1 Maximum likelihood estimation

1.1 Gaussian distribution

The univariate Gaussian is as follows.

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

The multivariate Gaussian is as follows.

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}^2) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}\underbrace{(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}_{\text{Mahalanobis distance from }\boldsymbol{\mu}\text{ to }\boldsymbol{x}}\right\}$$

D represents the dimensionality of the data, which means $\boldsymbol{x},\boldsymbol{\mu}$ are D-dimension vectors and $\boldsymbol{\Sigma}$ is $D\times D$ matrix. The covariance $\boldsymbol{\Sigma}$ determines the overall shape.

1.1.1 Maximum likelihood for Gaussian

💡 MLE is finding the $\boldsymbol{\mu},\boldsymbol{\Sigma}$ that best fits the given dataset.

💬 Likelihood is probability of observing the given data under a specific distribution. Likelihood of a dataset is a product of the individual probabilities.

A log-likelihood of data set $\boldsymbol{X} = (\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N)^T$ under multivariate Gaussian is as follows.

$$\ln p(\boldsymbol{X}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = -\frac{ND}{2}\ln(2\pi)-\frac{ND}{2}\ln|\boldsymbol{\Sigma}|-\frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu})$$

The sufficient statistics of the log-likelihood are as follows.

$$\sum_{i=1}^{N}\boldsymbol{x}_n \qquad \sum_{i=1}^{N}\boldsymbol{x}_n\boldsymbol{x}_n^T$$

The derivative of the log-likelihood is as follows.

$$\frac{\partial}{\partial\boldsymbol{\mu}}\ln p(\boldsymbol{X}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{n=1}^{N}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu})$$

By setting it to zero, we can find the MLE of $\boldsymbol{\mu}$.

$$\boldsymbol{\mu}_{\text{ML}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n \quad \boldsymbol{\Sigma}_{\text{ML}} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu}_{\text{ML}})(\boldsymbol{x}_n-\boldsymbol{\mu}_{\text{ML}})^T$$

1.1.2 Sequential estimation

In online learning setting, the following sequential estimation is important.

$$\begin{aligned}\boldsymbol{\mu}_{\text{ML}}^{(N)} &= \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n\\ &= \frac{1}{N}\boldsymbol{x}_N + \frac{1}{N}\sum_{n=1}^{N-1}\boldsymbol{x}_n\\ &= \frac{1}{N}\boldsymbol{x}_N + \frac{N-1}{N}\boldsymbol{\mu}_{\text{ML}}^{(N-1)}\\ &= \boldsymbol{\mu}_{\text{ML}}^{(N-1)} + \frac{1}{N}(\underbrace{\boldsymbol{x}_N-\boldsymbol{\mu}_{\text{ML}}^{(N-1)}}_{\text{deviation of new data}})\end{aligned}$$

The correctness of the sequential estimation can be proved. First, consider the general method called *Robbins-Monro* algorithm.

$$\theta^{(N)} = \theta^{(N-1)} + \alpha_{N-1}\frac{\partial}{\partial\theta^{(N-1)}}\ln p(x_N|\theta^{(N-1)})$$

Its convergence conditions are as follows.

$$\lim_{N\to\infty} a_N = 0 \quad \sum_{N=1}^{\infty}a_N = \infty \quad \sum_{N=1}^{\infty}a_N^2 < \infty$$

For a univariate Gaussian, we can rewrite previous sequential estimation in the form of Robbins-Monro algorithm.

$$\frac{\partial}{\partial\mu_{\text{ML}}}\ln p(x|\mu)\text{ML},\sigma^2) = \frac{1}{\sigma^2}(x-\mu_{\text{ML}}) \quad a_N = \sigma^2/N$$

1.1.3 The mixture of Gaussians

A superposition of K Gaussians are defined as follows.

$$p(\boldsymbol{x}) = \sum_{k=1}^{K}\pi_K\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k), \quad \text{Mixing coefficients}\sum_{k=1}^{K}\pi_k = 1$$

Its maximum likelihood is defined as follows.

$$\ln p(\boldsymbol{X}|\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{n=1}^{N}\ln\left\{\sum_{k=1}^{K}\pi_K\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right\}$$

1.2 Maximum a posterior (MAP) estimation

1.2.1 Coin flipping games

Let $X = 1$ if head. If random, $p(X=1), p(X=0)$ should always $= 0.5$. You suspect someone is using fraud coin. After observing N heads and M tails, what is $p(X=1)$ of the coin?

We only have one parameter $\theta = p(X=1)$ to learn since $p(X=0) = 1-\theta$. Flips are independent so $p(N,M|\theta) = \theta^N(1-\theta)^M$ is the likelihood we need to maximize.

$$\theta_{\text{ML}} = \underset{\theta}{\text{argmax}}\ln p(N,M|\theta) \qquad = \underset{\theta}{\text{argmax}}\ln\theta^N(1-\theta)^M$$

This maximization is a concave problem whose maximum can be calculated by setting its derivative to zero.

$$\begin{aligned}\frac{d\ln\theta^N(1-\theta)^M}{d\theta} &= \frac{N}{\theta}-\frac{M}{1-\theta} = 0\\ \therefore\theta_{\text{ML}} &= \frac{N}{N+M}\end{aligned}$$

Let us find the MAP of θ. First, we need to choose a prior distribution. The following beta distribution is typically used.

$$p(\theta)\sim\text{Beta}(\overline{N},\overline{M}) = \frac{\theta^{\overline{N}-1}(1-\theta)^{\overline{M}-1}}{B(\overline{N},\overline{M})}$$

The posterior distribution is proportional to the product of the likelihood and the prior.

$$\begin{aligned}&p(\theta|N,M)\propto p(N,M|\theta)\cdot p(\theta)\\ &\therefore p(\theta|N,M)\propto\theta^N(1-\theta)^M\cdot\frac{\theta^{\overline{N}-1}(1-\theta)^{\overline{M}-1}}{B(\overline{N},\overline{M})}\end{aligned}$$

The following is a concave maximization again.

$$\theta_{\text{MAP}} = \underset{\theta}{\text{argmax}}\ln p(\theta|N,M)$$

Solving in the same way,

$$\theta_{\text{MAP}} = \frac{N+\overline{N}-1}{N+\overline{N}-1+M+\overline{M}-1}.$$

Comparing inference methods

- Prior knowledge: Your father said that felines are small.
- Training sample: You visited India and saw a large tiger.
- Given the above prior and training sample,
  - **Prior**: still believe that felines are small.
  - **MLE**: believe that felines are large.
  - $\because$ We learn from observations only.
  - **MAP**: believe that some felines are small whereas others are large.
  - $\because$ We revise the prior knowledge with observations.

1.3 Nonparametric methods

In parametric methods, we assume a distribution and learn its parameters from data. Nonparametric methods do not assume any such underlying distribution (or model).

1.3.1 Kernel density estimation (KDE)

By averaging kernel functions centered on each observed data, we can derive PDF.

Kernel function k is $\int k(\boldsymbol{u})d\boldsymbol{u} = 1$ and non-negative.

Suppose observations drawn from an unknown density $p(\boldsymbol{x})$. The probability mass of a region $\mathcal{R}$ is as follows.

$$P = \int_{\mathcal{R}} p(\boldsymbol{x})d\boldsymbol{x}$$

The probability of K out of N observations fall into $\mathcal{R}$ is as follows.

$$\mathrm{Bin}(K|N,P) = \frac{N!}{K!(N-K)!}P^K(1-P)^{N-K} \quad K \simeq NP$$

If $\mathcal{R}$, whose volume is V, is sufficiently small,

$$P \simeq p(\boldsymbol{x})V \quad p(\boldsymbol{x}) = \frac{K}{NV}$$

When $\mathcal{R}$ is a hypercube with side h centered on $\boldsymbol{x}$, the total number K of points in the cube is as follows.

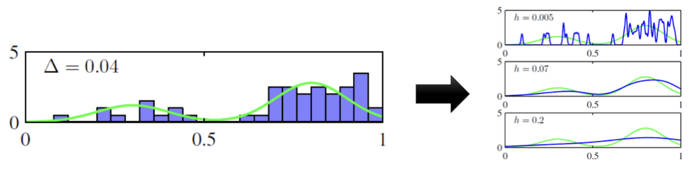
$$K = \sum_{n=1}^{N} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) \quad \text{where } k(\boldsymbol{u}) = \begin{cases} 1, & |u_i| \leqslant 1/2,\ (i=1,\cdots,D) \\ 0, & \text{otherwise} \end{cases}$$

Since $V = h^D$,

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)$$

We can use a kernel smoother to deal with artificial discontinuities. Following shows Gaussian kernel where h is stdev. It is placing a Gaussian for each $\boldsymbol{x}$ instead of a hypercube.

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{1/2}}\exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{x}_n\|^2}{2h^2}\right\}$$



KDE does not require training but require saving all training points and a linear complexity computation for each $\boldsymbol{x}$ to estimate.

1.3.2 Nearest neighbor method

In the kernel density estimation, h is fixed. In the nearest-neighbor method, h is locally decided until a sphere contains K data points, where h is the radius of the sphere.

We can extend this method to classification. In each class C_k there are N_k points. $\sum_k N_k = N$ points. For a test sample $\boldsymbol{x}$, we draw a circle containing K points. Let V be the volume of this circle and K_k the number of points of class C_k. Then,

$$p(C_k) = \frac{N_k}{N} \quad p(\boldsymbol{x}) = \frac{K}{NV} \quad p(\boldsymbol{x}|C_k) = \frac{K_k}{N_k V}$$

Using Bayes' theorem,

$$p(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{p(\boldsymbol{x})} = \frac{K_k}{K}$$

2 Information theory

Information quantity is inversely proportional to probability. We define *self-information* of an event x

$$I(x) = -\log P(x)$$

The *entropy* of a random variable x is

$$H[x] = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[\log P(x)]$$

Entropy means the minimum number of bits to encode the information drawn from P, *i.e.* the best coding scheme to deliver the information of P.

Huffman encoding

Each encoding unit is associated with a frequency. Create a binary tree whose children are the units with the smallest frequencies. The frequency of the root is the sum of leaves. Repeat.

$H(P)$ is the entropy of a distribution P. $H(P,Q)$ is the cross-entropy between P and Q. Gibb's inequality states always $H(P) \leq H(P,Q)$.

$$H(P,Q) = -\mathbb{E}_{x\sim P}[\log Q(x)]$$

Kullback-Leibler (KL) divergence measure the difference between P, Q.

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{x\sim P}[\log\frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - Q(x)]$$

$D_{\mathrm{KL}}(P\|Q)$ is always non-negative.

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= \mathbb{E}_{x\sim P}[\log P(x) - Q(x)] \\ &= \int P(x)(\log P(x) - \log Q(x))dx \\ &= \int P(x)\log P(x)dx - \int P(x)\log Q(x)dx \\ &= -H(P) + H(P,Q) \end{aligned}$$

3 Linear models

3.1 Linear models for regression

The goal of **regression** is to predict the value of one or more *target* variables t given the value of a D-dimensional vector $\boldsymbol{x}$ of *input* variables. The simplest form of linear models are linear functions of the input variables, called *linear regression*.

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots w_D x_D$$

We can obtain a much useful functions by taking linear combinations of a fixed set of nonlinear functions of the input variables, known as *basis functions*. Such models are linear functions of the *parameters* $\boldsymbol{w}$, which gives them simple analytical properties, and yet can be nonlinear w.r.t the input variables.

$$\begin{aligned} y(\boldsymbol{x}, \boldsymbol{w}) &= w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}) \\ &= \boldsymbol{w}^T \boldsymbol{\phi} \text{ (with } \phi_0(\boldsymbol{x}) = 1) \end{aligned}$$

The *bias* parameter w_0 allows for any fixed offset in the data.

ℹ **Basis functions**. Gaussian $\phi_j(x) = \exp\{-(x-\mu_j)^2/2s^2\}$. Sigmoidal $\phi_j(x) = \sigma((x-\mu_j)/s)$, $\sigma(a) = 1/(1+\exp(-a))$. The linear regression is the special case with identity basis $\phi(x) = x$.

3.1.1 Maximum likelihood and least square

We aim to fit the model to dataset using squared error function. It is equivalent as finding the maximum likelihood under the assumption that the prediction error is the result of Gaussian noise, *i.e.* $t = y(\boldsymbol{x}, \boldsymbol{w}) + \epsilon$ where $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \beta^{-1})$. Thus we can write

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

Now consider a dataset of N inputs $\{\boldsymbol{x}_n\}$ and their target values $\mathbf{t} = \{t_n\}$. The log-likelihood of the linear model is as follows. We drop $\boldsymbol{x}$ from condition for brevity.

$$\begin{aligned} \ln p(\mathbf{t}|\boldsymbol{w}, \beta) &= \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\boldsymbol{w}^T \phi(\boldsymbol{x}_n), \beta^{-1}) \\ &= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\boldsymbol{w}) \\ &\text{where } E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n)\}^2 \end{aligned}$$

MLE w.r.t $\boldsymbol{w}$ is as follows.

$$\begin{aligned} 0 = \nabla \ln p(\boldsymbol{t}|\boldsymbol{w}, \beta) &= \sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n)\}\phi(\boldsymbol{x}_n)^T \\ &= \sum_{n=1}^{N} t_n \phi(\boldsymbol{x}_n)^T - \boldsymbol{w}^T\left(\sum_{n=1}^{N}\phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T\right) \end{aligned}$$

$$\boldsymbol{w}_{\mathrm{ML}} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t} \text{ where } \Phi = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \cdots & \phi_{M-1}(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \cdots & \phi_{M-1}(\boldsymbol{x}_N) \end{bmatrix}$$

We define Moore-Penrose pseudo-inverse as follows.

$$\Phi^\dagger \equiv (\Phi^T\Phi)^{-1}\Phi^T$$

MLE w.r.t β is as follows.

$$0 = \frac{d}{d\beta}\ln p(\mathbf{t}|\boldsymbol{w}, \beta) = \frac{N}{2\beta} - E_D(\boldsymbol{w})$$

$$\beta_{\mathrm{ML}} = \frac{N}{2E_D(\boldsymbol{w}_{\mathrm{ML}})}, \text{ or } \frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}_{\mathrm{ML}}^T\phi(\boldsymbol{x}_n)\}^2$$

Prediction. Our goal is predicting the target t given new $\boldsymbol{x}$. Since we assumed a squared error, the optimal prediction is the conditional mean of p which is simply $y(\boldsymbol{x}, \boldsymbol{w})$.

❓ What happens if $\Phi^T\Phi$ is close to singular? → Numerical instability and high sensitivity to noise.

3.1.2 Analytical solution of linear regression

$$\mathcal{L} = (\boldsymbol{x\theta} - \boldsymbol{y})^T(\boldsymbol{x\theta} - \boldsymbol{y}), \text{ where } \boldsymbol{x} \in \mathbb{R}^{n\times d}, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{\theta} \in \mathbb{R}^{d\times 1}$$

Minimize this mean squared error loss function, $\underset{\theta}{\operatorname{argmin}}\, \mathcal{L}$.

$$\begin{aligned} \mathcal{L} &= ((\boldsymbol{x\theta})^T - \boldsymbol{y}^T)(\boldsymbol{x\theta} - \boldsymbol{y}) \\ &= (\boldsymbol{x\theta})^T\boldsymbol{x\theta} - \underbrace{(\boldsymbol{x\theta})^T\boldsymbol{y} - \boldsymbol{y}^T(\boldsymbol{x\theta})}_{\text{these are scalars}} + \boldsymbol{y}^T\boldsymbol{y} \\ &= \boldsymbol{\theta}^T\boldsymbol{x}^T\boldsymbol{x\theta} - 2\boldsymbol{y}^T(\boldsymbol{x\theta}) + \boldsymbol{y}^T\boldsymbol{y} \end{aligned}$$

ℹ Note that below holds.

$$\frac{\partial(\boldsymbol{A}x)}{\partial x} = \boldsymbol{A}^T, \frac{\partial(x^T\boldsymbol{A})}{\partial x} = \boldsymbol{A}, \frac{x^Tx}{\partial x} = 2x, \frac{\partial(x^T\boldsymbol{A}x)}{\partial x} = \boldsymbol{A}x + \boldsymbol{A}^Tx$$

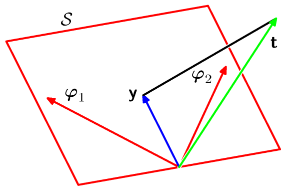
Using above,

$$\begin{aligned} &\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}} = \boldsymbol{x}^T\boldsymbol{x\theta} + \boldsymbol{x}\boldsymbol{x}^T\boldsymbol{\theta} = 2\boldsymbol{x}^T\boldsymbol{x\theta} - 2\boldsymbol{x}^T\boldsymbol{y} = 0 \\ &\therefore \boldsymbol{\theta}^* = (\boldsymbol{x}^T\boldsymbol{x})^{-1}\boldsymbol{x}^T\boldsymbol{y} \end{aligned}$$

n should be larger than d for this analytical solution to be derived.

3.1.3 Geometrical interpretation

Consider an N-dimensional space and a vector $\mathbf{t} = (t_1, \cdots, t_n)^T$ in this space. Each basis function $\phi_j(\boldsymbol{x}_n)$, evaluated at the N data points, can also be represented as a vector in the same space, denoted by φ_j. If $M < N$, M vectors φ_j will span a linear subspace $\mathcal{S}$ of dimensionality M. We define $\mathbf{y}$ to be a N-dimensional vector whose nth element is given by $y(\boldsymbol{x}_n, \boldsymbol{w})$. Since $\mathbf{y}$ is an arbitrary linear combination of vectors φ_j, it is in $\mathcal{S}$. The sum of squared error is equal to squared Euclidean distance between $\mathbf{y}$ and $\mathbf{t}$. Thus the least-squares solution for $\boldsymbol{w}$ corresponds to the orthogonal projection of $\mathbf{t}$ onto the subspace $\mathcal{S}$.



3.1.4 Sequential learning

Batch processing can be infeasible for large datasets. We use the following *stochastic gradient descent*, also known as *sequential gradient descent*, with the error function E.

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_n$$

In the case of the sum of squared error,

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \eta(t_n - \boldsymbol{w}^{(\tau)T}\phi_n)\phi_n \quad \phi_n = \phi(\boldsymbol{x}_n)$$

3.1.5 Regularized least square

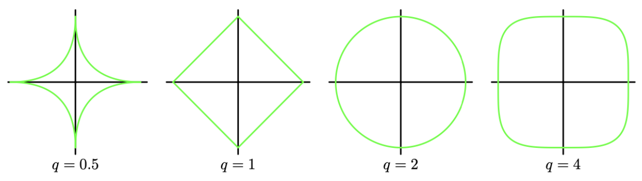
We can add regularization term to alleviate over-fitting.

$$E_D(\boldsymbol{w}) + \lambda E_W(\boldsymbol{w})$$

Weight decay or *L2 regularization* $E_W(\boldsymbol{w}) = 1/2\ \boldsymbol{w}^T\boldsymbol{w}$ encourages weight values to decay towards zero, unless supported by the data. It has the advantage that the error function remains a quadratic function of $\boldsymbol{w}$, and so its exact solution can be found in closed form.

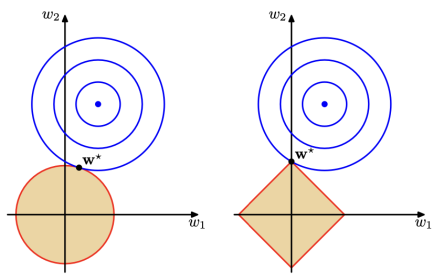
Total error function	$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n)\}^2 + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$
Solution	$\boldsymbol{w} = (\lambda\mathbb{I} + \Phi^T\Phi)^{-1}\Phi^T\mathbf{t}$

A more general regularizer $\frac{1}{2}\sum_{j=1}^{M}|w_j|^q$ is sometimes used. Quadratic regularizer $q = 2$ corresponds to weight decay. $q = 1$ is called *lasso* and has the property that if λ is sufficiently large, some of the coefficients w_j are driven to zero, leading to a sparse model.



Contours of the regularization term by q

3.1.6 Effect of regularization



The blue contours represent levels of the unregularized error function. Each contour line represents points in the parameter space that yield the same error value. The dot in the center of circles represents the minimum error, or the optimal parameters without regularization. The yellow regions represent constraint region for the quadratic regularizer $q = 2$ (left) and lasso $q = 1$ (right).

Black dots w^* represent the optimal parameters with regularization. These are the points where the first contour touches the edge of the constraint region, indicating the lowest error within the regularization constraint. Lasso gives a sparse solution in which $w_1^* = 0$.

3.2 The bias-variance decomposition

The expected squared loss can be written by

$$\mathbb{E}[L] = \int\int\{y(\boldsymbol{x}) - t\}^2 p(\boldsymbol{x}, t)d\boldsymbol{x}dt.$$

Our goal is to choose $y(\boldsymbol{x})$ so as to minimize $\mathbb{E}[L]$. Using the calculus of variations,

$$\frac{\delta\mathbb{E}[L]}{y(\boldsymbol{x})} = 2\int\{y(\boldsymbol{x} - t\}p(\boldsymbol{x}, t)dt = 0.$$

Solving for $y(\boldsymbol{x})$, we show that the optimal solution is the conditional expectation

$$y(\boldsymbol{x}) = \frac{\int tp(\boldsymbol{x}, t)dt}{p(\boldsymbol{x})} = \int tp(t|\boldsymbol{x})dt = \mathbb{E}_t[t|\boldsymbol{x}]$$

Using above, we expand the square term as follows.

$$\{y(\boldsymbol{x}) - t\}^2 = \{y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}] - t\}^2$$
$$= \{y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}]\}^2 + 2\{y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}]\}\{\mathbb{E}[t|\boldsymbol{x}] - t\} + \{\mathbb{E}[t|\boldsymbol{x}] - t\}^2$$

Performing the integral over t, we have following.

$$\mathbb{E}[L] = \int\{y(\boldsymbol{x}) - \mathbb{E}[t|\boldsymbol{x}]\}^2 p(\boldsymbol{x})d\boldsymbol{x} + \int\{\mathbb{E}[t|\boldsymbol{x}] - t\}^2 p(\boldsymbol{x})d\boldsymbol{x}$$

The second term, which is independent of $y(\boldsymbol{x})$ arises from the intrinsic noise on the data and represents the minimum achievable value of the expected loss.

We can interpret the uncertainty by considering multiple datasets. For any $\mathcal{D}$, we can obtain a prediction function $y(\boldsymbol{x}; \mathcal{D})$. Denoting the conditional expectation

$$h(\boldsymbol{x}) = \mathbb{E}[t|\boldsymbol{x}] = \int tp(t|\boldsymbol{x})dt,$$

the first term $\{y(\boldsymbol{x}; \mathcal{D}) - h(\boldsymbol{x})\}$ will be dependent on the particular dataset $\mathcal{D}$. If we add and subtract $\mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})]$ inside the braces, we obtain

$$\{y(\boldsymbol{x}; \mathcal{D}) - \mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})] + \mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})] - h(\boldsymbol{x})\}^2$$
$$= \{y(\boldsymbol{x}; \mathcal{D}) - \mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})]\}^2 + \{\mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})] - h(\boldsymbol{x})\}^2 + \text{ cross term}$$

If we take expectation, the cross term vanish, giving

$$\mathbb{E}_\mathcal{D}[\{y(\boldsymbol{x}; \mathcal{D}) - h(\boldsymbol{x})\}]$$
$$= \underbrace{\{\mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})] - h(\boldsymbol{x})\}^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_\mathcal{D}[\{y(\boldsymbol{x}; \mathcal{D}) - \mathbb{E}_\mathcal{D}[y(\boldsymbol{x}; \mathcal{D})]\}^2]}_{\text{variance}}$$

We can quantize bias and variance given multiple datasets $\mathcal{D}^{(l)}$ and the model $y^{(l)}$ trained with $\mathcal{D}^{(l)}$. The average prediction $\bar{y}$ can be estimated, resulting in the following decomposition.

$$\bar{y} = \frac{1}{L}\sum_{l=1}^{L} y^{(l)}(x)$$

$$\text{bias}^2 = \frac{1}{N}\sum_{i=1}^{N}\{\bar{y}(x_n) - h(x_n)\}^2$$

$$\text{variance} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{L}\sum_{i=1}^{L}\{y^{(l)}(x_n) - \bar{y}(x_n)\}^2$$

There is a trade-off between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance.

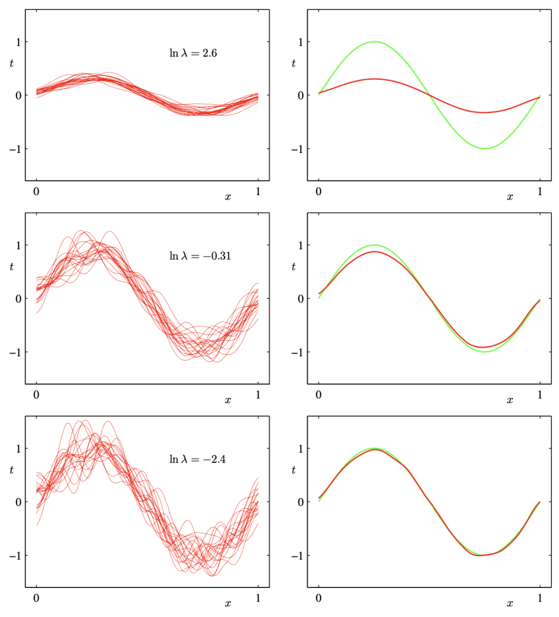
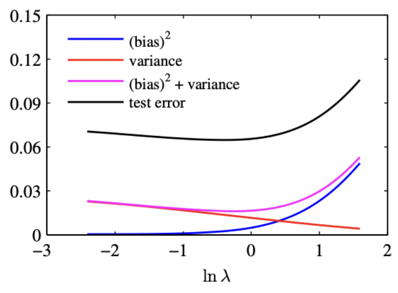


Illustration of the dependence of bias and variance on model complexity, governed by a regularization parameter λ. There are 100 data sets, each having 25 dat apoints, and there are 24 Gaussian basis functions in the model so that the total number of parameters is $M = 25$ including the bias parameter. The left shows the result of fitting the model to the data sets for various values of $\ln\lambda$. The right column shows the corresponding average of the 100 fits (red) along with the sinusoidal function from which the data sets were generated (green).

Plot of squared bias and variance, together with their sum, corresponding to the results shown in Figure 3.5. Also shown is the average test set error for a test data set size of 1000 points. The minimum value of $(\text{bias})^2 + \text{variance}$ occurs around $\ln\lambda = -0.31$, which is close to the value that gives the minimum error on the test data.



3.3 Bayesian linear regression

The choice of the number and form of the basis functions is still important in MLE. Appropriate model complexity cannot be decided simply by maximizing the likelihood function, because this always leads to excessively complex models and over-fitting. Independent hold-out data can be used to determine model complexity, but this can be both computationally expensive and waste of data. We therefore turn to a Bayesian linear regression, which will avoid the over-fitting problem of MLE, and lead to automatic methods of determining model complexity using the training data alone. As in Sec. 3.1.1, we will focus on the case of a single target variable $t = y(\boldsymbol{x}, \boldsymbol{w}) + \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \beta^{-1})$.

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

$$\ln p(\mathbf{t}|\boldsymbol{w}, \beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\boldsymbol{w})$$

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n)\}^2$$

Let a prior of $\boldsymbol{w}$ as zero-mean isotropic Gaussian governed by a single precision parameter α so that

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\mathbf{0}, \alpha^{-}1\boldsymbol{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right\}$$

Then posterior distribution of $\boldsymbol{w}$

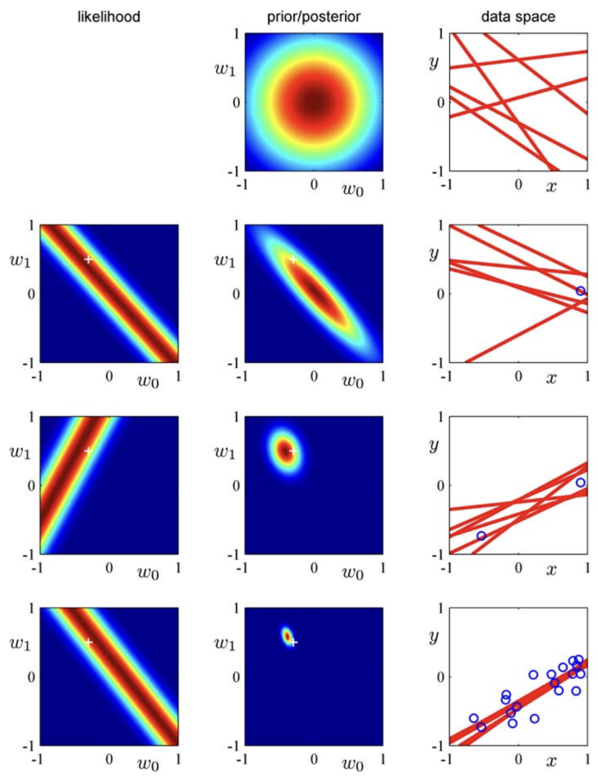
$$p(\boldsymbol{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{t}|\mathbf{x}, \boldsymbol{w}, \beta)p(\boldsymbol{w}|\alpha)$$

Its solution is to minimize the following. Note that solving this is as easy as the likelihood case due to the conjugate prior.

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

3.3.1 Sequential learning of MAP

We illustrate Bayesian learning in sequential update of a posterior distribution. We generate synthetic data from function $f(x) = -0.3 + 0.5x$ by choosing $x_n \sim U(x| - 1, 1)$ then evaluating $f(x)$ and adding Gaussian noise with standard deviation of 0.2 to obtain t_n. Our goal is to recover parameters of $f(x)$. We assume here that noise variance is known and hence set precision parameter to its true value $\beta = (1/0.2)^2 = 25$. Similarly, we fix $\alpha = 2.0$.



The figure above demonstrates the sequential nature of Bayesian learning in which the **current posterior distribution forms the prior** when a new data is observed. Third column shows the function $f(x, \boldsymbol{w})$ where $\boldsymbol{w}$ are drawn from the prior.

Row 1 Before any data points are observed.

Row 2 After observing a single data (blue circle). First plot shows likelihood $p(t|x, \boldsymbol{w})$ for **this** data as a function of $\boldsymbol{w}$. This provides a soft constraint that the line must pass close to the data, where *close* is determined by the noise precision β. True parameter values are shown by a white cross. **Multiply this likelihood by prior from previous row**, and normalize, obtaining **posterior** distribution (second plot).

Row 3 After observing a second data. Exactly same posterior distribution as combining the original prior with the likelihood function for 2 data. Since two are sufficient to define a line this already gives a relatively compact posterior distribution.

Row 4 After observing 20 data. First plot shows the likelihood function for the 20th data alone. Posterior is much sharper than in the third row. If number of data $\to \infty$, posterior distribution would be delta function centred on true parameter.

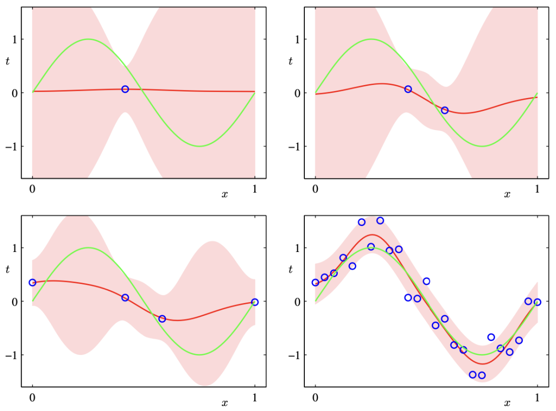
We are interested in making predictions of t for new values of $\boldsymbol{x}$. This requires that we evaluate the posterior *predictive distribution* (PPD) defined by

$$p(t|x,\mathbf{x},\mathbf{t}) = \int p(t|x,\boldsymbol{w})p(\boldsymbol{w}|\mathbf{x},\mathbf{t})d\boldsymbol{w}$$

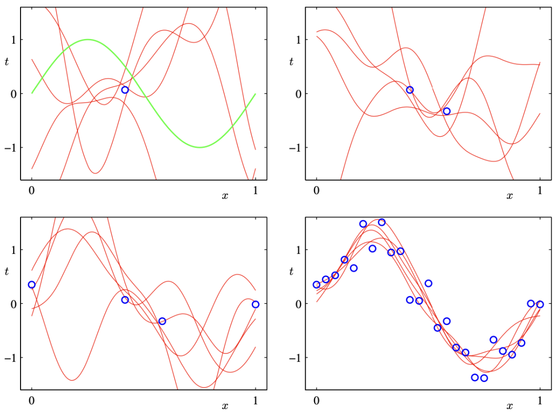
in which $\mathbf{x}$, $\mathbf{t}$ are the vector of input and target values from the training set. Since we used the conjugate prior:

$$\begin{aligned} p(t|x,\mathbf{x},\mathbf{t}) &= \mathcal{N}(t|m(x), s^2(x)) \\ m(x) &= \beta\phi(x)^T\boldsymbol{S}\sum_{n=1}^{N}\phi(x_n)t_n \\ s^2(x) &= \beta^{-1} + \phi(x)^T\boldsymbol{S}\phi(x) \\ \boldsymbol{S}^{-1} &= \alpha\boldsymbol{I} + \beta\sum_{n=1}^{N}\phi(x_n)\phi(x)^T \end{aligned}$$

The first term of $s^2(x)$, β^{-1} represents the noise on the data whereas the second term reflects the uncertainty associated with the parameters $\boldsymbol{w}$. Because the noise process and the distribution of $\boldsymbol{w}$ are independent Gaussians, their variances are additive.


Fit a model consisting of 9 Gaussian basis functions to sinusoidal wave (green) by the size of dataset ($N = 1, 2, 4, 25$). For each plot, the red curve shows the mean of the corresponding Gaussian predictive distribution, and the red shaded region spans standard deviation.

Note that the predictive uncertainty depends on x and is smallest in the neighbourhood of the data points. As additional data points are observed, the posterior distribution becomes narrower. It can be shown $s^2_{N+1}(\boldsymbol{x}) \le s^2_N(\boldsymbol{x})$. In the limit $N \to \infty$, the second term $\to 0$, and the variance of the predictive distribution arises solely from β.


We draw samples from the posterior distribution over w and plot the corresponding functions $y(x, w)$.

3.4 Linear models of classification

For regression problems, the target variable $\boldsymbol{t}$ is a vector of real numbers. For classification problems, $\boldsymbol{t}$ is a class label, *e.g.* $\boldsymbol{t} = (0, 1, 0, 0, 0)$ for $K = 5$ classes. The simplest linear discriminant function for binary classification can be defined as follows.

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 \quad \begin{cases} \text{class } \mathcal{C}_1 \text{ if } y(\boldsymbol{x}) \ge 0 \\ \text{class } \mathcal{C}_2 \text{ otherwise} \end{cases}$$

A more compact form where we introduce an additional dummy input value $x_0 = 1$ is as follows.

$$y(\boldsymbol{x}) = \tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}} \quad \tilde{\boldsymbol{w}} = (w_0, \boldsymbol{w}) \quad \tilde{\boldsymbol{x}} = (x_0, \boldsymbol{x})$$

3.4.1 Perceptron

Given $\boldsymbol{x}$, $\phi(\boldsymbol{x})$ is a fixed nonlinear transformation.

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^T\phi(\boldsymbol{x})), \quad \text{step function } f(a) = \begin{cases} +1, a \ge 0 \\ -1, a < 0 \end{cases}$$

Consider $\mathcal{M}$, a set of misclassified patterns. A natural choice of error function would be $|\mathcal{M}|$. However, methods based on changing $\boldsymbol{w}$ using the gradient of the error function cannot then be applied, because the gradient is zero almost everywhere. We therefore consider an error function where we multiply the target value $t \in \{-1, +1\}$:

$$E_P(\boldsymbol{w}) = -\sum_{n\in\mathcal{M}}\boldsymbol{w}^T\phi_n t_n$$

Applying the stochastic gradient descent algorithm,

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_P(\boldsymbol{w}) = \boldsymbol{w}^{(\tau)} + \eta\phi_n t_n$$

The *perceptron convergence theorem* states that if there exists an exact solution, in other words, if the training data set is linearly separable, then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps. For datasets that are not linearly separable, the perceptron learning algorithm will never converge.

3.4.2 Logistic regression

Suppose a binary classification. Its odds are defined as follows.

$$\text{odds} = \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_0)} = \frac{P(\mathcal{C}_1)}{1 - P(\mathcal{C}_1)}$$

We model the log odds, or *logits*, with a linear model. It is beneficial since (i) the probabilities are often non-linear, (then logits are linear) and (ii) logits range in $[-\infty, \infty]$.

$$\begin{aligned} &\log\left(\frac{P(\mathcal{C}_1|\boldsymbol{x})}{1 - P(\mathcal{C}_1|\boldsymbol{x})}\right) = \boldsymbol{w}^T\boldsymbol{x} \\ &\frac{P(\mathcal{C}_1|\boldsymbol{x})}{1 - P(\mathcal{C}_1|\boldsymbol{x})} = e^{\boldsymbol{w}^T\boldsymbol{x}} \\ &P(\mathcal{C}_1|\boldsymbol{x}) = e^{\boldsymbol{w}^T\boldsymbol{x}}(1 - P(\mathcal{C}_1|\boldsymbol{x})) \\ &P(\mathcal{C}_1|\boldsymbol{x})(1 + e^{\boldsymbol{w}^T\boldsymbol{x}}) = e^{\boldsymbol{w}^T\boldsymbol{x}} \\ &P(\mathcal{C}_1|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{1 + e^{\boldsymbol{w}^T\boldsymbol{x}}} = \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}} = \sigma(\boldsymbol{w}^T\boldsymbol{x}) \end{aligned}$$

Given dataset $\{\boldsymbol{x}_n, t_n\}_{n=1}^N$, consider the following likelihood function where $y_n = P(\mathcal{C}_1|\boldsymbol{x}_n)$, $t_n = 1$ if $\boldsymbol{x}_n$ belongs to $\mathcal{C}_1$, and $\mathbf{t} = (t_1 \cdots t_N)^T$.

$$p(\mathbf{t}|\boldsymbol{w}) = \prod_{n=1}^{N} y_n^{t_n}\{1-y_n\}^{1-t_n}$$

We take the negative logarithm of the likelihood derived above, which gives the *cross-entropy* error function.

$$E(\boldsymbol{w}) = -\ln p(\mathbf{t}|\boldsymbol{w}) = -\sum_{n=1}^{N} t_n \ln y_n + (1-t_n)\ln(1-y_n)$$

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{x}_n$$

This takes precisely the same form as the gradient of the sum-of-squares error function for the linear regression model.

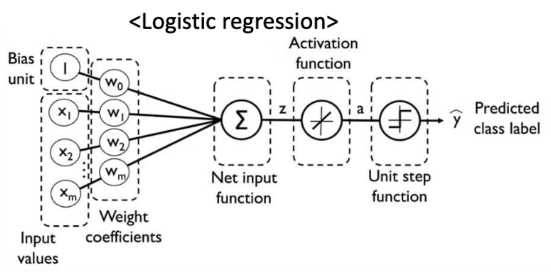
Decision boundary. We predict $y = 1$ when $P(y = 1|\boldsymbol{x}) > 0.5$, or equivalently $\boldsymbol{w}^T\boldsymbol{x} > 0$. In other words, hyperplane (A hyperplane is a subspace whose dimension is one less than that of its ambient space. If the space is 2-d, its hyperplanes are the 1-d lines.) corresponding to $\sigma = 0.5$ or $\boldsymbol{w}^T\boldsymbol{x} = 0$ separates the two classes.

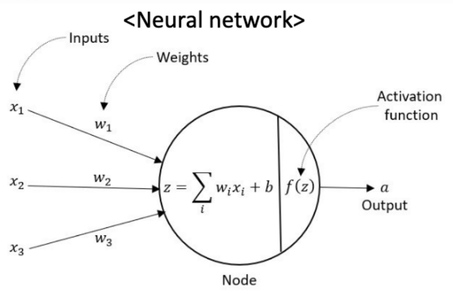
Overfitting. MLE can exhibit severe over-fitting for data sets that are linearly separable. It may push the decision boundary to extreme positions to achieve perfect classification on the training set. This results in parameter values $\boldsymbol{w}$ with very large magnitudes that are not robust to slight variations or noise in the data.

Solution. Due to the non-linearity of sigmoid, an analytical solution is not possible. The following iterative Newton-Raphson method can be used.

$$\boldsymbol{w}^{(\text{new})} = \boldsymbol{w}^{(\text{old})} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$$

where $\boldsymbol{H} = \nabla_{\boldsymbol{w}}\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$ is the Hessian matrix whose elements comprise the second derivatives of $E(\boldsymbol{w})$ w.r.t the components of $\boldsymbol{w}$.





4 Support Vector Machine

4.1 Maximum margin

The margin M is given by the perpendicular distance to the closest point, *i.e.* support vector. Consider a plus-plane $\boldsymbol{w}\cdot\boldsymbol{x} + b = +1$ and minus-plane $\boldsymbol{w}\cdot\boldsymbol{x} + b = -1$. The vector $\boldsymbol{w}$ is perpendicular to both planes. 💡 Let $\boldsymbol{u}, \boldsymbol{v}$ be two vectors on the plus plane. $\boldsymbol{w}\cdot(\boldsymbol{u} - \boldsymbol{v}) = 0$.

Let $\boldsymbol{x}^-$ be any point on the minus plane, and $\boldsymbol{x}^+$ be the closest plus-plane point to $\boldsymbol{x}^-$. They are any point in $\mathbb{R}^m$, not necessarily a data point. Since the vector from $\boldsymbol{x}^-$ to $\boldsymbol{x}^+$ is perpendicular to planes, $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda\boldsymbol{w}$ for some λ.

$$\begin{aligned}
\boldsymbol{w}\cdot\boldsymbol{x}^+ + b &= 1\\
\boldsymbol{w}\cdot(\boldsymbol{x}^- + \lambda\boldsymbol{w}) + b &= 1\\
\boldsymbol{w}\cdot\boldsymbol{x}^- + b + \lambda\boldsymbol{w}\cdot\boldsymbol{w} &= 1\\
\lambda\boldsymbol{w}\cdot\boldsymbol{w} &= 2\\
M = \|\boldsymbol{x}^+ - \boldsymbol{x}^-\| = \|\lambda\boldsymbol{w}\| &= \frac{2}{\|w\|}
\end{aligned}$$

Maximizing $\frac{2}{\|w\|}$ is minimizing $\frac{1}{2}\|w\|^2$

4.2 Learning maximum margin classifier

QP is a class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.

$$\min_{w,b}\frac{1}{2}\|w\|^2 \text{ (objective)} \quad (\boldsymbol{w}\cdot\boldsymbol{x}_j + b)y_j \geq 1 \text{ (constraints)}$$

Rewrite the constraints using one Largrange multiplier a_j per example. Largrangian is

$$L(\boldsymbol{w}, \boldsymbol{\alpha}) = \frac{1}{2}\|w\|^2 - \sum_j \alpha_j[(\boldsymbol{w}\cdot\boldsymbol{x}_j + b)y_j - 1]$$

where $\alpha_j \geq 0$. Our goal now is to solve

$$\min_{w,b}\max_{\alpha\geq 0} L(\boldsymbol{w}, \alpha)$$

KKT conditions are:

$$\frac{\partial L}{\partial w} = w - \sum_j a_j y_j \boldsymbol{x}_j = 0$$

$$\frac{\partial L}{\partial b} = -\sum_j a_j y_j = 0$$

Substituting these values back in, we obtain:

$$\max_{\alpha\geq 0, \sum \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j (\boldsymbol{x}_i\cdot\boldsymbol{x}_j)$$

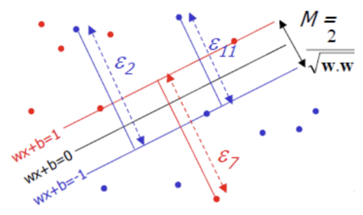
This dual formulation only depends on dot-products of the features.

4.3 Soft margin classifier

The data might not be linearly separable. Ideas?

1. Minimize margin M and the number of train set errors $N_{\text{number of errors}}$
   ⚠ ill-defined optimization
2. Minimize $M + C\cdot N_{\text{number of errors}}$ where C is tradeoff parameter
   ⚠ Not expressed as quadratic programming problem → Slow
   ⚠ Doesn't distinguish between disastrous errors and near misses

💡 **Minimize $M + C\cdot$ distance of error points to their correct place**



$$\frac{1}{2}\|w\|^2 + C\sum_{k=1}^{R}\xi_k \qquad \text{(objective)}$$

$$(\boldsymbol{w}\cdot\boldsymbol{x}_k + b)y_k \geq 1 - \xi_k,\ \xi_k \geq 0 \qquad \text{(constraints)}$$

The corresponding Lagrangian is

$$L(\boldsymbol{w},\boldsymbol{\alpha}) = \frac{1}{2}\|w\|^2 + C\sum_{k=1}^{R}\xi_k - \sum_j \alpha_j[(\boldsymbol{w}\cdot\boldsymbol{x}_j + b)y_j - 1 + \xi_j] - \sum_j \mu_j\xi_j$$

where α_j, μ_j are Lagrange multipliers. We can use KKT conditions eliminate $\boldsymbol{w}, b, \xi$ from the Lagrangian.

$$\max \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j)$$

It is identical to the separable case, except the constraints:

$$0 \leq \alpha_k \leq C, \quad \sum_k \alpha_k y_k = 0$$

Then

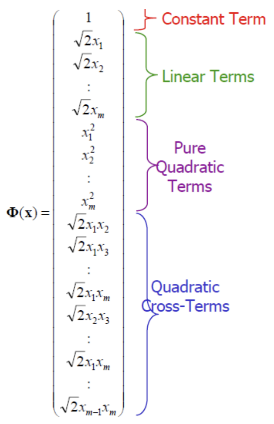
$$\boldsymbol{w} = \sum_k \alpha_k y_k \boldsymbol{x}_k \quad b = y_K(1-\xi_K) - \boldsymbol{x}_K\cdot\boldsymbol{w}$$

where $K = \underset{k}{\operatorname{argmax}}\ \alpha_k$

Classifier is

$$f(\boldsymbol{x},\boldsymbol{w},b) = \operatorname{sign}(\boldsymbol{wx} - b)$$

4.4 QP with basis functions



$\Phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \vdots, \sqrt{2}x_m, x_1^2, x_2^2, \vdots, x_m^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \vdots, \sqrt{2}x_1x_m, \sqrt{2}x_2x_3, \vdots, \sqrt{2}x_1x_m, \vdots, \sqrt{2}x_{m-1}x_m)$
Constant Term
Linear Terms
Pure Quadratic Terms
Quadratic Cross-Terms

Polynomial basis function (order=2)

Just substitute $\boldsymbol{x}_k$ with $\Phi(\boldsymbol{x})_k)$ in Lagrangian, in $\boldsymbol{w}$, and in classifier. A trick to compute $\Phi(\boldsymbol{x}_i)\cdot\Phi(\boldsymbol{x}_j)$ in $O(m)$ where m is number of features.

$$\Phi(a)\cdot\Phi(b) = 1 + 2\sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{m} a_i^2 b_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_i a_j b_i b_j$$

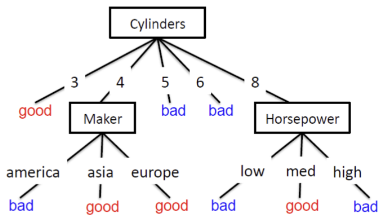
Polynomial $K(\boldsymbol{a},\boldsymbol{b}) = (\boldsymbol{a}\cdot\boldsymbol{b} + 1)^d$ is an example of kernel function. There are other kernel functions:

- Radial-basis-style: $K(\boldsymbol{a},\boldsymbol{b}) = \exp\left(-\frac{(\boldsymbol{a}-\boldsymbol{b})^2}{2\sigma^2}\right)$
- Neural-net-style: $K(\boldsymbol{a},\boldsymbol{b}) = \tanh(\kappa\boldsymbol{a}\cdot\boldsymbol{b} - \delta)$

SVM don't overfit as much as you'd think. No matter what the basis function, there are really only up to R parameters: $\alpha_1,\cdots,\alpha_R$, and usually most are set to zero by the maximum margin. Asking for small $\|\boldsymbol{w}\|^2$ is like weight decay in neural networks and like ridge regression (L2 regularization).

5 Tree-based models

5.1 Decision tree (DT)



- Each internal node tests and attribute x_i
- Each branch assigns an attirbute value $x_i = v$
- Each leaf assigns a class y
- To classify input x: traverse the tree from root to leaf

DTs can represent any function of the input attributes. Learning the simplest DT is an NP-complete problem. Resort to a greedy heuristic: recursively split on best attribute (feature).

5.1.1 Entropy

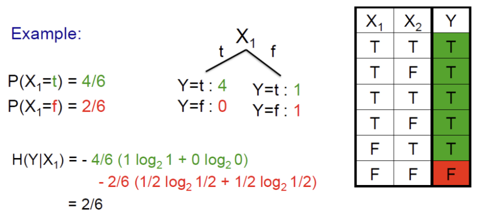
Good **split** if we are more certain about classification after split. We define an entropy $H(Y)$ of a random variable Y:

$$H(Y) = -\sum i = 1^k P(Y = y_i)\log_2 P(Y = y_i)$$

	High entropy	Low entropy
Y is from distribution	Uniform-like	Varied
Histogram	Flat	Many lows & highs
Valued sampled from it	Less predictable	More predictable

Conditional entropy $H(Y|X)$ of a random variable Y conditioned on a random variable X:

$$-\sum_{j=1}^{v} P(X = x_j)\sum_{i=1}^{k} P(Y = y_i|X = x_j)\log_2 P(Y = y_i|X = x_j)$$



Example:
$P(X_1=t) = 4/6$
$P(X_1=f) = 2/6$
X_1
t
f
Y=t : 4
Y=f : 0
Y=t : 1
Y=f : 1
$H(Y|X_1) = -\ 4/6\ (1 \log_2 1 + 0 \log_2 0)$
$-\ 2/6\ (1/2 \log_2 1/2 + 1/2 \log_2 1/2)$
$= 2/6$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

Information gain is decrease in entropy (uncertainty) after splitting.

$$\mathrm{IG}(X) = H(Y) - H(Y|X)$$

5.2 Learning decision trees

1. Start from empty decision tree
2. Split on next best attribute (feature)

$$\underset{i}{\operatorname{argmax}}\ \mathrm{IG}(X_i) = \underset{i}{\operatorname{argmax}}\ H(Y) - H(Y|X_i)$$

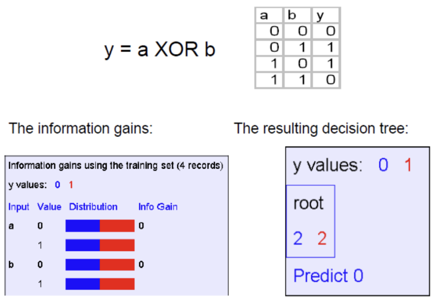
3. Recurse

5.2.1 Base cases

Stop if

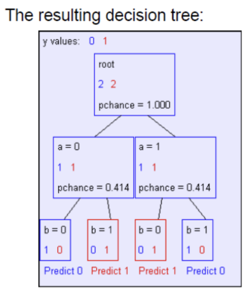
1. all records in current data subset have same output
2. all records have exactly the same set of input attributes
3. all attributes have zero information gain 💬

→ may enable perfect classification but can lead to overfitting.


y = a XOR b
The information gains:
The resulting decision tree:
Information gains using the training set (4 records)
y values: 0 1
Input Value Distribution Info Gain
root
2 2
Predict 0

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

(a) If we have 3



(b) If we omit 3

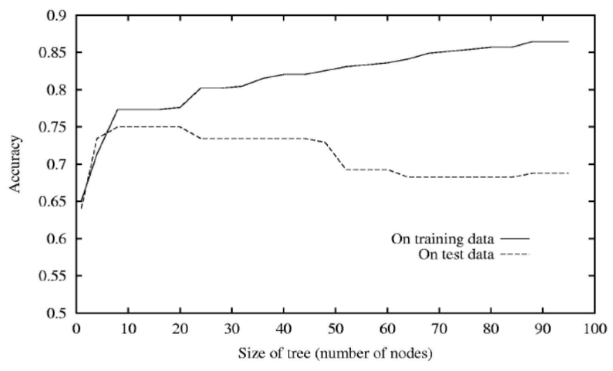
Base case 3: XOR problem

5.3 Overfitting

Standard decision trees have no learning bias.

- Training set error is always zero if no label noise.
- High variance
- Must introduce some bias towards simpler tree

Strategies for simpler trees: fixed depth, fixed number of leaves, etc.



5.3.1 Pruning a tree

Pchance is the probability that we observe the same probability distribution when Y is completely uncorrelated with specific X_j. Computed via chi-squared test. Prune if pchance > some value.

5.4 Real-valued features

We use threshold splits: split on attribute X at value t. One branch represent $X < t$, other $X \geq t$. Allow repeated splits on same variable.

5.4.1 Picking the best threshold

IG($Y|X$:t), information gain for Y when testing if $X \geq t$ or $X < t$.

$$\begin{aligned} H(Y|X{:}t) &= p(X<t)H(Y|X<t) + P(X \geq t)H(Y|X \geq t) \\ \mathrm{IG}(Y|X{:}t) &= H(Y) - H(Y|X{:}t) \\ \mathrm{IG}^*(Y|X) &= \max_t \mathrm{IG}(Y|X{:}t) \end{aligned}$$

5.5 Ensemble method

5.5.1 Bagging (Random forest)

Sampling with replacement (bootstrap) and then building an ensemble **reduces the variance** of the forest without increasing the bias. Sampling w/o replacement would lead to pretty high variance.

For regression, $y_{\text{COM}}(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\boldsymbol{x})$ (average). Suppose the model $y_m(\boldsymbol{x}) = h(\boldsymbol{x}) + \epsilon_m(\boldsymbol{x})$. The average sum-of-squared error is as follows:

$$\mathbb{E}_{\boldsymbol{x}}[\{y_m(\boldsymbol{x}) - h(\boldsymbol{x})\}^2] = \mathbb{E}_{\boldsymbol{x}}[\epsilon_m(\boldsymbol{x})^2]$$

```
Require: Classifiers $C$, train set $S = \{X, Y\}$, inducer $\mathcal{I}$, integer $T$
  for $T$ bootstrap samples do
      $S' \leftarrow$ bootstrap from $S$                ▷ i.i.d. sample w/ replacement
      $C_i \leftarrow \mathcal{I}(S')$
  $C^*(x) \leftarrow \operatorname{argmax}_{y \in Y} \sum_{i:C_i(x)=y} 1$        ▷ the most often predicted label $y$
```

The average error in the models is as follows.

$$E_{\text{AV}} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\boldsymbol{x}}[\epsilon_m(\boldsymbol{x})^2]$$

The average error of the committee is as follows.

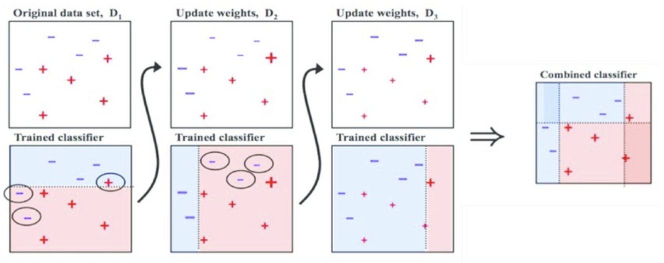
$$E_{\text{COM}} = \mathbb{E}_{\boldsymbol{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\boldsymbol{x}) - h(\boldsymbol{x})\right\}^2\right] = \mathbb{E}_{\boldsymbol{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\boldsymbol{x})\right\}^2\right]$$

Assuming zero mean and uncorrelated errors $\mathbb{E}_{\boldsymbol{x}}[\epsilon_m(\boldsymbol{x})] = 0$ and $\mathbb{E}_{\boldsymbol{x}}[\epsilon_m(\boldsymbol{x})\epsilon_l(\boldsymbol{x})] = 0 (m \neq l)$, $E_{\text{COM}} = \frac{1}{M}E_{\text{AV}}$. $\rightarrow$ **variance reduced**!

5.5.2 Boosting

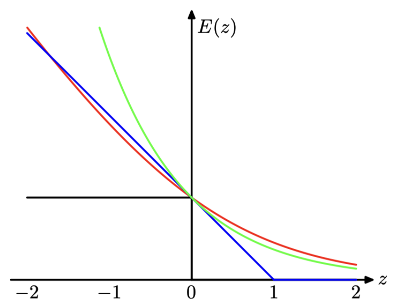
AdaBoost

```
for $i = 1$ to $N$ do $w_i^{(1)} = 1$
for $m = 1$ to $M$ iterations do
    Fit weak classifier $m$ to minimize $\epsilon_m = \sum_i w_i^{(m)} I(\boldsymbol{x}_i) / \sum_i w_i^{(m)}$
    where $I(\boldsymbol{x}_i) = 1$ if $f_m(\boldsymbol{x}_i) \neq y_i$ and 0 otherwise
    $\alpha_m \leftarrow \ln(1 - \epsilon_m)/\epsilon_m$
    for $i = 1$ to $N$ do
        $w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m I(\boldsymbol{x}_i)}$
```



AdaBoost use exponential loss.

Plot of the exponential (green) and rescaled cross-entropy (red) error functions along with the hinge error (blue) used in support vector machines, and the misclassification error (black). Note that for large negative values of $z = ty(\mathbf{x})$, the cross-entropy gives a linearly increasing penalty, whereas the exponential loss gives an exponentially increasing penalty.



6 Graphical models

The joint distribution of variables can be decomposed via the chain rule.

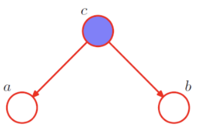
$$p(x_1, \cdots, x_K) = p(x_K|x_1, \cdots, x_{K-1}) \cdots p(x_2|x_1)p(x_1)$$

If the dependence structure is known, we can factorize the joint distribution where pa_k is the set of parents of x_k.

$$p(\boldsymbol{x}) = \prod_{k=1}^{K} p(x_k|\text{pa}_k)$$

We assume directed acyclic graphs (DAGs) which show conditional independence relationships.

$$p(a, b|c) = \frac{p(a, b, c)}{p(c)} = p(a|c)p(b|c)$$
$$a \perp\!\!\!\perp b \mid c$$



6.1 Bayesian linear model

Revisit the linear model in Sec. 3.3. We will treat $\mathbf{t}$,$\boldsymbol{w}$ as random variables (since they are outcomes that are not deterministically known before the observation). Their joint distribution can be written:

$$p(\mathbf{t}, \boldsymbol{w}|\mathbf{x}, \alpha, \sigma^2) = p(\boldsymbol{w}|\alpha) \prod_{n=1}^{N} p(t_n|\boldsymbol{w}, x_n, \sigma^2)$$

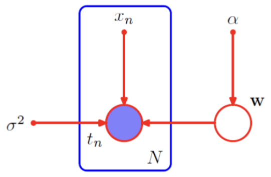
The equation assumes that:

- The weights $\boldsymbol{w}$ are drawn from some prior distribution that doesn't depend on the data.
- Given the weights, the likelihood of the observed data is independent across data points.

💬 Don't try to derive the equations. Just think that posterior is our updated belief on prior by observing data, which is multiplying the likelihood of data points on prior.

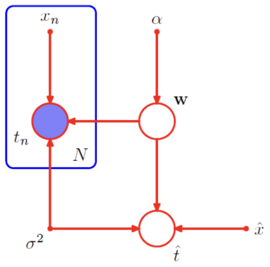
6.1.1 Graphical model

We draw a representative node t_n and surround this with a *plate*, labelled with N indicating that there are N nodes of this kind. We denote observed variables such as t_n by shading. Conversely, $\boldsymbol{w}$ is not observed, or, *latent/hidden* variable.



You may understand this as a probabilistic model of how data samples are generated, *i.e.* ancestral sampling.

Let us extend this for inference.



$$p(\hat{t}, \mathbf{t}, \boldsymbol{w}|\hat{x}, \mathbf{x}, \alpha, \sigma^2)$$
$$= \left[\prod_{n=1}^{N} p(t_n|x_n, \boldsymbol{w}, \sigma^2)\right] p(\boldsymbol{w}|\alpha)p(\hat{t}|\hat{x}, \boldsymbol{w}, \sigma^2)$$

$$p(\hat{t}|\hat{x}, \mathbf{x}, \mathbf{t}, \alpha, \sigma^2) \propto \int p(\hat{t}, \mathbf{t}, \boldsymbol{w}|\hat{x}, \mathbf{x}, \alpha, \sigma^2) d\boldsymbol{w}$$

6.2 Linear-Gaussian models

A multivariate Gaussian can be expressed as a directed graph. Consider an arbitrary DAG over D variables where node i represents a single continuous random variable x_i having a Gaussian distribution. The mean of this distribution is taken to be a linear combination of the states of its parent nodes pa_i,

$$p(x_i|\text{pa}_i) = \mathcal{N}\left(x_i \middle| \sum_{j \in \text{pa}_i} w_{ij}x_j + b_i, v_i\right)$$

where w_{ij}, b_i are parameters governing the mean, and v_i is the variance of the conditional distribution for x_i. The log of the joint distribution is then the log of the product of these conditionals over all nodes in the graph:

$$\begin{aligned}\ln p(\boldsymbol{x}) &= \sum_{i=1}^{D} \ln p(x_i|\text{pa}_i) \\ &= -\sum_{i=1}^{D} \frac{1}{2v_i}\left(x_i - \sum_{j \in \text{pa}_i} w_{ij}x_j - b_i\right)^2 + \text{const}\end{aligned}$$

This is a quadratic function of the components of $\boldsymbol{x} = (x_1, \cdots, x_D)^T$, and hence the joint distribution $p(\boldsymbol{x})$ is multivariate Gaussian.

We can determine the mean and covariance of the joint distribution recursively as follows. Each x_i has (conditional on the states of its parents) a Gaussian distribution and so

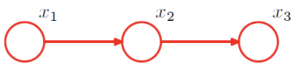
$$x_i = \sum_{j \in \text{pa}_i} w_{ij}x_j + b_i + \sqrt{v_i}\epsilon_i$$

where ϵ_i is a zero mean, unit variance Gaussian random variable satisfying $\mathbb{E}[\epsilon_i] = 0$ and $\mathbb{E}[\epsilon_i\epsilon_j] = I_{ij}$ where I_{ij} is the i, j element of $\mathbb{I}$.

$$\mathbb{E}[x_i] = \sum_{j \in \text{pa}_i} w_{ij}\mathbb{E}[x_j] + b_i$$

Thus, we can estimate $\mathbb{E}[\boldsymbol{x}] = (\mathbb{E}[x_1], \cdots, \mathbb{E}[x_D])^T$ by starting from bottom nodes and working recursively through the graph. The covariance matrix can also be similarly obtained.

A directed graph over three Gaussian variables, with one missing link.



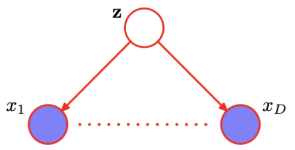
$$\boldsymbol{\mu} = (b_1, b_2 + w_{21}b_1, b_3 + w_{32}b_2 + w_{32}w_{21}b_1)^{\mathrm{T}}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} v_1 & w_{21}v_1 & w_{32}w_{21}v_1 \\ w_{21}v_1 & v_2 + w_{21}^2 v_1 & w_{32}(v_2 + w_{21}^2 v_1) \\ w_{32}w_{21}v_1 & w_{32}(v_2 + w_{21}^2 v_1) & v_3 + w_{32}^2(v_2 + w_{21}^2 v_1) \end{pmatrix}$$

6.3 Other examples

6.3.1 Naive Bayes

A graphical representation of the 'naive Bayes' model for classification. Conditioned on the class label $\mathbf{z}$, the components of the observed vector $\mathbf{x} = (x_1, \ldots, x_D)^{\mathrm{T}}$ are assumed to be independent.



z blocks the path between x_i, x_j and so $x_i \perp\!\!\!\perp x_j$. If, however, we marginalize out z (so that it is unobserved), they are no longer blocked. This tells us that in general the marginal density $p(x)$ will not factorize with respect to the components of x.

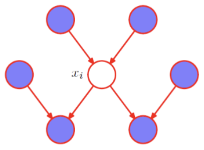
Suppose our observed variable consists of a D-dimensional vector $\boldsymbol{x}$, and we wish to assign $\boldsymbol{x}$ to one of K classes. z is one-hot encoded class vector. We can define a generative model by introducing a multinomial prior $p(z|\boldsymbol{\mu})$ over the class labels, where the μ_k is the prior probability of class $\mathcal{C}_k$, and conditional distribution $p(\boldsymbol{x}|z)$. The conditional independence assumption is helpful when dimensionality D of the input space is high, making density estimation challenging.

6.3.2 Markov blanket

Consider a joint distribution $p(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_D)$ where x_i conditioned on all of variables $\boldsymbol{x}_{j \neq i}$.

$$p(\boldsymbol{x}_i | \boldsymbol{x}_{j \neq i}) = \frac{p(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_D)}{\int p(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_D) d\boldsymbol{x}_i} = \frac{\prod_k p(\boldsymbol{x}_k | \mathrm{pa}_k)}{\int \prod_k p(\boldsymbol{x}_k | \mathrm{pa}_k) d\boldsymbol{x}_i}$$

We now observe that any factor $p(\boldsymbol{x}_k | \mathrm{pa}_k)$ that does not have any functional dependence on x_i can be taken outside the integral over $\boldsymbol{x}_i$, and will therefore cancel. $p(\boldsymbol{x}_i | \mathrm{pa}_i)$ will depend on the parents of nodes $\boldsymbol{x}_i$, whereas $p(\boldsymbol{x}_k | \mathrm{pa}_k)$ will depend on children of $\boldsymbol{x}_i$ as well as on the co-parents, which is, parents of node $\boldsymbol{x}_k$ other than $\boldsymbol{x}_i$. The set of nodes comprising the **parents**, the **children** and the **co-parents** is called the Markov blanket. We can think of the Markov blanket of a node $\boldsymbol{x}_i$ as being the minimal set of nodes that isolates $\boldsymbol{x}_i$ from the rest of the graph.



6.4 Markov random field

Markov random fields are undirected graphical models. You may consider nodes as particles having potentials. Its probability definition is influenced by the following Boltzmann distribution.

$$p_i = \frac{1}{Q} e^{-\epsilon_i/(kT)} = \frac{e^{-\epsilon_i/(kT)}}{\sum_{j=1}^{M} e^{-\epsilon_j/(kT)}}$$

Their conditional independence property can be written as follows when there is no direct link between x_i and x_j given all other nodes.

$$p(x_i, x_j | \boldsymbol{x}_{\backslash\{i,j\}}) = p(x_i | \boldsymbol{x}_{\backslash\{i,j\}}) p(x_j | \boldsymbol{x}_{\backslash\{i,j\}})$$

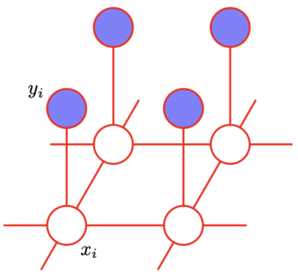
The joint probability of nodes (particles) is defined as follows with the positive potential function ψ_C where C means a maximal clique (complete subgraph) and $\boldsymbol{x}_C$ means the set of nodes in C.

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_C \psi_C(\boldsymbol{x}_C) \quad Z = \sum_x \prod_C \psi_C(\boldsymbol{x}_C)$$

We are restricted to potential functions which are strictly positive it is convenient to express them as exponentials, so that $\psi_C(\boldsymbol{x}_C) = \exp\{-E(\boldsymbol{x}_C)\}$ where E is called energy function. The joint distribution is defined as the product of potentials, and so the total energy is obtained by adding the energies of each of the maximal cliques. The Hammersley-Clifford theorem proves the correctness.

6.4.1 Image de-noising with MRF

An undirected graphical model representing a Markov random field for image de-noising, in which x_i is a binary variable denoting the state of pixel i in the unknown noise-free image, and y_i denotes the corresponding value of pixel i in the observed noisy image.



Binary pixel values $x_i, y_i \in \{-1, +1\}$. Noising is done by randomly flipping the sign of pixels with some small probability. Since noise level is small, we know that there will be a strong correlation between x_i and y_i. We also know that neighboring pixels x_i, x_j in an image are strongly correlated.

This prior knowledge can be captured using the Markov random field model. This graph has two types of cliques: $\{x_i, y_i\}$, and $\{x_i, x_j\}$ for neighboring pixels. The have associated energy function that expresses the correlation between two. We choose $-\eta x_i y_i$ and $-\beta x_i x_j$ where $\eta, \beta > 0$. to give a lower energy (thus encouraging a higher probability) when x_i, y_i have same sign.

Because a potential function is an arbitrary, nonnegative function over a maximal clique, we can multiply it by any nonnegative functions of subsets of the clique, or equivalently we can add the corresponding energies. We add an extra there hx_i to bias model towards particular sign. The complete energy function is:

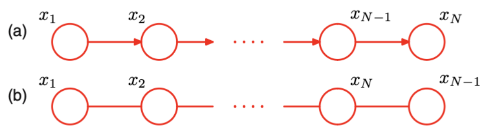
$$E(\boldsymbol{x}, \boldsymbol{y}) = h \sum_i x_i - \beta \sum_{\{i,j\}} x_i x_j - \eta \sum_i x_i y_i$$

which defines a joint distribution:

$$p(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{Z} \exp\{-E(\boldsymbol{x}, \boldsymbol{y})\}$$

Image denoising is achieved by, starting from $x_i = y_i$, iteratively updating x_i to decrease the energy. We stop if no further changes, *i.e.* a local optimum.

6.5 Inference on a chain



Consider converting a directed chain to undirected chain. The maximal cliques are simply the pairs of neighboring nodes, so

$$p(\boldsymbol{x}) = \frac{1}{Z} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$

The marginal distribution $p(x_n)$ is obtained by summing the joint distribution over all variables except x_n:

$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} p(\boldsymbol{x})$$

Because there are N variables each with K states, there are K^N values for $\boldsymbol{x}$ and so evaluation and marginalization will scale exponentially with the length N. We can, however, obtain a more **efficient** algorithm by **exploiting the conditional independence**.

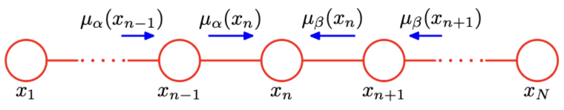
Consider the summation over x_N. The potential $\psi_{N-1,N}(x_{N-1}, x_N)$ is the only one that depends on x_N, so we can perform the summation

$$\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

first to give a function of x_{N-1}. We can then use this to perform the summation over x_{N-1}, which will involve only this new function and $\psi_{N-2,N-1}(x_{N-2}, x_{N-1})$, and so on. Because each summation effectively removes a variable from the distribution, this can be viewed as the removal of a node from the graph.

Computational costs. We have to perform $N - 1$ summations each of which is over K states and involves a function of two variables. For instance, the summation of x_1 involves only the function

$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1},x_n)\cdots\left[\sum_{x_2}\psi_{2,3}(x_2,x_3)\left[\sum_{x_1}\psi_{1,2}(x_1,x_2)\right]\right]\cdots\right]}_{\mu_\alpha(x_n)} \underbrace{\left[\sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\cdots\left[\sum_{x_N}\psi_{N-1,N}(x_{N-1},x_N)\right]\cdots\right]}_{\mu_\beta(x_n)}. \quad (8.52)$$



$\psi_{1,2}(x_1,x_2)$, which is a table of $K \times K$ numbers. We have to sum this table over x_1 for each value of x_2 so $O(K^2)$ cost. The resulting vector of K numbers is multiplied by the matrix of numbers $\psi_{2,3}(x_2,x_3)$ and so $O(K^2)$ cost. There are $N-1$ summations of multiplications of this kind, the **total cost of evaluating the marginal** $p(x_n)$ is $O(NK^2)$. This is **linear** in the length of the chain.

6.5.1 Message passing

We now give a powerful interpretation of this calculation in terms of the passing of *local messages* around on the graph.

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

We shall interpret $\mu_\alpha(x_n)$ as a message passed forwards along the chain from x_{n-1} to x_n. The message $\mu_\alpha(x_n)$ can be evaluated recursively because

$$\begin{aligned}\mu_\alpha(x_n) &= \sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\Big[\sum_{x_{n-2}}\cdots\Big]\\ &= \sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\mu_\alpha(x_{n-1})\end{aligned}$$

We therefore first evaluate

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

and then apply above repeatedly until we reach the desired node. Finally, consider joint distribution for two neighboring nodes. This is similar to the evaluation of the marginal for a single node, except that there are now two variables that are not summed out.

$$p(x_{n-1},x_n) = \frac{1}{Z}\mu_\alpha(x_{n-1})\psi_{n-1,n}(x_{n-1},x_n)\mu_\beta(x_n)$$

Thus we can obtain the joint distributions over all of the sets of variables in each of the potentials directly once we have completed the message passing required to obtain the marginals.

6.6 Inference on a tree

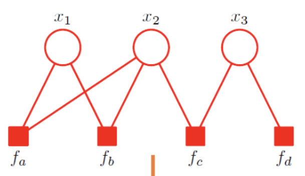
We now extend this toward the inference on a tree or graph.

6.6.1 Factor graph

Generalization of all previous directed and undirected cases. X_s denotes a set of variables. f_S is a function of variables. A factor, represented by a red square, represents a joint distribution of variables. The factor graph is for representing such factors more explicit and the factorization of the joint distribution becomes clear.

$$p(\boldsymbol{x}) = \prod_s f_s(\boldsymbol{x}_s)$$

Converting a graph to a factor graph is not deterministic (and may require domain knowledge).



A factor graph to its joint distribution

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

6.6.2 Sum-product algorithm

Utilizes the factor graph framework for the exact inference on tress. For simplicity, we assume discrete variables, *i.e.* belief propagation:

$$p(x) = \sum_{\boldsymbol{x}\setminus x} p(\boldsymbol{x})$$

Let's partition the factors in $p(\boldsymbol{x})$ into groups; each group associated with **each factor node** that is a neighbor of x, denoted ne(x).

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

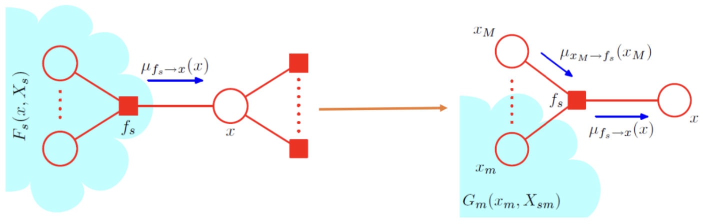
X_s denotes the set of all variables in the subtree connected to the x via f_s and $F_s(x,X_s)$ represents the product of all the factors in the group associated with f_s. Using this and interchanging $\sum$ and $\prod$ we obtain:

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

Here we define a **messages from the factor nodes** f_s to x:

$$\mu_{f_s\to x}(x) = \sum_{X_s} F_s(x,X_s)$$

The marginal $p(x)$ is the product of all the incoming messages.



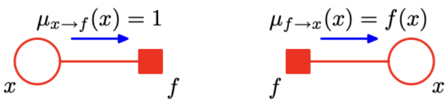
Note that $F_s(x,X_s)$ can itself be factorized.

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

where $x_1,\cdots,x_M$ are variables associated with f_x. The messages passed from variable nodes to factor nodes are defined as:

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

The algorithm begins with messages sent by the leaf nodes.



6.6.3 Max-sum algorithm

The sum-product algorithm is for calculating the marginal distribution given a joint probability expressed as a factor graph. The max-sum algorithm is for finding the largest probability values. Suppose the chain network we used before.

$$\begin{aligned}\max_{\boldsymbol{x}} p(\boldsymbol{x}) &= \frac{1}{Z}\max_{x_1}\cdots\max_{x_N}[\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\cdots] \\ &= \frac{1}{Z}\max_{x_1}[\psi_{1,2}(x_1,x_2)[\cdots\max_{x_N}\psi_{N-1,N}(x_{N-1},x_N)]]\end{aligned}$$

For each option of x_N, calculate in the reverse order for the following computation (with memorization), *i.e.* dynamic programming. Extend this to a tree, but using log-probability for numerical stability since the product of probabilities quickly decays to zero.

$$\mu_{f\to x}(x) = \max_{x_1,\cdots,x_M}\left[\ln f(x,x_1,\cdots,x_M) + \sum_{m\in \mathrm{ne}(f_s)\backslash x}\mu_{x_m\to f}(x_m)\right]$$

$$\mu x \to f(x) = \sum_{l\in\mathrm{ne}(x)\backslash f}\mu_{f_l\to x}(x)$$

7 Variational inference

A *functional* L is a mapping from function to value, *e.g.* entropy $H[p]$. The variational method is to solve the $\underset{f}{\operatorname{argmin}} L[f]$.

Suppose a Bayesian model where X denotes a set of observed variables and Z denotes a set of hidden variables and parameters. Our probabilistic model models $p(X,Z)$. We are interested in $p(Z|X)$. $q(Z)$ is for approximating $p(Z|X)$ when $p(Z|X)$ is complicated. We say $\mathcal{L}(q)$ a evidence lower bound (ELBO), and $\ln p(\boldsymbol{X}) \ge \mathrm{ELBO}(q)$.

$$\begin{aligned}\ln p(\boldsymbol{X}) &= \mathcal{L}(q) + \mathrm{KL}(q\|p) \\ \mathcal{L}(q) &= \int q(\boldsymbol{Z})\ln\left\{\frac{p(\boldsymbol{X},\boldsymbol{Z})}{q(\boldsymbol{Z})}\right\}d\boldsymbol{Z} \\ \mathrm{KL}(q\|p) &= -\int q(\boldsymbol{Z})\ln\left\{\frac{p(\boldsymbol{Z}|\boldsymbol{X})}{q(\boldsymbol{Z})}\right\}d\boldsymbol{Z} \\ q_*(z) &= \operatorname{argmin}\mathrm{KL}(q\|p)\end{aligned}$$

However, the exact KL divergence calculation is intractable:

$$\begin{aligned}\mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})) &= \int q(\mathbf{z})\log\frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}d\mathbf{z} \\ &= \int q(\mathbf{z})\big(\log q(\mathbf{z}) - \log p(\mathbf{z}|\mathbf{x})\big)d\mathbf{z} \\ &= \int q(\mathbf{z})\log q(\mathbf{z}) - q(\mathbf{z})\log p(\mathbf{z}|\mathbf{x})d\mathbf{z} \\ &= \int q(\mathbf{z})\log q(\mathbf{z})d\mathbf{z} - \int q(\mathbf{z})\log p(\mathbf{z}|\mathbf{x})d\mathbf{z} \\ &= \mathbb{E}_q\big[\log q(\mathbf{z})\big] - \mathbb{E}_q\big[\log p(\mathbf{z}|\mathbf{x})\big] \\ &= \mathbb{E}_q\big[\log q(\mathbf{z})\big] - \mathbb{E}_q\left[\log\frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})}\right] \\ &= \mathbb{E}_q\big[\log q(\mathbf{z})\big] - \mathbb{E}_q\big[\log p(\mathbf{x},\mathbf{z}) - \log p(\mathbf{x})\big] \\ &= \mathbb{E}_q\big[\log q(\mathbf{z}) - \log p(\mathbf{x},\mathbf{z})\big] + \mathbb{E}_q\big[\log p(\mathbf{x})\big] \\ &= \mathbb{E}_q\big[\log q(\mathbf{z}) - \log p(\mathbf{x},\mathbf{z})\big] + \underbrace{\log p(\mathbf{x})}_{\text{intractable}}\end{aligned}$$

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})} \qquad p(\mathbf{x}) = \int p(\mathbf{z},\mathbf{x})d\mathbf{z}.$$

The negative ELBO

$$\mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})) = \boxed{\mathbb{E}_q\big[\log q(\mathbf{z}) - \log p(\mathbf{x},\mathbf{z})\big]} + \log p(\mathbf{x}).$$

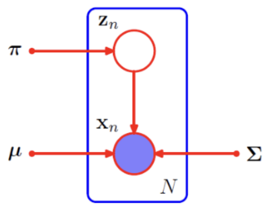
$$\mathrm{ELBO}(q) = \mathbb{E}_q\big[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z})\big] = \mathbb{E}_q\left[\log\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]$$

7.1 Mixture of Gaussians

There are K Gaussians. z_n is a one-hot vector which denotes a membership. π,μ,Σ are mixing coefficients, means, and precisions, respectively.

$$p(Z|\pi) = \prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{z_{nk}}$$

$$p(X|Z,\mu,\Lambda) = \prod_{n=1}^{N}\prod_{k=1}^{K}\mathcal{N}(x_n|\mu_k,\Lambda_k^{-1})^{z_{nk}}$$



We extend this to the variational mixture of Gaussians. We use the Dirichlet distribution for π, which is a conjugate prior for the multinomial likelihood. (A similar relationship can be observed between the beta prior and the binomial likelihood.)

$$f(x_1,\cdots,x_k;\alpha_1,\cdots\alpha_k) = \frac{1}{B(\alpha)}\prod_{i=1}^{k}x_i^{\alpha_i-1}$$

$$\text{where } B(\alpha) = \frac{\prod_{i=1}^{k}\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i)}$$

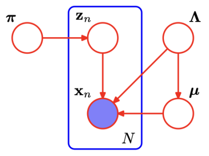
where gamma function Γ is extension for factorial to complex numbers. Using the Dirichlet prior is not only semantically correct but also mathematically advantageous.

$$p(\pi) = \mathrm{Dir}(\pi|\alpha_0) = C(\alpha_0)\prod_{k=1}^{K}\pi_K^{\alpha_0-1}$$

We use the Gaussian-Wishart prior for μ,Σ.

$$\begin{aligned}p(\mu,\Lambda) &= p(\mu|\Lambda)p(\Lambda) \\ &= \prod_{k=1}^{K}\mathcal{N}(\mu_k|m_0,(\beta_0\Lambda_k)^{-1})\mathcal{W}(\Lambda_k|W_0,\nu_0)\end{aligned}$$

7.1.1 Bayesian Gaussian mixture model



$$p(X,Z,\pi,\mu,\Lambda) = p(X|Z,\mu,\Lambda)p(Z|\pi)p(\pi)p(\mu|\Lambda)p(\Lambda)$$

Let's use variational distribution q which factorizes $q(Z,\pi,\mu,\Lambda) = q(Z)q(\pi,\mu,\Lambda)$ for simplicity. Generally, $q(Z) = \prod_{i=1}^{M}q_i(Z_i)$. Its ELBO is defined as follows.

$$\begin{aligned}\mathcal{L}(q) &= \int\prod_i q_i\{\ln p(X,Z) - \sum_i \ln q_i\}dZ \\ &= \int q_j\underbrace{\left\{\int \ln p(X,Z)\prod_{i\neq j}q_j dZ_i\right\}}_{=\mathbb{E}_{i\neq j}[\ln p(X,Z)]=\ln\tilde{p}(X,Z_j)+\text{const}}dZ_j - \int q_j\ln q_j dZ_j + \text{const} \\ &= \underbrace{\int q_j\ln\tilde{p}(X,Z_j)dZ_j - \int q_j\ln q_j dZ_j}_{\text{negative KL divergence}} + \text{const}\end{aligned}$$

The negative KL divergence is maximized when $q_j(Z_j) = \tilde{p}(X,Z_j)$. The optimal q_j is obtained by other nodes' joint distribution.

$$\ln q_j^\star(\mathbf{Z}_j) = \mathbb{E}_{i\neq j}[\ln p(\mathbf{X},\mathbf{Z})] + \boxed{\text{const}}$$

$$q_j^\star(\mathbf{Z}_j) = \frac{\exp\left(\mathbb{E}_{i\neq j}[\ln p(\mathbf{X},\mathbf{Z})]\right)}{\boxed{\int\exp\left(\mathbb{E}_{i\neq j}[\ln p(\mathbf{X},\mathbf{Z})]\right)\,\mathrm{d}\mathbf{Z}_j}}$$

We typically do not use $q^\star$ due to the normalization constant

$$\begin{aligned}\ln q^*(Z) &= \mathbb{E}_{\pi,\mu,\Lambda}[\ln p(X,Z,\pi,\mu,\Lambda)] + \text{const} \\ &= \mathbb{E}_{\pi}[\ln p(Z|\pi)] + \mathbb{E}_{\mu,\Lambda}[\ln p(X|Z,\mu,\Lambda)] + \text{const}\end{aligned}$$