

Natural Language Processing

Winter 2013

Parts of Speech

Luke Zettlemoyer - University of Washington

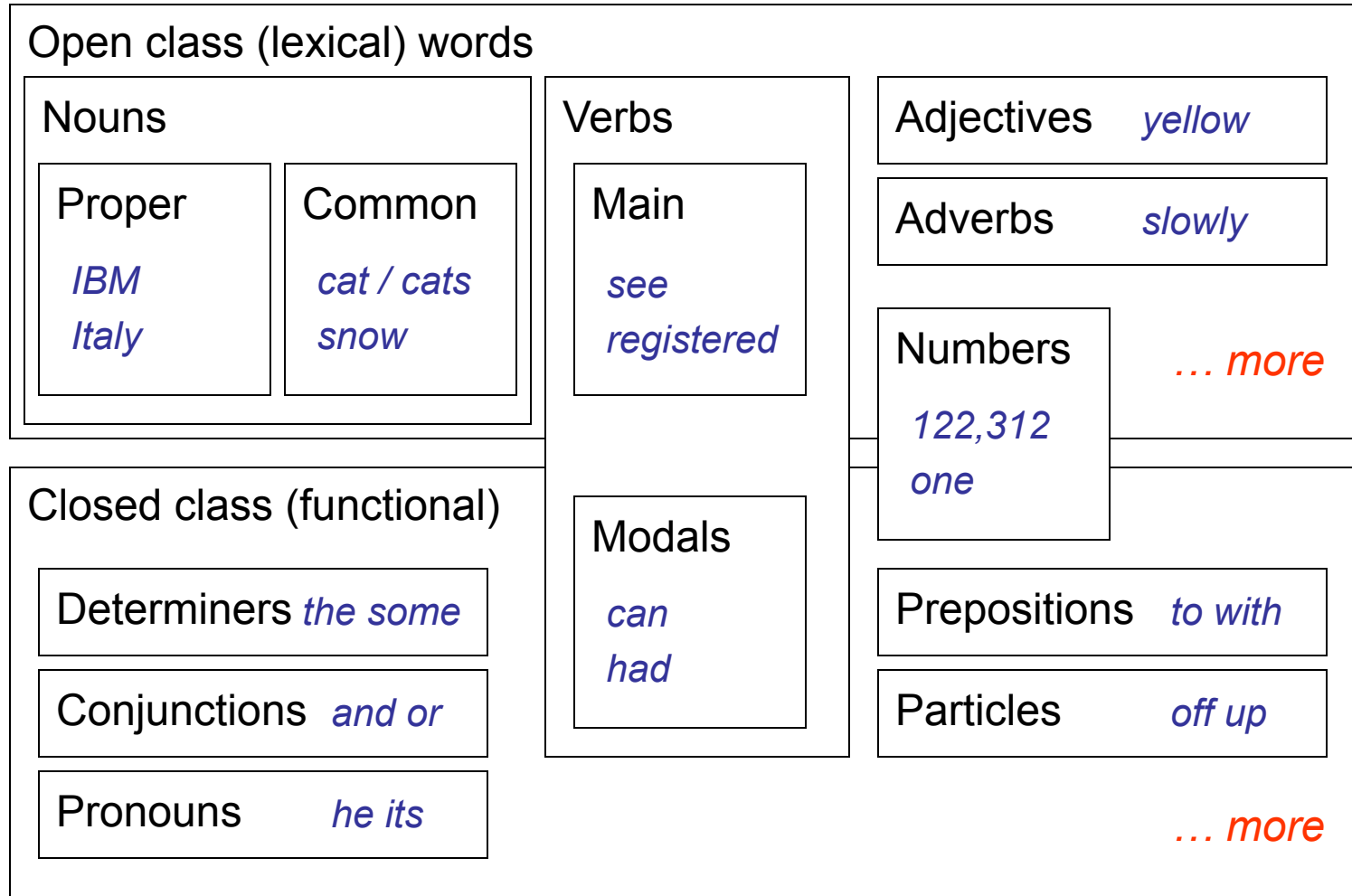
[Many slides from Dan Klein]

Overview

- POS Tagging
- Supervised Techniques
 - MEMMs, Structured Perceptron, CRFs
- Unsupervised
 - EM for HMMs
- Semi-supervised
 - Co-training

Parts-of-Speech (English)

- One basic kind of linguistic structure: syntactic word classes



CC	conjunction, coordinating	and both but either or
CD	numeral, cardinal	mid-1890 nine-thirty 0.5 one
DT	determiner	a all an every no that the
EX	existential there	there
FW	foreign word	gemeinschaft hund ich jeux
IN	preposition or conjunction, subordinating	among whether out on by if
JJ	adjective or numeral, ordinal	third ill-mannered regrettable
JJR	adjective, comparative	braver cheaper taller
JJS	adjective, superlative	bravest cheapest tallest
MD	modal auxiliary	can may might will would
NN	noun, common, singular or mass	cabbage thermostat investment subhumanity
NNP	noun, proper, singular	Motown Cougar Yvette Liverpool
NNPS	noun, proper, plural	Americans Materials States
NNS	noun, common, plural	undergraduates bric-a-brac averages
POS	genitive marker	's
PRP	pronoun, personal	hers himself it we them
PRP\$	pronoun, possessive	her his mine my our ours their thy your
RB	adverb	occasionally maddeningly adventurously
RBR	adverb, comparative	further gloomier heavier less-perfectly
RBS	adverb, superlative	best biggest nearest worst
RP	particle	aboard away back by on open through
TO	"to" as preposition or infinitive marker	to
UH	interjection	huh howdy uh whammo shucks heck
VB	verb, base form	ask bring fire see take
VBD	verb, past tense	pleaded swiped registered saw
VBG	verb, present participle or gerund	stirring focusing approaching erasing
VBN	verb, past participle	dilapidated imitated reunified unsettled
VBP	verb, present tense, not 3rd person singular	twist appear comprise mold postpone
VBZ	verb, present tense, 3rd person singular	bases reconstructs marks uses
WDT	WH-determiner	that what whatever which whichever
WP	WH-pronoun	that what whatever which who whom
WP\$	WH-pronoun, possessive	whose
WRB	Wh-adverb	however whenever where why

Part-of-Speech Ambiguity

- Words can have multiple parts of speech

VBD		VB			
VCN	VBZ	VBP	VBZ		
NNP	NNS	NN	NNS	CD	NN

Fed raises interest rates 0.5 percent

Mrs./NNP Shaefer/NNP never/RB got/VBD **around/RP** to/TO joining/VBG

All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB **around/IN** the/DT corner/NN

Chateau/NNP Petrus/NNP costs/VBZ **around/RB** 250/CD

- Two basic sources of constraint:
 - Grammatical environment
 - Identity of the current word
- Many more possible features:
 - Suffixes, capitalization, name databases (gazetteers), etc...

Why POS Tagging?

- Useful in and of itself (more than you'd think)
 - Text-to-speech: record, lead
 - Lemmatization: saw[v] → see, saw[n] → saw
 - Quick-and-dirty NP-chunk detection: `grep {JJ | NN}* {NN | NNS}`
- Useful as a pre-processing step for parsing
 - Less tag ambiguity means fewer parses
 - However, some tag choices are better decided by parsers

DT NNP NN VBD VBN **IN** RP NN NNS
The Georgia branch had taken **on** loan commitments ...

DT NN IN NN **VDN** VBD NNS VBD
The average of interbank **offered** rates plummeted ...

Baselines and Upper Bounds

- Choose the most common tag
 - 90.3% with a bad unknown word model
 - 93.7% with a good one
- Noise in the data
 - Many errors in the training and test corpora
 - Probably about 2% guaranteed error from noise (on this data)

JJ JJ NN
chief executive officer

NN JJ NN
chief executive officer

JJ NN NN
chief executive officer

NN NN NN
chief executive officer

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:

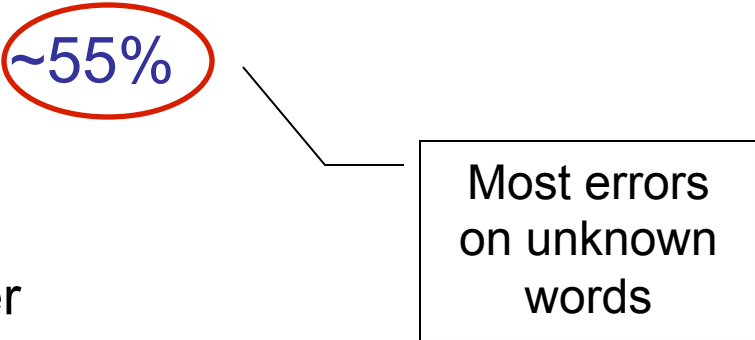
- Most freq tag: ~90% / ~50%

- Trigram HMM: ~95% / ~55%

- TnT (Brants, 2000):

- A carefully smoothed trigram tagger
- Suffix trees for emissions
- 96.7% on WSJ text (SOA is ~97.5%)

- Upper bound: ~98%



Most errors
on unknown
words

Common Errors

- Common errors [from Toutanova & Manning 00]

	JJ	NN	NNP	NNPS	RB	RP	IN	VB	VBD	VCN	VBP	Total
JJ	0	177	56	0	61	2	5	10	15	108	0	488
NN	244	0	103	0	12	1	1	29	5	6	19	525
NNP	107	106	0	132	5	0	7	5	1	2	0	427
NNPS	1	0	110	0	0	0	0	0	0	0	0	142
RB	72	21	7	0	0	16	138	1	0	0	0	295
RP	0	0	0	0	39	0	65	0	0	0	0	104
IN	11	0	1	0	169	103	0	1	0	0	0	323
VB	17	64	9	0	2	0	1	0	4	7	85	189
VBD	10	5	3	0	0	0	0	3	0	143	2	166
VCN	101	3	3	0	0	0	0	3	108	0	1	221
VBP	5	34	3	1	1	0	2	49	6	3	0	104
Total	626	536	348	144	317	122	279	102	140	269	108	3651

NN/JJ NN

official knowledge

VBD RP/IN DT NN

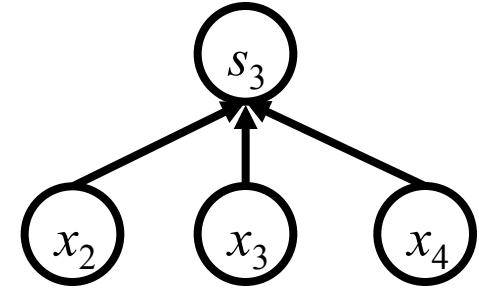
made up the story

RB VBD/VBN NNS

recently sold shares

What about better features?

- Choose the most common tag
 - 90.3% with a bad unknown word model
 - 93.7% with a good one
- What about looking at a word and its environment, but no sequence information?
 - Add in previous / next word
 - Previous / next word shapes
 - Occurrence pattern features
 - Crude entity detection
 - Phrasal verb in sentence?
 - Conjunctions of these things
- Uses lots of features: > 200K



the ___
X ___ X
[X: x X occurs]
___ (Inc.|Co.)
put ___

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
 - Most freq tag: ~90% / ~50%
 - Trigram HMM: ~95% / ~55%
 - TnT (HMM++): 96.2% / 86.0%
 - Maxent $P(s_i|x)$: 96.8% / 86.8%
- Q: What does this say about sequence models?
- Q: How do we add more features to our sequence models?
- Upper bound: ~98%

MEMM Taggers

- **One step up:** also condition on previous tags

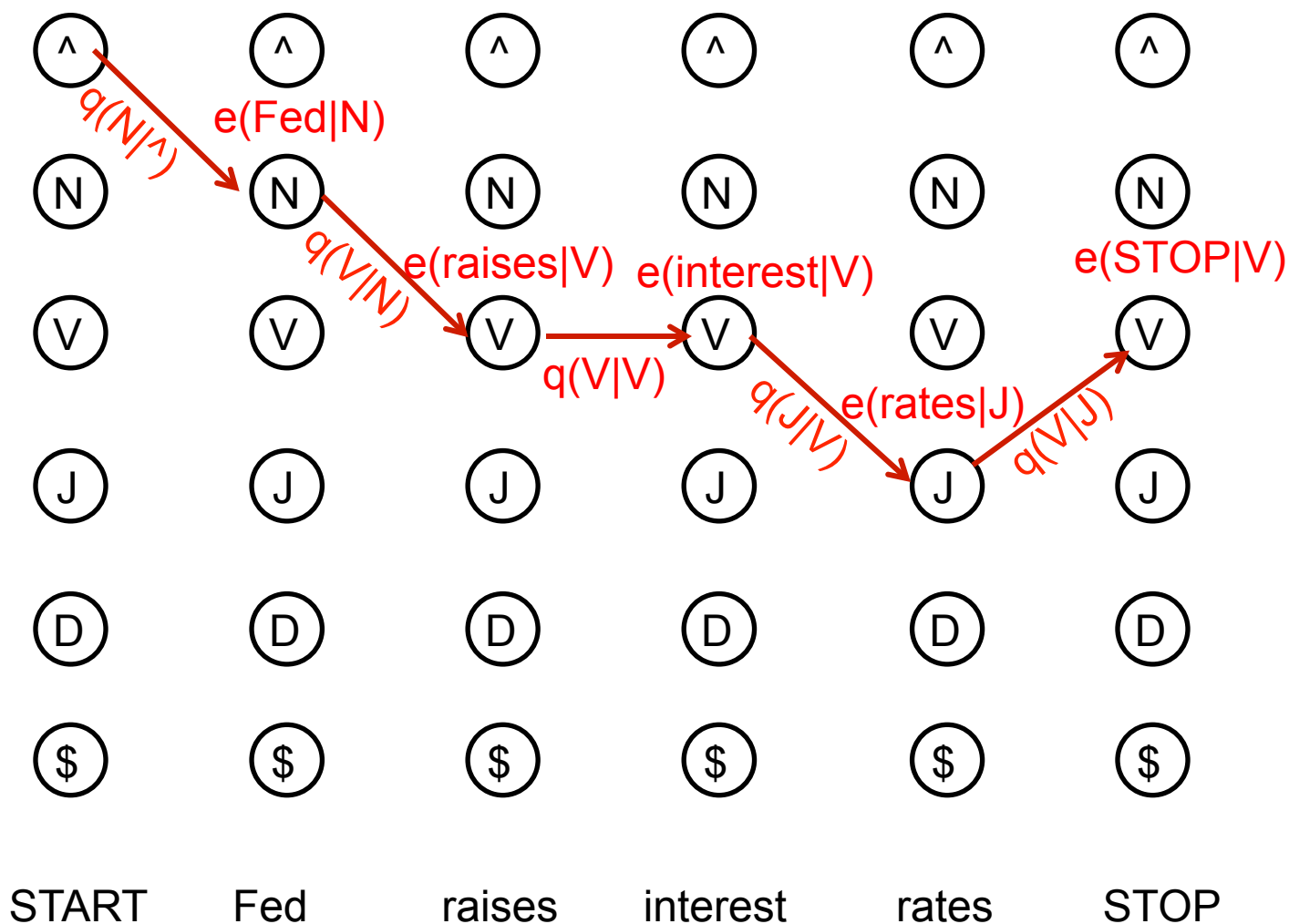
$$\begin{aligned} p(s_1 \dots s_m | x_1 \dots x_m) &= \prod_{i=1}^m p(s_i | s_1 \dots s_{i-1}, x_1 \dots x_m) \\ &= \prod_{i=1}^m p(s_i | s_{i-1}, x_1 \dots x_m) \end{aligned}$$

- Train up $p(s_i | s_{i-1}, x_1 \dots x_m)$ as a discrete log-linear (maxent) model, then use to score sequences

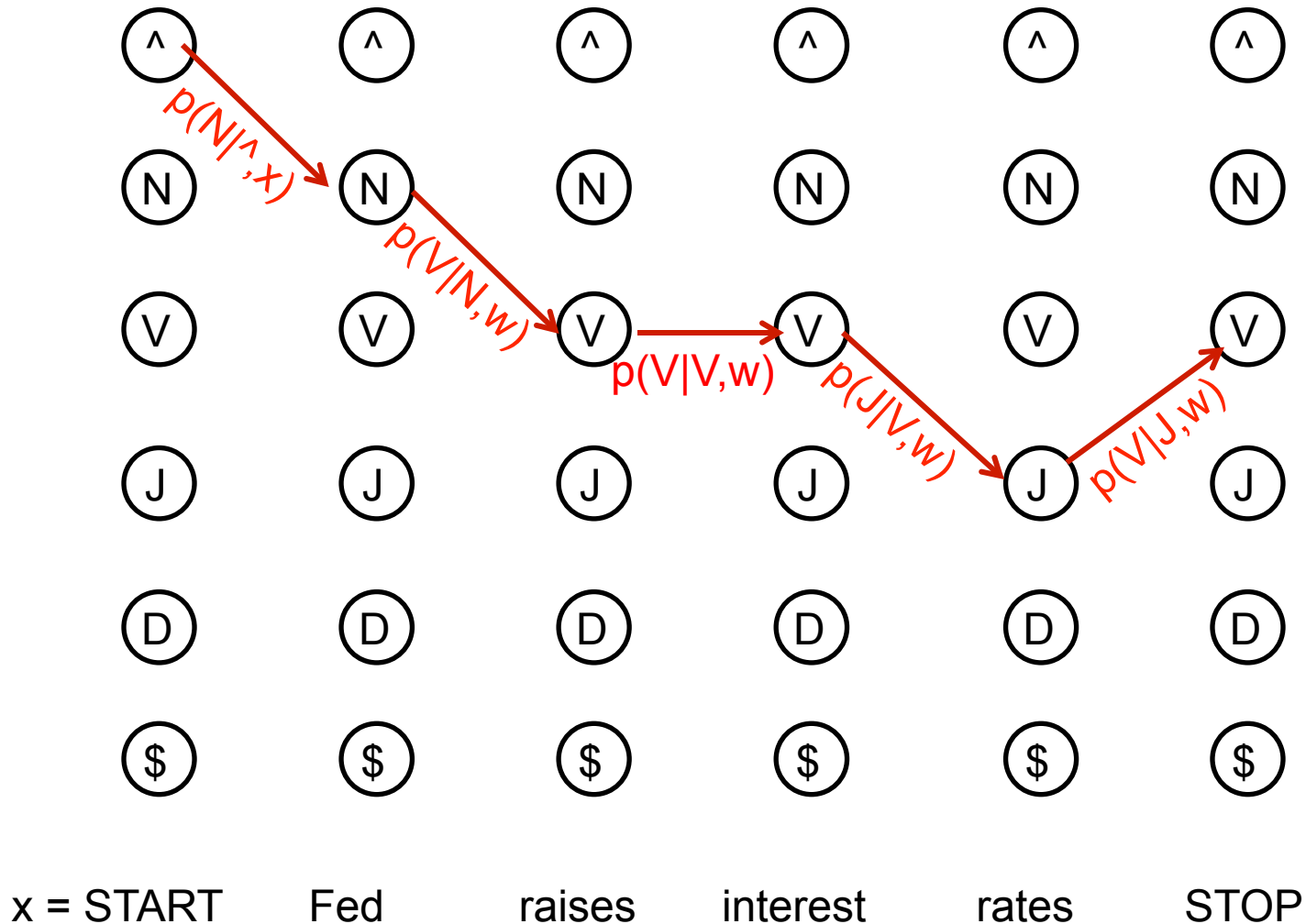
$$p(s_i | s_{i-1}, x_1 \dots x_m) = \frac{\exp(w \cdot \phi(x_1 \dots x_m, i, s_{i-1}, s_i))}{\sum_{s'} \exp(w \cdot \phi(x_1 \dots x_m, i, s_{i-1}, s'))}$$

- This is referred to as an MEMM tagger [Ratnaparkhi 96]
- Beam search effective! (Why?)
- What's the advantage of beam size 1?

The HMM State Lattice / Trellis



The MEMM State Lattice / Trellis



Decoding

- Decoding maxent taggers:

- Just like decoding HMMs
- Viterbi, beam search, posterior decoding

- Viterbi algorithm (HMMs):

- Define $\pi(i, s_i)$ to be the max score of a sequence of length i ending in tag s_i

$$\pi(i, s_i) = \max_{s_{i-1}} e(x_i | s_i) q(s_i | s_{i-1}) \pi(i-1, s_{i-1})$$

- Viterbi algorithm (Maxent):

- Can use same algorithm for MEMMs, just need to redefine $\pi(i, s_i)$!

$$\pi(i, s_i) = \max_{s_{i-1}} p(s_i | s_{i-1}, x_1 \dots x_m) \pi(i-1, s_{i-1})$$

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
 - Most freq tag: ~90% / ~50%
 - Trigram HMM: ~95% / ~55%
 - TnT (HMM++): 96.2% / 86.0%
 - Maxent $P(s_i|x)$: 96.8% / 86.8%
 - MEMM tagger: 96.9% / 86.9%

- Upper bound: ~98%

Global Discriminative Taggers

- **Newer, higher-powered discriminative sequence models**
 - CRFs (also perceptrons, M3Ns)
 - Do not decompose training into independent local regions
 - Can be deathly slow to train – require repeated inference on training set
- **Differences can vary in importance, depending on task**
- **However:** one issue worth knowing about in local models
 - “Label bias” and other explaining away effects
 - MEMM taggers’ local scores can be near one without having both good “transitions” and “emissions”
 - This means that often evidence doesn’t flow properly
 - Why isn’t this a big deal for POS tagging?
 - Also: in decoding, condition on predicted, not gold, histories

Linear Models: Perceptron

- The perceptron algorithm
 - Iteratively processes the training set, reacting to training errors
 - Can be thought of as trying to drive down training error
- The (online) perceptron algorithm:
 - Start with zero weights
 - Visit training instances (x_i, y_i) one by one
 - Make a prediction

$$y^* = \arg \max_y w \cdot \phi(x_i, y)$$

Sentence: $x = x_1 \dots x_m$

Tag Sequence:
 $y = s_1 \dots s_m$

- If correct ($y^* = y_i$): no change, goto next example!
- If wrong: adjust weights

$$w = w + \phi(x_i, y_i) - \phi(x_i, y^*)$$

Challenge: How to compute argmax efficiently?

Decoding

- **Linear Perceptron** $s^* = \arg \max_s w \cdot \Phi(x, s) \cdot \theta$

- Features must be local, for $x=x_1 \dots x_m$, and $s=s_1 \dots s_m$

$$\Phi(x, s) = \sum_{j=1}^m \phi(x, j, s_{j-1}, s_j)$$

- Define $\pi(i, s_i)$ to be the max score of a sequence of length i ending in tag s_i

$$\pi(i, s_i) = \max_{s_{i-1}} w \cdot \phi(x, i, s_{i-1}, s_i) + \pi(i-1, s_{i-1})$$

- **Viterbi algorithm (HMMs):**

$$\pi(i, s_i) = \max_{s_{i-1}} e(x_i | s_i) q(s_i | s_{i-1}) \pi(i-1, s_{i-1})$$

- **Viterbi algorithm (Maxent):**

$$\pi(i, s_i) = \max_{s_{i-1}} p(s_i | s_{i-1}, x_1 \dots x_m) \pi(i-1, s_{i-1})$$

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
 - Most freq tag: ~90% / ~50%
 - Trigram HMM: ~95% / ~55%
 - TnT (HMM++): 96.2% / 86.0%
 - Maxent $P(s_i|x)$: 96.8% / 86.8%
 - MEMM tagger: 96.9% / 86.9%
 - Perceptron 96.7% / ??

- Upper bound: ~98%

Conditional Random Fields (CRFs)

[Lafferty, McCallum, Pereira 01]

- Maximum entropy (logistic regression)

Sentence: $x = x_1 \dots x_m$

$$p(y|x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

Tag Sequence: $y = s_1 \dots s_m$

- Learning:** maximize the (log) conditional likelihood of training data $\{(x_i, y_i)\}_{i=1}^n$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^n \left(\phi_j(x_i, y_i) - \sum_y p(y|x_i; w) \phi_j(x_i, y) \right) - \lambda w_j$$

- Computational Challenges?**

- Most likely tag sequence, normalization constant, gradient

Decoding

$$s^* = \arg \max_s p(s|x; w)$$

■ CRFs

- Features must be local, for $x=x_1 \dots x_m$, and $s=s_1 \dots s_m$

$$p(s|x; w) = \frac{\exp(w \cdot \Phi(x, s))}{\sum_{s'} \exp(w \cdot \Phi(x, s'))} \quad \Phi(x, s) = \sum_{j=1}^m \phi(x, j, s_{j-1}, s_j)$$

$$\arg \max_s \frac{\exp(w \cdot \Phi(x, s))}{\sum_{s'} \exp(w \cdot \Phi(x, s'))} = \arg \max_s \exp(w \cdot \Phi(x, s))$$

$$= \arg \max_s w \cdot \Phi(x, s)$$

■ Same as Linear Perceptron!!!

$$\pi(i, s_i) = \max_{s_{i-1}} \phi(x, i, s_{i-1}, s_i) + \pi(i-1, s_{i-1})$$

CRFs: Computing Normalization

$$p(s|x; w) = \frac{\exp(w \cdot \Phi(x, s))}{\sum_{s'} \exp(w \cdot \Phi(x, s'))} \quad \Phi(x, s) = \sum_{j=1}^m \phi(x, j, s_{j-1}, s_j)$$

$$\begin{aligned} \sum_{s'} \exp(w \cdot \Phi(x, s')) &= \sum_{s'} \exp\left(\sum_j w \cdot \phi(x, j, s_{j-1}, s_j)\right) \\ &= \sum_{s'} \prod_j \exp(w \cdot \phi(x, j, s_{j-1}, s_j)) \end{aligned}$$

Define $norm(i, s_i)$ to sum of scores for sequences ending in position i

$$norm(i, y_i) = \sum_{s_{i-1}} \exp(w \cdot \phi(x, i, s_{i-1}, s_i)) norm(i-1, s_{i-1})$$

- **Forward Algorithm! Remember HMM case:**

$$\alpha(i, y_i) = \sum_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\alpha(i-1, y_{i-1})$$

- Could also use backward?

CRFs: Computing Gradient

$$p(s|x; w) = \frac{\exp(w \cdot \Phi(x, s))}{\sum_{s'} \exp(w \cdot \Phi(x, s'))} \quad \Phi(x, s) = \sum_{j=1}^m \phi(x, j, s_{j-1}, s_j)$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^n \left(\Phi_j(x_i, s_i) - \sum_s p(s|x_i; w) \Phi_j(x_i, s) \right) - \lambda w_j$$

$$\begin{aligned} \sum_s p(s|x_i; w) \Phi_j(x_i, s) &= \sum_s p(s|x_i; w) \sum_{j=1}^m \phi_k(x_i, j, s_{j-1}, s_j) \\ &= \sum_{j=1}^m \sum_{a,b} \sum_{s: s_{j-1}=a, s_b=b} p(s|x_i; w) \phi_k(x_i, j, s_{j-1}, s_j) \end{aligned}$$

- Need forward and backward messages

See notes for full details!

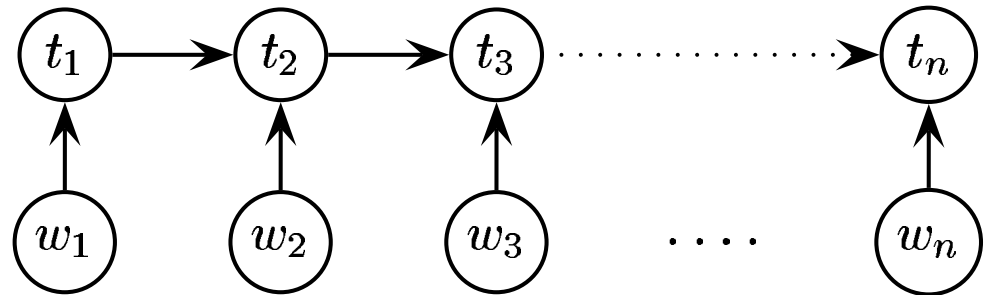
Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
 - Most freq tag: ~90% / ~50%
 - Trigram HMM: ~95% / ~55%
 - TnT (HMM++): 96.2% / 86.0%
 - Maxent $P(s_i|x)$: 96.8% / 86.8%
 - MEMM tagger: 96.9% / 86.9%
 - Perceptron 96.7% / ??
 - CRF (untuned) 95.7% / 76.2%
- Upper bound: ~98%

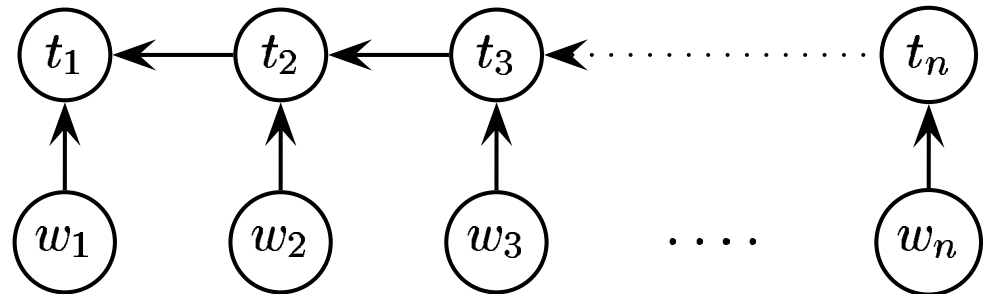
Cyclic Network

[Toutanova et al 03]

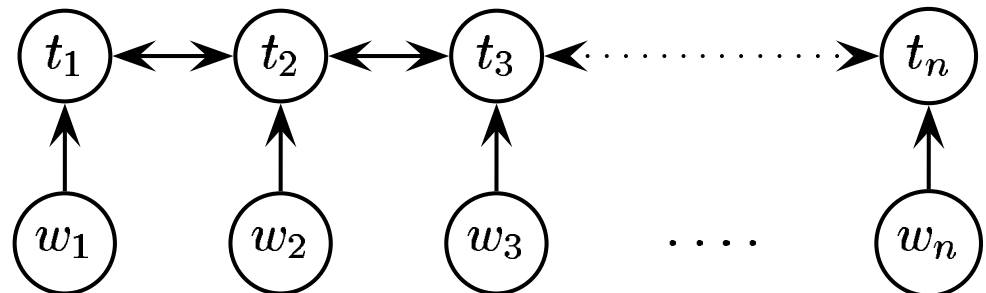
- Train two MEMMs, multiple together to score
- And be very careful
 - Tune regularization
 - Try lots of different features
 - See paper for full details



(a) Left-to-Right CMM



(b) Right-to-Left CMM



(c) Bidirectional Dependency Network

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
 - Most freq tag: ~90% / ~50%
 - Trigram HMM: ~95% / ~55%
 - TnT (HMM++): 96.2% / 86.0%
 - Maxent $P(s_i|x)$: 96.8% / 86.8%
 - MEMM tagger: 96.9% / 86.9%
 - Perceptron 96.7% / ??
 - CRF (untuned) 95.7% / 76.2%
 - Cyclic tagger: 97.2% / 89.0%
 - Upper bound: ~98%

Domain Effects

- Accuracies degrade outside of domain
 - Up to triple error rate
 - Usually make the most errors on the things you care about in the domain (e.g. protein names)
- Open questions
 - How to effectively exploit unlabeled data from a new domain (what could we gain?)
 - How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexicon, ontologies)

Unsupervised Tagging?

- AKA part-of-speech induction
- Task:
 - Raw sentences in
 - Tagged sentences out
- Obvious thing to do:
 - Start with a (mostly) uniform HMM
 - Run EM
 - Inspect results

EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters

$$q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \quad e_{ML}(x|y) = \frac{c(y, x)}{c(y)}$$

- **Crucial step:** we want to tally up (fractional) counts

$$c^*(y) = \sum_{j:y_j=y} p(x_1 \dots x_m, y_j) \quad c^*(y, x) = \sum_{j:x=x_j, y=y_j} p(y_j|x_1 \dots x_m)$$

$$c^*(y, y') = \sum_{j:y'=y_j, y=y_{j-1}} p(y_j, y_{j-1}|x_1 \dots x_m)$$

- We can do this with the forward backward algorithm!!!

Forward, Backward, Again...

$$p(x_1 \dots x_n, y_i) = p(x_1 \dots x_i, y_i) p(x_{i+1} \dots x_n | y_i)$$

- Sum over all paths, on both sides of each y_i

$$\begin{aligned} \alpha(i, y_i) &= p(x_1 \dots x_i, y_i) = \sum_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i) \\ &= \sum_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \alpha(i-1, y_{i-1}) \end{aligned}$$

$$\begin{aligned} \beta(i, y_i) &= p(x_{i+1} \dots x_n | y_i) = \sum_{y_{i+1} \dots y_n} p(x_{i+1} \dots x_n, y_{i+1} \dots y_n) \\ &= \sum_{y_{i+1}} e(x_{i+1} | y_{i+1}) q(y_{i+1} | y_i) \beta(i+1, y_{i+1}) \end{aligned}$$

EM for HMMs: Process

- From these quantities, can compute expected transitions:

$$c^*(y, y') = \frac{\sum_i \alpha(i, y) t(y'|y) e(x_i|y) \beta(i+1, y')}{p(x_1 \dots x_m)}$$

- And emissions:

$$c^*(y, x) = \frac{\sum_{j:x=x_j} \alpha(i, y) \beta(i+1, y)}{p(x_1 \dots x_m)}$$

Unsupervised Learning Results

- EM for HMM
 - POS Accuracy: 74.7%
- Bayesian HMM Learning [Goldwater, Griffiths 07]
 - Significant effort in specifying prior distributions
 - Integrate our parameters $e(x|y)$ and $t(y'|y)$
 - POS Accuracy: 86.8%
- Unsupervised, feature rich models [Smith, Eisner 05]
 - **Challenge:** represent $p(x,y)$ as a log-linear model, which requires normalizing over all possible sentences x
 - Smith presents a very clever approximation, based on local neighborhoods of x
 - POS Accuracy: 90.1%
- Newer, feature rich methods do better, not near SOTA

Semi-supervised Learning

- **AKA:** boot strapping, self training, etc.
- **Task:** learn from two types of data
 - Tagged Sentences
 - Raw / unlabeled sentences
- **Output:** a complete POS tagger
- What should we do?
 - Use labeled data to initialize EM?
 - Sum the counts (real and expected) together?
 - Something fancier?

Merialdo: Setup

- Some initial results [Merialdo 94]
- Setup
 - You know the set of possible tags for each word
 - You have k fully labeled training examples
 - Estimate $e(x|y)$ and $t(y'|y)$ on this data
 - Use the supervised model to initialize the EM algorithms, and run it on all of the data
- Question: Will this work?

Merrialdo: Results

Number of tagged sentences used for the initial model							
	0	100	2000	5000	10000	20000	all
Iter	Correct tags (% words) after ML on 1M words						
0	77.0	90.0	95.4	96.2	96.6	96.9	97.0
1	80.5	92.6	95.8	96.3	96.6	96.7	96.8
2	81.8	93.0	95.7	96.1	96.3	96.4	96.4
3	83.0	93.1	95.4	95.8	96.1	96.2	96.2
4	84.0	93.0	95.2	95.5	95.8	96.0	96.0
5	84.8	92.9	95.1	95.4	95.6	95.8	95.8
6	85.3	92.8	94.9	95.2	95.5	95.6	95.7
7	85.8	92.8	94.7	95.1	95.3	95.5	95.5
8	86.1	92.7	94.6	95.0	95.2	95.4	95.4
9	86.3	92.6	94.5	94.9	95.1	95.3	95.3
10	86.6	92.6	94.4	94.8	95.0	95.2	95.2

Co-Training / Self-Training

- Simple approach, often (but not always) works...
- Repeat
 - Learn N independent classifiers on supervised data
 - Use each classifier to tag new, unlabeled data
 - Select subset of unlabeled data (where models agree and are most confident) and add to labeled data (with automatically label tags)
- $N=1$: Self-training
- $N>1$: Co-Training [Blum and Mitchell, 1998]
 - assumed independent features sets, same learner
 - Proved bounds on when this will work well, see paper!
 - for POS, can do different models with the same features

English POS Self/Co-Training

- Two POS Taggers [Clark, Curran, Osbourne, 2003]

Self Training

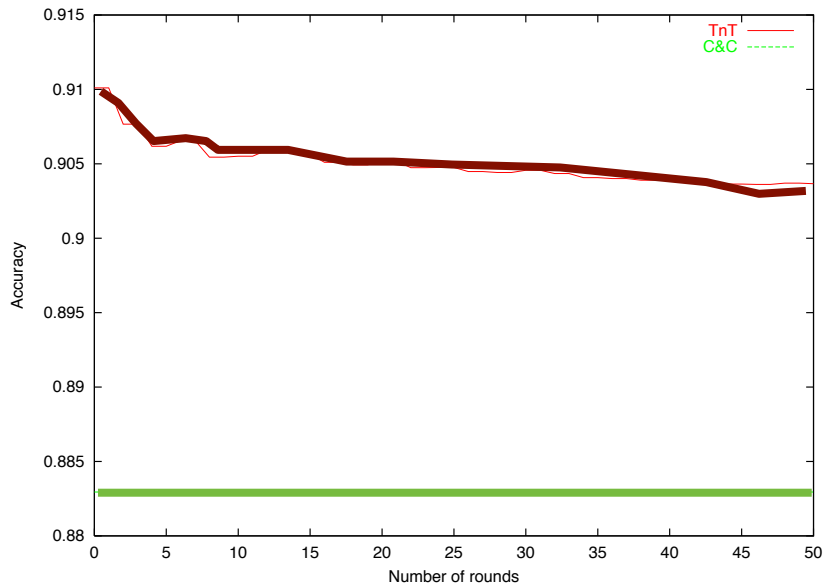


Figure 5: Self-training TNT and C&C (500 seed sentences). The upper curve is for TNT; the lower curve is for C&C.

500 seeds

Co-Training

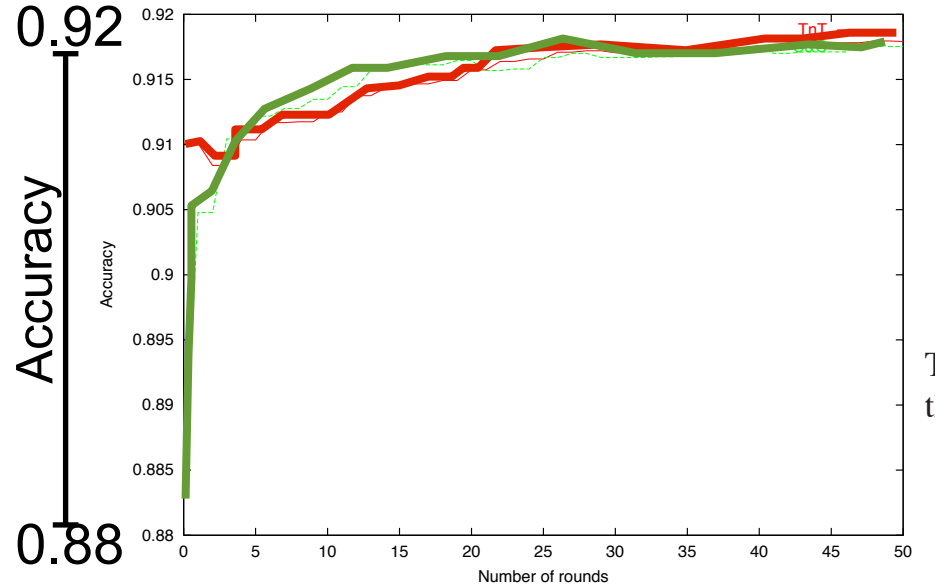


Figure 6: Agreement-based co-training between TNT and C&C (500 seed sentences). The curve that starts at a higher value is for TNT.

500 seeds

Mandarin POS Self/Co-Training

■ Two POS Taggers

[Wang, Huang, Harpaer, 2007]

- HMM
- MEMM

■ CTB: Chinese Penn Tree Bank

Table 3. Comparison of the tagging accuracy (%) of the HMM tagger and ME tagger when trained on the entire CTB corpus and the additional Mandarin BN seed corpus and tested on the Mandarin BN POS-eval test set. Known word, unknown word, and overall accuracies are included.

Tagger		Known	Unknown	Overall
HMM	CTB	80.0	69.2	79.0
	CTB+seed	90.5	75.1	89.6
ME	CTB	79.2	66.8	78.5
	CTB+seed	89.2	74.0	88.1

Table 4. Overall POS tagging accuracy (%) on the Mandarin BN POS-eval test set after applying self-training and co-training.

Training Condition		Tagger	
		HMM	ME
Initial (i.e., CTB+seed)		89.6	88.1
self-training		90.8	90.2
co-training	naive	91.9	91.8
	agreement-based	94.1	94.1
	max-score	93.2	93.1
	max-t-min-s	94.1	93.9