CS388: Natural Language Processing

Lecture 5: CRFs

Greg Durrett
Mini 1 grading underway

Project 1 is out, sample writeups on website
Recall: HMMs

- Observations O (= input x)  
  Output Q (sequence of states) = labels y

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

- Training: maximum likelihood estimation (with smoothing)

- Inference problem: \( \text{argmax}_y P(y | x) = \text{argmax}_y \frac{P(y, x)}{P(x)} \)

- Viterbi: \( \text{score}_i(s) = \max_{y_{i-1}} P(s | y_{i-1}) P(x_i | s) \text{score}_{i-1}(y_{i-1}) \)
Recall: Viterbi Algorithm

- **Initialization**
  \[ v_1(j) = a_{0j} b_j(o_1) \quad 1 \leq j \leq N \]

- **Recursion**
  \[ v_t(j) = \max_{i=1}^{N} v_{t-1}(i)a_{ij}b_j(o_t) \quad 1 \leq j \leq N, \quad 1 < t \leq T \]

- **Termination**
  \[ P^* = v_{T+1}(s_F) = \max_{i=1}^{N} v_T(i)a_{IF} \]

This only calculates the max. To get final answer (argmax),
- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

\[ a_0: \text{Initial state distribution} \]
\[ a_{ij}: \text{Probability of } i-j \text{ transition} \]
\[ b_j(o_t): \text{Probability of emitting symbol } o_t \text{ from state } j \]

slide credit: Ray Mooney
Viterbi/HMMs: Other Resources

- Lecture notes from my undergrad course (posted online)
- Eisenstein Chapter 7.3 **but** the notation covers a more general case than what’s discussed for HMMs
- Jurafsky+Martin 8.4.5
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
  - Insufficient features/capacity with multinomials (especially for unks)
CRFs
Where we’re going

- Flexible discriminative model for tagging tasks that can use arbitrary features of the input. Similar to logistic regression, but *structured*

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

- Curr_word=Barack & Label=B-PER
- Next_word=Obama & Label=B-PER
- Curr_word_starts_with_capital=True & Label=B-PER
- Posn_in_sentence=1st & Label=B-PER
- Label=B-PER & Next-Label = I-PER
...
HMMs, Formally

- HMMs are expressible as Bayes nets (factor graphs)

\[
y_1 \rightarrow y_2 \rightarrow \ldots \rightarrow y_n
\]

\[
x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n
\]

- 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
Conditional Random Fields

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

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

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

normalizer  any real-valued scoring function of its arguments

- Special case: linear feature-based potentials \( \phi_k(x, y) = w^\top f_k(x, y) \)

\[
P(y|x) = \frac{1}{Z} \exp \left( \sum_{k=1}^n w^\top f_k(x, y) \right)
\]

- Looks like our single weight vector multiclass logistic regression model
HMMs vs. CRFs

\[ P(y|x) = \frac{1}{Z} \exp \left( \sum_{k=1}^{n} w^T f_k(x, y) \right) \]

- Conditional model: x’s are observed

- Naive Bayes: logistic regression :: HMMs : CRFs
  local vs. global normalization <-> generative vs. discriminative
  (locally normalized discriminative models do exist (MEMMs))

- HMMs: in the standard setup, emissions consider one word at a time

- CRFs: features over many words simultaneously, non-independent features
  (e.g., suffixes and prefixes), doesn’t have to be a generative model
Problem with CRFs

\[ P(y|x) = \frac{1}{Z} \exp \left( \sum_{k=1}^{n} w^\top f_k(x, y) \right) \]

- Normalizing constant

\[ Z = \sum_{y'} \exp \left( \sum_{k=1}^{n} w^\top f_k(x, y') \right) \]

- Inference: \( y_{\text{best}} = \arg\max_{y'} \exp \left( \sum_{k=1}^{n} w^\top f_k(x, y') \right) \)

- If y consists of 5 variables with 30 values each, how expensive are these?

- Need to constrain the form of our CRFs to make it tractable
Sequential CRFs

Sequential CRF: (one form)

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

- Notation: omit \( x \) from the factor graph entirely (implicit), but every feature function connects to it.
- Two types of factors: \textit{transitions} \( \phi_t \) (look at adjacent \( y \)'s, but not \( x \)) and \textit{emissions} \( \phi_e \) (look at \( y \) and all of \( x \))
Features for NER
Feature Functions

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

- Phis are flexible (can be NN with 1B+ parameters). Here: sparse linear fcns (looks like Mini 1 features)

\[ \phi_e(y_i, i, x) = w^\top f_e(y_i, i, x) \quad \phi_t(y_{i-1}, y_i) = w^\top f_t(y_{i-1}, y_i) \]

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

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

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

Emissions: \( f_e(y_6, 6, x) = \text{Ind}[\text{B-LOC} \& \text{Current word = Hangzhou}] \)
\( \text{Ind}[\text{B-LOC} \& \text{Prev word = to}] \)

Barack Obama will travel to \textbf{Hangzhou} today for the G20 meeting.
Leicestershire is a nice place to visit...

I took a vacation to Boston

Apple released a new version...

According to the New York Times...

Leonardo DiCaprio won an award...

Texas governor Greg Abbott said

\( \phi_e(y_i, i, x) \)
Emission 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

- Leicestershire
- Boston
- Apple released a new version...
- According to the New York Times...
CRFs Outline

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

- **Inference**

- **Learning**

\[
P(y|x) \propto \exp w^\top \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_{i}, i, x) \right]
\]
Inference and Learning in CRFs
Computing (arg)maxes

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

- \( \text{argmax}_y P(y|x) \): can use Viterbi exactly as in HMM case

\[
\max_{y_1, \ldots, y_n} e^{\phi_t(y_{n-1}, y_n)} e^{\phi_e(y_n, n, x)} \ldots e^{\phi_e(y_2, 2, x)} e^{\phi_t(y_1, y_2)} e^{\phi_e(y_1, 1, x)}
\]

\[
= \max_{y_2, \ldots, y_n} e^{\phi_t(y_{n-1}, y_n)} e^{\phi_e(y_n, n, x)} \ldots e^{\phi_e(y_2, 2, x)} \max_{y_1} e^{\phi_t(y_1, y_2)} e^{\phi_e(y_1, 1, x)}
\]

\[
= \max_{y_3, \ldots, y_n} e^{\phi_t(y_{n-1}, y_n)} e^{\phi_e(y_n, n, x)} \ldots \max_{y_2} e^{\phi_t(y_2, y_3)} e^{\phi_e(y_2, 2, x)} \max_{y_1} e^{\phi_t(y_1, y_2)} \text{score}_1(y_1)
\]

- \( \exp(\phi_t(y_{i-1}, y_i)) \) and \( \exp(\phi_e(y_i, i, x)) \) play the role of the Ps now, same dynamic program
Can do efficient 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: \[ P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \]
  \[ P(y|x) \propto \exp w^\top \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right] \]

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

- Learning
Training CRFs

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

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

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

- Gradient is completely analogous to logistic regression:

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

\[ -\mathbb{E}_y \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right] \text{ intractable!} \]
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

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

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

- How do we compute these marginals $P(y_i = s | x)$?

  $$P(y_i = s | x) = \sum_{y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n} P(y | x)$$

- What did Viterbi compute? $P(y_{\text{max}} | x) = \max_{y_1, \ldots, y_n} P(y | x)$

- Can compute marginals with dynamic programming as well using forward-backward
Forward-Backward Algorithm

\[ P(y_3 = 2|x) = \frac{\text{sum of all paths through state 2 at time 3}}{\text{sum of all paths}} \]
Forward-Backward Algorithm

\[ P(y_3 = 2|x) = \frac{\text{sum of all paths through state 2 at time 3}}{\text{sum of all paths}} \]

- Easiest and most flexible to do one pass to compute and one to compute

slide credit: Dan Klein
Forward-Backward Algorithm

- **Initial:**
  \[ \alpha_1(s) = \exp(\phi_e(s, 1, x)) \]

- **Recurrence:**
  \[ \alpha_t(s_t) = \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1}) \exp(\phi_e(s_t, t, x)) \exp(\phi_t(s_{t-1}, s_t)) \]

- Same as Viterbi but summing instead of maxing!

- These quantities get very small!
  Store everything as log probabilities
Forward-Backward Algorithm

- **Initial:**
  \[ \beta_n(s) = 1 \]

- **Recurrence:**
  \[ \beta_t(s_t) = \sum_{s_{t+1}} \beta_{t+1}(s_{t+1}) \exp(\phi_e(s_{t+1}, t + 1, x)) \exp(\phi_t(s_t, s_{t+1})) \]

- **Big differences:** count emission for the next timestep (not current one)
Forward-Backward Algorithm

\( \alpha_1(s) = \exp(\phi_e(s, 1, x)) \)

\( \alpha_t(s_t) = \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1}) \exp(\phi_e(s_t, t, x)) \exp(\phi_t(s_{t-1}, s_t)) \)

\( \beta_n(s) = 1 \)

\( \beta_t(s_t) = \sum_{s_{t+1}} \beta_{t+1}(s_{t+1}) \exp(\phi_e(s_{t+1}, t+1, x)) \exp(\phi_t(s_t, s_{t+1})) \)

\[
P(s_3 = 2 \mid x) = \frac{\alpha_3(2)\beta_3(2)}{\sum_i \alpha_3(i)\beta_3(i)}
\]

- Does this explain why beta is what it is?
- What is the denominator here?  \( P(x) \)
Computing Marginals

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

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

- Analogous to \( P(\mathbf{x}) \) for HMMs

- For both HMMs and CRFs:

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

\[
P(\mathbf{x}) = \text{forward}_0(s) \text{backward}_0(s)
\]

\[
Z \quad \text{for CRFs, P(\mathbf{x})}
\]
**Posteriors vs. Probabilities**

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

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameter</th>
<th>Inferred Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>Model parameter (usually multinominal distribution)</td>
<td>Inferred quantity from forward-backward</td>
</tr>
<tr>
<td>CRF</td>
<td>Undefined (model is by definition conditioned on (x!))</td>
<td>Inferred quantity from forward-backward</td>
</tr>
</tbody>
</table>
Training CRFs

- For emission features:

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

gold features — expected features under model

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

- ...but you can build a pretty good system without learned transition features (use heuristic weights, or just enforce constraints like B-PER -> I-ORG is illegal)
CRFs Outline

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

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

- **Inference:** \(\text{argmax } P(y|\mathbf{x})\) from Viterbi

- **Learning:** run forward-backward to compute posterior probabilities; then

  \[
  \frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{y}^*, \mathbf{x}) = \sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x}) - \sum_{i=1}^{n} \sum_{s} P(y_i = s|\mathbf{x}) f_e(s, i, \mathbf{x})
  \]
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)
    - Is $\sum_i^{s} \text{forward}_i(s)\text{backward}_i(s)$ the same for all $i$?
    - Do probabilities normalize correctly + look “reasonable”? (Nearly uniform when untrained, then slowly converging to the right thing)
- **Learning**: is the objective going down? Try to fit 1 example / 10 examples. 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