Information Extraction and Question-Answering Systems

Foundations and methods

Dr. Günter Neumann
LT-Lab, DFKI
neumann@dfki.de

22/02/2002 1

What the lecture will cover

- Machine Learning for IE
- Lexical processing
- Evaluation Methods
- Basic Terms & Examples
- Parsing of Unrestricted Text
- Generic NL Core system
- Domain Modelling
- Question/Answering Core components
- Advanced Topics

22/02/2002 2

POS tagging

- Assigning morpho-syntactic categories to words in context

The	green	trains	run	down	that	track.
Det	Adj/NN	NNS/VBZ	NN/VB	Prep/Adv/Adj	SC/Pron	NN/VB
Det	Adj	NNS	VB	Prep	Pron	NN

- Disambiguation: a combination of lexical and local contextual constraints

POS tagging

- Major goal is to assign/select correct POS before syntactic analysis
  - Shallow processing
  - Handling of unknown words (robustness)
  - Reducing search space for next processing stages (parsing)
- Good enough for many applications
  - Information retrieval/extraction
  - Spelling correction
  - Text to speech
  - Terminology extraction/mining

POS tagging

- Corpus approach
  - ➢ NL text for which all correct POS are already assigned
  - ➢ Use it as a history for already made disambiguation decisions
- Major goal:
  - ➢ extract underlying rules/decisions so that new untagged corpus can be automatically tagged using the extracted rules

POS tagging

- Simple method:
  pick the most likely tag for each word
  - ➢ The probabilities can be estimated from a tagged corpus
  - ➢ Assumes independence between tags
  - ➢ Works pretty well (> 90% word tag accuracy)
  - ➢ But not good enough for processing of small NL text input: One in ten words wrong

Simple tagger example

- Brown corpus, 1M tagged wrds, 40 tags
- Example:
  - The representatives *put* the chairs on the table.
  - Word *put* occurs 41191,
    - 41145 times tagged as VBD
    - 46 times as NN
  - P(VBD | W=put) = 41145/ 41191 = 0.999
  - P(NN | W=put) = 46/ 41191 = 0.001
- Unknown words:
  - If *w is capitalized* then *tag(w)=NE* else *tag(w)=NN*

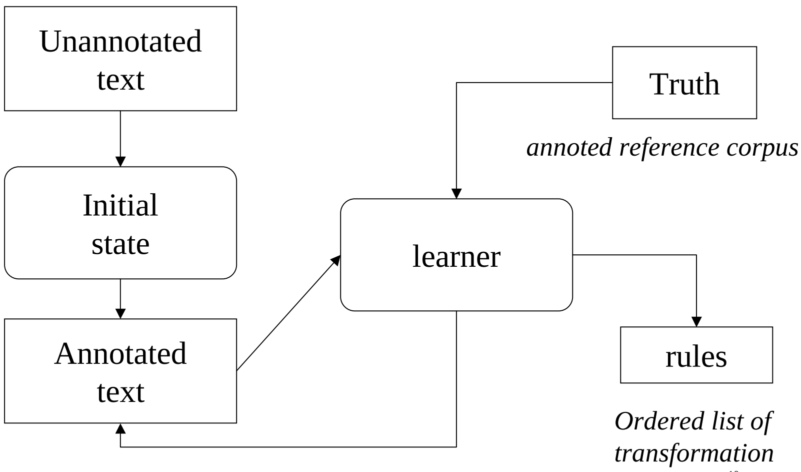
POS problems

- Long distance dependencies (*findet .... statt*),
- Annotation errors,
- not enough features, e.g., case assignment (Nom vs. Acc, *Er sieht das Haus*)
- If more features then more data sparseness problems
- Large corpora are needed

POS corpus approaches

- Rule based
  - ➢Transformation-based error-driven learning (Brill 95)
  - ➢Inductive logic programming (Cussens 97)
- Statistical based
  - ➢Markov models (TnT, Brants 00)
  - ➢Maximum entropy (Ratnaparkhi, 96)

Transformation-based error driven learning (Brill, 95)



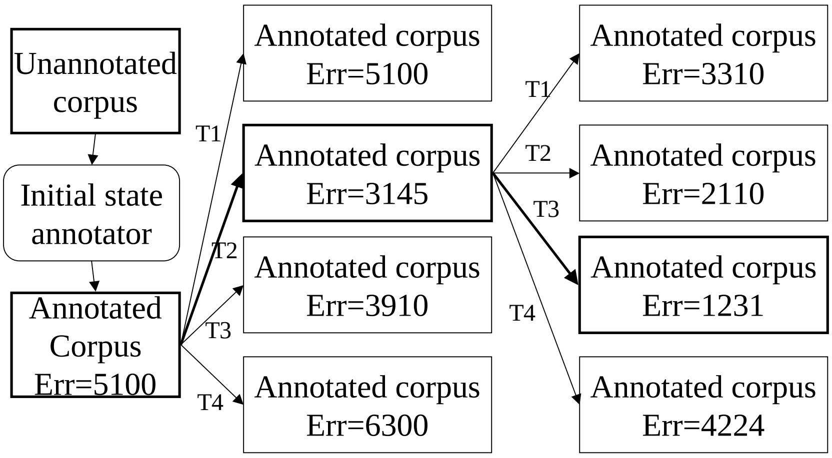
Structure of a transformation

- Rewrite rule
  - Change the tag from modal to noun
- Triggering environment
  - The preceding word is a determiner
- Application example
  - The/det can/modal rusted/verb.
  - The/det can/noun rusted/verb.

Generic learning method

- At each iteration of learning
  - Determine transformation t whose application results in the best score according to the objective function used
  - Add t to list of ordered transformations
  - Update training corpus by applying the learned transformation
  - Continue until no transformation can be found whose application results in an improvement to the annoated corpus
- Greedy search:
  - h(n) = estimated cost of the cheapest path from the state represented by the node n to a goal state
  - best-first search with h as its "eval" function.

Greedy search



Instances of TBL schema

- The initial state annotator
- Te space/structure of allowable transformations (patterns)
- The objective function for comparing the corpus to the truth
- Applications
  - Pos tagging
  - PP-attachment
  - Parsing
  - Word sense disambiguation

TBL POS tagging

- Initial annotator (our simple method)
  - Assign each word its most likely tag
  - Tag unknown words as proper noun if capitalized and common noun otherwise
- Error triple: <tag_a,tag_b,*number*>
  - *Number* of times tagger mistagged a word with tag_a when it should have been tagged with tag_b

Transformation patterns

Change tag *a* to tag *b* when

1. Preceding (following) word is tagged *z*
2. The word two before (after) is tagged *z*
3. One of the two preceeding (following) words is tagged *z*
4. One of the three preceeding (following) words is tagged *z*
5. The preceeding word is tagged *z* and the following word is tagged *w*
6. The preceeding (following) word is tagged *z* and the word two before (after) is tagged *w*

Learning task

- apply every possible transformation t
- count the number of tagging errors caused by t
- choose the transformation with the highest error reduction

Examples of learned transformation from PennWSJ

#	From	To	Condition
1	NN	VB	Previous tag is TO
2	VBP	VB	One of the prev. 3 tags is MD
3	NN	VB	One of the prev. 2 tags is MD
4	VB	NN	One of the prev. 2 tags is DT
5	VBD	VBN	One of the prev. 3 tags is VBZ
6	VBN	VBD	Prev. tag is PRP
7	VBN	VBD	Prev. tag is NNS
...			
16	IN	WDT	Next tag is VBZ
17	IN	DT	Next tag is NN

To/TO conflict/NN/VB

might/MD vanish/VBP/VB

might/MD not reply/NN/VB

Lexicalized transformations

- Change tag *a* to *b* when
  - The preceding (following) word is *w*
  - The word before (after) is *w*
  - ...
- Example (WSJ): $w_{i:}$ word at position i
  - From *IN* to *RB* if w_{i+2} = *as*
  - From *VBP* to *VB* is (w_{i-2} or w_{i-1}) = n‘t

Tagging Performance

Method	Tagging corpus size (words)	# rules or contex. Probs	Acc. (%)
Stochastic	64K	6,170	96.3
Stochastic	1M	10,000	96,7
TBL With Lex. rules	64	215	96,7
TBL With Lex. rules	600K	447	97.2
TBL W/o Lex. rules	600K	378	97

Closed vocabulary assumption:
All possible tags for all words in the test set are known

22/02/2002 19

Handling unkown words

- Change the tag of an unknown word (from *X*) to *Y* if
  - Deleting prefix (suffix) *x*, $|x| \leq 4$, results in a (known) word
  - The first (last) 1,2,3,4 characters of the word are *x*
  - Adding character string *x* as prefix (suffix) results in a word
  - Word *W* ever appears immediately to the left (right) of the word
  - Character *Z* appears in the word

22/02/2002 20

Some example transformations

#	From	To	Condition
1	NN	NNS	Has suffix -s
2	NN	CD	Has character .
3	NN	VBN	Has suffix -ed
4	??	RB	Has suffix -ly
5	NNS	VBZ	Word it appears to the left
...			

Tagging performance:
Unknown word accuracy on WSJ test corpus:
82.2%
Overall accuracy:
96.6%

Unsupervised TBL

- Goal: automatically train a rule-based POS tagger without using a manually tagged corpus
- Source: a dictionary listing the allowable parts of speech for each word
- Challenge: define an objective function for training that does not need a manually tagged corpus as truth

Core idea

- Note that for ambiguous words we can only randomly choose between the possible tags
  - *The can will be crushed*
- Using an unannotated corpus and a dictionary, we could discover, that of the words that appear after *The* that have only one possible tag, nouns are most common
  - Change *tag* of a *wrd* from (MD | NN | VB) to NN if the previous word is *The*

Transformation Templates

- Change the tag of a word from χ to Y in context C if:
  1. The previous tag is T
  2. The previous word is W
  3. The next tag is T
  4. The next word is W
- Different use of transformation: reduce uncertainty instead of changing one tag to a another $\Rightarrow Y \in \chi$

Examples of transformations

- Change the tag:
  - From NN|VB|VBP to VBP if the previous tag is NSS
  - From NN|VB to VB if the previous tag is MD
  - From JJ|NNP to JJ if the following tag is NNS

Scoring criterion

- Learner has no gold standard training corpus with which accuracy can be measured
- Instead: use information from the distribution of unambiguous words

- Initially, each word in the training corpus is tagged with all tags allowed for that word
- In later learning iterations, training set is transformed as a result of applying previously learned transformations

Computation of the score of a tranformation

- For each tag Z ∈ χ, Z≠Y, compute *freq(Y)/freq(Z)*incontext(Z,C)*
  - *freq(Y)* = # occurences of words unambiguously tagged with Y;
  - the same for *freq(Z)*
  - *incontext(Z,C)* = # times a word unambig. tagged as Z occurs in C
- Let: $R=argmax_z freq(Y)/freq(Z)*incontext(Z,C)$
- Then *Change the tag of a word from χ to Y in context C* is:
  - *Incontext(Y,C)-freq(y)/freq(R)*incontext(R,C)*

- Computing the difference between the number of unambiguous instances of tag Y in context C and the number of unambiguous instances of the most likely tag R in context C, where R∈ χ, R≠Y. Choose the transformation which maximizes this function.

Evaluation results

- When tagset of a word not fully disambiguated, choose randomly a single tag
- Results (Training/Test)
  - PennTB (120K wrds/200T wrds): 95.1%, 1,151 rules learned
  - BrownTB (350T wrds /200T wrds): 96.0%, ~1,729 rules learned

Final remarks

- Weakly supervised rule learning:
  - ➢Best score: 96.8% using 88,200 corpus (better than supervised on same corpus)
- Further issues
  - ➢Rules can be converted into deterministic FST: O(n), independent of # rules (cf. Roche&Schabes, 1995)
  - ➢Get the source code for free

Main Information Sources for statistical POS Tagging

- Paradigmatic information - the distribution of tags for the word in isolation: P(t|w)
  - ➢ P(VBD|W=put) = 41145/ 41191 = 0.999
  - ➢ P(NN|W=put) = 46/ 41191 = 0.001
- Syntagmatic information - look at the tags of other words in the context of the word we are interestd in
  - ➢ *a new play*: AT JJ NN versus AT JJ VBP

Statistical POS tagging

- We would like to have a statistical model that would be able to take into account both types of information in a principled way
  - ➢Learn that information from some annotated corpus (i.e., from examples, observations)
  - ➢Keep the syntagmatic context as local as possible

22/02/2002 31

Brief excursion: probability theory (1)

- $0 \leq P(A) \leq 1$
- $P(A+B)= P(A) + P(B)$, if A & B are independent
- $P(A+\neg A) = P(A) + P(\neg A)$, ¬ A is the negation of A
- $P(A) + P(\neg A) = 1$

22/02/2002 32

Brief excursion: probability theory (2)

- Let $P(A)=k/N$, $P(B)=l/N$, $P(AB)=m/N$
- Then $P(B|A) = P(AB)/P(A) = m/k$
- Chain rule: $P(A_n \mid A_1 \ldots A_{n-1}) = \dfrac{P(A_1 \ldots A_{n-1} A_n)}{P(A_1 \ldots A_{n-1})}$
- Bayes rule: $P(B \mid A) = \dfrac{P(B) P(A \mid B)}{P(A)}$

Brief excursion: probability theory (3)

- Maximum Likelihood Estimates

$$P_{MLE}(W_1 \ldots W_n) = \frac{C(W_1 \ldots W_n)}{N}$$

$$P_{MLE}(W_n \mid W_1 \ldots W_{n-1}) = \frac{C(W_1 \ldots W_n)}{C(W_1 \ldots W_{n-1})}$$

N-Gram Model of Language

- Predict a word or properties of a word on the basis of already observed sequences of words

  Morgen gehe ich ins ... (Kino/Theater/...)

  $P(w_n \mid w_1,\ldots, w_{n-1})$
- Question:
  - How many words should we look back?
- Markov assumption:
  - Only a few is enough
- N-gram: consider a context of length (N-1)
  - unigram (no context),
  - bigram (previous word),
  - trigram (last previous words)

22/02/2002 35

Markov Models

- A system
  - which may be described at any time as being in one of N distinct states
  - At each discrete time step, the system undergoes a change of state according to a set of probabilities associated with the state
- Let $Q = (q_1, \ldots, q_T)$ be a sequence of random variables taking values in some finite set $S=\{s_1,\ldots,s_N\}$. Markov properties:
  - Limited horizont: a word's tag only depends on the previous word's tag (order 1 Markov Model)

    $$P(q_{t+1}=s_k \mid q_1, \ldots, q_t) = P(q_{t+1}=s_k \mid q_t)$$
  - Time invariant: tag probabilities don't change over time

    $$\forall t,k : P(q_t = s_i \mid q_{t-1} = s_j) = P(q_{t+k} = s_i \mid q_{t+k-1} = s_i)$$

22/02/2002 36

Markov Model

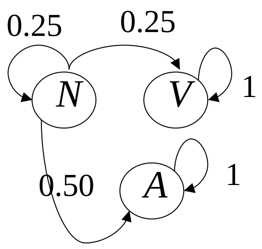
Stochastic transition matrix:

$$a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i),$$
$$a_{ij} \geq 0, \forall i, j,$$
$$\sum_{j=1}^{N} a_{ij} = 1, \forall i$$

Probs for initial states:

$$\pi_i = P(q_1 = s_i), \sum_{i=1}^{N} \pi_i = 1$$

Markov model as
a probabilistic FSA:



Hidden Markov Model

- Observation/Output is a probabilistic function; doubly embedded stochastic process
  1. Probability of observation (output sequence)
  2. Probability of internal process (internal state sequence), which are not observable (*hidden*); Indirectly observable only through 1.

- Thus a HMM λ is characterized by a tripple (A,B, π), with
  1. A, a matrix of transition probabilities
  2. B, a matrix of observation (emission) probabilities
  3. π, a vector of initial probabilities

Example HMM

In an HMM, *any* sequence of states can generate *any* sequence of observations with a given probability.

$$A = \begin{pmatrix} 0.3 & 0.5 & 0.2 \\ 0 & 0.3 & 0.7 \\ 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \\ 0 & 1 \end{pmatrix} \quad \pi = \begin{pmatrix} 0.6 \\ 0.4 \\ 0 \end{pmatrix}$$

Hidden Markov Models

- Prototypical tasks to which HMMs are applied include the following. Given a sequence of signals $O = \{o_1, \ldots, o_n\}$:
  - Determine the most probable state sequence that can give rise to this signal sequence. (Viterbi algorithm)
  - Determine the set of model parameters $\lambda = (A, B, \pi)$, maximizing the probability of this signal sequence. (Baum-Welch algorithm)
- Part-of-speech tagging is an example of the first task and training an HMM is an example of the second.

HMM (cont.)

- Ergodic model: every state of an HMM is connected with all other states
- HMM and POS
  - Observable sequence: words
  - Hidden sequence: part-of-speech
- Thus seen it is assumed that a string α was generated from some hidden POS sequence; therefore, the major goal is to determine the most probable POS sequence which could have generated α.

22/02/2002 41

Markov Models for POS Tagging

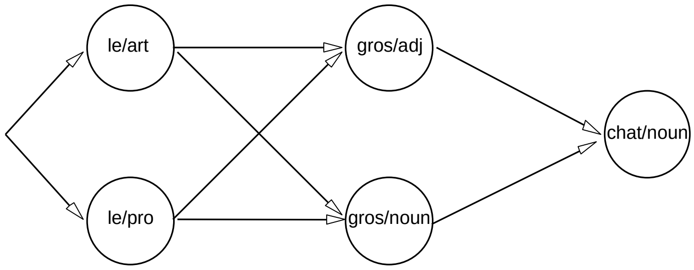
- Given a string of words $w_{1,n}$ find a sequence of tags $t_{1,n}$ that maximizes $P(t_{1,n} \mid w_{1,n})$
- Using Bayes Rule:

$$P(t_{1,n} \mid w_{1,n}) = \frac{P(w_{1,n} \mid t_{1,n}) * P(t_{1,n})}{P(w_{1,n})}$$

- We need to find $t_{1,n}$ that maximizes the numerator

22/02/2002 42

Example



1. Pr(art | ∅) . Pr(adj | art, ∅) . Pr(noun | adj, art) . Pr(*le* | art) . Pr(*gros* | adj) . Pr(*chat* | noun)
2. Pr(pro | ∅) . Pr(adj | pro, ∅) . Pr(noun | adj, pro) . Pr(*le* | pro) . Pr(*gros* | adj) . Pr(*chat* | noun)
3. Pr(art | ∅) . Pr(noun | art, ∅) . Pr(noun | noun, art) . Pr(*le* | art) . Pr(*gros* | noun) . Pr(*chat* | noun)
4. Pr(pro | ∅) . Pr(noun | pro, ∅) . Pr(noun | noun, pro) . Pr(*le* | pro) . Pr(*gros* | noun) . Pr(*chat* | noun)

Two main independence assumptions

- Words are independent of each other

$$P(w_{1,n} \mid t_{1,n}) = P(w_1 \mid t_{1,n}) * P(w_2 \mid t_{1,n}) * \ldots * P(w_n \mid t_{1,n})$$

- The probability of a word depends only on its tag (*bigram tagging*)

$$P(w_i \mid t_{1,n}) = P(w_i \mid t_i)$$

Probability model for $P(t_1,\ldots,t_n)$

- Breakdown of joint probability:

  $P(t_1,\ldots,t_n)=$
  $P(t_n \mid t_1,\ldots,t_{n-1}) * P(t_{n-1} \mid t_1,\ldots,t_{n-2}) * \ldots * P(t_1)$
- First order Markov Model:
  $P(t_n \mid t_1,\ldots,t_{n-1}) = P(t_n \mid t_{n-1})$
- Thus we get:
  $P(t_1,\ldots,t_n) = P(t_n \mid t_{n-1}) * P(t_{n-1} \mid t_{n-2}) * \ldots * P(t_1)$
- With a simple Markov model we need to find the POS sequence $P(t_1,\ldots,t_n)$ that maximizes

$$\prod_{i=1}^{n} P(w_i \mid t_i) * P(t_i \mid t_{i-1})$$

22/02/2002 45

Two main questions

- How do we obtain (train) these probabilities?
  - Calculate estimated probabilities based on frequency counts from a corpus
  - Smooth the estimated probabilities to avoid anomalies due to data sparseness
- How do we efficiently find the sequence of the tags that maximizes the joint product?
  - Viterbi algorithm

22/02/2002 46

Maximum Likelihood Estimation

- Back to our simple example: The representative put . . .
  AT NN VBD/NN
- *P(VBD|put,NN) = P(put|VBD)P(VBD|NN)*
  In the training data, we find 41,145 occurrences of put as a verb out of 7,305,323 verbs, so *P(put|VBD) = 41145/7305323 = 0.0056*
- And, out of 4,236,041 occurrences of the tag NN, the tag VBD comes next 389,612 times: *P(VBD|NN) = 389612/4236041 = 0.092*
- Combining
  - P(VBD|put,NN) = 41145/7305323 × 389612/4236041 = 0.00052
  - P(NN|put,NN) = 46/4236041 × 717415/4236041 = 0.0000018

Smoothing of the Probabilities

- Data sparseness is always a problem when estimating probabilities based on a corpus of data
- The „adding one" smoothing technique: add a count of one to all events, so that there are no zero probabilities

$$P(w_{1,n}) = \frac{C(w_{1,n}) + 1}{N + B}$$

C: absolute frequency of x
N: number of training instances
B: number of different types

Smoothing of the Probabilities

- Linear interpolation methods can compensate for data sparseness with higher order models. A common method is interpolating trigrams, bigrams and unigrams:

$$P(t_i \mid t_{1,i-1}) = \lambda_1 P_1(t_i) + \lambda_2 P_2(t_i \mid t_{i-1}) + \lambda_3 P_3(t_i \mid t_{i-1,i-2})$$

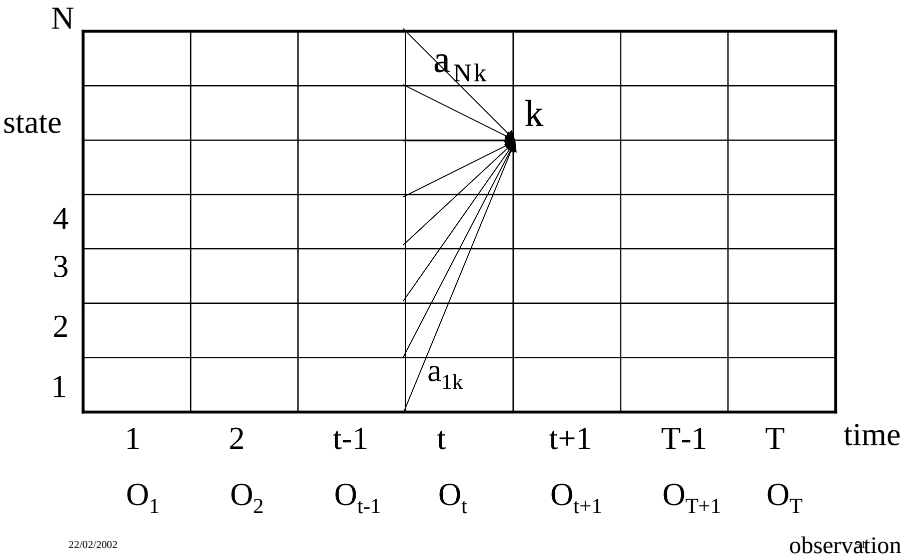
$$0 \le \lambda_i \le 1, \sum_i \lambda_i = 1$$

- Usually, the lambda values are automatically determined using a variant of the Expectation Maximation algorithm.

Pseudo code for viterbi algorithm

- Incremental left-to-right search using dynamic programming
- $\delta_i(j)$ is the probability of the most likely sequence of tags that ends with word i having the tag t_j
- $\psi_{i+1}(j)$ gives us the most likely tag at word i given that we are in state j at word $i+1$
- Induction
  - Find probability of most likely sequence that ends with word $i+1$ having tag j, by considering all possible tags for previous word i: $\delta_{i+1}(j) = max_{1 \le k \le T}[\delta_i(k) * P(t^j \mid t^k) * P(w_{i+1} \mid t^j)]$
  - For specific tag t^k that resulted in the above max, set: $\psi_{i+1}(j) = t^k$
- Start with $\delta_1(period)=1$, $\delta_1(t)=0$ for all $t \neq period$
- At the end find the tag t for which $\delta_{n+1}(t)$ is the greatest
- Reconstruct the most likely sequence of tags by backtracking: $X_{n+1}=t$, *for i=n downto 1 do:* $X_i = \psi_{i+1}(X_{i+1})$

Core Idea of Viterbi Algorithm



Further Issues with Markov Model Tagging

- Unknown words are a problem, since we don't have the required probabilities. Possible solutions:
  - assign the word probabilities based on corpus-wide distributions of POS
  - use morphological cues (capitalization, endings) to assign a more calculated guess
- Using higher order Markov Models:
  - using a trigram model for POS captures more context and is thus potentially a better probability model
  - However, data sparseness is much more of a problem
  - The Viterbi search for trigrams is more complex

TnT – Trigrams‘s Tags

- Efficient statistical part-of-speech tagger developed by Thorsten Brants, ANLP-2000
  - Stochastic models for English (Susanne corpus) and German (Negra corpus)
  - Performance:
    between 30,000 and 60,000 tokens per second on a Pentium 500 running Linux.
- TnT is a trainable tagger (you can use it with your own annotated corpus)
- Home page
  - http://www.coli.uni-sb.de/~thorsten/tnt/
  - Online testing of TnT

Example

Input	Output		Extended Output
Der	ART		| ART 1.000000e+00
Mandolinen-Club	NN	*	| NN 1.000000e+00 *
Falkenstein	NE	*	| NE 8.001280e-01 NN 1.998720e-01 *
und	KON		| KON 1.000000e+00
der	ART		| ART 1.000000e+00
Frauenchor	NN	*	| NN 9.828203e-01 NE 1.717975e-02 *
aus	APPR		| APPR 1.000000e+00
dem	ART		| ART 1.000000e+00
sächsischen	ADJA		| ADJA 1.000000e+00
Königstein	NN		| NN 7.762892e-01 NE 2.237108e-01
gestalten	VVINF		| VVINF 1.000000e+00
die	ART		| ART 9.796126e-01 PRELS 1.443545e-02 PDS 5.951974e-03
Feier	NN		| NN 1.000000e+00
gemeinsam	ADJD		| ADJD 1.000000e+00
.	$.		| $. 1.000000e+00

Accuracy

Corpus	*Language*	*Domain*	*Size*	*Accuracy*
NEGRA Corpus	German	Newspaper	350,000	96,7%
PennTB	English	Newspaper	1,200,00	96,7%
Susanne Corpus	English	Mixed	150,000	96,6%

Underlying Model

$$\arg\max_{t_1 \ldots t_T} \prod_{i=1}^{T} P(t_i \mid t_{i-1}, t_{i-2}) P(w_i \mid t_i) \mid P(t_{T+1} \mid t_T)$$

- Trigram modelling
  - The probability of a POS only depends on its two preceding POS
  - The probability of a word appearing at a particular position given that its POS occurs at that position is independent of everything else
- Explicit consideration of punctuation through special tags
  - $t_{-1,}$ t_0 beginning of sentence
  - t_{T+1} end of sentence
- Tagging is performed by a variant of the Viterbi algorithm (using beam search)

Training

- Maximum likelihood estimates

$$Unigrams \quad : \hat{P}(t_3) = \frac{c(t_3)}{N}$$

$$Bigrams \quad : \hat{P}(t_3 \mid t_2) = \frac{c(t_2, t_3)}{c(t_3)}$$

$$Trigrams \quad : \hat{P}(t_3 \mid t_1, t_2) = \frac{c(t_1, t_2, t_3)}{c(t_2, t_3)}$$

$$Lexical \quad : \hat{P}(w_3 \mid t_3) = \frac{c(w_3, t_3)}{c(t_3)}$$

- Smoothing: context-independent variant of linear interpolation   all trigrams get the same λs

$$P(t_3 \mid t_1, t_2) = \lambda_1 \hat{P}(t_3) + \lambda_2 \hat{P}(t_3 \mid t_2) + \lambda_3 \hat{P}(t_3 \mid t_1, t_2)$$

Smoothing algorithm

- Set λ_i=0
- Foreach trigram $t_1\ t_2\ t_3$ with $f(t_1, t_2, t_3) > 0$
  - Depending on the max of the following three values:
    - Case $(f(t_1, t_2, t_3)-1) / f(t_1, t_2)$: incr λ_3 by $f(t_1, t_2, t_3)$
    - Case $(f(t_2, t_3)-1) / f(t_2)$: incr λ_2 by $f(t_1, t_2, t_3)$
    - Case $(f(t_3)-1) / N-1$: incr λ_1 by $f(t_1, t_2, t_3)$
  - End
- End
- Normalize λ_i

-1 means: taking unseen data into account

Handling of unknown words

- Suffix analysis:
  - The probability distribution for a particular suffix is generated from all words in the training set that share the same suffix
  - Suffix here means „final sequence of characters of a word“, which is not necessarily linguistically meaningful
- Suffix length in TnT: use the longest suffix that can be found in the training set (>1)

Suffix estimation

- Calculate the probability of a tag *t* given the last *i* letters *l* of an *n* letter word

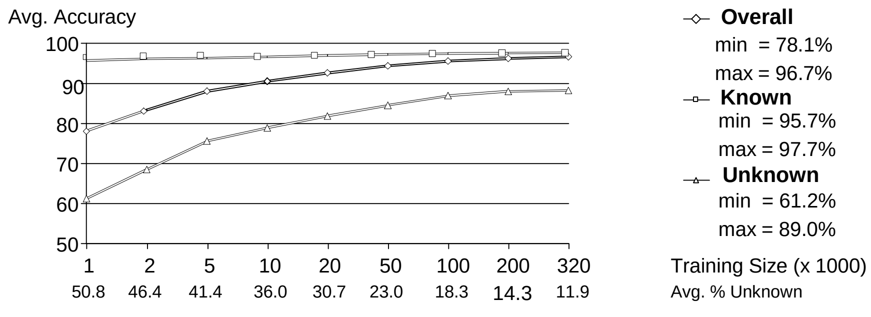
$$\hat{P}(t \mid l_{n-i+1}, \ldots, l_n) = \frac{c(t, l_{n-i+1}, \ldots, l_n)}{c(l_{n-i+1}, \ldots, l_n)}$$

- Smoothing: successive abstraction through sequences of increasingly more general contexts (i.e., omit more and more characters of the suffix)

Evaluation

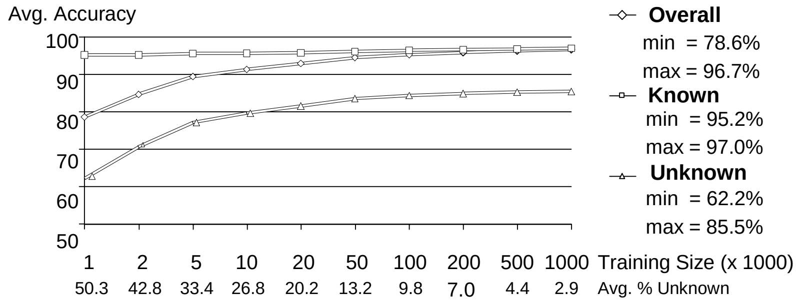
- Evaluation using two corpora:
  - NEGRA Corpus: Frankfurter Rundschau (German newspaper texts)
  - Penn Treebank: Wall Street Journal
- Disjoint training and test parts, 10-fold cross-validation
- Tagging accuracy: percentage of correctly assigned tags when assigning one tag to each token
- Tagging accuracy depending on the size of the training set
- Tagging accuracy depending on the existence of alternative tags within some beam (reliable vs. unreliable assignments)

***Part-of-Speech Tagging with TnT: NEGRA** Corpus*



NEGRA corpus: 350,000 tokens newspaper text (Frankfurter Rundschau)
randomly selected training (variable size) and test parts (30,000 tokens)
10 iterations for each training size; training and test parts are disjoint
No other sources were used for training.

Part-of-Speech Tagging with TnT: Penn Treebank



Penn Treebank: 1,2 million tokens newspaper text (Wall Street Journal)
randomly selected training (variable size) and test parts (100,000 tokens)
10 iterations for each training size; training and test parts are disjoint.
No other sources were used for training.

Additonal remarks

- Procedure of suffix handling is restricted to words with a certain frequency threshold (< 10)
- Capitalization information
- Beam search variant of Viterbi algorithm
  - Instead of considering all possible states during one iteration, only consider states whose values pass a certain predefined threshold θ
  - But note: now, it is not guaranteed that path with the highest probability is found
  - Brants shows that setting θ appropriately doubles performance speed without affecting the accuracy