

Conditional Random Fields

The problem with HMMs is the same as with naive Bayes: the independence assumptions are violated and this can lead to bad performance. This seems to be a particularly strong problem in sequence applications.

So, what do we do? We switch to a conditional model.

Remember, before, we had:

$$a_{p,l} = \text{probability of transitioning from } p \text{ to } l$$

$$b_{l,v} = \text{probability of emitting word } v \text{ from } l$$

Maximum Entropy Markov Models

We're going to flip this entirely and just model the probability of transitioning to state y_m given the input x and the previous state y_{m-1} . We'll make this an exponential:

$$p(y_m = l \mid x, y_{m-1} = p) = \frac{1}{\sum_{l'} \exp[\theta^\top \Phi(x, m, l', p)]} \exp[\theta^\top \Phi(x, m, l, p)]$$

Now, Φ can be *any* function of the input at *any* position.

Two questions: how to train and how to predict (aka "decode")?

Training: just like training a standard multiclass maxent/LR classifier.

Prediction: just like Viterbi for HMMs, but instead of using $a_{l,p}b_{l,v}$, we use $p(y_m \mid x, y_{m-1}; \theta)$.

Much more flexible model, but somewhat not ideal. Why? Label-bias problem. Essentially, the per-state normalization kills us.

Conditional Random Fields

(We begin with CRFs as Markov models; later, we'll extend)

The problem with MEMMs was the per-state normalization. CRFs remove the per-state normalization and introduce a per-sequence normalization. The difference is as follows:

$$\text{MEMM : } p(y_{1:M} \mid x; \theta) = \prod_m \frac{1}{Z_{x,m,y_{m-1},\theta}} \exp[\theta^\top \Phi(x, m, y_m, y_{m-1})]$$

$$\text{CRF : } p(y_{1:M} \mid x; \theta) = \frac{1}{Z_{x,\theta}} \prod_m \exp[\theta^\top \Phi(x, m, y_m, y_{m-1})]$$

Given θ , prediction is again Viterbi. The question is how to train. The problem is the normalization:

$$Z_{x,\theta} = \sum_{y'_{1:M}} \prod_m \exp [\theta^\top \Phi(x, m, y'_m, y'_{m-1})]$$

This requires summing over the entire lattice! Luckily, we can do this as efficiently as computing the optimal path.

Let's look at the gradient of the log-likelihood with respect to the parameters:

$$\ell(y_{1:M} | x; \theta) = \sum_m (\theta^\top \Phi(x, m, y_m, y_{m-1}) - \log Z_{x,\theta})$$

We can easily take the gradient for the first part, but cannot (apparently) easily do it for the log Z term. We take the gradient as:

$$\begin{aligned} \nabla_\theta \ell &= \sum_m (\Phi(x, m, y_m, y_{m-1}) - \nabla_\theta \log Z_{x,\theta}) \\ \nabla_\theta \log Z_{x,\theta} &= \frac{1}{Z_{x,\theta}} \nabla_\theta Z_{x,\theta} \\ &= \frac{1}{Z_{x,\theta}} \sum_{y'_{1:M}} \nabla_\theta \prod_m \exp [\theta^\top \Phi(x, m, y'_m, y'_{m-1})] \\ &= \sum_m \sum_{l,p} p(y_m = l, y_{m-1} = p | x, m; \theta) \Phi(x, m, l, p) \end{aligned}$$

So the remaining question is how to compute these “hypothesized” probabilities. We can do this with a straightforward modification of the “forward-backward” algorithm for HMMs.

This works by defining “forward probabilities, α ” and “backward probabilities, β ”. These are defined as:

$$\begin{aligned} \alpha_{m,l} &= p(y_{1:m-1}, y_m = l | x; \theta) \\ \beta_{m,l} &= p(y_m = l, y_{m+1:M} | x; \theta) \end{aligned}$$

That is, $\alpha_{m,l}$ is the probability of traversing the input up to element m and producing a label of l for y_m . Similarly, $\beta_{m,l}$ is the probability of labeling element y_m with label l and then labeling the rest of the sequence. You can think of the α s as computing probabilities forward through the lattice, and the β s as computing probabilities backward through the lattice.

We can compute both α and β recursively by:

$$\begin{aligned} \alpha_{m+1,l} &= \sum_p \alpha_{m,p} \exp [\theta^\top \Phi(x, m, l, p)] \\ \beta_{m,p} &= \sum_l \beta_{m+1,l} \exp [\theta^\top \Phi(x, m, l, p)] \end{aligned}$$

Now, using the α and β matrices, we can compute what we need: $p(y_m = l, y_{m-1} = p | x, m; \theta)$. We can do this by observing that this is just the probability of traversing *forward* to assign label p to position

y_{m-1} , then making an assignment of label l to y_m , and then traversing to the end (or, equivalently, traversing backward to position m) with label l . Thus:

$$p(y_m = l, y_{m-1} = p \mid x, m; \theta) \propto \alpha_{m-1,p} \exp[\theta^\top \Phi(x, m, l, p)] \beta_{m,l}$$

Phew!

So, putting this all together, we obtain the following algorithm for computing the *gradient* of the likelihood of the CRF:

1. For each data point $n = 1 \dots N$,
 - (a) Compute the forward lattice α for x_n
 - (b) Compute the backward lattice β for x_n
 - (c) For each position m and label pair (p, l) compute the label probability $p(y_m = l, y_{m-1} = p \mid x, m; \theta)$
 - (d) Compute the gradient for example n as:

$$\nabla_{\theta} \ell = \sum_m \left(\Phi(x, m, y_m, y_{m-1}) - \sum_{l,p} p(y_m = l, y_{m-1} = p \mid x, m; \theta) \Phi(x, m, l, p) \right)$$

- (e) Add this to an accumulating gradient

Given this gradient, there are hundreds (okay, tens) of ways to optimize. Simple gradient descent converges slowly, so more complicated procedures like limited-memory BFGS and stochastic meta-descent tend to be preferred.

CRFs in the general case

CRFs (unlike MEMMs) are more general beasts than just for sequence labeling. The idea is to represent what we want to predict as a graph, where nodes are labels and edges represent dependencies. A linear chain (aka sequence) model is the simplest (non-trivial) version of this.

But if you look at the algorithm above, there's really nothing that assumes a chain. The only issue is whether it is possible to compute the "forward" and "backward" lattices for general graphs.

The answer is "yes, but not tractably." In general, we measure the complexity of the graph by its treewidth (we'll talk about this in more detail after spring break). Linear chains have low treewidth; fully connected graphs have high treewidth. The complexity of computing the required probabilities scales *exponentially* in the treewidth. In fact, the forward-backward algorithm we just discussed is a specific example of inference in general graphs—in this case, it just happens to be tractable.