Learning in directed models

We now turn our attention to the third and last part of the course: *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: the graph structure, and the parameters of the factors induced by this graph. These components lead to two different learning tasks:

> *Parameter learning*, where the graph structure is known and we want to estimate the factors.
>
> *Structure learning*, where we want to estimate the graph, i.e. determine from data how the variables depend on each other.

We are going to first focus on parameter learning, and come back to structure learning in a later chapter.

We will consider parameter learning in both directed and undirected models. It turns out that the former will admit easy, closed form solutions, while the latter will involve potentially intractable numerical optimization techniques. In later sections, we will look at latent variable models (which contain unobserved hidden variables that succinctly model the event of interest) as well as Bayesian learning, a general approach to statistics that offers certain advantages to more standard approaches.

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 D of m 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 M, and our task is to learn some "good" model in M that defines a distribution p. For example, we could look at all Bayes nets with a given graph structure, with all the possible choices of the CPD tables.

The goal of learning is to return a model that precisely captures the distribution p^* from which our data was sampled. This is in general not achievable because limited data only provides a rough approximation of the true underlying distribution, and because of computational reasons. 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:

> 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, which we have seen when we covered variational inference:

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

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

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

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 $\mathbb{E}_{x\sim p^*}[\log p(x)]$ with the empirical log-likelihood (a Monte-Carlo estimate):

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

where D is a dataset drawn i.i.d. from p^*.

Maximum likelihood learning is then defined as

$$\max_{p\in M} \frac{1}{|D|} \sum_{x\in D} \log p(x),$$

An 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/tails 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 M is going to be the set of all probability distributions over $\{h, t\}$.

How should we choose p from 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 is $D = \{h, h, t, h, t\}$, our likelihood becomes $\prod_i p(x_i) = \theta \cdot \theta \cdot (1-\theta) \cdot \theta \cdot (1-\theta)$; maximizing this yields $\theta = 60\%$.

More generally, our log-likelihood function is simply

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

for which the optimal solution is

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

Likelihood, Loss and Risk

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

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

Notice that the loss function which corresponds to maximum likelihood estimation is the log loss $-\log p(x)$.

Another example of a loss is the conditional log-likelihood. Suppose we want to predict a set of variables y given x, e.g., for segmentation or stereo vision. We concentrate on predicting $p(y|x)$, and use a conditional loss function $L(x, y, p) = -\log p(y \mid x)$. Since the loss function only depends on $p(y \mid x)$, it suffices to estimate the conditional distribution, not the joint. This is the objective function we use to train conditional random fields (CRFs).

Suppose next that our ultimate goal is structured prediction, i.e. given x we predict y via $\arg\max_y p(y \mid x)$. What loss function should we use to measure error in this setting?

One reasonable choice would be the classification error:

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

which is the probability over all (x, y) pairs sampled from p^* that we predict the wrong assignment. 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*.

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.

Empirical Risk and Overfitting

Empirical risk minimization can easily overfit the data. The data we have is a sample, and usually there is vast amount of samples that we have never seen. Our model should generalize well to these "never-seen" samples.

The bias/variance tradeoff

Thus, 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 p^*, even with unlimited data. This type of limitation is called bias, as the learning is limited on how close it can approximate the target distribution

If we select a highly expressive hypothesis class, we might represent better the data. However, when we have small amount of data, multiple models can fit well, or even better than the true model. Moreover, small perturbations on D will result in very different estimates. This limitation is call the variance.

Thus, there is an inherent bias-variance trade off when selecting the hypothesis class. 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.

How to avoid overfitting?

High bias can be avoided by increasing the capacity of the model. We may avoid high variance using several approaches.

We may impose hard constraints, e.g. by selecting a less expressive hypothesis class: Bayesian networks with at most d parents or pairwise (rather than arbitrary-order) MRFs. We may also introduce a soft preference for "simpler" models by adding a regularizer term $R(p)$ to the loss $L(x, p)$, which will penalize overly complex p.

Generalization error

At training, we minimize empirical loss

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

However, we are actually interested in minimizing

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

We cannot guarantee with certainty the quality of our learned model. This is because the data 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 D, the learning procedure returns a model whose error is low. There exist a vast literature that quantifies the probability of observing a given error between the empirical and the expected loss given a particular type of model and a particular dataset size.

Maximum likelihood learning in Bayesian networks

Let us now apply this long discussion to a particular problem of interest: parameter learning in Bayesian networks.

Suppose that we are given a Bayesian network $p(x) = \prod_{i=1}^{n} \theta_{x_i | pa(x_i)}$ and i.i.d. samples $D = \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$. What is the maximum likelihood estimate of the parameters (the CPDs)?

We may write the likelihood as

$$L(\theta, D) = \prod_{i=1}^{n} \prod_{j=1}^{m} \theta_{x_i^{(j)}|pa(x_i^{(j)})}$$

Taking logs and combining like terms, this becomes

$$\log L(\theta, D) = \sum_{i=1}^{n} \#(x_i, pa(x_i)) \cdot \log(\theta_{x_i|pa(x_i)}).$$

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 we saw earlier (except with more categories). It's a simple calculus exercise to formally show that

$$\theta^*_{x_i|pa(x_i)} = \frac{\#(x_i, pa(x_i))}{\#(pa(x_i))}.$$

We thus conclude that in Bayesian networks with discrete variables, the maximum-likelihood estimate has a closed-form solution. Even when the variables are not discrete, the task is equally simple: the log-factors are linearly separable, 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.