6

Learning

We now turn our attention to the third and last part of this tutorial: *learning*. Given a dataset, we would like to fit a model that will make useful predictions on various tasks that we care about.

A graphical model has two components: (1) the graph structure and (2) the parameters of the factors induced by this graph. These components lead to two different learning tasks:

1. *Structure learning*, where we want to estimate the graph (i.e., determine from data how the variables depend on each other).

2. *Parameter learning*, where the graph structure is known and we want to estimate the factors.

The chapter begins in Section 6.1 with a basic review of learning theory. In Section 6.2 we address parameter learning in directed models, while Section 6.3 explores the undirected case. It turns out that the former will admit easy closed form solutions, while the latter will involve potentially intractable numerical optimization techniques. In Section 6.4, we consider latent variable models, which contain unobserved hidden variables that succinctly model the event of interest. Section 6.5 concludes the chapter with an overview of Bayesian learning, a general

statistical framework that offers advantages over more-conventional approaches.

6.1 Learning Theory Basics

6.1.1 The Learning Task

Before, we start our discussion of learning, let's first reflect on what it means to fit a model and what is a desirable objective for this task.

Let's assume that the domain is governed by some underlying distribution p^*. We are given a dataset $\mathcal{D}$ of N samples from p^*. The standard assumption is that the data instances are independent and identically distributed (iid). We are also given a family of models $\mathcal{M}$, and our task is to learn some "good" model in $\mathcal{M}$ defined by a distribution p. For example, a family $\mathcal{M}$ might consist of all Bayesian networks with a given graph structure and all possible choices of the corresponding conditional probability distribution tables.

The goal of learning is to return a model that accurately captures the distribution p^* from which our dataset $\mathcal{D}$ was sampled. Note that this is not achievable in general because limited data may only provide a rough approximation of the true underlying distribution. Still, we want to somehow select the "best" approximation to the underlying distribution p^*.

What is "best" in this case? It depends on what we want to do. Tasks of interest could include:

- *Density estimation.* We are interested in the full distribution (so later we can compute whatever conditional probabilities we want).
- *Specific prediction tasks.* We are using the distribution to make a prediction (e.g., is this email spam or not?).
- *Structure or knowledge discovery.* We are interested in the model itself (e.g., how do some genes interact with each other?).

Maximum Likelihood

Let's assume that we want to learn the full distribution so that later we can answer any probabilistic inference query. In this setting we can

view the learning problem as *density estimation*. We want to construct a p as "close" as possible to p^*. How do we evaluate "closeness"? We will again use the KL divergence (Definition 5.1), which we explored in our discussion of variational inference (Section 5.2). In particular, we aim to find a p that minimizes

$$\mathrm{KL}(p^* \,\|\, p) = \sum_{\mathbf{x}} p^*(\mathbf{x}) \log \frac{p^*(\mathbf{x})}{p(\mathbf{x})} = -H(p^*) - \mathbb{E}_{\mathbf{x}\sim p^*}[\log p(\mathbf{x})].$$

The first term does not depend on p; hence, minimizing the KL divergence is equivalent to maximizing the expected log-likelihood

$$\mathbb{E}_{\mathbf{x}\sim p^*}[\log p(\mathbf{x})].$$

This objective asks that p assign high probability to instances sampled from p^*, so as to reflect the true distribution. Although we can now compare models, since we are not computing $H(p^*)$, we don't know how close we are to the optimum.

However, there is still a problem: in general we do not know p^*. We will thus approximate the expected log-likelihood $\mathbb{E}_{\mathbf{x}\sim p^*}[\log p(\mathbf{x})]$ with the empirical log-likelihood (a Monte Carlo estimate):

$$\mathbb{E}_{\mathbf{x}\sim p^*}[\log p(\mathbf{x})] \approx \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}\in\mathcal{D}} \log p(\mathbf{x}),$$

where $\mathcal{D}$ is a dataset drawn iid from p^*. Maximum likelihood learning is then defined as producing the estimate

$$\hat{p} = \operatorname*{argmax}_{p\in\mathcal{M}} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}\in\mathcal{D}} \log p(\mathbf{x}).$$

A Step-by-Step Proof with the Empirical Distribution

How do we know that the estimate defined above will perform maximum likelihood learning? In particular, we'd like this estimate to maximize the expected likelihood (or log-likelihood). Here we will give a step-by-step proof using the notion of an *empirical distribution*.

The Monte Carlo samples provide empirical samples from the true data distribution. Suppose we define a distribution using the N samples from $\mathcal{D}$, called the empirical distribution $\tilde{p}$. This empirical distribution

is defined such that if we find some new data point $\mathbf{x}$, its probability is proportional to the number of times it is found in the training data set $\mathcal{D}$, which may be zero times:

$$\tilde{p}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x}, \bar{\mathbf{x}}_i),$$

where $\bar{\mathbf{x}}_i \in \mathcal{D}$ denotes a sample in our dataset, and δ is a function that evaluates to 1 if its two arguments are the same and 0 otherwise (similar to the Kronecker delta).

Now suppose we have a parameterized family of distributions $p(\mathbf{x};\theta)$, specified by a parameter $\theta \in \Theta$. The KL Divergence between the density defined by the empirical distribution $\tilde{p}$ and a distribution $p(\mathbf{x};\theta)$ from our model family is

$$\begin{aligned}\mathrm{KL}(\tilde{p}(\mathbf{x}) \,||\, p(\mathbf{x};\theta)) &= \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log \frac{\tilde{p}(\mathbf{x})}{p(\mathbf{x};\theta)} \\ &= \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log \tilde{p}(\mathbf{x}) - \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log p(\mathbf{x};\theta).\end{aligned}$$

We want to *minimize this divergence* over $\theta \in \Theta$. We ignore the left term (equivalent to $-H(\tilde{p})$, as shown earlier) because it does not depend on our model parameters θ. Our minimization problem then becomes

$$\operatorname*{argmin}_{\theta\in\Theta} \mathrm{KL}(\tilde{p}(\mathbf{x}) \,||\, p(\mathbf{x};\theta)) = \operatorname*{argmin}_{\theta\in\Theta} -\sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log p(\mathbf{x};\theta).$$

We can then convert the problem to a maximization problem by swapping the sign:

$$\operatorname*{argmax}_{\theta\in\Theta} -\mathrm{KL}(\tilde{p}(\mathbf{x}) \,||\, p(\mathbf{x};\theta)) = \operatorname*{argmax}_{\theta\in\Theta} \sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \log p(\mathbf{x};\theta).$$

Plugging in our definition of the empirical distribution $\tilde{p}$, we see that this then becomes

$$\operatorname*{argmax}_{\theta\in\Theta} \sum_{\mathbf{x}} \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{x}, \bar{\mathbf{x}}_i) \log p(\mathbf{x};\theta).$$

Swapping the order of two finite sums is always legal, and the sum over $\mathbf{x}$ only provides non-zero components when $\mathbf{x} = \bar{\mathbf{x}}_i$. The optimization problem thus reduces to maximizing the expected log-likelihood:

$$\operatorname*{argmax}_{\theta\in\Theta} \frac{1}{N}\sum_{i=1}^{N} \log p(\bar{\mathbf{x}}_i;\theta).$$

An Illustrative Example As a simple example, consider estimating the outcome of a biased coin. Our dataset is a set of tosses and our task is to estimate the probability of heads (h) or tails (t) on the next flip. We assume that the process is controlled by a probability distribution $p^*(x)$ where $x \in \{h, t\}$. Our class of models $\mathcal{M}$ is going to be the set of all probability distributions over $\{h, t\}$.

How should we choose p from $\mathcal{M}$ if 60 out of 100 tosses are heads? Let's assume that $p(x = h) = \theta$ and $p(x = t) = 1 - \theta$. If our observed data are $\mathcal{D} = \{h, h, t, h, t\}$, our likelihood becomes

$$\prod_i p(x_i) = \theta \cdot \theta \cdot (1 - \theta) \cdot \theta \cdot (1 - \theta)$$

and maximizing this yields $\theta = 60\%$. More generally, our log-likelihood function is simply

$$\ell(\theta) = \#\text{ heads} \cdot \log(\theta) + \#\text{ tails} \cdot \log(1 - \theta),$$

for which the optimal solution is

$$\hat{\theta} = \frac{\#\text{ heads}}{\#\text{ heads} + \#\text{ tails}}.$$

6.1.2 Likelihood, Loss, and Risk

We may now generalize this by introducing the concept of a *loss function*. A loss function $L(\mathbf{x}, p)$ measures the loss that a model distribution p makes on a particular instance $\mathbf{x}$. Assuming instances are sampled from some distribution p^*, our goal is to find the model that minimizes the *expected loss* or *risk*,

$$\mathbb{E}_{\mathbf{x} \sim p^*}[L(\mathbf{x}, p)] \approx \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} L(\mathbf{x}, p).$$

Notice that the loss function which corresponds to maximum likelihood estimation is the log loss, $L(\mathbf{x}, p) = -\log p(\mathbf{x})$.

Another example of a loss is the conditional log-likelihood. Suppose we want to predict a set of variables $\mathbf{y}$ given $\mathbf{x}$ (e.g., for segmentation or stereo vision). We concentrate on predicting $p(\mathbf{y} \mid \mathbf{x})$, and use a conditional loss function $L(\mathbf{x}, \mathbf{y}, p) = -\log p(\mathbf{y} \mid \mathbf{x})$. Since the loss

function only depends on $p(\mathbf{y} \mid \mathbf{x})$, it suffices to estimate the conditional distribution, not the joint. This is the objective function we use to train CRFs (Section 3.2.3).

Suppose next that our ultimate goal is structured prediction: given $\mathbf{x}$ we predict $\mathbf{y}$ via $\operatorname{argmax}_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$. What loss function should we use to measure error in this setting? One reasonable choice would be the classification error:

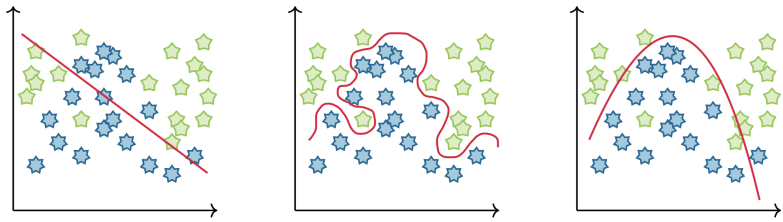
$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p^*}[\mathbb{I}\{\exists\, \mathbf{y}' \neq \mathbf{y} : p(\mathbf{y}' \mid \mathbf{x}) \geq p(\mathbf{y} \mid \mathbf{x})\}].$$

This is the probability that we predict the wrong assignment over all $(\mathbf{x}, \mathbf{y})$ pairs sampled from p^*. A somewhat better choice might be the Hamming loss, which counts the number of variables in which the MAP assignment differs from the ground truth label. There also exists a fascinating line of work on generalizations of the hinge loss to CRFs, which leads to a class of models called *structured support vector machines* (Tsochantaridis *et al.*, 2004; Tsochantaridis *et al.*, 2005; Finley and Joachims, 2008).

The moral of the story here is that it often makes sense to choose a loss that is appropriate to the task at hand (e.g., prediction rather than full density estimation).

6.1.3 Empirical Risk and Overfitting

Empirical risk minimization can easily *overfit* the observed data, overshooting our goal of learning the true generating distribution and instead learning spurious patterns in our random sample. Conversely, we do not want to *underfit* with an overly simplistic model that is not sufficiently informative. Thus, one of the main challenges of machine learning is to choose a model that is sufficiently rich to be useful, yet not so complex as to overfit the training set (Figure 6.1). In tackling this challenge, we must grapple with the fact that the data that we have access to is only a finite sample: there is generally a vast amount of samples that we have never seen, and we cannot generally verify the extent to which our sample is representative of the population. Nevertheless, we aim for some degree of *generalizability*: ideally, a machine learning model demonstrates robustness by generalizing well to such "never-seen" sam-



(a) UNDERFIT **(b)** OVERFIT **(c)** GOOD FIT

Figure 6.1: Underfit (**a**), overfit (**b**), and well fit (**c**) decision boundaries in a toy classification problem. The underfit linear model displays large bias, corresponding to high error rates at both training and test time. The overfit model is prone to high variance as it learns spurious patterns in the training data, resulting in low training error but high test error. A good fit will display greater robustness, offering relatively low training and test error.

ples. In this subsection, we briefly cover problems related to overfitting and generalization.

Generalization Error At training, we minimize the empirical loss

$$\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}).$$

However, we are actually interested in minimizing

$$\mathbb{E}_{\mathbf{x} \sim p^*} [\log p(\mathbf{x})].$$

We cannot guarantee with certainty the quality of our learned model. This is because the data $\mathcal{D}$ is sampled stochastically from p^*, and we might get an unlucky sample. The goal of learning theory is to prove that the model is approximately correct: for most $\mathcal{D}$, the learning procedure returns a model whose error is low. There exists a large literature that quantifies the probability of observing a given error between the empirical and the expected loss under a particular type of model and a particular dataset size.

The Bias-Variance Tradeoff In machine learning, we want to learn salient patterns in the observed training data while also generalizing well to unseen examples. These two objectives are often seen to compete,

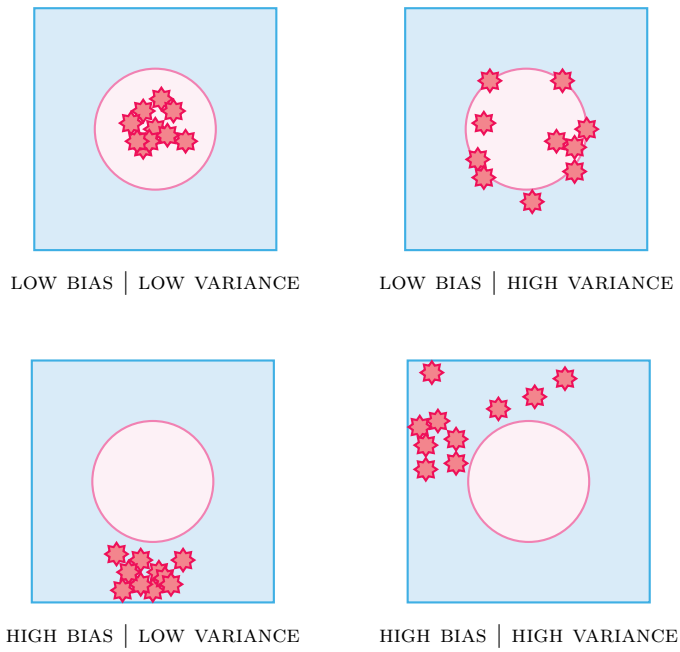


Figure 6.2: Samples with low to high bias and variance when the true distribution is centered on the pink circle.

resulting in the so-called *bias-variance tradeoff*. Note that model error can be decomposed into *bias*, *variance*, and *noise*, defined as follows:

- *Bias*: We typically restrict the hypothesis space of distributions that we search over. If the hypothesis space is very limited, it might not be able to represent the target distribution p^*, even with unlimited data. The assumptions encoded in our hypothesis space "bias" the model toward certain solutions (e.g., linear models assume linearity). Bias is a measure of how well our model learns p^*, and is an inherent error that cannot be resolved by increasing data size. Averaging performance across different training sets, an *unbiased* model will accurately learn the target distribution.

- *Variance*: If we select a highly expressive hypothesis class, we might represent the data better. Variance is a measure of how much the model fluctuates (or *varies*) when trained on different samples $\mathcal{D}$.

When our model has high variance, small perturbations on $\mathcal{D}$ can result in very different estimates. Models that generalize well to unseen data will closely approximate p^* with low variance across different training sets.

- *Noise*: This component of model error captures how much random measurement noise is present in the data sample.

See Figure 6.2 for an illustration of bias and variance. There is a general notion that larger hypothesis classes come with higher model variance but lower bias, as a larger hypothesis space is more likely to contain hypotheses close to or equal to the true distribution p^*. Thus, there is an inherent bias-variance tradeoff when selecting the hypothesis class.

High bias can be avoided by increasing the capacity of the model. We can avoid high variance using several approaches. We can impose hard constraints, such as selecting a less expressive hypothesis class: e.g., Bayesian networks with at most d parents, or pairwise (rather than arbitrary-order) MRFs. In general, more complex models are more prone to overfitting, while simpler models are more prone to underfitting. We can introduce a soft preference for "simpler" models by adding a regularization term $R(p)$ to the loss $L(\mathbf{x}, p)$, which penalizes overly complex p.

6.2 Learning in Directed Models

6.2.1 Maximum Likelihood Learning in Bayesian Networks

Let us now apply this discussion to a particular problem of interest: parameter learning in Bayesian networks. Suppose that we are given a Bayesian network with n variables,

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i \mid \mathbf{pa}(x_i); \theta_i).$$

This is the same formulation of a Bayesian network as given in Definition 3.2, but we have explicitly shown the dependence on model parameters θ_i in each factor. Recall that in the case of a discrete variables, each factor represents a vector or table of probabilities.

Suppose also that we have a set of iid samples $\mathcal{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}\}$. We wish to know the maximum likelihood estimate of the parameters — that is, to learn the conditional probability distributions. We express the likelihood as

$$L(\theta; \mathcal{D}) = \prod_{j=1}^{m} \prod_{i=1}^{n} p(x_i^{(j)} \mid \mathbf{pa}(x_i^{(j)}); \theta_i).$$

Taking logs and combining like terms, in the case of discrete variables, this becomes

$$\log L(\theta; \mathcal{D}) = \sum_{i=1}^{n} \sum_{\mathbf{pa}(x_i)} \sum_{x_i} \#(x_i, \mathbf{pa}(x_i)) \cdot \log p(x_i \mid \mathbf{pa}(x_i); \theta_i)$$

where $\#(x_i, \mathbf{pa}(x_i))$ denotes the number of times we see the joint assignment of a given x_i and $\mathbf{pa}(x_i)$ in the data $\mathcal{D}$. Thus, maximization of the (log) likelihood function decomposes into separate maximizations for the local conditional distributions! This is essentially the same as the head/tails example that we saw earlier (except with more categories). It is a simple calculus exercise to formally show that the maximum likelihood estimate for each factor is

$$\hat{\theta}_i(x_i \mid \mathbf{pa}(x_i)) = \frac{\#\left(x_i, \mathbf{pa}(x_i)\right)}{\#\left(\mathbf{pa}(x_i)\right)},$$

where $\hat{\theta}_i(x_i \mid \mathbf{pa}(x_i))$ denotes the estimate of a table of conditional probabilities for values x_i given parents $\mathbf{pa}(x_i)$. We can therefore draw the convenient conclusion that the maximum likelihood estimate has a closed-form solution for Bayesian networks with discrete variables. Even when the variables are not discrete, the task is nearly as simple: the log-factors are linearly separable, and hence the log-likelihood reduces to estimating each of them separately. The simplicity of learning is one of the most convenient features of Bayesian networks.

6.2.2 Structure Learning for Bayesian Networks

We now review the task of structure learning from observational data. As the search space of DAGs grows super-exponentially with respect to the cardinality of the node set, exhaustive search for the correct

TOTAL NODES	TOTAL POSSIBLE DAGS	POWER OF 10
1	1	
2	3	
3	25	
4	543	$> 10^2$
5	29281	$> 10^4$
6	3781503	$> 10^6$
7	1138779265	$> 10^9$
8	783702329343	$> 10^{11}$
8	1213442454842881	$> 10^{15}$
10	4175098976430598143	$> 10^{18}$

Table 6.1: The total number of possible DAGs with n labeled nodes grows super-exponentially with respect to n. When $n = 10$, this number already exceeds 4 quintillion (Sloane and Plouffe, 1995; Sloane, 2003).

DAG is intractable in the general case (Table 6.1). Thus, structure learning typically entails some form of heuristic search. Even still, DAG structure learning is NP-hard under various problem formulations (Chickering, 1996; Chickering *et al.*, 2004). Constraining the search space can potentially enable polynomial-time solutions (e.g., imposing sparsity constraints that limit the number of parents per node; Claassen *et al.* 2013), though such constraints are sometimes unrealistic for real-world data. Consequently, several popular and influential structure learning algorithms still display worst-case exponential time complexity.

There are two dominant paradigms for structure learning in DAGs: *score-based* methods (Huang *et al.*, 2018) and *constraint-based* methods (Spirtes *et al.*, 2001). Structure learning can also entail *hybrid* methods that draw from both traditions (Ogarrio *et al.*, 2016). Still other methods do not fall neatly into any of these categories (e.g., Janzing *et al.* 2012; Janzing *et al.* 2015). In this subsection, we provide a high-level overview of common structure learning approaches.

This discussion will also circle us back to concepts in causal graphical modeling. Structure learning is often deployed in causal inference settings, where certain assumptions can allow us to interpret the graphical

structure as a representation of the causal mechanisms that explain the data generating process. When structure learning is causal, we refer to this task as *causal discovery* (Glymour *et al.*, 2019). In addition to score-based, constraint-based, and hybrid causal discovery methods, there is also a rich class of algorithms based on *functional causal models* that make parametric assumptions on the data generating process in exchange for other mathematical conveniences (Hoyer *et al.*, 2008; Zhang and Hyvärinen, 2009). Throughout this section, we will reference both causal discovery methods and non-causal structure learning approaches.

Global, Local, and Time-Series Structures This section mainly focuses on *global* structure learning, where all relations in the graph are of interest. In practice, we might only be interested in substructures of the global graph, such that learning the full graph would require extraneous computation. *Local* structure learning can provide efficient solutions for such problems. This includes the bivariate case, where we are interested in learning whether X causes Y or Y causes X (Mooij *et al.*, 2016). Another common case of local structure learning is *Markov blanket learning*, which can be used for causal feature selection in machine learning (Aliferis *et al.*, 2010; Yu *et al.*, 2020). Additionally, when data have a temporal dimension, we can apply *time-series* structure learning approaches. This chapter will not address the time-series setting, though we refer the reader to external resources (e.g., Assaad *et al.* 2022).

6.2.3 Score-Based Methods

We can formulate score-based structure learning as a search problem that consists of two parts:

1. *The goodness-of-fit metric.* First, we define a criterion to evaluate how well the Bayesian network fits the observed data.

2. *The search algorithm.* Next, we define a procedure that searches over the space of DAGs (or a constrained subspace) for a structure achieving the maximal score.

Putting these two parts together, we aim to search among a set of candidate graphs $\mathbf{G}$ for the graph $\mathcal{G}$ that best fits our data $\mathcal{D}$ according

to our score metric:

$$\underset{\mathcal{G} \in \mathbf{G}}{\operatorname{argmax}} \ \operatorname{Score}(\mathcal{G}, \mathcal{D}).$$

Score-based structure learning is traditionally treated as a combinatorial optimization problem, though recent innovations have also given rise to continuous optimization approaches (Zheng *et al.*, 2018).

Score Metrics

Score metrics for a structure $\mathcal{G}$ and data $\mathcal{D}$ generally take the form

$$\operatorname{Score}(\mathcal{G}, \mathcal{D}) = \underbrace{\ell(\mathcal{G}; \mathcal{D})}_{\text{log-likelihood}} - \underbrace{\phi(|\mathcal{D}|)\, \|\mathcal{G}\|}_{\text{regularization}}.$$

Here $\ell(\mathcal{G}; \mathcal{D})$ refers to the log-likelihood of the data under the graph structure $\mathcal{G}$. The parameters in the Bayesian network $\mathcal{G}$ are estimated based on MLE and the log-likelihood score is calculated based on the estimated parameters. If the score function only consisted of the log-likelihood term, then the optimal graph would be a complete graph, which is probably overfitting the data. We can mitigate the potential for overfitting by introducing the regularization term $\phi(|\mathcal{D}|)\, \|\mathcal{G}\|$, where $|\mathcal{D}|$ is the number of data samples and $\|\mathcal{G}\|$ is the number of parameters in the graph $\mathcal{G}$. This kind of regularization penalizes models with many parameters, favoring simpler models.

When $\phi(t) = 1$, this score function is known as the *Akaike Information Criterion* (AIC; Akaike 1998; Cavanaugh and Neath 2019). When $\phi(t) = \log(t)/2$, this score function is known as the *Bayesian Information Criterion* (BIC; Schwarz 1978; Neath and Cavanaugh 2012). With the BIC, the influence of model complexity decreases as $|\mathcal{D}|$ grows, allowing the log-likelihood term to eventually dominate the score. The AIC and BIC differ in their properties and assumptions, and should be selected based on the needs of the experiment at hand (Vrieze, 2012).

Many other score functions have since been proposed. Another family of Bayesian score functions is known as the *Bayesian Dirichlet (BD) score*. The BD score first defines the probability of data $\mathcal{D}$ conditional on the graph structure $\mathcal{G}$ as

$$p(\mathcal{D}; \mathcal{G}) = \int p(\mathcal{D} \mid \Theta_{\mathcal{G}}; \mathcal{G})\, p(\Theta_{\mathcal{G}}; \mathcal{G})\, d\Theta_{\mathcal{G}},$$

where $p(\mathcal{D} \mid \Theta_{\mathcal{G}}; \mathcal{G})$ is the probability of the data given the network structure $\mathcal{G}$ and parameters $\Theta_{\mathcal{G}}$, and $p(\Theta_{\mathcal{G}}; \mathcal{G})$ is the prior probability of the parameters. When the prior probability is specified as a Dirichlet distribution,

$$p(\mathcal{D} \mid \Theta_{\mathcal{G}}) = \prod_i \prod_{\pi_i} \left[\frac{\Gamma(\sum_j N'_{i,\pi_i,j})}{\Gamma(\sum_j N'_{i,\pi_i,j} + N_{i,\pi_i,j})} \prod_j \frac{\Gamma(N'_{i,\pi_i,j} + N_{i,\pi_i,j})}{\Gamma(N'_{i,\pi_i,j})} \right].$$

Here π_i refers to the parent configuration of the variable i and $N_{i,\pi_i,j}$ is the count of variable i taking value j with parent configuration π_i. N' represents the counts in the prior.

With a prior over the model parameters $p(\Theta_{\mathcal{G}})$ (say, a uniform one), the BD score is defined as

$$\log p(\mathcal{D} \mid \Theta_{\mathcal{G}}) + \log p(\Theta_{\mathcal{G}}).$$

Notice that there is no penalty term appended to the BD score, as it will penalize overfitting implicitly via the integral over the parameter space.

While score-based methods do not suffer from the multiple testing problems that are innate to constraint-based structure learning, most score metrics impose parametric assumptions on the distributional forms of random variables and/or the functional forms of parent-child relationships. Such parametric assumptions can be difficult to justify in real-world settings, and violations of these assumptions can result in inaccurate structure learning. These limitations have given rise to various *nonparametric score metrics*, including those that accommodate mixed data types. These often employ regression in Reproducing Kernel Hilbert Space, though kernel-based methods can be slow to run in practice (Huang *et al.*, 2018).

Search Algorithms

Search procedures commonly take a *local search* or *greedy search* approach. Both approaches are computationally tractable and sometimes offer guarantees on correctness under infinite data and other sufficient conditions. However, neither type of approach can guarantee the quality of the learned graph in finite sample regimes (like those we encounter in

real data). Our search space is highly non-convex, and search algorithms can get stuck in sub-optimal regions (e.g., local maxima).

A popular set of search frameworks is owed to *Greedy Equivalence Search* (GES) and its variants (Chickering, 2002). To reasonably constrain our search space, GES searches over Markov equivalence classes (MECs; Definition 3.3) instead of the space of unique DAGs. First, we start with the empty graph and iteratively add edges until goodness-of-fit reaches a local maximum. Next, we iteratively eliminate edges until the score again reaches a local maximum. GES is proven to correctly identify the optimal solution in the large sample limit when applied to the sparsely-connected search space of MECs.

More concretely, the two phases of GES are as follows:

1. *Forward Equivalence Search (FES).* Starting with the equivalence class of no dependencies, greedily make single-edge additions until a local maximum is reached. When all assumptions are met, the local maximum reached after FES contains the generative distribution.

2. *Backward Equivalence Search (BES).* Consider all single-edge deletions within the current equivalence class until a local maximum is reached. When all assumptions are met, the equivalence class reached after BES must be a perfect map of the generative distribution.

Though GES has spawned many influential extensions, it presents with several limitations. While GES is guaranteed to return a perfect map of the generative distribution in the large sample limit when all assumptions are met, these conditions are generally unrealistic and unverifiable. Though computationally convenient, returning the MEC provides less graphical information than returning the unique DAG. While the original formulation of GES assumes no latent confounding, recent extensions address this limitation (Claassen and Bucur, 2022).

Chow-Liu Algorithm

The Chow-Liu Algorithm is a classical score-based approach for identifying the maximum likelihood tree-structured graph (Chow and Liu,

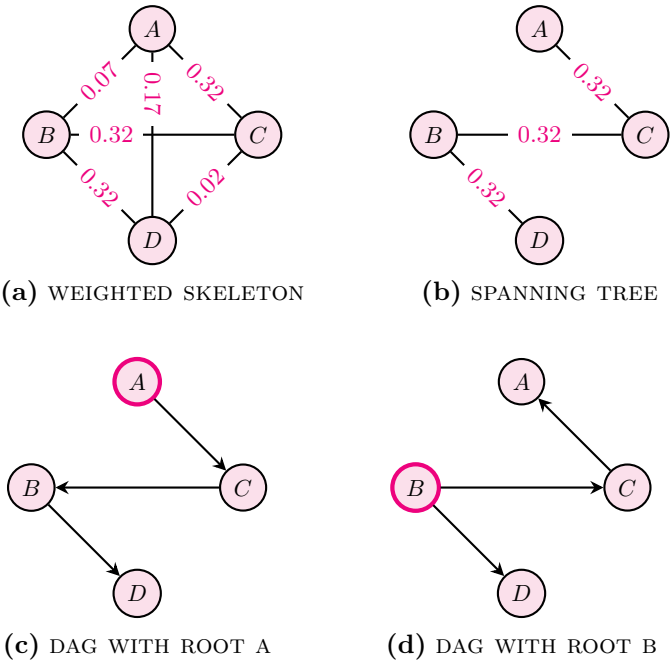


Figure 6.3: Graph learning with Chow-Liu (Algorithm 5). Step 1 (**a**) yields an undirected skeleton where each edge is weighted by the mutual information for the incident pair of random variables. Step 2 (**b**) identifies the maximum weight spanning tree. Step 3 (**c** and **d**) yields the DAG corresponding to the selected root.

1968). We restrict our attention to trees where each node has exactly one parent, except for a parentless root node (Definition 2.41). The score that we employ here is simply the log-likelihood. There is no penalty term for graph structure complexity, since the algorithm only considers tree structures.

The algorithm has three steps, which we outline in Algorithm 5. We visualize possible inputs and outputs for Algorithm 5 in Figure 6.3.

Time Complexity Let n be the number of nodes in our graph. The time complexity of the Chow-Liu Algorithm is quadratic in n, as it takes $O(n^2)$ time to compute the mutual information for all pairs of variables and $O(n^2)$ time to compute the maximum spanning tree.

Algorithm 5 *Chow-Liu Algorithm*

1. Compute the mutual information for all pairs of variables $\{X, U\}$, and form a complete graph from the variables where the edge between variables $\{X, U\}$ has weight equivalent to the mutual information term $MI(X, U)$:

$$\text{MI}(X, U) = \sum_{x,u} \hat{p}(x, u) \log \left[\frac{\hat{p}(x, u)}{\hat{p}(x)\hat{p}(u)} \right].$$

   This function measures how much information U provides about X. Recall that from our empirical distribution

$$\hat{p}(x, u) = \frac{Count(x, u)}{\#\text{ data points}}.$$

2. Find the *maximum weight spanning tree*, which can be done programmatically by using Kruskal's Algorithm (Kruskal, 1956) or Prim's Algorithm (Prim, 1957).

3. Pick any node to be the *root variable*, and assign directions radiating outward from this node (i.e., all arrows point away from the root). This step transforms the resulting undirected tree into a directed one. We then return the directed tree.

Why it Works Having described the algorithm, let's explain why this works. It turns out that the likelihood score decomposes into mutual information and entropy terms:

$$\log p(\mathcal{D}\,;\, \theta, \mathcal{G}) = |\mathcal{D}| \sum_i \underbrace{\text{MI}(X_i, \mathbf{pa}(X_i))}_{\text{mutual information}} - |\mathcal{D}| \sum_i \underbrace{H(X_i)}_{\text{entropy}}.$$

We would like to find a graph $\mathcal{G}$ that maximizes this log-likelihood. Since the entropies are independent of the dependency ordering in the tree, the only terms that change with the choice of $\mathcal{G}$ are the mutual

information terms. So we want

$$\operatorname*{argmax}_{\mathcal{G}} \log p(\mathcal{D}\,;\,\theta,\mathcal{G}) = \operatorname*{argmax}_{\mathcal{G}} \sum_i \mathrm{MI}(X_i, \mathbf{pa}(X_i)).$$

Now if we assume $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ is a tree (where each node has at most one parent), then

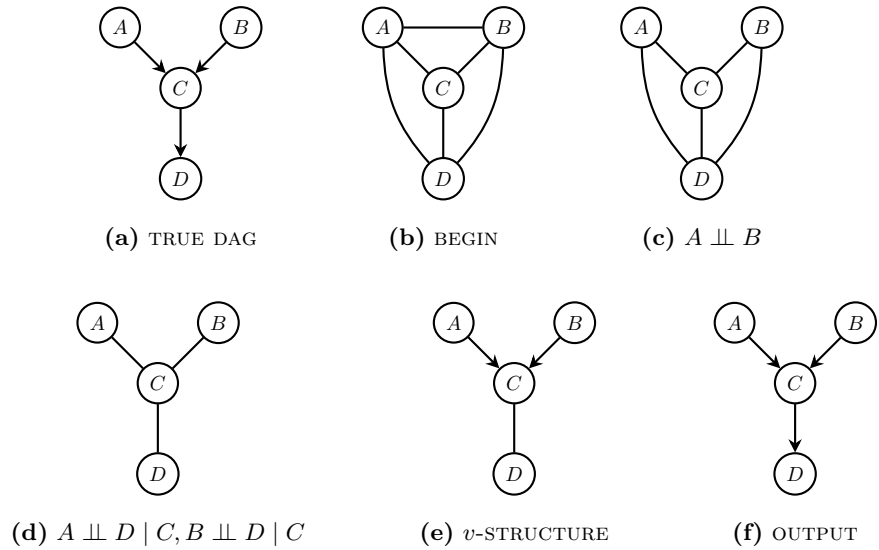
$$\operatorname*{argmax}_{\mathcal{G}\in\text{trees}} \log p(\mathcal{D}\,;\,\theta,\mathcal{G}) = \operatorname*{argmax}_{\mathcal{G}\in\text{trees}} \sum_{(i,j)\in\mathbf{E}} \mathrm{MI}(X_i, X_j).$$

The orientation of edges does not matter here, as mutual information is symmetric. Thus, we can see why the Chow-Liu algorithm finds a tree structure that maximizes the log-likelihood of the data. As illustrated in Figure 6.3, this can result in different DAGs depending on which root is selected.

6.2.4 Constraint-Based Methods

Constraint-based structure learning employs sequential conditional independence testing to identify a set of edge constraints for the graph. Then, edge orientation rules are applied to the undirected skeleton to obtain the final output. Often, the conditional independence test is chosen by the user and simply plugged into the general algorithm, enabling significant flexibility. This family of algorithms generally benefits from a well-established body of theory, asymptotic guarantees on correctness, human interpretability, and no innate parametric assumptions (though the user-selected independence test or specific algorithmic design choices might impose their own assumptions).

The most famous example of constraint-based structure learning is the causal discovery algorithm *PC*, which takes its name from the names of its authors (Spirtes *et al.*, 2001). Constraint-based methods like PC exploit the conditional independence properties of v-structures, which are distinct from the conditional independencies present in forks and chains (Table 3.1, Figure 3.2). Figure 6.4 provides a demonstration of PC for learning a y-structured DAG (itself a canonical structure that can be exploited for graph learning; see Mani *et al.* 2006). Note that in this case, PC is able to identify the full unique graph (i.e., all edges are



(a) TRUE DAG **(b)** BEGIN **(c)** $A \perp\!\!\!\perp B$

(d) $A \perp\!\!\!\perp D \mid C, B \perp\!\!\!\perp D \mid C$ **(e)** v-STRUCTURE **(f)** OUTPUT

Figure 6.4: A demonstration of the classic constraint-based PC algorithm for causal discovery (Spirtes *et al.*, 2001). In this case, the true DAG is a y-structure (**a**). To begin, we consider the complete undirected graph induced by $\{A, B, C, D\}$ (**b**). A test of marginal independence indicates that $A \perp\!\!\!\perp B$, and so the edge $A - B$ is removed (**c**). Next, tests of conditional independence indicate that $A \perp\!\!\!\perp D \mid C$ and $B \perp\!\!\!\perp D \mid C$, and so $A - D$ and $B - D$ are removed as well (**d**). As $A \perp\!\!\!\perp B$ and $A \not\perp\!\!\!\perp B \mid C$, we can orient the v-structure $A \rightarrow C \leftarrow B$ (**e**). Finally, we can apply orientation propagation rules to infer that $C \rightarrow D$ (**f**). This yields the correct output: a y-structure. Figure and caption adapted from Glymour *et al.* (2019).

oriented). In general, PC can only identify up to the MEC and outputs often contain at least some unoriented edges.

While the PC algorithm assumes that all confounders in the true graph are observed (i.e., *causal sufficiency*), the popular *Fast Causal Inference* (FCI) algorithm and its variants extend PC to allow for latent confounding (i.e., *causal insufficiency*). See Spirtes *et al.* (2013) for a deep discussion of related methods. Other prevalent assumptions in constraint-based learning involve the equivalence between properties of the data and properties of the causal graph:

1. *Causal Markov condition:* Any d-separation in graph $\mathcal{G}$ ($\perp\!\!\!\perp_{\mathcal{G}}$)

implies conditional independence in data distribution p ($\perp\!\!\!\perp_p$),

$$X \perp\!\!\!\perp_{\mathcal{G}} Y|Z \implies X \perp\!\!\!\perp_p Y|Z.$$

2. *Faithfulness:* Conditional independence implies d-separation,

$$X \perp\!\!\!\perp_p Y|Z \implies X \perp\!\!\!\perp_{\mathcal{G}} Y|Z.$$

Despite their popularity, constraint-based methods also face several drawbacks. Many recent advances in constraint-based discovery aim to alleviate the following challenges:

- *Statistical challenges.* Conditional independence testing is a hard problem in itself (Shah and Peters, 2020), and sequential testing raises its own set of statistical problems. Thus, the performance of constraint-based methods will be bottlenecked by the performance of the chosen independence test.

- *Sample complexity.* The sample complexity of conditional independence testing is exponential in the cardinality of the conditioning set, which can result in poor finite sample performance.

- *Time complexity.* Unless assumptions are imposed, the total number of conditional independence tests will be worst-case exponential in the total number of nodes. Several approaches can be taken to improve runtimes. For example, many constraint-based procedures are *embarrassingly parallelizable*: at least some speedups can be gained simply by paralellizing the sequential tests whose outputs do not rely on each other.

- *Informativeness.* As with GES, purely constraint-based methods can only identify the true graph up to its MEC. Additionally, methods that rely solely on conditional independence testing require at least three variables. Therefore, these cannot handle the case of bivariate direction inference, where only two variables are considered (Mooij *et al.*, 2016).

6.2.5 Functional Causal Models

If we are willing to impose parametric assumptions on our causal model, we can identify the unique DAG describing our data distribution instead of its MEC. A popular family of causal discovery algorithms takes this approach by specifying a constrained *functional causal model* (FCM). The most common FCM is the *additive noise model* (ANM), which represents effect Y as a function of its direct causes $\mathbf{X}$ and some exogenous random noise ϵ, where function f can take arbitrary forms:

$$y = f(\mathbf{x}) + \epsilon.$$

Crucially, the ANM assumes that $\mathbf{X}$ and ϵ are *independent* while Y and ϵ are *dependent*. This assumption allows us to detect asymmetries in the observed data that can be used to infer edge directionality. Consider the bivariate case of X and Y: if the hypothetical causal model $X \to Y$ displays independence between the putative parent and the noise term while the alternative model $Y \to X$ does not, the independent noise assumption dictates that we model the causal relationship as $X \to Y$ (Figure 6.5).

The ANM is not the only FCM, and we can instead define a more general class. The post-nonlinear causal model (PNL) is the most general of the well-defined FCMs. It is suited for bivariate inference and inference on larger structures. The PNL accounts for the nonlinear effect of the cause, the inner noise effect, as well as a measurement distortion effect in the observed variables. Under this model, each variable X_i in graph $\mathcal{G}$ takes the form

$$x_i = g_i(f_i(\mathbf{pa}(x_i)) + \epsilon_i),$$

where $\mathbf{pa}(x_i)$ are the parents of X_i, ϵ_i is the exogenous noise term, f_i is a nonlinear causal function, and g_i is an invertible post-nonlinear distortion.

The PNL comes with thorough identifiability results (Zhang and Hyvärinen, 2009). Most notably, the PNL is not identifiable in the linear-Gaussian setting. The identifiability conditions of the PNL extend to its special cases, which include the following:

- *Linear additive models.* Given $x_i = g_i(f_i(\mathbf{pa}(x_i)) + \epsilon_i)$, f_i is linear and g_i is the identity function. Examples include the family

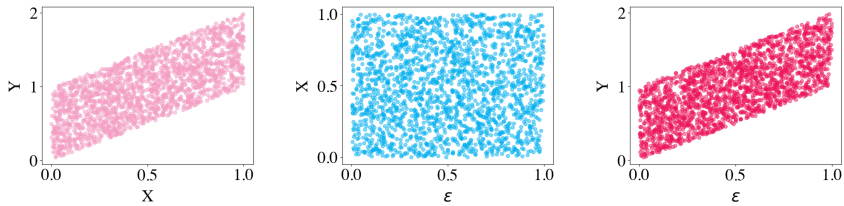


Figure 6.5: A bivariate ANM with linear causal functions and uniform noise distributions. In the true structural causal model, the variable X and exogenous random noise ϵ are the causes of variable Y. As expected under the assumptions of the ANM, when we plot X against ϵ, we observe independence (center). When we plot Y against ϵ, we observe dependence (right).

of linear non-Gaussian acyclic models, also known as LiNGAM (Shimizu *et al.*, 2011). While assuming linear causal functions is generally unrealistic for real-world data, LiNGAM methods can be very time-efficient.

- *Nonlinear additive models.* Here, f_i is nonlinear and g_i is the identity function. Examples include the nonlinear ANM presented by Hoyer *et al.* (2008). Methods for nonlinear ANM discovery have more realistic assumptions than LiNGAM, but can be slow to run in practice.

- *Multiplicative noise models.* The multiplicative noise model is a special case of the PNL, as $x_i = \mathbf{pa}(x_i) \cdot \epsilon_i$ can be expressed as $\exp(\log \mathbf{pa}(x_i) + \log \epsilon_i)$ where $f_i(\mathbf{pa}(x_i)) = \log(\mathbf{pa}(x_i))$ and $g_i(\cdot) = \exp(\cdot)$.

At a high-level, a simple formulation for FCM discovery can take the following approach. Assume the bivariate case with variables X and Y, where edge directionality is unknown.

1. Assume that $\{X, Y\}$ have a direct causal relationship with no confounders and specify the desired parametric assumptions.

2. Fit the FCM for both causal directions and test for independence between the estimated noise and the hypothetical cause. For example, we could perform nonlinear regression of y on x to obtain $\hat{f}$, where residuals $\hat{\epsilon} = y - \hat{f}(x)$.

3. The direction that finds the hypothetical cause and noise terms to be statistically independent is considered plausible.

In practice, causal discovery for FCMs often disaggregates the overall problem into two subproblems: (1) learning the causal ordering, where a topological sort (Definition 2.37) of the random variables is obtained; and (2) spurious edge pruning, where the final set of directed edges is obtained from the set of all possible edges admitted by the causal ordering (Peters *et al.*, 2014; Bühlmann *et al.*, 2014).

While the parametric assumptions of FCMs enable a wide range of interesting algorithms, these assumptions are not generally verifiable. Thus, justifying the realism of these assumptions can be challenging. Violations of these unverifiable parametric assumptions can impact performance on real-world data, which is a primary drawback to using FCMs in practice.

Further Reading

Structure Learning in Bayesian Networks

- D. Heckerman *et al.* (1995). "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data". *Machine learning*. 20: 197–243.
- D. M. Chickering *et al.* (1995). "Learning Bayesian Networks: Search Methods and Experimental Results". In: *Preproceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*. Ed. by D. Fisher and H.-J. Lenz. Vol. R0. PMLR. 112–128.

Causal Discovery

- P. Spirtes *et al.* (2001). *Causation, Prediction, and Search*. MIT Press.
- D. Heckerman *et al.* (2006). "A Bayesian Approach to Causal Discovery". *Innovations in Machine Learning: Theory and Applications*: 1–28.

- C. Glymour *et al.* (2019). "Review of Causal Discovery Methods Based on Graphical Models". *Frontiers in genetics*. 10.
- M. J. Vowels *et al.* (2022). "D'ya Like DAGs? A Survey on Structure Learning and Causal Discovery". *ACM Computing Surveys*. 55(4).

6.3 Learning in Undirected Models

We now turn our attention to parameter learning in undirected graphical models. Unfortunately, as in the case of inference, the higher expressivity of undirected models also makes them significantly more difficult to learn than directed models. Fortunately, maximum likelihood learning in these models can be reduced to repeatedly performing inference, which will allow us to apply all the approximate inference techniques that we have seen earlier.

6.3.1 Learning in Markov Random Fields

Let us begin with an MRF (Section 3.2.1) of the form

$$p(\mathbf{x} = x_1, \ldots, x_n) = \frac{1}{Z(\varphi)} \prod_{c \in \mathbf{C}} \phi_c(\mathbf{x}_c; \varphi),$$

where

$$Z(\varphi) = \sum_{x_1, \ldots, x_n} \prod_{c \in \mathbf{C}} \phi_c(\mathbf{x}_c; \varphi)$$

is the normalizing constant. We can reparametrize p as follows:

$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{Z(\varphi)} \exp\left(\sum_{c \in \mathbf{C}} \log \phi_c(\mathbf{x}_c; \varphi) \right) \\ &= \frac{1}{Z(\varphi)} \exp\left(\sum_{c \in \mathbf{C}} \sum_{\mathbf{x}'_c} 1\{\mathbf{x}'_c = \mathbf{x}_c\} \log \phi_c(\mathbf{x}'_c; \varphi) \right) \\ &= \frac{\exp(\theta^T f(\mathbf{x}))}{Z(\theta)}, \end{aligned}$$

where $f(\mathbf{x})$ is a vector of indicator functions and θ is the set of all the model parameters, as defined by the $\log \phi_c(\mathbf{x}'_c; \varphi)$.

Note that $Z(\theta)$ is different for each set of parameters θ. Therefore, we also refer to $Z(\theta)$ as the *partition function*. Bayesian networks are constructed such that $Z(\theta) = 1$ for all θ. MRFs do not make this modeling assumption, which makes them more flexible but also more difficult to learn.

Exponential Families

More generally, distributions of the above form are members of the *exponential family* of distributions. Many other distributions are in the exponential family, including the Bernoulli, Multinomial, Normal, and Poisson distributions. See Wainwright and Jordan (2008) for an overview of exponential families in graphical modeling.

Exponential families are widely used in machine learning. Suppose that you have an exponential family distribution of the form

$$p(\mathbf{x}; \theta) = \frac{\exp(\theta^T f(\mathbf{x}))}{Z(\theta)}.$$

Here are few facts about these distributions that are useful to keep in mind:

- Exponential families are log-concave in their *natural parameters* θ. The partition function $Z(\theta)$ is also log-convex in θ.
- The vector $f(\mathbf{x})$ is called the vector of *sufficient statistics*. These fully describe the distribution p. For example, if p is Gaussian, then $f(\mathbf{x})$ contains (simple reparametrizations of) the mean and the variance of p.
- Exponential families make the fewest unnecessary assumptions about the data distribution. More formally, the distribution maximizing the entropy $H(p)$ under the constraint $\mathbb{E}_p[\phi(\mathbf{x})] = \alpha$ (i.e., the sufficient statistics equal some value α) is in the exponential family.

Exponential families are also very convenient to work with computationally. Their sufficient statistics can summarize arbitrary amounts

of iid variables from the same distribution, and they admit so-called conjugate priors which makes them easily applicable in variational inference.

6.3.2 Maximum Likelihood Learning in MRFs

Suppose now that we are given a dataset $\mathcal{D}$ and we want to estimate θ via maximum likelihood. Since we are working with an exponential family, the maximum likelihood will be concave. We can express the log-likelihood as

$$\frac{1}{|\mathcal{D}|} \log p(\mathcal{D}; \theta) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \theta^T f(\mathbf{x}) - \log Z(\theta).$$

The first term is linear in θ and is easy to handle. The second term is equal to

$$\log Z(\theta) = \log \sum_{\mathbf{x}} \exp(\theta^T f(\mathbf{x})).$$

Unlike the first term, this one does not decompose across $\mathbf{x}$. It is not only hard to optimize, but also to evaluate.

Now consider the gradient of the log-likelihood. Obtaining the gradient of the linear part is very easy. However, the gradient of $\log Z(\theta)$ takes a more complicated form:

$$\nabla_\theta \log Z(\theta) = \mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})].$$

This expression follows from differentiating the summation and applying the chain rule.

Computing the expectation on the right hand side of the equation requires inference with respect to p. For example, we could sample from p and construct a Monte Carlo estimate of the expectation. However, as we have seen, inference in general is intractable, and therefore so is computing the gradient.

We can similarly derive an expression for the Hessian of $\log Z(\theta)$:

$$\nabla^2_\theta \log Z(\theta) = \mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x}) f(\mathbf{x})^T] - \mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})] \mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})]^T = \mathrm{Cov}[f(\mathbf{x})].$$

Since covariance matrices are always positive semi-definite, this proves that $\log Z(\theta)$ is convex (and therefore that the log-likelihood is concave).

Recall that this was one of the properties of exponential families that we stated earlier.

In summary, even though the log-likelihood objective is convex, optimizing it is hard. Usually non-convexity is what makes optimization intractable, but in this case, it is the computation of the gradient that poses challenges.

Approximate Learning Techniques

Interestingly, however, maximum likelihood learning reduces to inference in the sense that we may repeatedly use inference to compute the gradient and determine the model weights using gradient descent.

This observation lets us apply to learning many of the approximate inference methods that we introduced in Chapter 5, such as:

- *Gibbs sampling* (Algorithm 4) from the distribution at each step of gradient descent. We can then approximate the gradient using Monte Carlo.
- *Persistent contrastive divergence*, a variant of Gibbs sampling which re-uses the same Markov Chain between iterations. After a step of gradient descent, our model has changed very little. Thus, we can essentially keep taking samples from the same Gibbs sampler instead of starting a new one from scratch.[1]

These approaches replace exact gradient computation with approximations based on samples, allowing learning to proceed even when exact inference is intractable. While such methods may introduce bias and variance into the gradient estimates, they are sometimes the only practical option for learning in large-scale undirected models.

[1] As an aside, persistent contrastive divergence is one of the most popular methods for training Restricted Boltzmann Machines, an early deep learning model that is also an undirected graphical model (Tieleman, 2008).

Pseudo-Likelihood

Another popular approach to learning p is called the *pseudo-likelihood*. The pseudo-likelihood replaces the likelihood

$$\frac{1}{|\mathcal{D}|} \log p(\mathcal{D}; \theta) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log p(\mathbf{x}; \theta)$$

with the following approximation:

$$\ell_{\mathrm{PL}}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i} \log p(x_i \mid \mathbf{ne}(x_i); \theta),$$

where x_i is the i-th variable in $\mathbf{x}$ and $\mathbf{ne}(x_i)$ is the set of neighbors of x_i (i.e., the Markov blanket of x_i).

Note that each term $\log p(x_i \mid \mathbf{ne}(x_i); \theta)$ only involves one variable x_i and hence its partition function is going to be tractable (we only need to sum over the values of one variable).

However, this approximation is not equal to the likelihood. Note that the correct way to expand the likelihood would involve the chain rule. That is, the terms would be the $\log p(x_i \mid x_{-i}; \theta)$ objective, where x_{-i} are variables preceding i in some ordering.

Intuitively, the pseudo-likelihood objective assumes that x_i depends mainly on its neighbors in the graph and ignores the dependencies on other more distant variables. Observe also that if the pseudo-likelihood succeeds in matching all the conditional distributions to the data, a Gibbs sampler run on the model distribution will have the same invariant distribution as a Gibbs sampler run on the true data distribution, ensuring that they are the same.

More formally, we can show that the pseudo-likelihood objective is concave. Assuming the data are drawn from an MRF with parameters θ^*, we can show that as the number of data points gets large, $\theta_{\mathrm{PL}} \to \theta^*$. The pseudo-likelihood often works well in practice, although there are exceptions.

Moment Matching

We will end with an interesting observation about the maximum likelihood estimate $\hat{\theta}$ of the MRF parameters. Recall that the log-likelihood

of an MRF is

$$\frac{1}{|\mathcal{D}|} \log p(\mathcal{D}; \theta) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \theta^T f(\mathbf{x}) - \log Z(\theta).$$

Taking the gradient using our expression for the gradient of the partition function, we obtain the expression

$$\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} f(\mathbf{x}) - \mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})].$$

Note that this is precisely the difference between the expectations of the natural parameters under the empirical and the model distributions. Let's now look at one component of $f(\mathbf{x})$. Recall that we have defined f in the context of MRFs to be the vector of indicator functions for the variables of a clique: one entry of f equals $\mathbb{I}[\mathbf{x}_c = \bar{\mathbf{x}}_c]$ for some $\mathbf{x}_c, \bar{\mathbf{x}}_c$. The gradient over that component equals

$$\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbb{I}[\mathbf{x}_c = \bar{\mathbf{x}}_c] - \mathbb{E}_{\mathbf{x} \sim p}[\mathbb{I}[\mathbf{x}_c = \bar{\mathbf{x}}_c]].$$

This gives us an insight into how MRFs are trained. The log-likelihood objective forces the model marginals to match the empirical marginals.

We refer to the above property as *moment matching*. This property of maximum likelihood learning is very general: whenever we choose distribution q to minimize the inclusive KL divergence $\mathrm{KL}(p \,\|\, q)$ across q in an exponential family, the minimizer will match the moments of the sufficient statistics to the corresponding moments of p. Recall that the MLE estimate is the minimizer over q of $\mathrm{KL}(\tilde{p} \,||\, q)$, where $\tilde{p}$ is the empirical distribution of the data (Section 6.1.1), so MLE estimation is a special case of this minimization. This property has connections to variational inference, where this minimization over q in a smaller set of distributions $\mathcal{Q}$ is known as M-projection ("moment projection").

6.3.3 Learning in Conditional Random Fields

Finally, we will consider how maximum likelihood learning extends to CRFs (Section 3.2.3). Recall that a CRF is a probability distribution of the form

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x}, \varphi)} \prod_{c \in \mathbf{C}} \phi_c(\mathbf{y}_c, \mathbf{x}; \varphi),$$

where

$$Z(\mathbf{x}, \varphi) = \sum_{y_1, \ldots, y_n} \prod_{c \in \mathbf{C}} \phi_c(\mathbf{y}_c, \mathbf{x}; \varphi)$$

is the partition function. The feature functions now depend on $\mathbf{x}$ in addition to $\mathbf{y}$. The $\mathbf{x}$ variables are fixed and the distribution is only over $\mathbf{y}$. The partition function is thus a function of both $\mathbf{x}$ and φ. We can reparametrize p as we did for MRFs:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp(\theta^T f(\mathbf{x}, \mathbf{y}))}{Z(\mathbf{x}, \theta)},$$

where $f(\mathbf{x}, \mathbf{y})$ is again a vector of indicator functions and θ is a reparametrization of the model parameters. The log-likelihood for this model given a dataset $\mathcal{D}$ is

$$\frac{1}{|\mathcal{D}|} \log p(\mathcal{D}; \theta) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{D}} \theta^T f(\mathbf{x}, \mathbf{y}) - \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \log Z(\mathbf{x}, \theta).$$

Note that this is almost the same form as we had for MRFs, except that now there is a different partition function $\log Z(\mathbf{x}, \theta)$ for each data point $\mathbf{x}, \mathbf{y}$. The gradient is now

$$\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}, \mathbf{y} \in \mathcal{D}} f(\mathbf{x}, \mathbf{y}) - \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})}[f(\mathbf{x}, \mathbf{y})]$$

Similarly, the Hessian is going to be the covariance matrix

$$\mathrm{Cov}_{\mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})}[f(\mathbf{x}, \mathbf{y})].$$

The good news is that the conditional log-likelihood is still a concave function. We can optimize it using gradient ascent as before. The bad news is that computing the gradient now requires one inference per training data point $\mathbf{x}, \mathbf{y}$ in order to compute the term

$$\sum_{\mathbf{x}, \mathbf{y} \in \mathcal{D}} \log Z(\mathbf{x}, \theta).$$

This makes learning CRFs more expensive than learning in MRFs. In practice, however, CRFs are more widely used than MRFs: supervised learning is a widely used learning paradigm, and discriminative models (like CRFs) often learn a better classifier than their generative counterparts (which model $p(\mathbf{x}, \mathbf{y})$).

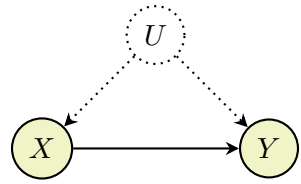


Figure 6.6: A DAG with latent variable U and observed variables X and Y.

To deal with the computational difficulties introduced by the partition function, we can use simpler models in which exact inference is tractable. This was the approach taken in the optical character recognition example that we introduced in Section 3.2.3 (Figure 3.14). More generally, one should try to limit the number of variables or make sure that the model's graph is not too densely connected.

Finally, we would like to add that there exists another popular objective for training CRFs called the *max-margin loss*, a generalization of the objective for training support vector machines. Models trained using this loss are called structured support vector machines or max-margin networks (Tsochantaridis *et al.*, 2004; Tsochantaridis *et al.*, 2005). This loss is more widely used in practice because it often leads to better generalization. Additionally, it requires only MAP inference to compute the gradient rather than general (e.g., marginal) inference, which is often more expensive to perform.

6.4 Learning in Latent Variable Models

So far, we have assumed that learning takes place in settings where all variables of interest are *observed* or *measured*. However, this is not always the case in practice.

Consider a probabilistic language model of news articles. Each article $\mathbf{x}$ focuses on a specific topic $\mathbf{t}$ (e.g., finance, sports, politics, etc.). Using this prior knowledge, we can build a more accurate model $p(\mathbf{x} \mid \mathbf{t})p(\mathbf{t})$, in which we introduce an additional variable $\mathbf{t}$ that we have not directly observed. In this case, we say that $\mathbf{t}$ is *unobserved*, *unmeasured*, *hidden*, or *latent*. Our new model $p(\mathbf{x} \mid \mathbf{t})p(\mathbf{t})$ can provide greater accuracy, as we can now learn a separate $p(\mathbf{x} \mid \mathbf{t})$ for each topic rather than trying to

model everything with $p(\mathbf{x})$. However, since $\mathbf{t}$ is unobserved, we cannot directly use the learning methods that we have seen so far. In fact, unobserved variables make learning much more difficult.

In this section, we will look at how to use and how to learn models that involve latent variables. More formally, we define a latent variable model (LVM) p as a probability distribution

$$p(\mathbf{x}, \mathbf{z}; \theta)$$

over two sets of variables $\mathbf{X}, \mathbf{Z}$, where variables in $\mathbf{X}$ are observed at learning time and variables in $\mathbf{Z}$ are never observed (and where θ are model parameters, as before). LVMs can be directed or undirected (Figure 6.6). There exist both discriminative and generative LVMs, although we will focus on the latter (though the key ideas will hold for discriminative models as well).

Why are Latent Variable Models Useful?

There are several notable reasons to use LVMs. The simplest reason is that some data are naturally unobserved. For example, clinical trials commonly see participants drop out before all measurements are taken. LVMs can be used to learn in the presence of such missing data.

Perhaps the most important reason for studying LVMs is that they enable us to leverage prior knowledge when defining a model. Our topic modeling example illustrates this. We know that our set of news articles is actually a mixture of K distinct distributions (one for each topic). LVMs allow us to design models that capture this.

LVMs can also be viewed as increasing the expressive power of our model. We illustrate this point by taking Gaussian Mixture Models (GMMs) as an example.

Gaussian Mixtures GMMs are probabilistic LVMs that assume the data of interest are generated from a mixture of Gaussian distributions — that is, the data represent a finite number of normally distributed subpopulations. GMMs are one of the most widely used models in machine learning. The distribution that we can model using a mixture

of Gaussian components is much more expressive than what we could have modeled using a single component (Figure 6.7).

In a GMM, each data point is a tuple $(\mathbf{x}_i, z_i)$ with d-dimensional vector $\mathbf{x}_i \in \mathbb{R}^d$ and discrete scalar $z_i \in \{1, 2, \ldots, K\}$. The joint p is a directed model

$$p(\mathbf{x}, z) = p(\mathbf{x} \mid z)p(z),$$

where $p(z = k) = \pi_k$ for vector of class probabilities $\pi \in \Delta_{K-1}$ and

$$p(\mathbf{x} \mid z = k) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$$

is a multivariate Gaussian with mean and variance μ_k, Σ_k. This model postulates that our observed data are comprised of K clusters with proportions specified by $\pi_1, \ldots, \pi_K$. The distribution within each cluster is a Gaussian. We can see that $p(\mathbf{x})$ is a mixture (i.e., a weighted sum) by explicitly writing out this probability:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{x} \mid z = k)p(z = k) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k).$$

To generate a new data point, we can sample a cluster k and then draw a sample from the corresponding Gaussian $\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$. This way, we could sample from either the red or blue cluster in Figure 6.7, which would not be possible if we had assumed that all data points were generated by a single Gaussian. We will revisit GMMs in Section 6.4.2 to illustrate our learning algorithms in action.

Marginal Likelihood Training

How do we train an LVM? Our goal is still to fit the marginal distribution $p(\mathbf{x})$ over the set of variables $\mathbf{X}$ to our observed dataset $\mathcal{D}$. Hence our previous discussion about KL divergences applies here as well. By the same argument, we should be maximizing the *marginal log-likelihood* of the data

$$\log p(\mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \left(\sum_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z}) \right).$$

This optimization objective is considerably more difficult than the regular log-likelihood, even for directed graphical models. For one,

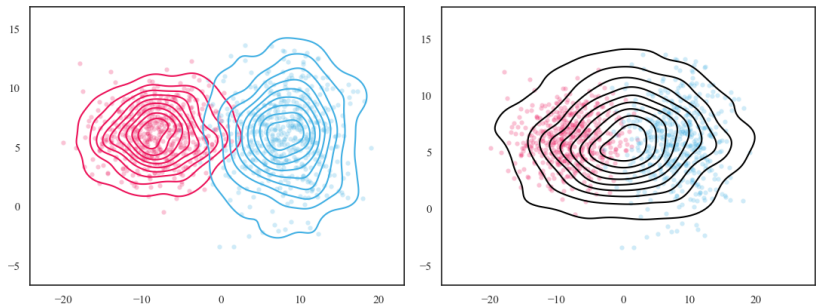


Figure 6.7: Example of a dataset that is better fit with a mixture of two Gaussians (left) than with only one (right). Mixture models allow us to model clusters in the dataset, as seen here in red and blue.

we can see that the summation inside the log makes it impossible to decompose $p(\mathbf{x})$ into a sum of log-factors. Even if the model is directed, we can no longer derive a simple closed-form expression for the parameters.

Looking closer at the distribution of a data point $\mathbf{x}$, we also see that it is actually a mixture

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})$$

of distributions $p(\mathbf{x} \mid \mathbf{z})$ with weights $p(\mathbf{z})$. Whereas a single exponential family distribution $p(\mathbf{x})$ has a concave log-likelihood (as we have seen in our discussion of undirected models), the log of a weighted mixture of such distributions is no longer concave or convex (Figure 6.8). This non-convexity requires the development of specialized learning algorithms.

6.4.1 Learning with Expectation-Maximization

Since our objective is non-convex, we will resort to approximate learning algorithms. These methods are widely used and quite effective in practice. We will focus our discussion on *Expectation-Maximization* (EM): a hugely important and widely used algorithm for learning directed latent variable graphical models (Dempster *et al.*, 1977).

The EM algorithm learns a LVM $p(\mathbf{x}, \mathbf{z}; \theta)$ with parameters θ and latent $\mathbf{z}$. This approach relies on two simple observations:

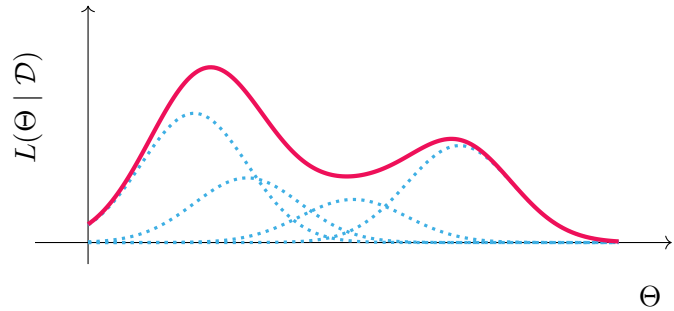


Figure 6.8: Exponential family distributions (dotted blue lines) have concave log-likelihoods. However, a weighted mixture of such distributions is no longer concave (solid red line).

1. If the latent $\mathbf{z}$ were fully observed, then we could optimize the log-likelihood exactly using the closed form solution for $p(\mathbf{x}, \mathbf{z})$.

2. Knowing the parameters θ, we can often efficiently compute the posterior $p(\mathbf{z} \mid \mathbf{x}; \theta)$. Note that this is an assumption, and this does not hold for some models.

EM follows a simple iterative two-step strategy (Algorithm 6): given an estimate θ_t of the parameters at time-step t, compute $p(\mathbf{z} \mid \mathbf{x})$ and use it to "hallucinate" values for $\mathbf{z}$. Then, find a new θ_{t+1} by optimizing the resulting tractable objective. This process will eventually converge.

We haven't exactly defined what we mean by "hallucinating" values for $\mathbf{z}$. The full definition is a bit technical, but its instantiation is very intuitive in most models (e.g., GMMs). Here, "hallucinating" means computing the expected log-likelihood

$$\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z} \mid \mathbf{x})} \log p(\mathbf{x}, \mathbf{z}; \theta).$$

This expectation is what gives the EM algorithm half of its name. If $\mathbf{z}$ is not too high-dimensional (e.g., in GMMs it is a one-dimensional categorical variable), then we can compute this expectation.

Since the summation is now outside the log, we can maximize the expected log-likelihood. In particular, when p is a directed model, $\log p$ again decomposes into a sum of log-conditional probability distribution terms that can be optimized independently.

Algorithm 6 *Expectation-Maximization*

Input: Dataset $\mathcal{D}$.

1: Initialize parameters at θ_0.
2: Repeat until convergence for $t = 1, 2, \ldots$:
3: - *E-Step*: For each $\mathbf{x} \in \mathcal{D}$, compute the posterior $p(\mathbf{z} \mid \mathbf{x}; \theta_t)$.
4: - *M-Step*: Compute new parameters via

$$\theta_{t+1} = \operatorname*{argmax}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z} \mid \mathbf{x}; \theta_t)} \log p(\mathbf{x}, \mathbf{z}; \theta).$$

EM as Variational Inference

Why exactly does EM converge? We can understand the behavior of EM by casting it in the framework of variational inference.

Consider the posterior inference problem for $p(\mathbf{z} \mid \mathbf{x})$, where the $\mathbf{x}$ variables are held fixed as evidence. We can apply our variational inference framework by taking $p(\mathbf{x}, \mathbf{z})$ to be the unnormalized distribution. In that case, $p(\mathbf{x})$ will be the normalization constant. Recall that variational inference maximizes the evidence lower bound (ELBO)

$$\mathcal{L}(p, q) = \mathbb{E}_{q(\mathbf{z})} \left[\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}) \right]$$

over distributions q. The ELBO satisfies the equation

$$\log p(\mathbf{x}; \theta) = \mathrm{KL}(q(\mathbf{z}) \parallel p(\mathbf{z} \mid \mathbf{x}; \theta)) + \mathcal{L}(p, q).$$

Hence, $\mathcal{L}(p, q)$ is maximized when $q = p(\mathbf{z} \mid \mathbf{x})$. In that case, the KL term becomes zero and the lower bound is tight: $\log p(\mathbf{x}; \theta) = \mathcal{L}(p, q)$.

The EM algorithm can be seen as iteratively optimizing the ELBO over q (at the E-step) and over θ (at the M-step). Starting at some θ_t, we compute the posterior $p(\mathbf{z} \mid \mathbf{x}; \theta_t)$ at the E-step. We evaluate the ELBO for $q = p(\mathbf{z} \mid \mathbf{x}; \theta_t)$. This makes the ELBO tight:

$$\log p(\mathbf{x}; \theta_t) = \mathbb{E}_{p(\mathbf{z} \mid \mathbf{x}; \theta_t)} \log p(\mathbf{x}, \mathbf{z}; \theta_t) - \mathbb{E}_{p(\mathbf{z} \mid \mathbf{x}; \theta_t)} \log p(\mathbf{z} \mid \mathbf{x}; \theta_t)$$

Next, we optimize the ELBO over p, holding q fixed. We then solve the problem

$$\theta_{t+1} = \operatorname*{argmax}_{\theta} \mathbb{E}_{p(\mathbf{z} \mid \mathbf{x}; \theta_t)} \log p(\mathbf{x}, \mathbf{z}; \theta) - \mathbb{E}_{p(\mathbf{z} \mid \mathbf{x}; \theta_t)} \log p(\mathbf{z} \mid \mathbf{x}; \theta_t).$$

Note that this is precisely the optimization problem solved at the M-step of EM (in the above equation, there is an additive constant independent of θ). Solving this problem increases the ELBO. However, since we fixed q to $\log p(\mathbf{z} \mid \mathbf{x}; \theta_t)$, the ELBO evaluated at the new θ_{t+1} is no longer tight. But since the ELBO was equal to $\log p(\mathbf{x}; \theta_t)$ before optimization, we know that the true log-likelihood $\log p(\mathbf{x}; \theta_{t+1})$ must have increased.

We now repeat this procedure, computing $p(\mathbf{z} \mid \mathbf{x}; \theta_{t+1})$ (the E-step), plugging $p(\mathbf{z} \mid \mathbf{x}; \theta_{t+1})$ into the ELBO (which makes the ELBO tight), and maximizing the resulting expression over θ. Every step increases the marginal likelihood $\log p(\mathbf{x}; \theta_t)$, which is what we wanted to show.

Properties of EM

Following from the above discussion, EM has the following properties:

1. The marginal likelihood increases after each EM cycle.

2. Since the marginal likelihood is upper-bounded by its true global maximum and it increases at every step, EM must eventually converge.

However, since we are optimizing a non-convex objective, we have no guarantee to find the global optimum. In fact, EM in practice converges almost always to a local optimum. Moreover, that optimum heavily depends on the choice of initialization. Different initial θ_0 can lead to very different solutions, and so it is very common to use multiple restarts of the algorithm and choose the best one in the end. In fact EM is so sensitive to the choice of initial parameters, that techniques for choosing these parameters are still an active area of research.

6.4.2 Illustrative Example: EM in Gaussian Mixture Models

Consider the use of EM in the context of GMMs. Suppose we have a dataset $\mathcal{D}$. In the E-step, we can compute the posterior for each data point $\mathbf{x}$ as follows:

$$p(z \mid \mathbf{x}; \theta_t) = \frac{p(z, \mathbf{x}; \theta_t)}{p(\mathbf{x}; \theta_t)} = \frac{p(\mathbf{x} \mid z; \theta_t) p(z; \theta_t)}{\sum_{k=1}^{K} p(\mathbf{x} \mid z_k; \theta_t) p(z_k; \theta_t)}.$$

Note that each $p(\mathbf{x} \mid z_k; \theta_t)p(z_k; \theta_t)$ is simply the probability that $\mathbf{x}$ originates from component k given the current set of parameters θ. After normalization, these form the K-dimensional vector of probabilities $p(z \mid \mathbf{x}; \theta_t)$.

Recall that in the original model, z is an indicator variable that chooses a component for $\mathbf{x}$. We can view this as a "hard" assignment of $\mathbf{x}$ to one component. The result of the E step is a K-dimensional vector (whose components sum to one) that specifies a "soft" assignment to components. In that sense, we have hallucinated a "soft" instantiation of z. This is what we meant earlier by an "intuitive interpretation" for $p(z \mid \mathbf{x}; \theta_t)$.

At the M-step, we optimize the expected log-likelihood of our model.

$$\begin{aligned}\theta_{t+1} &= \operatorname*{argmax}_{\theta} \sum_{\mathbf{x}\in\mathcal{D}} \mathbb{E}_{z\sim p(z|\mathbf{x};\theta_t)} \log p(\mathbf{x}, z; \theta) \\ &= \operatorname*{argmax}_{\theta} \sum_{k=1}^{K} \sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x}; \theta_t) \log p(\mathbf{x} \mid z_k; \theta) \\ &\qquad + \sum_{k=1}^{K} \sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid x; \theta_t) \log p(z_k; \theta).\end{aligned}$$

We can optimize each of these terms separately. We will start with $p(\mathbf{x} \mid z_k; \theta) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$. We have to find the μ_k, Σ_k that maximize

$$\sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x}; \theta_t) \log p(\mathbf{x} \mid z_k; \theta) = c_k \cdot \mathbb{E}_{\mathbf{x}\sim Q_k(\mathbf{x})} \log p(\mathbf{x} \mid z_k; \theta),$$

where $c_k = \sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x}; \theta_t)$ is a constant that does not depend on θ and $Q_k(\mathbf{x})$ is a probability distribution defined over $\mathcal{D}$ as

$$Q_k(x) = \frac{p(z_k \mid \mathbf{x}; \theta_t)}{\sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x}; \theta_t)}.$$

We know that $\mathbb{E}_{\mathbf{x}\sim Q_k(x)} \log p(\mathbf{x} \mid z_k; \theta)$ is optimized when $p(\mathbf{x} \mid z_k; \theta)$ equals $Q_k(\mathbf{x})$ (as discussed in the section on learning directed models, this objective equals the KL divergence between Q_k and p, plus a constant). Moreover, since $p(\mathbf{x} \mid z_k; \theta) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$ is in the exponential family, it is entirely described by its sufficient statistics. Thus, we can set the mean and variance μ_k, Σ_k to those of $Q_k(\mathbf{x})$, which are

$$\mu_k = \mu_{Q_k} = \sum_{\mathbf{x}\in\mathcal{D}} \frac{p(z_k \mid \mathbf{x}; \theta_t)}{\sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x}; \theta_t)} \mathbf{x}$$

and

$$\Sigma_k = \Sigma_{Q_k} = \sum_{\mathbf{x}\in\mathcal{D}} \frac{p(z_k \mid \mathbf{x};\theta_t)}{\sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x};\theta_t)} (x-\mu_{Q_k})(x-\mu_{Q_k})^T.$$

Note how these are the just the mean and variance of the data, weighted by their cluster affinities! Similarly, we may find out that the class priors are

$$\pi_k = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}\in\mathcal{D}} p(z_k \mid \mathbf{x};\theta_t).$$

Although we have derived these results using general facts about exponential families, it is equally possible to derive them using standard calculus techniques.

Further Reading

- C. M. Bishop. (1998). "Latent Variable Models". In: *Learning in Graphical Models*. Springer. 371–403.
- Chapter 27 in K. P. Murphy. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

6.5 Bayesian Learning

The learning approaches we have discussed so far are based on the principle of maximum likelihood estimation. While being extremely general, there are limitations of this approach as illustrated in the two examples below.

Example 1 Suppose we are interested in modeling the outcome of a biased coin, $X \in \{\text{heads}, \text{tails}\}$. We toss the coin 10 times and observe 6 heads. If θ denotes the probability of observing heads, the maximum likelihood estimate (MLE) is given by

$$\hat{\theta} = \frac{n_{\text{heads}}}{n_{\text{heads}} + n_{\text{tails}}} = 0.6.$$

Now, suppose we continue tossing the coin such that after 100 total trials (including the 10 initial trials), we observe 60 heads. Again, we

can compute the maximum likelihood estimate as

$$\hat{\theta} = \frac{n_{\text{heads}}}{n_{\text{heads}} + n_{\text{tails}}} = 0.6.$$

In both of the above situations, the maximum likelihood estimate does not change as we observe more data. This seems counterintuitive — our *confidence* in predicting heads with probability 0.6 should be higher in the second setting where we have seen many more trials of the coin. The key problem is that we represent our belief about the probability of heads θ as a single number $\hat{\theta}$, so there is no way to represent whether we are more or less sure about θ.

Example 2 Consider a language model for sentences based on the bag-of-words assumption. A bag-of-words model has a generative process where a sentence is formed from a sample of words which are metaphorically "pulled out of a bag" (i.e., sampled independently). In such a model, the probability of a sentence can be factored as the probability of the words appearing in the sentence. For a sentence S consisting of words $w_1, \ldots, w_n$, we have

$$p(S) = \prod_{i=1}^{n} p(w_n).$$

For simplicity, assume that our language corpus consists of a single sentence, "Probabilistic graphical models are fun. They are also powerful." We can estimate the probability of each of the individual words based on the counts. Our corpus contains 10 words with each word appearing once. Thus, each word in the corpus is assigned a probability of 0.1. Now, while testing the generalization of our model to the English language, we observe another sentence, "Probabilistic graphical models are hard." The probability of the sentence under our model is $0.1 \times 0.1 \times 0.1 \times 0.1 \times 0 = 0$. We did not observe one of the words ("hard") during training. This caused our language model to infer that the sentence is impossible, even though it is a perfectly plausible sentence.

Out-of-vocabulary words are a common phenomenon, even for language models trained on large corpus. One of the simplest ways to handle these words is to assign a prior probability of observing an out-of-vocabulary word, such that the model will assign a low but non-zero

probability to test sentences containing such words. As an aside, modern systems commonly use *tokenization*, where a set of fundamental tokens can be combined to form any word. Hence the word "Hello" as a single token and the word "Bayesian" is encoded as "Bay" + "esian" under the common Byte Pair Encoding. This can be viewed as putting a prior over all words, where longer words are less likely.

Modeling Uncertainty in Bayesian Learning

We can address some of the limitations of maximum likelihood estimation by adopting a Bayesian framework. In contrast to maximum likelihood learning, Bayesian learning explicitly models uncertainty over both the observed variables $\mathbf{X}$ and the parameters θ. In other words, the parameters θ are random variables as well.

A prior distribution over the parameters $p(\theta)$ encodes our initial beliefs. These beliefs are subjective. For example, we can choose the prior over θ for a biased coin to be uniform between 0 and 1. If we expect the coin to be fair, the prior distribution can be peaked around $\theta = 0.5$. We will discuss commonly used priors later in this chapter.

Say we observe the dataset $\mathcal{D} = \{x_1, \cdots, x_N\}$ (e.g., in the coin toss example, each X_i is the outcome of one toss of the coin). We can update our beliefs using Bayes' rule

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{p(\mathcal{D})},$$

where

$$\underbrace{p(\theta \mid \mathcal{D})}_{\text{posterior}} \propto \underbrace{p(\mathcal{D} \mid \theta)}_{\text{likelihood}} \underbrace{p(\theta)}_{\text{prior}}.$$

Thus, Bayesian learning provides a principled mechanism for incorporating prior knowledge into our model. Bayesian learning is useful in many situations, such as when want to provide uncertainty estimates about the model parameters (Example 1) or when the data available for learning a model is limited (Example 2).

This framework is flexible in how much influence we place on prior knowledge versus the observed data. When we do not have much prior

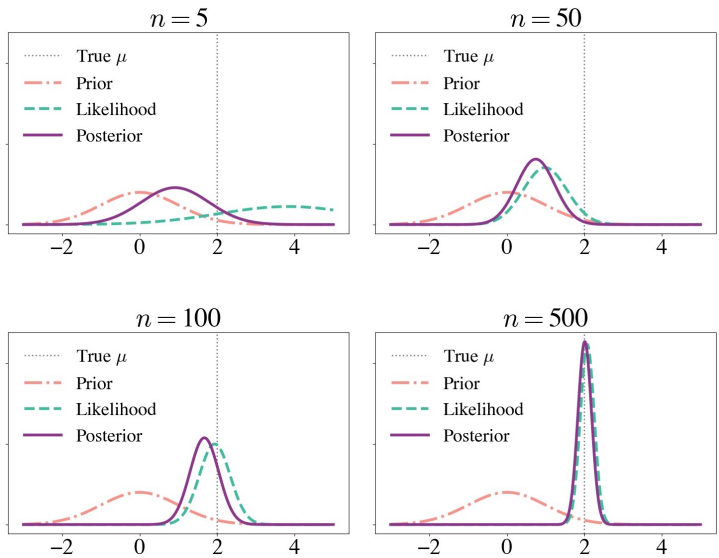


Figure 6.9: As the number of data samples n increases, the influence of the prior on our posterior diminishes. Meanwhile, the influence of the likelihood comes to dominate. In this example, our data are drawn from a Gaussian distribution with a true mean parameter (μ) of 2.

knowledge or certainty about θ, we can choose a weak prior or uninformative prior. When we are very confident in our domain expertise, we can select a strong informative prior. Further, the prior $p(\theta)$ is independent of $\mathcal{D}$, while the likelihood changes as the sample size n grows (Figure 6.9). When n is small, the posterior is heavily influenced by the prior because the likelihood is relatively weak. As n increases, the likelihood becomes sharper and more concentrated around the true parameter value. Consequently, the influence of the prior on the posterior diminishes as n increases, as the likelihood function incorporates increasingly more information about the data.

6.5.1 Conjugate Priors

When calculating the posterior distribution using Bayes' rule, it should be pretty straightforward to calculate the numerator. However, calcu-

LIKELIHOOD	PRIOR & POSTERIOR
Bernoulli	Beta
Binomial	Beta
Geometric	Beta
Poisson	Gamma
Categorical	Dirichlet
Multinomial	Dirichlet

Table 6.2: Conjugate priors for common discrete likelihood distributions.

lating the denominator $p(\mathcal{D})$ requires that we compute the integral

$$p(\mathcal{D}) = \int_{\theta} p(\mathcal{D} \mid \theta) p(\theta) d\theta.$$

This might cause us trouble, since integration is often difficult. Computing this integral might be feasible for very simple problems. However, computing integrals can be challenging when θ is high-dimensional.

To tackle this issue, people have observed that for some choices of prior $p(\theta)$, the posterior distribution $p(\theta \mid \mathcal{D})$ can be directly computed in closed form. This computational convenience is known as a *conjugate prior*, where *conjugacy* is formally defined as follows.

Definition 6.1 (Conjugacy, Gelman *et al.* 1995)**.** Let $\mathcal{F}$ denote a class of sampling distributions $p(\mathcal{D} \mid \theta)$. Let $\mathcal{P}$ denote a class of prior distributions for θ. We say that $\mathcal{P}$ is conjugate for $\mathcal{F}$ if

$$p(\theta \mid \mathcal{D}) \in \mathcal{P} \text{ for all } p(\cdot \mid \theta) \in \mathcal{F} \text{ and } p(\cdot) \in \mathcal{P}.$$

When the prior and posterior are *conjugate distributions* with respect to the likelihood, we can evade intractable numerical integrations. Table 6.2 lists conjugate priors for some common likelihood distributions.

We can return to the coin toss example where we are given a sequence of N coin tosses

$$\mathcal{D} = \{x_1, \ldots, x_N\}$$

and we want to infer the probability of getting heads (θ) using Bayes' rule. Suppose we choose the prior $p(\theta)$ as the Beta distribution defined

by

$$p(\theta) = \mathsf{Beta}(\theta \mid \alpha_H, \alpha_T) = \frac{\theta^{\alpha_H - 1}(1-\theta)^{\alpha_T - 1}}{B(\alpha_H, \alpha_T)}$$

where α_H and α_T are the two parameters that determine the shape of the distribution (similar to how the mean and variance determine a Gaussian distribution), and $B(\alpha_H, \alpha_T)$ is some normalization constant that ensures $\int p(\theta)d\theta = 1$. We will go into more details about the Beta distribution later. What matters here is that the Beta distribution has a very special property: the posterior $p(\theta \mid \mathcal{D})$ is always another Beta distribution (but with different parameters). More concretely, out of N coin tosses, if the number of heads and the number of tails are N_H and N_T respectively, then it can be shown that the posterior is

$$\begin{aligned} p(\theta \mid \mathcal{D}) &= \mathsf{Beta}(\theta \mid \alpha_H + N_H, \alpha_T + N_T) \\ &= \frac{\theta^{N_H + \alpha_H - 1}(1-\theta)^{N_T + \alpha_T - 1}}{B(N_H + \alpha_H, N_T + \alpha_T)}, \end{aligned}$$

which is another Beta distribution with parameters $(\alpha_H + N_H, \alpha_T + N_T)$. In other words, if the prior is a Beta distribution (i.e., we can represent it as two numbers α_H, α_T) then the posterior can be immediately computed by a simple addition $\alpha_H + N_H, \alpha_T + N_T$. Conveniently, there is no need to compute the complex integral $p(\mathcal{D})$.

The Beta Distribution

Now we try to understand the Beta distribution better. If θ has distribution $\mathsf{Beta}(\theta \mid \alpha_H, \alpha_T)$, then the expected value of θ is

$$\frac{\alpha_H}{\alpha_H + \alpha_T}.$$

Intuitively, α_H is larger than α_T if we believe that heads are more likely. The variance of the Beta distribution is the somewhat complex expression,

$$\frac{\alpha_H \alpha_T}{(\alpha_H + \alpha_T)^2(\alpha_H + \alpha_T + 1)},$$

but we remark that (very roughly) the numerator is quadratic in α_H, α_T while the denominator is cubic in α_H, α_T. Hence if α_H and α_T increases then the variance decreases, so we are more certain about the value of

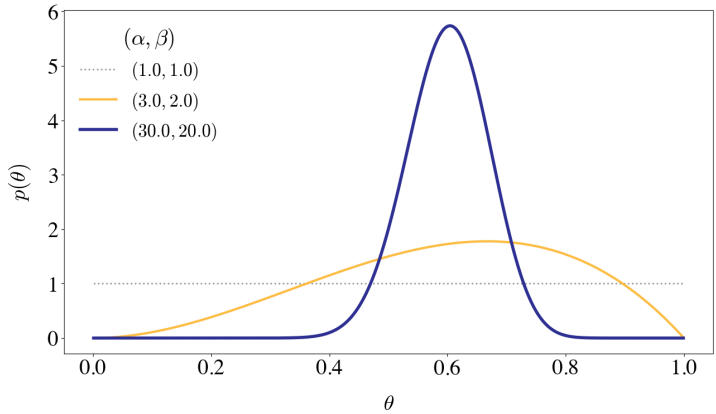


Figure 6.10: The expectation of both $\mathsf{Beta}(3, 2)$ and $\mathsf{Beta}(30, 20)$ are 0.6, but $\mathsf{Beta}(30, 20)$ is much more concentrated. This can be used to represent different levels of uncertainty in θ.

θ (Figure 6.10). We can use this observation to better understand the posterior update rule: after observing more data $\mathcal{D}$, the prior parameters α_H and α_T increase by N_H and N_T, respectively. Thus, the variance of $p(\theta \mid \mathcal{D})$ should be smaller than the variance of $p(\theta)$ (i.e., we are more certain about the value of θ after observing data $\mathcal{D}$).

What we have shown here is that the Beta distribution family is a conjugate prior to the Bernoulli distribution family. Relating this back to the example above, if $p(\theta)$ is a Beta distribution and $p(\mathcal{D} \mid \theta)$ is a Bernoulli distribution, then $p(\theta \mid \mathcal{D})$ is still a Beta distribution. In general, we usually have a simple algebra expression to compute $p(\theta \mid \mathcal{D})$ (such as computing $\alpha_H + N_H, \alpha_T + N_H$ in the example above).

The Categorical Distribution

We now introduce another example of a conjugate prior, which generalizes the Bernoulli example above. Instead of being limited to binary outcomes, we can now consider the categorical distribution (think of a K-sided dice). Let $\mathcal{D} = \{x_1, \ldots, x_N\}$ be N rolls of the dice, where $x_j \in \{1, \ldots, K\}$ is the outcome of the j-th roll. The parameter of the

categorical distribution is denoted by

$$\theta = (\theta_1, \cdots, \theta_K) := (p(X_j = 1), \ldots, p(X_j = K))$$

where $\sum_{k=1}^{K} \theta_k = 1$.

We claim that the Dirichlet distribution is a conjugate prior for the categorical distribution. A Dirichlet distribution is defined by K parameters $\alpha = (\alpha_1, \ldots, \alpha_K)$, and its PDF is given by

$$p(\theta) = \mathsf{Dirichlet}(\theta \mid \alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$$

where $B(\alpha)$ is still a normalization constant.

To show that the Dirichlet distribution is a conjugate prior for the categorial distribution, we need to show that the posterior is also a Dirichlet distribution. To calculate the posterior $p(\theta \mid \mathcal{D})$ with Bayes' rule, we first calculate the likelihood $p(\mathcal{D} \mid \theta)$ as

$$p(\mathcal{D} \mid \theta) = \prod_{k=1}^{K} \theta_k^{\sum_{j=1}^{N} 1\{X_j = k\}}.$$

To simplify the notation we denote $N_k = \sum_{j=1}^{N} 1\{X_j = k\}$ as the number of times we roll out k, so $p(\mathcal{D} \mid \theta) = \prod \theta_k^{N_k}$. Using this new notation, the posterior can be calculated as

$$\begin{aligned} p(\theta \mid \mathcal{D}) &\propto p(\mathcal{D} \mid \theta) p(\theta) \\ &\propto \prod_{k=1}^{K} \theta_k^{N_k + \alpha_k - 1} \\ &:= \mathsf{Dirichlet}(\theta \mid \alpha_1 + N_1, \cdots, \alpha_K + N_K). \end{aligned}$$

In other words, if the prior is a Dirichlet distribution with parameter $(\alpha_1, \cdots, \alpha_K)$, then the posterior $p(\theta \mid \mathcal{D})$ is a Dirichlet distribution with parameters $(\alpha_1 + N_1, \cdots, \alpha_K + N_K)$. In example 2 above, we added a prior probability to observing an out-of-vocabulary word. We can see that this corresponds exactly to choosing a prior with nonzero prior $\alpha = \alpha_1 = \ldots = \alpha_K$.

Some Concluding Remarks

Many distributions have conjugate priors. In fact, any exponential family distribution has a conjugate prior. Although conjugacy seemingly solves the problem of computing Bayesian posteriors, there are two caveats:

1. Usually practitioners will want to choose the prior $p(\theta)$ to best capture their knowledge about the problem, and using conjugate priors is a strong restriction.

2. For more complex distributions, the posterior computation is not as easy as those in our examples. There are distributions for which the posterior computation is still NP-hard.

Conjugate priors are powerful tools used in many real-world applications and popular models, such as topic modeling (e.g., latent Dirichlet allocation (Blei *et al.*, 2003)) and generalized linear models. However, their limitations should be considered when modeling tasks demand richer or more-nuanced prior structures.

Further Reading

- A. Gelman *et al.* (1995). *Bayesian Data Analysis*. Chapman and Hall/CRC.
- D. Barber. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.
- K. P. Murphy. (2022). *Probabilistic Machine Learning: An Introduction*. MIT Press.