

Apertium part-of-speech tagger internals

Frankie Robertson
`frrobert@student.jyu.fi`

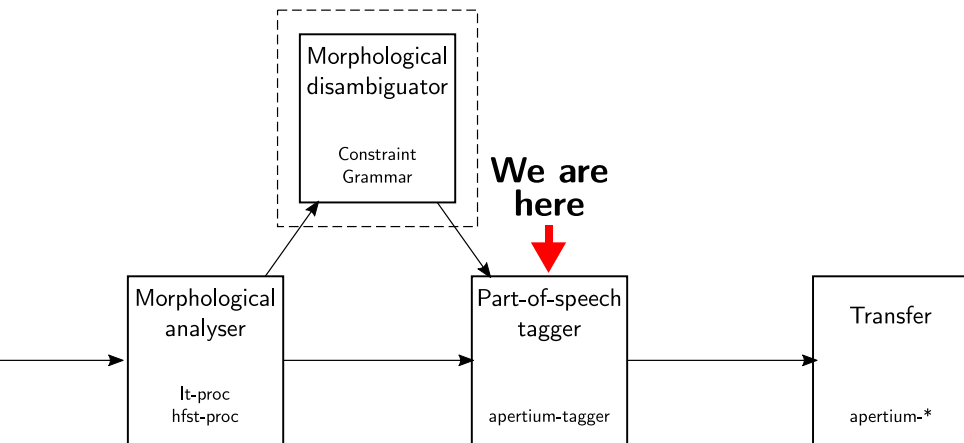
University of Jyväskylä

<https://www.github.com/frankier/tagger-internals-slides/>

18th of July, 2016



Orientation



An idealised part-of-speech tagger



- ▶ Hard-working linguist picks the correct morphological analysis of every sentence that ever has been or will be.
- ▶ Tagger looks up all the analyses of our input sentence. Outputs the most frequent analysis of each word.
- ▶ $\arg \max_{t_i \in T} P(t_i | w_0, w_1, \dots, w_n)$
- ▶ “Pick the most likely tag for the word in the context of the sentence”

- ▶ This is impossible. Instead our linguist works for a while and comes back with a tagged corpus of 10 000 sentences.
- ▶ If the sentence to tag isn't in our corpus, the previous algorithm can't do anything. Our tagger must deal with data sparsity.
- ▶ If the sentence is in our corpus once, it could be tagged with a rare interpretation. Our tagger shouldn't overfit.
- ▶ How? Reduce the number of parameters and smooth over incomplete data.

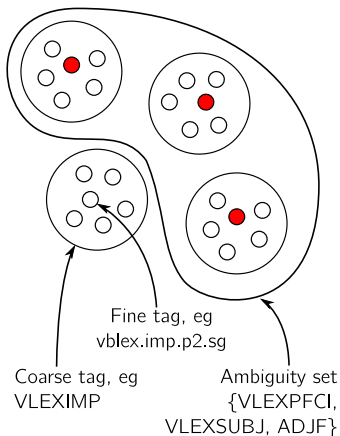
A zero-parameter, zero-gram model

- ▶ Assume that all tags are uniformly distributed independently from context.
- ▶ Since they are evenly distributed, we can just pick the first one.
- ▶ Usually implemented with Apertium as `cg-proc -1`.
- ▶ This is our baseline.

Unigram tagger

- ▶ In general, discard all context apart from the word itself.
- ▶ $\arg \max_{t_i \in T} P(t_i | w_i)$
- ▶ In Apertium, result of a Google Code-In project by m5w based on <http://coltekin.net/cagri/papers/trmorph-tools.pdf> — Supervised (needs a tagged corpus).
- ▶ Model one: Just pick the most frequent whole analysis. So if our corpus is just *work* $\langle n \rangle \langle sg \rangle$ once and *work* $\langle vblex \rangle \langle imp \rangle$ zero times our tagger always picks *work* $\langle n \rangle \langle sg \rangle$. If a word isn't in our corpus at all this acts like the zero-gram case due to smoothing (“cross fading” between different models).
- ▶ Model two: decompose using Bayes' theorem:
 $P(t) = P(\text{root}, \text{analysis}) = P(\text{root} | \text{analysis}) P(\text{analysis})$. So count each analysis string and count for each analysis how many of each roots it has to estimate. From previous example, our model can now tag an unseen word as $\langle n \rangle \langle sg \rangle$.

From analyses to coarse tags and ambiguity classes



- ▶ Cluster morphological analyses into coarse tags — reduces number of parameters.
- ▶ *coarsen*: *Analysis* \rightarrow *Coarse Tag*;
 $\text{coarsen}(\text{di}\langle\text{vblex}\rangle\langle\text{imp}\rangle\langle\text{p2}\rangle\langle\text{sg}\rangle)$
 $= \text{VLEXIMP}$
- ▶ *coarsen-many*: *Analyses* \rightarrow *AmbiguitySet*;
 $\text{coarsen-many}(\{\text{work}\langle\text{n}\rangle\langle\text{sg}\rangle, \text{work}\langle\text{vblex}\rangle\langle\text{imp}\rangle\})$
 $= \text{NOUNSG}, \text{VBLEXIMP}$
- ▶ *discriminate*: *Analyses, CoarseTag* \rightarrow *Analysis*;
 $\text{discriminate}(\{\text{work}\langle\text{n}\rangle\langle\text{sg}\rangle, \langle\text{vblex}\rangle\langle\text{imp}\rangle\}, \text{NOUNSG}) = \text{work}\langle\text{n}\rangle\langle\text{sg}\rangle$
- ▶ *coarsen-many* and *discriminate* can be implemented in terms of *coarsen*.

The reduced morphological disambiguation problem

- ▶ First run, *coarsen-many* for each incoming token to get a stream of ambiguity classes.
- ▶ Then, run our coarse tagging tagger on the ambiguity classes to get a coarse tag for each token.
- ▶ Then, run *discriminate* on each tag in the resulting coarse tag stream to get stream of disambiguated morphological analyses.
- ▶ Coarse tags are defined by globbing (pattern matching) fine tags and sometimes additionally by listing lemmas.
- ▶ This scheme deals with multiwords (tokens made of a series of (lemma, fine tags) pairs, eg compound words) by assigning a tag for the whole token, usually made by combining other coarse tags eg *maan-kamera* is made up of a NOUNGEN followed by a NOUNSG so we make a new tag, NOUNGEN_NOUNSG.

Picking coarse tags

- ▶ Currently you need to define coarse tags for your language in a `tsx` file.
- ▶ <https://github.com/jimregan/tag-clusterer> is a wrapper around `mkcls`¹ which picks coarse tags for you based on a corpus.
- ▶ `mkcls` maximises $P(x_i | \text{coarsen}(x_i))$: the probability we can guess the correct analysis given a coarse tag. Usually it should be possible to get this value near 1 (\Rightarrow the *discriminate* step is reliable).
- ▶ This program also tries to maximise $P(\text{coarsen}(x_i) | \text{coarsen}(x_{i-1}))$: the probability we can predict a coarse tag from a previous coarse tag. Very common lemmas sometimes end up with their own coarse tag.

¹More info at <http://statmt.blogspot.fi/2014/07/understanding-mkcls.html>

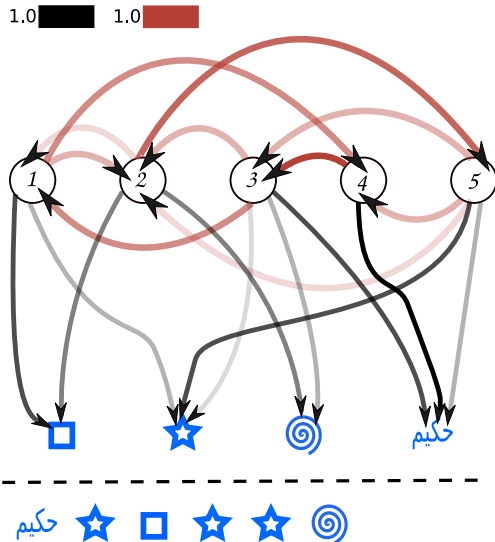
Coarse tags unigram tagger (not in Apertium)

- ▶ Not in Apertium — explained here to help motivate what follows.
- ▶ Estimate $P(\text{tag}|\text{ambiguity-class})$.
- ▶ Model consists of triples: $(\text{ambiguity-class}, \text{tag}, \text{occurrence-count})$.
- ▶ Supervised training: Inspect each tagged token and increment *occurrence-count* in the corresponding triple.
- ▶ Tagging: For each input ambiguity class, pick the *tag* with the highest *occurrence-count*.

Hidden Markov Model bigram tagger

- ▶ Now we use the tag the previous word has received as context. We now try to estimate $P(\text{current-tag}|\text{previous-tag})$ — the transition probabilities.
- ▶ The actual tags are the hidden part of the model.
- ▶ We can observe the ambiguity class — HMM jargon: they are “emitted” by the model.
- ▶ $P(\text{ambiguity-class}|\text{tag})$ (conditional inverted from previous slide) is the emission probabilities.

Diagram + intuition of tagging process



Viterbi tagging algorithm

- ▶ Keep track of the path probabilities through the Markov model.
- ▶ At an unambiguous token, backtrack and output the tags we've remembered.
- ▶ First step, set the probability of a sentinel tag to 1: $V_{1,START} = 1$
- ▶ At step n for each possible tag we calculate $V_{n,tag}$
- ▶ We consider each $prev-tag$ (from all possible previous tags S),

$$V_{n,tag} = \max_{prev-tag \in S} (P(ambg-class|tag) \cdot P(tag|prev-tag) \cdot V_{n-1,prev-tag})$$

- ▶ At each step n for each tag we save the value of the $prev-tag$ for use during backtracking.

- ▶ Supervised training: just estimate the probabilities with their frequency in the corpus.
- ▶ Unsupervised training using Baum-Welch. Idea: iteratively improve our model to maximise the probability of the observations in our untagged corpus. We “fit our model to our data”.

Lightweight Sliding Window Part of Speech Tagger (LSWPoST)

- ▶ Based on work in 2005 by Sanchez-Villamil, M. L. Forcada, and R. C. Carrasco.
- ▶ In Apertium, considers a context of the previous and next tag.
- ▶ It estimates the best tag by summing all possible $P((prev-tag, tag, next-tag)|(prev-ambg-class, tag, next-ambg-class))$ (no search/decoding process)
- ▶ Unsupervised learning of these probabilities by an iterative process.
- ▶ Paper at <http://arxiv.org/pdf/1509.05517v1.pdf>.

Comparison

- For Catalan:

Model	Per token accuracy (%)
0-gram	86.50
Bigram unsupervised	91.51±1.16
LSWPoST	92.99±0.84
Unigram model 1	93.86±1.13
Unigram model 2	93.90±1.09,
Bigram supervised	96.00±0.87.

- Take with a pinch of salt. Per token means includes full stops. If we have sentences 20 tokens long: 86.5% token accuracy means 5.4% sentence rate, 96% accuracy means 44.2% sentence rate.
- A lot more info available at http://wiki.apertium.org/wiki/Comparison_of_part-of-speech_tagging_systems

Comparison cont.

Model	untagged crp	tagged crp	tsx
0-gram	✗	✗	✗
Unigram	✗	✓	✗
Bigram unsup	✓	✗	✓
LSWPoST	✓	✗	✓
Bigram sup	✗	✓	✓

My work on the tagger

- ▶ In process GSOC project trying to improve apertium-tagger in two directions.
- ▶ Experimenting with different ways to integrate It-proc output and CG disambiguated output into the current tagger training and tagging processes.
- ▶ A new tagger based on the generalised perceptron aiming to:
 - ▶ Improve accuracy rates over the bigram tagger for supervised tagging.
 - ▶ Provide a usable tagger for languages where writing a complete tsx ranges from very annoying to infeasible, e.g. Turkic languages
 - ▶ Possibly in the process provide a more usable tagger for other languages too.
 - ▶ Allow user configuration over which features within the sentence are used for disambiguating tokens, and which parts of a candidate token to consider important for disambiguation.

- ▶ See the links throughout the presentation (these slides are available at `https://www.github.com/frankier/tagger-internals-slides/`)
- ▶ At `https://www.github.com/frankier/apertium-hmm2dot` there are some tools I made to explore some of the tagger internals. This includes a tool to visualise bigram HMM tagger models, tools to show what's inside a tagger model file, and tools to show the input streams to the taggers.
- ▶ Read the source code