Administtrivia

› Mini 1 graded by next lecture
› Project 1 is out, sample writeups on website

Recall: HMMs

› Input $x = (x_1, \ldots, x_n)$  
  Output $y = (y_1, \ldots, y_n)$

\[
P(y, x) = P(y_1) \prod_{i=2}^{n} P(y_i | y_{i-1}) \prod_{i=1}^{n} P(x_i | y_i)
\]

› Training: maximum likelihood estimation (with smoothing)
› Inference problem: $\arg \max_y P(y|x) = \arg \max_y \underbrace{P(y, x)}_{P_{\text{top}}}$
› Viterbi: $\text{score}_i(s) = \max_{y_{i-1}} P(s | y_{i-1}) P(x_i | s) \text{score}_{i-1}(y_{i-1})$

This Lecture

› CRFs: model (+features for NER), inference, learning
› Named entity recognition (NER)
› (if time) Beam search
Named Entity Recognition

BIO tagset: begin, inside, outside

Sequence of tags — should we use an HMM?

Why might an HMM not do so well here?
  ▶ Lots of O’s, so tags aren’t as informative about context
  ▶ Insufficient features/capacity with multinomials (especially for unks)

CRFs

Conditional Random Fields

- HMMs are expressible as Bayes nets (factor graphs)

  \[
  P(y, x) = P(y_1)P(x_1|y_1)P(y_2|y_1)P(x_2|y_2) \ldots
  \]

  This reflects the following decomposition:

  \[
  P(y, x) = P(y_1)P(x_1|y_1)P(y_2|y_1)P(x_2|y_2) \ldots
  \]

  Locally normalized model: each factor is a probability distribution that normalizes

- CRFs: discriminative models with the following globally-normalized form:

  \[
  P(y|x) = \frac{1}{Z} \prod_k \exp(\phi_k(x, y))
  \]

  any real-valued scoring function of its arguments

- Naive Bayes: logistic regression :: HMMs : CRFs
  local vs. global normalization <-> generative vs. discriminative

- Locally normalized discriminative models do exist (MEMMs)

- How do we max over y? Intractable in general — can we fix this?
Sequential CRFs

- **HMMs**: $P(y, x) = P(y_1)P(x_1|y_1)P(y_2|y_1)P(x_2|y_2)\ldots$

- **CRFs**: $P(y|x) \propto \prod_k \exp(\phi_k(x, y))$

$$P(y|x) \propto \exp(\phi_o(y_1)) \prod_{i=2}^n \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^n \exp(\phi_e(x_i, y_i))$$

- **Notation**: omit $x$ from the factor graph entirely (implicit)

- **Feature Functions**: $P(y|x) = \frac{1}{Z} \prod_{i=2}^n \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^n \exp(\phi_e(y_i, i, x))$

$$P(y|x) \propto \exp \left[ \sum_{i=2}^n f_t(y_{i-1}, y_i) + \sum_{i=1}^n f_e(y_i, i, x) \right]$$

- **Phis can be almost anything!** Here we use linear functions of sparse features

- **Don’t include initial distribution, can bake into other factors**

- **Looks like our single weight vector multiclass logistic regression model**

- **We condition on $x$, so every factor can depend on all of $x$ (including transitions, but we won’t do this)**

- **$y$ can’t depend arbitrarily on $x$ in a generative model**

Token index — lets us look at current word
Basic Features for NER

\[ P(y|x) \propto \exp w^T \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right] \]

Barack Obama will travel to **Hangzhou** today for the G20 meeting.

Transitions: \( f_t(y_{i-1}, y_i) = \text{Ind}[y_{i-1} \land y_i] = \text{Ind}[O \land B-LOC] \)

Emissions: \( f_e(y_{i, 6}, 6, x) = \text{Ind}[B-LOC \& \text{Current word} = \text{Hangzhou}] \)
\( \text{Ind}[B-LOC \& \text{Prev word} = \text{to}] \)

Features for NER

- Word features (can use in HMM)
  - Capitalization
  - Word shape
  - Prefixes/suffixes
  - Lexical indicators
- Context features (can’t use in HMM!)
  - Words before/after
  - Tags before/after
  - Word clusters
  - Gazetteers

CRFs Outline

\[ P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \]

\[ P(y|x) \propto \exp w^T \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right] \]

- Inference
- Learning
Computing (arg)maxes

\[
P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x))
\]

\[
\phi_t(y_{1:n}, x) = \max_{y_1, \ldots, y_n} e^{\phi_t(y_{n-1}, y_n)} e^{\phi_e(y_n, n, x)} \ldots e^{\phi_t(y_2, y_1)} e^{\phi_e(y_1, 1, x)}
\]

Inference in General CRFs

- Can do inference in any tree-structured CRF

Max-product algorithm: generalization of Viterbi to arbitrary tree-structured graphs (sum-product is generalization of forward-backward)

CRFs Outline

- Model: \( P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \)

\[
P(y|x) \propto \exp w^T \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right]
\]

- Inference: argmax \( P(y|x) \) from Viterbi

- Learning

Training CRFs

\[
P(y|x) \propto \exp w^T \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right]
\]

- Logistic regression: \( P(y|x) \propto \exp w^T f(x, y) \)

- Maximize \( L(y^*, x) = \log P(y^*|x) \)

- Gradient is completely analogous to logistic regression:

\[
\frac{\partial}{\partial w} L(y^*, x) = \sum_{i=2}^{n} f_t(y_{i-1}^*, y_i^*) + \sum_{i=1}^{n} f_e(y_i^*, i, x)
\]

\[
- E_y \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right]
\]

intractable!
### Training CRFs

\[
\frac{\partial}{\partial w} \mathcal{L}(y^*, x) = \sum_{i=2}^{n} f_t(y_{i-1}^*, y_i^*) + \sum_{i=1}^{n} f_e(y_i^*, i, x) - \mathbb{E}_y \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right]
\]

- Let's focus on emission feature expectation

\[
\mathbb{E}_y \left[ \sum_{i=1}^{n} f_e(y_i, i, x) \right] = \sum_{i \in \mathcal{Y}} P(y_i|x) \left[ \sum_{i=1}^{n} f_e(y_i, i, x) \right] = \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} P(y_i|x) f_e(y_i, i, x)
\]

\[
= \sum_{i=1}^{n} \sum_{s} P(y_i = s|x) f_e(s, i, x)
\]

### Computing Marginals

\[
P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x))
\]

- Normalizing constant
- \[Z = \sum_{y} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x))\]

- Analogous to \(P(x)\) for HMMs

- For both HMMs and CRFs:

\[
P(y_i = s|x) = \frac{\text{forward}_i(s) \text{backward}_i(s)}{\sum_{s'} \text{forward}_i(s') \text{backward}_i(s')} \text{ for HMMs}
\]

### Posterior vs. Probabilities

\[
P(y_i = s|x) = \frac{\text{forward}_i(s) \text{backward}_i(s)}{\sum_{s'} \text{forward}_i(s') \text{backward}_i(s')}
\]

- Posterior is derived from the parameters and the data (conditioned on \(x\)!

- for HMMs:

\[
P(x_i|y_i), P(y_i|y_{i-1})
\]

- HMM: Model parameter (usually multinomial distribution)

- CRF: Undefined (model is by definition conditioned on \(x\))

- Inferred quantity from forward-backward

- gold features — expected features under model

### Training CRFs

- For emission features:

\[
\frac{\partial}{\partial w} \mathcal{L}(y^*, x) = \sum_{i=1}^{n} f_e(y_i^*, i, x) - \sum_{i=1}^{n} \sum_{s} P(y_i = s|x) f_e(s, i, x)
\]

- Transition features: need to compute \(P(y_i = s_1, y_{i+1} = s_2|x)\) using forward-backward as well

- ...but you can build a pretty good system without transition features
CRFs Outline

- **Model:**
  \[ P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \]
  \[ P(y|x) \propto \exp \left( w^T \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right] \right) \]

- **Inference:** argmax \( P(y|x) \) from Viterbi

- **Learning:** run forward-backward to compute posterior probabilities; then
  \[ \frac{\partial}{\partial w} \mathcal{L}(y^*, x) = \sum_{i=1}^{n} f_e(y^*_i, i, x) - \sum_{i=1}^{n} \sum_{s} P(y_i = s|x) f_e(s, i, x) \]

Pseudocode

- for each epoch
  - for each example
    - extract features on each emission and transition (look up in cache)
    - compute potentials \( \phi \) based on features + weights
    - compute marginal probabilities with forward-backward
    - accumulate gradient over all emissions and transitions

Implementation Tips for CRFs

- Caching is your friend! Cache feature vectors especially
  
- Try to reduce redundant computation, e.g. if you compute both the gradient and the objective value, don’t rerun the dynamic program
  
- Exploit sparsity in feature vectors where possible, especially in feature vectors and gradients
  
- Do all dynamic program computation in log space to avoid underflow
  
- If things are too slow, run a profiler and see where time is being spent. Forward-backward should take most of the time

Debugging Tips for CRFs

- Hard to know whether inference, learning, or the model is broken!
  
- Compute the objective — is optimization working?
  
  - **Inference:** check gradient computation (most likely place for bugs)
    
  1. Is \( \sum_{\text{forward}_i(s)} \text{backward}_i(s) \) the same for all \( i \)?
    
  2. Do probabilities normalize correctly + look “reasonable”? (Nearly uniform when untrained, then slowly converging to the right thing)
  
  - **Learning:** is the objective going down? Can you fit a small training set? Are you applying the gradient correctly?
  
  - If objective is going down but model performance is bad:
    
  - **Inference:** check performance if you decode the training set
NER

- CRF with lexical features can get around 85 F1 on this problem
- Other pieces of information that many systems capture
- World knowledge:
  The delegation met the president at the airport, Tanjug said.

Nonlocal Features

- More complex factor graph structures can let you capture this, or just decode sentences in order and use features on previous sentences

Semi-Markov Models

- Chunk-level prediction rather than token-level BIO
- \( y \) is a set of touching spans of the sentence
- Pros: features can look at whole span at once
- Cons: there’s an extra factor of \( n \) in the dynamic programs

Finkel and Manning (2008), Ratinov and Roth (2009)

Evaluating NER

- Prediction of all Os still gets 66% accuracy on this example!
- What we really want to know: how many named entity chunk predictions did we get right?
- Precision: of the ones we predicted, how many are right?
- Recall: of the gold named entities, how many did we find?
- F-measure: harmonic mean of these two

Barack Obama will travel to Hangzhou today for the G20 meeting.

How well do NER systems do?

<table>
<thead>
<tr>
<th>System</th>
<th>Resources Used</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ LBJ-NER</td>
<td>Wikipedia, Nonlocal Features, Word-class Model</td>
<td>90.80</td>
</tr>
<tr>
<td>- (Suzuki and Isozaki, 2008)</td>
<td>Semi-supervised on 1G-word unlabeled data</td>
<td>89.92</td>
</tr>
<tr>
<td>- (Ando and Zhang, 2005)</td>
<td>Semi-supervised on 27M-word unlabeled data</td>
<td>89.31</td>
</tr>
<tr>
<td>- (Kazama and Torisawa, 2007a)</td>
<td>Wikipedia</td>
<td>88.02</td>
</tr>
<tr>
<td>- (Krishnan and Manning, 2006)</td>
<td>Non-local Features</td>
<td>87.24</td>
</tr>
<tr>
<td>- (Kazama and Torisawa, 2007b)</td>
<td>Non-local Features</td>
<td>87.17</td>
</tr>
<tr>
<td>+ (Finkel et al., 2005)</td>
<td>Non-local Features</td>
<td>86.86</td>
</tr>
</tbody>
</table>

Ratinov and Roth (2009)

Lample et al. (2016)

90.94

92.2

Beam Search

Viterbi Time Complexity

- n word sentence, s tags to consider — what is the time complexity?

- $O(n s^2)$ — s is ~40 for POS, n is ~20
Viterbi Time Complexity

Many tags are totally implausible
• Can any of these be:
  • Determiners?
  • Prepositions?
  • Adjectives?

Features quickly eliminate many outcomes from consideration — don’t need to consider these going forward

Beam Search

Maintain a beam of $k$ plausible states at the current timestep

Expand all states, only keep $k$ top hypotheses at new timestep

Maintain priority queue to efficiently add things

Beam size of $k$, time complexity $O(nks \log(ks))$

How good is beam search?

$k=1$: greedy search

Choosing beam size:
• 2 is usually better than 1
• Usually don’t use larger than 50
• Depends on problem structure

If beam search is much faster than computing full sums, can use structured SVM instead of CRFs, but we won’t discuss that here

Next Time

Neural networks