Learning Markov Logic Networks

Log-linear models Suppose we are given factors $\phi_1, \phi_2, \ldots$, each of which maps a state $\mathbf{x}_i$ of the random variables $\mathbf{X}$ to a non-negative number. Note that each factor may take a different subset of the variables as input. This is a factor graph (or equivalently a Markov network, depending on how you draw the graph) whose semantics define a joint probability distribution as follows.

$$\Pr(\mathbf{x}) = \frac{\prod_i \phi_i(\mathbf{x}_i)}{Z} \text{ where } Z = \sum_{\mathbf{x}} \prod_i \phi_i(\mathbf{x}_i). \tag{1}$$

The normalizing constant Z is called the partition function. We will find it convenient to represent this factor graph as a *log-linear model*. In such a model, the factors ϕ_i are represented as exponentiated weighted features, with real-valued weights denoted w_i and features denoted $f_i(\mathbf{x})$:

$$\phi_i(\mathbf{x}_i) = \exp(w_i f_i(\mathbf{x}_i)) \tag{2}$$

The joint distribution can then be written as

$$\Pr(\mathbf{x}) = \frac{\prod_i \exp(w_i f_i(\mathbf{x}_i))}{Z} \tag{3}$$

$$\Pr(\mathbf{x}) = \frac{\exp\left(\sum_i w_i f_i(\mathbf{x}_i)\right)}{Z} \tag{4}$$

where now

$$Z = \sum_{\mathbf{x}} \exp\left(\sum_i w_i f_i(\mathbf{x}_i)\right). \tag{5}$$

One benefit of this transformation is that we can write $\sum_i w_i f_i(\mathbf{x}_i)$ instead as the dot product $\mathbf{w} \cdot \mathbf{f}$ of a weight vector $\mathbf{w}$ and a feature vector $\mathbf{f}$ (a linear model), which is a common notation in machine learning.

Markov logic networks Syntactically, a Markov logic network (MLN) is a set of weighted first-order logic formulas $\{w_i \quad \alpha_i\}$ where α_i is an arbitrary formula. One way to define MLN semantics is to think of them as templates for a log-linear model. Specifically, each grounding of the free variables in α_i defines a Boolean feature. For example

$$\alpha = [\mathit{Smokes}(x) \wedge \mathit{Friends}(x, y) \Rightarrow \mathit{Smokes}(y)] \tag{6}$$

has many groundings for x and y. For any particular choice, say

$$\theta = \{x \setminus \mathit{Alice}, y \setminus \mathit{Bob}\} \tag{7}$$

the sentence

$$\alpha\theta = [Smokes(Alice) \wedge Friends(Alice, Bob) \Rightarrow Smokes(Bob)] \tag{8}$$

is either true or false in a world. Therefore, we can use it as a feature:

$$f(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{x} \models \alpha\theta \\ 0 \text{ otherwise} \end{cases} \tag{9}$$

If α contains quantifiers, the semantics change:

$$\alpha = [\exists y . Smokes(x) \wedge Friends(x, y) \Rightarrow Smokes(y)] \tag{10}$$

$$\alpha\{x \setminus Alice\} = [\exists y . Smokes(Alice) \wedge Friends(Alice, y) \Rightarrow Smokes(y)] \tag{11}$$

Still, the sentences obtained after grounding the free variables are either true or false in a world, and are therefore Boolean features.

Let g_{ij} denote the feature induced by the jth grounding of formula α_i. Then, if the MLN semantics are simply defining a log-linear model with weights w_i for each of these features obtained by grounding the formula α, we get the following.

$$\Pr(\mathbf{x}) = \frac{\exp\left(\sum_k w_k f_k(\mathbf{x}_k)\right)}{Z} \tag{12}$$

$$\Pr(\mathbf{x}) = \frac{\exp\left(\sum_i \sum_j w_i g_{ij}(\mathbf{x}_{ij})\right)}{Z} \tag{13}$$

$$\Pr(\mathbf{x}) = \frac{\exp\left(\sum_i w_i \sum_j g_{ij}(\mathbf{x}_{ij})\right)}{Z} \tag{14}$$

$$\Pr(\mathbf{x}) = \frac{\exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)}{Z} \quad \text{where } Z = \sum_{\mathbf{x}} \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right) \tag{15}$$

and where

$$n_i(\mathbf{x}) = \sum_j g_{ij}(\mathbf{x}_{ij}) \tag{16}$$

Essentially, n_i is a new feature, which is counting how many groundings of α_i are true in world $\mathbf{x}$. That new feature has the same weight w_i as all the grounding features it summarizes.

Maximum-Likelihood Weight Learning Suppose we want to learn maximum-likelihood weights given the dataset $\mathcal{D} = \{\mathbf{x}^1, \mathbf{x}^2, \dots\}$, that is, find

$$w = \arg\max_w \Pr(\mathcal{D}; w) \tag{17}$$

$$= \arg\max_w \log \Pr(\mathcal{D}; w) \tag{18}$$

$$= \arg\max_w \log \prod_k \Pr(\mathbf{x}^k; w) \tag{19}$$

$$= \arg\max_w \sum_k \log \Pr(\mathbf{x}^k; w) \tag{20}$$

$$= \arg\max_w \sum_k \log \left(\frac{\exp\left(\sum_i w_i n_i(\mathbf{x}_i^k)\right)}{Z} \right) \tag{21}$$

$$= \arg\max_w \sum_k \log \left(\exp \left(\sum_i w_i n_i(\mathbf{x}_i^k) \right) \right) - \log(Z) \tag{22}$$

$$= \arg\max_w \sum_k \sum_i w_i n_i(\mathbf{x}_i^k) - \log(Z) \tag{23}$$

Intuitively, the maximum likelihood weights trade off two terms: increasing the weight of the data $\sum_i w_i n_i(\mathbf{x}_i)$ vs. how much this increase also increases the partition function Z.

This optimization problem has no easy closed-form solution, hence we will do gradient-based optimization. In order to do so, we need to compute all partial derivatives of the form

$$\frac{\partial \log \Pr(\mathbf{x}; w)}{\partial w_i} = \frac{\partial \sum_i w_i n_i(\mathbf{x}_i) - \log(Z)}{\partial w_i} \tag{24}$$

$$= \frac{\partial \sum_i w_i n_i(\mathbf{x}_i)}{\partial w_i} - \frac{\partial \log(Z)}{\partial w_i} \tag{25}$$

$$= n_i(\mathbf{x}_i) - \frac{\partial \log(Z)}{\partial w_i} \tag{26}$$

That is, the partial derivative of the likelihood given a single example w.r.t. weight w_i is the difference between $n_i(\mathbf{x}_i)$, the number of true groundings of the formula in the database, and the partial derivative of the log-partition function. We can further simplify the latter:

$$\frac{\partial \log(Z)}{\partial w_i} = \frac{\partial \log\left(\sum_{\mathbf{x}} \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)\right)}{\partial w_i} \tag{27}$$

$$= \frac{\left(\frac{\partial \sum_{\mathbf{x}} \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)}{\partial w_i}\right)}{\sum_{\mathbf{x}} \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)} \tag{28}$$

$$= \frac{\left(\frac{\partial \sum_{\mathbf{x}} \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)}{\partial w_i}\right)}{Z} \tag{29}$$

$$= \frac{\sum_{\mathbf{x}} \frac{\partial \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)}{\partial w_i}}{Z} \tag{30}$$

$$= \frac{\sum_{\mathbf{x}} n_i(\mathbf{x}_i) \exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)}{Z} \tag{31}$$

$$= \sum_{\mathbf{x}} n_i(\mathbf{x}_i) \frac{\exp\left(\sum_i w_i n_i(\mathbf{x}_i)\right)}{Z} \tag{32}$$

$$= \sum_{\mathbf{x}} n_i(\mathbf{x}_i) \Pr(\mathbf{x}) \tag{33}$$

$$= \mathbb{E}[n_i(\mathbf{x}_i)] \tag{34}$$

The partial derivative of the log-partition function is simply the expectation of the corresponding feature in the distribution parameterized by w. Hence, the partial derivative of the likelihood for a single data point $\mathbf{x}$ is

$$\frac{\partial \log \Pr(\mathbf{x}; w)}{\partial w_i} = n_i(\mathbf{x}_i) - \mathbb{E}_w[n_i(\mathbf{x}_i)]. \tag{35}$$

It is the difference between the count in the data $n_i(\mathbf{x}_i)$ and the expected count $\mathbb{E}_w[n_i(\mathbf{x}_i)]$ in the distribution parametrized by the current weights w. It is clear that at the maximum-likelihood weights $w^\star$, these partial derivatives are all zero, and therefore

$$n_i(\mathbf{x}_i) = \mathbb{E}_{w^\star}[n_i(\mathbf{x}_i)]. \tag{36}$$

Therefore, algorithms that maximize this equation are also called *moment matching* algorithms: they ensure that the "moment" $n_i(\mathbf{x}_i)$ in the data distribution and the learned distribution are equal.

To do gradient optimization in practice, we need the ability to estimate the expectation $\mathbb{E}[n_i(\mathbf{x}_i)]$. Recall that

$$n_i(\mathbf{x}) = \sum_j g_{ij}(\mathbf{x}_{ij}) \tag{37}$$

and that therefore

$$\begin{aligned}
\mathbb{E}[n_i(\mathbf{x}_i)] &= \sum_{\mathbf{x}} n_i(\mathbf{x}_i) \Pr(\mathbf{x}) && (38)\\
&= \sum_{\mathbf{x}} \sum_j g_{ij}(\mathbf{x}_{ij}) \Pr(\mathbf{x}) && (39)\\
&= \sum_j \sum_{\mathbf{x}} g_{ij}(\mathbf{x}_{ij}) \Pr(\mathbf{x}) && (40)\\
&= \sum_j \mathbb{E}\left[g_{ij}(\mathbf{x}_{ij})\right] && (41)
\end{aligned}$$

Finally, because g_{ij} is a logical sentence (and thus it is a Boolean random variable), we have that

$$\mathbb{E}[n_i(\mathbf{x}_i)] = \sum_j \Pr\left(g_{ij}(\mathbf{x}_{ij})\right). \tag{42}$$

Thus, the expectation of the count feature can be computed as the sum of the probabilities of all the groundings of the corresponding formula.

Bayesian Weight Learning One issue with the approach above is with the scenario where the count $n_i(\mathbf{x}_i) = 0$ for some sentence α_i. There is no way to make $\mathbb{E}_{w^\star}[n_i(\mathbf{x}_i)] = 0$ in a log linear-model, and gradient-based learning will simply keep reducing w_i, pushing it towards negative infinity, and never converge. Moreover, we do not even want to make $\mathbb{E}_{w^\star}[n_i(\mathbf{x}_i)] = 0$: it means we are clearly overfitting the training data.

Bayesian learning assumes a prior distribution on the parameters: $\Pr(w)$. The likelihood model is still of the form above, now denoted $\Pr(\mathbf{x}|w)$. The joint distribution over data and parameters is therefore $\Pr(\mathbf{x}, w)$. Learning in this context is simply obtaining the posterior distribution on parameters $\Pr(w|\mathbf{x})$. The parameters we seek to learn are now either the expected value of w in this posterior (Bayesian estimates), or the most likely value of w (maximum-a-posteriori estimates):

$$\begin{aligned} w &= \arg\max_w \Pr(w|\mathbf{x}) && (43) \\ &= \arg\max_w \log \Pr(w|\mathbf{x}) && (44) \\ &= \arg\max_w \log \frac{\Pr(\mathbf{x}|w)\Pr(w)}{\Pr(\mathbf{x})} && (45) \\ &= \arg\max_w \log \Pr(\mathbf{x}|w) + \log \Pr(w) - \log \Pr(\mathbf{x}) && (46) \end{aligned}$$

Since $\Pr(\mathbf{x})$ does not depend on w, this is also.

$$w = \arg\max_w \log \Pr(\mathbf{x}|w) + \log \Pr(w) \tag{47}$$

The first term in this objective function is again the log-likelihood, as in maximum-likelihood learning, which we already know how to optimize. The second term aims to learn weights w that are more likely a priori. Depending on the assumptions on the distribution of this prior (typically, that $\Pr(w)$ is a Gaussian), this second term can be written as a norm of the weight vector (typically L2 norm). Hence, Bayesian learning amounts to simply adding a regularization term to the weight learning loss function objective.