EE5907 Pattern Recognition

Liu Zichen @ April 28, 2019

1 Statistical Learning

1.1 Basics

A random variable x is a quantity that is uncertain. Some values occur more than others; this information captured by probability distribution $p(x)$.

Bernoulli: $\Pr(x) = \lambda^x(1-\lambda)^{1-x}$; Categorical: $Pr(x=k) = \lambda_k$; Gaussian: $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Marginalization: $p(x) = \sum_y p(x,y)$. Conditional probability: $p(x|y=y^*) = \frac{p(x,y=y^*)}{p(y=y^*)} = \frac{p(x,y=y^*)}{\int p(x,y=y^*)dx}$ or $p(x|y) = \frac{p(x,y)}{p(y)}$. Bayes' rule: $p(y|x) = \frac{p(y)p(x|y)}{p(x)} = \frac{p(y)p(x|y)}{\sum_y p(y)p(x|y)}$.

Independence: $p(x,y) = p(x)p(y|x) = p(x)p(y)$.

Expectation: $E[f(x)] = \sum_x f(x)p(x)$, if $f(x) = x$, mean; if $f(x) = (x-\mu_x)^2$, variance. $\mathrm{Cov}(x,y) = E(xy) - E(x)E(y)$. Conditional expectation: $E[f(x,y)|y] \triangleq E_{p(x|y)}[f(x,y)] \triangleq \int_x f(x,y)p(x|y)dx$.

Conditional independent: $p(x_1,x_2|x_3) = p(x_1|x_3)\,p(x_2|x_3)$.

Suppose $Y = g(X)$, where g is monotonic; Change of variable: If $X \sim f(x)$, then $Y \sim \left|\frac{d}{dy}\left(g^{-1}(y)\right)\right| f\left(g^{-1}(y)\right)$.

$Ga(x|a,b) = \frac{b^a}{\Gamma(a)} x^{a-1}e^{-xb}$, $\Gamma(t+1) = t\Gamma(t)$.

KL divergence: $D_{\mathrm{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$.

Leibniz rule: $\frac{d}{dx}\left(\int_{a(x)}^{b(x)} f(x,t)dt\right) = f(x,b(x))\cdot \frac{d}{dx}b(x) - f(x,a(x))\cdot\frac{d}{dx}a(x) + \int_{a(x)}^{b(x)}\frac{\partial}{\partial x}f(x,t)dt$

1.2 Estimation and Conjugate Priors

MAP estimation: $y_{MAP} \triangleq \underset{y}{\mathrm{argmax}} p(y|x) = \underset{y}{\mathrm{argmax}} \frac{p(y)p(x|y)}{p(x)} = \underset{y}{\mathrm{argmax}} p(y)p(x|y)$. If the prior $p(y)$ is constant (uniform), $y_{MAP} = \underset{y}{\mathrm{argmax}} p(x|y) \triangleq y_{ML}$.

Much of ML is about how to choose a model for $p(y|x)$ and how to optimize model parameters. To model posterior $p(y|x,\boldsymbol{\theta})$, we have **generative** (model $p(x|y,\boldsymbol{\theta})$ and $p(y|\boldsymbol{\theta})$ so that can generate new data $p(x,y|\boldsymbol{\theta}) = p(y|\boldsymbol{\theta})p(x|y,\boldsymbol{\theta})$) and **discriminative** models ($P(y|x,\boldsymbol{\theta})$ only). We treat the parameter $\boldsymbol{\theta}$ as random variable and perform *probabilistic estimation*.

- Beta-Binomial Generative Model

$p(D|\theta) = \mathrm{Bin}\left(N_1|N,\theta\right) = \binom{N}{N_1}\theta^{N_1}(1-\theta)^{N_0}$, $p(\theta|a,b) = \mathrm{Beta}(\theta|a,b) = \frac{1}{B(a,b)}\theta^{a-1}(1-\theta)^{b-1}$, $p(\theta|D) = \mathrm{Beta}\left(\theta|N_1+a, N_0+b\right)$.

For $\mathrm{Beta}(x|c,d)$, mode $= \frac{c-1}{c+d-2}$, mean $= \frac{c}{c+d}$, so $\hat{\theta}_{MAP} = \underset{\theta}{\mathrm{argmax}} p(\theta|D) = \frac{N_1+a-1}{N+a+b-2}$ and $\hat{\theta}_{ML} = \frac{N_1}{N}$ by setting $a=b=1$ (uniform prior) . $\hat{\theta}_{MAP} = (1-\lambda)\hat{\theta}_{ML} + \lambda m_0$, $m_0 = \frac{a-1}{a+b-2}$ (prior mode) . ***To predict future*** coin toss, plug-in: $p(\tilde{x}=1) = \theta_{ML}$ or $p(\tilde{x}=1) = \theta_{MAP}$; posterior predictive: $p(\tilde{x}=1|D) = E(\theta|D) = \frac{N_1+a}{N+a+b}$.

- Dirichlet-Multinomial Generative Model

$p(D|\theta) = \frac{N!}{N_1!\cdots N_K!}\prod_{k=1}^K \theta_k^{N_k}$, $p(\theta|\alpha) = \mathrm{Dir}(\theta|\alpha), \alpha = \{\alpha_k\}_{k=1:K}, \alpha_k > 0$, $p(\theta|D) = \mathrm{Dir}\left(\theta|N_1+\alpha_1,\cdots,N_K+\alpha_K\right)$. $\hat{\theta}_k^{MAP} = \frac{N_k+\alpha_k-1}{N+\sum_k\alpha_k-K}$, $\hat{\theta}_k^{ML} = \frac{N_k}{N}$ by setting $\alpha_k = 1$ (uniform prior) . Again, posterior predictive: $p(\tilde{x}=i|D) = \frac{N_i+\alpha_i}{N+\sum_k\alpha_k}$. (more important since the data is more sparse).

Posterior predictive: $p\left(x_{n+1}|D\right) = \int p\left(x_{n+1}|\lambda\right)p\left(\lambda|D\right)d\lambda$, or $p\left(x_{N+1}|D\right) = \frac{p\left(x_{N+1}|\lambda,D\right)p\left(\lambda|D\right)}{p\left(\lambda|x_{N+1},D\right)} = \frac{p\left(x_{N+1}|\lambda\right)p\left(\lambda|D\right)}{p\left(\lambda|x_{N+1},D\right)}$.

1.3 Univariate Gaussian and Naive Bayes

$p(x) = \mathcal{N}\left(x|\mu,\sigma^2\right)$, ML estimation: $\hat{\mu} = \frac{1}{N}\sum_{n=1}^N x_n$, $\hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^N (x_n-\hat{\mu})^2$; $p\left(\mu,\sigma^2\right) = \mathrm{NormInvGam}\left(\mu,\sigma^2|\alpha,\beta,\gamma,\delta\right)$, MAP estimation: $\hat{\mu} = \frac{\sum_{n=1}^N x_n+\gamma\delta}{N+\gamma}$, $\hat{\sigma}^2 = \frac{\sum_{n=1}^N(x_n-\hat{\mu})^2+2\beta+\gamma(\delta-\hat{\mu})^2}{N+3+2\alpha}$. ***To predict future*** Gaussian sample, ML:$p\left(x^*\right) = \mathcal{N}\left(\mu_{ML},\sigma^2_{ML}\right)$; MAP: $p\left(x^*\right) = \mathcal{N}\left(\mu_{MAP},\sigma^2_{MAP}\right)$; or posterior predictive: $p\left(x^*|D\right)$. (omitted here).

- Naive Bayes Classifier

We need to model 1) class prior: $P\left(Y=c_k\right), \quad k=1,2,\cdots,K$ and 2) feature likelihood: $P\left(X=x|Y=c_k\right) = \prod_{j=1}^n P\left(X^{(j)}=x^{(j)}|Y=c_k\right)$ (Naive). To classify: $y = \arg\max_{c_k} P\left(Y=c_k\right)\prod_{j=1}^n P\left(X^{(j)}=x^{(j)}|Y=c_k\right)$.

1.4 Logistic Regression

$p(y=1|x) = \frac{1}{1+e^{-w^Tx}}$, $p(y=0|x) = \frac{1}{1+e^{w^Tx}}$.

Log odds is $\log\frac{p(y=1|x)}{p(y=0|x)} = w^Tx$. $NLL(w) = -\sum_{i=1}^N\left[y_i\log\mu_i + (1-y_i)\log\left(1-\mu_i\right)\right]$, where $\mu_i = \frac{1}{1+\exp\left(-w^Tx_i\right)}$. We want to minimize $NNL(w)$ as optimization problem.

$g = \frac{d}{dw}NLL(w) = X^T(\mu-y)$, $H = \frac{d}{dw}g(w)^T = X^TSX$, where $S = diag_{N\times N}(\mu_i(1-\mu_i))$. Impose regularization to avoid overfitting: $NLL_{reg}(\mathbf{w}) = NLL(\mathbf{w}) + \frac{1}{2}\lambda w^Tw$,

$$g_{reg}(\mathbf{w}) = g(\mathbf{w}) + \lambda\begin{pmatrix}0_{1\times1}\\ w_{D\times1}\end{pmatrix},$$

$$H_{reg}(\mathbf{w}) = H(\mathbf{w}) + \lambda\begin{pmatrix}0_{1\times1} & \cdots\\ \vdots & I_{D\times D}\end{pmatrix}.$$

1.5 Non-parametric Techniques

In density estimation, we don't know the type of the distribution; in classification, we don't know the kind of mapping function → **non-parametric** approach (does not mean no parameter!; number and nature of parameters change with data!).

Given $x_1,\cdots,x_N$ i.i.d from unknown $p(x)$, For particular x, count number of samples k falling in window of volume V centered at x. $p(x) \approx \frac{k/N}{V}$.

- Parzen window: fix V, estimate k

Hypercube: $\phi(u) = \begin{cases}1 & |u_j| \le 0.5, j=1,\cdots,d\\ 0 & \text{otherwise}\end{cases}$, $k(x) = \sum_i\phi\left(\frac{x-x_i}{h}\right) =$ # samples in hypercube centered at x, $p(x) = \frac{k(x)/N}{V} = \frac{\sum_i\phi\left(\frac{x-x_i}{h}\right)/N}{h^d} = \frac{1}{N}\sum_{i=1}^N\frac{1}{h^d}\phi\left(\frac{x-x_i}{h}\right)$. Can take different h and test it on $\sum_i\log p_h\left(x_i\right)$, pick highest log probability.

- k-NN: grow V until capture a certain k

$p_N(x) = \frac{k_N(x)/N}{V_N}$. Mostly used for non-parametric classification:

Joint probability $p(x,y=c) = \frac{k_c/N}{V}$

Posterior $p(y=c|x) = \frac{p(x,y=c)}{\sum_{c'=1}^C p(x,y=c')} = \frac{(k_c/N)/V}{\sum_{c'=1}^C\left(k_{c'}/N\right)/V} = \frac{k_c}{\sum_{c'=1}^C k_{c'}} = \frac{k_c}{K}$

Distance metric: $\mathrm{dist}(a,b) = \left(\sum_{j=1}^D|a_j-b_j|^p\right)^{1/p}$.

1.6 Bayesian Statistics

A few topics about Bayesian statistics: Bayesian way is to use posterior, one of which is MAP plug-in approximation; but it can overfit, and sensitive to parametrization. But ML or posterior predictive don't have such problem.

Bayesian model selection. $\hat{m} = \underset{m}{\mathrm{argmax}} p(m|D) = \underset{m}{\mathrm{argmax}} p(D|m)p(m) = \underset{m}{\mathrm{argmax}} p(D|m)$, $p(D|m)$ is called **marginal likelihood**. Previously, if we want to estimate model parameter θ: $\theta_{MAP} = \underset{\theta}{\mathrm{argmax}} p(\theta|D) = \underset{\theta}{\mathrm{argmax}}\frac{p(\theta)p(D|\theta)}{p(D)}$ and we can drop $p(D)$; but consider the implicit model choice, becomes: $\frac{p(\theta|m)p(D|\theta,m)}{p(D|m)}$, and the one we drop is actually marginal likelihood $p(D|m)$ (because it's constant when we only consider one model).

$BIC(M,\mathcal{D}) \triangleq \log p\left(\mathcal{D}|\hat{\boldsymbol{\theta}}_{MLE}\right) - \frac{\mathrm{dof}(M)}{2}\log N$, larger BIC, better model. Bayesian factor: $BF_{1,0} = \frac{p(D|M_1)}{p(D|M_0)}$.

Bayesian decision theory. Bayes' estimator: $\delta(x) = \underset{a\in\mathcal{A}}{\mathrm{argmin}}\rho(a|x)$, where $\rho(a|x) = \mathbb{E}_{p(y|x)}[L(y,a)] = \sum_y L(y,a)p(y|x)$. [nature picks $y\in\mathcal{Y}$ and generates observation $x\in\mathcal{X}$, we choose action $a\in\mathcal{A}$ with minimum loss.]

MAP minimizes 0-1 loss: $\rho(a|x) = \sum_y L(y,a)p(y|x) = p(y\neq a|x) = 1-p(y=a|x)$;

MMSE corresponds to posterior mean, which minimizes l_2 loss: $\rho(a|x) = \sum_y L(y,a)p(y|x) = E\left[(y-a)^2|x\right] = E\left(y^2|x\right) - 2aE(y|x) + a^2$, $\frac{\partial}{\partial a}\rho(a|x) = -2E(y|x) + 2a = 0 \Longrightarrow a = E(y|x)$; posterior median minimizes l_1 loss.

We can threshold the decision by false negative loss and false positive loss: $\frac{p(y=1|x)}{p(y=0|x)} > \frac{L_{FP}}{L_{FN}} = \tau$. We can vary τ to draw ROC.

Mixture of conjugate priors. $p(\theta) = \sum_k p(z=k)p(\theta|z=k)$, so posterior $p(\theta|D) = \sum_k p(z=k|D)p(\theta|D,z=k)$.

2 Representation Learning

2.1 Unsupervised: PCA, NMF (& ICA)

In ***PCA***, we want to capture the big **variability** and ignore the small variability, and after ordering, we choose the first k "direction" to project the raw data on. Each PC is **orthogonal** to each other. Given data: $\{x_1,x_2,\ldots,x_n\}\subset\mathbb{R}^d$ Using 1^{st} PC we project data onto $z_1 = a_1^Tx$, a_1 is chosen such that $\mathrm{var}\left[z_1\right] = a_1^TSa_1$ is maximized (under the constraint $a_1^Ta_1 = 1$), where S is the covariance matrix. The solution: a_1 is the eigenvector of S corresponding to the largest eigenvalue. And in general, $\mathrm{var}\left[z_k\right] = a_k^TSa_k = \lambda_k$, the k^{th} PC z_k retains the k^{th} greatest variation.

STEPS:

1. covariance matrix: $S = \frac{1}{n}\sum_{i=1}^n\left(x_i-\overline{x}\right)\left(x_i-\overline{x}\right)^T$

2. eigenvectors: $\{a_i\}_{i=1}^d$
3. first p eigenvectors: $\{a_i\}_{i=1}^p$
4. transformation: $G \leftarrow [a_1, a_2, \ldots, a_p]_{d\times p}$
5. $y = G^T x$

Using SVD:

- centering: $X_{d,n} = [(x_1 - \overline{x}), \ldots, (x_n - \overline{x})]$
- SVD: $X = U_{d,d} D_{d,n} (V_{n,n})^T$ $(S = XX^T = UD^2U^T)$
- reconstruct: $\tilde{x}_i = \overline{x} + U_{d,p} U_{d,p}^T (x_i - \overline{x})$

A criterion for p: $\frac{\sum_{i=1}^p \lambda_i}{\sum_{i=1}^d \lambda_i} \geq Threshold(e.g. 0.95)$

In ***NMF***, we express an image vector as the linear combination of a set of basis images. Unlike PCA, the coefficients are positive only (PCA involves subtraction due to some negative values after de-mean). Factorization: $\mathrm{V_{n,m}} \approx \mathrm{W_{n,r} H_{r,m}}$. V contains m images, one of which has n non-negative pixel values; W has r columns of basis images; each column in H is called encoding.

STEPS:

1. objective: $\min_{W,H} \|V - WH\|^2 \quad \text{s.t.} \quad W \geq 0, H \geq 0$
2. a pixel: $V_{i\mu} = (WH)_{i\mu} = \sum_{a=1}^r W_{ia} H_{a\mu}$
3. gradient descent: $H_{a\mu} \leftarrow H_{a\mu} + \eta_{a\mu} \left[\left(W^T V\right)_{a\mu} - \left(W^T WH\right)_{a\mu} \right]$
4. multiplicative update rule: $H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T WH)_{a\mu}}$ $(\eta_{a\mu} = \frac{H_{a\mu}}{(W^T WH)_{a\mu}})$

In *summary*: PCA has holistic representation, while NMF has part-based; PCA uses eigenfaces as basis image, while NMF uses localized features; each face in PCA is a linear combination of all eigenfaces, while in NMF only additive combinations.

2.2 Supervised: LDA, GE

LDA finds the most discriminative projection by *maximizing* between-class distance and *minimizing* within-class distance. We define:

Class-specific mean: $\mu_i = \frac{1}{n_i} \sum_{\mathbf{x}\in C_i} \mathbf{x}$

Class-specific covariance: $\mathbf{S}_i = \frac{1}{n_i} \sum_{\mathbf{x}\in C_i} (\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^T$

Total mean: $\mu = \frac{1}{N} \sum_{\mathbf{x}} \mathbf{x}$

Within-class scatter: $\mathbf{S}_W = \sum_{i=1}^C \frac{n_i}{N} \mathbf{S}_i = \sum_{i=1}^C P_i \mathbf{S}_i$

Between-class scatter: $\mathbf{S}_B = \sum_{i=1}^C P_i (\mu_i - \mu)(\mu_i - \mu)^T$

Total covariance:
$$\mathbf{S}_T = \frac{1}{N} \sum_{\mathbf{x}} (\mathbf{x} - \mu)(\mathbf{x} - \mu)^T = \mathbf{S}_W + \mathbf{S}_B$$

In *two-class case*, given data: $\{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$, we seek a projection w (onto a scalar): $y = w^T x$. Thus within-class scatter: $\tilde{S}_i^2 = \sum_{x\in c_i} \left(w^T x - w^T \mu_i\right)^2$; between-class scatter: $J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = \left|w^T (\mu_1 - \mu_2)\right|$. And Fisher linear discriminant is to maximize $J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{S}_1^2 + \tilde{S}_2^2} = \frac{w^T S_B w}{w^T S_W w}$. (NOTE that S_B is the outer product of two vectors, its rank is at most 1). Taking derivative and setting to zero yield: $\left(w^T S_W w\right) S_B w - \left(w^T S_B w\right) S_W w = S_B w - J(w) S_W w = 0 \Rightarrow S_W^{-1} S_B w = J(w) w$ (**Eigenvalue decomposition** - since S_B is at most rank 1, only take the first eigenvalue)

In *multi-class case*, we can project on $[y_1, y_2, \ldots y_{C-1}]$, hence projection is a matrix $W = [w_1 | w_2 | \ldots | w_{C-1}]$. We want to maximize $J(W) = \frac{|\tilde{S}_B|}{|\tilde{S}_W|} = \frac{|W^T S_B W|}{|W^T S_W W|}$, and the optimal projection W^* is the one whose columns are the eigenvectors corresponding to the largest eigenvalues of the following generalized eigenvalue problem $W^* = \arg\max \frac{|W^T S_B W|}{|W^T S_W W|} \Rightarrow (S_B - \lambda_i S_W) w_i^* = 0$, where $\lambda_i = J(w_i) = \text{scalar}$. **Limitations** of LDA are 1) at most $C - 1$ feature projections; 2) parametric (assumes unimodal Gaussian likelihoods).

Graph embedding is a general framework for feature extraction. We define intrinsic graph $G = [x_i, S_{ij}]$ and penalty graph $G^P = \left[x_i, S_{ij}^P\right]$, S and S^P are similarity matrices, L and B are Laplacian matrices from S and S^P $(L = D - S, \quad D_{ii} = \sum_{j\neq i} S_{ij} \quad \forall i)$. Data in high-dimensional space and low-dimensional space (1-D assumed): $X = [x_1, x_2, \ldots, x_N] \quad y = [y_1, y_2, \ldots, y_N]^T$

To preserve graph similarity:

$$y^* = \underset{y^T y=1 \text{ or } y^T By=1}{\arg\min} \sum_{i\neq j} \|y_i - y_j\|^2 S_{ij} = \underset{y^T y=1 \text{ or } y^T By=1}{\arg\min} y^T Ly$$

After linearization $y = X^T w$,

$$w^* = \underset{w^T w=1 \text{ or } w^T XBX^T w=1}{\arg\min} w^T XLX^T w$$

Solution can be obtained by solving $\tilde{\boldsymbol{L}}\boldsymbol{v} = \lambda \tilde{\boldsymbol{B}}\boldsymbol{v}$ where $\tilde{L} = XLX^T$ and $\tilde{B} = XBX^T$.

Letting $L = D - W$ and $B = D^P - W^P$, we have *Marginal Fisher Criterion*:

$w^* = \arg\min_w \frac{w^T X(D-W)X^T w}{w^T X(D^P - W^P)X^T w}$, where $W_{ij} = W_{ji} = 1$ if x_i is among the k_1-nearest neighbours of x_j in the same class (intraclass compactness); $W_{ij}^p = 1$ for the pair (i, j) if x_i is among the k_2-nearest neighbours of x_j in different classes. Compared to LDA no distribution assumption, more projection directions, interclass margin can better characterize the separability of different classes.

2.3 Clustering

There is an important distinction between *hierarchical (organized as a hierarchical tree)* and *partitional (into non-overlapping subsets)* sets of clusters. Clusters can also be fuzzy such that a point belongs to every cluster with some probability. Some types of clusters: well-separated, center-based, contiguous, density-based.

K-means is a partitional clustering method, which simply initialize K random centroids, and repeats 1) assigning all points to the closest centroid; 2) recompute the centroid of each cluster. Solutions to initial centroid problem: a) multiple runs; b) use hierarchical clustering to determine initial centroids; c) select more than k initial centroids then choose from them. Most common measure for evaluation is *Sum of Squared Error*: $SSE = \sum_{i=1}^K \sum_{x\in C_i} dist^2(m_i, x)$. We can use the "elbow finding" method to decide K.

Hierarchical clustering can be visualized as a dendrogram, which is a tree-like diagram that records the sequence of merges or splits. Clusters are produced when cutting at proper level. It can be agglomerative or divisive. Algorithm for agglomerative clustering: 1) compute proximity matrix; 2) let each point be a cluster; 3) repeat: [a. merge two closest clusters, b. update the proximity matrix] until only one cluster remains. We can use *MIN (noise-sensitive), MAX (biased towards globular clusters), Group Average (same as MAX) or Distance Between Centroid* to define the inter-cluster similarity.

3 Models

3.1 Gaussian Mixture Model, Boosting

GMM is generative while boosting is discriminative.

GMM: $P(y|\theta) = \sum_{k=1}^K \alpha_k \phi(y|\theta_k)$, where $\alpha_k \geqslant 0, \quad \sum_{k=1}^K \alpha_k = 1$ and $\phi(y|\theta_k) = \phi\left(y|\left(\mu_k, \sigma_k^2\right)\right) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(y-\mu_k)^2}{2\sigma_k^2}\right)$.

EM algorithm: After initializing $\theta^{(0)}$, compute

$$Q\left(\theta, \theta^{(i)}\right) = E_Z\left[\log P(Y, Z|\theta)|Y, \theta^{(i)}\right] = \sum_z \log P(Y, Z|\theta) P\left(Z|Y, \theta^{(i)}\right), \text{ and}$$

then update $\theta^{(i+1)} = \arg\max_\theta Q\left(\theta, \theta^{(i)}\right)$ until convergence.

Back in GMM, the latent variable is r.v. $\gamma_{jk} = \begin{cases} 1, \text{observation j from component k} \\ 0, \text{otherwise} \end{cases}$. Its expectation $\hat{\gamma}_{jk} = \frac{\alpha_k \phi(y_j|\theta_k)}{\sum_{k=1}^K \alpha_k \phi(y_j|\theta_k)}$. To maximize the Q function (partial and set to zero): $\hat{\mu}_k = \frac{\sum_{j=1}^N \hat{\gamma}_{jk} y_j}{\sum_{j=1}^N \hat{\gamma}_{jk}}$, $\hat{\sigma}_k^2 = \frac{\sum_{j=1}^N \hat{\gamma}_{jk} (y_j - \mu_k)^2}{\sum_{j=1}^N \hat{\gamma}_{jk}}$, $\hat{\alpha}_k = \frac{n_k}{N} = \frac{\sum_{j=1}^N \hat{\gamma}_{jk}}{N}$.

Boosting defines a strong classifier using an additive model of many weak classifier. (Omitted here)

3.2 Support Vector Machines

Separating hyper-plane: $w^* \cdot x + b^* = 0$ or decision function $f(x) = \text{sign}(w^* \cdot x + b^*)$ for binary classification. Functional margin: $\hat{\gamma}_i = y_i (w \cdot x_i + b)$; geometric margin: $\gamma_i = y_i \left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right)$. If we want to maximize the geometric margin, we need

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i (w \cdot x_i + b) - 1 \geqslant 0, \quad i = 1, 2, \cdots, N$$

to find w^* and b^*. Lagrange function: $L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^N \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^N \alpha_i$,

$$\text{partial} \quad \begin{matrix} \nabla_w L(w, b, \alpha) = w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \\ \nabla_b L(w, b, \alpha) = \sum_{i=1}^N \alpha_i y_i = 0 \end{matrix},$$

plug back

$$L(w, b, \alpha) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^N \alpha_i y_i \left(\left(\sum_{j=1}^N \alpha_j y_j x_j \right) \cdot x_i + b \right) + \sum_{i=1}^N \alpha_i = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_{i=1}^N \alpha_i$$

Its dual is

$$\min_\alpha \quad \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^N \alpha_i \quad \text{s.t.} \quad \sum_{i=1}^N \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, \cdots, N$$

The resulted separating hyper-plane: $\sum_{i=1}^N \alpha_i^* y_i (x \cdot x_i) + b^* = 0$ and decision function $f(x) = \text{sign}\left(\sum_{i=1}^N \alpha_i^* y_i (x \cdot x_i) + b^*\right)$. For non-linearly separable data, introduce slack variable:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^N \xi_i \quad \text{s.t.} \quad y_i (w \cdot x_i + b) \geqslant 1 - \xi_i, \xi_i \geqslant 0, i = \cdots,$$

and its dual is

$$\min_\alpha \quad \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^N \alpha_i \quad \text{s.t.} \quad \sum_{i=1}^N \alpha_i y_i = 0, 0 \leqslant \alpha_i \leqslant C, i = 1, 2, \cdots, N$$

Using kernel trick, the kernel function $K(x, z) = \phi(x) \cdot \phi(z)$ and the mapping $\phi(x) : \mathcal{X} \to \mathcal{H}$. The objective function becomes $W(\alpha) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i$, and the separating hyper-plane becomes $f(x) = \text{sign}\left(\sum_{i=1}^{N_s} a_i^* y_i \phi(x_i) \cdot \phi(x) + b^*\right) = \text{sign}\left(\sum_{i=1}^{N_s} a_i^* y_i K(x_i, x) + b^*\right)$.

3.3 Neural Networks

Simple neuron: $h_{w,b}(x) = f\left(W^T x\right) = f\left(w^T x + b\right)$. f can be sign activation, sigmoid $(f(x) = \frac{1}{1+\exp(-x)})$, ReLU ... Weight updating: $w_i = w_i + \Delta w_i$; $\Delta w_i = \eta \sum_d (y_d - o_d) x_{id}$; $o_d = f\left(\sum_{i=0}^n w_i x_{id}\right)$.

Softmax (probability of class k given input): $P(c_k = 1|x) = \frac{e^{o_k}}{\sum_{j=1}^c e^{o_j}}$; crossentropy loss: $L(x, y; \theta) = -\sum_{j=1}^c y_j \log p(c_j|x)$. So learning is to find $\theta^* = \text{argmin}_\theta \sum_{n=1}^N L(x^n, y^n; \theta)$