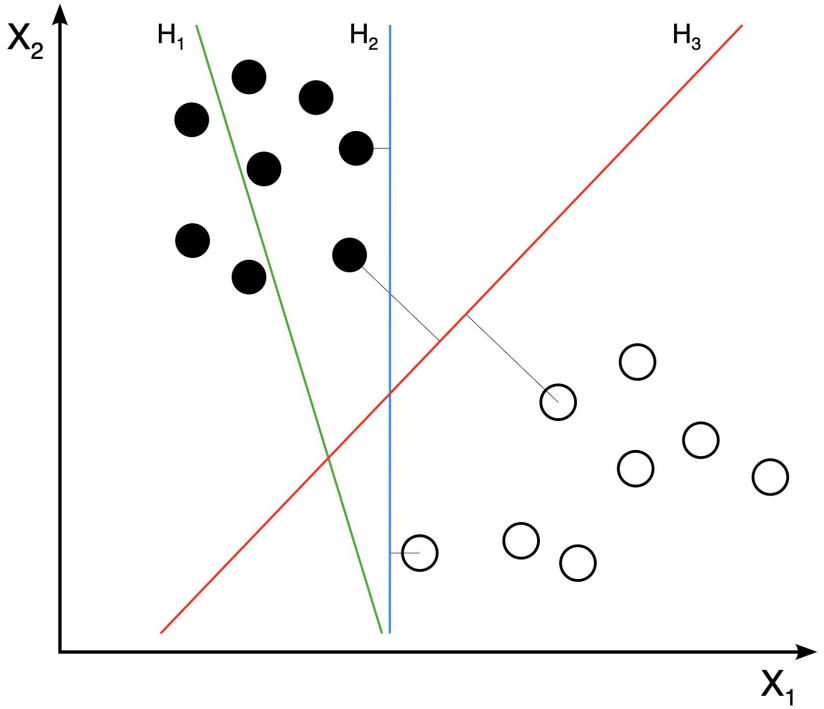
SVMs offer a powerful approach to text classification by finding the optimal separation between categories in a high-dimensional space. Is based on:

1. **Text preprocessing:** text data is converted into numerical features, (e.g., tfidf, bag-of-words)
2. **High-dimensional space:** these features act as coordinates, placing each document in a high-dimensional space where similar documents are positioned closer together.

SVM seeks to identify a hyperplane, which is like a dividing line, that separates the different classes (e.g., "spam" and "not spam").

The SVM aims to maximize the margin, which is the distance between the hyperplane and the closest data points from each class (called support vectors). These support vectors are crucial for defining the decision boundary.

Machine Learning: Support Vector Machines (SVM)



- H_1 does not separate the classes.
- H_2 does, but only with a small margin.
- H_3 separates them with the maximal margin.

source: https://en.wikipedia.org/wiki/Support_vector_machine#/media/File:Svm_separating_hyperplanes_(SVG).svg

Support Vector Machines SVM): example

A system to automatically filter spam emails. A large training dataset containing both spam and non-spam emails is provided.

1. **Data Preprocessing and feature engineering:**
   a. BagOfWords: counting the frequency of each word in the email.
   b. TF-IDF: assigning weights to words based on their importance within the email and the entire dataset.
   c. Other features engineering techniques (e.g., certain keywords or URLs).
2. **Training the SVM:** learning to distinguish between spam and non-spam email features. (identifies the hyperplane that best separates the two classes in the high-dimensional feature space).
3. **Classifying New Emails:** new email features are extracted and positioned in the same high-dimensional space.

Step (5)

Evaluating

Evaluating

Assessment/Evaluating within the NLP pipeline is crucial, involving **intrinsic** and **extrinsic** evaluations to thoroughly measure model performance from both technical and practical perspectives.

- **Intrinsic Evaluation**
  Focuses on assessing the technical aspects and capabilities of the model in isolation, without considering its real-world application.
- **Extrinsic evaluation**
  Measures the model's performance in real-world applications (e.g. business contexts), considering its impact and utility in practical scenarios.

NLP today

Large Language Models (LLMs)

Large language models (LLMs) are massive neural networks trained on enormous amounts of text data (i.e., hundreds of billions of words). ChatGPT and Gemini are Large language models.

LLMs can comprehend long-term dependencies, intricate relationships among words, and subtleties inherent in natural language. LLMs have the capability to process all words simultaneously, leading to accelerated training and inference.

Transformer models (Deep learning technique) is used during the modelling step (i.e. step 4) of the NLP pipeline

Transformers are specifically designed to capture relationships in sequential data, relying on a self-attention mechanism to grasp global dependencies between input and output.

Transformers: How does it work?

Pretend we have the sentence: ***"The dog chased the cat"***

- **Self-attention:** for each word It examines all the other words in the sentence to determine which words are most relevant. To create a richer, contextual representations for each word.
    - Example: when focusing on "chased", it might assign higher weights to "dog" (what's doing the chasing) and "cat" (what's being chased).

- **Multi-head attention:** transformers have multiple attention heads running in parallel. Each head can focus on different parts of a sentence or different relationships between words. This allows them to capture even more complex patterns.

- **Positional encoding:** transformers don't process words sequentially (unlike traditional recurrent models). To keep track of word order, positional encoding injects information about a word's position in the sequence into its representation.

- **Feed-forward networks:** each word's representation (self and multi) passes through a feed-forward neural network for further processing. This adds non-linearity and helps refine the representations.