Learning Markov Networks: Parameters and Structure

Vibhav Gogate
(Some slides borrowed from Pedro Domingos)

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

What we will cover?

- Learning parameters (or weights) given structure: FOD + MLE/Bayesian
  - Generatively: Learn $P(\mathbf{X}, \mathbf{Y})$
  - Discriminatively: Learn $P(\mathbf{Y}|\mathbf{X})$
- Learning with incomplete data given structure: POD+ MLE/Bayesian
- Learning structure (features): Two algorithms

Alternative Parameterization for Convenience:

- Write the Markov network as a **log-linear model**

$$P_{\Theta}(x) = \frac{1}{Z(\Theta)} \exp\left(\sum_i \theta_i f_i(x_i)\right)$$

where f_i is a feature, namely a 0/1 function, θ_i is the weight associated with feature f_i, x_i is the projection of x on f_i and $Z(\Theta)$ is the partition function.

Generative Weight/Parameter Learning: FOD case

- Learn $P(\mathbf{X})$ by Maximizing likelihood or posterior probability
- Unlike Bayesian networks, no closed-form solution. Use iterative optimization algorithms such as gradient ascent.
- Good news: No local maxima (i.e., a single global maxima). Concave Objective function
- However, bad news: Requires Inference at each iteration of gradient ascent! Too slow.

Derivative of Log-Likelihood: FOD + MLE case: Part 1

We are given a **log-linear model**: $P_\Theta(x) = \frac{1}{Z(\Theta)} \exp\left(\sum_i \theta_i f_i(x_i)\right)$ and a **dataset** $\mathcal{D} = (x^{(1)}, \ldots, x^{(M)})$

- Log-likelihood of the log-linear model given data

$$
\begin{aligned}
LL(\Theta : \mathcal{D}) &= \ln\left(\prod_{j=1}^{M} P_\Theta(x^{(j)})\right) = \sum_{j=1}^{M}\sum_{j=1}^{M} \ln\left(\frac{1}{Z(\Theta)} \exp\left(\sum_i \theta_i f_i(x_i^{(j)})\right)\right) \\
&= \sum_{j=1}^{M}\left(\sum_i \theta_i f_i(x_i^{(j)}) - \ln Z(\Theta)\right) \\
&= \left[\sum_i \theta_i \left(\sum_{j=1}^{M} f_i(x_i^{(j)})\right)\right] - M \ln Z(\Theta)
\end{aligned}
$$

Throughout i indexes the features (e.g., f_i) or the parameters (e.g., θ_i)) and j indexes the examples (e.g., $x^{(j)}$).

Derivative of Log-likelihood: Part 2

- For convenience: Rewrite the Log-likelihood by dividing both sides by M

$$\frac{1}{M} LL(\Theta : \mathcal{D}) = \frac{1}{M} \left[\sum_i \theta_i \left(\sum_{j=1}^{M} f_i(x_i^{(j)}) \right) \right] - \ln Z(\Theta)$$

- The first expression on the right hand side is expected value of the feature from the data, multiplied by θ_i. Therefore, we can rewrite the Likelihood as:

$$\frac{1}{M} LL(\Theta : \mathcal{D}) = \sum_i \theta_i \mathbb{E}_{\mathcal{D}}[f_i(x_i^{(j)})] - \ln Z(\Theta)$$

Derivative of Log-likelihood: Part 3

- Taking partial derivative with respect to θ_i

$$\frac{\partial \frac{1}{M} LL(\Theta : \mathcal{D})}{\partial \theta_i} = \frac{\partial \sum_i \theta_i \mathbb{E}_{\mathcal{D}}[f_i(x_i^{(j)})]}{\partial \theta_i} - \frac{\partial \ln Z(\Theta)}{\partial \theta_i}$$

- The first expression on the RHS equals $\mathbb{E}_{\mathcal{D}}[f_i(x_i^{(j)})]$. The estimate of this value is the number of times the feature f_i is true in the data! Nice, easy to compute.
- The second expression, we have to derive separately

Derivative of Log-likelihood: Part 4

- Recall that $Z(\Theta)$ is given by:

$$Z(\Theta) = \sum_x \exp\left(\sum_i \theta_i f_i(x_i)\right)$$

- Partial derivative of $\ln Z(\Theta)$:

$$\begin{aligned}
\frac{\partial \ln Z(\Theta)}{\partial \theta_i} &= \frac{1}{Z(\Theta)} \frac{\partial}{\partial \theta_i} Z(\Theta) = \frac{1}{Z(\Theta)} \sum_x \frac{\partial}{\partial \theta_i} \exp\left(\sum_i \theta_i f_i(x_i)\right) \\
&= \frac{1}{Z(\Theta)} \sum_x \exp\left(\sum_i \theta_i f_i(x_i)\right) \frac{\partial}{\partial \theta_i} \sum_i \theta_i f_i(x_i) \\
&= \sum_x \left[\frac{1}{Z(\Theta)} \exp\left(\sum_i \theta_i f_i(x_i)\right)\right] f_i(x_i) = \sum_x P_\Theta(x) f_i(x_i) \\
&= \mathbb{E}_{P_\Theta}[f_i(x_i)]
\end{aligned}$$

Derivative of Log-Likelihood: Part 5

- In summary, We have the following expression:

$$\frac{1}{M}\frac{\partial}{\partial\theta_i}LL(\Theta : \mathcal{D}) = \mathbb{E}_{\mathcal{D}}[f_i(x_i)] - \mathbb{E}_{P_\Theta}[f_i(x_i)]$$

- The first expression on the RHS is the number of times feature f_i is true in the data
- The second expression is the expected number of times feature f_i is true given current set of parameters Θ. This requires inference over the model (sum-product inference)
- A simple gradient ascent procedure:
  - Start with random parameters
  - At each iteration t, update each θ_i^t using:

$$\theta_i^{t+1} = \theta_i^t + \eta\left(\mathbb{E}_{\mathcal{D}}[f_i(x_i)] - \mathbb{E}_{P_{\Theta^t}}[f_i(x_i)]\right)$$

  - Stop when converged.

Gradient of Posteriors: MAP Estimation

- MAP estimation: to find the parameters that maximize $P(\Theta)P(\mathcal{D}|\Theta)$
- Use an independent Gaussian or Laplacian Prior:

$$P_2(\Theta) = \prod_i \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\theta_i^2}{2\sigma^2}\right) \qquad \text{or} \qquad P_1(\Theta) = \prod_i \frac{1}{2\beta} \exp\left(-\frac{|\theta_i|}{\beta}\right)$$

- In log-space we have $\ln(P(\Theta)) + \ln(P(\mathcal{D}|\Theta))$ and the first term for the two priors becomes:

$$-\lambda_2 \sum_i \theta_i^2 \qquad \text{or} \qquad -\lambda_1 \sum_i |\theta_i|$$

- Gradients of the new terms: $\{-2\lambda_2\theta_i\}$ and $\{-\lambda_1 \frac{\theta_i}{|\theta_i|}$ s.t. $\theta_i \neq 0\}$ respectively.
- Small change in gradient ascent for Gaussian Priors:

$$\theta_i^{t+1} = \theta_i^t + \eta\left(\mathbb{E}_{\mathcal{D}}[f_i(x_i)] - \mathbb{E}_{P_{\Theta^t}}[f_i(x_i)] - 2\lambda_2\theta_i^t\right)$$

Interpretation of Priors as ℓ_2 and ℓ_1 penalty

- $-\lambda_2 \sum_i \theta_i^2$ places a quadratic penalty on the magnitude of the weights (namely penalizes large parameter values), generally called an ℓ_2-**regularization** term.
- $-\lambda_1 \sum_i |\theta_i|$ places a linear penalty, measured using the $\ell_1 - norm$ and therefore called ℓ_1-**regularization** term.
- ℓ_1 penalty is linear and quite sharp at zero. Therefore, in practice it may yield many parameters that have zero weights!
  - The models learned with an ℓ_1 penalty tend to be much sparser than those learned with an ℓ_2 penalty.

Issues and Possible Approximations: ML/Bayesian Parameter Estimation in Log-Linear Models

- MLE with or without ℓ_1, ℓ_2 penalty is impractical because each iteration of gradient ascent requires exact inference (exponential in the treewidth!)
- Two approaches to address this issue
  - **Use Alternative Objective Functions** such that exact inference is not required or is fast (Example: Pseudo Likelihood, Contrastive Divergence, etc.)
  - **Use Approximate Inference Algorithms** (Belief Propagation, Sampling-Based Inference, MAP inference, etc.)
- No free lunch: Possible loss of convergence/consistency guarantees and solution may have low likelihood (far from optimal).

Alternative Objective Functions: Pseudo Likelihood

- Likelihood of each variable given its neighbors in the data

$$PL(x) = \prod_{i=1}^{n} P_{\Theta}(x_i | x_{-i}) = \prod_{i=1}^{n} P_{\Theta}(x_i | x_{neighbors(X_i)})$$

- Does not require inference at each step. Expression for Gradient given in the book.
- Consistent estimator
- Widely used in vision, spatial statistics, etc.
- But PL parameters may not work well for long inference chains
- Improvements: Block PL

Discriminative/Conditional Parameter Learning

- Maximize conditional likelihood of query (**Y**) given evidence (**X**)
- Gradient:

$$\sum_{j=1}^{m} \left(f_i(x^{(j)}, y^{(j)}) - \mathbb{E}_{\Theta} \left[f_i | x^{(j)} \right] \right)$$

- For each example gradient equals the number of times (either 0 or 1) the feature is true in the data minus the number of times the feature is true according to the model conditioned on the evidence.
- In generative models, each gradient ascent iteration required only a single execution of inference. In the discriminative or conditional case, we require inference for each example!
- However, in the conditional case, inference is easier because all variables in the set **X** are assigned a value (evidence).

Discriminative Learning: Voted Perceptron

- Approximate expected counts in the MPE state of the query variables **Y** given evidence $\mathbf{X} = \mathbf{x}$
- Originally proposed for training HMMs discriminatively (Collins, 2002)
- Gradient:

$$\sum_{j=1}^{m} \left(f_i \left(x^{(j)}, y^{(j)} \right) - f_i \left(x^{(j)}, y_{MPE|x^{(j)}} \right) \right)$$

- To reduce bias, typically return an average. Suppose the gradient ascent algorithm is run for T iterations:

$$\theta_i = \frac{1}{T} \sum_{t=1}^{T} \theta_i^t$$

Learning Parameters with Missing Data: POD

- **Recall:** Maximize Log-Likelihood but the likelihood function is multi-modal!
- W.L.o.G. Let **Y** be missing (or hidden) and **X** be observed in the data
- Gradient:

$$\frac{1}{M}\sum_{j=1}^{M}\mathbb{E}_{\Theta}\left[f_i|x^{(j)}\right]-\mathbb{E}_{\Theta}[f_i(x_i)]$$

- Gradient equals feature expectation over the data and the hidden variables minus the feature expectation over all of the variables.

Learning Structure of Markov Networks

- Active Research area.
- Use Bayesian scores because similar to Bayes nets, complete graph is the optimal solution according to the Max-Likelihood criteria.
- Popular Algorithms
  - Greedy Structure search.
    - Start with atomic features
    - Greedily conjoin features to improve score
    - Problem: Need to reestimate weights for each new candidate
    - Approximation: Keep weights of previous features constant
  - Logistic Regression with ℓ_1 regularization for learning pairwise networks
    - Run logistic regression with ℓ_1 penalty for each variable X with X as the class variable and all other variables as features
    - Remove all edges that have zero weights. If there is a conflict on an edge, either take unions or intersections.

Summary

- Learning parameters (or weights) given structure: FOD + MLE/Bayesian
  - Generatively: Learn $P(\mathbf{X}, \mathbf{Y})$.
  - Discriminatively: Learn $P(\mathbf{Y}|\mathbf{X})$
- ML/Bayesian estimation is slow. Possible fixes: use approximate inference algorithms or alternative objective functions.
- Learning with incomplete data given structure: POD+ MLE/Bayesian. (Just the gradient changes)
- Learning structure (features): Two algorithms