

## Part I

# Natural Language Processing

## 1 Introduction

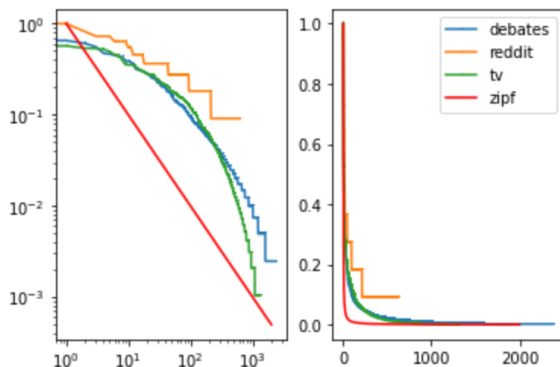
### Challenges

- Complicated structure in sentence
- Syntactic ambiguities (“*time flies like an arrow*”, “*get the cat with the gloves*”).
- Metaphores, humor, irony, ...
- Semantics can be very rich and dependent on context, not easy to distinguish
- Language requires knowledge about the world
- Hard to really formalise the notion of “meaning”

**Basic approach** Gather information on word based on its context. Given a large text corpus, we can assume this to be meaningful statistics.

**Zipf's Law** The number of elements with a given frequency follows a power law distribution. That is, there is a small number of elements which appear very often and the majority of elements appears rarely. In its most simple form, the probability of the  $n$ -th most common word  $p(n)$  is

$$p(n) = 1/n$$



**Elements of language** We can have different points of view on language:

- Phonetics (sound)
- Grammar

- Phonology
- Morphology
- Syntax
- Semantics (meaning)

**Morphology** How words are built up from smaller meaningful units, for instance *un-lady-like*, *dog-s*

- **Inflection** – variation in the form of a word (usually affix) that expresses a grammatical contrast
  - adds tense, number, person, mood, aspect, etc
  - e.g. *run* → *run* — *running*
  - does not change word class
- **Derivation** – formation of a new word from another
  - e.g. nominalization (*computer* → *computerization*)
  - e.g. formation of adjectives (*computational*, *clueless*)
  - changes word class

### Morphemes

- **Root** – equivalence class of a word when all affixes are removed; not further decomposable into meaningful elements.
- **Stem** – part of word that never changes when morphologically inflected, *i.e. without affixes describing tense, number, person, ...*
- **Lemma** – Base form of word
- From *produced*, lemma is *produce* but stem is *produc*

## 2 Tokenization

**Token** an individual occurrence of a word (as opposed to a vocabulary/dictionary item)

### Challenges

- Keep abbreviations, dates, numbers as single tokens
- distinguish abbreviations from words
- names and phrases (*queen of england*, *TU Wien*)
- compound words
- apostrophes, umlauts, etc and other linguistic characteristics
- encoding issues like RTL/LTR

**Maximum Matching algorithm** Use a dictionary of known terms. Take the longest prefix of the input string that matches a dictionary item. Does not always make sense (“*Theta bled own there*”).

### 3 Stemming

**Stemming/Lemmatization** reduce tokens to equivalence classes. Usually to gather words that are morphologically different but semantically quite similar to the same set, i.e. to improve comparability.

**Porter Stemmer** Rules for stripping suffixes. Applicability of rules is based on *measure* of a word  $w$ , which is the number  $m$  s.t.  $w = C(VC)\{m\}V$  where  $C, V$  are arbitrary sequences of consonants, vowels, resp. — Indeed reduces the words to their *stems*.

#### WordNet MORPHY

- Has a sophisticated set of rules about inflections
- exception list
- Checks the result of transformation against an extensive dictionary

**Note** MORPHY reduces to *lemmas*, while PORTER reduces to *stems*.

- **Over-stemming** Two words are reduced to the same root when they should not be
- **Under-stemming** Should be reduced to the same root but are not.

### 4 POS-Tagging

Given some input text and some tags (usually word types such as *noun*, *verb*, etc.), want to assign tags to tokens. (*Sequence classification problem*).

Tagging can help with other procedures such as stemming, NER, parsing, ...

**Def.** Can divide words into two different classes

- **Closed class** — can enumerate all members, e.g. determiners, pronouns, prepositions, ...
- **Open class** — don't know all members, e.g. nouns, verbs, adjectives, ...

#### Note

- A single term (dictionary entry) can have different optimal POS-tags depending on its context.
- Tagging helps to resolve ambiguities that exist on term-level (e.g. *leaves* as NN or as VB)

- Tagging removes unnecessary distinctions e.g. all personal pronouns are PRP, determiners
- Naive method (assigning most frequent tag in training data to term) already has 90% accuracy.

#### Def.

- **Informativeness** — Assignment of tag adds information, reduces ambiguity
- **Specifiability** — Ease of mapping a term to a tag
- **Example:** Collapsing multiple related tags into one decreases informativeness, decreases specifiability.

**Feature selection** Can look at word-local features (term, pre-, suffixes, capitalization); but very often the tag of a word depends on its context in the sentence.

#### Main techniques

- **Probabilistic tagging** — consider lexical frequencies of tag in training data – good when large training corpora are available
- **Rule-based tagging** — use rules based on linguistic understanding – good to tailor solution to very specific problems

#### 4.1 Probabilistic tagging

Consider the definition of *conditional probability*

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

This gives rise to the *chain rule*

$$\Rightarrow P(A, B) = P(A|B) \cdot P(B)$$

or, more generally

$$P(w_1, \dots, w_n) = \prod P(w_i | w_1, \dots, w_{i-1})$$

The problem here is that we cannot realistically obtain all the components of the product because there are way too many possible sequences of words. Instead, we employ the *Markov assumption* that says that we can estimate the probabilities by only considering only the  $k$  preceding terms

$$P(w_i | w_1, \dots, w_{i-1}) \approx P(w_i | w_{i-k}, \dots, w_{i-1})$$

For  $k = 1$ , this yields the *unigram model*, for  $k = 2$  the *bigram model* (i.e.  $P(w_i | w_{i-1})$ ).

$n$ -gram modelling is insufficient because language has *long-distance dependencies*.

**Unigram Tagger** Assume that a unigram model generates the current tagging.

Assign a token  $w$  its most frequent tag, i.e.

$$t(w) := \operatorname{argmax}_t P(t \mid w)$$

**Improvement:** Use Bayes' formula, i.e.  $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ , omitting the quotient:

$$t(w) := P(t \mid w) = \operatorname{argmax}_t P(t) \cdot P(w \mid t)$$

**TODO** example

**$n$ -gram tagger** Use information about the previous  $n$  tokens in addition to information about current token. Can have **word-based** and **tag-based** (tags are more common, training data covers more ground).

Assume a bigram language model (generating the sequence of POS-tags).

Pick the tag  $t_i$  for word  $w_i$  that maximises

$$P(t_i \mid t_{i-1}) \cdot P(w_i \mid t_{i-1})$$

For finding  $P(t_i \mid t_{i-1})$ , use the **Maximum Likelihood Estimate** (where  $c$  is the count of observations)

$$P(t_i \mid t_{i-1}) \approx \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

*(Use start and end symbols to be able to calculate probs for first and last words)*

**TODO** example

## 4.2 Rule-based tagging

Try to incorporate linguistic insight.

**Brill tagger**

1. Tag each word using a **baseline tagger** (e.g. unigram tagger, i.e. most common tag)
2. Apply patches that improve the result
  - e.g. if one of the two preceding words is a determiner, change the tag from verb to noun
  - Based on training data, compute the error between any two should-be/is-assigned:  $(t_a, t_b, freq)$ .
  - For each error triple, apply the patch that results in the greatest improvement, apply it.
  - Repeat until no further improvement is possible.

## 5 Parsing

We try to determine the grammatical structure of a sentence.

**Parsing problem** Identify the parse tree (w.r.t some grammar) for a given sentence. *(There may be multiple parse trees, or none.)*

**Parse tree** depicts derivation of sentence beginning from start symbol. Obviously requires context-free grammar *(each set of children has exactly one parent)*

**Motivation**

- Grammar checking
- Question answering
- Machine translation
- ...

### 5.1 Constituency parsing

**Basic assumption** Language is made up of **constituents**, i.e. basic, nested building blocks *(terminals and nonterminals of a formal grammar.)*

**Leftmost derivation** Always expand the leftmost expandable nonterminal.

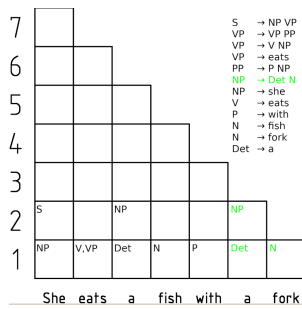
**Shift-reduce parser** *(bottom-up approach)* Push tokens onto stack until top of stack matches the *rhs* of a rule, then reduce (on stack). Accepts the word if start symbol alone is left on the stack.

**Chomsky normal form** The *rhs* of a rule is of one of these shapes

- $A \rightarrow BC$
- $A \rightarrow a$
- $S \rightarrow \varepsilon$

**CYK Parser** *(dynamic programming, bottom-up)* Assumes a grammar in CNF, i.e. *rhs* of rules is two nonterminals or a terminal. From the bottom up, in a dynamic programming fashion, remember if two subcells are the *rhs* of a production rule. The topmost cell produces the entire sentence. Note that this finds all possible parse trees.

# Similarity Measures



## 5.2 Statistical Parsing

*Motivation: Sentences can have many different parse trees but some are clearly more likely than others.*

### Basic idea

- Associate rules of a grammar with probabilities
- The probability of a parse tree is the product of all used productions/derivations.
- Probability of a sentence is the sum of all possible parse tree probabilities.
- Probabilities can be learned from a training corpus.
- Can use probabilities in parsing algorithms such as the CYK parser to find the most likely parse tree.

**Lexicalised Parsing** Extend production rules to be specific to terms (e.g.  $VP(ate) \rightarrow VP(ate) PP(with)$ ). Then, again, assign/learn probabilities. **TODO** Head of sentence

## 5.3 Dependency Parsing

**Basic concept** Dependencies (in the linguistic sense) between tokens (such as *determiner*, *subject*, ...), these form a tree. Again, want to find the most likely parse/dependency structure

**Constraint-based Parsing** Come up with rules describing what a dependency can possibly look like. Begin with a complete graph of pairwise dependencies, then iteratively eliminate dependencies according to rules.

**Global linear models** Assign a *local feature vector*  $g(x, h, m)$  to a specific dependency  $(h, m)$  in a specific sentence  $x$ . Try to maximise the (weighted) sum of all dependencies in the sentence, i.e. the best parse  $y = F(x)$  is given by

$$F(X) = \operatorname{argmax}_y \vec{w} \cdot \sum_{(h,m) \in y} g(x, h, m)$$



# Opinion Mining & Information Extraction

## 6 Information Extraction

to extract • the entities and • relationships between such entities (i.e. clear, factual information)

### 6.1 Named Entity Extraction

used for • summarizing text • answering questions • integrating into knowledge bases • associating information (e.g. sentiments) to sentiments (e.g. of parts of printer in question)

Possible types of entities • location • time • person • ...

**Supervised learning models** Based on labelled training sequences (of tokens), train a classifier to predict labels (*Sequence Labelling Problem*)

- new data must fit training data
- time-consuming

To make this easier, use features that go beyond single tokens, e.g. context window of  $k$  words.

**Sequence Labelling** • reminiscent to POS-tagging • assuming that label is dependent on context. Typical models are • Markov models • Conditional Random Fields • Bidirectional LSTMs

Once we have identified the entities, we'd like to find relationships between them (e.g. triples of operators *is-a*, *daughter-of*, ...) (*Can save these triples in a knowledge base e.g. for question-answering; cf RDF-triples*).

**Relationship Extraction** Try to find type of relationship between two entities. Possibilities

- Extract RDF triples from large corpora like Wikipedia
- Use (specialised) ontologies / knowledge bases (for ex. medical applications)

Methods to extract information:

- handwritten rules – e.g. “*Y such as X*”, “*X, especially Y*”, “*X, including Y*” all express an *is-a-relationship*. – there can be more specific relations that only make sense between certain types of entities (e.g. *cures(drug, disease)*) – pros: • precise •

can be tailored to specific domains – cons: • low recall • high effort

- supervised,
- unsupervised machine learning.

semi-supervised learning: extract less common patterns based on training corpus?

# Question Answering & Text Summarization