Discrete Sequential Models + General CRF

Kayhan Batmanghelich

Slides Credit: Matt Gormley (2016)
1. Data

\[ \mathcal{D} = \{ \mathbf{x}^{(n)} \}_{n=1}^{N} \]

2. Model

\[ p(\mathbf{x} \mid \theta) = \frac{1}{Z(\theta)} \prod_{C \in \mathcal{C}} \psi_{C}(\mathbf{x}_C) \]

3. Objective

\[ \ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} \mid \theta) \]

4. Learning

\[ \theta^* = \arg\max_{\theta} \ell(\theta; \mathcal{D}) \]

5. Inference

1. Marginal Inference

\[ p(\mathbf{x}_C) = \sum_{\mathbf{x}' : \mathbf{x}'_C = \mathbf{x}_C} p(\mathbf{x}' \mid \theta) \]

2. Partition Function

\[ Z(\theta) = \sum_{\mathbf{x}} \prod_{C \in \mathcal{C}} \psi_{C}(\mathbf{x}_C) \]

3. MAP Inference

\[ \hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{x} \mid \theta) \]
Today’s Lecture...

... is really about Conditional Random Fields (CRFs), but in the guise of two case studies:

1. Part-of-speech (POS) tagging
2. Image segmentation
Outline

1. Case Study: Supervised Part-of-speech tagging (NLP)
   - Hidden Markov Model (HMM)
   - Maximum-Entropy Markov Model (MEMM)
   - Linear-chain CRF
   - Digression: Minimum Bayes Risk (MBR) Decoding
   - Digression: Generative vs. Discriminative

2. Case Study: Image Segmentation (Computer Vision)
   - General CRF (e.g. grid)
   - Hidden-state CRF (HCRF)
1. CASE STUDY: SUPERVISED PART-OF-SPEECH TAGGING (NLP)

HMMs, MEMMs, Linear-chain CRFs
Dataset for Supervised Part-of-Speech (POS) Tagging

Data: \( D = \{ \mathbf{x}^{(n)}, \mathbf{y}^{(n)} \}_{n=1}^N \)

<table>
<thead>
<tr>
<th>Sample 1:</th>
<th>Sample 2:</th>
<th>Sample 3:</th>
<th>Sample 4:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Sample 1" /></td>
<td><img src="image2" alt="Sample 2" /></td>
<td><img src="image3" alt="Sample 3" /></td>
<td><img src="image4" alt="Sample 4" /></td>
</tr>
</tbody>
</table>

\[ \mathbf{D} = \{ \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \} \]

\[ \mathbf{D} = \{ \mathbf{x}^{(2)}, \mathbf{y}^{(2)} \} \]

\[ \mathbf{D} = \{ \mathbf{x}^{(3)}, \mathbf{y}^{(3)} \} \]

\[ \mathbf{D} = \{ \mathbf{x}^{(4)}, \mathbf{y}^{(4)} \} \]
Factors have local opinions (≥ 0)

Each black box looks at some of the tags $Y_i$ and words $X_i$

Note: We chose to reuse the same factors at different positions in the sentence.
Factors have local opinions ($\geq 0$)

Each black box looks at some of the tags $Y_i$ and words $X_i$

\[ p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = ? \]
Global probability = product of local opinions

Each black box looks at some of the tags $Y_i$ and words $X_i$

$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$

Uh-oh! The probabilities of the various assignments sum up to $Z > 1$. So divide them all by $Z$. 
Markov Random Field (MRF)

Joint distribution over tags $Y_i$ and words $X_i$
The individual factors aren’t necessarily probabilities.

$$p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$
Bayesian Networks

But sometimes we choose to make them probabilities. Constrain each row of a factor to sum to one. Now $Z = 1$.

$$p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z} \cdot (0.3 \cdot 0.8 \cdot 0.2 \cdot 0.5 \cdot \ldots)$$
Markov Random Field (MRF)

Joint distribution over tags $Y_i$ and words $X_i$

$$p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z}(4 \times 8 \times 5 \times 3 \times \ldots)$$
Conditional Random Field (CRF)

Conditional distribution over tags $Y_i$ given words $x_i$. The factors and $Z$ are now specific to the sentence $x$.

$$p(n, v, p, d, n \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 \ast 8 \ast 5 \ast 3 \ast \ldots)$$
Conditional Random Field (CRF)

Conditional distribution over tags $Y_i$ given words $x_i$. The factors and $Z$ are now specific to the sentence $x$.

$$p(n, v, p, d, n \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$

We say the variables $X_i$ have been "clamped" to their values $x_i$.

This is equivalent to multiplying in an "evidence potential" which is a point mass with all its weight on $X_i = x_i$. 
Forward-Backward Algorithm

- Sum-product BP on an HMM is called the **forward-backward algorithm**
- Max-product BP on an HMM is called the **Viterbi algorithm**
# Learning and Inference Summary

For discrete variables:

<table>
<thead>
<tr>
<th>Learning</th>
<th>Marginal Inference</th>
<th>MAP Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td>MEMM</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td>Linear-chain CRF</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
</tbody>
</table>
CRF Tagging Model

Could be verb or noun

Could be adjective or verb

Could be noun or verb
Forward algorithm = message passing (matrix-vector products)

Backward algorithm = message passing (matrix-vector products)

- Forward-backward is a message passing algorithm.
- It’s the simplest case of belief propagation.
So Let’s Review Forward-Backward …

find

preferred

tags

Could be verb or noun

Could be adjective or verb

Could be noun or verb
So Let’s Review Forward-Backward ...

• Show the possible *values* for each variable
So Let’s Review Forward-Backward …

Let’s show the possible values for each variable
One possible assignment
• Let’s show the possible values for each variable
• One possible assignment
• And what the 7 factors think of it …
Viterbi Algorithm: Most Probable Assignment

\[ P(X) = \frac{1}{Z_{01}} \prod C_{i} \psi_{i}(X_{i}) \]

- So \( p(v, a, n) = \frac{1}{Z} \times \text{product of 7 numbers} \)
- Numbers associated with edges and nodes of path
- Most probable assignment = path with highest product
Viterbi Algorithm: Most Probable Assignment

- So $p(v \ a \ n) = (1/Z) *$ product weight of one path
Forward-Backward Algorithm: Finds Marginals

\[ p(v a n) = \frac{1}{Z} \times \text{product weight of one path} \]

- Marginal probability \( p(Y_2 = a) \)
  
  \[ = \frac{1}{Z} \times \text{total weight of all paths through all} \]

- So \( p(v a n) \) is the product weight of a single path.
- Marginal probability \( p(Y_2 = a) \) is the total weight of all paths passing through \( a \).
Forward-Backward Algorithm: Finds Marginals

- So \( p(v \ a \ n) = (1/Z) \times \text{product weight of one path} \)
- Marginal probability \( p(Y_2 = a) = (1/Z) \times \text{total weight of all paths through } \Delta \n \)
Forward-Backward Algorithm: Finds Marginals

- So $p(v \ a \ n) = (1/Z) \times \text{product weight of one path}$
- Marginal probability $p(Y_2 = a) = (1/Z) \times \text{total weight of all paths through } v$
Forward-Backward Algorithm: Finds Marginals

- So \( p(v \ a \ n) = (1/Z) \times \) product weight of one path
- Marginal probability \( p(Y_2 = a) = (1/Z) \times \) total weight of all paths through the node \( n \)
Forward-Backward Algorithm: Finds Marginals

\[ \alpha_2(n) = \text{total weight of these path prefixes} \]

(found by dynamic programming: matrix-vector products)
Forward-Backward Algorithm: Finds Marginals

\[ \beta_2(n) = \text{total weight of these path suffixes} \]

(found by dynamic programming: matrix-vector products)
Forward-Backward Algorithm: Finds Marginals

\[ \alpha_2(n) = \text{total weight of these path prefixes} \]
\[ (a + b + c) \]

\[ \beta_2(n) = \text{total weight of these path suffixes} \]
\[ (x + y + z) \]

Product gives \[ ax + ay + az + bx + by + bz + cx + cy + cz \] = total weight of paths
Oops! The weight of a path through a state also includes a weight at that state. So \( \alpha(n) \cdot \beta(n) \) isn’t enough. The extra weight is the opinion of the unigram factor at this variable.

Forward-Backward Algorithm: Finds Marginals

\[
P(Y_2 = n) = \alpha_2(n) \beta_2(n) \psi_2(n)
\]

“belief that \( Y_2 = n \)”

The total weight of all paths through \( n \) is

\[
\alpha_2(n) \psi_{\{2\}}(n) \beta_2(n)
\]
Forward-Backward Algorithm: Finds Marginals

“belief that $Y_2 = v$”
“belief that $Y_2 = n$”

$$\psi\{2\}(v)$$

preferred

total weight of all paths through

$$= \alpha_2(v) \psi\{2\}(v) \beta_2(v)$$
Forward-Backward Algorithm: Finds Marginals

divide by $Z=6$ to get marginal probs

“belief that $Y_2 = v$”

“belief that $Y_2 = n$”

“belief that $Y_2 = a$”

Sum = $Z$ (total probability of all paths)

total weight of all paths through $a$

= $\alpha_2(a) \ psi_{\{2\}}(a) \ beta_2(a)$
Hidden Markov Model

\[
P(x_{1:n}, y_{1:n}) = \prod_{i=1}^{n} P(x_i | y_i) P(y_i | y_{i-1})
\]
Shortcomings of Hidden Markov Model (1): locality of features

- HMM models capture dependences between each state and only its corresponding observation
  - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, indentation, amount of white space, etc.

- Mismatch between learning objective function and prediction objective function
  - HMM learns a joint distribution of states and observations $P(Y, X)$, but in a prediction task, we need the conditional probability $P(Y|X)$. 

© Eric Xing @ CMU, 2005-2015
A Solution: Maximum Entropy Markov Model (MEMM)

Why not providing the full observation sequence explicitly
- More expressive than HMMs (not the direction of arrow – no causal interpretation, X is just covariates)

Discriminative model
- Completely ignores modeling P(X): saves modeling effort
- Learning objective function consistent with predictive function: P(Y|X)
Then, shortcomings of MEMM (and HMM) (2): the Label bias problem

What the local transition probabilities say:

- State 1 almost always prefers to go to state 2
- State 2 almost always prefers to stay in state 2
MEMM: the Label bias problem

### Graphical Representation

State 1 → Observation 1
State 2 → Observation 2
State 3 → Observation 3
State 4 → Observation 4

**Path Probabilities**

<table>
<thead>
<tr>
<th>Path</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 → 1 → 1 → 1</td>
<td></td>
</tr>
</tbody>
</table>
MEMM: the Label bias problem

Path | Probability
---|---
1 → 1 → 1 → 1 | 0.4 x 0.45 x 0.5 = 0.090
MEMM: the Label bias problem

<table>
<thead>
<tr>
<th>Path</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 → 1 → 1 → 1</td>
<td>0.4 x 0.45 x 0.5 = 0.090</td>
</tr>
<tr>
<td>2 → 2 → 2 → 2</td>
<td>0.2 x 0.30 x 0.3 = 0.018</td>
</tr>
</tbody>
</table>
MEMM: the Label bias problem

Path  | Probability     
------|----------------- 
1 → 1 → 1 → 1 | 0.4 x 0.45 x 0.5 = 0.090 
2 → 2 → 2 → 2 | 0.2 x 0.30 x 0.3 = 0.018 
1 → 2 → 1 → 2 | 0.6 x 0.20 x 0.5 = 0.060
MEMM: the Label bias problem

Path | Probability
--- | ---
1 → 1 → 1 → 1 | 0.4 x 0.45 x 0.5 = 0.090
2 → 2 → 2 → 2 | 0.2 x 0.30 x 0.3 = 0.018
1 → 2 → 1 → 2 | 0.6 x 0.20 x 0.5 = 0.060
1 → 1 → 2 → 2 | 0.4 x 0.55 x 0.3 = 0.066
MEMM: the Label bias problem

### Path Probability Calculations

<table>
<thead>
<tr>
<th>Path</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 → 1 → 1 → 1</td>
<td>0.4 x 0.45 x 0.5 = 0.090</td>
</tr>
<tr>
<td>2 → 2 → 2 → 2</td>
<td>0.2 x 0.30 x 0.3 = 0.018</td>
</tr>
<tr>
<td>1 → 2 → 1 → 2</td>
<td>0.6 x 0.20 x 0.5 = 0.060</td>
</tr>
<tr>
<td>1 → 1 → 2 → 2</td>
<td>0.4 x 0.55 x 0.3 = 0.066</td>
</tr>
</tbody>
</table>
MEMM: the Label bias problem

Yet locally it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.

Most likely path

<table>
<thead>
<tr>
<th>Path</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 → 1 → 1 → 1</td>
<td>0.4 x 0.45 x 0.5 = 0.090</td>
</tr>
<tr>
<td>2 → 2 → 2 → 2</td>
<td>0.2 x 0.30 x 0.3 = 0.018</td>
</tr>
<tr>
<td>1 → 2 → 1 → 2</td>
<td>0.6 x 0.20 x 0.5 = 0.060</td>
</tr>
<tr>
<td>1 → 1 → 2 → 2</td>
<td>0.4 x 0.55 x 0.3 = 0.066</td>
</tr>
</tbody>
</table>
MEMM: the Label bias problem

Yet locally it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.

Why does this happen?
• State 1 has only two transitions but state 2 has 5
• Average transition probability from state 2 is lower

This is the Label Bias Problem in MEMM: a preference for states with lower number of transitions over others.
Solution:
Do not normalize probabilities locally

From local probabilities…
Solution:
Do not normalize probabilities locally

From local probabilities to local potentials!
States with lower transitions do not have an unfair advantage!
From MEMM ....

\[ P(y_{1:n} | x_{1:n}) = \prod_{i=1}^{n} P(y_i | y_{i-1}, x_{1:n}) = \prod_{i=1}^{n} \frac{\exp(w^T f(y_i, y_{i-1}, x_{1:n}))}{Z(y_{i-1}, x_{1:n})} \]
From MEMM to Linear-chain CRF

\[ P(x_1, x_2, x_3) = P(x_1) P(x_2|x_1) P(x_3|x_2) \]

- CRF is a partially directed model
  - Discriminative model like MEMM
  - Unlike MEMM, each factor is not normalized. Hence, usage of global \( Z(x) \) overcomes the label bias problem of MEMM
  - Models the dependence between each state and the entire observation sequence (like MEMM)

\[
P(y_{1:n} | x_{1:n}) = \frac{1}{Z(x_{1:n})} \prod_{i=1}^{n} \phi(y_i, y_{i-1}, x_{1:n}) = \frac{1}{Z(x_{1:n}, w)} \prod_{i=1}^{n} \exp(w^T f(y_i, y_{i-1}, x_{1:n}))
\]
Linear-chain CRF

- Linear-chain Conditional Random Field parametric form:

\[
P(y|x) = \frac{1}{Z(x, \lambda, \mu)} \exp\left(\sum_{i=1}^{n} \left( \lambda_k f_k(y_i, y_{i-1}, x) + \sum_{l} \mu_l g_l(y_i, x) \right) \right)
\]

where

\[
Z(x, \lambda, \mu) = \sum_{y} \exp\left(\sum_{i=1}^{n} (\lambda^T f(y_i, y_{i-1}, x) + \mu^T g(y_i, x)) \right)
\]

\[x \rightarrow Y_1 \rightarrow Y_2 \rightarrow \ldots \rightarrow Y_n \rightarrow x_{1:n}\]
Whiteboard

• CRF model
• CRF data log-likelihood
• CRF derivatives

(side-by-side with MRF)
Learning and Inference Summary

For discrete variables:

<table>
<thead>
<tr>
<th></th>
<th>Learning</th>
<th>Marginal Inference</th>
<th>MAP Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HMM</strong></td>
<td>Just counting</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>MEMM</strong></td>
<td>Gradient based – decomposes and doesn’t require inference (GLM)</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>Linear-chain CRF</strong></td>
<td>Gradient based – doesn’t decompose because of $Z(\mathbf{x})$ and requires marginal inference</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
</tbody>
</table>
Features

General idea:

• Make a list of interesting substructures.
• The feature $f_k(x,y)$ counts tokens of $k^{th}$ substructure in $(x,y)$. 
Features for tagging ...

Time flies like an arrow

- Count of tag P as the tag for “like”

Weight of this feature is like log of an emission probability in an HMM
Features for tagging ...

- Count of tag P as the tag for “like”
- Count of tag P

Time flies like an arrow
Features for tagging ...

- Count of tag P as the tag for “like”
- Count of tag P
- Count of tag P in the middle third of the sentence
Features for tagging ...

- Count of tag P as the tag for “like”
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P

Time flies like an arrow

Weight of this feature is like log of a transition probability in an HMM
Features for tagging ...

- Count of tag P as the tag for “like”
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by “an”

*Time flies like an•arrow*
Features for tagging ...

- Count of tag P as the tag for “like”
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by “an”
- Count of tag bigram V P where P is the tag for “like”
Features for tagging ...

- Count of tag P as the tag for “like”
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by “an”
- Count of tag bigram V P where P is the tag for “like”
- Count of tag bigram V P where both words are lowercase

*Time flies like an arrow*
Features for tagging ...

- Count of tag trigram N V P?
  - A bigram tagger can only consider within-bigram features: only look at 2 adjacent blue tags (plus arbitrary red context).
  - So here we need a trigram tagger, which is slower.
  - Why? The forward-backward states would remember *two* previous tags.

We take this arc once per N V P triple, so its weight is the total weight of the features that fire on that triple.
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes ("basic features") that you can compute at each position in \((x,y)\).

For position \(i\) in a tagging, these might include:

- Full name of tag \(i\)
- First letter of tag \(i\) (will be "N" for both "NN" and "NNS")
- Full name of tag \(i-1\) (possibly BOS); similarly tag \(i+1\) (possibly EOS)
- Full name of word \(i\)
- Last 2 chars of word \(i\) (will be "ed" for most past-tense verbs)
- First 4 chars of word \(i\) (why would this help?)
- "Shape" of word \(i\) (lowercase/capitalized/all caps/numeric/...)
- Whether word \(i\) is part of a known city name listed in a "gazetteer"
- Whether word \(i\) appears in thesaurus entry \(e\) (one attribute per \(e\))
- Whether \(i\) is in the middle third of the sentence
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes (“basic features”) that you can compute at each position in \((x,y)\).
2. Now conjoin them into various “feature templates.”

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))\).

At each position of \((x,y)\), exactly one of the many template7 features will fire:

At \(i=1\), we see an instance of “template7=(BOS,N,-es)” so we add one copy of that feature’s weight to \(\text{score}(x,y)\).
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes ("basic features") that you can compute at each position in \((x,y)\).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))\).

At each position of \((x,y)\), exactly one of the many template7 features will fire:

```
[Time flies like an arrow]
```

At \(i=2\), we see an instance of "template7=(N,V,-ke)" so we add one copy of that feature’s weight to \(\text{score}(x,y)\).
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes ("basic features") that you can compute at each position in \((x,y)\).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix}2(i+1))\).

At each position of \((x,y)\), exactly one of the many template7 features will fire:

Time flies like an arrow

At \(i=3\), we see an instance of "template7\(=(N,V,-\text{an})\)" so we add one copy of that feature’s weight to \(\text{score}(x,y)\).
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes ("basic features") that you can compute at each position in \((x,y)\).

2. Now **conjoin** them into various "feature templates."

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))\).

At each position of \((x,y)\), exactly one of the many template7 features will fire:

```
N V P D N
```

Time flies like an arrow

At \(i=4\), we see an instance of "template7\((P,D,-ow)\)" so we add one copy of that feature’s weight to \(score(x,y)\).
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes (“basic features”) that you can compute at each position in \((x,y)\).
2. Now conjoin them into various “feature templates.”

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix}2(i+1))\).

At each position of \((x,y)\), exactly one of the many template7 features will fire:

```
N  V  P  D  N
```

Time flies like an arrow

At \(i=5\), we see an instance of “template7=(D,N,-)” so we add one copy of that feature’s weight to \(score(x,y)\).
How might you come up with the features that you will use to score \((x, y)\)?

1. Think of some attributes (“basic features”) that you can compute at each position in \((x, y)\).
2. Now conjoin them into various “feature templates.”

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix}2(i+1))\).

This template gives rise to many features, e.g.:

\[
\begin{align*}
\text{score}(x, y) &= \ldots \\
+ \theta[\text{“template7=(P,D,-ow)”}] \ast \text{count(“template7=(P,D,-ow)”)} \\
+ \theta[\text{“template7=(D,D,-xx)”}] \ast \text{count(“template7=(D,D,-xx)”)} \\
+ \ldots
\end{align*}
\]

With a handful of feature templates and a large vocabulary, you can easily end up with millions of features.
How might you come up with the features that you will use to score \((x,y)\)?

1. Think of some attributes (“basic features”) that you can compute at each position in \((x,y)\).
2. Now conjoin them into various “feature templates.”

E.g., template 7 might be \((\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))\).

Note: Every template should mention at least some blue.

- Given an input \(x\), a feature that only looks at red will contribute the same weight to score\((x,y_1)\) and score\((x,y_2)\).
- So it can’t help you choose between outputs \(y_1, y_2\).
Liang & Jordan (ICML 2008) compares HMM and CRF with identical features

• Dataset 1: (Real)
  – WSJ Penn Treebank (38K train, 5.5K test)
  – 45 part-of-speech tags

• Dataset 2: (Artificial)
  – Synthetic data generated from HMM learned on Dataset 1 (1K train, 1K test)

• Evaluation Metric: Accuracy
CRFs: some empirical results

- Parts of Speech tagging

<table>
<thead>
<tr>
<th>model</th>
<th>error</th>
<th>oov error</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>5.69%</td>
<td>45.99%</td>
</tr>
<tr>
<td>MEMM</td>
<td>6.37%</td>
<td>54.61%</td>
</tr>
<tr>
<td>CRF</td>
<td>5.55%</td>
<td>48.05%</td>
</tr>
<tr>
<td>MEMM*</td>
<td>4.81%</td>
<td>26.99%</td>
</tr>
<tr>
<td>CRF*</td>
<td>4.27%</td>
<td>23.76%</td>
</tr>
</tbody>
</table>

+ Using spelling features

- Using same set of features: HMM >= CRF > MEMM
- Using additional overlapping features: CRF* > MEMM* >> HMM
Minimum Bayes Risk Decoding

• Suppose we given a loss function \( l(y', y) \) and are asked for a single tagging

• How should we choose just one from our probability distribution \( p(y|x) \)?

• A minimum Bayes risk (MBR) decoder \( h(x) \) returns the variable assignment with minimum expected loss under the model’s distribution

\[
h_\theta(x) = \arg\min_{\hat{y}} \mathbb{E}_{y \sim p_\theta(\cdot|x)} [l(\hat{y}, y)]
\]

\[
= \arg\min_{\hat{y}} \sum_{y} p_\theta(y \mid x) l(\hat{y}, y)
\]
Minimum Bayes Risk Decoding

\[ h_{\theta}(x) = \arg\min_{\hat{y}} \mathbb{E}_{y \sim p_{\theta}(\cdot|\cdot)}[\ell(\hat{y}, y)] \]

Consider some example loss functions:

The **0-1 loss function** returns 1 only if the two assignments are identical and 0 otherwise:

\[ \ell(\hat{y}, y) = 1 - \mathbb{I}(\hat{y}, y) \]

The MBR decoder is:

\[ h_{\theta}(x) = \arg\min_{\hat{y}} \sum_{y} p_{\theta}(y|\cdot|x) \left(1 - \mathbb{I}(\hat{y}, y)\right) \]

\[ = \arg\max_{\hat{y}} p_{\theta}(\hat{y}|x) \]

which is exactly the MAP inference problem!
Minimum Bayes Risk Decoding

\[ h_\theta(x) = \arg\min_{\hat{y}} \mathbb{E}_{y \sim p_\theta(x)}[\ell(\hat{y}, y)] \]

Consider some example loss functions:

The **Hamming loss** corresponds to accuracy and returns the number of incorrect variable assignments:

\[
\ell(\hat{y}, y) = \sum_{i=1}^{V} (1 - \mathbb{I}(\hat{y}_i, y_i))
\]

The MBR decoder is:

\[
\hat{y}_i = h_\theta(x)_i = \arg\max_{\hat{y}_i} p_\theta(\hat{y}_i \mid x)
\]

This decomposes across variables and requires the variable marginals.
General CRFs, Hidden-state CRFs

2. CASE STUDY: IMAGE SEGMENTATION (COMPUTER VISION)
Other CRFs

- So far we have discussed only 1-dimensional chain CRFs
  - Inference and learning: exact

- We could also have CRFs for arbitrary graph structure
  - E.g: Grid CRFs
  - Inference and learning no longer tractable
  - Approximate techniques used
    - MCMC Sampling
    - Variational Inference
    - Loopy Belief Propagation
  - We will discuss these techniques soon
Applications of CRF in Vision

Stereo Matching

Image Restoration

Image Segmentation
Application: Image Segmentation

\[ \phi_i(y_i, x) \in \mathbb{R}^{\approx 1000} : \text{local image features, e.g. bag-of-words} \]
\[ \rightarrow \langle w_i, \phi_i(y_i, x) \rangle : \text{local classifier (like logistic-regression)} \]
\[ \phi_{i,j}(y_i, y_j) = [y_i = y_j] \in \mathbb{R}^1 : \text{test for same label} \]
\[ \rightarrow \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle : \text{penalizer for label changes (if } w_{ij} > 0) \]
combined: \[ \arg\max_y p(y|x) \text{ is smoothed version of local cues} \]
Application: Handwriting Recognition

\[ \phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}: \text{image representation (pixels, gradients)} \]

\[ \rightarrow \langle w_i, \phi_i(y_i, x) \rangle: \text{local classifier if } x_i \text{ is letter } y_i \]

\[ \phi_{i,j}(y_i, y_j) = e_{y_i} \otimes e_{y_j} \in \mathbb{R}^{26 \times 26}: \text{letter/letter indicator} \]

\[ \rightarrow \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle: \text{encourage/suppress letter combinations} \]

combined: \[ \arg\max_y p(y|x) \] is "corrected" version of local cues

\[ \text{local classification} \]

\[ \text{local + "correction"} \]
Application: Pose Estimation

Local classifier for each part

Penalizes unrealistic poses

\[ p(l|x) \propto \exp \left[ \sum_{ij} \theta_{ij}^T \phi_{ij}(l_i, l_j, x) + \sum_i \theta_i^T \phi_i(l_i, x) \right] = e^{\theta^T \phi(l,x)}. \]

\[ \arg\max_y p(y|x) \] is cleaned up version of local prediction

Cascaded Models for Articulated Pose Estimation, B. Sapp, A. Toshev, B. Taskar
Feature Functions for CRF in Vision

$\phi_i(y_i, x)$: local representation, high-dimensional
→ $\langle w_i, \phi_i(y_i, x) \rangle$: local classifier

$\phi_{i,j}(y_i, y_j)$: prior knowledge, low-dimensional
→ $\langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalize outliers

Learning adjusts parameters:
- unary $w_i$: learn local classifiers and their importance
- binary $w_{ij}$: learn importance of smoothing/penalization

$\arg\max_y p(y|x)$ is cleaned up version of local prediction
Case Study: Image Segmentation

- Image segmentation (FG/BG) by modeling of interactions btw RVs
  - Images are noisy.
  - Objects occupy continuous regions in an image.

\[ \text{Unary Term} \quad \begin{cases} \sum_{i \in S} V_i(y_i, X) \quad \text{for each } i \in S \end{cases} \]

\[ \text{Pairwise Term} \quad \sum_{i \in S} \sum_{j \in N_i} V_{i,j}(y_i, y_j) \]

\[ Y^* = \arg \max_{y \in \{0,1\}^n} \left[ \sum_{i \in S} V_i(y_i, X) + \sum_{i \in S} \sum_{j \in N_i} V_{i,j}(y_i, y_j) \right]. \]

- Input image
- Pixel-wise separate optimal labeling
- Locally-consistent joint optimal labeling

[Nowozin,Lampert 2012]

© Eric Xing @ CMU, 2005-2015
Grid CRF

• Suppose we want to image segmentation using a grid model
Grid CRF

- Suppose we want to image segmentation using a grid model

Assuming we divide into foreground / background, each factor is a table with $2^2$ entries.
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

• Suppose we want to image segmentation using a grid model
• What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?

This new factor has $2^5$ entries
Grid CRF

• Suppose we want to image segmentation using a grid model
• What happens when we run variable elimination?

For an MxM grid the new factor has $2^M$ entries
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?

For an \( M \times M \) grid the new factor has \( 2^M \) entries

In general, for high treewidth graphs like this, we turn to approximate inference (which we’ll cover soon!)
Case Study: Object Recognition

Data consists of images $x$ and labels $y$. 

- pigeon
- rhinoceros
- leopard
- llama
Case Study: Object Recognition

Data consists of images $x$ and labels $y$.

- Preprocess data into “patches”
- Posit a latent labeling $z$ describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- $z$ is not observed at train or test time
Case Study: Object Recognition

Data consists of images $x$ and labels $y$.

- Preprocess data into “patches”
- Posit a latent labeling $z$ describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- $z$ is not observed at train or test time
Case Study: Object Recognition

Data consists of images $x$ and labels $y$.

- Preprocess data into “patches”
- Posit a latent labeling $z$ describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- $z$ is not observed at train or test time
Hidden-state CRFs

Data: \[ D = \{ x^{(n)}, y^{(n)} \}_{n=1}^N \]

Joint model: \[ p_\theta(y, z | x) = \frac{1}{Z(x, \theta)} \prod_\alpha \psi_\alpha(y_\alpha, z_\alpha, x) \]

Marginalized model: \[ p_\theta(y | x) = \sum_z p_\theta(y, z | x) \]
Hidden-state CRFs

Data: \[ \mathcal{D} = \{ x^{(n)}, y^{(n)} \}_{n=1}^{N} \]

Joint model: \[ p_{\theta}(y, z \mid x) = \frac{1}{Z(x, \theta)} \prod_{\alpha} \psi_{\alpha}(y_{\alpha}, z_{\alpha}, x) \]

Marginalized model: \[ p_{\theta}(y \mid x) = \sum_{z} p_{\theta}(y, z \mid x) \]

We can train using gradient based methods:
(\text{the values } x \text{ are omitted below for clarity})

\[ \frac{d\ell(\theta | \mathcal{D})}{d\theta} = \sum_{n=1}^{N} \left( \mathbb{E}_{z \sim p_{\theta}(\cdot \mid y^{(n)})} [f_{j}(y^{(n)}, z)] - \mathbb{E}_{y,z \sim p_{\theta}(\cdot, \cdot)} [f_{j}(y, z)] \right) \]

\[ = \sum_{n=1}^{N} \sum_{\alpha} \left( \sum_{z_{\alpha}} p_{\theta}(z_{\alpha} \mid y^{(n)}) f_{\alpha,j}(y^{(n)}, z_{\alpha}) - \sum_{y_{\alpha},z_{\alpha}} p_{\theta}(y_{\alpha}, z_{\alpha}) f_{\alpha,j}(y_{\alpha}, z_{\alpha}) \right) \]

\text{Inference on clamped factor graph}

\text{Inference on full factor graph}
## Learning and Inference Summary

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Learning Method</th>
<th>Marginal Inference</th>
<th>MAP Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HMM</strong></td>
<td>Just counting</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>MEMM</strong></td>
<td>Gradient based – decomposes and doesn’t require inference (GLIM)</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>Linear-chain CRF</strong></td>
<td>Gradient based – doesn’t decompose because of $Z(x)$ and requires marginal inference</td>
<td>Forward-backward</td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>General CRF</strong></td>
<td>Gradient based – doesn’t decompose because of $Z(x)$ and requires (approximate) marginal inference</td>
<td>(approximate methods)</td>
<td>(approximate methods)</td>
</tr>
<tr>
<td><strong>HCRF</strong></td>
<td>Gradient based – same as General CRF</td>
<td>(approximate methods)</td>
<td>(approximate methods)</td>
</tr>
</tbody>
</table>
Summary

• HMM:
  – Pro: Easy to train
  – Con: Misses out on rich features of the observations

• MEMM:
  – Pro: Fast to train and supports rich features
  – Con: Suffers (like the HMM) from the label bias problem

• Linear-chain CRF:
  – Pro: Defeats the label bias problem with support for rich features
  – Con: Slower to train

• MBR Decoding:
  – the principled way to account for a loss function when decoding from a probabilistic model

• Generative vs. Discriminative:
  – gen. is better if the model is well-specified
  – disc. is better if the model is misspecified

• General CRFs:
  – Exact inference won’t suffice for high treewidth graphs
  – More general topologies can capture intuitions about variable dependencies

• HCRF:
  – Training looks very much like CRF training
  – Incorporation of hidden variables can model domain specific knowledge