Natural Language Processing

Name: Marcel Aguilar Garcia

December 26, 2020

1 Linguistic Levels of Analysis

Linguistic analysis The main levels of linguistic analysis are morphological, syntactic, semantic, pragmatic and phonological.

1.1 Morphological Analysis

Morphological analysis is the study of words, how they are formed, and their relationship to other words in the same language. It analyzes the structure of words and parts of words, such as stems, root, words, prefixes and suffixes. Let's see concepts associated to morphological analysis:

Stemming

Stemming is the process of reducing inflected words to their word stem, base or root form - generally a written word form. Example: change, changing, changes, changed and changer have a common stem: chang.

Lemmatisation

Lemmatisation is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form. Example: change, changing, changes, changed and changer share the same common lemma: change.

Inflection

Inflection is the process of word formation in which a word is modified to express different grammatical categories. It analysis the internal structure of words into lemma with inflectional features. Inflectional patterns are shared across words class, e.g., regular verbs. One of the challenges of inflection is the ambiguity meaning of words e.g., 'books' can be interpreted as a verb or as a noun. Example: 'work' →, 'works', 'worked', etc.

Derivation

Derivation is the formation of a word by changing the form of the base or by adding affixes to it. Example: agreement (Noun) is derived from agree (Verb).

Compounds

A compound is a word that consists of more than one stem. Compounding is the process of word formation that creates compound words. Example: Flachbildschirm in German is a compound from flach and Bildschirm (flat and screen).

Multi-word expressions

Multi-word expressions are linguistic objects formed by two or more words that behave like a 'unit' by displaying formal and/or functional idiosyncratic properties with respect to free word combinations. Example: 'Cardiac arrest'

Tokens & Types

Tokens and Types are often used in NLP. Morphological analysis can often impact the nature and number of them through inflection, derivation and decomposition. **Tokens** are the collection of individual words in a text while **types** is the set of tokens, i.e., with no repetition.

1.2 Syntactic Analysis

Parsing or **syntactic analysis** is the process of analyzing a string of symbols conforming to the rules of a formal grammar. Within computational linguistics, the term is used to refer to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other. Let's see concepts associated to syntactic analysis:

Context-Free grammar

A context-free Grammar is a 4-tuple G = (N, Σ, P, S) that consists of:

- A set of non-terminal symbols N, e.g, ' noun (N)', 'verb (V)', 'noun phrase (NP)', 'verb phrase (VP)', 'sentence / start (S)', etc.
- A set of terminal symbols Σ, e.g, 'astronomer', 'cat', 'see', 'the'
- A set of productions or rules, e.g, S $\rightarrow$ NP VP. Note that rules may lead to recursion.
- A start symbol, normally 'S'.

A specific case of Context-Free grammar are phrase structure grammars which are all those grammars that are based on the constituency relation, as opposed to the dependency relation associated with dependency grammars.

Part-of-speech tagging or grammatical tagging

Part-of-speech tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. An example is PoS tag set for English which has 36 tags.

Lexical rule (Lexicon)

Lexical rules provide a mechanism for expressing redundancies in the lexicon, such as the kinds of inflectional morphology used for word classes. They can be used to find the production rules for the terminal symbols. E.g., Det $\rightarrow$ the, Noun $\rightarrow$ cat|mat|dots.

Parse Tree Based on Grammar & Lexicon

A parse tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar.

- **Constituency-based parse tree**s: The constituency-based parse trees of constituency grammars (= phrase structure grammars) distinguish between terminal and non-terminal nodes. The interior nodes are labeled by non-terminal categories of the grammar, while the leaf nodes are labeled by terminal categories.

- **Dependency-based parse trees**: Dependency structures represent the grammatical relations that hold between constituents. They are more specific in terms of semantics and the notion of relations across words is explicit. A dependency tree for a sentence is a directed a-cyclic graph with words as nodes and relations as edges.

1.3 Semantic Analysis

Semantic analysis is the process of relating syntactic structures, from the levels of phrases, clauses, sentences and paragraphs to the level of the writing as a whole, to their language-independent meanings. Semantic analysis can begin with the relationship between individual words. This requires an understanding of lexical hierarchy, including hyponymy and hypernymy, meronomy, polysemy, synonyms, antonyms, and homonyms. It also relates to concepts like connotation (semiotics) and collocation, which is the particular combination of words that can be or frequently are surrounding a single word.

Semantic analysis is concerned with how linguistic symbols refer to concepts about referents. Relation between linguistic symbols and concepts commonly referred to as 'meaning'. Let's see concepts associated to semantic analysis:

Lexical Semantics

Lexical semantics looks at how the meaning of the lexical units correlates with the structure of the language or syntax. Lexical items participate in regular patterns of association with each other. Some relations between lexical items include:

- hyponymy: is a word whose semantic field is included within that of another word.
- hypernymy: while hyponym refers to a specific. instance, hypernym refers to the generic term. The semantic field of a hypernym is broader than that of a hyponym.
- meronomy: is a type of hierarchy that deals with part–whole relationships
- polysemy: is the capacity for a word to have multiple meanings.
- synonyms: is a word that means exactly or nearly the same as another word.
- antonyms: is a word that means the exactly or nearly the opposite as another word.
- homonyms: each of two or more words having the same spelling or pronunciation but different meanings and origins.

The study of lexical semantics looks at:

- the classification and decomposition of lexical items
- the differences and similarities in lexical semantic structure cross-linguistically
- the relationship of lexical meaning to sentence meaning and syntax.

Compositional Semantics

Compositional Semantics combines the meaning of its subphrases using rules which are driven by syntactic structure to determine the meaning of a phrase. In other words, compositional semantics deals with how lexical meanings combine to form more complex phrasal meanings.

Discourse Semantics

It is an extension of semantics to include an analysis of the relation between a sentence and the context or discourse in which it is embedded.

Semantic Analysis Tasks

Natural Language Processing tasks that are associated with the previous concepts are:

- Word Sense Disambiguation: identify intended meaning of a word in a specific language context.
- Semantic Role Labeling: identify implied semantic role of an entity expressed by a word or phrase.
- Coreference Resolution: identify which words or phrases express the same entity.

2 Types of language data

2.1 Corpus

A corpus is a language dataset. There are different types of corpus such as:

- General Corpus mostly represents a whole language and is used for training general purpose NLP tools.
- Domain Corpus is representative of a domain specific subset of a language, e.g., in healthcare, finance, legal, etc.
- Annotated Corpus is a corpus with annotations on word, phrase, sentence or document level (e.g, PoS, word senses, etc). Annotated corpus is used in NLP for supervised training and/or evaluation.
- Monolingual, Bilingual and Multilingual Corpus: While monolingual in only one language, bilingual and multilingual corpus can have pairs of tuples of translations and are mostly used for training machine translations systems.
- Parallel and Comparable Corpus: Parallel corpus are a collection of translated documents while comparable corpus are a collection of documents on the same topic but different languages.

2.2 Lexicon

A lexicon is a database of words with lexical properties such as spelling, pronunciation, morphology, PoS, semantics, etc.

In NLP, the most widely used lexicons are semantic lexicons, specifically WordNet and also FrameNet.

WordNet

Word meaning represented by use of "synsets". Synsets are organized hierarchically in WordNet, expressing generalization (hypernyms) and specialization (hyponyms)

FrameNet

Based on Frame Semantics. Frame is a description of a type of event, relation or entity and the participants in it. Lexical units within a frame are synonyms.

3 Vector Space Model & Word Embeddings

3.1 Vector Space Model

Vector space model or term vector model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers (such as index terms). It is used in information filtering, information retrieval, indexing and relevancy rankings. Documents and queries are represented as vectors. Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. There are different ways of computing these values, also known as (term) weights:

- Inverted index is an index data structure storing a mapping from words to its locations set of documents
- Document Vectors can be used to represent each document. These are binary vectors with length equal to the number of unique terms. A 1 is used when the term is in the document, 0 otherwise. This technique is often used in Document Retrieval using similarity between the query and the Document Vector.
- Term Vectors can be used to represent each term. These are binary vectors with length equal to the number of documents. It is similar to Document Vector. It can help to identify which terms are more similar between them based on the documents that they occurred.

3.2 Distributional Semantics

Distributional Semantics quantifies and categorises semantic similarities between linguistic items based on their distributional properties in large samples of language data. The basic idea of distributional semantics can be summed up in the so-called Distributional hypothesis: linguistic items with similar distributions have similar meanings. Examples of mathematical representations to analyze Distributional Semantics are:

- Co-occurrence matrix has the purpose to present the number of times each word appears in the same context than other words.
- Distributional Similarity computes the similarity of two words using their vector representations. An example of this is the cosine similarity given by the cosine distance.

3.3 Context in Distributional Representations

Important aspects of the meaning of a word are a function of (can be approximated by) the set of contexts in which it occurs in texts. There are different contexts that can be used:

- **Context window**: It is used to determine the context of a word within a document, sentence, phrase, word sequence, etc.
- **Context content**: It is used to determine the context of a word by removing stopwords, PoS Tagging, using Syntactic Dependencies, etc.

The weighting of the context shows different measures that can be used within each of the previous contexts. Examples of these are:

- **Context occurrence**: Binary vector that shows if context c occurs with word w
- **Context frequency**: Vector that shows the number of times that context occurs with word w
- **Context weight**: Vector that shows how relevant or specific context c is for word w

The most used Context Weights are:

Term Frequency - Inverse Document Frequency

The tf-idf value increases proportionally to the number of times a word appear in the document and is offset by the number of document in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. There are two concepts used to calculate the tf-idf:

The **term frequency** shows how frequently a word occurs in a document. However, this frequency by its own would tend to be higher in long texts. Therefore, term frequency can be informative when it is set in relation to the text length.

The **inverse document frequency** is a weight indicating how commonly a word is used. The lower the score, the less important the word becomes. IDF is computed as $IDF = \log(\frac{N}{DF_t+1})$ where N is the total number of documents in your text collection and DF_t is the number of documents containing the term t.

Finally, **TF-IDF** measures how important a term is in a document where it occurs, in comparison to the overall occurrence of this term in a set of Documents. TF-IDF is mostly used in IR to put weights on index terms but can be used also in distributional semantics, using words (Terms) and sentences (Docs). TF-IDF is the product of these two statistics, term frequency and document frequency:

$$\text{TF-IDF}(w) = \text{TF(w)} \cdot \text{IDF}(w)$$

Pointwise Mutual Information (PMI)

The PMI is a correlation or association measure that quantifies the likelihood of co-occurrence of two independent events. In distributional semantics we use PMI to measure the likelihood of co-occurrence of two words across a corpus.

The PMI for two given words can be calculated as $PMI = \log(\frac{P(w_1,w_2)}{P(w_1)P(w_2)+1})$ where $P(w_1, w_2)$ the probability of seeing words w_1 and w_2 together and $P(w_1)P(w_2)$ is the probability of the same if w_1 and w_2 are independent.

PMI is over-sensitive to the chance co-occurrence of infrequent words.

3.4 Word Embeddings

Word embedding is any of a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension.

1. Initialize high-dimensional vector space with one-hot vectors over a vocabulary V derived from training corpus.

2. Count or predict (language model) co-occurrence for all words in V within a given context (e.g bigrams) with co-occurrence taken as positive instance and as negative instance otherwise.

3. Train a low dimensional embedding space on a classification task that correctly classifies positive and negative co-occurrence instances

4. Results in a word embedding for this corpus, model and task

Example: Word2Vec

4 Language Modeling

Language modeling (LM) is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Language models analyze bodies of text data to provide a basis for their word predictions.

4.1 Probability Concepts

Conditional probability is a measure of the probability of an event occurring, given that another event has already occurred. The probability of an event A given an event B is denoted as $P(A|B)$ and defined as:

$$P(A|B)P(B) = P(A \cap B)$$

From this definition, it is possible to prove Bayes' theorem. It is stated mathematically as the following equation:

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

where A and B are events and $P(B) \neq 0$.

The probability of a word w in a corpus can be calculated as

$$P(w) = \frac{c(w)}{\sum_{v \in W} c(v)}$$

where $c(w)$ is the number of occurrences of a word w in the corpus. This can be interpreted as the number of occurrences of a specific word w over the total number of words in the corpus.

Using this probability we can get a first estimation of the probability of a sentence $w_1 w_2 ... w_n$ in a corpus:

$$P(w_1 w_2 ... w_n) = P(w_1|w_2 ... w_n)P(w_2|w_3 ... w_n)...P(w_n) \approx P(w_1)P(w_2)...P(w_n)$$

The main issue with this estimation is what happens when we try to estimate a sentence which contains a word that is not in the corpus vocabulary. In this case the probability of the sentence will be always zero.

A first idea that can solve this issue is **Add-one smoothing** in which we add one to every count. Therefore a word that is not in the vocabulary would have a probability of $P(w) = \frac{1}{\sum_{v \in V} c(v) + |V|}$. The general formula is given by:

$$P(w) = \frac{c(w) + 1}{\sum_{v \in V} c(v) + |V|}$$

Note that in Add-one smoothing, the probability distribution with out of vocabulary words does not sum to one. Additionally, it overestimates low probabilities and underestimates frequent probabilities.

Another alternative would be using **Laplace smoothing** (or add alpha) which introduces two parameters α and N. α is a positive real number known as 'pseudocount'. A 'pseudocount' is an amount added to the number of observed cases in order to change the expected probability in a model of those data, when not known to be zero. N is the expected number of OOV. The probability formula for Laplace smoothing is given by:

$$P(w) = \frac{c(w) + \alpha}{\sum_{v \in V} c(v) + \alpha(|V| + N)}$$

However, it is not clear how to find N and α.

The total probability of words in the vocabulary can be calculated as:

$$P(\text{known words}) = \sum_{v \in V} P(v) = \sum_{v \in V} \left(\frac{c(v) + \alpha}{\sum_{v \in V} c(v) + \alpha(|V| + N)}\right) = \frac{\sum_{v \in V} c(v) + \alpha|V|}{\sum_{v \in V} c(v) + \alpha(|V| + N)}$$

In a similar way, OOV words can be calculated as:

$$P(\text{OOV words}) = 1 - P(\text{known words}) = 1 - \frac{\sum_{v \in V} c(v) + \alpha|V|}{\sum_{v \in V} c(v) + \alpha(|V| + N)} = \frac{\alpha N}{\sum_{v \in V} c(v) + \alpha(|V| + N)}$$

Let's consider the table of 'Count of counts'. This table has in each row i the number of words that occurs i times in the corpus. Using this data we can using, e.g linear regression, to have an approximation of the number of words that appear zero times. Therefore we can estimate N which would give us an approximation of $P(\text{OOV words})$.

4.2 n-grams

Recalling the definition of conditional probability in a sequence of words:

$$P(w_1 w_2 ... w_n) = P(w_n | w_1 ... w_{n-1}) P(w_1 ... w_{n-1}) = \Pi_{k=1,...,n} P(w_k | w_1 ... w_{k-1})$$

A better approximation can be given when using n-grams but some changes need to be done due to 'data sparsity', e.g. nearly every ten word sentences are going to be unique which would make this probability zero.

In order to solve this issue, we can restrict the number of words. So an n-gram model, only looks back at the last m words from the current word.

$$P(w_1 w_2 ... w_n) \approx \Pi_{k=1,...,n} P(w_k | w_{k-m+1} ... w_{k-1})$$

Let's take a look to some smoothing techniques that are specific for calculations of probabilities with n-grams.

Back-Off

One of the problems is that while different n-grams may have similar probabilities, we may want to take into account the probability of each word individually, e.g, probability of the bigram (w_1, w_2), might be similar to the probability of the bigram (w_1, w_3). However, if $P(w_2) > P(w_3)$, we may want to consider the first bigram rather than the second.

A technique to solve this would be applying **back-off** which says that if $c(w_1 w_2) = 0$ then we should use $c(w_2)$ instead. However, the solutions of a single word are usually higher than the bigrams. This method would tend to give higher weights to words. In order to solve this we must discount the counts:

$$c_{\text{back-off}}(w_1 w_2) = \begin{cases} d_r c(w_1 w_2) \text{ if } c(w_1 w_2) > 0 \\ \alpha c(w_2) \text{ if } c(w_1 w_2) = 0 \end{cases} \tag{1}$$

where d_r is the probability mass of seen examples and can be calculated by linear regression as seen previously. α is such that $\sum_{w \in V} c_{\text{back-off}}(w_1 w) = \sum_{w \in V} c(w_1 w)$

Interpolation

An alternative technique would be using Interpolation, e.g. 2-grams:

$$P_{LM}(w_n | w_{n-1}) = (1 - \lambda_1) P(w_n | w_{n-1}) + \lambda_1 P(w_n)$$

It can generalise to n-grams:

$$P_{LM}(w_n | w_{n-k+1} .. w_{n-1}) = (1 - \lambda_{k-1}) P(w_{n-k+1} | w_{n-1}) + \lambda_{k-1} P(w_n | w_{n-k+2} ... w_{n-1})$$

Witten-Bell Smoothing

This technique is based on the idea that some bigrams nearly occur almost as much as the unigram, e.g, "San Francisco". Therefore, in this cases $P(w_1w_2) \approx P(w_2)$ and, at the same time, $P(w_t|w_1) << P(w_2|w_1)$.

In order to achieve this we can introduce the concept of 'diversity of history':

$$N_1 + (w_1-) = |w_2 : c(w_1w_2) > 0|$$

$$\lambda = \frac{N_1 + (w_1-)}{N_1 + (w_1-) + c(w_1-)}$$

This allow us to calculate an optimal λ that can be used in our interpolation.

Evaluation of Languages Models

A good Language Model should produce high probabilities for valid (e.g in English) sentences. Should produce low probabilities for ungrammatical sentences.

Evaluation normally looks at first condition. If language model produces valid probability distribution. Perplexity is the most widely used evaluation metric:

$$H(P_{LM}) = -\frac{1}{n}P_{LM}(w_1, ..., w_n)$$
$$PP = 2^{H(P_{LM})}$$

5 Hidden Markov Model

Let $\{X_n\}$ and $\{Y_n\}$ be discrete-time stochastic processes and $n \geq 1$. The pair $(\{X_n\}, \{Y_n\})$ is a Hidden Markov Model if:

- $\{X_n\}$ is a Markov process and whose states and transition probabilities are not directly observable ("hidden")
- $P(Y_n \in A|X_1 = x_1, ..., X_n = x_n) = P(Y_n \in A|X_n = x_n)$ for every $n \geq 1$, $x_1, ..., x_n$, and arbitrary measurable set A.

A HMM has the following components:

- Set of states (tags) $S = s_1, ..., s_N$
- Output alphabet $K = k_1, ..., k_M$
- Emission probabilities $p(w_i|t_j) = B_{ij}$ where w_i is a word from the output alphabet, t_j is a tag and A is an M x N matrix.
- State transition probabilities $P(t_i|t_j) = A_{ij}$ where A is an N x N transition matrix.
- State sequences (hidden) $T = \{t_1, ..., t_t\}$
- Output sequence (observed) $W = w_1, ..., w_T$

HMM can be used to know if a sequence of words are correct in a specific language given the PoS tags. The following formula can be used to calculate the probability of a sequence of words and their tags:

$$\begin{aligned} P(w_1, ..., w_N, t_1, ..., t_N) = \\ P(w_N|w_1, ..., w_{N-1}, t_1, ..., t_N) \\ P(t_N|w_1, ..., w_{N-1}, t_1, ..., t_{N-1}) \\ P(w_{N-1}|w_1, ..., w_{N-2}, t_1, ..., t_{N-1}) \\ P(t_{N-1}|w_1, ..., w_{N-3}, t_1, ..., t_{N-2}) \\ ... \\ P(t_2|w_1, t_1) \\ P(w_1|t_1) \\ P(t_1) \end{aligned} \tag{2}$$

This can be simplified by using an approximation:

$$\begin{aligned} P(w_1, ..., w_N, t_1, ..., t_N) \approx \\ P(w_N|t_N) \\ P(t_N|t_{N-1}) \\ P(w_{N-1}|t_{N-1}) \\ P(t_{N-1}|t_{N-2}) \\ ... \\ P(t_2|t_1) \\ P(w_1|t_1) \\ P(t_1) \end{aligned} \tag{3}$$

This is $P(w_1, ..., w_N, t_1, ..., t_N) \approx \Pi_{i=1,...N} P(t_i|t_{i-1})P(w_i|t_i)$ where $t_0 =$ 'START'.

5.1 Tagging

Tagging is the process of assigning (exactly) one class to (each) word. Examples of tagging are Part-of-Speech and Named Entity Recognition. HMM can be use for tagging by solving:

$$\arg \max_{t_1,..,t_n} P(t_1, ..., t_n|w_1, ..., w_n)$$

This is the most likely sequence of tags given a sequence of words. However, if you have T part of speech tags and N words, the complexity of checking all combinations would be $\mathcal{O}(TN^T)$. Using Dynamic Programming it is possible to have a more efficient algorithm.

Viterbi Algorithm

The **Viterbi algorithm** is a dynamic programming algorithm for finding the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models (HMM).

Algorithm 1: Viterbi Algorithm

Result: $\arg\max_{t_1,..,t_n} P(t_1,...,t_n|w_1,...,w_n)$
Set $\pi_{s,0} = 0$ except for $\pi_{Start,0} = 1$;
Set $y_s = []$;
for *i from 1 to n* **do**
  for $s \in S$ **do**
    Set $\pi_{s,i} = \max_{t \in S} \pi_{t,i-1} p(s|t) p(w_i|s)$;
    Append t to y_s;
  end
end
Return $y_s + s$ where s maximizes $\pi_{s,T}$

Example of Viterbi Algorithm:

$$\text{Emission Probabilities} = \begin{pmatrix} - & John & cat & has & a \\ N & 0.3 & 0.3 & 0.1 & 0.3 \\ V & 0.1 & 0.1 & 0.7 & 0.1 \end{pmatrix}$$

$$\text{Transition Probabilities} = \begin{pmatrix} - & P(N|\cdot) & P(V|\cdot) \\ N & 0.7 & 0.3 \\ V & 0.5 & 0.5 \\ S & 0.9 & 0.1 \end{pmatrix}$$

For i = 0,

$$\pi_{s,0} = \begin{pmatrix} \pi_{S,0} & \pi_{N,0} & \pi_{V,0} \\ 1.0 & 0.0 & 0.0 \end{pmatrix} \tag{4}$$

$$y_t = [START]$$

For i=1,

$$\begin{aligned} \pi_{s,1} &= \begin{pmatrix} \pi_{S,1} & \pi_{N,1} & \pi_{V,1} \\ 0.0 & max(0.9 \cdot 0.3, 0.0, 0.0) & max(0.1 \cdot 0.1, 0.0, 0.0) \end{pmatrix} \\ &= \begin{pmatrix} \pi_{S,1} & \pi_{N,1} & \pi_{V,1} \\ 0.0 & 0.27 & 0.01 \end{pmatrix} \end{aligned} \tag{5}$$

Note that in this case:

$$\pi_{S,1} = max \begin{cases} \pi_{S,0} P(S|S) P(w_1|S) = 0 \\ \pi_{N,0} P(S|N) P(w_1|S) = 0 \\ \pi_{V,0} P(S|V) P(w_1|S) = 0 \end{cases} \tag{6}$$

$$\pi_{N,1} = max \begin{cases} \pi_{S,0} P(N|S) P(w_1|N) = 0.27 \\ \pi_{N,0} P(N|N) P(w_1|N) = 0 \\ \pi_{V,0} P(N|V) P(w_1|N) = 0 \end{cases} \tag{7}$$

$$\pi_{V,1} = max \begin{cases} \pi_{S,0} P(V|S) P(w_1|V) = 0.01 \\ \pi_{N,0} P(V|N) P(w_1|V) = 0 \\ \pi_{V,0} P(V|V) P(w_1|V) = 0 \end{cases} \tag{8}$$

Therefore we can say that $y_t = [START, N]$.

For i=2,

$$\begin{aligned}\pi_{s,2} &= \begin{pmatrix} \pi_{S,2} & \pi_{N,2} & \pi_{V,2} \\ 0.0 & max(0.27 \cdot 0.7 \cdot 0.1, 0.01 \cdot 0.5 \cdot 0.1) & max(0.27 \cdot 0.3 \cdot 0.7, 0.01 \cdot 0.5 \cdot 0.7) \end{pmatrix} \\ &= \begin{pmatrix} \pi_{S,2} & \pi_{N,2} & \pi_{V,2} \\ 0.0 & 0.019 & 0.057 \end{pmatrix}\end{aligned} \tag{9}$$

Note that in this case:

$$\pi_{S,2} = max \begin{cases} \pi_{S,1} P(S|S) P(w_2|S) = 0 \\ \pi_{N,1} P(S|N) P(w_2|S) = 0 \\ \pi_{V,1} P(S|V) P(w_2|S) = 0 \end{cases} \tag{10}$$

$$\pi_{N,2} = max \begin{cases} \pi_{S,1} P(N|S) P(w_2|N) = 0 \\ \pi_{N,1} P(N|N) P(w_2|N) = 0.019 \\ \pi_{V,1} P(N|V) P(w_2|N) = 0.0005 \end{cases} \tag{11}$$

$$\pi_{V,2} = max \begin{cases} \pi_{S,1} P(V|S) P(w_2|V) = 0 \\ \pi_{N,1} P(V|N) P(w_2|V) = 0.057 \\ \pi_{V,1} P(V|V) P(w_2|V) = 0.035 \end{cases} \tag{12}$$

Therefore we have $y_t = [START, N, V]$

Forward and Backward Algorithm

HMM can also work as language models. Using forward or backward algorithm it is possible to calculate the probability of a sequence of words. This is:

$$\begin{aligned} P(w_1, ...w_n) &= \sum_{t_1 \in S, ..., t_n \in S} P(w_1, ...w_n, t_1, ..., t_n) \\ &\approx \sum_{t_1 \in S} ... \sum_{t_n \in S} \Pi_{i=1,...,n} p(t_i|t_{i-1}) p(w_i|t_i) \end{aligned} \tag{13}$$

Algorithm 2: Forward Algorithm

Result: $P(w_1, ..., w_n)$
Set $\alpha_{s,0} = 0$ except for $\alpha_{Start,0} = 1$;
Set $y_s = []$;
for *i from 1 to n* **do**
  for $s \in S$ **do**
    Set $\alpha_{s,i} = \sum_{t \in S} \alpha_{t,i-1} p(s|t) p(w_i|s)$;
  end
end
Return $\sum \alpha_{s,T}$

Example of Forward Algorithm:

$$\text{Emission Probabilities} = \begin{pmatrix} - & John & cat & has & a \\ N & 0.3 & 0.3 & 0.1 & 0.3 \\ V & 0.1 & 0.1 & 0.7 & 0.1 \end{pmatrix}$$

$$\text{Transition Probabilities} = \begin{pmatrix} - & P(N|\cdot) & P(V|\cdot) \\ N & 0.7 & 0.3 \\ V & 0.5 & 0.5 \\ S & 0.9 & 0.1 \end{pmatrix}$$

For i = 0,

$$\alpha s, 0 = \begin{pmatrix} \alpha_{S,0} & \alpha_{N,0} & \alpha_{V,0} \\ 1.0 & 0.0 & 0.0 \end{pmatrix} \tag{14}$$

For i=1,

$$\begin{aligned} \alpha_{s,1} &= \begin{pmatrix} \alpha_{S,1} & \alpha_{N,1} & \alpha_{V,1} \\ 0.0 & 0.9 \cdot 0.3 + 0.0 + 0.0 & 0.1 \cdot 0.1 + 0.0, 0.0 \end{pmatrix} \\ &= \begin{pmatrix} \alpha_{S,1} & \alpha_{N,1} & \alpha_{V,1} \\ 0.0 & 0.27 & 0.01 \end{pmatrix} \end{aligned} \tag{15}$$

For i=2,

$$\begin{aligned} \pi_{s,2} &= \begin{pmatrix} \pi_{S,2} & \pi_{N,2} & \pi_{V,2} \\ 0.0 & 0.27 \cdot 0.7 \cdot 0.1 + 0.01 \cdot 0.5 \cdot 0.1 & 0.27 \cdot 0.3 \cdot 0.7 + 0.01 \cdot 0.5 \cdot 0.7 \end{pmatrix} \\ &= \begin{pmatrix} \pi_{S,2} & \pi_{N,2} & \pi_{V,2} \\ 0.0 & 0.019 & 0.60 \end{pmatrix} \end{aligned} \tag{16}$$

Therefore we have that the final probability is $P(John, cat) = 0.019 + 0.060 = 0.0796$.

In a similar way, the backward algorithm can calculate the same probability:

Algorithm 3: Backward Algorithm

Result: $P(w_1, ..., w_n)$
Set $\beta_{s,0} = 0$ except for $\beta_{Start,0} = 1$;
Set $y_s = []$;
for *i from 1 to n* **do**
  for $s \in S$ **do**
    Set $\beta s, i = \sum_{t \in S} \beta_{t,i+1} p(t|s) p(w_{i+1}|t)$;
  end
end
Return $\sum \beta_{s,T}$

By mixing both algorithms, we have the Forward-Backward Algorithm which can give the probabilities of a single tag:

$$P(T_i = s, W|\mu) = \alpha_{s,i} \beta_{s,i}$$

Learning Hidden Markov Models

When using Supervised Learning, if we now the tags for some set of words we can directly obtain the probabilities by counting:

$$P(s_i|s_j) = \frac{c(t_{i-1} = s_j, t_i = s_i)}{\sum_{s'} c(t_{i-1} = s_j, t_i = s')}$$

$$P(w|s) = \frac{c(w_i = w, t_i = s)}{c(t_i = s)}$$

However, for Unsupervised Learning, we want to find a model that maximizes the likelihood of the data given that we do not have any part-of-speech tagging but just an idea of the number of states we have. Therefore, we are trying to calculate the model which is most likely, so the most likely matrices A and B that maximizes essentially the language model probability we saw before.

There is no known analytic method to select the μ that maximizes the above formula but we can use the **Baum-Welch algorithm**. In general, a expectation-maximization Algorithm will follow the steps:

1. Initialize the model
2. Estimate the counts
3. Re-compute the model given the estimated counts
4. Repeat 2 and 3 until convergence

Using Baum-Welch Algorithm, we can calculate the probability of each state as:

$$\gamma_i(t) = P(t_i = t|W, \mu) = \frac{\alpha_{t,i}\beta_{t,i}}{\sum_{s \in S} \alpha_{s,i}\beta_{s,j}}$$

and the transition probability as:

$$\varepsilon_i(s,t) = P(t_i = s, t_{i+1} = t|W, \mu) = \frac{\alpha_{s,i}a_{s,t}\beta_{t,i+1}b_{t,w_{i+1}}}{\sum_{s' \in S}\sum_{t' \in T} \alpha_{s',i}a_{s',t'}\beta_{t',i+1}b_{t',w_{i+1}}}$$

The most likely estimators of our are probabilities are thus:

$$a^*_{s,t} = \frac{\sum_{i=1}^{T-1} P(t_i = s, t_{i+1} = t|W, \mu)}{\sum_{i=1}^{T-1} P(t_i = s|W, \mu)} = \frac{\sum_{i=1}^{T-1} \varepsilon_i(s,t)}{\sum_{i=1}^{T-1} \gamma_i(s)}$$

$$b^*_{s,w} = \frac{\sum_{i=1}^{T} \mathbf{1}(w_i = w)\gamma_i(s)}{\sum_{i=1}^{T} \gamma_i(s)}$$

In practice, Baum-Welch is most useful as semi-supervised method. Some tags are observed, some not.

Semi-supervised Baum-Welch, combines the supervised and unsupervised modes:

$$a^*_{s,t} = \frac{\sum_i \varepsilon(s,t) + |\{i : t_i = s, t_{i+1} = t\}|}{\sum_i \gamma_i(s) + |i : t_i = s|}$$

and similarly for emission probabilities.

An example of semi-supervised Baum-Welch would be a corpus with WAV files and IPA transcription.

6 Probabilistic Context-free Grammars

6.1 Parsing with probabilistic Context-free Grammars

As seen before, parsing is the problem of finding the structure of a sentence. The problem with parsing is that is often ambiguous. Probabilities on parses allow us to choose the best (most-likely) parse tree and allow parses to be language models.

Let's introduce probabilistic context-free grammar based on our previous definition of context-free grammar.

A context-free grammar is a 5-tuple $G = (N, \Sigma, P, S, D)$ that consists of:

- A context-free grammar $G = (N, \Sigma, P, S)$
- A function D: $P \to [0, 1]$ which assigns a probability to each production.

A PCFC is consistent iff $\sum_{\{\beta:A\to\beta\in P\}} D(A \to \beta) = 1 \ \forall A \in A$ and there are no infinite derivations for any finite string (e.g., $S \to S$).

Let's see an example of PCFG:

Rule	Prob	Rule	Prob
S → NP VP	0.8	VP → V NP	0.9
S → Aux NP VP	0.2	VP → V NP NP	0.1
NP → PN	0.45	N → flights	1.00
NP → Nom	0.05	V → book	1.00
NP → Pro	0.5	Aux → can	1.00
Nom → N	0.90	PN → Lufthansa	1.00
Nom → PN Nom	0.1	Pro → you	1.00

Given a parse T and an input S we know that $P(T|S) = P(T, S)P(S)$. From here, we can calculate the best parse by using:

$$T^* = \arg\max_T P(T, S)$$

As the parse tree contains all words in the sentence $P(T, S) = P(T)P(S|T) = P(T)$. Hence $P(T, S) = \Pi_{n\in T} D(r(n))$ where $r(n)$ is the rule used to generate n. Therefore, given two different parsings for the same sentence, we can calculate the probability of each parsing by multiplying the probabilities of the rules. This will give us the most-likely parsing for that sentence with the given probabilities.

6.2 Cocke-Younger-Kasami (CYK) algorithm

It's not practical to find all parse trees but we can find the parse of a CFG in $\mathcal{O}(N^3)$ and we can find the best parse in $\mathcal{O}(N^3)$.

CYK is essentially a bottom-up parser that can be adapted to PCFGs with dynamic programming that returns $T^* = \arg\max_T P(T, S)$. CYK has as input a sequence of words $w_1, ..., w_n$. The data structure is a table $t_{i,j,a}$ where $1 \le i \le j \le n$ and $a \in N$. This gives us the probability that we have parsed between word i up to word j a parse tree where the root symbol is a. Let's see an example of CYK with the previous rules for the sentence: **Can you book Lufthansa flights?**

At the first level of the table, we use the rules that produce the set of terminal symbols (words) and select the non-terminal symbols. Additionally, we repeat the same with the rules that produce these non-terminal symbols:

can	you	book	Lufthansa	flights
Aux (1.0)	Pro (1.0), NP (0.5)	V (1.0)	PN (1.0), NP (0.45)	N (1.0), Nom (0.95), NP (0.0475)

The following rules have been used for these first row:

Rules
N → flights
V → book
Aux → can
PN → Lufthansa
Pro → you
NP → PN
NP → Pro
Nom → N
NP → Nom

At the second level of the table, we will have combinations of two consecutive terminals symbols of words. Therefore we will have one less column:

can you	you book	book Lufthansa	Lufthansa flights

Now, "**can you**" gives the following possible combinations:

Rule	Combination
-	Aux, Pro
-	Aux, NP

However, there is no rule from which a non-terminal symbol produces them. Therefore, we can discard them as they have probability zero.

"**you book**" gives the following combinations:

Rule	Combination
-	Pro, V
-	NP, V

Again, there is no rule from which a non-terminal symbol produces any of these combinations. Therefore, we can discard them with probability zero.

"**book Lufthansa**" gives the following combinations:

Rule	Combination
-	V , PN
VP → V, NP (0.9)	V (1.0), NP (0.45)

Therefore, in this case there is a rule we can consider for our table.

Finally, **Lufthansa flights** has the following combinations:

Rule	Combination
-	PN, N
Nom → PN, Nom (0.1)	PN (1.0), Nom (0.95)
-	PN, NP
-	NP, N
-	NP, Nom
-	NP, NP
NP → Nom (0.05)	Nom (0.095)

In this case there is a rule we can consider for our table. The last row has been added as Nom can be produced by NP, and following the same idea as in the first row we should consider that rule as well. With this in mind, the second row of the table is:

can you	you book	book Lufthansa	Lufthansa flights
-	-	VP (0.405)	Nom (0.095), NP (0.00475)

Following the same idea, the third level is:

can you book	you book Lufthansa	book Lufthansa flights

Now, let's consider the first column **can you book**. This column can be considered in the following way **can (you book)**, **(can you) book** and **can you book**. The information in parenthesis is given by the previous table while the individual words can be taken from the first level of the table.

Rule	Combination
-	Aux, Pro, V
-	Aux, NP, V

There are no possible rules for any of these combinations.

The next column gives us these two possible groupings **you (book Lufthansa)**, **(you book) Lufthansa** and **you book Lufthansa**:

Rule	Combination
-	Pro, VP
S → NP,VP (0.8)	NP (0.5), VP (0.405)
-	Pro, V, PN
-	Pro, V, NP
-	NP, V, PN
-	NP, V, NP

There is a rule that produces one of the the sequences of non-terminals. Hence, we can include this rule to the table.

The final column gives us these possible combinations **book (Lufthansa flights)**, **(book Lufthansa) flights** and **book Lufthansa flights**. Let's see the set of rules that these combinations give us:

Rule	Combination
-	V, Nom
VP → V,NP (0.9)	V (1.0), NP (0.00475)
-	VP, N
-	VP, Nom
-	VP, NP
-	V, PN, N
-	V, PN, Nom
-	V, PN, NP
-	V, NP, N
-	V, NP, Nom
VP → V,NP,NP (0.1)	V (1.0), NP (0.45), NP (0.0475)

While there are two possible rules, we have to consider just the one with higher probability. Finally, this gives us the level 3 of the table:

can you book	**you book Lufthansa**	**book Lufthansa flights**
-	S (0.162)	~~VP (0.0021375)~~, VP (0.0045125)

Let's take a look to the level four of the table:

can you book Lufthansa	**you book Lufthansa flights**

The first column gives us the following combinations **can (you book Lufthansa)**, **(can you) (book Lufthansa)**, **(can you book) Lufthansa**, **can (you book) Lufthansa**, **(can you) book Lufthansa**, **can you (book Lufthansa)** and **can you book Lufthansa**:

Rule	**Combination**
S → Aux,NP,VP (0.2)	Aux (1.0), S (0.162)
-	Aux, VP
-	Aux, Pro, VP
S → NP VP (0.2)	Aux (1.0), NP (0.5), VP (0.405)
-	Aux, Pro, V, PN
-	Aux, NP, V, PN
-	Aux, Pro, V, NP
-	Aux, NP, V, NP

The second column gives the following combinations **you (book Lufthansa flights)**, **(you book) (Lufthansa flights)**, **(you book Lufthansa) flights**, **you (book Lufthansa) flights**, **(you book) Lufthansa flights**, **you book (Lufthansa flights)** and **you book Lufthansa flights**:

Rule	**Combination**
-	Pro, VP
S → NP,VP (0.8)	NP (0.5), VP (0.0045)
-	S, N
-	S, Nom
-	S, NP
-	Pro, VP, N
-	NP, VP, N
-	Pro, VP, Nom
-	NP, VP, Nom
-	Pro, VP, NP
-	NP, VP, NP
-	Pro, V, Nom
-	Pro, NP, Nom
-	Pro, V, NP
-	Pro, NP, NP
-	Pro, V, PN, N
-	Pro, V, PN, Nom
-	Pro, V, PN, NP
-	Pro, V, NP, N
-	Pro, V, NP, Nom
-	Pro, V, NP, NP
-	NP, V, PN, N
-	NP, V, PN, Nom
-	NP, V, PN, NP
-	NP, V, NP, N
-	NP, V, NP, Nom
-	NP, V, NP, NP

Therefore, the final row for the table is:

can you book Lufthansa	**you book Lufthansa flights**
~~S (0.0324)~~, S (0.0405)	S (0.0018)

The final state is given by the full sentence: **can you book Lufthansa flights**. This gives us many combinations and I will not include them in this notes.

Putting all together, the final table is:

can	**you**	**book**	**Lufthansa**	**flights**
Aux (1.0)	Pro (1.0), NP (0.5)	V (1.0)	PN (1.0), NP (0.45)	N (1.0), Nom (0.95),NP (0.0475)
-	-	-	VP (0.405)	Nom (0.095),NP (0.00475)
-	-	-	S (0.162)	VP (0.0045125)
-	-	-	S (0.0405)	S (0.0018)
-	-	-	-	S (0.00043)

6.3 Chomsky Normal Form

This algorithm can be simplified by using **Chomsky Normal Form** as a grammar such that every rule is of the form:

$$A \to BC$$

$$A \to a$$

It is known that for any CFG there is a weakly equivalent grammar in CNF(G). CYK is more efficient and easier to implement with a CNF grammar. To convert to a normal form:

1. Merge rules with a single non-terminal RHS. Example: Nom →N:0.95 and N→flights:1.0 can be merged to Nom → flights:0.95.

2. Split rules with more than two non-terminal RHSs. Example: S → Aux NP VP:0.2 can be split to S → Aux X:0.2 and X → NP VP:1.0

CYK algorithm can be adapted to generate language model scores (e.g check the parse tree probability given against the best parse tree that is given by CYK).

Algorithm 4: CYK Algorithm

Result: Most likely parse for a given sequence of words $w_1 ... w_n$
Set $t_{i,j,a} = -\infty$ for all values;
for *i = 1,...,n* **do**
  for $A \to w_i \in P$ **do**
    $t_{i,i+1,A} = D(A \to w_i)$;
  end
end
for *k=1,..,n; i=1,..n-k+1; j=i+k* **do**
  for $A \to \beta \in P$ **do**
    if *β matches between i and j* **then**
      $S = D(A \to \beta) \cdot \Pi_{i',j',A'} t_{i',j',A'}$ where {i',j',A} are the matches;
      if $s > t_{i,j,A}$ **then**
        $t_{i,j,A} = s$
      end
    end
  end
end

If we have a gold-standard corpus of known trees we can learn from this. PCFG probabilities can be obtained by counting:

$$D(A \to \beta) = \frac{c(A \to \beta)}{c(A)}$$

Note that smoothing techniques can be applied.

Finally, unsupervised learning can be used in order to calculate probabilities. Suppose we have the grammar but not the probabilities. We cannot just apply our grammar as there are many ambiguous parse. We can apply a method called the Inside-Outside algorithm, analogous to the forward-backward algorithm. Inside–outside algorithm is a way of re-estimating production probabilities in a probabilistic context-free grammar.

Let's see some of the common problems and solutions with PCFGs:

Lexical Dependencies

In a CFG the expansion of one non-terminal is independent of any other non-terminal. An example of these is from a study of "In Francis et. al (1999)." that found out that subjects are 91% pronouns and 9% lexical noun phrases. In contrast, "direct objects" are 34% pronouns and 66% lexical noun phrases. Ergo, we would need to distinguish between NP in subject and direct object positions which is not possible in the previous methodology.

Lexical Attachment

Lexical attachment is very word dependent. Example: 'He hit the man with a bat' vs 'He hit the man with a hat'. PCFGs cannot model lexical dependencies.

Parse Ambiguity

Sometimes PCFG do not succeed in solving certain kinds of parse ambiguity. Two parse trees from the same grammar can be used to generate a sentence by using the same exact rules. Therefore, both trees may have the same probability.

Formalisms for statistical parsing

Two solutions that can be used to the problems above are:

- Lexicalized PCFGs: Further differentiate non-terminals by associating them with lexical items. Leads to increase sparsity.
- Dependency grammars: Model links between individual words in the parse. Complex to combine with standard CFG.

In a **lexicalized grammar** each non-terminal is further distinguished with a terminal. Heuristic rules decide which non-terminal is the head of the parent. Lexicalizing the grammar creates more rules. It is common to make some partial independence assumption, e.g,

$$P(VP(built) \to V(built)NP((pyramids))) \approx P(VP(built) \to V(\cdot)NP(\cdot))P(VP(built)|NP(pyramids))$$

Dependency grammars are different to constituency grammars as: links are directed and between words, links have labels (e.g., 'subj') and they form an acyclic graph. Dependency grammars are especially good for languages with free word order however of course in practice is better to do CFG and dependency grammar simultaneously and to use one to correct the order.

6.3.1 Evaluation of Parsing

Parses are generally evaluated according to the PARSEVAL metrics:

$$\text{recall} = \frac{|\text{correct constituents in parse}|}{|\text{constituents in treebank}|}$$

$$\text{precision} = \frac{|\text{correct constituents in parse}|}{|\text{constituents in parse}|}$$

Where **Constituents** are the number of rules that we used correctly and **treebank** is a parsed text corpus.

In addition we report cross-brackets. This is where the parser outputs ((A B) C) but the candidate is (A (B C)). Modern parsers achieve 90% precision and recall and 1% cross-bracketing.

6.4 Comparison of LMs, HMMs and PCFGs

Language models, Hidden Markov Models and Probabilistic Context Free Grammars are special cases of probabilistic graphical models. They consist of:

- A set of observed variables X
- A set of hidden variables Y
- A set of parameters (probabilities) μ
- An independence assumption - this generates the structure of the model

Let's see some Language Models examples:

Bigram Language Model we have no hidden variables: $\mu = \{P(X_i|X_{i-1})\}$.

HMM have observed variables $\{X_n\}_n$ and hidden variables $\{y_n\}_n$.

PCFG has observed variables again as text $\{X_n\}_n$ but now our observed variables are actually the parse tree.

When we come to solving these we tend to end up with very similar solutions. So we want to find a probability of an observed sequence or the most likely hidden state. For learning problem, we have supervised and unsupervised problems. Supervised can be usually achieved by 'counting' while unsupervised models require more complex techniques such as E-M algorithm.

7 Semantic Analysis Tasks

Let's take a look to different tasks that can be applied for the analysis of semantics:

7.1 Word Sense Disambiguation

Word-sense disambiguation is an open problem concerned with identifying which sense of a word is used in a sentence.

Word senses are defined by a semantic lexicon such as WordNet. WordNet Senses uses 'synsets' to express word senses and Glosses which is similar to Data Labeling and in which it associates each sense to glosses (examples). A semantic lexicon defines word senses explicitly, by use of lists. Word senses can also be defined implicitly by use of clusters. Here we focus on explicitly defined word senses. Finally, WordNet offers Corpus Data which offers corpus that is annotated for senses.

Algorithm 5: Simplified Lesk

```
Result: Best sense of w
Set best_sense = most frequent sense for word;
Set max_overlap = 0;
Set context = set of words in sentence;
for sense in senses of word do
    signature = set of words in the gloss and examples of senses;
    overlap = COMPUTEROVERLAP(signature,context);
    if overlap > max_overlap then
        max_overlap = overlap;
        best_sense = sense;
    end
end
Return best_sense;
```

The same can be done using the Vector Space Model using binary vectors and the different words from the Glosses in order to describe each sense.

Next, let's take to examples of semi-supervised and unsupervised cases. Wikipedia can be used to extract the context of different senses for a word. This Wikipedia data can be used together with WordNet for semi-unsupervised learning.

WordNet can be used as a graph between a word and relationships of its sentences (Graph-Based WSD). This graph can be used to disambiguate words. Example, in the sentence 'She drank some milk'. This graph can be used to interpret the words drink and milk. This can be done by first identifying the shortest path between the different senses of each word and checking distances between the inside nodes.

7.2 Semantic Role Labelling

Semantic role labeling is the process that assigns labels to words or phrases in a sentence that indicates their semantic role in the sentence, such as that of an agent, goal, or result.

Examples of semantic roles for an entity are 'Agent','Experiencer','Instrument','Force','Theme','Result','Content', 'Instrument','Beneficiary','Source' and 'Goal'.

Semantic roles enable 'event identification' across different sentences, which can be very useful in applications such as Question Answering. They are not very well defined with different names and interpretations of roles across the literature. Automatic 'semantic role labeling' depends largely on supervised training with limited labeled data existing.

Algorithm 6: SemanticRoleLabeling

```
Result: Labeled Tree from a sequence of words
Set parse = PARSE(words);
for predicate in parse do
    for node in parse do
        feature_vector = EXTRACTFEATURES(node,predicate,parse);
        CLASSIFYNODE(node,feature_vector,parse)
    end
end
```

Features that can be extracted are predicate, voice (active,passive), position (before/after of classified constituent with regard to predicate), head word of classified constituent, phrase type, minimal path from classified constituent to predicate, sub-categorization for parent of predicate, etc.

7.3 Coreference Resolution

Coreference resolution is the task of finding all expressions that refer to the same entity in a text. One example of application of coreference resolution is in dialog systems. There are two main steps: Mention Detection and Mention Clustering. In Mention Detection identifies the possible candidates and in clustering identifies the relationships. There are three kind of mentions that needs to be identified in the first phase:

- Pronouns: I, you, it, she, him, ... (Part-of-speech tagger can be used to identify pronouns)
- Named Entities: person, location, organization,... (Named Entity Recognizer system can be used to identify named entities)
- Noun Phrases: "a dog, "the big fluffy cat stuck in the tree",... (Constituency (Phrase) parse can be used to identify noun phrases)

The features that can be used in clustering are: Recency (more recently mentioned entities are preferred), syntactic agreement, other syntactic constraints, semantic compatibility and grammatical role - prefer entities in the subject position.

8 Information Extraction

Information Extraction refers to the automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources. Some applications of IE are:

- Semantic Normalization: Normalize instances to a canonical form. Often for statistical purposes to aggregate data across variations (e.g., Ireland, IRL, IE, Eire $\rightarrow$ IRL)
- Database curation: It's the process of completing or correcting a database. An example of this would be to curate information on missing or incorrect attributes. IE provides a semi-automated support by checking a DB against related textual data.
- Fact Extraction: Extracts information by labeling and finding relations in the text. Examples of FE are Named Entity Recognition systems which recognizes entity types such as person, organization, location, time, etc.

Some Information Extraction approaches:

8.1 Lexical lookup

In lexical lookup, a database with common lexical variants for entities is used to match text to entities. A new database may be required for each application, therefore this is usually an expensive approach.

8.2 Rules or Hearst Patterns

The idea of Hearts Patterns is to match text with manually defined extraction patterns in order to identify entities and relations. These predefined rules are called 'Hearst Patterns'. Examples of Hearst Patterns are "X and other Y", "Y such as X", "Y including X", etc. Again, these rules need to be defined for each new application domain which is expensive labour/time-intensive. Additionally, expertise is required on domain semantics as well as domain-specific language.

8.3 Machine Learning

Supervised training uses a manual annotation as labeled data. A manual annotation is required for each new application domain. Additionally, an agreement between annotators may be low for complex entities/relations. Inter-Annotator Agreement (IAA) are measures that can be used to find the level of agreement between multiple annotators. An example is Cohen's kappa coefeficient:

$$\mathbf{k} = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}$$

where $Pr(a)$ is the observed agreement and $Pr(e)$ is the expected agreement.

An IAA lower than 0.21 is considered poor, while greater than 0.81 is considered high. In order to improve K we should adjust annotation guidelines for re-annotation and discuss disagreements between annotators.

Semi-supervised training can be used on 'weakly' labeled data on known examples to bootstrap annotation and subsequent supervised training. Known examples can be taken from existing databases or through 'distant supervision' on data sources such as Wikipedia.

Unsupervised ML for IE is very cheap to implement and maintain but the results can be hard to interpret.

8.4 Named Entity Recognition and Entity Linking

NER systems were developed to cover small sets of basic named entity types. The NE Annotation is called ENAMEX and is XML format. Another format for NE annotation is BIO which uses B for beginning, and span Inside/Outside (I/O). Optionally with entity type.

Beyond small set of basic NE types (Person, Location, Time, ...), we can use ground entities in large-scale knowledge graphs such as 'DBpedia' and 'Yago'. It significantly expands the number of entity types but increases level of ambiguity. In order to do entity linking we need to disambiguate. Context around the entity, popularity of the entity, coherence across difference entities are features that can be used to disambiguate. Features can be Text based or Graph-based (e.g., page rank, in/out degree, etc.).

8.5 Relation Extraction

Relation Extraction (RE) is the task of extracting semantic relationships from text, which usually occur between two or more entities. These relations can be of different types. E.g "Paris is in France" states a "is in" relationship from Paris to France.

The relation annotation can be presented as positive (there is a relation) and negative (there is no relation). 'Features' can be extrapolate to the same concept of 'positive' and 'negative'

Semi-Supervised Relation Extraction can be used without needing manual annotation or having to use IAA measures. System databases with examples and 'distant supervision' with Wikipedia can be used. DB seeds for relations provide only positive instances and annotation in text still needs to be established. "Distant supervision" is a learning scheme in which a classifier is learned given a weakly labeled training set (training data is labeled automatically based on heuristics / rules).

Unsupervised Relation Extraction does not require annotation or domain language. The usual processing is: Sentence Segmentation $\rightarrow$ tokenization $\rightarrow$ part of speech tagging $\rightarrow$ entity detection $\rightarrow$ relation detection. It's not easy to interpret results and the extractions may not have any semantic meaning. Knowledge Base Population (KBP) has a knowledge graph against which extraction results can be integrated. No knowledge graph in OpenIE.

8.6 Evaluation of IE

IE Evaluation metrics are derived from Information Retrieval: Precision, Recall and F-Score. Evaluation can be done against a "Gold Standard" that has human annotated data items.

$$\text{Precision} = \frac{\text{correctly extracted items}}{\text{Total extracted items}}$$

$$\text{Recall} = \frac{\text{correctly extracted items}}{\text{Total gold items}}$$

$$\text{F-Score} = \frac{2PR}{P+R}$$

9 Knowledge Graphs and Chatbots

A chatbot is a software application used to conduct an on-line chat conversation via text or text-to-speech, in lieu of providing direct contact with a live human agent.

9.1 Intent Classification

Intent classification is the automated association of text to a specific purpose or goal. Some examples of approaches for iuntent classification are:

Question/Answer Pairs

Defaulting the answer to the one matching more words from a predefined repository.

Semantic Matching

Semantic matching is a technique used in computer science to identify information which is semantically related. Example: using a vector space model

Knowledge Graph

Knowledge Graph can be used to provide different answers to more open questions in which the answer may have subclasses.

A knowledge graph is a knowledge base that uses a graph-structured data model or topology to integrate data. From a data science point of view, knowledge graphs are representations of semantic metadata with classes and properties that describe the 'meaning of data'. A taxonomy is a way to generalise this structure of classes and attributes. A formal representation can represent such a structure using Description logic. Description logics (DL) are a family of formal knowledge representation languages. General Knowledge Graphs can be found in: DBpedia, Global WordNet Association, BabelNet, Yago, WikiData, Google Knowledge Graph.

9.2 Taxonomy Extraction

Knowledge changes rapidly over time and between specific domains or enterprises. Costly to develop and maintain a knowledge graph manually. NLP techniques can be used in knowledge graph extraction from text. A knowledge graph comprises; classes, class instances, class properties, class hierarchy, other relation between classes.

Taxonomy extraction consists of extracting a set of terms from a corpus and organising them into a taxonomic structure. A taxonomy class corresponds to a single or multi-word 'term' (term extraction). A taxonomy relation corresponds to a pair fo terms {c,d} where c is more specific than d (term pair identification). A taxonomy corresponds to an optimal tree constructed from term pairs.

Term Extraction

Term Extraction can be done with Unsupervised Learning using Extract, rank and filter Noun Phrases. Unsupervised extraction term can build on syntactic analysis as a term often corresponds to a Noun Phrase (NP). Not all NPs are equally good term candidates. PMI can be used to rank NPs based on the correlative strenght between the individual words in the NP. Recall teh PMI formula used previously:

$$PMI(w_1, w_2) = \log(\frac{P(w_1, w_2)}{P(w_1)P(w_2)})$$

Supervised term extraction approaches train a classifier on manually labeled data (term annotation). General challenges in manual annotation apply also here: Inter-Annotator Agreement, cost of manual labour, renewed annotation for different domains. Instead of manual term annotation we can use Wikipedia 'anchors' as indicator for terms.

Term Pair Identification

Class hierarchy often corresponds to substrings, where a shorter, more general term is embedded in a longer, more specific term. Substring analysis depends on Noun Phrase analysis (nominal head). Recall that Hearst Patterns are used for IE, in particular for the acquisition of hypernyms from text. Recall that hypernyms are taxonomic relations between word senses in WordNet. Hearst patterns can therefore be used also directly in term pair identification. A different approach can be used using Clustering with a vector space model by constructing vectors for all terms (use pre-trained models), computing cosine similarity between vectors and building clusters over similar vectors. Labelling of the clusters can be unclear and therefore difficult to evaluate. The same approach can be done by using supervised training but in this case it can construct combined vectors for term pairs and optimize through supervised training to predict taxonomic relation between a given term pair. The supervised training on taxonomy pairs from existing taxonomies is based on Domain classifications, WordNet hypernyms and Wikipedia categorization.

Taxonomy Extraction

TE is an IE task. Therefore the use of IE evaluation metrics can be used (Precision, Recall and F-score). Using an existing taxonomy as Gold Standard (GS). Computer overlap between GS and extracted taxonomy on terms/pairs

10 Opinion Mining

Opinion mining, or sentiment analysis, is a text analysis technique that uses computational linguistics and natural language processing to automatically identify and extract sentiment or opinion from within text (positive, negative, neutral, etc.).

10.1 Sentiment Analysis

Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information.

Some of the challenges are in implicit, ambiguous and informal language. The classifier can be developed on the basis of a sentiment lexicon (unsupervised) or sentiment labeled data (supervised).

Using a sentiment lexicon provides a list of positive and negative words. Calculating the percentage of positive vs negative words can give a classification for the sentence. SentiWordNet adds sentiment polarity on words in WordNet synsets. Using techniques for Word Sense Disambiguation we can find the correct synset of the word. Other alternatives for lexicons are the Linguistic Inquiry and Word Count lexicon which

is more focus on emotions. Most widely used emotions are Happy, Sad, Fear, Anger, Surprise, Disgust. Plutchik is an example of a more complex emotion model or Lovheim's model which usees 3 continuous values related to neurotransmitters that have been interpreted as Valence, Aurosal and Dominance.

Using the previous methods such as feature selection and engineering, PoS-tagging, negation, etc. can help to improve the accuracy of the classifier.

Bias can be on gender, ethnic identity, social demography, geographical location, etc.... A data statement is a characterization of a dataset that provides context to allow developers and users to better understand what biases might be reflected in systems built on the software.