Fast and Accurate Arc Filtering for Dependency Parsing

Shane Bergsma
University of Alberta

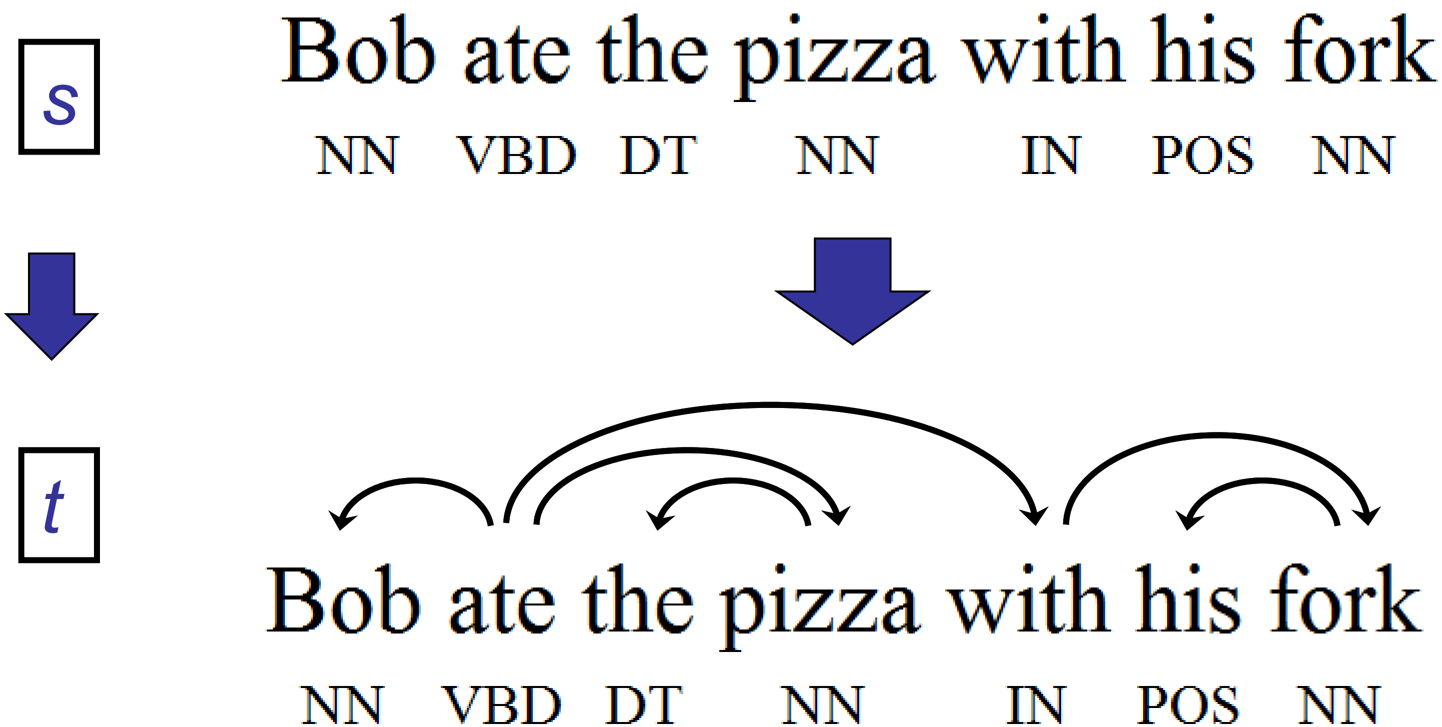
Colin Cherry
National Research Council Canada

COLING 2010

Overview

- Goal:
  - speed-up graph-based dependency parsing by removing implausible head-mod pairs (arcs)
- Method:
  - Classifier says which nodes are leaves/roots/etc.
- Results:
  - remove 78% of arcs, keep 99.5% of true ones
  - speeds up 2 state-of-the-art dependency parsers

Dependency Parsing (DP)



Motivation

- DP could be used to parse the web
  - Requirements: Top Accuracy and Fast
  - This is not a contradiction [Bohnet, 25 minutes ago]
- Graph-based vs. Transition-based
  - Transition-based are *fast*, but make orthogonal errors to graph-based
  - To combine graph-based and transition-based parsers in practice, both must be fast

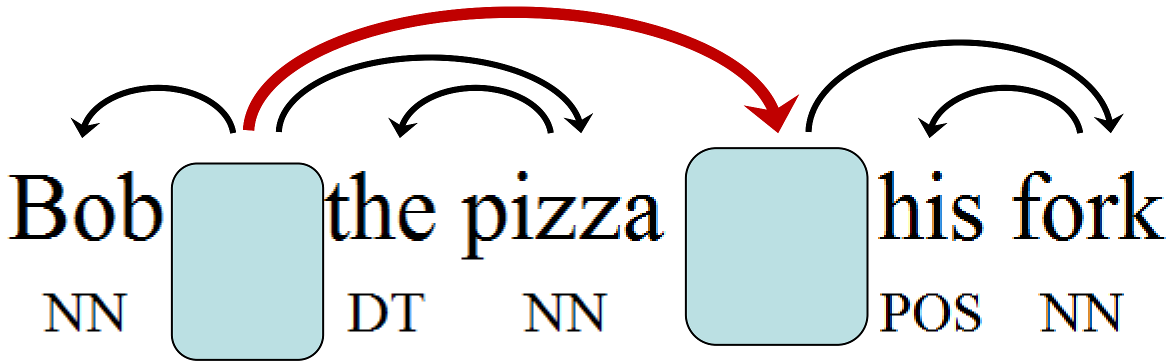
Graph-Based DP

- Tree score decomposes into scores over arcs ([h,m] pairs):

$$\text{parse}(s) = \text{argmax}_{t \in s} \sum_{[h,m] \in t} \bar{w} \cdot \bar{f}(h, m, s)$$

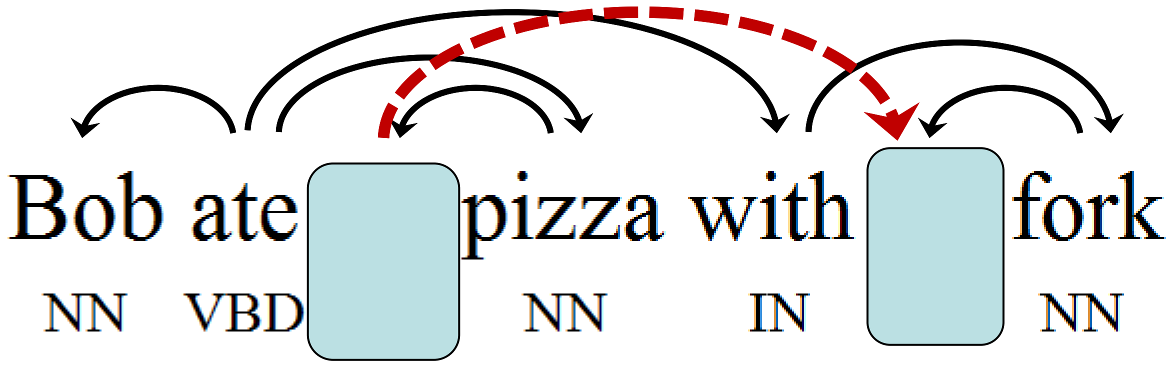
- argmax computed using MST $O(n^3)$ or [Eisner, 1996]'s algorithm $O(n^2)$
- But what if $|\bar{f}| >> n$? ($\bar{f}$ for all $O(n^2)$ arcs)

Features



- $\bar{f}$ = {head=ate, mod=with, ate-with, ate-IN, VBD-with, VBD-IN, ate-the-with, ate-pizza-with, ate-DT-with, ate-NN-with, VBD-the-with, VBD-the-IN, VBD-pizza-with, VBD-pizza-IN, ate-pizza-IN, ate-the-IN, VBD-DT-with, VBD-DT-IN, VBD-NN-with, VBD-NN-IN, ate-DT-IN...}
- Also conjoined with direction, distance!

Features



- Do we really need to generate all the features?
- DT isn’t usually a head
- POS usually has its head on the right

Filter Framework

- 3 stages that prune arcs before parsing
- Each stage is a supervised SVM classifier
  - Extract training decisions from Treebank data
- SVM biased in training to have very high precision (so good arcs don't get filtered)
  - Optimize ***j*** parameter (per-class cost factor) and ***C*** parameter (regularization) in *LIBLINEAR* [Fan et al., 2008]

Rules

not a h	” “ , . ; | CC PRP$ PRP EX -RRB- -LRB-
no $* \leftarrow m$	EX LS POS PRP$
no $m \rightarrow *$	. RP
not a root	, DT
no $h \leftarrow m$	DT←{DT,JJ,NN,NNP,NNS,.} CD←CD NN←{DT,NNP} NNP←{DT,NN,NNS}
no $m \rightarrow h$	{DT,IN,JJ,NN,NNP}→DT NNP→IN IN→JJ

Table 1: Learned rules for filtering dependency arcs using PoS tags. The rules filter 25% of possible arcs while recovering 99.9% of true links.

Linear

- For each word in a sentence [O(n)] decide:
    1. It is *not* a head (i.e., it's a leaf)
    2. Its head is on the left/right
    3. Its head is within 5 words on the left/right
    4. Its head is immediately to the left/right
    5. It is the root
- Filter arcs accordingly
- E.g., if i^{th} word is the root, no crossing arcs

Linear Features

- Have same feature vector for each of 8 linear classifiers (for speed)
  - Build feature vector once, multiply with 8 different weights

$\bar{f}$ = {tag/word$_i$ tag/word$_{i+1}$ tag/word$_{i-1}$ ...
tag/word$_i$^tag$_{i+2}$ tag/word$_i$^tag$_{i+3}$ ...}

Search for Best Linear Filters

- Could optimize each linear classifier separately on dev data
- Better: jointly optimize linear filter *hyperparameters* (C, j for *LibLinear*)
- Ongoing work: jointly optimize linear filter *parameters* (weights) as a single latent SVM optimization

Quadratic

- Decide yes/no to filtering each arc
  - Theoretically, $O(n^2)$ just like the parser's arc-scoring function (so why bother?)
- Can be a *light* preprocessing step, use less features

Quadratic Features

Bob ate the pizza with his fork

NN NN

Features:

NN→ →NN

NN→ →NN

...

- With a feature for every *between-tag*, feature extraction actually $O(n^3)$ [Galley & Manning, 2009]
- Us: only between-tags within ±5 of head or mod
- Plus: a few real-valued tags+distance features

Related Work

- Vine parsing [Eisner & Smith, 2005]
  - Hard cap on arc length
  - Can depend on tags being linked (Tag-Vine)
  - Root + L/R head-marking [Søgaard and Kuhn, 2009]
- CFG cell classification for constituency parsers [Roark & Hollingshead, 2008]
- Coarse-to-fine Parsing
  - Used in DP [Carreras et al., 2008]

Experiments

- WSJ portion of Treebank, standard train/dev/test sets
- Stanford tagger

Test Set Performance

Filter	Coverage	Reduction	Time(s)
Tag-Vine	99.6%	44%	2.9s
Rules	99.9%	26%	1.3s
Linear	99.7%	54%	7.3s
Quadratic	99.5%	78%	16.1s

Analysis

- Which linear filters are most important?
  - See paper for details

Incorporate filters into parsers

- McDonald et al.'s MST parser
  http://sourceforge.net/projects/mstparser/
  - Trained using 5-best MIRA
  - First and second-order (MST-1, MST-2)
- In-house "DepPerceptron"
  - Trained via averaged perceptron [Collins, 2002]
  - Features mix of [McDonald et al. 2005] and [Koo et al. 2008]

Big Parsing Speed-Ups

		DepPercep-1		DepPercep-2		MST-1		MST-2	
Filter	Cost	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time
None	+0	91.8	348	92.5	832	91.2	153	91.9	200
Vine	+3	91.7	192	92.3	407	91.2	99	91.8	139
Rules	+1	91.7	264	92.4	609	91.2	125	91.9	167
Linear	+7	91.7	168	92.4	334	91.2	88	91.8	121
Quad.	+16	91.7	79	92.3	125	91.2	58	91.8	80

- MST-2 goes from 12 to 23 sentences/s

Conclusion

- Linear and quadratic-time filtering leads to speed-ups in even carefully-optimized dependency parsers
- Negligible loss in accuracy
- Download today: http://code.google.com/p/arcfilter/

Thanks

- Ryan McDonald & MST coders, Google
- Terry Koo, MIT (now Google)
- Dekang Lin, Google
- Peter Turney, NRC