Reports on PRML Reading Talk

Siyu Wang
Department of Computer Science
Tsinghua University
thuwangsy@gmail.com

1 Introduction to ML, Probability Basics

PRML chap.1,2; MLAPP chap.1,2

The key purpose of machine learning is to extract good **features** and **patterns** from **data**. To realize this purpose, we mainly use function fitting as the tool to solve this problem. In this section we will focus on a polynomial fitting problem, solving it from different angles and considering differences and relations among them.

But we must keep in mind that, function fitting is only a mathematical tool, not our purpose.

Consider the following problem: given some observed points:

$$D = \{(x_1, t_1), (x_2, t_2), \cdots, (x_N, x_N)\}$$

in which, $t = \sin(2\pi x) + \mathcal{N}(0, \sigma^2)$. Then we want to find a polynomial function

$$t = y(x, \mathbf{w}) = \mathbf{w_0} + \mathbf{w_1 x} + \mathbf{w_2 x^2} + ... + \mathbf{x_M x^M} \tag{1.1}$$

to fit these points, so when given new values of x, we can predict the corresponding t.

From function fitting angle

We want to optimize the unknown parameter w in equation.1.1 by minimize the *error function*

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathrm{w}) - t_n\}^2. \tag{1.2}$$

So we can set the error function's derivative with respect to **w** to zero and easily get the optimal parameter $\mathbf{w}^*$.

To reduce the over-fitting problem, we may make a little change about the error function by adding a regularization item:

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathrm{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathrm{w}\|^2. \tag{1.3}$$

From probabilistic angle

Here, we shall assume that, given the value of x, the corresponding value of t has a Gaussian distribution with a mean equal to the value $y(x, \mathrm{w})$ and a fixed variance β. So we have

$$p(t|x, \mathrm{w}, \beta) = \mathcal{N}(y(x, \mathrm{w}), \beta^{-1}) \tag{1.4}$$

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

MLE: maximum likelihood estimation

Given N input value $\mathbf{x} = (x_1, \cdots, x_N)^T$ and their corresponding target value $\mathbf{t} = (t_1, \cdots, t_N)^T$, we have the likelihood function with the form as

$$p(\mathbf{t}|\mathbf{x}, \mathrm{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(x_n, \mathrm{w}), \beta^{-1}). \tag{1.5}$$

It's convenient to maximize the logarithm of the likelihood function

$$\ln p(\mathbf{t}|\mathbf{x}, \mathrm{w}, \beta) = -\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathrm{w}) - t_n\}^2 + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi). \tag{1.6}$$

So when determining $\mathbf{w}$, maximizing likelihood function is equal to minimizing the error function in equation.1.2. And we obtain

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathrm{w}_{ML}) - t_n\}^2. \tag{1.7}$$

So we get a probability distribution of t when given a new value of x

$$p(t|x, \mathrm{w}_{ML}, \beta_{ML}) = \mathcal{N}(t|y(x, \mathrm{w}_{ML}, \beta_{ML}) \tag{1.8}$$

MAP: maximum a posteriori estimation

Now we take a step towards a more Bayesian approach and introduce a prior distribution over the polynomial coefficient w and for simplicity, consider a Gaussian distribution of the form

$$p(\mathrm{w}|\alpha) = \mathcal{N}(\mathrm{w}|0, \alpha^{-1}\mathrm{I}) = (\frac{\alpha}{2\pi})^{(M+1)/2}\exp\{-\frac{\alpha}{2}\|w\|^2\} \tag{1.9}$$

where α is the precision of the distribution and $M + 1$ is the total number of elements in the vector $\mathbf{w}$. Using Bayes' theorem, we can get the posterior distribution for $\mathbf{w}$, which is proportional to the product of the prior distribution and the likelihood function

$$p(\mathrm{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{t}|\mathrm{w}, \mathbf{x}, \beta)p(\mathrm{w}|\alpha). \tag{1.10}$$

And finally we find that the maximum of the posterior is given by the minimum of

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathrm{w}) - t_n\}^2 + \frac{\alpha}{2}\mathrm{w}^T\mathrm{w}. \tag{1.11}$$

Thus we see that maximizing the posterior distribution is equivalent to minimizing the regularized sum-of-square error function in the form (1.3)

Inference and decision

Here, we come to a classification problem and we have three different approaches to doing inference and decision.

1. First solve $p(\mathrm{x}|\mathcal{C}_k)$ as well as $p(\mathcal{C}_k$ for each class individually, then figure out $p(\mathcal{C}_k|\mathrm{x}) = \frac{p(\mathrm{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathrm{x})}$ and $p(\mathrm{x})$ can be gotten from $p(\mathrm{x}) = \sum_k p(\mathrm{x}|\mathcal{C}_k)p(\mathcal{C}_k)$. This is equivalent to model the joint distribution $p(\mathrm{x}, \mathcal{C}_k)$
2. Determine posterior class probabilities $p(\mathcal{C}_k|\mathrm{x})$ and then use decision theory. *discriminative model.*
3. Find $f(\mathrm{x})$ which maps x directly to a class label.

The first approach is too time-consuming and demanding, while the last one is simplest but not so robust as approach.2 because we can see a lot of information from the posterior probabilities.

For others

Other than the main ideas and problems discussed above, there are still some other small items here: *Gaussian probability distribution, exponential family, students' t distribution, conjugate prior,* and so on. For more details about these, we can refer to the original book.

2 Statistics, Information Theory Basics

MLAPP Chap.5,6

Bayesian Statistics

Bayes' theorem:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{1}$$

Priors are assumptions about values of parameters and we can choose an appropriate prior from many different types: *conjugate prior, uninformative prior, robust prior, Jeffreys prior*.

Jeffreys prior

Given a fixed likelihood $p(\mathrm{x}|\phi)$ where x represents the data, ϕ represents parameters. Then we define Fisher information as the form:

$$I_\phi(\phi) = -\mathrm{E}_{\mathrm{X}\sim p(\mathrm{x}|\phi)}[(\frac{\mathrm{d}\log p(\mathrm{X}|\phi)}{\mathrm{d}\phi})^2] \tag{2}$$

and Jeffreys prior is

$$p(\phi) \propto (I_\phi(\phi))^{1/2} \tag{3}$$

An important property of Jeffreys prior is that if we assume $\theta = h(\phi)$, then the prior of θ is also Jeffreys prior:

$$p(\theta) \propto (I_\theta(\theta))^{1/2} \tag{4}$$

And we need to notice that the mixtures of conjugate priors is still a conjugate prior.

Hierarchical Bayes

We can change a 1-level model with the form as:

$$\theta \to D \tag{5}$$

into a 2-level one by adding a parameter over the prior of θ, then we have

$$\eta \to \theta \to D \tag{6}$$

If we apply maximum likelihood respectively to these two models, we can obtain

$$1-\text{levelmodel}: \theta^* = \arg\max_\theta p(D|\theta) \tag{7}$$

$$2-\text{levelmodel}: \eta^* = \arg\max_\eta p(D|\eta) = \arg\max_\eta [\int p(D|\theta)p(\theta|\eta)\mathrm{d}\theta] \tag{8}$$

. The maximum likelihood for 2-level model is also called **Empirical Bayes**.

Bayesian decision theory

Frequentist Statistics

In this part, we see inference from another angle, avoiding treating parameters like random variables, avoiding the use of priors and starting from sampling distribution.

Given a distribution $p(x|\theta^*)$ where θ is fixed and we have data $D = (x_1, x_2, \ldots, x_n)$ where $x_i \sim p(x|\theta^*)$ and are i.i.d. We want to obtain an estimator $\hat{\theta} = \delta(D)$, which is a function of data to estimate the true parameters θ.

3 Linear models(1)

MLAPP Chap.7; PRML Chap.3

In this section, we mainly want to solve the regression problem using a linear model from MLE angle and MAP angle. **Problem setting:** Consider outputs is a linear combination of the inputs

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \epsilon = \sum_{j=1}^{d} w_j x_j + \epsilon. \tag{9}$$

And we can rewrite the model in a probabilistic form:

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(y|\mu(\mathbf{x}), \sigma^2(\mathbf{x})) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x}, \sigma^2(\mathbf{x})) \tag{10}$$

To extend this model linear model, we introduce basis functions, and now the model is

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(y|\mathbf{w}^T\phi(\mathbf{x}), \sigma^2(\mathbf{x})) \tag{11}$$

where ϕ is a set of functions and $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_M\mathbf{x})$. And in most cases we assume that $\sigma^2(\mathbf{x}) = \sigma^2$

Maximum Likelihood Estimator

In this view, we want to find a set of parameters θ so that likelihood is maximized:

$$\hat{\theta} = \arg\max_{\theta} \log p(\mathcal{D}|\theta). \tag{12}$$

And we usually consider the inputs and outputs are i.i.d, so we can obtain

$$L(\theta) = \log p(\mathcal{D}|\theta) = \log p(\mathbf{y}|\mathbf{x}, \theta) = \sum_{i=1}^{D} \log p(y_i|\mathbf{x}_i, \theta). \tag{13}$$

Then we replace $p(y_i|\mathbf{x}_i, \theta)$ by Gaussian distribution, set the derivative with respect to $\mathbf{w}$ to zero and we can obtain

$$\hat{\mathbf{w}}_{\text{MLE}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{14}$$

Ans we can understand this method from geometry view.

There is usually a serious over-fitting problem in MLE so we introduce a regularization item in the object function:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|^2, \tag{15}$$

and

$$\hat{\mathbf{w}}_{\text{ridge}} = (\lambda \mathrm{I}_d + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{16}$$

Maximum A Posterior – Bayesian Linear Regression

Using Bayes' theorem, we obtain

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta) \tag{17}$$

The conjugate prior corresponding to Gaussian likelihood is still Gaussian, so we assume the prior of $\mathbf{w}$ has a Gaussian form:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{V}_0). \tag{18}$$

So

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}, \sigma^2) \propto \mathcal{N}(\mathbf{w}|\mathbf{w}_0, \mathbf{V}_0)\mathcal{N}(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma^2\mathbf{I}_N) = \mathcal{N}(\mathbf{w}|\mathbf{w}_N, \mathbf{V}_N) \tag{19}$$

And if we choose $\mathbf{V}_0 = \alpha^{-1}\mathbf{I}$ and $\alpha \to 0$, we will see $\hat{\mathbf{w}}_{\text{MLE}} = \mathbf{w}_N$

Bayesian Model Selection

$$\text{Bayesianfactor} := \frac{p(\mathcal{D}|\mathcal{M}_i)}{p(\mathcal{D}|\mathcal{M}_j)} \tag{20}$$

A model $\mathcal{M}_i$ is controlled by a set of parameters $\mathbf{w}$, and

$$p(\mathcal{D}|\mathcal{M}_i) = \int p(\mathcal{D}|\mathbf{w}, \mathcal{M}_i) p(\mathbf{w}|\mathcal{M}_i) \mathrm{d}\mathbf{w}. \tag{21}$$

4 Linear models(2)

MLAPP Chap.8; PRML Chap.4

Logistic Regression

Using Bayes Theorem we have

$$p(C_1|\mathrm{x}) = \frac{p(\mathrm{x}|C_1)p(C_1)}{p(\mathrm{x})} = \frac{p(\mathrm{x}|C_1)p(C_1)}{p(\mathrm{x}|C_1)p(C_1) + p(\mathrm{x}|C_2)p(C_2)} = \sigma(a) \tag{22}$$

where $a = \log \frac{p(\mathrm{x}|C_2)p(C_2)}{p(\mathrm{x}|C_1)p(C_1)}$.

Suppose $p(\mathrm{x}|C_k)$ is Gaussian, then we can obtain $a = \mathrm{w}^T\mathrm{x} + \mathrm{w}_0$ and this is very simple to compute $p(C_1|\mathrm{x})$.

Maximum Conditional Likelihood Estimator.

$$\hat{\mathrm{w}} = \arg\max_{\mathrm{x}} \prod_{i=1}^{N} p(y_i|\mathrm{x}_i, \mathrm{w}). \tag{23}$$

Take the logarithm form to be likelihood function and we have

$$L(\mathrm{w}) = -\log \prod_{i=1}^{N} p(y_i|\mathrm{x}_i, \mathrm{w}) = -\sum_{i=1}^{N} \log(p(y_i|\mathrm{x}_i, \mathrm{w})) \tag{24}$$

Which is not solvable analytically, so we have no closed-form solution and we need some descent algorithms to minimize $L(\mathrm{w})$. **Gradient descent** In each iteration, we update the parameter by taking a step backwards the direction which the gradient points to

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{\tau} - \eta \nabla_{\mathrm{w}} L(\mathrm{w}) \tag{25}$$

Bayesian Logistic Regression

Estimate the posterior of parameters w rather than just a point estimation.

Generative model: learn a model of the **joint probability**, $p(x, y)$, of the inputs x and the label y, and make their predictions by using Bayes rules to calculate $p(ylx)$, and then picking the most likely label y.
Discriminative model: learn the posterior $p(ylx)$ directly, or learn a direct map from inputs x to the class labels.

5 Kernels

MLAPP Chap.14; PRML Chap.6,7

A big difference between this chapter and previous ones is that here, the training data points or s subset of them are kept during the prediction phase rather than are discarded. and predictions for new inputs are based purely on the learned parameter vector $\mathbf{w}$.

radial basis functions: depend only on the magnitude of the distance between the arguments so that

$$k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|). \tag{26}$$

Gram matrix is a matrix with kernel values as its elements. And a Gram matrix is Mercer when it is non-negative definite.

5.1 Construction of kernels

One approach is to choose a feature space mapping $\phi(\mathbf{x})$ and then use it to find the corresponding kernel.

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x})\phi_i(\mathbf{x}'). \tag{27}$$

But, an alternative approach is to construct kernel functions directly, at the same time, we should ensure it is valid. A necessary and sufficient condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a valid kernel is that the Gram matrix $\mathbf{K}$ whose elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$, should be positive semi-definite for all possible choices of the set $\{\mathbf{x}_n\}$.

There are a series of techniques to construct valid kernels out of simpler kernels. A commonly used kernel takes the from:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/2\sigma^2) \tag{28}$$

which is called Gaussian kernel. And another powerful approach to constructing kernels starts from a probabilistic generative model:

$$k(\mathbf{x}, \mathbf{x}') = p(\mathbf{x})p(\mathbf{x}'). \tag{29}$$

5.2 Radial Basis Function Networks

Expressing $f(\mathbf{x})$ as a linear combination of radial basis functions, one centred on every data point

$$f(\mathbf{x}) = \sum_{n=1}^{N} w_n h(\|\mathbf{x} - \mathbf{x}_n\|) \tag{30}$$

and we can use least square s to find $\{w_n\}$.

Nadaraya-Waatson model

Assume a Parzen density estimator to model the joint distribution $p(\mathbf{x}, t)$ so that :

$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^{N} f(\mathbf{x} - \mathbf{x}_n, t - t_n) \tag{31}$$

$f(\cdot)$ is the component density function. And finally we can obtain :

$$k(\mathbf{x}, \mathbf{x}_n) = \frac{g(\mathbf{x} - \mathbf{x}_n)}{\sum_m g(\mathbf{x} - \mathbf{x}_m)} \tag{32}$$

where $g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x}, t)\mathrm{d}t$ with a property that gives more weight to the data points $\mathbf{x}_n$ that are close to $\mathbf{x}$.

5.3 Sparse kernel

Here we shall look at kernel-based algorithms that have sparse solutions, so that predictions for new inputs depend only on the kernel function evaluated at a subset of the training data points.

SVM is a decision machine and so does not provide posterior probabilities. It has a property that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum.

Maximum Margin Classifiers

Margin is defined as the smallest distance between the decision boundary and any of the samples. The idea of SVM is to choose a boundary that maximize margin. Here, when the data set if not linearly seperable in the original space, we can project them into another feature space using basis functions $\phi(\cdot)$ to make them linearly separable.

6 Graphical Models

MLAPP Chap.10,19; PRML Chap 8.1-8.3

We shall find it highly advantageous to augment the analysis using diagrammatic representations of probability distributions, called *probabilistic graphical models*. These offer several useful properties:

1. They provide a simple way to visualize the structure of a probabilistic model and can be used to design and motivate new models.
2. Insights into the properties of the model, including conditional independence properties, can be obtained by inspection of the graph.
3. Complex computations, required to perform inference and learning in sophisticated models, can be expressed in terms of graphical manipulations, in which underlying mathematical expressions are carried along implicitly.

We have random variables as *nodes* and probabilistic relationships between variable as *links*.

Then we focus on *Bayesian Networks*. First we can decomposite a joint probability distribution in the form

$$p(a, b, c) = p(c|a, b)p(b|a)p(a). \tag{33}$$

Then we can extend to K variables:

$$p(x_1, \ldots, x_K) = p(x_K|x_1, \ldots, x_{K-1}) \ldots p(x_2|x_1)p(x_1). \tag{34}$$

And the corresponding directed graph is called *fully connected* because there is a link between every pair of nodes. And we can write it in a simple form

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k) \tag{35}$$

where $\mathrm{pa}_k = \mathrm{pa}(k)$ denotes the set of parents of node x_k.

The directed graphs that we are considering are subject to an important restriction namely that there must be no *directed cycles*, in other words there are no closed paths within the graph such that we can move from node to node along links following the direction of the arrows and end up back at the starting node. Such graphs are also called *directed acyclic graphs*, or *DAGs*. This is equivalent to the statement that there exists an ordering of the nodes such that there are no links that go from any node to any lower numbered node.

There are three basic structures in Bayesian networks: *head-to-head, head-to-tail, tail-to-tail*, and the example can be seen in 1

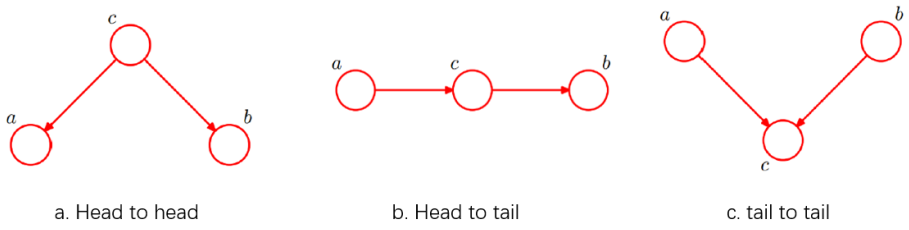


Figure 1: Basic structures of Bayesian networks

Then, we can do d-separation on a DAG. Suppose $\mathcal{C}$ is a set of random variables, a and b are two variables that are not in $\mathcal{C}$. We call $\mathcal{C}$ 'd-separate' a and b iff for any route from a to b:

- if any node v is on the route is head-to-head or head-to-tail, $v \in \mathcal{C}$.
- if any node v on the route is tail-to-tail, $v \notin \mathcal{C}$ and $\mathrm{de}(v) \notin \mathcal{C}$ where $\mathrm{de}(v)$ denotes the set of descendants of node v.

7 Latent Variable Model, Clustering and EM

MLAPP Chap.11,25; PRML Chap.9

In this section, We use mixture models to cluster data, as well as build more complex probability distributions. Therefore we begin our discussion of mixture distributions by considering the problem of finding clusters in a set of data points. Then we introduce the latent variable view of mixture distributions in which the discrete variables can be interpreted as defining assignments of data points to specific component of the mixture. Finally we discuss EM in some generality.

7.1 K-means

Suppose we have input data set $\{\mathrm{x}_1, \cdots, \mathrm{x}_N\}$ consisting of N observations of a random $D-$dimensional Euclidean variable x. Our goal is to partition the data set into some number K of clusters. Firstly we consider a fixed value of K. We can formalize this notion by first introducing a set of $D-$dimensional vectors μ_k, where $k = 1, \cdots, K$, in which μ_k is a prototype associated with the k^{th} cluster. For each data point x_n, we introduce a corresponding set of binary indicator variables $r_{nk} \in \{0, 1\}$, describing which cluster x_n is assigned to. We can then define an objective function

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathrm{x}_n - \mu_k\|^2. \tag{36}$$

Firstly we choose some initial values for the μ_k. Then in the first phase we minimize J with respect to the r_{nk}, keeping the μ_k fixed. In the second phase we minimize J with respect to the μ_k, keeping r_{nk} fixed. This two-stage optimization is then repeated until convergence. And we obtain

$$r_{nk} = \begin{cases} 1 & \text{if} k = \mathrm{argmin_j} \|\mathrm{x}_n - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \tag{37}$$

$$\mu_k = \frac{\sum_n r_{nk} \mathrm{x}_n}{\sum_n r_{nk}} \tag{38}$$

A direct implementation of the $K-$means algorithm as discussed here cna be slow because in each E step it is necessary to compute the Euclidean distance between every prototype vector and every data point. Various schemes have been proposed for speeding up this algorithm.

7.2 Mixture of Gaussian

The Gaussian mixture distribution can be written as linear superposition of Gaussians in the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k). \tag{39}$$

Then we can introduce a $K-$dimensional binary random variable $\mathbf{z}$ having a 1-of-K representation in which a particular element z_k is equal to 1 and all other elements are equal to 0. So $z_k \in \{0, 1\}$, $\sum_k z_k = 1$, and $p(z_k = 1) = \pi_k$. So

$$p(\mathrm{z}) = \prod_{k=1}^{K} \pi_k^{z_k}. \tag{40}$$

So the marginal distribution of x is then obtained:

$$p(\mathrm{x}) = \sum_{\mathrm{z}} p(\mathrm{z}) p(\mathrm{x}|\mathrm{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathrm{x}|\mu_k, \Sigma_k) \tag{41}$$

And we can gain the conditional probability of z given x using Bayes' theorem

$$\gamma(z_k) \equiv p(z_k = 1|\mathrm{x}) = \frac{p(z_k = 1)p(\mathrm{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\mathrm{x}|z_j = 1)} = \frac{\pi_k \mathcal{N}(\mathrm{x}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathrm{x}|\mu_j, \Sigma_j)}. \tag{42}$$

Suppose we have a data set of observations $\{\mathrm{x}_1, \cdots, \mathrm{x}_N\}$, and we wish to model this data using a mixture of Gaussians. Denote this data set as an $N \times D$ matrix X in which the n^{th} row is given by x_n^T, latent variables as an $N \times K$ matrix Z with rows z_n^T. Assume i.i.d observations, we can obtain ;pg likelihood function in the form:

$$\ln p(\mathrm{X}|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathrm{x}_n|\mu_k, \Sigma_k)\}. \tag{43}$$

7.3 EM algorithm for Gaussian mixtures

We can not formulate a closed-form solution for maximum log likelihood, so we may use EM algorithm to solve the problem :

1. Initialize the means μ_k, covariances Σ_k and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
2. **E step.** Evaluate the responsibilities using the current parameter values.
3. **M step.** Re-estimate the parameters using the current responsibilities.
4. Evaluate the log likelihood

$$\ln p(\mathrm{X}|\mu, \Sigma, \pi) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathrm{x}_n|\mu_k, \Sigma_k)\} \tag{44}$$

8 Continuous Latent Variables

MLAPP Chap.12,13; PRML Chap.12

This section is lectured by myself.

9 Exact Inference

MLAPP Chap.20; PRML Chap.8.4

9.1 Inference on a Chain

Consider the graph shown in figure2. The joint distribution for this graph takes the form

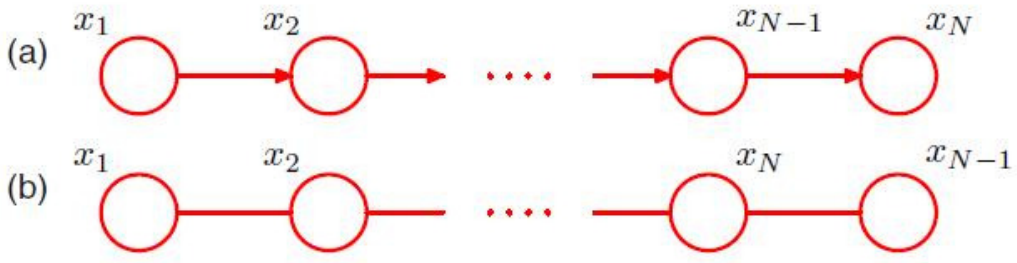


Figure 2: Chain Model

$$p(\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{Z}}\psi_{\mathbf{1,2}}(\mathbf{x_1}, \mathbf{x_2})\psi_{\mathbf{2,3}}(\mathbf{x_2}, \mathbf{x_3})...\psi_{\mathbf{N-1,N}}(\mathbf{x_{N-1}}, \mathbf{x_N}). \tag{45}$$

If we want to get marginal distribution $p(x_n)$ for a specific node, we should sum the joint distribution over all variables except x_n, so that

$$p(x_n) = \sum_{x_1} ... \sum_{x_{n-1}} \sum_{x_{n+1}} ... \sum_{x_N} p(\mathbf{x}) \tag{46}$$

But this way will cost so much computational resources, so we want to find another simpler way:

$$p(x_n) = \frac{1}{Z}[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \dots [\sum_{x_2} \psi_{2,3}(x_2, x_3)[\sum_{x_1} \psi_{1,2}(x_1, x_2)]] \dots] \tag{47}$$

$$[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \dots [\sum_{x_N}]\psi_{N-1,N}(x_{N-1}, x_N)] \dots] \tag{48}$$

$$= \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n) \tag{49}$$

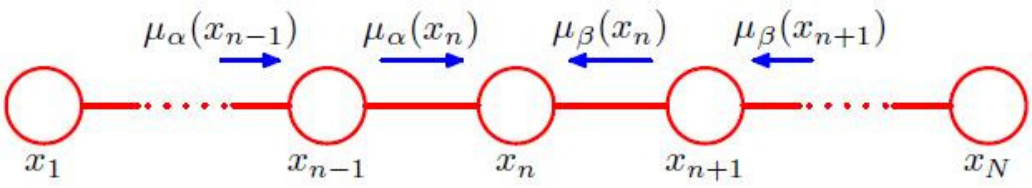
As shown in figure9.1, we can see message passed in the chain and we can compute $\mu_\alpha(x_i)$ and $\mu_\beta(x_i)$ recursively for any $i = 1, 2, \dots, N$.

$$\mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n)\mu_\alpha(x_{n-1}). \tag{50}$$

and

$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2). \tag{51}$$

Similar for $\mu_\beta(x_n)$



9.2 Inference on a Tree: sum-product algorithm

factor graphs

In a factor graph, there is a node (depicted as usual by **a circle**) for every variable in the distribution, as was the case for directed and undirected graphs. There are also additional nodes (depicted by **small squares**) for each factor fs(xs) in the joint distribution.
For example,

$$p(\mathbf{x}) = \mathbf{f_a}(\mathbf{x_1}, \mathbf{x_2})\mathbf{f_b}(\mathbf{x_1}, \mathbf{x_2})\mathbf{f_c}(\mathbf{x_2}, \mathbf{x_3})\mathbf{f_d}(\mathbf{x_3}) \tag{52}$$

can be expressed by the factor graph shown in figure3 Factor graphs are said to be bipartite because they consist of two distinct kinds of nodes, and all links go between nodes of opposite type.

sum-product algorithm

We shall now make use of the factor graph framework to derive a powerful class of efficient, exact inference algorithms that are applicable to tree-structured graphs. Here we shall focus on the problem of evaluating local marginals over nodes or subsets of nodes, which will lead us to the sum-product algorithm.
Our goal is to exploit the structure of the graph to achieve two things: (i) to obtain an efficient, exact

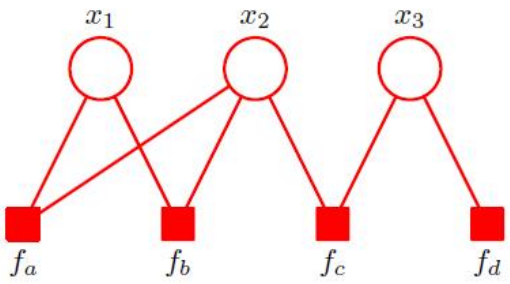


Figure 3: Example of a factor graph

inference algorithm for finding marginals; (ii) in situations where several marginals are required to allow computations to be shared efficiently. **finding the marginal** $p(x)$ **for a particular variable node** x

$$p(\mathbf{x}) = \prod_{\mathbf{s}\in\mathbf{ne}(\mathbf{x})} \mathbf{F_s}(\mathbf{x}, \mathbf{X_s})$$

$$p(x) = \prod_{s\in ne(x)} [\sum_{X_s} F_s(x, X_s)] = \prod_{s\in ne(x)} \mu_{f_s\to x}(x)$$

$$F_s(x, X_s) = f_s(x, x_1, ...x_M)G_1(x, X_{s1})...G_M(x, X_{sM})$$

$$\mu_{x_m\to f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm})$$

We have therefore introduced two distinct kinds of message, those that go from factor nodes to variable nodes denoted $\mu_{f\to x}(x)$, and those that go from variable nodes to factor nodes denoted $\mu_{x\to f}(x)$. **Illustration: example max-sum algorithm** Two other common tasks are to find a setting of the variables that has the largest probability and to find the value of that probability. We therefore seek an efficient algorithm for finding the value of x that maximizes the joint distribution p(x) and that will allow us to obtain the value of the joint distribution at its maximum.

9.3 Inference on a Graph

Here we will discuss graphs with loops. Our method is **Loopy belief propagation** One simple approach to approximate inference in graphs with loops, which builds directly on the previous discussion of exact inference in trees. The idea is simply to apply the sum-product algorithm even though there is no guarantee that it will yield good results. For some graphs, the algorithm will converge, whereas for others it will not.
We will say that a (variable or factor) node a has a message pending on its link to a node b if node a has received any message on any of its other links since the last time it send a message to b.

10 Variational Inference

MLAPP Chap.21,22; PRML Chap.10

11 Monte Carlo

MLAPP Chap.23,24; PRML Chap.11

11.1 Monte Carlo Inference

Standard distribution

Reject sampling

Importance sampling

11.2 Markov Chain Monte Carlo

Markov chain

$$p(\mathbf{x}_{1:\tau}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)\dots p(\mathbf{x}_\tau|\mathbf{x}_{\tau-1}) \tag{53}$$

Stationary distribution $\boldsymbol{\pi}$:

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{A} \tag{54}$$

We say that a Markov chain $\mathbf{A}$ is time reversible if there exists a distribution xxx

The basic idea in Metroplis-Hasting is that at each step, we propose to move from current state $\mathbf{x}$ to a new state $\mathbf{x}'$ with probability $q(\mathbf{x}'|\mathbf{x})$

12 Sequential Data

MLAPP Chap.17,18; PRML Chap.13

13 Gaussian Process

MLAPP Chap.15

14 Neural Network

MLAPP Chap.16, PRML CHap.5 In this section, our focus is on neural networks as efficient models for statistical pattern recognition.

14.1 Feed-forward Network Functions

The linear models for regression and classification, respectively, are based on linear combinations of fixed nonlinear basis functions $\phi_j(\mathrm{x})$ and take the form

$$y(\mathrm{x}, \mathrm{w}) = f(\sum_{j=1}^{M} w_j \phi_j(\mathrm{x})) \tag{55}$$

where $f(\cdot)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression. Our goal is to extend this model by making the basis function $\phi_j(\mathrm{x})$ depend on parameters and then allow these parameters to be adjusted, along with the coefficients $\{w_j\}$, during training.

For the first layer,

$$a_j = \sum_{j=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{56}$$

$$z_j = h(a_j) \tag{57}$$

This quantities correspond to the outputs of the basis functions in eq55 that, in the context of neural networks, are called *hidden units*. The activation functions $h(\cdot)$ are generally chosen to be sigmoidal function or the 'tanh'. Following eq55 these values are again linearly combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{58}$$

where $k = 1, \ldots, K$ and K is the total number of outputs. Finally, the output unit activations are transformed using an appropriate activation function to give a set of network output y_k. For standard regression problems, the activation function is the identity so that $y_k = a_k$. Similarly, for multiple binary classification problems, $y_k = \sigma(a_k)$. And for multiclass problems, a softmax function is used.

Combining these various stages to give the overall network function that, for sigmoidal output unit activation functions, we obtain

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma(\sum_{j=1}^{M} w_{kj}^{(2)} h(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}) + w_{k0}^{(2)}) \tag{59}$$

where the set of all weight and bias parameters have been grouped together into a vector $\mathbf{w}$. The process of evaluating 59 can then be interpreted as a *forward propagation* of information through the network.

One property of feed-forward networks, which will play a role when we consider Bayesian model comparison, is that multiple distinct choices for the weight vector $\mathbf{w}$ can all give rise to the same mapping function from inputs to outputs.

14.2 Network Training

There is a simple idea to train the parameter: minimize the error function. e.g.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2 \tag{60}$$

However we can provide a more general view of network training by first giving a probabilistic interpretation to the network outputs.

Consider firstly **regression problem** with a single target variable t and t has a Gaussian distribution with an $\mathbf{x}$-dependent mean:

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}). \tag{61}$$

For the conditional distribution given by 61, it is sufficient to take the output unit activation function to be identity. Given a data set of N i.i.d observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N$ along with corresponding target values $\mathbf{t} = \{t_1, \ldots, t_N\}$, we can construct the corresponding likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta). \tag{62}$$

Taking the negative logarithm, we obtain the error function

$$\frac{\beta}{2} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi). \tag{63}$$

Consider first the determination of $\mathbf{w}$, we want to minimize the error function given by

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(\mathbf{x}, \mathbf{w}) - t_n\}^2. \tag{64}$$

The value of $\mathbf{w}$ found by minimizing $E(\mathbf{w})$ will be denoted $\mathbf{w}_{\mathrm{ML}}$. Then

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}} - t_n\}^2. \tag{65}$$

Then consider the case of **binary classification problem**. The conditional distribution given inputs is then a Bernoulli distribution of the form

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}. \tag{66}$$

If we consider a training set, then the error function is then a *cross-entropy* error function of the form

$$E(\mathbf{w}) = \sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{67}$$

Finally we consider the standard multiclass classification problem in which each input is assigned to one of K mutually exclusive classes, and the networks outputs are interpreted as $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$, leading to the following error function

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}). \tag{68}$$

And the output unit activation function is given by

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w})}{\sum_j \exp(a_j(\mathbf{x}_n, \mathbf{w}))} \tag{69}$$

Parameter optimization

Then we turn next to the task of finding a weight vector w which minimizes the chosen function $E(\mathrm{w})$. But the error function typically has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes.

Because there is clearly no hope of finding an analytical solution to the equation $\nabla E(\mathbf{w}) = 0$ we resort to iterative numerical procedures.**The optimization of continuous nonlinear functions is a widely studied problem.** Most techniques involve choosing some initial value $\mathbf{w}^{(0)}$ for the weight vector and then moving through weight space in a succession of steps of the form

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \delta\mathbf{w}^{(\tau)}. \tag{70}$$

Then gradient information is important here. So we consider a local approximation of the error function based on Taylor expansion.

Local quadratic approximation

Consider the Taylor expansion of $E(\mathbf{w})$ around some point $\hat{\mathbf{w}}$ in weight space

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}) \tag{71}$$

where

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}} \tag{72}$$

and the Hessian matrix $\mathbf{H} = \nabla\nabla E$.

From 71, we obtain

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}). \tag{73}$$

Considering the minimum point $\mathbf{w}^*$, we have

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \mathbf{w}^*). \tag{74}$$

So if $\mathbf{w}^*$ is a minimum point, the Hessian matrix $\mathbf{H}$ should be *positive definite*, which means all the eigenvalues of $\mathbf{H}$ is positive.

Gradient descent optimization

The simplest approach to using gradient information is to choose the weight update in 70 to comprise a small step in the direction of the negative gradient, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)}) \tag{75}$$

where η is called *learning rate*. Note that the whole data set is used and this technique was once called *batch* method. At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as *gradient descent* or *steepest descent*.

For batch optimization, there are more efficient methods, such as *conjugate gradients* and *quasi-Newton* methods.

Also, there is an online version of gradient descent called *sequential gradient descent or stochastic gradient descent*, make update to the weight vector based on one data point at a time, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n(\mathbf{w}^{(\tau)}). \tag{76}$$

14.3 Error Backpropagation

Goal: find an efficient technique for evaluating the gradient of an error function $E(\mathrm{w})$ for a feed-forward neural network.

Algorithm:

1. Apply an input vector xn to the network and forward propagate through the network to find the activations of all the hidden and output units.
2. Evaluate the δk for all the output units using.
3. Backpropagate the δ's using to obtain δj for each hidden unit in the network.
4. Evaluate the required derivatives.

14.4 Regularization in Neural Networks

To avoid over-fitting, we can add a regularizer to the error function:

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}. \tag{77}$$

This regularizer can be interpreted as the negative logarithm of a zero-mean Gaussian prior distribution over the weight vector $\mathbf{w}$.

There are also many other ways to avoid over-fitting:

- Early stopping
- Invariance
- Tangent propagation
- Training with transformed data

More details can be found in section5.5 in *PRML.*

15 Deep Learning

MLAPP Chap.28

16 Combining Models

PRML Chap.14