Introduction to text representation and processing

Matthieu Labeau - Associate Professor, Telecom Paris, IPP

APM_4AI12_TP: Introduction to Text Processing - 21st February 2025

Outline

- **Course presentation**
- **Introduction to text processing**
- **Representing sentences and documents**
- **Comparing text**
- **Classifying documents**
- **Using context**

Language Processing: goals

Speech and Language Processing (Jurafsky and Manning) (Chapter I)

Interdisciplinary field, whose goal is to get computers to perform useful tasks [..] like enabling human-machine communication, improving human-human communication, or simply doing useful processing of text or speech.

Applications ?

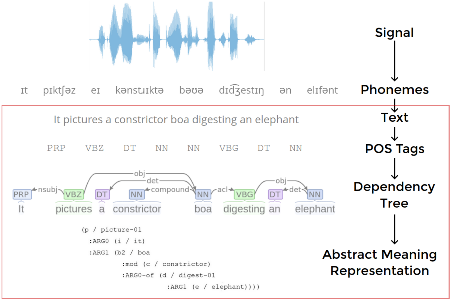
- Any task dealing with **natural language** as **textual data**
  - Can be *edited text*, *user-generated content*, *speech transcriptions*. . .
- Could be roughly divided in three categories:
  - **Text understanding and analysis**
    - Classification and prediction tasks
  - Text generation and transformation
  - Human-machine interaction

Text analysis tasks

Classification tasks:

- At the **document** level: Sentiment analysis, Intent classification, Topic classification, Spam detection..
- At the **word** level: Semantic role labeling, word sense disambiguation, named entity detection

The classic Natural Language Processing Pipeline: to extract **information** and **structure** from text



Statistical Language Processing

Historically, two paradigms: **symbolic** and **stochastic**

- On the symbolic side, formal systems based on logic and *grammars*
- Probabilistic models used early for tasks like *optical character recognition* regained popularity towards the end of the 80s, with models like **Hidden Markov Models** (HMMs), also applied to *speech recognition*
- Probablistic **data-driven** models then rapidly became standard
- Statistical models took over rapidly from 2000, thanks to:
  - A large amount of material and resources (+computational)
  - Efficiency of statistical learning (+unsupervised approaches)
  - Community effort: shared tasks, evaluation campaigns
- .. and now soon represented the state-of-the-art for almost any task; now, **deep learning**

In this course

A fast introduction to *pre-deep learning* statistical language processing focused on methods for **classical NLP tasks** on text:

- What challenges do we encounter when trying to represent and process textual data ?
  - How did *pre-deep learning* methods deal with them ?
- How to exploit context and work on *sequences* efficiently ?
- How to understand and leverage *large amount of unlabeled text data* ?

- We will almost *not talk about neural approaches*
- We will *stay away from language models and generating text*

We will work on text classification and analysis tasks:

- Labs: Python with Scikit-learn
  - Language processing libraries: NLTK and Gensim

Schedule and instructions

- 21/02: **Introduction to text representation and classification** (Matthieu Labeau)
- 07/03: **Hidden Markov Models** (Laurence Likforman)
- 14/03: Lab: Hidden Markov Models
- 28/03: **Structured prediction for natural language processing** (Maria Boritchev)
- 04/04: **Unsupervised models and distributed representations of text** (Matthieu Labeau)
- 11/04: Lab: Embeddings for classification tasks
- 18/04: Exam (3h, no documents)

- Slides and references to further content on e-campus
- Exam based on class questions, small application exercises and what was done in the labs (50% of the final grade)
- Both labs are graded (25% of the final grade each). They **should be submitted at the end of the class**

Introduction to text processing

What are linguistics interested in ?

Different level of **linguistic analysis**:

- **Morphology**: how words are built from *morphemes*
  - *Inflectional* morphology (*dog* → *dogs*, *play* → *played*)
    - **Variation**: same word, but modifies tense, number, degree
  - *Derivational* morphology (*happy* → *happiness*, *teach* → *teacher*)
    - **Formation**: changes meaning and often, grammatical category

- **Syntax**: how words are organised
  - Can be encoded in various ways (*constituency trees, dependency trees*) with their own pros and cons

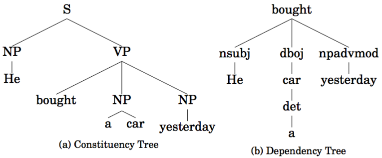


Fig. 3: Examples of the results of constituency and dependency parsing.

What are linguistics interested in ?

- **Semantics**: how words and sentences carry a meaning
  - Lexical semantics: meaning relationships between words (*synonymy, antonymy, hypernymy*)
  - **Compositionality**:
    - Dealing with idioms: *kick the bucket*
    - Polysemy: *apple*
  - **Pragmatics**: how context influences meaning
    - Meaning depends on speaker intent, situation, and shared knowledge (implicit, social context)
    - Examples:
      Can you close the window ?
      It's cold in here...
      Oh, great, air conditionning in winter !

Processing textual data

- A text is a **sequence of characters** = a *string*
  - Letters, symbols, punctuation, separators (but not always !)

- In NLP, decompose text at different levels
  - **Words**: minimal units
  - **Sentences**: processed independantly from each other, as sequences of words

First steps:

- How to **segment** texts into sentences, words ?
- How to **represent** those sentences and those words ?

$\rightarrow$ Depends on the information (*features, structure*) we hope to capture

Corpora and resources

Data-driven methods are based on corpora, which have many distinct properties:

- Language (7000+) and varieties, code switching...
- Genre (news, scientific, fiction..), specific domain (medical, law...)
- Source and authors: how was it written ? Collected ? Why ?
- How representative is the sample of data you are using ?

Resources are unevenly distributed along languages ! They may include:

- Labeled data for many tasks - will allow supervised learning
  - Careful: *annotation* is a difficult and subjective process
- Lexical Databases like **WordNet** (but also for other languages)

Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science (Bender and Friedman, 2018)

Use data statement to avoid biases !
→ Dataset rationale, data provenance, annotator demographic, ...

Basic text processing

What is the bare minimum to process text ?

→ Example of ELIZA

Speech and Language Processing (Jurafsky and Manning) (Chapter II)

```
User:   I am unhappy.
ELIZA:  DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
User:   I need some help, that much seems certain.
ELIZA:  WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
User:   Perhaps I could learn to get along with my mother.
ELIZA:  TELL ME MORE ABOUT YOUR FAMILY
User:   My mother takes care of me.
ELIZA:  WHO ELSE IN YOU FAMILY TAKES CARE OF YOU
User:   My father.
ELIZA:  YOUR FATHER
User:   You are like my father in some ways.
```

Weizenbaum (1966)

- Uses pattern matching to recognize phrases
- Uses a set of rules to translate them into suitable outputs
    - Remarkably successful !

- For pattern matching: regular expressions
- Used then for text *tokenization* and *normalization*

Regular expressions

An algebraic notation for characterizing a set of string - used practically everywhere → Can be used to find desired occurences of words in a large text

Example: looking for 'the':

- Will miss some occurences: `the`
- Will find undesired ones: `[tT]he`
- Will probably have very few false positives or negatives: `[^a-zA-Z][tT]he[^a-zA-Z]`

→ Can be used to capture and substitute text

- Replacing 'the' with 'The': `s/the/The`
- Capturing any string ending with 'er': `/(.*)er/`
- Getting a superlative: `s/the (.*)er/the (\1)est/`

Regular expressions and ELIZA

Regular expressions also allows for more complex fonctionalities.. but especially, allows for ELIZA:

- Early NLP system that imitated a Rogerian psychotherapist
- Has a series or cascade of regular expression substitutions, matching and changing some part of the input lines:
  - Input lines are uppercased
  - Change all instances of MY to YOUR, I'M to YOU ARE, etc...
  - Then, other patterns are replaced:

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Segmentation: Token and types

What counts as a word ? No unique definition:

- Approximately, words can be defined differently for each linguistic level (phonological, morphological, syntactical, semantic)
- We are interested in the *graphic* word, called **tokens**

- **Tokens** are instances of words in the running text
  - Usually based on arbitrary conventions (rules) *with no ambiguity*
- **Word types** are the number of distinct tokens in a corpus
  - Grouped in a set, the **vocabulary** = *words a model knows*

- First step: segmenting text into word tokens. This is **tokenization** !
- Second step: creating the vocabulary

Tokenization

- Simplest approach: **space-based** - segments along separators like whitespaces
  - Does not work with some languages
  - What to do with punctuation ? Example: *'But they answered: "Frighten? Why would anyone be frightened by a hat?" My drawing was not a picture of a hat.'*
  - Issues with specific texts (emojis, hashtags)

- Traditionnal approach: **rule-based**
  - Separate sequences containing neither separators nor punctuation, and punctuation marks
  - Tools: use packages like NLTK (https://www.nltk.org/)

- Recently popularized, **data-driven tokenization** (*Byte Pair Encoding*, *Wordpiece*) based on subwords
  - Learns a vocabulary of tokens; segment using that vocabulary
  - Specific designs for difficult cases (scripts with no separators)

Word Normalization

We will want to choose a standard format for words:

- What to do with upper-cases ?
  $\rightarrow$ depends on the task ! Information retrieval does not need them - but other tasks do

- **Lemmatization**: represent words as their lemma
  - Done by morphological parsing
  - Tools - we can also use **NLTK**
- Other process: **stemming** - crudely cutting words

- What would be the advantages of the various strategies for text normalization ?
  - How many words in the vocabulary ?
    - Collapse together different forms (*sit, sat, sits*)
  - Necessary for some languages with rich morphology: e.g, Turkish.

Segmentation: what is a sentence ?

The format will **heavily depend on the target task**

- For many classification tasks, labels are at a higher level
    - **Document**: possibly many sentences

- For core NLP tasks, it's more complicated
    - Ideally, it:
        - has a **complete syntactic structure**
        - is **semantically related to other sentences**

    - In practice, it is *typograhically marked*, but:
        - The full stop may be ambiguous, the uppercase too
        - Difficulty with embedded sentences (e.g, with quotes)
        - No clear markers in some languages

Representing documents

Bag-of-words representations

In order to classify a document, we use lexical features. The simplest way to represent a document is as a **bag-of-words**

I walked down the street
I walk down the the avenue
I walked down the avenue
I ran down the street
I walk down the city

Compute the **vocabulary** !

- Again: Which tokenization ? How many words ?
- How to store it ?

$\rightarrow \mathcal{V} = \{\text{I}, \text{the}, \text{down}, \text{walked}, \text{street}, \text{avenue}, \text{walk}, \text{ran}, \text{city}\}$

- Next, count !

Bag-of-words representations

The bag-of-words contains frequency counts, **unordered**: they are simple *lexical features*

I walked down the street
I walk down the the avenue
I walked down the avenue
I ran down the street
I walk down the city

	I	the	down	walked	street	avenue	walk	ran	city
D_1	1	1	1	1	1	0	0	0	0
D_2	1	2	1	0	0	1	1	0	0
D_3	1	1	1	1	0	1	0	0	0
D_4	1	1	1	0	1	0	0	1	0
D_5	1	1	1	0	0	0	1	0	1

Document representation: motivation

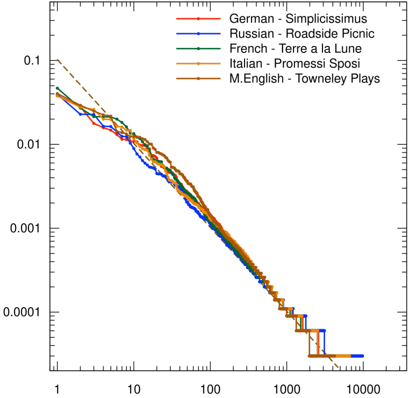
- Assumes that position does not matter
  - Indifferent to *syntax* and *semantics* $\rightarrow$ purely **lexical**
- Used as **features** representing documents to perform various tasks

- **Retrieval** - or any task based on similarity (~distance) between documents
  - Document clustering, information retrieval
  - Intuition: search based on the number of common words

- **Text classification** (sentiment, spam . . .) !
  - We can learn a classifier model that will use word frequencies as features
    - Naïve Bayes: assuming independance between words
  - Or combine them with lexical knowledge and rules:
    - Example: with *SentiWordNet*, each 'word' is associated to a positive and negative score

Issue #1: Sparsity

- Tied to **Zipf's Law**: $frequency \propto 1/rank$



- Zipf's law seems to hold for most natural languages and many language-related phenomena
  - Examples: *meaning-frequency law*, *law of abbreviation*

Smoothing

What issues may frequent words cause ?

- **Too much weight** in models when they are not necessarily significant
  - We can keep a list of frequent but useless *stopwords*
- Makes the useful features **harder to extract**

$\rightarrow$ Use a function to *smooth* counts (for example, $\log$)

What issues may rare words cause ?

- Issues when **normalizing** counts into frequencies. We get *NaN* with
  - Word present at training time but not in the test set
  - Document composed of new words at test time
- Issues when applying smoothing transformation to counts

$\rightarrow$ To avoid *counts being zero*, we simply **add one to every possible count**; this is also called **Laplace Smoothing**

TF-IDF Representations

What would be a better way than directly removing frequent-but-not-significant words (stopwords) ?

- Idea 1: instead of using count $c(w,d)$ of word w in document d, **use smoothing** and define term frequency as TF$(w,d) = \log_{10}(c(w,d)+1)$

- Idea 2: give higher weight to words that occur in only a few documents, using their inverse document frequency. Noting $cd(w)$ the count of documents w appears in and N the total number of documents,

$$\text{IDF}(w) = \log_{10}\left(\frac{N}{cd(w)}\right)$$

The weight given to word w in document d is TF$(w,d) \times$ IDF(w)

- What happens if a word is present in every document ?

Comparing text

Similarity between documents: comparing texts

Goals:

- Any task requiring distance measure (e.g, *clustering*)
- *Information retrieval*: find most relevant documents in a set given a query

For now, focus on **lexical** features only:

- Texts can be compared using their *strings*
  - **Minimum Edit Distance**

- Texts can be compared using their vector representations
  - Comparison is made in **vector space**
  - Choose appropriate space and similarity measure
    - *Cosine* distance, *Jaccard* distance
    - On bag-of-words = **word overlap**

Minimum Edit Distance

- Edit distance: the minimum edit distance (MED) between two strings is the minimum number of editing operations - *Insertion, Deletion, Substitution* - needed to transform one into the other.
- Example:

Speech and Language Processing (Jurafsky and Manning) (Chapter II)

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

- The distance is an *alignement*: here, each operation has a cost of 1 - total cost of 5. (Adapting costs: **Levenshtein** distance)
- Finding this distance is searching for a *path of edits*, the less costly one, in a huge space - too large for a naïve search.

MED with Dynamic Programming

Noting a string **X** of length n, and a string **Y** of length m, we define:

- The edit distance $D(i, j)$ between **X**$[1..i]$ and **Y**$[1..j]$
- The goal is to minimize $D(n, m)$

We will use dynamic programming (solving problems by combining solutions to subproblems): here, **bottom-up**.

- **Initial steps**: we compute $D(i, j)$ for small $i = 0$ $j = 0$;
- **Recurrence**: we compute $D(i, j)$ as a function of smaller distances: iteratively, $\forall i, j$ such that $0 < i < n$ and $0 < j < m$!

MED with Dynamic Programming

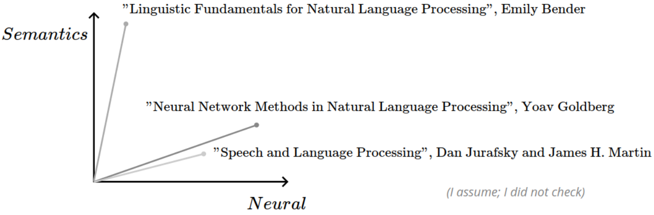
Algorithm:

- Initialize: $D(i, 0) = i$ and $D(0, j) = j$
- Recurrence:
  - For i from 1 to n:
    - For j from 1 to m:

$$D(i,j) = \min \begin{cases} D(i-1,j) & +1 \\ D(i,j-1) & +1 \\ D(i-1,j-1) & + \begin{cases} 2 & \text{if} \quad X[i] \neq Y[j] \\ 0 & \text{if} \quad X[i] = Y[j] \end{cases} \end{cases}$$

- This computes the shortest distance, but we need to keep track of the alignement used to obtain it: this is **backtracking**
- We get an optimal alignment composed of *optimal subalignments*
- We may want to change weights, *e.g* to account for spelling errors

Document as Vectors



- Words are **dimensions** of *documents vectors*
- You can vizualize vectors in a particular set of dimensions of your choosing
- Vectors should be similar for documents that are *related*
    - But what does *similar* mean here ?

Similarity between documents: cosine distance

Most common similarity metric in NLP:

- Based on the dot product, which alone favors long documents: more words, higher values.
- Normalizing the dot product gives us the cosine of the angle between the two vectors:

$$\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1||\ ||\mathbf{d}_2||}$$

- Values range in $[-1, 1]$; frequencies $\rightarrow$ similarity is always positive
- $\text{cosine}(\mathbf{d}_1, \mathbf{d}_2) = 0 \rightarrow$ the documents have no words in common

Still, frequency is not the best measure of association between words:

- It is skewed $\rightarrow$ Zipf's law
- Again: very frequent words are rarely the most useful

Information retrieval

Improving representations for information retrieval ?

- Term frequency (TF) grows linearly, which isn't always effective: **saturation effect**
- There is **no proper document length normalization**, favoring longer documents

Lexical search is usually performed with **BM25** (Best Match 25). Ideas:

- Assume a *generative model* of documents where:
  - Term frequency follows a poisson distribution
    $\rightarrow$ repeating a term too many times doesn't unfairly boost relevance
  - Penalizes long documents naturally rather than dividing by total length
- Takes in account *word repetition* and *length normalization* with a minimal amount of new parameters

Classifying documents

On building classification datasets:

Labeling data **manually** is difficult:

- Annotations must capture the phenomenon of interest
- Annotations should be **replicable**
  - Formalize the instructions for the annotation task (*annotation guidelines*)
  - Compute **interannotator agreement measures** → Cohen's Kappa for discrete labels

- Recent trend: *perspectivism*
  - Modeling annotator disagreement

- Large-scale annotations ?
  - **Crowdsourcing**
  - Metadata as labels

Classification with Naïve Bayes

Multinomial naive Bayes classifier:

- Generative modeling:
  - Assume a model of document generation given an underlying variable: for a document d and classes c, $\mathbb{P}(d) = \sum_{c \in \mathcal{C}} \mathbb{P}(d|c)\mathbb{P}(c)$
  - Goal: for a document d return the class $\hat{c}$ with maximum a posteriori probability among classes: $\hat{c} = \text{argmax}_{c \in \mathcal{C}} \mathbb{P}(c|d)$
- Applying **Baye's rule** and the **naïve independance assumption**, and noting $d = (w_1, ..., w_n)$ we get a (*prior* $\times$ *likelihood*) decomposition:

$$\hat{c} = \text{argmax}_{c \in \mathcal{C}} \left[\mathbb{P}(c) \prod_{i=1}^{n} \mathbb{P}(w_i|c) \right]$$

- Idea: we compute $\mathbb{P}(w|c)$ as the frequency of w among all documents $d \in \mathcal{D}$ of class c:

$$\mathbb{P}(w|c) = \frac{count_{\mathcal{D}}(w, c)}{\sum_{w' \in \mathcal{V}} count_{\mathcal{D}}(w', c)}$$

- Given these assumptions, it is legitimate (and practical) to represent a document d with its bag-of-word representation $\mathbf{d}$!

Classification with Naïve Bayes

Practical points: Use **log-probabilities** (why ?) and Laplace smoothing

Training Algorithm: Given data $\mathcal{D}$ and classes $\mathcal{C}$

- Create the vocabulary $\mathcal{V}$ from documents $d \in \mathcal{D}$
- From $\mathcal{D}$: For each class $c \in \mathcal{C}$: compute the log-prior $\log \mathbb{P}(c)$
  - For each word $w \in \mathcal{V}$ compute the log-likelihood $\log \mathbb{P}(w|c)$

- **Inference Algorithm**: Given document d to classify:
  - For each class $c \in \mathcal{C}$: $S(c) = \log \mathbb{P}(c)$
    - For each position $i \in d$:
      If $w_i \in \mathcal{V}$: $S(c) = S(c) + \log \mathbb{P}(w_i|c)$
  - Return $\text{argmax}_{c \in \mathcal{C}} S(c)$

Classification with logistic regression

A *discriminative* linear classifier: learns directly $\mathbb{P}(c|d)$ through computing a **linear score** and applying a **logistic function**.

- Binary case: for a set of documents $d \in \mathcal{D}$ represented by vectors $\mathbf{d}$ learn a vector $\mathbf{w}$ and a bias b maximizing the likelihood of making a good classification into $c = 1$ or $c = 0$.
- The probability $\mathbb{P}(c = 1)$ is obtained by applying the sigmoid to the *scalar product* plus *intercept*:

$$\mathbb{P}(c = 1) = \sigma(\mathbf{w} \cdot \mathbf{d} + b)$$

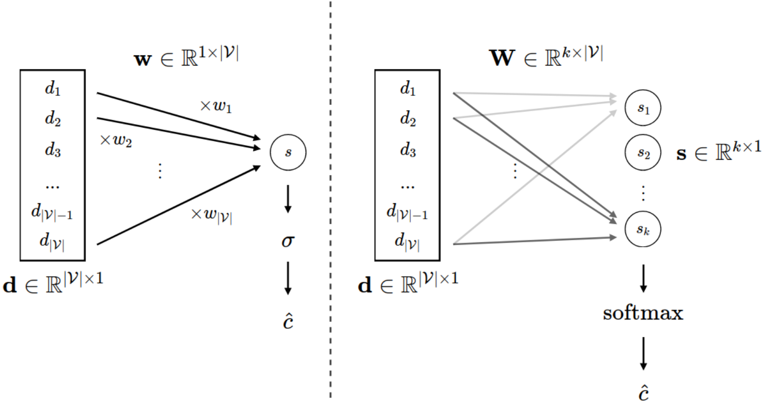
- We want to maximize the likelihood of our data by minimizing the **cross-entropy** between true and predicted classes $\hat{c}$ and c:

$$L(\hat{c}, c) = -\log \mathbb{P}(c|d) = -\left[c \log \hat{c} + (1 - c)\log(1 - \hat{c})\right]$$

- Here, the training is made through **gradient descent**: we minimize that loss function by finding iteratively the direction in which the loss decreases the most and updating the weights accordingly

Classification with logistic regression

The model is easily extended to a multinomial case through using a matrix $\mathbf{W}$, a vector $\mathbf{b}$ and the *softmax* function



Evaluating classifiers

- As usual: reserve held-out validation set for hyperparameters tuning / test set for evaluation
- Simplest measure: **Accuracy**

$$acc(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}|} \delta(\hat{y}^d = y^d)$$

- For each label $c \in \mathcal{C}$, look at the *type* of
  - Errors: False positive (**FP**) and False negative (**FN**)
  - Correct predictions: True positive (**TP**) and True negative (**TN**)

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**Positive (P)**	**True positive (TP)**	**False negative (FN)**
	Negative (N)	**False positive (FP)**	**True negative (TN)**

Evaluating classifiers

- Compute **recall** and **precision**:

$$Recall(\hat{\mathbf{y}}, \mathbf{y}, c) = \frac{TP}{TP + FN}$$

$$Precision(\hat{\mathbf{y}}, \mathbf{y}, c) = \frac{TP}{TP + FP}$$

- F-measure: combines recall (r) and precision (p) using the *harmonic mean*

$$F - measure(\hat{\mathbf{y}}, \mathbf{y}, c) = \frac{2rp}{r + p}$$

- Evaluating **multi-class classification**:
  - Balance across instances: Add up **TP**, **FP**, **TN**, **FN** over classes and compute the **Micro** F-measure
  - When classes are **imbalanced**, average over classes: **Macro** F-measure

$$\text{Macro} - F(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} F - measure(\hat{\mathbf{y}}, \mathbf{y}, c)$$

Using context

Word-level classification tasks

Assigning labels to **individual words** in a text rather than the entire sentence or document.

- Word Sense Disambiguation (**WSD**)

  "Apple CEO Tim Cook spoke at the conference in San Francisco last Friday."

  [COMPANY]

- Part-of-Speech (**POS**) Tagging

  "Apple CEO Tim Cook spoke at the conference in San Francisco last Friday."

  [NNP] [NNP] [NNP] [NNP] [VDB] [IN] [DT] [NN] [IN] [NNP] [NNP] [JJ] [NNP]

- Named Entity Recognition (**NER**)

  "Apple CEO Tim Cook spoke at the conference in San Francisco last Friday."

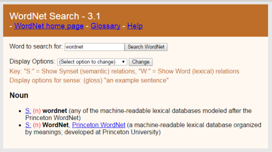
  [ORG] 0 [PERSON] [PERSON] 0 0 0 0 0 [LOC] [LOC] 0 [DATE]

Representing words

- Most basic: **one-hot** vector representations
  - No notion of *similarity* between two words

	run	ran	walk	walked
ran	0	1	0	0
run	1	0	0	0

- **Explicit** features:
  - List of attributes describing the object
  - Natural language definition (*dictionnary*)
  - Other lexical resources, including *senses* and associated properties, and morphological features: **WordNet**



Issue #2: Ambiguity

NLP system: must uncover **the structure of text**, which is made difficult by

- **Lexical** ambiguity: homography, polysemy
  - Part-of-speech tagging
  - Word sense disambiguation
- **Syntactic** ambiguity:
  - Dependency parsing
- Ambiguity in **semantic scope**:
  - Semantic parsing

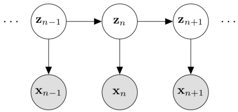
These tasks are often complicated by lacking *implicit knowledge*

- Background, commonsense knowledge
- Contextual knowledge

Next: Sequence modeling

Work on **modelisation** of the context - which is the sequence:

- Use appropriate **sequence models** which will get the necessary information from the immediate context
- Model **dependency within the sequence**: *Markov models*
- Generative modeling:
    - Assume an **unknown sequence of states** (*grammatical tags*) and observations depending on those states (*observed words*)
    - Find the *most probable underlying tag sequence*
      $\rightarrow$ Hidden Markov Models !

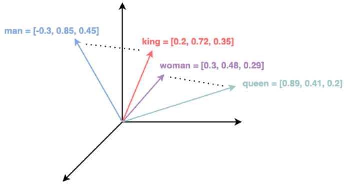


- Later: Deep learning sequential models (recurrent, transformers)

Next: Distributed representations

Work on **representations** integrating information from the context
→ Usually through **unsupervised learning**

- Propose to represent both words and document with **intermediary concepts**:
  - **Topic modeling**: often based on . . . generative modeling
  - **Vector spaces** for words: encode **contextual** information
    - *Distributional hypothesis*: two words are similar if they have similar contexts
    - Create sparse then dense representations → Embeddings !



- Later: Deep learning sequential models for word representations