Probabilistic Graphical Models

Learning

1 Overview
2 Parameter Learning with Complete Data
  2.1 Learning in Bayesian Network
    2.1.1 Maximum Likelihood Estimation
    2.1.2 Bayesian Parameter Estimation
  2.2 Learning in Markov Network and Conditional Random Fields
    2.2.1 Maximum Likelihood Estimation
    2.2.2 MAP Estimation
3 Structure Learning with Complete Data in Bayesian Network
  3.1 Scoring Functions
    3.1.1 Likelihood Score
    3.1.2 BIC Score
    3.1.3 Bayesian Score
  3.2 Searching over Structures
    3.2.1 Tree/Forest Structure Learning
    3.2.2 General Structure Learning

1 Overview

For now, we assume we know the complete structure and all parameters in the Graphical Model, but it is not always the case. Sometimes we need to find a good model to represent the whole distribution, sometimes we need to estimate the CPDs or factors in order to answer further queries, or sometimes we want to find the most suitable relationship between variables(i.e., finding the structures). So here comes the task of learning.

During learning, we may encounter different scenarios and different tasks:

- Unknow parameter
- Unknow structure
- Incomplete data or latent variables

For different scenarios, we need different strategies.

2 Parameter Learning with Complete Data

2.1 Learning in Bayesian Network

2.1.1 Maximum Likelihood Estimation

In Frequentist's framework, we can maximum the likelihood of generating the observed data to find the optimal parameter. This is the basic idea of Maximum Likelihood Estimation. In MLE framework, in most cases, we do not to keep all data as to maintain the property of the distribution, instead only keeping some crucial statistics suffices. These statistics are called **sufficient statistics**. Formally, a function $s(\boldsymbol{D}) : \text{Instance} \to \mathbb{R}^K$ is a sufficient statistic, if for any two datasets $\boldsymbol{D}$ and $\boldsymbol{D}'$ and any $\theta \in \boldsymbol{\theta}$, we have $\sum_{x[i]\in \boldsymbol{D}} s(x[i]) = \sum_{x[i]\in \boldsymbol{D}'} s(x[i]) \Rightarrow L(\theta; \boldsymbol{D}) = L(\theta; \boldsymbol{D}')$, where L is the likelihood function and $x[i]$ means the ith data in the dataset.

NB. This definition is slightly different from that in *Pattern Recognition and Machine Learning*, which basically define the summation $\sum_{x[i]\in \boldsymbol{D}} s(x[i])$ as the sufficient statistic.

More specifically, in Bayesian networks, if θ_j in $\boldsymbol{\theta}$ are disjoint,

$$
\begin{aligned}
L(\boldsymbol{\theta}; \boldsymbol{D}) &= \prod_i P(\boldsymbol{x}[i]; \boldsymbol{\theta}) \\
&= \prod_i \prod_j P(x_j[i] | Par_{x_j}[i]; \theta_j) \\
&= \prod_j \prod_i P(x_j[i] | Par_{x_j}[i]; \theta_j) \\
&= \prod_j L_i(D; \theta_j)
\end{aligned}
$$

This equation means we can optimize each parameter by local likelihood separately. That means we do not need to do global computation, but we only need to consider a part of the network (a CPD) each time and optimize it.

If some parameters are shared for multiple variables, e.g., variables in HMM, we just accumulate over all those variables to calculate sufficient statistics.

2.1.2 Bayesian Parameter Estimation

In Bayesian framework, we treat parameters also as random variables, following some prior distribution whose parameters we already know. So the learning task is turned into an inference task with complete parameters, i.e., we want to answer the query $P(\boldsymbol{\theta}|\boldsymbol{X})$. And when predicting, what we want to find is the probability $P(\boldsymbol{x}'|\boldsymbol{X}) = \int P(\boldsymbol{x}'|\boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{X})\mathrm{d}\boldsymbol{\theta}$.

More specifically, in Bayesian Network, if the prior distributions of 2 parameters are independent, then the posterior distributions are also independent given complete observed data. So we can solve the posteriors separately. Just like what we can do in MLE.

2.2 Learning in Markov Network and Conditional Random Fields

In Bayesian Network, we can optimize CPDs separately, but in Markov Network and CRFs, it is unfortunately not the case. To illustrate learning in Markov Network, we consider Log-Linear Model.

2.2.1 Maximum Likelihood Estimation

In Log-Linear Model, the probability of generating one sample X is as follows:

$$P(X;\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\exp\sum_{i=1}^{k}\theta_i f_i(\boldsymbol{D}_i)$$

$\boldsymbol{D}_i$ is the scope for $f_i(\,.\,)$, Z is the partition function.

So the log-likelihood is:

$$\ell(\boldsymbol{\theta};\mathcal{D}) = \sum_i \theta_i(\sum_{m=1}^{M} f_i(\boldsymbol{D}_i[m]) - M\ln Z(\boldsymbol{\theta})$$

, where $\boldsymbol{D}_i[m]$ means the variables in ith scope in the mth data point, and $Z(\boldsymbol{\theta}) = \ln\sum_{\boldsymbol{x}}\exp\Big\{\sum_i \theta_i f_i(\boldsymbol{D}_i)\Big\}$ (the 1st summation is over all possible data $\boldsymbol{x}$, the second summation sums over all features, and $\boldsymbol{D}_i$ means the value of the variables in the ith scope in $\boldsymbol{x}$).

Because of the partition function, we can not optimize each features separately, and we do not have analytical solution. So we need to consider gradient.

We can prove that the element in Jacobian Matrix and Hessian Matrix of Z is as follows:

$$\frac{\partial}{\partial \theta_i} \ln Z(\boldsymbol{\theta}) = \mathbb{E}[f_i(\boldsymbol{D}_i)]$$

$$\frac{\partial^2}{\partial \theta_i \partial \theta_j} \ln Z(\boldsymbol{\theta}) = \mathrm{Cov}[f_i(\boldsymbol{D}_i), f_j(\boldsymbol{D}_j)]$$

, the expectation is taken with respect to all possible $\boldsymbol{x}$, and $\boldsymbol{D}_i$ means the value of the variables in the ith scope in $\boldsymbol{x}$.

And we can find that $\nabla\nabla\ell = -\nabla\nabla \ln Z$, so the negative log-likelihood is positive defined, which means that we can always find the global maximum by Gradient Descent or Newton method or quasi-Newton method.

The element in gradient is:

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \ell(\boldsymbol{\theta}; \mathcal{D}) = \mathbb{E}_{\mathcal{D}}[f_i(\boldsymbol{D}_i)] - \mathbb{E}[f_i(\boldsymbol{D}_i)]$$

, where the first term is the empirical mean of the data set. So we can conclude that $\hat{\boldsymbol{\theta}}$ is MLE if and only if $\mathbb{E}_{\mathcal{D}}[f_i(\boldsymbol{D}_i)] = \mathbb{E}[f_i(\boldsymbol{D}_i)]$.

The first term is rather easy to calculate: we just need to sum over the value of this feature for all data, but the second can be expensive, because we need to solve $\mathbb{E}[f_i(\boldsymbol{D}_i)] = \sum_{\boldsymbol{x}} f_i(\boldsymbol{D}_i) P(\boldsymbol{x})$, which requires us to do inference. **NB**. $P(\boldsymbol{x})$ in the second term implicitly contains the value of our parameter.

And now let us consider CRFs, in CRFs,

$$P(\boldsymbol{Y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \tilde{P}(\boldsymbol{x}, \boldsymbol{Y})$$

$Z(\boldsymbol{x})$ is a function of $\boldsymbol{x}$, it sums over all possible $\boldsymbol{y}$ when $\boldsymbol{X}$ takes the value $\boldsymbol{x}$.

So the log-likelihood is

$$\ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta}; \mathcal{D}) = \sum_i \theta_i (\sum_{m=1}^{M} f_i(\boldsymbol{D}_i[m]) - M \ln Z_{\boldsymbol{X}[m]}(\boldsymbol{\theta})$$

, where $Z_{\boldsymbol{X}[m]}(\boldsymbol{\theta})$ is just parametric $Z(\boldsymbol{x})$ when $\boldsymbol{x}$ takes the value of $\boldsymbol{x}$ in the mth data.

And similarly, the derivative is

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{M} \sum_{m=1}^{M} \Big\{ f_i(\boldsymbol{D}_i[m]) - \mathbb{E}[f_i(\boldsymbol{D}_i, \boldsymbol{X} = \boldsymbol{x}[m])] \Big\}$$

, the last term means sum over all possible values of variables but $\boldsymbol{X}$ in scope $\boldsymbol{D}_i$, and $\boldsymbol{X}$ only take the value of $\boldsymbol{x}$ in the mth data.

This means that at each step, we not only need to do one inference, rather for each value of $\boldsymbol{X}$ in the dataset, we need to do one inference.

To be more specific, let's consider probabilistic generation model and probabilistic discrimination model. A probabilistic generation model is equivalent to a Bayesian Network, so we can optimize MLE for distributions over features and target given features separately. While a probabilistic discrimination model is equivalent to a CRF, so at each step we need to recalculate the predicted target for each input data.

2.2.2 MAP Estimation

In Log-Linear Model of MRF and CRF, we can assign prior distribution to parameters. Because of the complexity and intractability of the posterior distribution, instead of finding $P(\boldsymbol{\theta}|\boldsymbol{X})$and solve $P(\boldsymbol{x}'|\boldsymbol{X}) = \int P(\boldsymbol{x}'|\boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{X})\mathrm{d}\boldsymbol{\theta}$ as in BN, we turn to do MAP estimation, and we can use optimization method to solve it.

And it turns out that if we assign a Gaussian prior, the MAP process is equivalent to minimize loss function with L2 regularization, while if a Laplacian prior, that is equivalent to L1 regularization.

3 Structure Learning with Complete Data in Bayesian Network

We preciously discussed the problem of parameters learning of graphical model whose structure is known to us. But it is not always the case, we sometimes need structure learning to learning model for new queries, or just for structure discovery. Here we only discuss the structure learning task in Bayesian Network.

If a structure misses an edge, which means it contains incorrect independencies (the structure is not an I-map for the real distribution, because $I(G) \not\subset I(P)$), then the correct distribution P cannot be learned, but it can generalize better. While if a structure has an extra edge(the structure is an I-map for the real distribution, because $I(G) \subset I(P)$), it can correctly learn the real distribution, but as there are more parameters, its generalization performance will be worse.

To find the optimal structure, we need to define a scoring function for each structure, then choose the structure which can maximize the score as the optimal one. **This is the basic idea of structure learning.**

3.1 Scoring Functions

We first talk about commonly used scoring functions.

3.1.1 Likelihood Score

The simplest score is Likelihood Score:

$$\mathrm{score}_L(\mathcal{G};\mathcal{D}) = \ell(\mathcal{G};\mathcal{D})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}$$

$\mathcal{G}$ is the structure, $\mathcal{D}$ is data, ℓ is log-likelihood, $\boldsymbol{\theta}$ is the set of parameters, and $\boldsymbol{\theta}^*$ is the MLE parameters in this certain structure $\mathcal{G}$.

This means for each structure, we perform parameter learning, and take the maximum log-likelihood as the score for this structure.

But adding edges can almost always improve the score, so likelihood score tends to overfit. In practice, we can restrict the hypothesis space(restrict the number of parents or the number of parameters), or adding penalty term.

3.1.2 BIC Score

$$\text{score}_{BIC}(\mathcal{G};\mathcal{D}) = \ell(\mathcal{G};\mathcal{D})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} - \frac{\log M}{2}\text{Dim}[\mathcal{G}]$$

M is the number of training samples and $\text{Dim}[\mathcal{G}]$ is the number of independent parameters.

BIC score is **asymptotically consistent**, which means that if data generated by G^*(the ground truth), networks I-equivalent to G^* will have the highest score if the sample size M goes to infinity.

3.1.3 Bayesian Score

$$P(\mathcal{G}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{G})P(\mathcal{G})}{P(\mathcal{D})} \propto P(\mathcal{D}|\mathcal{G})P(\mathcal{G})$$

So we can define:

$$\text{score}_B(\mathcal{G};\mathcal{D}) = \log P(\mathcal{D}|\mathcal{G}) + \log P(\mathcal{G})$$

, where $P(\mathcal{D}|\mathcal{G}) = \int P(\mathcal{D}|\mathcal{G},\boldsymbol{\theta}_\mathcal{G})P(\boldsymbol{\theta}_\mathcal{G}|\mathcal{G})\text{d}\boldsymbol{\theta}_\mathcal{G}$.

As for the structure priors $P(\mathcal{G})$, we can choose:

- uniform prior;
- prior penalizing number of edges: $P(\mathcal{G}) \propto c^{|G|}$, where G is the number of edges and $c \in (0,1)$;
- prior penalizing number of parameters.

As for the parameter prior $P(\boldsymbol{\theta}_\mathcal{G}|\mathcal{G})$, a commonly used one is BDe prior. It is defined by a prior network B_0 representing prior probability of events and an α representing equivalent sample size.

And the hyperparameter for one node $X_i = x_i$ and its parent $Par_i^\mathcal{G} = pa_i^\mathcal{G}$(written as $\alpha(x_i, pa_i^\mathcal{G})$), is calculated by the total hyperparameter α multiplied by the probability that observing $X_i = x_i$ and $Par_i^\mathcal{G} = pa_i^\mathcal{G}$ in the prior network B_0. **NB.** $Par_i^\mathcal{G}$ are not necessarily the same as parents of X_i in B_0, actually we always define a network without any edge as B_0 and use it as the prior for all $\mathcal{G}$.

By using BDe score, we can get the same Bayesian Score for all I-equivalent networks.

And another property of BDe score is as M goes to infinity, a network with Dirichlet priors satisfies

$$\log P(\mathcal{D}|\mathcal{G}) = \ell(\mathcal{G};\mathcal{D})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} - \frac{\log M}{2}\text{Dim}[\mathcal{G}] + O(1)$$

This is the form of BIC score.

3.2 Searching over Structures

Having defined some commonly use score, we now discuss how we do searching over structures. First we consider the case that the network is a tree/forest.

3.2.1 Tree/Forest Structure Learning

A tree structure is elegant in math, can be optimized efficiently, and can provide a sparse parameterization, which reduce the overfitting. The most import property that makes the learning feasible is decomposability of the score:

$$\text{score}(\mathcal{G};\mathcal{D}) = \sum_i \text{score}(X_i|\boldsymbol{Par}^{\mathcal{G}}_{X_i};\mathcal{D})$$

, where $\boldsymbol{Par}^{\mathcal{G}}_{X_i}$ represents the parents of X_i in Graph $\mathcal{G}$and will be 0 if X_i has no parent. This means that we can break down the total score into score for each edge.

Further decomposition(Notice: In a tree/forest, $\boldsymbol{Par}^{\mathcal{G}}_{X_i}$ is actually a scalar because each node can only have at most one parent):

$$\begin{aligned}
\text{score}(\mathcal{G};\mathcal{D}) &= \sum_i \text{score}(X_i|Par^{\mathcal{G}}_{X_i};\mathcal{D}) \\
&= \sum_{i:Par^{\mathcal{G}}_{X_i}\neq 0} \text{score}(X_i|Par^{\mathcal{G}}_{X_i};\mathcal{D}) + \sum_{i:Par^{\mathcal{G}}_{X_i}= 0} \text{score}(X_i;\mathcal{D}) \\
&= \sum_{i:Par^{\mathcal{G}}_{X_i}\neq 0} \Big\{\text{score}(X_i|Par^{\mathcal{G}}_{X_i};\mathcal{D}) - \text{score}(X_i;\mathcal{D})\Big\} + \sum_i \text{score}(X_i;\mathcal{D})
\end{aligned}$$

The last term is constant for all structure, it is the score for "empty" network. And the 1st term is the sum of edges scores, i,e., the "improvement" after adding each edge.

And if we use likelihood score, all edges' scores are nonnegative and thus the optimal structure is always a tree; while if using BIC or Bayesian Score with BDe prior, some edge can have negative score and thus the optimal structure can be a forest.

Another useful property is that if likelihood/BIC/Bayesian Score with BDe prior is used, and because in a tree there is no V-structure, so the direction of the edge does not matter(will lead to I-equivalent network). So it allows to optimize the structure efficiently using undirect graph.

Concretely, we define the weight for an edge between node i and j as :

$$w(i,j) = max\Big\{\text{score}(X_i|Par^{\mathcal{G}}_{X_i};\mathcal{D}) - \text{score}(X_i;\mathcal{D}), 0\Big\}$$

And we can use standard algorithms(e.g., Prim's or Kruskal's) to find max-weight spanning tree in $O(n^2)$ time, then remove all edges of weight 0 to get a forest as the final optimal structure.

3.2.2 General Structure Learning

If we release the restriction on the graph, then the problem turns into an NP-hard problem, i.e., finding maximal scoring network structure with at most k parents for each node is NP-hard for any $k > 1$. So we need heuristic searching.

What we basically do is we start at a network. It can be am empty network, or the best tree, or just a random network, or a network by prior knowledge. And then we iteratively optimal the structure. At each iteration, we try some operators: edge addition, deletion, and reversal; or some global steps which can make more significant difference, and apply change that most improves the score, until there is no change that can improve the score.

Using such **Greedy Hill Climbing** Algorithm, we may encounter some pitfalls:

- Local Maxima
- Plateaux: Typically because I-equivalent networks are often neighbors in the search space.

To partially address the problem, we can do random restart, which means when we get stuck, take some number of random steps and then start climbing again; or we can build a tabu list, which means make a list pf K steps most recently taken, and let searching not reverse any of these steps. For example, after adding an edge from A to B, in K steps, we can not removing this edge even it can lead to higher score.

Let's consider the complexity of this algorithm at each iteration:

In the most naïve case, we need to test all operators for all possible edges($n(n-1)/2$ for a network with n nodes), the time complexity is $O(n^2)$. And for each operator of each edge, we need to calculate the score, which need to sum over $O(n)$ components and for each component, we need to calculate sufficient statistics which need time of $O(M)$. And after each operator, we need to check the acyclicity which takes $O(m)$ where m is the number of edges in current structure. So in summary the time complexity per iteration is $O(n^2(Mn+m))$.

But in practice, we can do it more clearly. At each iteration, we only need to calculate the Δscore for those changed edges, because others have no change in score.

First we need to choose which edge to be changed.

We can build an priority queue to store the operators for each edge, sorted by its Δscore. At each iteration, we just pick the first one to conduct operation.

After conducting the operation, we need to maintain our priority queue. After perform an operation, only one or two nodes are affected. All edges linked to these one or two edges are affected. In total, there are $O(n)$ edges. We need to recalculate $O(n)$ Δscore. For each one we need $O(M)$ time to calculate the sufficient statistics. After that, we need to insert these values into our priority queue, each takes time of $O(\log n)$, so totally we need time of $O(n \log n)$ to insert all values.

So totally, at each iteration, the time complexity is $O(nM + n\log n)$.

And there are methods to make calculation even more efficient, e.g., reuse sufficient statistics, restrict the set of operations considered in the search, etc.