Lecture 13:
Discriminative Sequence Models (MEMM and Struct. Perceptron)

Intro to NLP, CS585, Fall 2014

http://people.cs.umass.edu/~brenocon/inlp2014/

Brendan O'Connor (http://brenocon.com)

Models for sequence tagging

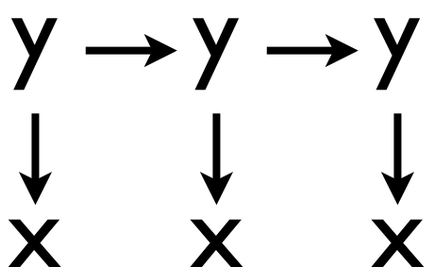
- x: words (inputs)
- y: POS tags (outputs)
- Training: (x,y) pairs
- Testing: given x, predict $y^* = \arg\max_y p(y|x)$

Model diagram (hypothetical!)

Model

Inference

HMM



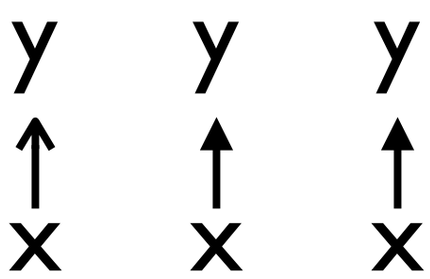
$$p(\vec{y}, \vec{x}) = \prod_t p(y_t|y_{t-1})p(x_t|y_t)$$

Transition multinom.

Emission multinom.

Viterbi

Indep Log Reg, no context



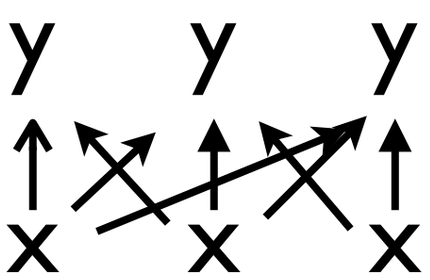
$$p(\vec{y}|\vec{x}) = \prod_t p(y_t|\vec{x}, t)$$

Any features you want... from the input.

$$p(y_t|x_t) \propto \exp(\theta^\mathsf{T} f(x_t, y_t))$$

Indep

ILR with context features



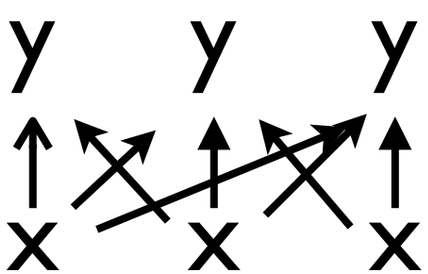
$$p(y_t|\vec{x}, t) \propto \exp(\theta^\mathsf{T} f(\vec{x}, t, y_t))$$

Indep

Features are good

- Typically, feature-based NLP taggers use many many binary features.
  - Is this the first token in the sentence?
  - Second? Third? Last? Next to last?
  - Word to left? Right?
  - Last 3 letters of this word? Last 3 letters of word on left? On right?
  - Is this word capitalized? Does it contain punctuation?
- Indep LogReg can't consider POS context, only word-based context. This seems non-ideal.

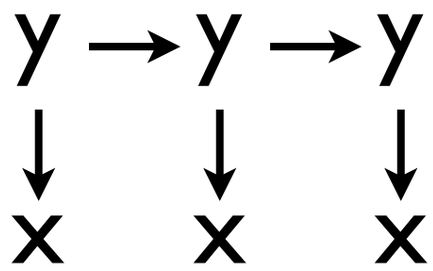
ILR with context features



$$p(y_t|\vec{x},t) \propto \exp(\theta^\mathsf{T} f(\vec{x},t,y_t))$$

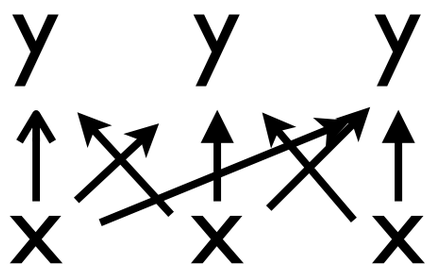
From HMMs to MEMMs

HMM



$$p(\vec{y}, \vec{x}) = \prod_t p(y_t|y_{t-1})p(x_t|y_t)$$

Indep Log Reg

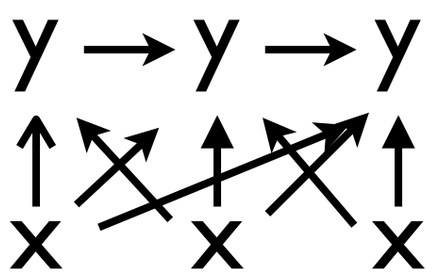


$$p(\vec{y}|\vec{x}) = \prod_t p(y_t|\vec{x}, t)$$

$$p(y_t|\vec{x}, t) \propto \exp(\theta^\mathsf{T} f(\vec{x}, t, y_t))$$

- "Max entropy Markov Model"... is a conditional log-linear Markov Model

MEMM



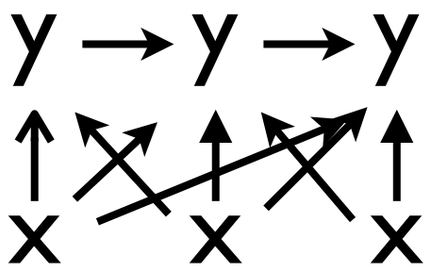
$$p(\vec{y}|\vec{x}) = \prod_t p(y_t|y_{t-1}, \vec{x}, t)$$

$$p(y_t|\vec{x}, t) \propto \exp(\theta^\mathsf{T} f(\vec{x}, t, y_{t-1}, y_t))$$

MEMMs can have features from the input, *and* the previous state!

Using an MEMM

MEMM



$$p(\vec{y}|\vec{x}) = \prod_t p(y_t|y_{t-1}, \vec{x}, t)$$

$$p(y_t|\vec{x}, t) \propto \exp(\theta^\mathsf{T} f(\vec{x}, t, y_{t-1}, y_t))$$

- Training: this is the same training algorithm as logistic regression.
- Testing: like an HMM, e.g. greedy or Viterbi inference.
- Issues
  - Asymmetry in model: underappreciates future tags.
  - (Though it appreciates them a little bit: Viterbi gives different answers than greedy.)
- For POS tagging, MEMM's are state of the art, esp. with tricks (e.g use both left-to-right and right-to-left)

Global structure prediction

- HMMs and MEMMs are ways to score possible output structures, with *local* probabilistic models
- Why not learn parameters for a *global* model of structures? Don't assume any generation process. Instead, use features to score the *entire* structure at once. ("goodness function")

$$\mathrm{G}(y) = \theta^{\mathsf{T}} f(x, y) \qquad \textit{Prediction:}\ \arg\max_y G(y)$$

- Is inference efficient? Depends on the structure of f.
- How to learn? If you know how to evaluate the argmax, you can use the *structured perceptron algorithm*.
- There are several different global structure prediction models out there. Related: *conditional random fields*, where p(y|x) $\propto$ exp(goodness(y))

	finna	get	good
gold y =	V	V	A

Two simple feature templates

f(x,y) is...

For every pair of tag types (*A*,*B*)
"Transition features"

$$\sum_t 1\{y_{t-1} = A, y_t = B\}$$

V,V: 1
V,A: 1

For every tag type *A* and word type *w*
"Observation features"

$$\sum_t 1\{y_t = A, x_t = w\}$$

V,finna: 1
V,get: 1
A,good: 1

	finna	get	good
gold y =	V	V	A

Mathematical convention is numeric indexing, though sometimes convenient to implement as hash table.

Transition features

Observation features

θ

-0.6	-1.0	1.1	0.5	0.0	0.8	0.5	-1.3	-1.6	0.0	0.6	0.0	-0.2	-0.2	0.8	-1.0	0.1	-1.9	1.1	1.2	-0.1	-1.0	-0.1	-0.1

$f(x, y)$

1	0	2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	0	0

$$f_{trans:V,A}(x, y) = \sum_{t=2}^{N} 1\{y_{t-1} = V, y_t = A\}$$

$$f_{obs:V,\text{finna}}(x, y) = \sum_{t=1}^{N} 1\{y_t = V, x_t = \text{finna}\}$$

Goodness(y) = $\theta^\top f(x, y)$

If every element of f(x,y) is based on only a single tag, or neighboring tags, then the Viterbi algorithm can calculate *argmax*$_y$ *G(y)*.

- added after lecture: I did an example on the board to show that Viterbi can do it. those feature counts decompose into local parts of the output structure.

finna get good

y1 *y2* *y3*

$$G(y) = \theta_{trans:y1,y2} + \theta_{trans:y2,y3}$$
$$+ \theta_{obs:y1,finna} + \theta_{obs:y2,get} + \theta_{obs:y3,good}$$

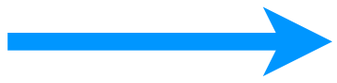
- each term involves two neighboring tags, or just one tag. Therefore to maximize *G* wrt *y*, we can use the NK^2 version of Viterbi we have done before.

	finna	get	good
gold y =	V	V	A
*pred y** =	N	V	A

Learning idea: want gold y to have high scores. Update weights so y would have a higher score, and y* would be lower, next time.

f(x, y)
V,V: 1
V,A: 1
V,finna: 1
V,get: 1
A,good: 1

f(x, y*)
N,V: 1
V,A: 1
N,finna: 1
V,get: 1
A,good: 1



f(x,y) - f(x, y*)
V,V: +1
N,V: -1
V,finna: +1
N,finna: -1

Perceptron update rule:

$$\theta := \theta + \eta[f(x, y) - f(x, y^*)]$$

Learning rate (hyperparam)

$$\theta := \theta + \eta[f(x, y) - f(x, y^*)]$$

Transition features

Observation features

θ

-0.6	-1.0	1.1	0.5	0.0	0.8	0.5	-1.3	-1.6	0.0	0.6	0.0	-0.2	-0.2	0.8	-1.0	0.1	-1.9	1.1	1.2	-0.1	-1.0	-0.1

$f(x, y)$

1	0	**2**	0	0	1	0	0	0	0	0	0	0	1	0	0	0	**0**	0	3	0	0	0

$f(x, y^*)$

1	0	**1**	0	0	1	0	0	0	0	0	0	0	1	0	0	0	**1**	0	3	0	0	0

The update vector:

η (

		+1														-1					

)

Structured Perceptron

- Initialize theta=0
- For each example (x,y) in dataset, iterating many times through the dataset:
  - Predict $y^* = \text{argmax}_y$ *theta*'f(x,y)
  - (If $y^* = y$, no update.)
  - Calculate feature differences and update theta.

$$\theta := \theta + \eta[f(x, y) - f(x, y^*)]$$

Averaged perceptron

- Perceptron is error-driven: update theta when there are mistakes.
- If the learning algorithm never gets 100% accuracy on the training set, it never converges. (The perceptron doesn't care about the *magnitude* of its mistakes! Unlike log-linear models.)
- One popular solution: *averaging*. Average all seen values of theta into a final solution.

$$\bar{\theta} = \frac{1}{n} \sum_{t} \theta^{(t)}$$

- *t* indexes every example. (If you pass through the dataset 5 times, you have 5*(num train examples) timesteps.)
- The above is inefficient, but there's a trick to make it much faster in practice.

- Added after lecture, will review on Thursday:
  - The three main structpred models are: (1) struct perceptron, (2) crf's, (3) structsvm's. All of them work the same at test time (decoding via the viterbi algorithm, by maximizing a linear goodness score). Only at training time are they different.
  - Averaged perceptron is probably the simplest to implement and use. Lots of practitioners in NLP who don't care about fancy machine learning often use it. I actually like CRF's myself because of they have a probabilistic interpretation, but that doesn't always matter. Training CRF's is slightly more complicated than struct perceptrons (not that much more complicated, but like a lecture's worth of material), so I figured we could skip it in this class.
  - Instead of averaging, you can also do early stopping: keep a development set and evaluate accuracy on it every iteration through the data. Choose the theta that did best. I don't know which method is better (different researchers may prefer different methods). Averaging has the advantage that there aren't really any hyperparameters to tune (well, the learning rate to a certain extent).
  - Why does averaging work? Theta is bouncing a lot around the space, because the perceptron doesn't know how to prefer solutions according to the magnitude of the errors it makes. The value of theta will be overfitted towards doing well on the most recent examples it's seen. If you average, you average away some of the noise. Averaging is used in other areas of machine learning too. It's a form of regularization.
  - Perceptron learning is actually a form of gradient descent. It's not on the logistic regression log-likelihood, but instead the gradients of a different function (the "1-0" loss).
    - The Collins 2002 paper that introduced the structured perceptron is still great to read for more details: http://www.cs.columbia.edu/~mcollins/papers/tagperc.pdf
  - More on the classification perceptron: see Hal Daume's book chapter draft, http://ciml.info/dl/v0_9/ciml-v0_9-ch03.pdf

Recap

- Discriminative sequence models give scoring for output structures.
- MEMMs combine Markovian hidden structure, plus features from words.
- Global structure prediction models allow any scoring of the hidden structure. The structured perceptron is a simple and effective learning algorithm for them.