Conditional Random Fields

Conditional Random Fields

- The **Conditional Random Field** is a conditional probabilistic model for sequence labeling based on logistic regression
- The name is derived from **Markov Random Fields**, a class of models in which the probability of a configuration of variables is proportional to a product of scores across pairs of variables in a factor graph.

The probability model

- The basic probability model is

$$P(\boldsymbol{y}|\boldsymbol{w}) = \frac{\exp(\Psi(\boldsymbol{w}, \boldsymbol{y}))}{\sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{w})} \exp(\Psi(\boldsymbol{w}, \boldsymbol{y'}))}$$

Assignment Project Exam Help

- This is almost identical to Logistic Regression, except that the label space is sequence of tags

https://powcoder.com

- This requires efficient algorithms to *find the best tag sequence* in decoding, and for *summing over all tag sequences* in training

Add WeChat powcoder

- The usual locality assumption on the scoring function:

$$\Psi(\boldsymbol{w}, \boldsymbol{y}) = \sum_{m=1}^{M+1} \psi(\boldsymbol{w}, y_m, y_{m-1}, m)$$

Decoding in CRFs

- The Viterbi algorithm can be used to find the best tag sequence, just as it can be used for decoding HMM and Perpcetron models
- The decoding algorithm is identical to that of perceptron, because the denominator is the same for all tag sequences and can thus be ignored:

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \operatorname*{argmax}_{\boldsymbol{y}} \log P(\boldsymbol{y}|\boldsymbol{w}) \\
&= \operatorname*{argmax}_{\boldsymbol{y}} \Psi(\boldsymbol{y}, \boldsymbol{w}) - \log \sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{w})} \exp(\Psi(\boldsymbol{y'}, \boldsymbol{w})) \\
&= \operatorname*{argmax}_{\boldsymbol{y}} \Psi(\boldsymbol{y}, \boldsymbol{w}) \\
&= \operatorname*{argmax}_{\boldsymbol{y}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1})
\end{aligned}
$$

Learning in CRFs

- As with logistic regression, the weights can be learned by minimizing the regularized negative log-probability loss:

$$
\begin{aligned}
\mathcal{L} &= \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 - \sum_{i=1}^{N} \log P(\boldsymbol{y}^{(i)} | \boldsymbol{w}^{(i)}; \boldsymbol{\theta}) \\
&= \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 - \sum_{i=1}^{N} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{w}^{(i)}, \boldsymbol{y}^{(i)}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{w}^{(i)})} \exp\left(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{w}^{(i)}, \boldsymbol{y}')\right)
\end{aligned}
$$

- The second term requires computing *the sum of all possible labelings*. There are $|\mathcal{Y}|^M$ possible labeling for an input of length M, so an efficient algorithm is required to compute this sum.

Computing the gradients of the loss function

- As in logistic regression, the gradient of the negative log likelihood with respect to the parameters is the difference between observed and expected feature counts:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \lambda \theta_j + \sum_{i=1}^{N} \mathbb{E}[f_j(\boldsymbol{w}^{(i)}, \boldsymbol{y}] - f_j(\boldsymbol{w}^{(i)}, \boldsymbol{y}^{(i)})$$

- Computing this gradient involves computing the posterior of a sequence that includes the transition $Y_{m-1} \rightarrow Y_m$
- Recall the feature function for bigram tag sequences is of the form $f(Y_{m-1}, Y_m, \boldsymbol{w}, m)$. To compute the expected count of a feature, we need $P(Y_{m-1} = k', Y_m = k | \boldsymbol{w})$.

Marginal probabilities over tag bigrams

$$Pr(Y_{m-1} = k', Y_m = k \mid \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^{M+1} \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y'}} \prod_{n=1}^{M+1} \exp s_n(y'_n, y'_{n-1})}$$

How do we compute the numerator and denominator?

- The naive way to compute this probability is to enumerate all possible labelings for the sequence. Since there are $\mathcal{Y}^M$ possible labelings, this is prohibitively expensive for a typical tag set and sentence length.
- So we need find a more efficient way of doing this.

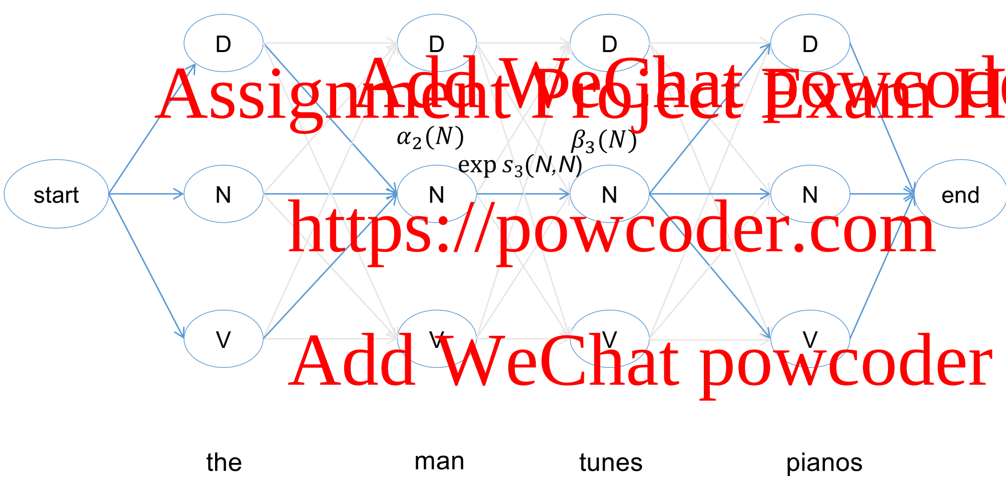
Computing the numerator

The numerator sums over all tag sequences that includes the transition $(Y_{m-1} = k') \to (Y_m = k)$. This sum can be decomposed into three parts: the prefix $\boldsymbol{y}_{1:m-1}$, terminating in $Y_{m-1} = k'$, the transition $(Y_{m-1} = k') \to (Y_m = k)$, and the suffix $\boldsymbol{y}_{m:M}$, beginning with the tag $Y_m = k$

$$\sum_{\boldsymbol{y}:Y_m=k,Y_{m-1}=k'} \prod_{n=1}^{M+1} \exp s_n(y_n, y_{n-1}) = \sum_{\boldsymbol{y}_{1:m-1}:Y_{m-1}=k'} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1})$$
$$\times \exp s_m(k, k')$$
$$\times \sum_{\boldsymbol{y}_{m:M}:Y_m=k} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1})$$

Trellis

We can illustrate the numerator graphically with a trellis:



We compute the numerator by positing the first term as a **forward variable** $\alpha_{m-1}(k')$, and the third term as a **backward variable** $\beta_m(k)$.

Defining a forward variable that can be cached

- A **forward variables** $\alpha_m(y_m)$ is equal to the sum of scores of all paths leading to the tag y_m at position m:

$$
\begin{aligned}
\alpha_m(y_m) &\triangleq \sum_{\boldsymbol{y}_{1:m-1}} \exp \sum_{n-1}^{m} s_n(y_n, y_{n-1}) \\
&= \sum_{\boldsymbol{y}_{1:m-1}} \prod_{n=1}^{m} \exp s_n(y_n, y_{n-1})
\end{aligned}
$$

- The forward recurrence is also known as the **sum-product algorithm** and can be computed through a recurrence

The forward recurrence

- Computing the forward variable at position m

$$\begin{aligned}
\alpha_m(y_m) &= \sum_{\boldsymbol{y}_{1:m-1}} \prod_{n=1}^{m} \exp s_n(y_n, y_{n-1}) \\
&= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \sum_{\boldsymbol{y}_{1:m-2}} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \\
&= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \times \alpha_{m-1}(y_{m-1})
\end{aligned}$$

The backward recurrence

$$
\begin{aligned}
\beta_m(k) &\triangleq \sum_{\boldsymbol{y}_{m:M}:Y_m=k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \\
&= \sum_{k' \in \mathcal{Y}} \exp s_m(k', k) \sum_{\boldsymbol{y}_{m+1:M}:Y_{m+1}=k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \\
&= \sum_{k' \in \mathcal{Y}} \exp s_m(k', k) \times \beta_{m+1}(k')
\end{aligned}
$$

where k' is the label at position $m+1$.

Computing the denominator

The denominator, the score of all possible labelings for the entire sequence, can be computed either via the forward recurrence or backward recurrence, or at any given position m:

- The score of all possible labelings for the entire sequence is the value of the forward variable at position $M+1$ in state $\blacklozenge$:

$$\sum_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{w})} \Psi(\boldsymbol{w},\boldsymbol{y}) = \alpha_{M+1}(\blacklozenge)$$

$$= \sum_{\boldsymbol{y}_{1:M}} (\exp s_{M+1}(\blacklozenge, y_M)) \prod_{m=1}^{M} \exp s_m(y_m, y_{m-1})$$

- It can also be computed via backward recurrence as the value of the backward variable at position 0:

$$\sum_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{w})} \Psi(\boldsymbol{w},\boldsymbol{y}) = \beta_0(\lozenge)$$

Features and their weights

f_k	y_{m-1}	y_m	w_{-2}	w_{-1}	w_0	w_1	w_2	θ_k
f_1	D	D	-	-	-	-	-	-0.5
f_2	N	D	-	-	-	-	-	-0.8
f_3	V	D	-	-	-	-	-	1.2
f_4	D	N	-	-	-	-	-	2.5
f_5	N	N	-	-	-	-	-	0.5
f_6	V	N	-	-	-	-	-	3
f_7	D	V	-	-	-	-	-	0.4
f_8	N	V	-	-	-	-	-	4
f_9	V	V	-	-	-	-	-	0.6
f_{10}	-	D	-	-	man	-	-	-0.5
f_{11}	-	N	-	-	man	-	-	2
f_{12}	-	V	-	-	man	-	-	1
f_{13}	-	D	-	the	-	-	-	-4
f_{14}	-	N	-	the	-	-	-	5
f_{15}	-	V	-	the	-	-	-	-2

Computing local transition matricies

Recall the local score between two positions in a tag sequence is $\exp s_m(y_m, y_{m-1})$. The transition scores from each tag y_{m-1} to the tag y_m can be arranged in a $K \times K$ matrix, where K is the number of tags. Suppose we are computing the score at the position "man" based on the feature weights in the previous slide:

$$\exp(f_1\theta_1 + f_{10}\theta_{10} + f_{13}\theta_{13}) = \exp(-0.5 - 0.5 - 4) = 0.007$$

$$\begin{array}{c c} & \begin{array}{ccc} D & N & V \end{array} \\ \begin{array}{c} D \\ N \\ V \end{array} & \begin{bmatrix} 0.007 & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \end{array}$$

Forward computation in CRF

$M_1(k',k)$

	D	N	V
D	1.2	0	0
N	0	2	0
V	0	0	0.5

$M_2(k',k)$

	D	N	V
D	0.2	4	0.1
N	0.3	0.8	2.5
V	2	5	1

$M_3(k',k)$

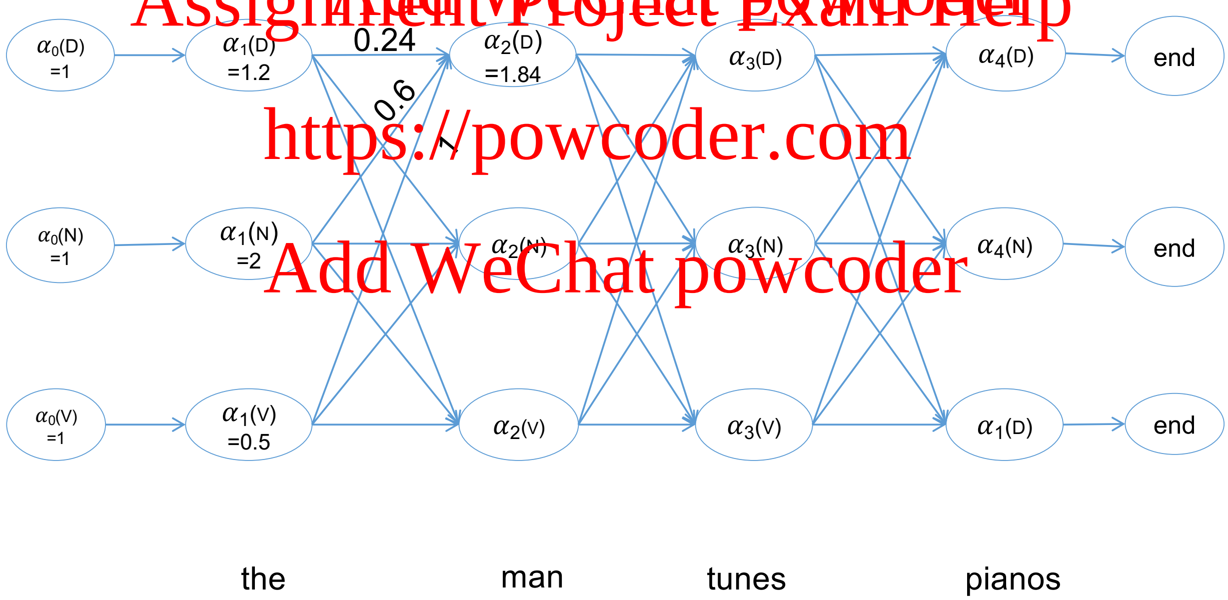
	D	N	V
D	0.3	4.5	0.8
N	0.4	1.2	3.5
V	3	6	3

$M_4(k',k)$

	D	N	V
D	0.5	5	0.4
N	0.6	1.5	6
V	3.5	8	2.5

$M_5(k',k)$

	D	N	V
D	1.2	1.2	1.2
N	0.3	0.3	0.3
V	4	4	4



Representing features as matricies

Some features span multiple state (tag) transitions:

$$f_1(k', k, \boldsymbol{w}, m) = 1 \text{ iff } w_m = run \;\&\; k = V$$

$$f_2(k', k, \boldsymbol{w}, m) = 1 \text{ iff } w_m = run \;\&\; k = N$$

If represented as matricies, these features have multiple cells that have a value of 1:

$$f_1(k', k, \boldsymbol{w}, m) = \begin{array}{c c} & \begin{array}{ccc} D & N & V \end{array} \\ \begin{array}{c} D \\ N \\ V \end{array} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$f_2(k', k, \boldsymbol{w}, m) = \begin{array}{c c} & \begin{array}{ccc} D & N & V \end{array} \\ \begin{array}{c} D \\ N \\ V \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

Feature expectations

- Let f be any local feature $f_m[k', k] = f(k', k, \boldsymbol{w}, m)$. The count of the this feature in a particular sequence is

$$F(\boldsymbol{y}, \boldsymbol{w}) = \sum_{m}^{M+1} f(k', k, \boldsymbol{w}, m)$$

- Expectation for a single feature *over* a sequence is:

$$\mathbb{E}[F(\boldsymbol{w}, \boldsymbol{y})] = \sum_m Pr(Y_{m-1} = k', Y_m = k | \boldsymbol{w}) f(k', k, \boldsymbol{w}, m)$$

$$= \sum_m \frac{\alpha_{m-1}(f \odot M_m)\beta_m^\top}{Z(\boldsymbol{w})}$$

  where $Z(\boldsymbol{w})$ is the total score of all labelings, also known as the **partition function**.

- Given the feature expectations, we can now compute the gradient of each feature.

Example computation of feature expectations

- Assume the transition matrix at position m:

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

$$
\begin{array}{cc}
M[k',k] & \begin{array}{ccc} D & N & V \end{array} \\
\begin{array}{c} D \\ N \\ V \end{array} & \begin{bmatrix} 2.2 & 0.1 & 0.4 \\ 1.2 & 3.4 & 0.3 \\ 6.1 & 7.2 & 0.01 \end{bmatrix}
\end{array}
$$

- Assume the following vectors:
  - $\alpha_{m-1}(k') = \begin{bmatrix} 40 & 30 & 65 \end{bmatrix}$
  - $\beta_m(k) = \begin{bmatrix} 45 & 65 & 30 \end{bmatrix}$

Computing Feature Expectations

$$\alpha_{m-1}(f_1 \odot M_m)\beta_m^\top = \begin{bmatrix}40 & 30 & 65\end{bmatrix} \begin{bmatrix}0 & 0.1 & 0 \\ 0 & 3.4 & 0 \\ 0 & 7.2 & 0\end{bmatrix} \begin{bmatrix}45 \\ 65 \\ 30\end{bmatrix}$$

$$= \begin{bmatrix}0 & 574 & 0.0\end{bmatrix} \begin{bmatrix}45 \\ 65 \\ 30\end{bmatrix} = 37310.0$$

$$\alpha_{m-1}(f_1 \odot M_m)\beta_m^\top = \begin{bmatrix}40 & 30 & 65\end{bmatrix} \begin{bmatrix}0 & 0 & 0.4 \\ 0 & 0 & 0.3 \\ 0 & 0 & 0.01\end{bmatrix} \begin{bmatrix}45 \\ 65 \\ 30\end{bmatrix}$$

$$= ?$$

CRFs

- Avoids the strong independence assumptions of HMMs
- Allows arbitrary features over observations (but not hidden states)
- Sound probabilistic model - gives us a distribution over sequence labelings
- Uses the same efficient algorithm (the Viterbi Algorithm) for decoding as structured perceptron and HMMs
- Estimation/learning is harder (than say Perceptron) because we have to compute the posterior for a sequence
- Empirical results generally show CRFs outperforms HMMs (and other classifiers)
- Feature estimation can be replaced with LSTM RNNs, resulting in what's called RNN-CRFs, of which LSTM-CRFs are the most widely used.