



Intelligent Systems – Lecture 11

Probabilistic Language Processing: Chapter 23

MSc 2008





Summary of the previous lecture

- Agents communicate with each other through utterances (speech acts) in a common language.
- A **language** consists of signs that convey meaning. A **grammar** describes the structure of allowed messages in a language.
- **Formal language theory** and **phrase structure grammars** can be used to deal with natural language.
- **Context-free** language sentences can be parsed in polynomial time by a **chart parser**.
- Augmenting grammars is a useful tool to deal with more sophisticated NL aspects.
- Logical inference can be used with **DCG**, parsing and semantic interpretation.



Communication: Speaker

Speaker

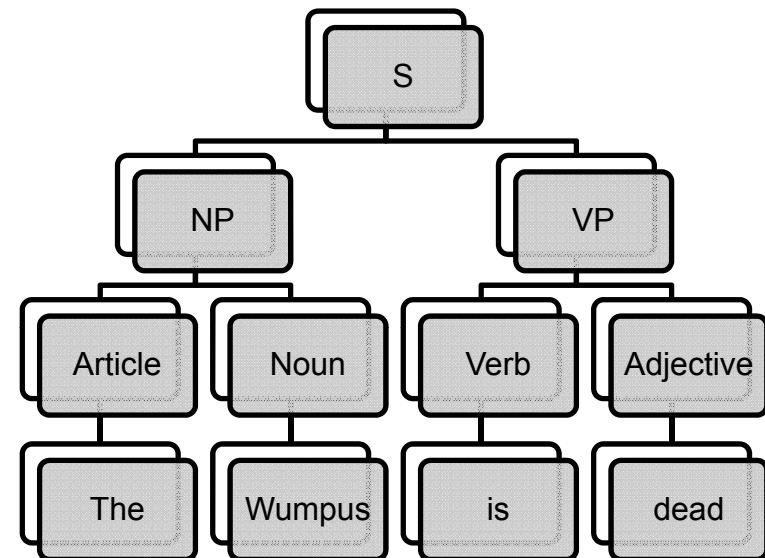
- **Intention**
 - Know (H, \neg ALIVE(Wumpus, S3))
- **Generation:**
 - The Wumpus is dead
- **Synthesis**
 - „thaxwahmpaxsihzdehd“



Communication: Hearer

Hearer

- Perception
 - The Wumpus is dead
- Analysis: **Parsing**
 - Parse tree
- Semantic interpretation
 - ⊢ ALIVE(Wumpus, Now)
 - TIRED (Wumpus, Now)
- Pragmatic interpretation
 - ⊢ ALIVE(Wumpus, S3)
 - TIRED (Wumpus, S3)
- Disambiguation
 - ⊢ ALIVE(Wumpus, S3)
- Incorporation
 - TELL (KB, ⊢ ALIVE(Wumpus, S3))





Language Models

- So far: **logical language models**
 - Grammars are used to determine whether a string is member or is not member of a language.
- Now: **probabilistic language models**
 - Define a probability distribution over a (possibly infinite) set of strings.
 - Use a corpus (collection of text), as well as statistics and learning.



Advantages of Probabilistic Language Models

- Can conveniently be trained from data - learning is just a matter of counting occurrences ~ computing probabilities
- More robust ~ can accept **any** string, albeit with a low probability.
- Reflect the fact that not 100% of speakers agree on which sentences are part of the language.
- Can be used for disambiguation.



N-gram models

- A probabilistic language model defines a probability distribution over a (possibly infinite) set of strings
- Unigram model – defines $P(w)$
 - Probability of a string is just the product of the probability of its words
- Bigram model – defines $P(w_i | w_{i-1})$
- n-gram model – defines $P(w_i | w_{i-(n-1)} \dots w_{i-1})$



Examples

- Unigram model
 - Logical are as are confusion a may right tries the agent goal the was diesel more object then information-gathering search is
- Bigram model
 - Planning purely diagnostic expert systems are very similar computational approach would be represented compactly using tic tac toe a predicate
- n-gram model
 - Planning and scheduling are integrated the success of naïve Bayes model is just a possible prior source by that time



Smoothing

- Bigram models over a data set can assign many word pairs with zero probability.
- Problem: Our model should not say that these pairs are impossible.
- Solution: Smoothing over the zero counts.
 - **Add-one smoothing** – add one to the count of every possible n-gram.
 - **Linear interpolation smoothing** – combining trigram, bigram and unigram by linear interpolation to get better estimates.



Smoothing (cont)

- Add-one
 - If there are N words in the corpus and B possible bigrams, then each bigram with an actual count of c is assigned a probability estimate of $(c+1)/(N+B)$
- Linear interpolation smoothing
 - Probability estimate: $P_{li}(w_i | w_{i-2}w_{i-1}) = c_3P_{li}(w_i | w_{i-2}w_{i-1}) + c_2P_{li}(w_i | w_{i-1}) + c_1P_{li}(w_i)$ where $c_1 + c_2 + c_3 = 1$
 - The parameters c_1, c_2, c_3 can be fixed or trained with an EM algorithm.



Evaluating language models

- Split corpus into a training corpus and a test corpus.
- Determine the parameters of the model from the training data.
- Calculate the probability assigned to the test corpus: **the higher the probability the better.**



Evaluating language models (cont)

- Problem: $P(\text{words})$ is quite small for long strings.
- Solution: Instead of probability compute **perplexity of model** on a test string of words:
 - $\text{Perplexity}(\text{words}) = 2^{-\log_2(P(\text{words}))/N}$ (N – Number of words)
 - An n-gram model assigning every word a probability of $1/k$ has a perplexity of k .
 - The lower the perplexity the better.



Using n-gram models

- Segmentation
 - It is easy to read words without no spaces
- This sentence can be easily decoded by a simple unigram word model (using a Viterbi algorithm).



Probabilistic context-free grammars (PCFGs)

- Problem: n-gram models have no notion of grammar at distances greater than n
- Solution: Probabilistic context-free grammars consists of a CFG wherein each rewrite rule has an associated probability



Example: PCFGs

$S \rightarrow NP VP$ [1.00]

$NP \rightarrow Pronoun$ [0.10]

| *Name* [0.10]

| *Noun* [0.20]

| *Article Noun* [0.50]

| *NP PP* [0.10]

$VP \rightarrow Verb$ [0.60]

| *VP NP* [0.20]

| *VP PP* [0.20]

$PP \rightarrow Preposition NP$ [0.10]

Noun → **breeze** [0.10] | **wumpus** [0.15] | **agent** [0.05] | ...

Verb → **sees** [0.15] | **smells** [0.10] | **goes** [0.25] | ...

Pronoun → **me** [0.05] | **you** [0.10] | **I** [0.25] | **it** [0.20] | ...

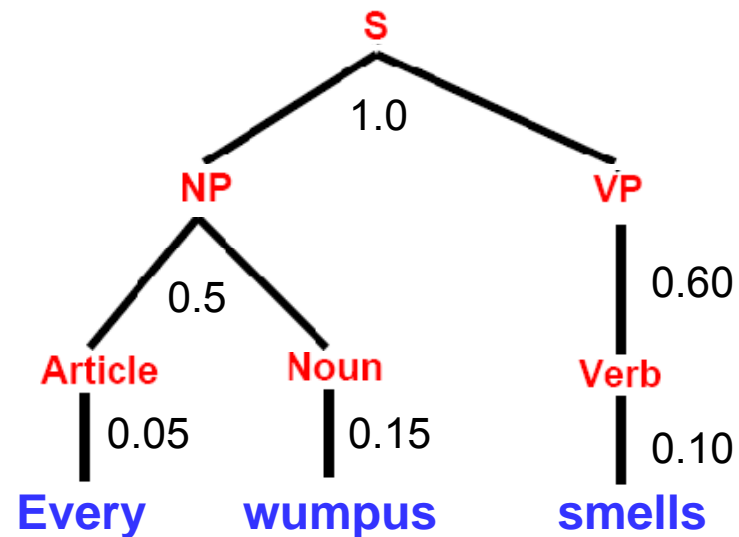
Article → **the** [0.30] | **a** [0.35] | **every** [0.05] | ...

Proposition → **to** [0.30] | **in** [0.25] | **on** [0.05] | ...



PCFGs

- In the PCFG model, the probability of a string $P(\text{words})$, is just the product of the probabilities of its parse trees.



- The probability of the whole tree is $1.0 \times 0.5 \times 0.05 \times 0.15 \times 0.60 \times 0.10 = 0.000225$



Disadvantages of PCFGs

- Problem with PCFGs is that they are context-free
 - The difference between $P(\textit{“eat a banana”})$ and $P(\textit{“eat a bandanna”})$ depends only on $P(\textit{“banana”})$ versus $P(\textit{“bandanna”})$ and not the relation between *“eat”* and the respective objects.
 - Possible solution is having some kind of context-sensitive model (**lexicalized PCFG**)
 - The most important word e.g. the noun plays a role in the probability.
- PCFGs tend to have a strong preference for short sentences.
 - Difficult to use in corpora which tend to use long sentences (e.g. Wall Street Journal).



Learning probabilities for PCFGs

- Creating a PCFG means constructing a CFG and then assign probabilities for each rule.
- Alternative approach: **learning** the grammar (and the associated probabilities)
 - From parsed corpus
 - Count trees
 - From an unparsed corpus
 - Rule structure known - use EM (inside-outside algorithm)
 - Rules unknown --Chomsky normal form



Information Retrieval (IR)

- Goal: **Google**. Find docs relevant to user's needs.
- A IR system has
 - A document collection
 - A query posed in a query language
 - A result set
 - A presentation of the result set.



IR

- Early IR systems used **Boolean keyword model**
 - *true* if a word occurs in document, *false* if it does not.
 - Easy to understand and implement.
 - Disadvantages
 - Relevance of document is single bit: no guidance how to order documents for presentation.
 - Boolean expressions can be unfamiliar to users.
 - Hard to formulate specific queries.



IR (cont)

- To date IR systems use models based on the **statistics of word counts**.
- Given a query we want to find the documents that are most likely to be relevant.
- $P(R = \textit{true} \mid D, Q)$
D is a document, Q is a query, R is a Boolean random variable indicating relevance.





Language Modeling

- Language modeling approach to decompose the joint distribution $P(R = \textit{true} \mid D, Q)$.

- If we take r to denote $R = \textit{true}$ then

$$\begin{aligned} P(r \mid D, Q) &= P(D, Q \mid r)P(r)/P(D, Q) \quad (\text{by Bayes' rule}) \\ &= P(Q \mid D, r)P(D \mid r)P(r)/P(D, Q) \quad (\text{by chain rule}) \\ &= \alpha P(Q \mid D, r)P(r \mid D)/P(D, Q) \quad (\text{by Bayes' rule,} \\ &\hspace{15em} \text{for fixed } D) \end{aligned}$$

- We rank the documents based on the score

$$\frac{P(r \mid D, Q)}{P(\neg r \mid D, Q)} = \frac{P(Q \mid D, r)P(r \mid D)}{P(Q \mid D, \neg r)P(\neg r \mid D)}$$



Language Modeling (cont)





Bag of words

- $P(r | D, Q)$ is the probability of that a document is relevant given a query
- To estimate the probability we must choose a language model of how queries are related to relevant documents.
- **Bag of words** (an unigram word model) – frequency of each word in document matters, not their order
 - “man bites dog” will behave the same as “dog bites man”
- To calculate the probability of a query given a relevant document, we just multiply the probabilities of the words in the query (naïve Bayes)





Evaluating IR systems

- Precision is proportion of results that are relevant.
- Recall is proportion of relevant docs that are in results
- ROC curve (there are several varieties) - standard is to plot false negatives vs. false positives.
- More “practical” for Web: reciprocal rank of first relevant result, or just “time to answer”



IR Refinements

- Case
- Stems
- Synonyms
- Spelling correction
- Metadata - keywords



Case folding and stemming

- If query is “couch” do not exclude “COUCH” or “couches”
- Sometimes it can decrease precision
 - stemming “stocking” to “stock”
- Stemming algorithms based on rules cannot avoid this problem (e.g. remove “-ing”) ; algorithms based on dictionaries can
- Stemming important in languages such as German
 - For words like
“Lebensversicherungsgesellschaftsangesteller”



Synonyms, spelling and metadata

- Recognizing synonyms
 - “sofa” = “couch”
- If applied too aggressively it can decrease precision
 - e.g. Searching for football player Tim Couch
- Spelling correction can be used to correct errors in both documents and queries
- Use metadata e.g. human-supplied keywords and hypertext links between documents



IR Presentation

- Give list in order of relevance, deal with duplicates.
- Provide relevance feedback to improve IR performance.
- Document classification – results are returned into a preexisting taxonomy of topics.



Document Clustering

- A tree of categories is created from scratch for each result set
- Clustering is an unsupervised learning problem
- **Agglomerative clustering** creates a tree of clusters going all the way down to individual documents
- **K-means clustering** creates a flat set of k categories



Agglomerative clustering

1. Consider each document as a cluster
2. Find the two closest clusters according to some distance measure
3. Merge them into one cluster
4. Repeat the process until one cluster remains



K-means clustering

1. Pick k documents at random to represent the k categories
2. Assign every document to the closest category
3. Compute the mean of each cluster and use the k means to represent the new values of the k categories
4. Repeat steps (2) and (3) until convergence



Implementing IR

- Two key data structures
 - The lexicon – all the words in the document collection.
 - Inverted index – lists all the places where each word appears in the document selection.



Lexicon

- Supports one operation: given a word, return the location in the inverted index that stores the occurrences of the word
- Implementation by a hash table
- Ignore words like “the”, “of”, “a”, “be” etc.
 - Keep them in lexicon to support phrases e.g. “to be or not to be”



Inverted index

- Consists of a set of ***hit places*** – places where each word occurs
- For phrase search must include the positions within each document where the word occurs
- Answering a query takes $O(H + R \log R)$,
R – size of desired result set
H – number of documents in the hit list





Vector model spaces

- Real IR systems use vector model spaces (VMS)
- VMS uses the same bag-of-words approach
- Each document and query is represented as a vector of unigram frequencies
- Relevant documents are selected by finding the document vectors that are nearest neighbors to the query vector



Information Extraction

- Goal: create structured information from textual documents
- Information extraction systems – mid-way between information retrieval systems and full-text parsers
- One looks for occurrences of a particular class of object or event and for relationships among those objects and events



Information Extraction (cont)

- Attribute – based system
 - Assumes that the entire text refers to a single object
 - The task is to extract attributes of that object
 - Can be built as a series of regular expressions, one for each attribute
- Example
 - “17in SXGA Monitor for only \$249.99”
 - $\exists m \text{ ComputerMonitor}(m) \wedge \text{Size}(m, \text{Inches}(17)) \wedge \text{Price}(m, \text{Dollar}(249.99)) \wedge \text{Resolution} \dots$



Information Extraction

- Relational – based extraction systems
 - Take into account the objects and their inter-relationships (e.g. €249.99 and the object that has that price)
 - Built using ***cascaded finite-state transducers*** – a series of FSAs where each one receives text as input, transduces, and passes it to the next one
- Example
 - FAUSTUS handles news stories about corporate mergers and acquisitions.





FAUSTUS - Steps

1. Tokenization – segments the stream of characters into tokens
2. Complex word handling – collocations and proper names – recognized by a combination of lexical entries and finite-state grammar rules
 - CapitalizedWord+ (“Company”|”Co”|”Ltd”|”Inc”)
3. Basic group handling – noun and verb groups
4. Complex phrase handling – combinations of basic groups
 - CompanyName acquires CompanyName
5. Structure merging



Summary

- Probabilistic language models based on n-grams recover a large amount of information about a language
- CFGs can be extended to probabilistic CFGs making it easier to learn them from data and easier to do disambiguation
- Information retrieval systems use a very simple language model based on bags of words, yet still manage to perform well in terms of recall and precision on very large data sets
- Information extraction systems use a more complex model that includes limited notions of syntax and semantics. They are typically implemented using a cascade of FSAs.



See you next week!

Thank you for your attention.

