CS838-1 Advanced NLP: Conditional Random Fields

Xiaojin Zhu

2007
Send comments to jerryzhu@cs.wisc.edu

1 Information Extraction

Current NLP techniques cannot fully understand general natural language articles. However, they can still be useful on restricted tasks. One example is Information Extraction. For example, one might want to extract the title, authors, year, and conference names from a researcher's Web page. Or one might want to identify person, location, organization names from news articles (NER, named entity recognition). These are useful to automatically turn free text on the Web into knowledge databases, and form the basis of many Web services.

The basic Information Extraction technique is to treat the problem as a text sequence tagging problem. The tag sets can be {title, author, year, conference, other}, or {person, location, organization, other}, for instance. Therefore HMMs has been naturally and successfully applied to Information Extraction.

However, HMMs have difficulty modeling *overlapping, non-independent* features of the output. For example, an HMM might specify which words are likely for a given state (tag) via $p(x|z)$. But often the part-of-speech of the word, as well as that of the surrounding words, character n-grams, capitalization patterns all carry important information. HMMs cannot easily model these, because the generative story limits what can be generated by a state variable.

Conditional Random Field (CRF) can model these overlapping, non-independent features. A special case, linear chain CRF, can be thought of as the *undirected graphical model* version of HMM. It is as efficient as HMMs, where the sum-product algorithm and max-product algorithm still apply.

2 The CRF Model

Let $x_{1:N}$ be the observations (e.g., words in a document), and $z_{1:N}$ the hidden labels (e.g., tags). A linear chain Conditional Random Field defines a *conditional* *probability* (whereas HMM defines the joint)

$$p(z_{1:N}|x_{1:N}) = \frac{1}{Z}\exp\left(\sum_{n=1}^{N}\sum_{i=1}^{F}\lambda_i f_i(z_{n-1}, z_n, x_{1:N}, n)\right). \tag{1}$$

Let us walk through the model in detail. The scalar Z is the normalization factor, or *partition function*, to make it a valid probability. Z is defined as the sum of exponential number of sequences,

$$Z = \sum_{z_{1:N}}\exp\left(\sum_{n=1}^{N}\sum_{i=1}^{F}\lambda_i f_i(z_{n-1}, z_n, x_{1:N}, n)\right), \tag{2}$$

therefore is difficult to compute in general. Note Z implicitly depends on $x_{1:N}$ and the parameters λ.

The big exp() function is there for historical reasons, with connection to the exponential family distribution. For now, it is sufficient to note that λ and $f()$ can take arbitrary real values, and the whole exp function will be non-negative.

Within the exp() function, we sum over $n = 1, \ldots, N$ word positions in the sequence. For each position, we sum over $i = 1, \ldots, F$ *weighted features*. The scalar λ_i is the weight for feature $f_i()$. The λ_i's are the *parameters* of the CRF model, and must be learned, similar to $\theta = \{\pi, \phi, A\}$ in HMMs.

3 Feature Functions

The feature functions are the key components of CRF. In our special case of linear-chain CRF, the general form of a feature function is $f_i(z_{n-1}, z_n, x_{1:N}, n)$, which looks at a pair of adjacent states z_{n-1}, z_n, the *whole* input sequence $x_{1:N}$, and where we are in the sequence (n). These are arbitrary functions that produce a real value.

For example, we can define a simple feature function which produces binary values: it is 1 if the current word is John, and if the current state z_n is PERSON:

$$f_1(z_{n-1}, z_n, x_{1:N}, n) = \begin{cases} 1 & \text{if } z_n = \text{PERSON and } x_n = \text{John} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

How is this feature used? It depends on its corresponding weight λ_1. If $\lambda_1 > 0$, whenever f_1 is active (i.e. we see the word John in the sentence and we assign it tag PERSON), it increases the probability of the tag sequence $z_{1:N}$. This is another way of saying the CRF model should prefer the tag PERSON for the word John. If on the other hand $\lambda_1 < 0$, the CRF model will try to avoid the tag PERSON for John. Which way is correct? One may set λ_1 by domain knowledge (we know it should probably be positive), or learn λ_1 from corpus (let the data tell us), or both (treating domain knowledge as prior on λ_1). Note $\lambda_1, f_1()$ together is equivalent to (the log of) HMM's ϕ parameter $p(x = \text{John}|z = \text{PERSON})$.

As another example, consider

$$f_2(z_{n-1}, z_n, x_{1:N}, n) = \begin{cases} 1 & \text{if } z_n = \text{PERSON and } x_{n+1} = \text{said} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

This feature is active if the current tag is PERSON and the *next* word is 'said'. One would therefore expect a positive λ_2 to go with the feature. Furthermore, note f_1 and f_2 can be both active for a sentence like "John said so." and $z_1 = \text{PERSON}$. This is an example of *overlapping features*. It boosts up the belief of $z_1 = \text{PERSON}$ to $\lambda_1 + \lambda_2$. This is something HMMs cannot do: HMMs cannot look at the next word, nor can they use overlapping features.

The next feature example is rather like the transition matrix A in HMMs. We can define

$$f_3(z_{n-1}, z_n, x_{1:N}, n) = \begin{cases} 1 & \text{if } z_{n-1} = \text{OTHER and } z_n = \text{PERSON} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

This feature is active if we see the particular tag transition (OTHER, PERSON). Note it is the value of λ_3 that actually specifies the equivalent of (log) transition probability from OTHER to PERSON, or $A_{\text{OTHER, PERSON}}$ in HMM notation. In a similar fashion, we can define all K^2 transition features, where K is the size of tag set.

Of course the features are not limited to binary functions. Any real-valued function is allowed.

4 Undirected Graphical Models (Markov Random Fields)

CRF is a special case of undirected graphical models, also known as Markov Random Fields. A *clique* is a subset of nodes in the graph that are fully connected (having an edge between any two nodes). A maximum clique is a clique that is not a subset of any other clique. Let X_c be the set of nodes involved in a maximum clique c. Let $\psi(X_c)$ be an arbitrary non-negative real-valued function, called the *potential function*. In particular $\psi(X_c)$ does not need to be normalized. The Markov Random Field defines a probability distribution over the node states as the normalized product of potential functions of all maximum cliques in the graph:

$$p(X) = \frac{1}{Z} \prod_c \psi(X_c), \tag{6}$$

where Z is the normalization factor. In the special case of linear-chain CRFs, the cliques correspond to a pair of states z_{n-1}, z_n as well as the corresponding x nodes, with

$$\psi = \exp(\lambda f). \tag{7}$$

This is indeed the direct connection to factor graph representation as well. Each clique can be represented by a factor node with the factor $\psi(X_c)$, and the factor node connects to every node in X_c. There is one addition special factor node which represents Z.

A welcome consequence is that the sum-product algorithm and max-sum algorithm immediately apply to Markov Random Fields (and CRFs in particular). The factor corresponding to Z can be ignored during message passing.

5 CRF training

Training involves finding the λ parameters. For this we need fully labeled data sequences $\{(\mathbf{x}^{(1)}, \mathbf{z}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{z}^{(m)})\}$, where $\mathbf{x}^{(1)} = x^{(1)}_{1:N_1}$ the first observation sequence, and so on[1]. Since CRFs define the conditional probability $p(\mathbf{z}|\mathbf{x})$, the appropriate objective for parameter learning is to maximize the conditional likelihood of the training data

$$\sum_{j=1}^{m} \log p(\mathbf{z}^{(j)}|\mathbf{x}^{(j)}). \tag{8}$$

Often one can also put a Gaussian prior on the λ's to regularize the training (i.e., smoothing). If $\lambda \sim N(0, \sigma^2)$, the objective becomes

$$\sum_{j=1}^{m} \log p(\mathbf{z}^{(j)}|\mathbf{x}^{(j)}) - \sum_{i}^{F} \frac{\lambda_i^2}{2\sigma^2}. \tag{9}$$

The good news is that the objective is concave, so the λ's have a unique set of optimal values. The bad news is that there is no closed form solution[2].

The standard parameter learning approach is to compute the gradient of the objective function, and use the gradient in an optimization algorithm like L-BFGS. The gradient of the objective function is computed as follows:

$$\begin{aligned}
& \frac{\partial}{\partial \lambda_k} \sum_{j=1}^{m} \log p(\mathbf{z}^{(j)}|\mathbf{x}^{(j)}) - \sum_{i}^{F} \frac{\lambda_i^2}{2\sigma^2} && (10)\\
= \;& \frac{\partial}{\partial \lambda_k} \sum_{j=1}^{m} \left(\sum_n \sum_i \lambda_i f_i(z_{n-1}^{(j)}, z_n^{(j)}, \mathbf{x}^{(j)}, n) - \log Z^{(j)} \right) - \sum_{i}^{F} \frac{\lambda_i^2}{2\sigma^2} && (11)\\
= \;& \sum_{j=1}^{m} \sum_n f_k(z_{n-1}^{(j)}, z_n^{(j)}, \mathbf{x}^{(j)}, n) \\
& - \sum_{j=1}^{m} \sum_n E_{z'_{n-1}, z'_n}[f_k(z'_{n-1}, z'_n, \mathbf{x}^{(j)}, n)] - \frac{\lambda_k}{\sigma^2}, && (12)
\end{aligned}$$

[1] Unlike HMMs which can use the Baum-Welch (EM) algorithm to train on unlabeled data $\mathbf{x}$ only, CRFs training on unlabeled data is difficult

[2] If this reminds you of logistic regression, you are right: logistic regression is a special case of CRF where there is no edges among hidden states. In contrast, HMMs when trained on fully labeled data have simple and intuitive closed form solutions.

where we used the fact

$$\begin{aligned}
\frac{\partial}{\partial \lambda_k} \log Z &= E_{\mathbf{z}'}[\sum_n f_k(z'_{n-1}, z'_n, \mathbf{x}, n)] & (13)\\
&= \sum_n E_{z'_{n-1}, z'_n}[f_k(z'_{n-1}, z'_n, \mathbf{x}, n)] & (14)\\
&= \sum_n \sum_{z'_{n-1}, z'_n} p(z'_{n-1}, z'_n|\mathbf{x}) f_k(z'_{n-1}, z'_n, \mathbf{x}, n). & (15)
\end{aligned}$$

Note the edge marginal probability $p(z'_{n-1}, z'_n|\mathbf{x})$ is under the current parameters, and this is exactly what the sum-product algorithm can compute.

The partial derivative in (12) has an intuitive explanation. Let us ignore the term λ_k/σ^2 from the prior. The derivative has the form of (observed counts of feature f_k) minus (expected counts of feature f_k). When the two are the same, the derivative is zero, and there is no longer an incentive to change λ_k. Therefore we see that training can be thought of as finding λ's that match the two counts.

6 Feature Selection

A common practice in NLP is to define a very large number of candidate features, and let the data select a small subset to use in the final CRF model in a process known as feature selection. Often the candidate features are proposed in two stages:

1. Atomic candidate features. These are usually a simple test on a specific combination of words and tags, e.g.(x =John, z =PERSON), (x =John, z =LOCATION), (x =John, z =ORGANIZATION), etc. There are VK such "word identity" candidate features, which is obviously a large number. Although it is called the word identity test, it should be understood as *in combination with each tag value.* Similarly one can test whether the word is capitalized, the identity of the neighboring words, the part-of-speech of the word, and so on. The state transition features are also atomic.

   From the large number of atomic candidate features, a small number of features are selected by how much they improve the CRF model (e.g., increase in the training set likelihood).

2. "Grow" candidate features. It is natural to combine features to form more complex features. For example, one can test for current word being capitalized, the next word being "Inc.", and both tags being ORGANIZATION. However, the number of complex features grows exponentially. A compromise is to only grow candidate features on selected features so far, by extending them with one atomic additions, or other simple boolean operations.

Often any remaining atomic candidate features are added to the grown set. A small number of features are selected, and added to the existing feature set. This stage is repeated until enough features have been added.