Multilingual Dependency Parsing using Bayes Point Machines

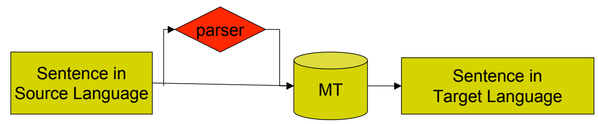
Kevin Duh
University of Washington

Joint work with:
Simon Corston-Oliver, Anthony Aue (Microsoft Research),
Eric Ringger (Brigham Young U.)

Motivation / Project Goal

Project goal:
Build a trainable dependency parser that is easily portable to many languages (given annotated training data)

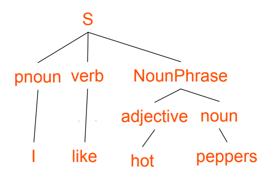
Application: Microsoft Research's Machine Translation System:



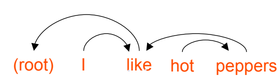
Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as "Structured Classification"
3. Parser architecture
4. Training by Bayes Point Machines
5. Experiments

Constituency vs. Dependency Parsing



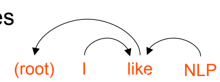
Constituency parse:
- indicates phrase structures
- context free grammar rules



Dependency parse:
- relationships between words
- arrow indicates head-child relations
- e.g. "hot" modifies "peppers"
- e.g. "peppers" is argument of "like"

Dependency Parsing for different languages

- Projective dependency parses



- Non-projective: (crossing arrows)

  - *Free word-order languages (e.g. Czech, Arabic) have more non-projective trees*
    - *Czech treebank: 25% sentences, 2% dependencies, (Nivre, 2005)*

Why Dependency Parsing?

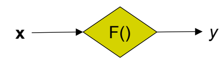
- Some NLP systems need only word-to-word relationship information, e.g.:
  - Machine translation [Quirk et.al., ACL 05]
  - Information extraction [Bunescu&Mooney, HLT05]
  - Question answering [Punyakanok et.al, AIMath04]
- Ease of annotation
  - No grammar building
  - Native speakers can do the job

Outline

1. Intro to Dependency Parsing
2. **Dependency Parsing as "Structured Classification"**
3. Parser architecture
4. Training by Bayes Point Machines
5. Experiments

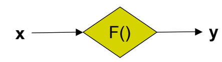
"Structured classification"

- Conventional classification problem:



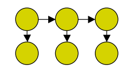
x : vector of input features
y : scalar output

- Structured classification problem:



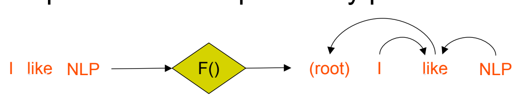
x : vector of input features
y : complex set of outputs (e.g. vector, parse)
values in output may be interdependent

- Popular solutions:
  - Graphical models
  - M^3 Nets (Taskar), Structured SVM (Joachims)

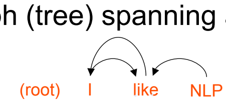


Dependency Parsing as Structured Classification

- Input: features of a sentence
- Output: a whole dependency parse



- Structure constraints: parse is a directed acyclic graph (tree) spanning all words



Malformed parse

A Solution to Structured Classification

$$\hat{y} = \arg\max_{y \in GEN(x)} F(x, y)$$

- x: input sentence
- GEN(x): generates all possible parses of x
- F(x,y): function that scores a parse
- ArgMax: choose output with the best parse

Challenges

$$\hat{y} = \arg\max_{y \in GEN(x)} F(x, y)$$

- How to define and learn F(x,y)?
- How to efficiently compute ArgMax?

Outline

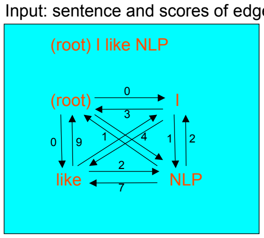
1. Intro to Dependency Parsing
2. Dependency Parsing as "Structured Classification"
3. **Parser architecture**
   1. **Defining F(x,y)**
   2. **ArgMax implementation (Decoder)**
4. Training by Bayes Point Machines
5. Experiments

Defining F(x,y): decomposition

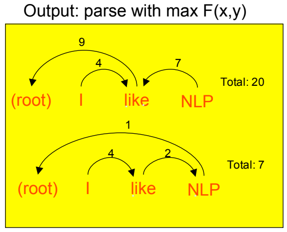
Decompose by edges

$$\arg\max_{y \in GEN(x)} F(x,y) \longrightarrow \arg\max_{y \in GEN(x)} \sum_{(i,j)\in y} \text{score}(i,j)$$

Input: sentence and scores of edges



Output: parse with max F(x,y)



Defining F(x,y): edge scores

$$\arg\max_{y \in GEN(x)} \sum_{(i,j)\in y} \text{score}(i,j) \longrightarrow \arg\max_{y \in GEN(x)} \sum_{(i,j)\in y} w \cdot h(i,j)$$

-**h(i,j):** feature vector of pair word i and word j
-- define based on linguistic knowledge
-- specify different features for different languages

-**w:** weights
-- trained by machine learning methods (discriminatively)

Parser Architecture: 3 components

$$\boxed{\arg\max_{y \in GEN(x)}} \sum_{(i,j)\in y} \boxed{w} \cdot \boxed{h(i,j)}$$

Decoder

Weights
(to be trained)

Features

Decoder/ARGMAX

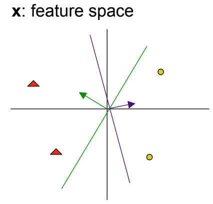
- Requirements:
  - Must search all possible parses for a given sentence
  - Must search fast
    - ArgMax will be invoked multiple times in discriminative training
    - (Preferably) Don't do exhaustive search, don't enumerate malformed parse
- We used:
  - Eisner's decoder for projective trees [Eisner, ACL96]
  - Chu-Liu-Edmonds decoder for non-projective [McDonald, et.al. HLT2005]

Outline

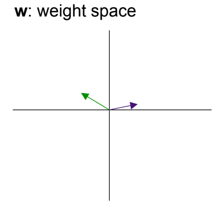
1. Intro to Dependency Parsing
2. Dependency Parsing as "Structured Classification"
3. Parser architecture
4. **Training by Bayes Point Machines**
   1. Version Space
   2. BPM: Bayesian averaging of classifiers
5. Experiments

Weight space/Feature space Duality and Version Space

x: feature space

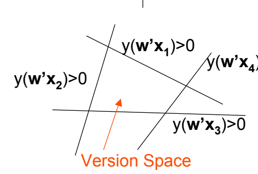


w: weight space

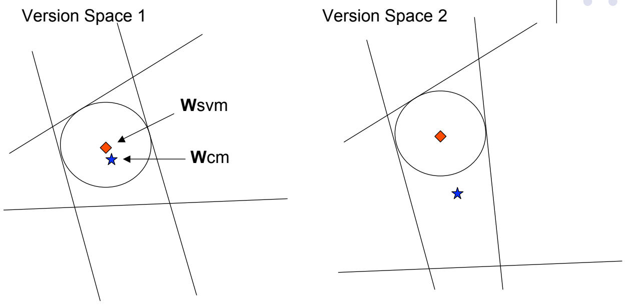


Point in weight space
<=> hyperplane in feature space

Point in feature space
<=> hyperplane in weight space
(defines a halfspace)



SVM & Center-of-Mass solutions in Version Space



The SVM solution is the center of the largest ball enclosed by version space
The Center-of-Mass solution is the “middle” point of the version space
One may argue that **W**cm is better than **W**svm in some situations

Bayes Point Machines (Herbrich, 2001)

- Motivation:
  - Bayesian averaging of classifiers
  - Find the Center-of-Mass solution (**W**cm)
- Main Idea:
  1. Approximate **W**cm by sampling the version space
  2. Sampling is achieved by running perceptron training on randomly shuffled data
  3. Each perceptron gives a **w**, which is then combined to form the BPM solution

BPM Equations

- Ideal Bayesian averaging to achieve **W**cm:

$$w_{BPM} = E_{p(w|D)}[w] = \sum_{k}^{|V(D)|} p(w_k \mid D) w_k$$

- In practice…
  - version space is large => take finite sample of w
  - assume uniform prior p(w)

$$w_{BPM} = E_{p(w|D)}[w] \approx \sum_{k=1}^{K} \frac{1}{K} w_k$$

BPM Pseudo-code

INPUT:
x_i: set of training points, $i = 1,...,N$
$y_i \in \{-1,1\}$: labels of x_i

OUTPUT:
w: discriminatively trained weight vector
Linear model: $\hat{y}_i = sign(w \cdot x_i)$

0. for $k = 1$:K
1. Initialize w_k=0; Randomly shuffle training data
2. for $i = 1 : N$
3. $\hat{y}_i = sign(w_k \cdot x_i)$
4. if $\hat{y}_i \neq y_i$
5. $w_k = w_k + y_i x_i$
6. Repeat until convergence
7. end
8. $w = \frac{1}{K}\sum_{k=1}^{K} w_k$

Bayes Point Machine Pros & Cons

- Pros:
  - Good generalization
  - Online learning
  - Easy to implement
  - Parallel computation
- Cons:
  - Sampling scheme is only approximate
  - Computation grows with number of perceptrons

Outline

1. Intro to Dependency Parsing
2. Dependency Parsing as “Structured Classification”
3. Parser architecture
4. Discriminative Training of Parameters
5. **Experiments**
   1. **Data & Features**
   2. **Evaluation on English, Czech, Arabic, Chinese**

Data

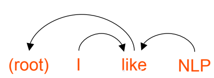
Language	Tokens	Train Sent	Test Sent	Source
Arabic	116k	2100	449	Prague Arabic Dependency Treebank (v1)
Chinese	527k	14k	2080	Chinese Treebank (v5)
Czech	1.6M	73k	7507	Prague Dependency Treebank (v1)
English	1M	40k	2416	Penn Treebank

Features

- Extract for every given pair of dependencies in Training Set:
  - ParentToken
  - ChildToken
  - ParentPOS
  - ChildPOS
  - POS of intervening words
- Backoff features:
  - Czech/English: first five characters “stem”
  - Arabic: stem from a morphological analyzer
  - Chinese: first character “stem”
- Combinations of above to achieve “polynomial kernels”

Evaluation

<u>Evaluation Measures:</u>
- Dependency Accuracy
- Root Accuracy/F1
- Complete Accuracy



Report dependency acc with/without punctuation
What’s best depends on application, e.g.:
- If used for semantic analysis, no need for punctuation
- If used for sentence simplification, need punctuation

Comparison to state-of-the-art

- BPM better than MIRA in Complete Acc, worse in Dependency/Root Acc.

ENGLISH	Dependency Accuracy	Root Accuracy	Complete Accuracy
Perceptron	90.6	94.0	36.5
MIRA (McDonald, 05)	**90.9**	**94.2**	37.5
Bayes Point Machines	90.8	93.7	**37.6**

BPM vs. Perceptrons

Dependency Accuracy

	Arabic	Chinese	Czech	English
Bayes Point Machine	78.4	83.8	-	91.2
Best Perceptron	77.9	83.1	83.8	90.8
Worst Perceptron	77.4	82.6	83.7	90.5

Observation:
BPM result is always better than the best perceptron
=> averaging classifiers works!

Comparing results across languages

With Punctuation

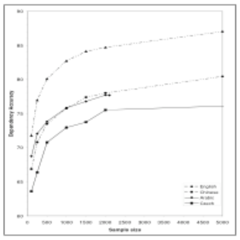
Language	DA	RA	CM
Arabic	79.9	90.0	9.80
Chinese	71.2	66.2	17.5
Czech	83.3	88.3	29.2
English	90.0	93.7	35.1

Without Punctuation

Language	DA	RA	CM
Arabic	79.8	87.8	10.2
Chinese	73.3	66.2	18.2
Czech	83.6	75.5	30.1
English	90.8	93.7	37.6

What makes accuracy vary for different languages?
- language characteristics (e.g. inflectional morphology leading to data sparsity)
- annotation scheme
- training data size

Comparing results across languages: Data reduction exp.



Observations:
- At all sample sizes, English wins
  => reasons: (1) homogeneous data (2) POS tagset encodes morphology
-Czech has worse results, but improves with more data (not shown)
-Why does Chinese and Arabic have similar results?

Summary/Conclusions

- View Dependency Parsing as “Structured Classification”

$$\arg\max_{y \in GEN(x)} F(x,y) \longrightarrow \underset{y \in GEN(x)}{\text{argmax}} \sum_{(i,j) \in y} w \cdot h(i,j)$$

- Bayes Point Machine training
  - Bayesian averaging of classifiers => **W**cm
  - As simple to implement as the perceptron, yet competitive with large margin methods
- Results in four different languages
  - Further work on cross-language comparison needed

Thank you!

- Questions?

Data (more)

English:
- Penn Treebank
- Extract dependencies by Yamada&Matsumoto (IWPT03) heuristics
- POS: use human-annotation for training, Toutanova’s tagger for test

Chinese:
- Chinese treebank (v5)
- Extract dependencies using heuristics
- POS: use human-annotation for training, Toutanova’s tagger for test (tagger has 92.0% token accuracy, 63.8% sentence accuracy on devset)

Czech:
- Prague Dependency Treebank (v1)
- use human-annotated POS & auto-tagged morphological info in train/test

Arabic:
- Prague Arabic Dependency Treebank (v1)
- use human-annotated POS & auto-tagged morphological info in train/test