PoS Tagging· June 2, 2009

Text Annotation

Beáta B. Megyesi
beata.megyesi@lingfil.uu.se

Goal

- What are the main components used for grammatical annotation?
- How do we get running texts morho-syntactically annotated?
- What methods are used by computational linguists for grammatical tagging?
- How can we measure the correctness of the annotation?

Components of grammatical annotation

- Running text
- Morphological segmentation, lemmatisation (start-ed, start)
- Part-of-speech tagging: to annotate tokens with their correct PoS (start/V)
- Chunking: to find non-overlapping group of words (NP: a nice journey PP: to NP: Vinstra)
- Syntactic parsing: to recover the complete syntactic structure

Overview

- Preparing text for grammatical annotation
- Methods for part-of-speech tagging
- Tagger evaluation
- Summary
- About the assignment

Preparing text for annotation

- Grammatical annotations are usually added to words and also to punctuation marks (period, comma)
- Tokenisation (1)
  - segmenting running text into words/tokens and
  - separating punctuation marks from words
  - white space marks token boundary, but not sufficient even for English:
  - ”Book that flight!”, he said.
  - Treat punctuation as word boundary:
  - ” Book that flight ! ” , he said .

Preparing text for annotation

- Tokenisation (2)
  - Punctuation often occurs word internally
  - Examples: Ph.D., google.com, abbreviations (e.g.), numeral expressions: dates (06/02/09), numbers (25.6, 100,110.10 or 100.110,10)
  - Clitic contractions marked by apostroph: we’re - we are
  - Apostroph also as genitive case marker: book’s
  - Multiword expressions (White house, New York, etc) cen be also handled by a tokenizer by using a multiword expression dictionary - Named Entity

Recognition (NER)

Preparing text for annotation

- Grammatical annotation is usually carried out on the sentence level
- Sentence/utterance segmentation (1)
  - segmenting a text into sentences is based on punctuation
  - certain kinds of punctuation (period, question mark, exclamation point) tend to mark sentence boundary
  - relatively unambiguous markers: ?, !

Preparing text for annotation

- Sentence/utterance segmentation (2)
  - Problematic: period as ambiguous between sentence boundary marker and a marker of abbreviations (Mr.) or both (This sentence ends with etc.).
  - Disambiguating end-of-sentence punctuation (period, question mark) from part-of-word punctuation (e.g., etc.)
  - Sentence segmentation and tokenization tend to be addressed jointly

Preparing text for annotation

- Sentence tokenization methods
  - build a binary classifier that decides if a period is part of the word, or is a sentence boundary marker
  - State-of-the-art methods are based on machine learning but many people use regular expressions
  - Grefenstette (1999) Perl word tokenization algorithm:
    1. separate unambiguous punctuation: ?, (,)
    2. segment commas unless they are inside numbers
    3. disambiguate apostrophs and pull off word-final clitics
    4. periods are handled by abbreviation dictionary

Preparing text for annotation

They
neither
liked
nor
disliked
the
Old
Man
.

The
...

Methods for annotation

- Manual:
  - time consuming, expensive
  - lack of consistency
- Automatic:
  - fast
  - consistent errors
  - methods: rule-based, data-driven or combinations

Rule-based

- a set of rules
- requires expert knowledge
- 60s-90s
- tokenization, morphological segmentation, tagging, parsing

Data-driven methods

- automatically build a model
- require data
- easy to apply to new domains
- fast, effective and robust
- can combine systems: consensus, majority

Machine learning

- automatic learning of structure given some data
- data-driven/corpus-based methods
- given some example learn the structure
- supervised vs unsupervised learning
- symbiotic relation between corpus development and data-driven classifier
- many different types of ML algorithms

Data-driven methods within NLP

- Transformation-based error-driven learning (Brill 1992)
- Memory-based learning (Daelemans, 1996)
- Information-theoretic approaches:
  - Maximum entropy modeling (Ratnaparkhi, etc)
  - Hidden Markov Models (Charniak, Brants, etc)
- Decision trees (Quinlan, Daelemans)
- Inductive Logic Programming (Cussens)
- Support Vector Machines (Vapnik, Joachims, etc.)

Machine learning in NLP

- Applications:
  - PoS tagging
  - chunking
  - parsing
  - semantic analysis (word sense disambiguation)
- Languages: 90s - Western European languages
- Today: Arabic, Chinese, Hungarian, Japanese, Turkish, ...

Part-of-Speech (PoS) tagging

- Goal: to assign each word a unique part-of-speech
- CONtent/N or conTENT/A (e.g. TTS, SR, parsing, WSD)
- PoS: noun, verb, pronoun, preposition, adverb, conjunction, participle, article, ...
- Tagset: a tag represents PoS with or without morphological information
  - 87 tags in Brown corpus (Francis, 1979)
  - 45 tags in Penn Treebank (Marcus et al., 1993)

Part-of-speech tagging

- Example:
- The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
- Input: string of words and a specified tagset
- Output: single best tag for each word

Tagging in NLP

- tagging is a standard problem
- taggers exist for many languages
- same principles for other applications, e.g.
  - chunking
  - partial parsing (“shallow parsing”)
  - named entity recognition

Part-of-speech tagging, cont.

- Trivial
  - non-ambiguous words
- Non-trivial:
  - resolve ambiguous words (more than one possible PoS)
    - Book/VB that/DT flight/NN ./.
    - book NN VB
    - that DT CS
  - unknown words not present in the training data

Types of tagger

- Rule-based
  - Earliest taggers (Harris, 1962; Klein and Simmons, 1963; Green and Rubin, 1971)
  - Two-stage architecture:
    1. Use a dictionary to assign each word a list of potential PoS
    2. Use large lists of hand-written disambiguation rules to assign a single PoS for each word
  - The dictionaries and the set of rules get larger
  - Ambiguitities often left unsolved in case of uncertainty

Constraint Grammar

- Constraint Grammar approach (Karlsson et al, 1995)
- Example: EngCG tagger (Voutilainen, 1995, 1999)
  - Run each word through (the 2-level) lexicon (transducer)
  - Return the entries for all possible PoS of the word
  - Morphological heuristics for words not in lexicon
  - Apply a set of constraints (3,744 in EngCG-2) to the input sentence to rule out incorrect PoS

Constraint Grammar

- Constraints: example

```
(@w =0
 VFIN (-1 TO))
```

Remove the tag VFIN if the preceding word is ”to”

Constraint Grammar

- EngCG rule development
  - hand-written rules compiled to finite-state automata
  - a linguist changes a set of rules iteratively to minimize tagging errors
  - at each iteration the rules are applied, errors are detected and rules are changed

Example: Output

- I started work
- Annotated text:
- `"<*i>" "i" <*> <NonMod> PRON PERS NOM SG1 SUBJ @SUBJ`
- `"<started>" "start" <SV> <SVO> <P/on> V PAST VFIN @+FMAINV`
- `"<work>" "work" N NOM SG @OBJ`

Constraint Grammar

- EngCG grammar for morphological disambiguation:
  - 1100 grammar-based constraints for disambiguation of multiple PoS and other inflectional tags
  - accuracy: 99.7-100 %
  - leaves 3-6 % morphological ambiguity
  - 200 heuristic constraints to resolve 50 % of remaining ambiguities

Constraint Grammar

- EngCG syntax:
  - for syntactic functions and disambiguation
  - 300 mapping rules: attach all possible syntactic alternatives to the morphologically disambiguated output
  - 250 syntactic constraints for syntactic ambiguity resolution
  - 75-85% of all words become syntactically unambiguous and
  - 95.5-98% of all words retain the appropriate syntactic-function tag

Constraint Grammar

- Some other grammars
  - PALAVRAS parser for Portuguese (Bick 2000) with generalized dependency markers and semantic prototype tags
  - DanGram
  - The Oslo-Bergen Tagger (Bokmål and nynorsk)
  - And grammars for Sami, Swedish (SWECG), French, German, Catalan, Estonian, Spanish, Esperanto etc.
  - Used for corpus annotation, grammar checking (e.g. Norwegian) and machine translation systems (e.g. Danish-English)

Constraint Grammar

- New CG development:
  - CG2 (Tapanainen 1996) and VISL CG2
  - VISL CG3 with new possibilities such as dependency grammar
  - An overwiev: http://visl.sdu.dk/constraint_grammar.html

Data-driven tagging

- Goal: each word recieves a unique PoS (no ambiguities left)
- Usual steps in tagging:
  - Input: text/transcribed speech
  - Lexikon lookup: tagging with "default" tags
  - Disambiguation of ambiguous words
  - Output: Each word is annotated with one PoS tag

Data-driven taggers

- requires data set (supervised training)
- learning: algorithm to find the best explanation for the observation in a corpus
- classification problem (discret classes)

To decide

- algorithm/learning method to use
- represent the class (tagset)
- attributes to use (linguistic analysis)
- data size
  - training set
  - validation set
  - test set
- evaluation method

Transformation-based learning (TBL)

- Eric Brill 1992, 1995
- also called Brill tagger
- one of the first popular data-driven taggers
- based on rules (or transformations) which determine when ambiguous words should have a given tag
- ML component: grammar rules are automatically induced from a tagged training corpus
- system learns by detecting errors

Transformation-based tagging

- Principle:

  lexicon lookup: choose the most frequent tag for each word according to the lexicon, otherwise use heuristics

  disambiguation: change the initial tagging by looking at the context (tags and words)

  trigger: lexical and contextual features

  transformations: rewrite rules that change a tag given a certain context

Transformation-based tagging (forts)

- 2 types of rules:
  - Lexical: to annotate unknown words
  - Contextual: to improve the tagging of the lexical module
- Rule form:
  - Lexical: if condition, tag the word with tag T
    - Condition: word contains character X, has prefix/suffix of max. 4 characters, if prefix/suffix is removed/added we get a known word, bigrams

Transformation-based tagging, rules

- Contextual: if conditon, change tag T1 to T2
  - Condition: the word, tags or words in the context

scheme	t_{i-3}	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}	t_{i+3}
1			■	*			
2				*	■		
3		■	■	*			
4				*	■	■	
5	■	■	■	*			
6				*	■	■	■
7			■	*	■		
8			■	*		■	
9		■		*	■		

Table 10.7, M&S, s. 363

Transformation-based tagging (cont)

- transformations:

original tag	resulting tag	trigger
NN	VB	preceeding tag is T0 go to school
VBP	VB	one of the preceeding 3 tags is MD cut
JJR	RBR	next tag is JJ more valuable player

Table 10.8, M&S, s. 363

Transformation-based tagging (cont.)

- How do we get the rules?
  - from an annotated corpus
    → supervised machine learning

1. define triggers
2. train on a training data set

Transformation-based learning

1. initialize the model: each word in a corpus receives the most frequent tag
2. instantiate all possible transformations and choose the one that reduces the error rate the most
3. use the choosen transformation and apply it to the corpus, and continue with 2 as far as you get approval
4. stop learning and save the rules in the same order as they have been learned

Transformation-based learning

- learning results: transformations instead of probabilities (categorial/symbolic method)
- rules are ordered
- rules can be read and modified
- learning is slow

Transformation-based learning

- Advantages
  - rich pattern system (lexical and contextual triggers)
  - new patterns can be added
  - comprehensible rules
  - rules can be changed
- Disadvantages
  - slow
  - ordered set of rules

Transformation-based learning

- Different implementations:
  - fnTBL (Grace Ngai and Florian Radu, 2000)
    - fast version, used for chunking, word-sense disambiguation, etc.
  - μTBL (Lager, 2000)
    - implementation in prolog for PoS tagging, chunking, dialog act tagging, word sense disambiguation

Stochastic taggers

- Resolve ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context
- Hidden Markov Model or HMM tagger
  - HMM tagging is a task of choosing a tag-sequence with the maximum probability
  - Tagging is treated as a sequence classification task:
    - What is the best sequence of tags which corresponds to a particular sequence of words?

How an HMM tagger works?

- We consider all possible sequences of tags and choose the tag sequence which is most probable given the observation sequence of *n* words
- HMM tagging algorithm chooses as the most likely sequence of tags the one that maximizes the product of two terms:
  - the probability of each tag generating a word
  - the probability of the sequence of tags

$$\underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

How an HMM tagger works?

- Compute tag frequencies for each tag
- Calculate the word likelihood probabilities, $\mathrm{P}(word_i|tag_i)$, represent the probability, given that we see a given tag associated with a given word, i.e., we compute lexical frequencies by PoS-category for each word
- Calculate the tag sequence probabilities $P(t_i|t_{i-1})$ (bigram frequencies)
- Calculate products of lexical likelihood and tag sequence probabilities and decide the PoS tag.

Computing the most probable tag sequence

- Secretariat/NNS is/VBZ expected/VBN to/TO race/VB tomorrrow/NR
- Example: race / VB or NN?
- NNS VBZ VBN TO VB NR
- NNS VBZ VBN TO NN NR
- Ambiguity resolves globally (not locally) picking the best tag sequence for the whole sentence

Computing the most probable tag sequence

- How likely are we to expect a verb/noun given the previous tag?

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

- We can derive the maximum likelihood estimate of a tag transition probability from corpus counts:
- $P(NN|TO) = \frac{C(TO,NN)}{C(TO)} = .00047$
- $P(VB|TO) = \frac{C(TO,VB)}{C(TO)} = .83$

Computing the most probable tag sequence

- What is the likelyhood that the word race has VB and NN tag?

$$P(w_i|t_i)$$

- We can derive the probabilites (lexical likelihoods) from corpus counts.
- $P(race|NN)$ = .00057 (How likely that the noun is *race*?)
- $P(race|VB)$ = .00012 (How likely that the verb is *race*?)

Computing the most probable tag sequence

- What is the tag sequence probability for the following tag (tomorrow/NR)?

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

- We can derive the probabilites from corpus counts.
- $P(NR|VB) = \frac{C(VB,NR)}{C(VB)} = .0027$
- $P(NR|NN) = \frac{C(NN,NR)}{C(NN)} = .0012$

Computing the most probable tag sequence

- Putting together the results:

$$\underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

- $P(VB|TO)P(NR|VB)P(race|VB)$ = .83*.00012*.0027 = .00000027
- $P(NN|TO)P(NR|NN)P(race|NN)$ =.00047*.0012*.00057 = .00000000032
- The prob of the sequence with the VB tag is higher and *race* is tagged as VB although it is less likely for *race*.

How an HMM tagger works?

- HMM is a weighted finite-state automaton in which state transitions (arcs) have probabilities indicating how likely that path is, and whose output is also probabilistic.
- one state of each PoS, output is the words of the sentence
- HMM has 2 types of probs: the observation *likelihoods* (emission probs) of the word string and *prior* transition probalities of the tag sequence

$$\underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}\overbrace{P(t_i|t_{i-1})}$$

How an HMM tagger works?

- Viterbi algorithm: takes as input an HMM and a set of observed words and returns the most probable tag sequence.
- Probability matrix with one column for each observation t and one raw for each state graphs.

How a trigram HMM tagger works?

- Most modern HMM taggers, like Trigrams'n Tags (Brants, 2000) use more context, i.e., letting the probability of the tag depend on the two previous tags:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1}, t_{i-2})$$

- Sententence boundaries are marked so the tagger know the location of the end of the sentence by a special sentence boundary tag added to the tagset. Therefore, sentence boundaries must be marked in your data by an empty line!

How a trigram HMM tagger works?

- Problem: Data sparsity
- A particular sequence of tags t_{i-2}, t_{i-1}, t_i in the test set may not exist in the training set.
- We cannot compute the following:

$$P(t_i|t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}$$

- but we can estimate $P(t_i|t_{i-1}, t_{i-2})$!

How a trigram HMM tagger works?

- Estimate the prob by combining weaker estimators.
- The maximum likelihood estimation of each prob can be computed from corpus counts:

$$Trigrams\ \hat{P}(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})}$$

$$Bigrams\ \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$$

$$Unigrams\ \hat{P}(t_i) = \frac{C(t_i)}{N}$$

How a trigram HMM tagger works?

- The estimators for trigrams, bigrams, and unigrams are combined.
- The maximum likelihood estimation of each prob can be computed from corpus counts:

  $$P(t_i|t_{i-1},t_{i-2}) = \lambda_3\hat{P}(t_i|t_{i-1},t_{i-2}) + \lambda_2\hat{P}(t_i|t_{i-1}) + \lambda_1\hat{P}(t_i)$$

  where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, i.e., P is a probability distribution.
- λ is set by using deleted interpolation (Jelinek and Mercer, 1980).

How a trigram HMM tagger works?

- Deleted interpolation:
  - We successively delete each trigram from the training corpus and choose λs so to maximize the likelihood of the rest of the corpus, i.e., to generalize to unseen data and not overfit the training corpus.
- TnT accuracy: 96.7% on Penn Treebank with a trigram tagger.
- Open source reimplementation of TnT is HunPoS (Halacsy et al, 2006)

Class representation

- tagset size: depending of the corpus and language type
- tagset size for English: 50-100 tags
- for agglutinative and highly inflectional languages, the tagset size is much larger as they are sequences of morphological tags rather than a single tag
- comparisons in the morphologically tagged MULTEXT-East corpora

Language	Tagset size
English	139
Czech	970
Estonian	476
Hungarian	401
Romanian	486
Slovene	1033

(Hajic, 2000)

Attributes

Info	MB	ME	TBL	TnT
word	+	+	+	+
suffix	3	4	4	10
prefix	-	4	4	-
versal	+	+	+	+
number	+	+	-	-
word before	1	2	3	-
word after	1	2	3	-
tag before	2	2	3	2
tag after	1	-	3	-

PoS Tagging· June 2, 2009

Evaluation

Beáta B. Megyesi
beata.megyesi@lingfil.uu.se

Evaluating Taggers

- Evaluation proceeds by comparing tagger output against gold-standard answers
- Measures: Accuracy, Precision, Recall and F-measure (from IR)
- Accuracy: the percentage of all tags in the test set where the tagger and the gold standard agree

Evaluating Taggers

- Precision: the percentage of tags/chunks that are provided by the system that are correct

$$Precision = \frac{\#\,of\ correctly\ tagged\ tokens\ with\ PoS\ tag\ X}{Total\ \#\,of\ tagged\ tokens\ with\ PoS\ tag\ X} \quad (1)$$

- Recall: the percentage of tags are actually present in the input that were correctly identified by the system

$$Recall = \frac{\#\,of\ correctly\ tagged\ tokens\ with\ PoS\ tag\ X}{Total\ \#\,of\ tokens\ with\ PoS\ tag\ X\ in\ reference} \quad (2)$$

- F-measure: harmonic mean, a way of combining P and R

$$F = \frac{(\beta^2 + 1) * Precision * Recall}{\beta^2 * Precision + Recall} \quad (3)$$

Evaluation: example

Our: Nora/N saw/N a/D good/Adv movie/N on/P TV/N./F

Gold: Nora/N saw/V a/D good/A movie/N on/P TV/N ./F

Accuracy = 6/8 = 0.75

N: Precision = 3/4 = 0.75, Recall = 3/3 = 1.0

D: Precision = 1/1 = 1.0, Recall = 1/1 = 1.0

Adv: Precision = 0/1 = 0, Recall = 0/0 = –

P: Precision = 1/1 = 1.0, Recall = 1/1 = 1.0

F: Precision = 1/1 = 1.0, Recall = 1/1 = 1.0

A: Precision = 0/0 = –, Recall = 0/1 = 0

Evaluation: Method

- Decide the baseline (at least most frequent class baseline), that a system should have as a bottom line
- Always separate training, development, and test set!
- Use development set while you are improving the system, and test it on the test set in the end!
- Use n-fold cross validation where appropriate!
- Use statistical tests to determine if the difference between two models is significant! Paired test: paired t-test, McNemar test (see Cohen, 1995; Dietterich, 1998)

Important

- size of data (the more the better)
- tagset size
- the type of training and test set
- use n-fold cross validation in case of small data size

Results for various taggers for Swedish

Table 1: The tagging accuracy for all the words, and the accuracy of known and unknown words for each PoS tagger. Training and test set are disjoint, consisting of 100k tokens, respectively. Tagset includes 139 tags.

ACCURACY	MB	ME	TBL	TNT
Total (%)	89.28	91.20	89.06	**93.55**
Known (%)	92.85	93.34	94.35	**95.50**
Unknown (%)	68.65	78.85	58.52	**82.29**

The most common errors

Correct	Wrong tag
adjective (AQPNSNIS)	adverb (RGPS)
particle (QS)	preposition (SPS)
noun plural (NCNPNIS)	noun singular (NCNSNIS)
adjective singular (A...S...)	adjective plural (A...P...)
adverb (RG0S)	particle (QS)

Corpus-based NLP

- ”Every time I fire a linguist the performance of the recognizer goes up” (F. Jelinek, IBM Research Group, 80-tal)
- data-driven methods preferred
- problem with rule-based approaches
  - language constructions are accepted or not
  - no preferences among ambiguous analysis

Drawbacks with data-driven methods

- need a large corpus (collected and analyzed)
- disambiguated material cost to produce (partly manual work)
- corpus representativity is not always prior
- language models are hard to understand in linguistic research
- models are hard to modify after learning
- require knowledge in mathematics and computer science

Advantages with data-driven methods

- automatic learning
- algorithms are available and implemented
- more and more data available
- bootstrapping: technique that iteratively trains and evaluates a classifier in order to improve its performance
- computers handle large amounts of data
- statistical models are robust

Assignment

- Train several models using TnT (Brants, 2000) and evaluate the result.
  - experiment with various parameters (ignoring case, using bigrams and unigrams, suffix analysis)
  - improving models by adding more training data and using bootstrapping
  - construct a model using a large training corpus