Conditional Random Fields and the Structured Perceptron

CS 585, Fall 2017

Introduction to Natural Language Processing
http://people.cs.umass.edu/~brenocon/inlp2017

Brendan O'Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

Log-linear models (NB, LogReg, HMM, CRF...)

- x: Text Data
- y: Proposed class or sequence
- θ: Feature weights (model parameters)
- f(x,y): Feature extractor, produces feature vector

$$p(y|x) = \frac{1}{Z} \exp \underbrace{\left(\theta^{\mathsf{T}} f(x, y)\right)}_{G(y)}$$

Decision rule: $\arg \max_{y^* \in outputs(x)} G(y^*)$

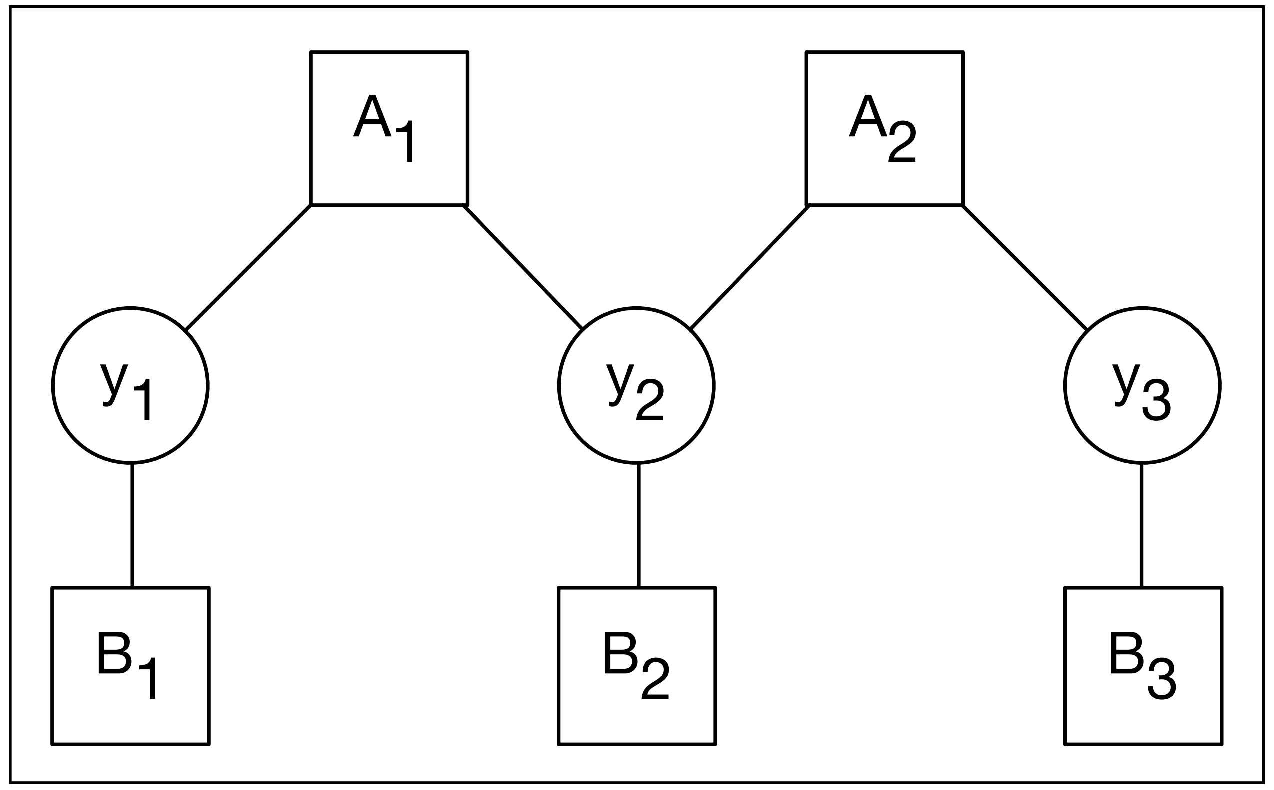
How to we evaluate for HMM/CRF? Viterbi!

Things to do with a log-linear model

$$p(y|x) = \frac{1}{Z} \exp \underbrace{\left(\theta^{\mathsf{T}} f(x, y)\right)}_{G(y)}$$

	f(x,y) Feature extractor (feature vector)	**x** Text Input	**y** Output	**θ** Feature weights
decoding/prediction $\arg\max_{y^* \in outputs(x)} G(y^*)$	given	given (just one)	obtain (just one)	given
parameter learning	given	given (many pairs)	given (many pairs)	obtain
feature engineering (human-in-the-loop)	fiddle with during experiments	given (many pairs)	given (many pairs)	obtain in each experiment

HMM as factor graph



$$p(y, w) = \prod_t p(w_y | y_t) \; p(y_{t+1} | y_t)$$

$$\log p(y, w) = \sum_t \log p(w_t | y_t) + \log p(y_t | y_{t-1})$$

↑ $G(y)$

goodness

↑ $B_t(y_t)$

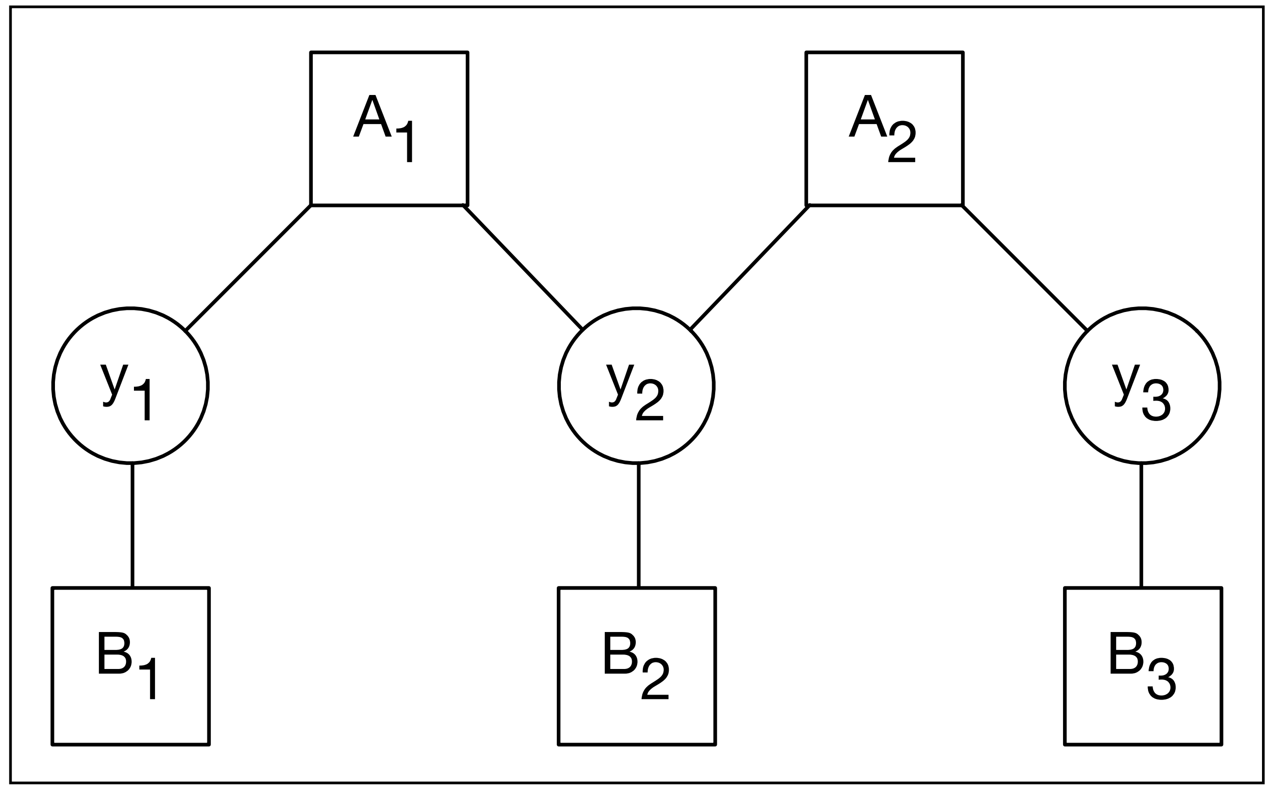
emission factor score

↑ $A(y_t, y_{t+1})$

transition factor score

(Additive) Viterbi: $\arg \max_{y^* \in outputs(x)} G(y^*)$

HMM as log-linear



$$p(y, w) = \prod_t p(w_y | y_t)\ p(y_{t+1} | y_t)$$

$$\log p(y, w) = \sum_t \log p(w_t | y_t) + \log p(y_t | y_{t-1})$$

$\uparrow$ $G(y)$ *goodness*

$\uparrow$ $B_t(y_t)$ *emission factor score*

$\uparrow$ $A(y_t, y_{t+1})$ *transition factor score*

G(y) =

$$\sum_t \left[\sum_{k \in K} \sum_{w \in V} \mu_{w,k} 1\{y_t = k \wedge w_t = w\} + \sum_{k,j \in K} \lambda_{j,k} 1\{y_t = j \wedge y_{t+1} = k\} \right]$$

=

$$\sum_t \sum_{i \in \text{allfeats}} \theta_i f_{t,i}(y_t, y_{t+1}, w_t)$$

=

$$\sum_{i \in \text{allfeats}} \theta_i f_i(y_t, y_{t+1}, w_t)$$

CRF

$$\log p(y|x) = C + \theta^\mathsf{T} f(x, y)$$

Prob. dist over *whole* sequence
(log-linear model of sequence output)

$$f(x, y) = \sum_t f_t(x, y_t, y_{t+1})$$

Linear-chain CRF: whole-sequence feature function decomposes into features over neighboring tags (Markovian: no long-distance effects)

- advantages
  - 1. Features: why just word identity features? add many more!
  - 2. Discriminative learning: can train it to optimize accuracy of sequence tagging
- Viterbi can be used for efficient prediction (due to locality assumption)

	finna	get	good
gold y =	V	V	A

Two simple feature templates

"Transition features"

$$f_{trans:A,B}(x,y) = \sum_t 1\{y_{t-1} = A, y_t = B\}$$

"Observation features"

$$f_{emit:A,w}(x,y) = \sum_t 1\{y_t = A, x_t = w\}$$

f(x,y) is...

V,V: 1
V,A: 1
V,N: 0
....
V,finna: 1
V,get: 1
A,good: 1
N,good: 0
...

	finna	get	good
gold y =	V	V	A

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

Transition features (first 7 entries) | Observation features (remaining 17 entries)

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

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

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

Goodness(y) = $\theta^{\mathsf{T}} f(x,y)$

CRF: prediction with Viterbi

$$\log p(y|x) = C + \theta^\mathsf{T} f(x,y)$$

Prob. dist over *whole* sequence

$$f(x,y) = \sum_t f_t(x, y_t, y_{t+1})$$

Linear-chain CRF: whole-sequence feature function decomposes into pairs

- Scoring function has *local decomposition*

$$f(x,y) = \sum_t^T f^{(B)}(t,x,y) + \sum_{t=2}^T f^{(A)}(y_{t-1}, y_t)$$

$$\theta^\mathsf{T} f(x,y) = \sum_t \theta^\mathsf{T} f^{(B)}(t,x,y) + \sum_{t=2}^T + f^{(A)}(y_{t-1}, y_t)$$

Structured/multiclass Perceptron

- For ~10 iterations
  - For each (x,y) in dataset
    - PREDICT

$$y^* = \arg\max_{y'} \theta^\mathsf{T} f(x, y')$$

    - IF y=y*, do nothing
    - ELSE update weights

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

learning rate constant
e.g. r=1

Features for
TRUE label

Features for
PREDICTED label

Update rule

y=POS
x="this awesome movie ..."
Make mistake: y*=NEG

$$\theta := \theta + \overset{\text{learning rate e.g. r=1}}{r}\underbrace{[\overset{\text{Features for TRUE label}}{f(x,y)} - \overset{\text{Features for PREDICTED label}}{f(x,y^*)}]}_{\text{f(x, POS) – f(x, NEG)}}$$

	POS_awesome	POS_this	POS_oof	....	NEG_awesome	NEG_this	NEG_oof	....
real f(x, POS) =	1	1	0	....	0	0	0	....
pred f(x, NEG) =	0	0	0	....	1	1	0	....
f(x, POS) – f(x, NEG) =	+1	+1	0	....	-1	-1	0	....

Update rule

$$\theta := \theta + \overset{\substack{\text{learning rate}\\ \text{e.g. r=1}}}{r}[\underbrace{\overset{\substack{\text{Features for}\\ \text{TRUE label}}}{f(x, y)} - \overset{\substack{\text{Features for}\\ \text{PREDICTED label}}}{f(x, y^*)}}_{\text{f(x, POS) – f(x, NEG)}}]$$

For each feature j in true y but not predicted y*:

$$\theta_j := \theta_j + (r) f_j(x, y)$$

For each feature j not in true y, but in predicted y*:

$$\theta_j := \theta_j - (r) f_j(x, y)$$

finna get good

gold y = V V A

Two simple feature templates

"Transition features"

$$f_{trans:A,B}(x,y) = \sum_t 1\{y_{t-1} = A, y_t = B\}$$

"Observation features"

$$f_{emit:A,w}(x,y) = \sum_t 1\{y_t = A, x_t = w\}$$

f(x,y) is...

V,V: 1
V,A: 1
V,N: 0
....
V,finna: 1
V,get: 1
A,good: 1
N,good: 0
...

	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) = $\theta^{\mathsf{T}} f(x,y)$

	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:

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

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

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:

+ r (			**+1**															**-1**					)

Perceptron notes/issues

- Issue: does it converge? (generally no)
  - Solution: the *averaged* perceptron
- Can you regularize it? No... just averaging...

- By the way, there's also *likelihood* training out there (gradient ascent on the log-likelihood function: the traditional way to train a CRF)
  - structperc is easier to implement/conceptualize and performs similarly in practice

Averaged perceptron

- To get stability for the perceptron: *Voted* perc or *Averaged* perc
- Averaging: For *t*'th example... average together vectors from every timestep

$$\bar{\theta}_t = \frac{1}{t} \sum_{t'=1}^{t} \theta_{t'}$$

- Efficiency?
  - Lazy update algorithm in HW (and Daume reading)