EM for Structured Models

Micha Elsner

March 20, 2014
Readings

Main reading:
- Charniak 3.6 (EM for HMMs) and 3.9 (unsupervised POS tagging)

Additional readings:
- Charniak 4.4 (EM for PCFGs)
- Petrov et al: Berkeley parser
Review

Last class:

- Clustering with mixture models
- Model specifies conditionals $P(d|C)$ and prior $P(C)$
- $C$ is one of $k$ (clusters/classes/labels/tags)
- Learning with hard and soft EM
How do you start an EM algorithm?

- Assign items to random clusters
- Assign cluster centers/parameters at random
  - Watch for empty clusters
- Use decisions from a simpler model (rule-based?)
What to watch for

Example: consider a single feature (male/female):

- Max-likelihood solution for $k = 2$: all men in one group, all women in another

- Ties (a “saddle point”): each cluster 50/50 male/female
  - Two cluster centers in the same place
  - Points are perfectly ambiguous: $P(d|C_1) = P(d|C_2)$
  - Nothing ever changes!

- Empty clusters/zero-probability parameters
  - Cluster has no members (so $P(C) = 0$)
  - Cluster has no men (so $P(\text{male}|C) = 0$)
  - Will remain empty forever/never get any men

- Good initialization matters!
  - For complex models, using a simpler model can be very helpful
  - We’ll see an example later
This lecture

- Supervised learning: classification
  - Naive Bayes (k labels)
- Unsupervised learning: clustering
  - Various Naive-Bayes-like mixture models (k labels)
- In supervised learning, we’ve also covered models of many related labels
  - Sequences of tags: $T_{1:n}$ (HMM)
  - Trees
  - Other collections of dependent variables (generic graphical models)
- How do we do unsupervised learning for these models?
Basic EM approach

As always, we iterate between:

- **E-step (inference)**
  - Compute probabilities of tag sequences $T_{1:n}$
  - Either pick the best (hard) or compute responsibilities (soft)
  - Add counts to training set for next iteration

- **M-step (estimation)**
  - Use training counts to reestimate parameters

However, these models have complicated inference procedures

- Which makes our E-step more complicated
Example: the HMM

Inference in the HMM:
- Find max-probability sequence: Viterbi
- Find various marginals: forward-backward (sum-product)

HMM learning with hard EM:
- E-step: compute Viterbi tags $T^*_{1:n}$ for each sentence
- M-step: retrain HMM on Viterbi tag sequences
Soft EM for the HMM

What we need to compute: *expected counts* of important events:

- Which counts?

Recall parameterization of HMM:

- $P_{emit}(W|T)$: probability of words given tags
- $P_{trans}(T_{i+1}|T_i)$: probability of next tag given prev
Emission probabilities

Max-likelihood estimator for $P_{emit}$:

$$
\hat{P}(W_i = w \mid T_i = t) = \frac{\#(t, w)}{\#(t)}
$$

In soft EM, counts replaced by expected counts

- Weighted by conditional probability of labels

$$
\hat{P}(W_i = w \mid T_i = t) = \frac{E_{P(T_i=t)}[\#(t, w)]}{E_{P(T_i=t)}[\#(t)]}
$$

That is, when we observe a word: $W_1 = \text{the}$:

- Calculate the probability that its tag is $t$, $P(T_1 = t \mid W_1:n)$
- Increment expected $\#(t, \text{the})$ by that amount
Calculating expected counts for soft EM:

- Need to calculate \( P(T_i | W_{1:n}) \) for each word
- The probability of the tag for that word, given the sentence

Luckily, these are the tag marginals we calculated in the HMM section:

- At the time, motivated by max-marginal decoding

\[
P(T_i = t | W_{1:n}) = \sum_{T_{-i}} P(W_{1:n}, T_{-i}, T_i = t)
\]

Computed by forward-backward algorithm
Review: computing tag marginals

To get the tag marginal, need to sum over all paths which go through a state
Break this into paths entering state, paths leaving state:

- We can do this because of the Markov property

\[
\begin{align*}
\alpha_t(i) &= P(W_{1:i}, T_i = t) \\
&= P(W_i | T_i = t) \sum_{t'} P(T_i = t | T_{i-1} = t') \alpha_{t'}(i - 1)
\end{align*}
\]
Review: forward-backward

\[
P(T_i = t, W_{1:n})
= \sum_{T_{-i}} P(W_{1:n}, T_{-i}, T_i = t)
= \sum_{T_{1:i-1}} P(W_{1:i}, T_{1:i-1}, T_i = t) \sum_{T_{i+1:n}} P(W_{i+1:n}, T_{i+1:n} | T_i = t)
= \alpha_t(i) \beta_t(i)
\]

Where we define:

\[
\beta_t(i) = \sum_{T_{i+1:n}} P(W_{i+1:n}, T_{i+1:n} | T_i = t)
\]
\( \alpha \) computes prob of this stuff

\( \beta \): prob of this stuff

1

2

\( <s> \quad a \quad b \quad b \quad b \quad b \quad a \)
Max-likelihood estimator for $P_{\text{trans}}$:

$$\hat{P}(T_{i+1} = t \mid T_i = t') = \frac{\#(t', t)}{\#(t')}$$

We need to replace these with expectations

- Which means we need to compute
  $$P(T_{i+1} = t, T_i = t' \mid W_{1:n})$$
Computing this

We need to sum over incoming paths at $T_i$:

$\alpha_{t'}(i)$

And over outgoing paths at $T_{i+1}$:

$\beta_t(i + 1)$

And we need to connect things up by actually going from $t'$ to $t$:

$P_{\text{trans}}(t|t') P_{\text{emit}}(w_{i+1}|t)$

\[
P(T_{i+1} = t, T_i = t' | W_{1:n}) = \frac{\alpha_{t'}(i) P_{\text{trans}}(t|t') P_{\text{emit}}(w_{i+1}|t) \beta_t(i + 1)}{P(W_{1:n})}
\]
Soft EM for HMMs

- **E-step**
  - Run forward-backward on each sentence
  - Use $\alpha$ and $\beta$ to find expected counts for each word/tag and tag/tag
  - Add expected counts to training set

- **M-step**
  - Compute estimates of $P_{emit}$ and $P_{trans}$ from training counts
Same idea for grammars?

EM for PCFGs is basically the same:

- Obtain marginals over trees using inside/outside probabilities
- Add expected counts to training set
- Count and divide (possibly with smoothing)

See Charniak 4.4.2-4 for details
Bottom-up parser computes $\mu$ probabilities:

$$\mu_A(i, l) = \max_{\text{tree } T} P(W_{i:i+l}, T, \text{root of } T = A)$$

The probability of spanning the $l$ words starting at $i$ with nonterminal $A$

Recursive definition in terms of left and right children

$$\mu_A(i, l) = \max_{0 < j < l, A \to BC} P(A \to BC) \mu_B(i, j) \mu_C(i + j, l - j)$$
Calculating the marginal probability of the sentence

As with the HMM, we can calculate a sum instead of a max:

$$\beta_A(i, l) = \sum_{\text{tree } T} P(W_{i:i+l}, T, \text{root of } T = A)$$

This is the *inside probability*, since it tells us the probability that the string $W_{i:i+l}$ appears inside the span of nonterminal $A$ (marginalizing over all the substructure under $A$).

$$\beta_A(i, l) = \sum_{0<j<l, A \rightarrow BC} P(A \rightarrow BC) \beta_B(i, j) \beta_C(i + j, l - j)$$
Outside probabilities

Marginals for specific nonterminals require both an inside and an outside probability...

- Like forward and backward in HMM

The outside probability is what eliminates analyses like: (S (NP the cat) (VP sat)) on the mat

- Remember that our chart parser put an S on the chart for this span
- But it won’t be part of the full parse
- The outside probability describes how likely we are to fit the constituent into an analysis of the rest of the sentence
Outside probability (definition)

\[ \alpha_A(i, l) = \sum_{\text{tree outside } i:l \ T} P(W_{0:i-1,i+l:n}, T | \text{nonterminal from } i : i+l = A) \]

The definition is still recursive (not surprisingly)...

- To build the tree outside \( i : i + l \), we need three parts
  - The grammar rule tying \( A \) to its parent in the tree, \( B \rightarrow AC \) or \( B \rightarrow CA \)
  - (It can be a left child or a right child of its parent)
  - The probability of all the stuff under its sibling \( C \)
  - The probability of all the stuff outside the parent
Illustration

```
S
  │
  term-3
    │
    term-3
      │
      stuff
            │
            A
              │
              W_i ...
              W_{i+1}
            │
            C
              │
              term
                2
```
\[ \alpha_A(i, l) = \sum_{k > l} \alpha_B(i, k) \sum_{B \to AC} P(B \to AC) \beta_C(l, k) + \sum_{h < i} \alpha_B(h, l) \sum_{B \to CA} P(B \to CA) \beta_C(h, i) \]
EM for generic graphical models

We can use soft EM whenever we can get marginals:
  - Using message passing (dynamic program)

For some models, we can find max but not marginals
  - Or finding the max is much faster
  - ILP-based inference finds maxima without finding marginals
    - Or doing any dynamic programming

In these cases:
  - Hard EM still possible (only requires max)
  - Soft EM not usually possible
    - (Sometimes “stochastic” or “Monte Carlo” variants using samples)
Latent-state grammars

Petrov, Barret, Thibaux and Klein “Learning accurate, compact and interpretable tree annotation” 2006
Core of the Berkeley parser
Motivation

In parsing section, spent lots of time annotating parse tree nodes:

- Good decisions about what to annotate: parser improves
- Bad decisions: data sparsity makes parser worse
Idea

Use EM algorithm to learn annotations automatically:

For each nonterminal, define $k$ subclasses, NP-C1, NP-C2, NP-C3 ... NP-C$k$
What the grammar looks like

Old grammar:

\[ S \rightarrow NP \ VP \]

New \( k = 2 \):

\[
\begin{align*}
S-C1 & \rightarrow NP-C1 \ VP-C1 & P(NP_{C1} VP_{C1} | S_{C1}) \\
S-C1 & \rightarrow NP-C2 \ VP-C1 & P(NP_{C2} VP_{C1} | S_{C1}) \\
S-C1 & \rightarrow NP-C1 \ VP-C2 & P(NP_{C1} VP_{C2} | S_{C1}) \\
S-C1 & \rightarrow NP-C2 \ VP-C2 & P(NP_{C2} VP_{C2} | S_{C1}) \\
S-C2 & \rightarrow NP-C1 \ VP-C1 & P(NP_{C1} VP_{C1} | S_{C2}) \\
S-C2 & \rightarrow NP-C2 \ VP-C1 & P(NP_{C2} VP_{C1} | S_{C2}) \\
S-C2 & \rightarrow NP-C1 \ VP-C2 & P(NP_{C1} VP_{C2} | S_{C2}) \\
S-C2 & \rightarrow NP-C2 \ VP-C2 & P(NP_{C2} VP_{C2} | S_{C2})
\end{align*}
\]
What do the symbols mean?

How is an NN-C1 different from an NN-C2?
- As in any unsupervised learning system, the symbols have no inherent meaning
- But acquire meaning in context
- We can look at the grammar and try to find out
- For preterminals (POS tags) this works pretty well...

Figure 2: Evolution of the DT tag during hierarchical splitting and merging. Shown are the top three words for each subcategory and their respective probability.
Phrasal symbol meanings

Situation not as clear for phrasal nonterminals
(Some analysis from paper)

- Embedded versus main-clause sentence types
- Matrix clause (S-12) goes to...
- NP-8 (usually subj NP) goes to...
- PRP-0 (nominative pronoun), DT-1,2,6 (capital determiner)
  etc

Main clause VPs more likely to produce complement-taking verbs “said”, “reported”
- Subclause VP more likely to produce “did”, “began”
Designing an EM algorithm

Using the sum-product version of CKY:

- In this case, the tree structure and basic labels (NP, VP) are known
- Uncertainty only about the sub-labels
- Much faster than normal CKY

```
the  cat  sat  on  the  mat
```

```
DT  NN  VB  IN  D  T  NN
NP  NP
PP
VP
S
```
EM algorithm

- E-step
  - Run constrained inside-outside (sum-product) on each sentence
  - Use $\alpha$ and $\beta$ to find expected counts for each rule
  - Add expected counts to training set

- M-step
  - Compute estimates of $P(A_{CX} \rightarrow B_{CY} C_{CZ})$ from training counts
EM can have problems with local maxima

- The final Berkeley grammar has up to 64 subsymbols per nonterminal
- Starting EM randomly with 64 subsymbols, you get stuck
  - Quickly reach poor local maximum
- Not all nonterminals need 64 tags
  - Comma, TO, CC are all small classes to begin with
- Instead, use a step-by-step process to initialize each round of EM
  - And determine which symbols need more refinement

Procedure known as “split-merge”
Splitting

Two-phase procedure:

- Splitting makes more symbols
- Merging makes useless symbols go away

Start with basic grammar (no subsymbols)

Split phase

- Split each symbol into two: NP into NP-C1, NP-C2
- Rule probabilities copied over from old grammar
  - But add a small random number to each to break ties
- Now run EM!
  - With only 2 subsymbols per nonterm, less chance of getting stuck
Merging

Merge phase

- Symbols like TO don’t benefit from being split
- Use likelihood to find out whether our split helped
  - Compare likelihood using split grammar...
  - And likelihood if we re-merge each split: TO-C1, TO-C2 with TO
    - We don’t have to run EM to relearn this grammar, just add up the counts
- If likelihood doesn’t decrease too much, accept the merge
Repeated splitting and merging

- Do this over and over
- Each split doubles the number of tags
- ex. NP into NP-C1, NP-C2
- And then: NP-C11, NP-C12, NP-C21, NP-C22
- 6 cycles yields up to $2^6 = 64$ tags per nonterm
  - Which took a couple of days back when the paper was published

<table>
<thead>
<tr>
<th>tag</th>
<th>states learned</th>
<th>nt</th>
<th>states learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP</td>
<td>62</td>
<td>NP</td>
<td>37</td>
</tr>
<tr>
<td>JJ</td>
<td>58</td>
<td>VP</td>
<td>32</td>
</tr>
<tr>
<td>NN</td>
<td>56</td>
<td>PP</td>
<td>28</td>
</tr>
<tr>
<td>VBN</td>
<td>49</td>
<td>S</td>
<td>21</td>
</tr>
<tr>
<td>VBG</td>
<td>40</td>
<td>SBAR</td>
<td>15</td>
</tr>
<tr>
<td>IN</td>
<td>27</td>
<td>SINV</td>
<td>3</td>
</tr>
<tr>
<td>DT</td>
<td>17</td>
<td>SQ</td>
<td>2</td>
</tr>
<tr>
<td>TO</td>
<td>1</td>
<td>LST</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion

- Hard and soft EM allow learning for structured models
  - Which model many related variables
- Including HMMs, PCFGs, general graphical models
- Hard EM alternates inference (max) with estimation (training)
  - Always possible
- Soft EM alternates computing marginals (sum-product) with estimation
  - Use message passing dynamic program to find marginals
- Berkeley parser: learns a specialized grammar
- Initialization is important!
  - Use splitting and merging to iteratively expand the tag set