Jin Gu
Department of Automation, Tsinghua University
Email: jgu@tsinghua.edu.cn
Phone: (010) 62794294-866

Chapter 10 Parameter Learning with Complete Data

2021 Fall

Jin Gu (古槿)

Move to *Learning*

- Representation

$$\boxed{P \Leftrightarrow \{P, G\}}$$

  - Parents→child structures & cliques
  - Gaussian models & exponential families
- Inference

$$\boxed{P\left(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e}, \boldsymbol{\theta}\right)}$$

  - Particle-based inference
  - Inference as optimization
- Learning

$$\boxed{\max_{\boldsymbol{\theta}} P\left(\boldsymbol{x}[1], \boldsymbol{x}[2], \cdots \qquad \mid \boldsymbol{\theta}\right)}$$

$$\boxed{P\left(\boldsymbol{\theta} \mid \boldsymbol{x}[1], \boldsymbol{x}[2], \cdots \qquad \right)}$$

Three Lectures

- Parameter learning with complete data
- Learning with hidden models or missing values
- Structure learning
- Learning with deep models & general graphs

- ***Quiz II, 2021/12/20, Inference & Learning***
- OOH: Friday 8:30-9:30
- Anytime with TAs (make appointment by email)

Outlines

- Learning Basics
- Maximum Likelihood Parameter Estimation
  - Basics for Maximum Likelihood Estimation
  - Maximum Likelihood Estimation in BNs
- Bayesian Parameter Estimation
  - Basics for Bayesian Estimation
  - Priors for Bayesian Estimation
  - Bayesian Estimation in BNs
- MAP Parameter Estimation

Outlines

- Parameter Learning in Markov Networks
  - The likelihood of Markov networks
  - The convexity of partition function
  - Recall: stochastic gradient descent
    - Example: CRFs learning revisited

- *Approximate methods*
  - *Pseudo likelihood (cannot deal with hidden values)*
  - *Contrastive divergence*

Chapter 10 Parameter Learning with Complete Data

Textbook1

Chapter 17.1-17.2 Maximum Likelihood Estimation in Bayesian Networks
Chapter 17.3-17.4 Bayesian Learning in Bayesian Networks
Chapter 20.1-20.3 Maximum Likelihood Estimation in Markov Networks

Textbook2

Chapter 10.4 Learning in Directed Graphical Models
Chapter 19.5 Learning in Undirected Graphical Models

From the View of ***Probability***

What is Learning?

- Draw *probabilistic models* from *observed data*
- Learning as *optimization* $\boxed{\max_{\boldsymbol{\theta}} P\left(\boldsymbol{x}[1], \boldsymbol{x}[2], \cdots \qquad \mid \boldsymbol{\theta}\right)}$
  - Find an optimal probabilistic model which can maximize a given objective function
- Learning as *probabilistic* inference
  - Treat the parameters as variables and infer their posteriors based on observed data

$$\boxed{P\left(\boldsymbol{\theta} \mid \boldsymbol{x}[1], \boldsymbol{x}[2], \cdots \qquad \right)}$$

Learning as Optimization

- Maximum likelihood $\boxed{\boldsymbol{\theta}^* = \max_{\theta} P\left(\boldsymbol{x}[1], \boldsymbol{x}[2], \cdots, \boldsymbol{x}[M] \mid \boldsymbol{\theta}\right)}$

- Maximum a posterior $\boxed{\boldsymbol{\theta}^* = \max_{\theta} P\left(\boldsymbol{\theta} \mid \boldsymbol{x}[1], \boldsymbol{x}[2], \cdots, \boldsymbol{x}[M]\right)}$

- *Not as a probabilistic problem*
  - *As minimum error*
  - *As maximum margin*
  - *As compressive sensing*
  - ….

Define proper *loss* or *likelihood* functions *based on different principles*

Learning as Probabilistic Inference

- Learning in a probabilistic framework
  - Parameter learning: $\{x[m]\}_{m=1\sim M}|_G \rightarrow P(\theta|\mathcal{D})$
  - Structure learning: $\{x[m]\}_{m=1\sim M} \rightarrow P(\mathcal{G},\theta|\mathcal{D})$

- Generative models
  - Learn the joint representation $\tilde{P}(Y,X|\theta)$
- Discriminative models
  - Learn the conditional probability $\tilde{P}(Y|X=x,\theta)$

Learning Basics: IID Samples

- For most learning tasks, the variables are required to be "independent and identical distributed" (IID or *i.i.d.*). The observed samples are generated from those variables are called IID samples.
- IID means that the generation of a specific sample **is unrelated** to all the other samples
- *How to collect samples from Gibbs sampling?*

Learning Basics: Avoid Overfitting

- Overfitting
  - If the model complexity (or the freedom of model) is much larger than the training data, we can always get "good" learning with zero empirical risk
  - But when testing on new data, the learned model does bad predictions
- Generalization
  - To avoid overfitting or increase the generalization of learning methods, we need to *penalize the model complexity* and *separate training/testing*

Avoid Overfitting (Theoretical)

- The regularization term in loss function
  - Bayesian learning
  - Statistical learning theory: structural risk
  - Sparsity
  - Low-rank
  - …..

Model **complexity** should fit the complexity of data!

Avoid Overfitting (Empirical)

- Cross-validation
  - LOOCV (leave one out cross-validation)
  - N-fold cross-validation

- 0.632 bootstraping $(0.632 = 1 - e^{-1})$

Training performance should be similar with testing performance!

0.632 Bootstrapping

- For *M* samples, *x*[1], ..., *x*[*M*]
- Randomly generated *M* new samples by with-replacement sampling: each time randomly choose a sample *x*[*k*] and then put it back to the sample pool
- On average, 0.632**M* samples will be selected into the training dataset
- The left 0.368**M* samples will be used as the testing dataset

0.632 Bootstrapping

- Then, learn your model on the training dataset and test its performance on the testing data
- Repeat above steps many times
- Check whether the performances are significant lower in testing than training (if yes, your learning is over-fitting)
- If similar, you get the estimated performance

$$0.368 * Perform_{Train} + 0.632 * Perform_{Test}$$

Parameter Estimation

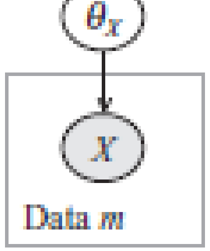
- For a set of samples, they are correlated if the distribution parameters are not given or observed

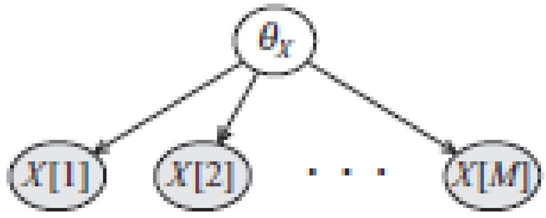
$$P(x[1], x[2], \dots, x[M]) \neq P(x[1]) \cdots P(x[M])$$

- If we assume the distribution parameters are given, all the samples are *i.i.d.*

$$P(x[1], \dots, x[M] | \theta) = P(x[1] | \theta) \cdots P(x[M] | \theta)$$

i.i.d assumption!





Maximum Likelihood Parameter Estimation

- What is *likelihood*?
  - The probability or confidence for parameter assignment, given a number of data
  - Log likelihood is commonly used for better calculation
    - $l(\theta : \mathcal{D}) \propto \log \tilde{P}(\mathcal{D}|\theta) = \sum_i \log \phi_i(x[i]; \theta)$
- *MLE*: the parameter estimation is to find the optimal parameter assignment θ^* which can maximize the likelihood (optimization)

MLE: Maximum Likelihood Estimation

- We throw a five cent coin M times with $M[T]$ text-face up and $M[L]$ as lotus-face up. Please find θ^* of the coin $(P(T) = \theta)$ which is most likely to generate that result.

- The log-likelihood

$$l(\theta : X) = \log\left(\theta^{M[T]}(1-\theta)^{M[L]}\right).$$

- Set the derivative of the log-likelihood as 0

$$\frac{M[T]}{\theta^*} - \frac{M[L]}{1-\theta^*} = 0 \rightarrow \theta^* = \frac{M[T]}{M[T]+M[L]}.$$

Maximum Likelihood Estimation

- In many situations, optimization algorithms should be used to maximize the likelihood function (commonly in log-linear format)
  - For complex likelihood function, it is hard to get the close form of its derivative
  - For incremental or updating learning, we want to improve the likelihood sample by sample
  - Regularization terms can be added to control the model complexity (such as LASSO)

Gradient-Based Methods

- Gradient descent/ascent is a common strategy to find the optimal parameter setting. Simply update the model along the gradient:
$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \lambda^{(k)} \nabla l(\boldsymbol{\theta}; \boldsymbol{x})$$
- If the likelihood function $l(\theta; x)$ is convex, the gradient ascent will converge to the optimal
- In most cases θ_i is independent with each other, you can update them one by one using partial derivative on θ_i

Stochastic Gradient Ascent

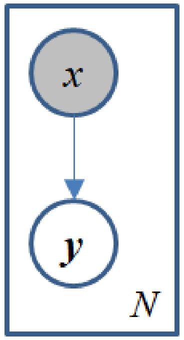
- If the observed samples are *i.i.d.*, the likelihood function are the sum across each sample
$$l(\boldsymbol{\theta}; \boldsymbol{x}) = \sum_{m=1}^{M} l_m(\boldsymbol{\theta}; \boldsymbol{x}[m]).$$
- In many situations, the computational cost is huge for the sum
- Stochastic gradient ascent updates parameters based on one sample each time
$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \lambda^{(k)} \nabla l_{m_k}(\boldsymbol{\theta}; \boldsymbol{x}[m_k]).$$

$$\boxed{\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \lambda^{(k)} \nabla l_{m_k}(\boldsymbol{\theta}; \boldsymbol{x}[m_k])}$$

- The order of the input samples are randomly given in each run
- The input samples can be used repeatedly (each time with a different random order) until the optimization process converges

Comments: to avoid *local minimal*, heuristic methods should be used, such as simulated annealing and genetic algorithm

Logistic Regression Gradient



- Logistic regression

$$p\left(y=1 \mid \bar{x}\right)=\frac{e^z}{1+e^z}, \quad z=-\sum_j \beta_j x_j$$

- If y = 0 the partial derivative of $log(1-p)$ is

$$\frac{\partial}{\partial \beta_j}\log\left(1-p\right)=\frac{\partial}{\partial \beta_j}\left(\log\left(\frac{1}{1+e^z}\right)\right)=px_j$$

- If y = 1 the partial derivation of $log(p)$ is

$$\frac{\partial}{\partial \beta_j}\log\left(p\right)=\frac{\partial}{\partial \beta_j}\left(\log\left(\frac{e^z}{1+e^z}\right)\right)=\frac{\partial}{\partial \beta_j}\left(\log\left(e^z\right)+\log\left(\frac{1}{1+e^z}\right)\right)=\left(p-1\right)x_j$$

Logistic Regression Gradient

- So the partial derivative can be written as

$$\sum_{i:y=0} p_i x_{ij} + \sum_{i:y=1} (p_i - 1) x_{ij} = \sum_i (p_i - y_i) x_{ij}$$

- Set the partial derivation as 0, so we get

$$\sum_i y_i x_{ij} = \sum_i p_i x_{ij}$$

- This equation can be used to check the correctness of a trained model

Logistic Regression Gradient

For any single training sample, the partial derivative of the log likelihood is

$$\boxed{\frac{\partial}{\partial \beta_j} \log L(x, y; \beta) = (y - p) x_j}$$

- Learning algorithm:
  - The parameters are given initial values $\beta^{(0)}$
  - Calculate $p^{(0)}$ according to the regression
  - Then, update the parameters based on the partial derivative

$$\beta_j^{(k+1)} := \beta_j^{(k)} + \lambda^{(k)} \left(y - p^{(k)} \right) x_j$$

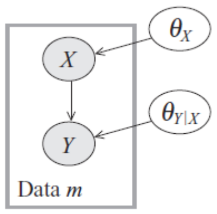
Learning rate λ should be decreased as the iterative steps increase

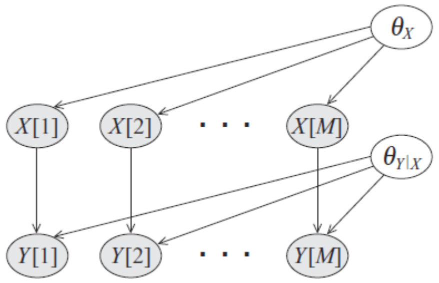
MLE in Bayesian Networks

- Global likelihood decomposition
  - Recall: $P(X) = \prod_i P\left(X_i | Pa_{X_i}\right)$

  $L(\theta : X) = \prod_m P(x[m]; \theta) = \prod_i \prod_m P\left(x_i[m] | Pa_{x_i[m]};\ \theta_{x_i | Pa_{x_i}}\right)$
- We get the local likelihood function

  $L_i(\theta : X) = \prod_m P\left(x_i[m] | Pa_{x_i[m]};\ \theta_{x_i | Pa_{x_i}}\right)$
- According to the decomposition, we can maximize each local likelihood **independently** and combine them as an MLE solution





MLE in Bayesian Networks

- For a local likelihood, only take the data scopes of the variables X_i and its parents Pa_{X_i}
- Learning as table CPDs
  - Count the frequency as the parameter

$$\theta_{X_i=x_i|Pa_{X_i}=Pa_{x_i}} = M[x_i, Pa_{x_i}]/M[Pa_{x_i}]$$

- Learning as generalized linear models
  - The same approach as logistic regression above

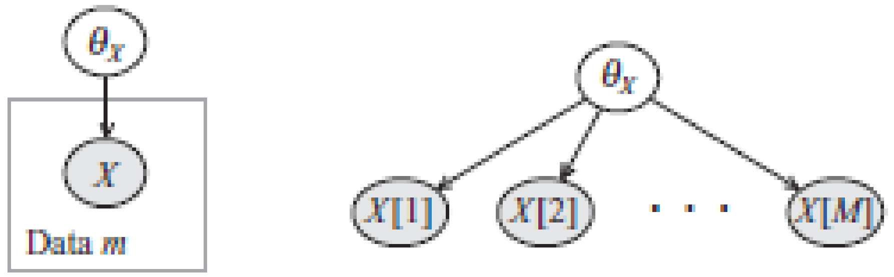
The learning process is totally independent!

If the network structure is sparse, the learning algorithm can work very efficiently

Bayesian Parameter Estimation

- MLE is for *point estimation*, but not for *prediction* or *distribution estimation*
- MLE is maximizing the joint probability of the observed data for finding the optimal parameter, but it *cannot* use the prior knowledge or constraint of the parameter
- We can do estimation using Bayes' rule

$$P(\theta|X) = {}^{P(X|\theta)P(\theta)}\Big/_{\int P(X|\theta)P(\theta)d\theta}$$



- We get the posterior distribution of the parameter rather than a point estimation
- In most cases, the Bayesian model is used to do prediction (on new data)

$$P(X^{n+1}|X^1,\cdots,X^n) = \frac{1}{P(X^1,\cdots,X^n)}\int P(X^{n+1}|\theta)P(X^1,\cdots,X^n|\theta)P(\theta)d\theta$$

- Recall the first homework: we assume θ is set as 0.5, and we observed 63 “upward” out of 100 random throws. Please show the probability that we get “upward” for the 101-th throw.

Priors for Parameters

- The priors of parameters

$$\frac{1}{P(X^1,\cdots,X^n)}\int P(X^{n+1}|\theta)P(X^1,\cdots,X^n|\theta)\boldsymbol{P(\theta)}d\theta$$

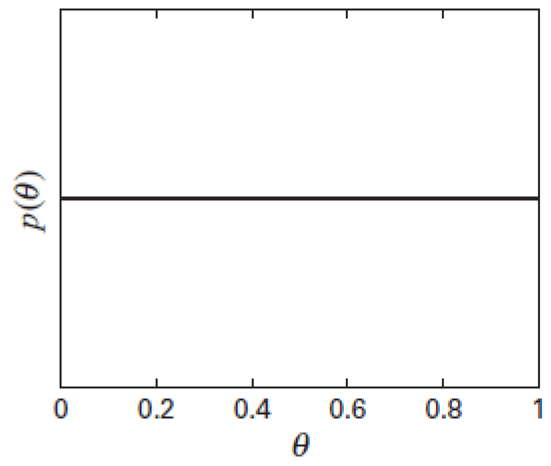
- Use *uniform* distribution if no prior
- The *Beta* distribution is more commonly used

$\theta \sim \mathrm{Beta}(\alpha_1, \alpha_0)$ if $p(\theta) = \gamma\theta^{\alpha_1 - 1}(1-\theta)^{\alpha_0 - 1}$

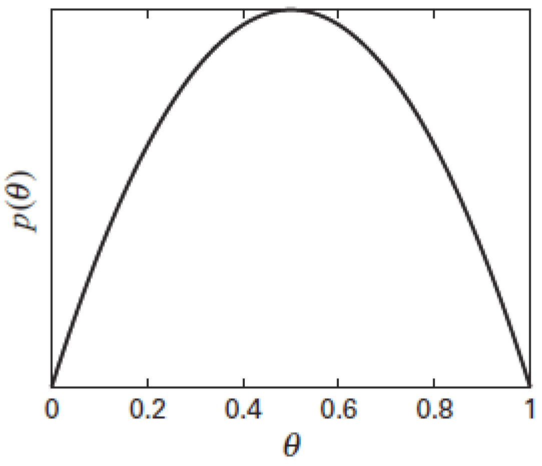
  - $\gamma = \frac{\Gamma(\alpha_1+\alpha_0)}{\Gamma(\alpha_1)\Gamma(\alpha_0)}$ is a normalizing factor
  - $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt, \Gamma(n) = (n-1)!\, for\ integer$

$$\boxed{p(\theta) = \gamma\theta^{\alpha_1 - 1}(1-\theta)^{\alpha_0 - 1}}$$

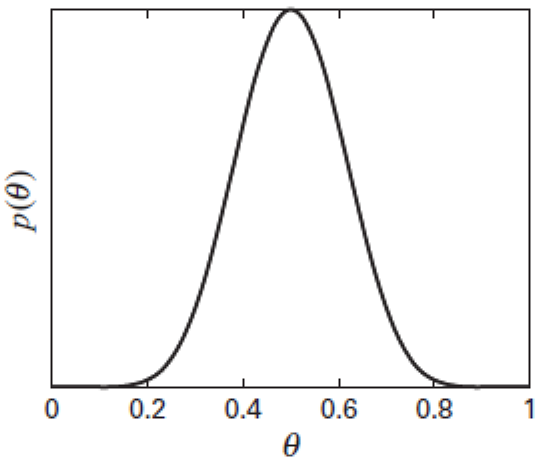
Priors: Beta Distribution



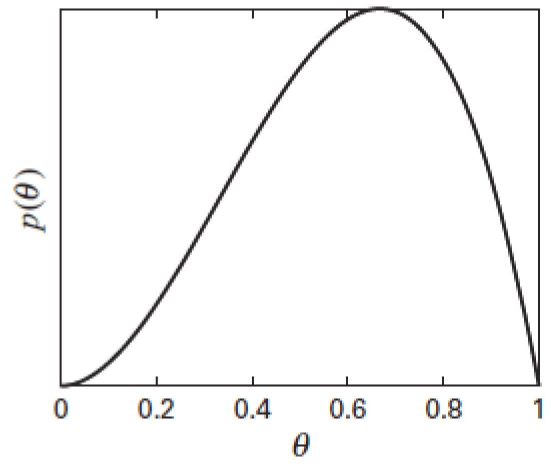
$Beta(1,1)$



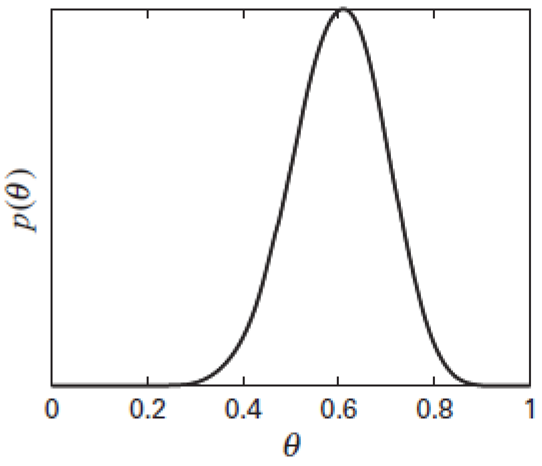
$Beta(2,2)$



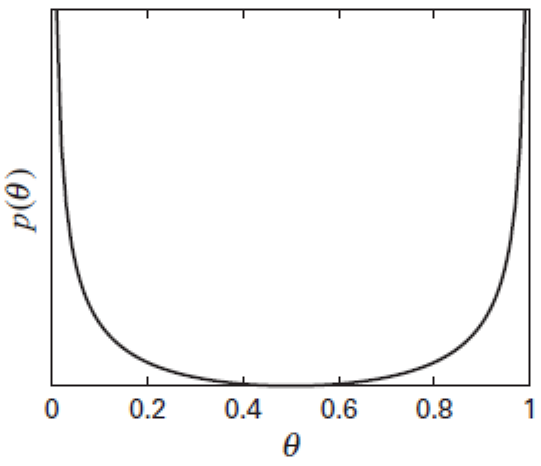
$Beta(10,10)$



$Beta(3,2)$



$Beta(15,10)$



$Beta(0.5,0.5)$

Priors: Beta Distribution

- For parameter distribution estimation

$$P(\theta | X) = \frac{P(X|\theta)P(\theta)}{\int P(X|\theta)P(\theta)d\theta}$$

All calculations are in the same family. We say that Beta distribution is *conjugate*.

$$= \frac{\theta^{M[1]}(1-\theta)^{M[0]}\gamma\theta^{\alpha_1-1}(1-\theta)^{\alpha_0-1}}{\int \theta^{M[1]}(1-\theta)^{M[0]}\gamma\theta^{\alpha_1-1}(1-\theta)^{\alpha_0-1}d\theta} = Beta\left(M[1]+\alpha_1, M[0]+\alpha_0\right)$$

Priors: Beta Distribution

$$\boxed{(uv)' = u'v + uv' \Rightarrow \int_a^b uv' = uv\,|_a^b - \int_a^b u'v}$$

- For prediction

$$P\left(x[m+1]=1 \mid X\right) = \int P\left(x[m+1] \mid \theta, X\right) P(\theta \mid X) d\theta$$

$$= \int P\left(x[m+1] \mid \theta\right) P(\theta \mid X) d\theta$$

$$P(\theta \mid X) \propto P(X \mid \theta) P(\theta) \propto \theta^{M[1]} (1-\theta)^{M[0]} \theta^{\alpha_1 - 1} (1-\theta)^{\alpha_0 - 1}$$

$$\therefore P\left(x[m+1]=1 \mid X\right) = \int \theta P(\theta \mid X) d\theta = \frac{M[1]+\alpha_1}{M+\alpha}$$

- Special case: uniform prior

$$P\left(x[m+1]=1 \mid X\right) = \frac{M[1]+1}{M+2}$$

  – We have *Laplace's correction* for the prediction

How about doing prediction based on the MLE parameter?

Priors: *Dirichlet* Distribution

- *Beta* distribution can only be used for binomial distribution (one independent parameter) with constraint $\theta_1 + \theta_2 = 1$
- *Dirichlet* distribution is an extension for multinomial distribution $\sum_k \theta_k = 1$
- The likelihood function for multinomial

$$L(\theta : \mathcal{D}) = \prod_k \theta_k^{M[k]}$$

- The *Dirichlet* distribution

$$\theta \sim Dirichlet(\alpha_k) \; if \; P(\theta) \propto \prod_k \theta_k^{\alpha_k - 1}$$

Priors: *Dirichlet* Distribution

- *Dirichlet* distribution has good properties
  - It is conjugate
  - Its posterior is simple $\alpha_k^* = M[k] + \alpha_k$
- So, we can quickly get the prediction

$$P(x[M+1] = k) = \frac{M[k]+\alpha_k}{M+\alpha}$$

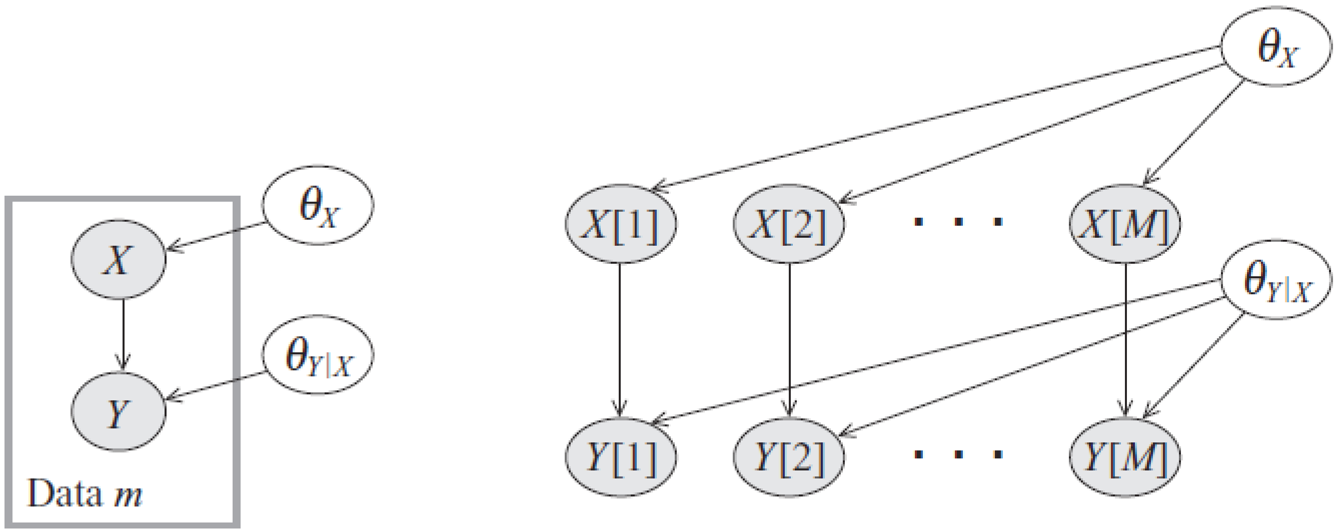
- Let θ_k' is the prior expectation for $x[M+1] = k$. Surely, $\theta_k' = \alpha_k / \alpha$

$$\frac{M[k]+\alpha_k}{M+\alpha} = \frac{\alpha}{M+\alpha}\theta_k' + \frac{M}{M+\alpha}\frac{M[k]}{M}$$

We can treat the posterior as the weighted average between prior and MLE estimation

Bayesian Estimation in BNs

- Meta-network for IID samples:



- So we can do global decomposition, if $P(\theta)$ satisfies the global independence:

$$P(\theta|\mathcal{D}) = \prod_i P\left(\theta_{X_i|Pa_{X_i}}|\mathcal{D}\right)$$

Bayesian Estimation in BNs

- Then we can further do local decomposition

$$\prod_i P\left(\theta_{X_i|Pa_{X_i}}|\mathcal{D}\right) = \prod_i \prod_{pa_{X_i}} P\left(\theta_{X_i|pa_{X_i}}|\mathcal{D}\right)$$

- If we use the *Dirichlet* distribution as prior, the Bayesian prediction should be

$$P\left(X_i[M+1] = x_{ij}|Pa_{X_i}[M+1] = u, \mathcal{D}\right) = \frac{\alpha_{x_{ij}|u} + M[x_{ij}, u]}{\sum_j \alpha_{x_{ij}|u} + M[u]}$$

Bayesian Estimation in BNs

How to choose the parameters in prior?

- *BDe prior*: use another distribution $P'(X)$ to get the priors
  - Usually the same structure as $\mathcal{G}$, but not required
  - First set the total confidence α of the prior, which represent the *pseudo* counts in predictions
    - Set $\alpha_{x_{ij}|pa_{X_i}} = \alpha \cdot P'\left(X_i = x_{ij},\ Pa_i = pa_{X_i}\right)$
    - Standard inferences are needed to calculate the marginal distribution in *P*’

MAP Parameter Estimation

- If we can get a full solution of the Bayesian estimation, MAP estimation is defined as

$$\tilde{\theta} = \arg\max_{\theta} \log P(\theta|\mathcal{D})$$

  - For example, for binomial distribution with Beta prior $\alpha_1 = \alpha_0 = 1$: $P(\theta|\mathcal{D}) \propto \theta^{M[1]}(1-\theta)^{M[0]}$. The MAP estimation is the same as the MLE.
  - *Optimization methods* is usually used for MAP estimation if the full solution is hard to calculate

Maximum a posterior (极大后验)

MAP Parameter Estimation

- When we have a large amount of data, the MAP estimation is mainly affected by training data

$$\tilde{\theta} = \arg\max_{\theta}(\log P(\mathcal{D}|\theta) + \log P(\theta))$$

  - The likelihood function $\log P(\mathcal{D}|\theta)$ will dominate
  - $\log P(\theta)$ can be regarded as *regularization*

- The posterior for prediction is often sharply peaked around its MAP estimation

$$P(X|\mathcal{D}) = \int P(X|\theta)P(\theta|\mathcal{D})d\theta \approx P\left(X|\tilde{\theta}\right)$$

  - The **shape** of Bayesian distribution of $\tilde{\theta}$ is **sharp**

Comments

- Bayesian estimation can control the generalization ability when the training data are not enough relative to the model complexity (many assignments will get zero probability for MLE)
- Using Bayesian estimation to do predictions can be regarded as a model averaging over all possible parameter settings
- When the training data are increasing, the weight of the prior is reducing. The resulting estimation will be mainly determined by the data
- When the number of data is large, the shape of the posterior will be sharply peaked around MAP

The Likelihood of Markov Networks

- For discretized random variables

$$P(X) = \frac{1}{Z}\exp(\sum_i \theta_i f_i(C_i))$$

$$l(\theta : D) = \sum_i \theta_i \sum_m f_i(x_i[m]) - M\ln Z$$

$$\frac{1}{M} l(\theta : D) = \sum_i \theta_i E_D\left(f_i(x_i)\right) - \ln Z$$

- Bad: the second term couples all factors!

$$Z = \sum_{c_i} \exp(\sum_i \theta_i f_i(c_i))$$

The Convexity of Partition Function

- Convexity

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$

- Hessian matrix is semi-positive ⇒ convexity
- Hessian matrix is the matrix of the secondary partial derivatives

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

The First Derivative of *Z*

- The first derivative of Z is the expectation of each feature function

$$
\begin{aligned}
\frac{\partial}{\partial \theta_i} \ln Z(\boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})} \sum_{\xi} \frac{\partial}{\partial \theta_i} \exp\left\{ \sum_j \theta_j f_j(\xi) \right\} \\
&= \frac{1}{Z(\boldsymbol{\theta})} \sum_{\xi} f_i(\xi) \exp\left\{ \sum_j \theta_j f_j(\xi) \right\} \\
&= \mathbb{E}_{\boldsymbol{\theta}}[f_i].
\end{aligned}
$$

The Secondary Derivative of *Z*

$$
\begin{aligned}
\frac{\partial^2}{\partial\theta_j\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \frac{\partial}{\partial\theta_j}\left[\frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\}\right] \\
&= -\frac{1}{Z(\boldsymbol{\theta})^2}\left(\frac{\partial}{\partial\theta_j}Z(\boldsymbol{\theta})\right)\sum_{\xi} f_i(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\} \\
&\quad + \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)f_j(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\} \\
&= -\frac{1}{Z(\boldsymbol{\theta})^2}Z(\boldsymbol{\theta})\boldsymbol{E}_{\boldsymbol{\theta}}[f_j]\sum_{\xi} f_i(\xi)\tilde{P}(\xi:\boldsymbol{\theta}) \\
&\quad + \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)f_j(\xi)\tilde{P}(\xi:\boldsymbol{\theta}) \\
&= -\boldsymbol{E}_{\boldsymbol{\theta}}[f_j]\sum_{\xi} f_i(\xi)P(\xi:\boldsymbol{\theta}) \\
&\quad + \sum_{\xi} f_i(\xi)f_j(\xi)P(\xi:\boldsymbol{\theta}) \\
&= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i f_j] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]\boldsymbol{E}_{\boldsymbol{\theta}}[f_j] \\
&= \boldsymbol{Cov}_{\boldsymbol{\theta}}[f_i; f_j].
\end{aligned}
$$

The secondary derivative is the *covariance matrix* between different factors

The Convexity of Partition Function

- Hessian matrix of the partition function is the covariance matrix of the feature functions
- As we known, *covariance matrix* is always positive semi-definite

- The partition function is convex. So, we can implement efficient gradient ascent algorithm to find the global optimal solution for the likelihood function!

Recall: Gradient Ascent

- Gradient descent is a common strategy to find the optimal parameter setting. Simply update the model along the gradient:
$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \lambda^{(k)} \nabla l(\boldsymbol{\theta}; \boldsymbol{x})$$
- If the likelihood function $l(\theta; x)$ is convex, the gradient ascent will converge to the optimal
- In most cases θ_i is independent with each other, you can update them one by one using partial derivative on θ_i

Stochastic Gradient Ascent

- If the observed samples are *i.i.d.*, the likelihood function are the sum across each sample

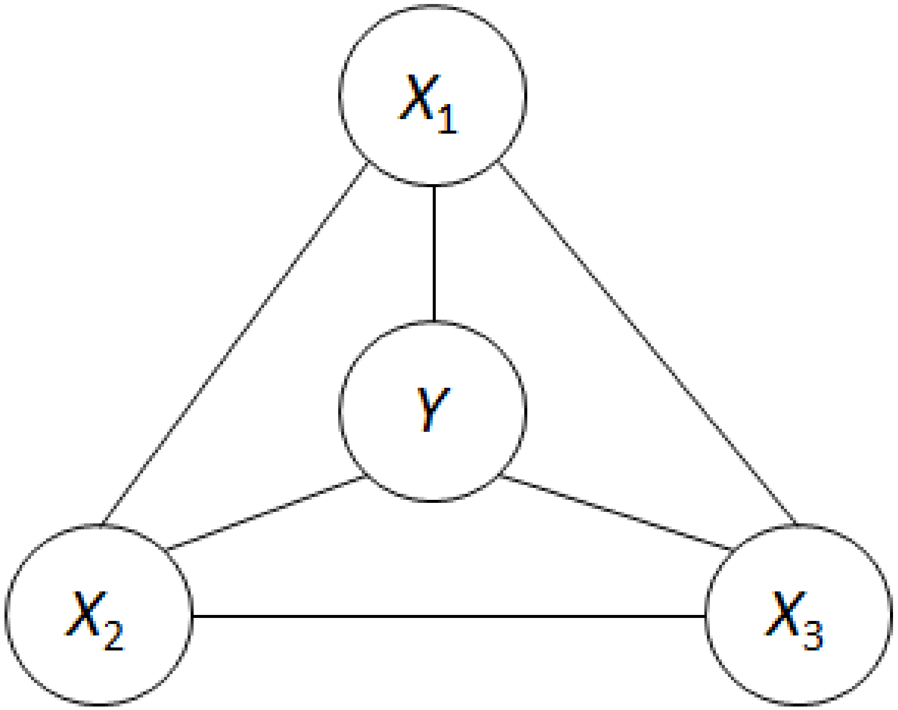
$$l(\boldsymbol{\theta}; \boldsymbol{x}) = \sum_{m=1}^{M} l_m(\boldsymbol{\theta}; \boldsymbol{x}[m])$$

- In many situations, the computational cost is huge for the sum
- Stochastic gradient ascent updates parameters based on one sample each time

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \lambda^{(k)} \nabla l_{m_k}(\boldsymbol{\theta}; \boldsymbol{x}[m_k])$$

- The order of the input samples are randomly given in each run
- The input samples can be used repeatedly (each time with a different random order) until the optimization process converges

- Comments: to avoid local minimal, heuristic methods should be used, such as simulated annealing and genetic algorithm



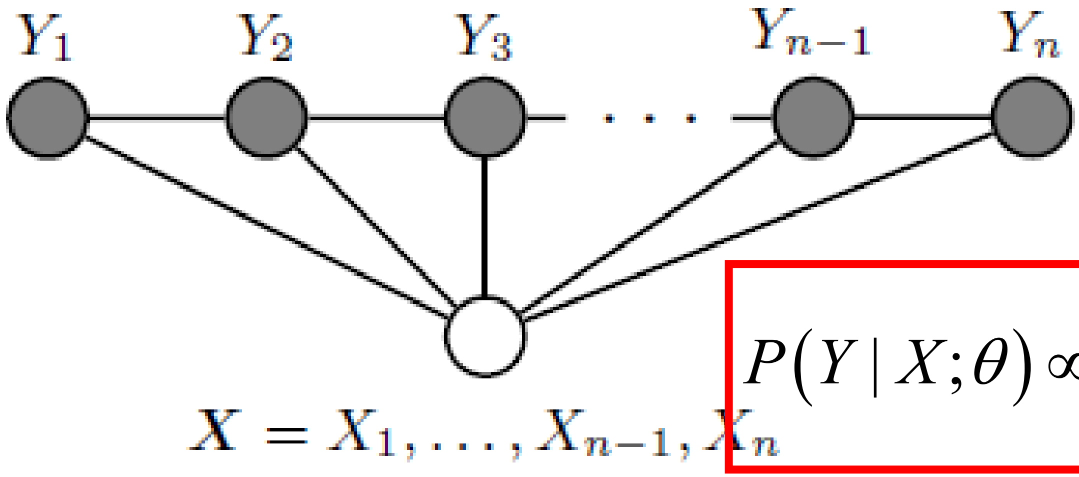
x: $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$

y: β

xx: $w_{12} = w_{13} = w_{23} = w$

xy: $h_{1y} = h_{2y} = h_{3y} = h$

Linear-Chain CRFs (Revisited)



$$\boxed{P(Y \mid X;\theta) \propto \exp\left(\sum_{i=1}^{n}\sum_{m} w_m f_m(Y_{i-1}, Y_i, X, i)\right)}$$

$$P(\bar{y} \mid \bar{x};\theta) = \frac{1}{Z(w,\bar{x})}\exp\left(\sum_{i=1}^{n}\sum_{m} w_m f_m(y_{i-1}, y_i, \bar{x}, i)\right)$$

$$Z(w,\bar{x}) = \sum_{\bar{y}}\exp\left(\sum_{i=1}^{n}\sum_{m} w_m f_m(y_{i-1}, y_i, \bar{x}, i)\right)$$

Stochastic Gradient Ascent for CRFs

- Calculate the partial derivative of each w_m
- Let $F_m(\boldsymbol{x}, \boldsymbol{y}) = \sum_i f_m(y_{i-1}, y_i, \boldsymbol{x}, i)$ (sum across the sequence for the *m*-the feature)
- The derivative of the log likelihood is

$$\frac{\partial}{\partial w_m} \log p = F_m(\bar{x}, \bar{y}) - \sum_{\bar{s}} F_m(\bar{x}, \bar{s}) \, p(\bar{s} \mid \bar{x}; w)$$

First item is the number of that the feature is positive, given the training label sequence

Second item is the expected number of that the feature is positive *across all possible labels*

$$P(\bar{y} \mid \bar{x}; w) = \frac{1}{Z(w, \bar{x})} \exp\left(\sum_m w_m F_m(\bar{x}, \bar{y}) \right)$$

$$Z(w, \bar{x}) = \sum_{\bar{y}} \exp\left(\sum_m w_m F_m(\bar{x}, \bar{y}) \right)$$

Stochastic Gradient Ascent for CRFs

- Update the parameters according to its partial derivative (the same as logistic regression)

$$w_m^{(k+1)} := w_m^{(k)} + \lambda^{(k)} \left(F_m(\bar{x}, \bar{y}) - \sum_{\bar{s}} F_m(\bar{x}, \bar{s})\, p\left(\bar{s} \mid \bar{x}; w^{(k)}\right) \right)$$

- The computational cost is huge for the second item, especially for the training sequence is very long and the scope of the possible labels is very large

The Collins Perceptron

- Given the initial weight $w^{(0)}$, we can calculate the optimal sequence $\hat{y}^{(0)}$
- Then we update w using the formula below

$$w_m^{(k+1)} := w_m^{(k)} + \lambda^{(k)} \left(F_m(\bar{x}, \bar{y}) - F_m(\bar{x}, \hat{y}^*) \right)$$

- Collins. **Discriminative training methods for hidden Markov models: Theory and experiments with the perceptron algorithm**. *EMNLP* 2002.

The End of Chapter 10