

# **Machine Learning**

LI Tao

February 18, 2014



# Contents



# 1 Bayesian Decision Theory

**Bernoulli** :  $P(X) = p_0^X(1 - p_0)^{1-X}$ ,  $p_0$  is the param.

Estimation of  $p_0$  form  $\mathcal{X} = \{x^{(l)}\}_{l=1}^N$ :

$$\hat{p}_0 = \frac{\#heads}{\#tosses} = \frac{\sum_{l=1}^N x^{(l)}}{N}$$

Predict outcome = head if  $P_0 > 1/2$ , tail otherwise.

**Baye's Rule** :

$$\text{Posterior } P(C|\mathbf{x}) = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} = \frac{p(\mathbf{x}|C)P(C)}{p(\mathbf{x})}$$

Baye for  $K > 2$  classes:

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{\sum_{k=1}^K p(\mathbf{x}|C_k)P(C_k)}$$

Optimal decision: Choose  $C_i$  if  $P(C_i|\mathbf{x}) = \max_k (C_k|\mathbf{x})$

**Losses and Risks** :

$$R(\alpha_i|\mathbf{x}) = \sum_{k=1}^K \lambda_{ik} P(C_k|\mathbf{x}),$$

$\alpha_i$  the action assigned to class  $C_i$ ,  $\lambda_{ik}$  loss for  $\alpha_i$  if  $C_k$

Optimal:  $\alpha_i$  if  $R(\alpha_i|\mathbf{x}) = \min_k R(\alpha_k|\mathbf{x})$

**0-1 loss** :  $R(\alpha_i|\mathbf{x}) = \sum_{k=1}^K \lambda_{ik} P(C_k|\mathbf{x}) = 1 - P(C_i|\mathbf{x})$ ,  
 $\lambda_{ik} = 1$  if  $i \neq k$ , 0 otherwise

**Reject Option** : Loss:

$$\lambda_{ik} = \begin{cases} 1 & \text{if } i = k \\ \lambda & \text{if } i = K + 1 \\ 0 & \text{otherwise} \end{cases}$$

$\lambda$  is the loss for choosing reject.

Expected risk:

$$R(\alpha_i|\mathbf{x}) = \begin{cases} \sum_{k=1}^K \lambda P(C_k|\mathbf{x}) = \lambda & \text{if } i = K + 1 \\ \sum_{k \neq i} P(C_k|\mathbf{x}) = 1 - P(C_i|\mathbf{x}) & \text{if } i \in 1, \dots, K \end{cases}$$

Optimal Decision: choose  $C_i$  if

$$R(\alpha_i|\mathbf{x}) = \min_{1 \leq k \leq K} R(\alpha_k|\mathbf{x}) < R(\alpha_{K+1}|\mathbf{x}),$$

Reject otherwise.

**Discriminant Functions** : choose  $C_i$  if

$$g_i(\mathbf{x}) = \max_k g_k(\mathbf{x})$$

$$g_i(\mathbf{x}) = -R(\alpha_i|\mathbf{x}) = P(C_i|\mathbf{x}) = p(\mathbf{x}|C_i)P(C_i)$$

Two class may define single discriminant functions:  $g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$ , choose  $C_i$  if it greater than zero.

**Decision Regions**  $\mathcal{R}_i = \{\mathbf{x} | g_i(\mathbf{x}) = \max_k g_k(\mathbf{x})\}$

**Bayesian Network** Joint probability:

$$P(X_1, \dots, X_d) = \prod_{i=1}^d P(X_i | \text{parents}(X_i))$$

## 2 Parameter Estimation

### 2.1 Max Likelihood Estimation

#### 2.1.1 Settings

Assume data follow a distribution model, with probability density function  $f(\mathbf{x})$ .

Dataset:

$$\mathcal{X} = \{\mathbf{x}^{(l)}\}$$

Follows the probability distribution:

$$\mathbf{x}^l \sim p(\mathbf{x})$$

,

Assume some parametric form for  $p(\mathbf{x}|\theta)$ .  $\theta$  is the parameter (possibly a vector).

The goal: Estimate  $\theta$  using  $\mathcal{X}$

**Approach to classification** In Baye's rule for classification,  $p(\mathbf{x}|C_i)$ (likelihood) and  $P(C_i)$  (prior) need to be estimated from the sample

### 2.2 MLE

Seeks to find  $\theta$  that makes data samples from  $p(\mathbf{x}|\theta)$  as likely as possible

Likelihood:

$$L(\theta|\mathcal{X}) \equiv p(\mathcal{X}|\theta) = \prod_{l=1}^N p(\mathbf{x}^{(l)}|\theta)$$

Log likelihood:

$$\mathcal{L} \equiv \log L(\theta|X) = \sum_{l=1}^N \log p(\mathbf{x}^{(l)}|\theta)$$

**Max Likelihood estimate**

$$\hat{\theta} = \arg \max_{\theta} \mathcal{L}(\theta|\mathcal{X})$$

In words, find a parameter  $\theta$  such that, if we sample some data point follow this distribution, then it is most likely that we will get our learning set.

### 2.2.1 Example: Assume the data follow Bernoulli

**Bernoulli Distribution**  $x \in \{0, 1\}$ , take 1 with success probability  $p_0$ .

$$P(x|p_0) = p_0^x(1 - p_0)^{1-x}$$

**Log Likelihood**

$$\mathcal{L}(p_0|\mathcal{X}) = \sum_{l=1}^N [x^{(l)} \log p_0 + (1 - x^{(l)}) \log(1 - p_0)]$$

**ML estimation**

$$\hat{p}_0 = \frac{1}{N} \sum_{l=1}^N x^{(l)}$$

This is obtain by taking derivative of  $\mathcal{L}$  with respect to  $p_0$ , and set the derivative to zero.

### 2.2.2 Example: Multinomial Ran Var.

$\mathbf{x}$  with  $K \geq 2$  possible value

**Indicator variable** :

$$x_i = \begin{cases} 1 & \text{if outcome is state } i \\ 0 & \text{otherwise} \end{cases}$$

**Density function**

$$P(\mathbf{x}|\theta) = P(x_1, \dots, x_K | p_1, \dots, p_K) = \prod_{i=1}^K p_i^{x_i}, \quad \sum_{i=1}^K p_i = 1$$

**Log likelihood** :

$$\mathcal{L}(p_0|\mathcal{X}) = \sum_l \sum_{i=1}^K x_i^{(l)} \log p_i$$

**ML estimate** :

$$\hat{p}_i = \frac{1}{N} \sum_{l=1}^N x_i^{(l)}$$

### 2.2.3 Example Normal

**p. d. f.** :

$$p(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right]$$

**Log likelihood**

$$\mathcal{L}(\mu, \sigma|\mathcal{X}) = -\frac{N}{2} \log(2\pi) - N \log \sigma - \frac{1}{2\sigma^2} \sum_{l=1}^N (x^{(l)} - \mu)^2$$



**ML Estimates**

$$\hat{\mu} = \frac{1}{N} \sum_{l=1}^N x^{(l)}$$

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{l=1}^N (x^{(l)} - \hat{\mu})^2$$

**2.2.4 Multivariable Normal**

$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ ,  $\boldsymbol{\mu}$  mean vec,  $\boldsymbol{\Sigma}$  covariance matrix

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right]$$

$$\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathcal{X}) = \frac{Nd}{2} \log(2\pi) - \frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{l=1}^N (\mathbf{x}^{(l)} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}^{(l)} - \boldsymbol{\mu})$$

**ML estimates**

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{l=1}^N \mathbf{x}^{(l)}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{l=1}^N (\mathbf{x}^{(l)} - \hat{\boldsymbol{\mu}})(\mathbf{x}^{(l)} - \hat{\boldsymbol{\mu}})^T$$

**2.3 Bias and Variance**

Bias:

$$b_{\theta}(d) = E[d] - \theta$$

Variance:

$$E[d - E[d]]^2$$

(d is estimator of param  $\theta$ )

**Mean Squared error**

$$r(d, \theta) = E[(d - \theta)^2] = \text{bias}^2 + \text{variance}^2$$

**2.4 Bayesian Estimation**

Treat  $\theta$  as a Random Variable, that is, the parameter  $\theta$  follows a prior distribution  $p(\theta)$

**Posterior** :

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\theta)p(\theta)}{\int p(\mathcal{X}|\theta')p(\theta')d\theta'}$$

**Estimation of density at  $x$  :**

$$p(x|\mathcal{X}) = \int p(x|\theta)p(\theta|\mathcal{X})d\theta$$

Regression  $y = g(x|\theta)$ :

$$y = \int g(x|\theta)p(\theta|\mathcal{X})d\theta$$

## 2.5 Computational Considerations

**Max a posteriori (MAP)**

$$\theta_{MAP} = \arg \max_{\theta} p(\theta|\mathcal{X})$$

$$p(x|\mathcal{X}) \approx p(x|\theta_{MAP})$$

$$y \approx y_{MAP} = g(x|\theta_{MAP})$$

ML estimation:

$$\theta_{ML} = \arg \max_{\theta} p(\theta|\mathcal{X})$$

Bayes' estimation – expectation w.r.t. posterior density:

$$\theta_{Bayes} = E[\theta|\mathcal{X}] = \int \theta p(\theta|\mathcal{X})d\theta$$

Example: Bayesian estimation with known  $\mu$ ,  $\sigma$  and  $\sigma_0$

$$x^{(l)} \sim \mathcal{N}(\theta, \sigma_0^2), \theta \sim \mathcal{N}(\mu, \sigma^2)$$

$$\text{MLE: } \theta_{ML} = \frac{1}{N} \sum_{l=1}^N \mathbf{x}^{(l)} = m$$

$$\theta_{Map} = \theta_{Bayes} = E(\theta|\mathcal{X}) = \frac{N/\sigma_0^2}{N/\sigma_0^2 + 1/\sigma^2} m + \frac{1/\sigma^2}{N/\sigma_0^2 + 1/\sigma^2} \mu$$

**Classification with Discriminant Functions** Gaussian density for each class:  $p(x|C_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp \left[ -\frac{(x-\mu_i)^2}{2\sigma_i^2} \right]$

$$\text{Discriminant functions: } g_i(x) = \log [p(x|C_i)p(C_i)] = -\frac{1}{2} \log 2\pi - \log \sigma_i - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \log P(C_i)$$

$$\text{Sample } \mathcal{X} = \{(x^{(l)}, \mathbf{y}^{(l)})\}_{l=1}^N (y_i^{(l)} = 1 \text{ if } x^{(l)} \in C_i)$$

$$\text{ML Estimates: } \hat{P}(C_i) = \frac{1}{N} \sum_{l=1}^N y_i^{(l)}$$

$$m_i = \frac{\sum_{l=1}^N x^{(l)} y_i^{(l)}}{\sum_{l=1}^N y_i^{(l)}} \quad s_i^2 = \frac{\sum_{l=1}^N (x^{(l)} - m_i)^2 y_i^{(l)}}{\sum_{l=1}^N y_i^{(l)}}$$

Discriminant Functions:

$$g_i(x) = -\log s_i - \frac{(x - m_i)^2}{2s_i^2} + \log \hat{P}(C_i)$$

**Additive Parametric Model** Functional relationship in additive form:  $r = f(x) + \epsilon$

$$\text{Parametric modeling: } f(x) \approx g(x|\theta), \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Conditional probability of output given input:

$$p(r|x) \sim \mathcal{N}(g(x|\theta), \sigma^2)$$

Log likelihood given:  $\mathcal{X} = \{(x^{(l)}, r^{(l)})\}$ :

$$\mathcal{L}(\theta|\mathcal{X}) = \log \prod_{l=1}^N p(x^{(l)}, r^{(l)}) = \frac{1}{2\sigma^2} \sum_{l=1}^N \left[ r^{(l)} - g(x^{(l)}|\theta) \right]^2 + \text{const}$$

Equivalent to minimizing error function:

$$E(\theta|\mathcal{X}) = \frac{1}{2} \sum_{l=1}^N \left[ r^{(l)} - g(x^{(l)}|\theta) \right]^2$$

Called least squares estimates

### Polynomial Regression

$$g(x^{(l)}|w_0, w_1, \dots, w_k) = w_k(x^{(l)})^k + \dots + w_2(x^{(l)})^2 + w_1x^{(l)} + w_0$$

Least square estimate:  $\hat{\mathbf{w}} = (\mathbf{D}^T \mathbf{D})^{-1} \mathbf{D}^T \mathbf{r}$ , where:

$$D = \begin{bmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^k \\ 1 & x^{(2)} & (x^{(2)})^2 & \dots & (x^{(2)})^k \\ \dots & & & & \\ 1 & x^{(N)} & (x^{(N)})^2 & \dots & (x^{(N)})^k \end{bmatrix}$$

$$\mathbf{r} = \left( r^{(1)}, r^{(2)}, \dots, r^{(N)} \right)^T$$

**Bias and Variance** Expected squared error of sample  $\mathcal{X}$   $E[(r - g(x))^2|x] = (E[r|x] - g(x))^2 + E[(r - E[r|x])^2|x] =$   
squared err + noise

Average over  $\mathcal{X}$ :  $E_{\mathcal{X}}[(E[r|x] - g(x))^2|x] = (E[r|x] - E_{\mathcal{X}}[g(x)])^2 + E_{\mathcal{X}}[(g(x) - E_{\mathcal{X}}[g(x)])^2] =$   
bias + variance



### 3 Multivariate Method

**Multivariate Data** N i.i.d. instances:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_d^{(N)} \end{bmatrix}$$

**Parameters** Mean Vector:  $E[x] = \mu = (\mu_1, \dots, \mu_d)^T$

Covariance of  $x_i$  and  $x_j$ :

$$\sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)] = E[x_i x_j] - \mu_i \mu_j$$

Variance of  $x_i$ :  $\sigma_i^2 = E[(x_i - \mu_i)^2]$

Covariance matrix:

$$\begin{aligned} \Sigma &= Cov(\mathbf{x}) = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] \\ &= \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \dots & \sigma_{2d} \\ \dots & \dots & \dots & \dots \\ \sigma_{d1} & \sigma_{d2} & \dots & \sigma_d^2 \end{bmatrix} \end{aligned}$$

Correlation:  $\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j}$

$x_i$  and  $x_j$  are independent  $\Rightarrow \sigma_{ij} = \rho_{ij} = 0$

**Parameter Estimation** Sample Mean  $\mathbf{m} = \frac{1}{N} \sum_{l=1}^N \mathbf{x}^{(l)}$

Sam. Cov.:  $\mathbf{S} = [s_{ij}]_{i,j=1}^d = \frac{1}{N} \sum_{l=1}^N (\mathbf{x}^{(l)} - \mathbf{m})(\mathbf{x}^{(l)} - \mathbf{m})^T$

**Multivariate Normal Distribution**  $\mathbf{x} \sim \mathcal{N}_d(\boldsymbol{\mu}, \Sigma)$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

Mahalanobis distance:  $(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$  (d-dimensional hyperellipsoid.)

**Bivariate Normal Distribution** Covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho \sigma_1 \sigma_2 \\ \rho \sigma_1 \sigma_2 & \sigma_2^2 \end{bmatrix}$$

$$p(x_1, x_2) = \frac{1}{2\pi \sigma_1 \sigma_2 \sqrt{1-\rho^2}} \exp \left[ -\frac{1}{2(1-\rho^2)} (z_1^2 - 2\rho z_1 z_2 + z_2^2) \right], \text{ where } z_i = \frac{x_i - \mu_i}{\sigma_i}$$

**Parametric Classification** Class-conditional densities:  $p(\mathbf{x}|C_i) \sim \mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ :

$$p(\mathbf{x}|C_i) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_i|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \right]$$

$$\begin{aligned} \text{Discriminant functions: } g_i(\mathbf{x}) &= \log p(\mathbf{x}|C_i) + \log P(C_i) \\ &= -\frac{1}{2} \log 2\pi - \frac{1}{2} \log |\boldsymbol{\Sigma}_i| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \log P(C_i) \end{aligned}$$

**Estimation of Parameters**  $\hat{P}(C_i) = \frac{1}{N} \sum_l r_i^{(l)}$

$$\begin{aligned} \mathbf{m}_i &= \frac{\sum_l r_i^{(l)} \mathbf{x}^{(l)}}{\sum_l r_i^{(l)}} \\ \mathbf{S}_i &= \frac{\sum_l r_i^{(l)} r_i^{(l)} (\mathbf{x}^{(l)} - \mathbf{m}_i)(\mathbf{x}^{(l)} - \mathbf{m}_i)^T}{\sum_l r_i^{(l)}} \end{aligned}$$

**Quadratic Discriminant Functions**

$$g_i(\mathbf{x}) = \mathbf{x}^T \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + w_0$$

$$\begin{aligned} \text{where } \mathbf{W}_i &= -\frac{1}{2} \mathbf{S}_i^{-1} \\ \mathbf{w}_i &= \mathbf{S}_i^{-1} \mathbf{m}_i \\ w_{i0} &= -\frac{1}{2} \mathbf{m}_i^T \mathbf{S}_i^{-1} \mathbf{m}_i - \frac{1}{2} \log |\mathbf{S}_i| + \log \hat{P}(C_i) \end{aligned}$$

## 4 Dimensionality reduction

**Forward Search** Start with no features, add them one by one, at each step adding the one that decreases most.

**Backward Search** Start with all features and so a similar process

**Principle Component Analysis** Projection of  $\mathbf{x}$  on the direction of  $\mathbf{w}$ :  $z = \mathbf{w}^T \mathbf{x}$

Finding the first principle component  $\mathbf{w}_1$  such that  $Var(z_1)$  is maximized:

$$\begin{aligned} Var(z_1) &= Var(\mathbf{w}^T \mathbf{x}) = E[(\mathbf{w}^T \mathbf{x} - \mathbf{w}^T \boldsymbol{\mu})^2] \\ &= \mathbf{w}^T E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] \mathbf{w} = \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w} \end{aligned}$$

$$Cov(\mathbf{x}) = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] = \boldsymbol{\Sigma}$$

$$\text{Lagrangian: } \mathbf{w}_1^T \boldsymbol{\Sigma} \mathbf{w}_1 - \alpha(\mathbf{w}_1^T \mathbf{w}_1 - 1)$$

Taking derivative:  $\boldsymbol{\Sigma} \mathbf{w}_1 = \alpha \mathbf{w}_1$  (eigenvalue equation)  $\mathbf{w}_1^T \boldsymbol{\Sigma} \mathbf{w}_1 = \alpha \mathbf{w}_1^T \mathbf{w}_1 = \alpha$  ( $\mathbf{w}_1$  is unit)

We choose the eigenvector with the largest eigenvalue for the variance to be maximum.

$$\text{Second PC: } \mathbf{w}_2^T \boldsymbol{\Sigma} \mathbf{w}_2 - \alpha(\mathbf{w}_2^T \mathbf{w}_2 - 1) - \beta(\mathbf{w}_2^T \mathbf{w}_1 - 0)$$

$$\text{Derivative: } 2\boldsymbol{\Sigma} \mathbf{w}_2 - 2\alpha \mathbf{w}_2 - \beta \mathbf{w}_1 = 0 \text{ (times } \mathbf{w}_1^T \text{ on both side, } \mathbf{w}_1^T \mathbf{w}_2 = 0)$$

Then  $\boldsymbol{\Sigma} \mathbf{w}_2 = \alpha \mathbf{w}_2$ ,  $\mathbf{w}_2$  is the second largest eigenvalue.

Proportion of variance (PoV) Explained:  $\frac{\lambda_1, \dots, \lambda_k}{\lambda_1, \dots, \lambda_d}$

**Factor Analysis** Assume latent factors  $z_j$  Sample  $\mathcal{X} = \{\mathbf{x}^{(l)}\}$ ,  $E(\mathbf{x}) = \boldsymbol{\mu}$ ,  $Cov(\mathbf{x}) = \boldsymbol{\Sigma}$

Factors  $z_j$ ,  $E[z_j] = 0$ ,  $Var(z_j) = 1$

Noise:  $\epsilon_i$ :  $E[\epsilon_i] = 0$ ,  $Var(\epsilon_i) = \Psi_i$ ,  $Cov(\epsilon_i, \epsilon_j) = 0$

$x_i - \mu_i = \sum_{j=1}^k v_{ij} z_j + \epsilon_i$ , assume  $\boldsymbol{\mu} = 0$ ,  $v_{ij}$  are called factor loading

$$Var(x_i) = \sum_{j=1}^k v_{ij}^2 Var(z_j) + Var(\epsilon_i) = \sum_{j=1}^k v_{ij}^2 + \Psi_i$$

$$\text{Covariance Matrix: } \boldsymbol{\Sigma} = Cov(\mathbf{V}\mathbf{z} + \boldsymbol{\epsilon}) = \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi}$$

$$\text{Factor loading: } Cov(\mathbf{x}, \mathbf{z}) = \mathbf{V}$$

Dim reduc: Given  $\mathbf{S}$  as the estimator of  $\boldsymbol{\Sigma}$ , we want to find  $\mathbf{V}$  and  $\boldsymbol{\Psi}$  s.t.  $\mathbf{S} = \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi}$ ,  $\boldsymbol{\Psi} = diag(\Psi_i)$

**Multidimensional Scaling** lower dimension preserve pairwise distances.

Sample  $\mathcal{X} = \{\mathbf{x}^{(l)} \in \mathbb{R}^d\}_{l=1}^N$

Squared Euclidean distance between point  $r$  and  $s$ :

$$d_{rs}^2 = \sum_{j=1}^d (x_j^{(r)} - x_j^{(s)})^2 = b_{rr} + b_{ss} - 2b_{rs}$$

$$b_{rs} = \sum_{j=1}^d x_j^{(r)} x_j^{(s)}, \text{ or matrix form } \mathbf{B} = \mathbf{X}\mathbf{X}^T$$

$$\text{Constraint: } \sum_l x_j^{(l)} = 0, \forall j, \text{ define: } T = \sum_{l=1}^N b_{ll}$$

$$\text{Then } \sum_r d_{rs}^2 = T + Nb_{ss}, \sum_r \sum_s = d_{rs}^2 = 2NT$$

defining:

$$d_{*s}^2 = \frac{1}{N} \sum_r d_{rs}^2, d_{r*}^2 = \frac{1}{N} \sum_s d_{rs}^2, d_{**}^2 = \frac{1}{N^2} \sum_r \sum_s d_{rs}^2$$

So  $b_{rs} = \frac{1}{2}(d_{r*}^2 + d_{*s}^2 - d_{**}^2 - d_{rs}^2)$   $\mathbf{B} = \mathbf{X}\mathbf{X}^T$  is p.s.d. :

$$\mathbf{B} = \mathbf{C}\mathbf{D}\mathbf{C}^T = (\mathbf{C}\mathbf{D}^{1/2})(\mathbf{C}\mathbf{D}^{1/2})^T$$

Ignore small eigenvalues, let  $\mathbf{c}_j$  be the  $k$  eigenvectors chosen with eigenvalues  $\lambda_j$ , the new dimensions:  $z_j^{(l)} = \sqrt{\lambda_j} c_j^{(l)}$

**LDA** Sample mean after projection:

$$m_1 = \mathbf{w}^T \mathbf{m}_1, m_2 = \mathbf{w}^T \mathbf{m}_2$$

Between class scatter:  $(m_1 - m_2) = \mathbf{w}^T \mathbf{S}_B \mathbf{w}$ ,  $\mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$

Within Class Scatter:  $s_1^2 = \mathbf{w}^T \mathbf{S}_1 \mathbf{w}$ ,  $\mathbf{S}_1 = \sum_l (\mathbf{x}^{(l)} - \mathbf{m}_1)(\mathbf{x}^{(l)} - \mathbf{m}_1)^T y^{(l)}$ , similarly  $\mathbf{S}_2 = \sum_l (\mathbf{x}^{(l)} - \mathbf{m}_2)(\mathbf{x}^{(l)} - \mathbf{m}_2)^T (1 - y^{(l)})$ , so

$$s_1^2 + s_2^2 = \mathbf{w}^T \mathbf{S}_w \mathbf{w}$$

$$, \mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2$$

**Fisher's LD**  $J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}$ , take derivative of  $J$  w.r.t.  $\mathbf{w}$  setting it to 0:  $\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}$ ,  
or  $\mathbf{S}_w^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$  (Eigen equation)

$K > 2$  Within-class scatter  $\mathbf{S}_i = \sum_l y_i^{(l)} (\mathbf{x}^{(l)} - \mathbf{m}_i)(\mathbf{x}^{(l)} - \mathbf{m}_i)^T$ ,  $y_i^{(l)} = 1$  if  $\mathbf{x}^{(l)} \in C_i$

Total class scatter:  $\mathbf{S}_W = \sum_{i=1}^K \mathbf{S}_i$

Between Class Scatter:  $\mathbf{S}_B = \sum_{i=1}^K N_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T$

Optimal is  $\mathbf{W}$  that max:  $J(\mathbf{W}) = \frac{\text{Tr}(\mathbf{W}^T \mathbf{S}_B \mathbf{W})}{\text{Tr}(\mathbf{W}^T \mathbf{S}_w \mathbf{W})}$  Corresponds to eigenvectors of  $\mathbf{S}_w^{-1} \mathbf{S}_B$



## 5 Non Parametric Method

**Nonparametric Density Estimation** Sample:  $\mathcal{X} = \{x^{(l)}\}_{l=1}^N$ , Probability density:  $p(X)$ , cumulative distribution:  $F(X)$

Estimator of  $F(X)$ :  $\hat{F}(x) = \frac{\#\{x^{(l)} \leq x\}}{N}$

Estimator of  $p(X)$ :  $\hat{p}(x) = \frac{1}{h} \left[ \frac{\#\{x^{(l)} + h \leq x\}}{N} - \frac{\#\{x^{(l)} \leq x\}}{N} \right]$ , where  $h$  is the interval and instances  $x^{(l)}$  that fall in this interval are assumed to be close enough

**Histogram Estimator** bin:  $[x_0 + mh, x_0 + (m+1)h]$ ,  $x_0$  origin,  $h$  bin width

$$\hat{p}(x) = \frac{\#\{x^{(l)} \text{ in the same bin as } x\}}{Nh}$$

$$\text{Naive Estimator: } \hat{p}(x) = \frac{\#\{x-h/2 < x^{(l)} \leq x+h/2\}}{Nh}$$

Alternative form:  $\hat{p}(x) = \frac{1}{Nh} \sum_{l=1}^N w\left(\frac{x-x^{(l)}}{h}\right)$  with weight function

$$w = \begin{cases} 1 & \text{if } |u| < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

**Kernel Estimator** Kernel  $K(u) = \frac{1}{2\pi} \exp(-\frac{u^2}{2})$ ,

$$\text{Estimator: } \hat{p} = \frac{1}{Nh} \sum_{l=1}^N K\left(\frac{x-x^{(l)}}{h}\right)$$

**KNN**  $\hat{p}(x) = \frac{k}{2Nd_k(x)}$ ,  $d_k(x)$  is the distance from  $x$  to the  $k$ th nearest instance.

KNN with kernel:  $\hat{p}(\mathbf{x}) = \frac{1}{Nh^d} \sum_{l=1}^N K\left(\frac{\mathbf{x}-\mathbf{x}^{(l)}}{h}\right)$ , with  $\int_{\mathbb{R}^d} K(\mathbf{x}) d\mathbf{x} = 1$ ,

Multivariate ellipsoidal Gaussian kernel:

$$K(\mathbf{u}) = \frac{1}{(2\pi)^{d/2} |\mathbf{S}|^{1/2}} \exp\left(-\frac{1}{2} \mathbf{u}^T \mathbf{S}^T \mathbf{u}\right)$$

**Nonparametric Classification** Kernel estimator of class-conditional densities:  $\hat{p}(\mathbf{x}|C_i) = \frac{1}{N_i h^d} \sum_{l=1}^N K\left(\frac{\mathbf{x}-\mathbf{x}^{(l)}}{h}\right) y_i^{(l)}$ ,  $y_i^{(l)} = 1$  if  $\mathbf{x}^{(l)}$  is in  $C_i$ , and  $N = \sum_l y_i^{(l)}$ ,  $\hat{P}(C_i) = \frac{N_i}{N}$

Discriminant:

$$g_i(\mathbf{x}) = \hat{p}(\mathbf{x}|C_i) \hat{P}(C_i) = \frac{1}{Nh^d} \sum_{l=1}^N K\left(\frac{\mathbf{x}-\mathbf{x}^{(l)}}{h}\right) y_i^{(l)}$$

**KNN classifier**  $\hat{p}(\mathbf{x}|C_i) = \frac{k_i}{N_i V^k(\mathbf{x})}$ ,  $\hat{P}(C_i|\mathbf{x}) = \frac{k_i}{k}$ , Choose  $C_i$  if  $i = \arg \max_j \hat{P}(C_j|\mathbf{x}) = \arg \max_j k_j$

**Non param regression**  $y^{(l)} = g(\mathbf{x}^{(l)}) + \epsilon$ ,

$$\hat{g}(x) = \frac{\sum_{l=1}^N K\left(\frac{x-x^{(l)}}{h}\right) y^{(l)}}{\sum_{l=1}^N K\left(\frac{x-x^{(l)}}{h}\right)}$$

**Regularized cost function** balance bias and variance:

$$\sum_l [y^{(l)} - \hat{g}(x^{(l)})]^2 + \lambda \int_a^b [\hat{g}''(x)]^2 dx$$



# 6 Decision Trees

## 6.1 Constructing Decision Trees

### 6.1.1 Basic algorithm

Can be expressed recursively.

1. select an attribute to place at the root node
  - Make one branch for each possible value.
  - This splits up the example set into subsets, one for each possible value.
2. Repeat the process recursively for each branch
3. If at any tie all instances at a node have the same classification, stop.

Only thing: how to determine which attribute to split on.

### 6.1.2 Measure of purity

If we had a measure of the purity of each node, we could choose the attribute that produces the purest daughter nodes.

We use *information* measured with unit of *bits*.

- It represents the expected amount of information that would be needed to specify whether a new instance should be classified yes or no
- Given the example reached that node.
- The value is often less than 1

We calculate the information gained on different splits and choose the one that gain most.

### Calculating Information

Properties we expect to have:

- When the number of either *yes*'s or *no*'s is zero, the information is zero
- When the number of *yes*'s and *no*'s equal, the information reaches a maximum
- The information should obey the multistage property that we have illustrated.

## Highly Branching Attributes

The information gain measure tends to prefer attributes with large numbers of possible values.

To compensate for this, a modification of the measure called the gain ratio is widely used. The gain ratio is derived by taking into account the number and size of daughter nodes into which an attribute splits the dataset, disregarding any information about the class.

## 6.2 Pruning

Fully expanded decision trees often contain unnecessary structure, and it is generally advisable to simplify them before they are deployed.

- Prepruning would involve trying to decide during the tree building process when to stop developing subtrees. – avoid all the work of developing subtrees only to throw them away afterward.
- Postpruning: situations occur in which attributes individually seem to have nothing to contribute but are powerful predictors when combined.

Two operations that have been considered for postpruning: *subtree replacement* and *subtree raising*.

At each node, a learning scheme might decide whether it should perform subtree replacement, subtree raising, or leave the subtree as it is, unpruned.

**Subtree replacement** The idea is to select some subtrees and replace them with single leaves. This will certainly cause the accuracy on the training set to decrease, however, it may **increase the accuracy on an independently chosen test set**.

When subtree replacement is implemented, it proceeds **from the leaves and works back up toward the root**.

**Subtree raising** Replace an internal node by one of the nodes below it.

### 6.2.1 Estimating Error Rates

How to decide whether to replace an internal node by a leaf.

Estimate the error rate that would be expected at a particular node given an independently chosen test set, at internal nodes and at leaf nodes.

If we had such an estimate, it would be clear whether to replace, or raise. A particular subtree simply by comparing the estimated error of the subtree with that of its proposed replacement.

Standard verification technique: Hold back some of the data originally given and use it as an independent test set to estimate the error at each node. Drawback: the actual tree is based on less data.

Alternative: try to make some estimate of error based on the training data itself. It is a heuristic based on some statistical reasoning.

The idea is to consider the set of instances that reach each node and imagine that the majority class is chosen to represent the node. That gives us a certain number of errors,  $E$ , out of the total number of instances,  $N$ .

Now imagine the true probability of error at the node is  $q$  and that  $N$  instances are generated by a Bernoulli process with parameter  $q$ , of which  $E$  turn out to be errors.

Given a particular confidence  $c$ , we find confidence limits  $z$  such that:

$$Pr \left[ \frac{f - q}{\sqrt{q(1-q)/N}} > z \right] = c$$

Where  $N$  is the number of samples,  $f = E/N$  is the observed error rate, and  $q$  is the true error rate.

Now we use that upper confidence limit as a estimate for the error  $e$  at the node:

$$e = \frac{f + \frac{z^2}{2N} + z \sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}$$

**Classification Trees** for node  $m$ ,  $N_m$  training instances,  $N_m^i$  instances belong to class  $C_i$ , estimate for the probability of class  $C_i$   $\hat{P}(C_i|\mathbf{x}, m) = \frac{N_m^i}{N_m} = p_m^i$ , pure if  $p_m^j = 1$

**Entropy**  $\mathcal{T}_m = - \sum_{i=1}^K p_m^i \log_2 p_m^i$ , assume  $0 \log 0 = 0$ , largest is  $\log_2 K$  when all  $p_m^i = 1/K$

**Other Impurity Measures** Properties:  $\phi(1/2, 1/2) \geq \phi(p, 1-p)$ ,  $\phi(0, 1) = \phi(1, 0) = 0$ ,  $\phi(p, 1-p)$  increase in  $p$  on  $[0, 1/2]$  and decrease on  $[1/2, 1]$

**Best Split** Node  $m$ ,  $N_{mj}$  take branch  $j$ , if  $f_m(\mathbf{x}) = j$ , the estimate for the probability of class  $C_i$  is:

$$\hat{P}(C_i|\mathbf{x}, m, j) = p_{mj}^i = \frac{N_{mj}^i}{N_{mj}}, \text{total impurity after split:}$$

$$\mathcal{T}_m' = - \sum_{i=1}^K \frac{N_{mj}^i}{N_m} \sum_{i=1}^K p_{mj}^i \log p_{mj}^i$$

**Regression Trees**  $b_m(\mathbf{x}) = 1$  if in  $\mathcal{X}_m$ ,  
estimated value at node  $m$ :

$$g_m = \frac{\sum_l b_m(\mathbf{x}^{(l)}) y^{(l)}}{\sum_l b_m(\mathbf{x}^{(l)})}$$

,

mean square error after split:

$$E_m = \frac{1}{N_m} \sum_l (y^{(l)} - g_m)^2 b_m(\mathbf{x}^{(l)})$$

tree expansion:

$$b_{mj}(\mathbf{x}) = 1 \text{ if } \mathbf{x} \in \mathcal{X}_{mj}$$

,

estimate val in branch  $j$ :  $g_{mj} = \frac{\sum_l b_{mj}(\mathbf{x}^{(l)}) y^{(l)}}{\sum_l b_{mj}(\mathbf{x}^{(l)})}$ ,

error after split  $E_m' = \frac{1}{N_m} \sum_j \sum_l (y^{(l)} - g_{mj}^2 b_{mj}(\mathbf{x}^{(l)}))$

Best split: split that results in smallest error, or worst possible error.

**Pruning** Prepruning stop split when the number of instances reaching a node is belong a certain percentage. Post pruning: Replace subtree by a leaf node, if the leaf node does not perform worse, the subtree is pruned and replaced by leaf.



# 7 Linear Model

## 7.1 Linear Discriminant functions

A discriminant is a function that take an input vector  $\mathbf{x}$  and assigns it to one of  $K$  classes:

$$g_i(\mathbf{x}|\mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

generalized use base function  $g_i(\mathbf{x}) = \sum_{j=1}^k w_j \phi_{ij}(\mathbf{x})$

### 7.1.1 Two classes

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$$

,  $C_1$  if  $g(\mathbf{x}) > 0$

### 7.1.2 Geometry Interpretation

The decision boundary is defined by

$$g(\mathbf{x}) = 0$$

Corresponds to a  $(D-1)$  dimensional hyperplane within the  $D$ -dimensional space.

Express any point as

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

where  $x_p$  is the projection of  $\mathbf{x}$  onto hyperplane.  $r$  distance from  $\mathbf{x}$  to hyper plane. plane, we have  $r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$

### 7.1.3 Multi classes

K discriminant:

$$g_i(\mathbf{x}|\mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

Linear separable:

$$g_i(\mathbf{x}|\mathbf{w}_i, \mathbf{w}_{i0}) > 0$$

if  $\mathbf{x} \in C_i$

Choose  $C_i$  if

$$g_i(\mathbf{x}) = \max_{j=1}^K g_j(\mathbf{x})$$

### 7.1.4 Pairwise Separation

Discriminant function for class  $i$  and  $j$ :

$$g_{ij}(\mathbf{x}|\mathbf{w}_{ij}, w_{ij0}) = \mathbf{w}_{ij}^T \mathbf{x} + w_{ij0} = \begin{cases} > 0 & \text{if } \mathbf{x} \in C_i \\ \leq 0 & \text{if } \mathbf{x} \in C_j \\ \text{don't care} & \text{if } \mathbf{x} \in C_k, k \neq i, k \neq j \end{cases}$$

## 7.2 Logistic Discrimination

### 7.2.1 Two classes

Assume that the log likelihood ratio is linear:

$$\log \frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} = \mathbf{w}^T \mathbf{x} + w_0^o$$

Using Baye's rule we have;

$$\begin{aligned} \text{logit}(P(C_1|x)) &= \log \frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})} \\ &= \log \frac{p(\mathbf{x}|C_1)}{p(\mathbf{x}|C_2)} + \log \frac{p(C_1)}{p(C_2)} \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

where  $w_0 = w_0^o + \log \frac{P(C_1)}{P(C_2)}$

Rearranging terms:

$$\begin{aligned} y &= \text{sigmoid}(\mathbf{w}^T \mathbf{x} + w_0) \\ &= \hat{P}(C_1|\mathbf{x}) = \frac{1}{1 + \exp[-(\mathbf{w}^T \mathbf{x} + w_0)]} \end{aligned}$$

As our estimator of  $P(C_1|\mathbf{x})$

### Gradient Decent

In the discriminant-based approach, the parameters are those of the discriminants, and they are *optimized to minimize the classification error*

**Error**  $\mathbf{w}$  denotes the set of parameters and  $E(\mathbf{w}|\mathcal{X})$  is the error parameters  $\mathbf{w}$  on the given training set  $\mathcal{X}$ , we look for:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w}|\mathcal{X})$$

No analytical solution

**Gradient Vector** When  $E(\mathbf{w})$  is a differentiable function of a vector of variables, we have the gradient vector composed of the partial derivatives:

$$\nabla_{\mathbf{w}} E = \left[ \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_d} \right]^T$$



**Gradient Descent** starts from a *random*  $\mathbf{w}$ , at each step, update  $w$  in a *opposite direction* of the gradient:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i$$

$$w_i = w_i + \Delta w_i$$

$\eta$  is *step size*, or *learning factor*

When we get to minimum, the derivative is 0 and the procedure terminates.

This indicates that the procedure finds the nearest minimum that can be *local minimum*. There is no guarantee of finds the nearest minimum that can be a local minimum

**Learning parameters** Given a sample of two classes,  $\mathcal{X} = \mathbf{x}^{(l)}, \mathbf{r}^{(l)}$ , where  $\mathbf{r}^{(l)} = 1$  if  $\mathbf{x} \in C_1$

We assume  $\mathbf{r}^{(l)}$ , given  $\mathbf{x}^{(l)}$  is Bernoulli with probability  $y^{(l)} = p(C_1|\mathbf{x}^{(l)})$  :

$$\mathbf{r}^{(l)}|\mathbf{x}^{(l)} \sim \text{Bernoulli}(y^{(l)})$$

Note that in this discriminant-based approach, we model directly  $\mathbf{r}|\mathbf{x}$  The sample likelihood is:

$$L(\text{mathbfbf}w, w_0|\mathcal{X}) = \prod_t (y^{(l)})^{r^{(l)}} (1 - y^{(l)})^{1-r^{(l)}}$$

We can always turn it in an error function to minimize:  $E = -\log L$  So we have *cross-entropy*:

$$E(\text{mathbfbf}w, w_0|\mathcal{X}) = -\sum_t \mathbf{r}^{(l)} \log y^{(l)} + (1 - r^{(l)}) \log(1 - y^{(l)})$$

We use gradient descent to minimize cross-entropy If  $y = \text{sigmoid}(a) = \frac{1}{1+\exp(-a)}$ , its derivative is given as:

$$\frac{dy}{da} = y(1 - y)$$

and we get the following update equations:

$$\begin{aligned} \Delta w_j &= -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t \left( \frac{r^{(l)}}{y^{(l)}} - \frac{1 - r^{(l)}}{1 - y^{(l)}} x_j^{(l)} \right) \\ &= \eta \sum_t (r^{(l)} - y^{(l)}) x_j^{(l)}, \quad j = 1, \dots, d \\ \Delta w_0 &= -\eta \frac{\partial E}{\partial w_0} = \eta \sum_t (r^{(l)} - y^{(l)}) \end{aligned}$$

## 7.2.2 Multiple Classes

### Generalization of sigmoid

Take one of the classes  $C_k$ , as reference class and assume that:

$$\log \frac{p(\mathbf{x}|C_i)}{p(\mathbf{x}|C_K)} = \text{mathbfbf}w_i^T \mathbf{x} + w_{i0}^o, \quad i = 1, \dots, K - 1$$

Then we have:

$$\frac{P(C_i|\mathbf{x})}{P(C_K|\mathbf{x})} = \exp[\text{mathbfbf}w_i^T \mathbf{x} + w_{i0}]$$

With  $w_{i0} = w_{i0}^o + \log \frac{P(C_i)}{P(C_K)}$

Summing over  $i$  we can deduce:

$$P(C_K|\mathbf{x}) = \frac{1}{1 + \sum_{i=1}^{K-1} \exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})}$$

$$P(C_i|\mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \mathbf{x} + w_{j0})}$$

, where  $k = 1, \dots, K - 1$

### Softmax

Treat all classes uniformly

$$y_i = \hat{P}(C_i|\mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \mathbf{x} + w_{i0})}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x} + w_{j0})}, \quad i = 1, \dots, K$$

### Learning

$$\frac{\partial y_i}{\partial a_j} = y_i(\delta_{ij} - y_i)$$

$$\Delta \mathbf{w}_j = \eta \sum_l (r_j^{(l)} - y_j^{(l)}) \mathbf{x}^{(l)}, \quad \Delta w_{j0} = \eta \sum_l (r_j^{(l)} - y_j^{(l)})$$

**Regression for two-class Classification**  $r^{(l)} = y^{(l)} + \epsilon$ ,  $r^{(l)} \in \{0, 1\}$ ,  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ ,  $y^{(l)} = \text{sigmoid}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0)$ ,

$$\text{Likelihood: } L(\mathbf{w}, w_0|\mathcal{X}) = \prod_l \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[ -\frac{(r^{(l)} - y^{(l)})^2}{2\sigma^2} \right],$$

$$\text{Error: } E(\{\mathbf{w}_i, w_{i0}\}_i|\mathcal{X}) = \frac{1}{2} \sum_l (r^{(l)} - y^{(l)})^2,$$

$$\text{Learning } \Delta \mathbf{w} = \eta \sum_l (r^{(l)} - y^{(l)}) y^{(l)} (1 - y^{(l)}) \mathbf{x}^{(l)},$$

$$\Delta w_0 = \eta \sum_l (r^{(l)} - y^{(l)}) y^{(l)} (1 - y^{(l)})$$

$$K > 2 \text{ classes } \mathbf{r}^{(l)} = \mathbf{y}^{(l)} + \boldsymbol{\epsilon},$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}_K(0, \sigma^2 \mathbf{I}_K), \quad y_i^{(l)} = \frac{1}{1 + \exp[-(\mathbf{w}_i^T \mathbf{x}^{(l)} + w_{i0})]},$$

Likelihood:

$$L(\{\mathbf{w}_i, w_{i0}\}_i|\mathcal{X}) = \prod_l \frac{1}{(2\pi)^{K/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[ -\frac{\|\mathbf{r}^{(l)} - \mathbf{y}^{(l)}\|^2}{2\sigma^2} \right]$$

$$\text{Error Func: } E(\{\mathbf{w}_i, w_{i0}\}_i|\mathcal{X}) = \frac{1}{2} \sum_l \|\mathbf{r}^{(l)} - \mathbf{y}^{(l)}\|^2$$

$$\text{Learning } \Delta \mathbf{w}_i = \eta \sum_l (r_i^{(l)} - y_i^{(l)}) y_i^{(l)} (1 - y_i^{(l)}) \mathbf{x}^{(l)},$$

$$\Delta w_0 = \eta \sum_l (r_i^{(l)} - y_i^{(l)}) y_i^{(l)} (1 - y_i^{(l)})$$

## 7.3 Kernel Methods

Linear parametric models: after training, the training data is then discarded, and predictions from new inputs are based purely on the learned parameters. This methods also used in nonlinear such as neural networks.

There is a class of techniques, in which the training data points or a subset of them, are kept and used also during the prediction phase.

The kernel is a symmetric function of its arguments so that

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$$

The general ideal is that, if we have an algorithm formulated in such a way that the input vector  $\mathbf{x}$  enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel.

Key ideal of kernel methods:

- Instead of defining a nonlinear model in the original input space, the problem is mapped to a new feature space.
- The new space is produced by performing a nonlinear transformation using suitably chosen **basis functions**
- A linear model is then applied to a new nonlinear-transformed space
- The basis functions are often defined implicitly via defining **kernel functions directly**



# 8 Multilayer Perceptrons

## 8.1 Perceptron

The output  $y$  is a **weighted sum** of the input  $\mathbf{x} = (x_0, x_1, \dots, x_d)^T$ :

$$y = \sum_{j=1}^d w_j x_j + w_0 = \mathbf{w}^T \mathbf{x}$$

where  $x_0$  is a special *bias unit* with  $x_0 = 1$  and  $\mathbf{w} = (x_0, w_1, \dots, w_d)^T$  are called **connection weight or synaptic weights**

To implement a linear discriminant function, we need threshold function:

$$s(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

to define the following decision rule:

$$\text{Choose } \begin{cases} C_1 & \text{if } s(\mathbf{w}^T \mathbf{x}) = 1 \\ C - 2 & \text{otherwise} \end{cases}$$

Use **sigmoid** instead of threshold to gain differentiability:

$$y = \text{sigmoid}(\mathbf{w}^T \mathbf{x}), \text{ where } \text{sigmoid}(a) = \frac{1}{1 + \exp(-a)}$$

The output may be interpreted as the *posterior probability* that the input  $\mathbf{x}$  belongs to  $C_1$

### 8.1.1 $K > 2$ Outputs

$K$  perceptrons, each with a weight vector  $\mathbf{w}_i$ ,

$$y_i = \sum_{j=1}^d w_{ij} x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x}, \text{ or } \mathbf{y} = \mathbf{W} \mathbf{x}$$

where  $w_{ij}$  is the weight from input  $x_j$  to output  $y_i$  and each row of the  $K \times (d + 1)$  matrix  $\mathbf{W}$  is the weight vector of one perceptron.

Choose  $C_i$  if  $y_i = \max_k y_k$

**Posterior probability** Use softmax to define  $y_i$  as:

$$y_i = \frac{\exp(\mathbf{w}_i^T \mathbf{x})}{\sum_{k=1}^K \exp(\mathbf{w}_k^T \mathbf{x})}$$

### 8.1.2 Stochastic Gradient Descent

Gradient descent for online learning, for regression, the error on a single instance

$$E^{(l)}(\mathbf{w}|\mathbf{x}^{(l)}, r^{(l)}) = \frac{1}{2}(r^{(l)} - y^{(l)})^2 = \frac{1}{2} \left[ r^{(l)} - (\mathbf{w}^T \mathbf{x}^{(l)}) \right]^2$$

gives the online update rule:

$$\Delta w_j^{(l)} = \eta(r^{(l)} - y^{(l)})x_j^{(l)}$$

where  $\eta$  is step size.

For Binary classification:

Likelihood:

$$L = (y^{(l)})^{r^{(l)}} (1 - y^{(l)})^{1-r^{(l)}}$$

Cross Entropy:

$$E^{(l)}(\mathbf{w}|\mathbf{x}^{(l)}, r^{(l)}) = -\log L = -r^{(l)} \log y^{(l)} - (1 - r^{(l)}) \log(1 - y^{(l)})$$

Online update rule:

$$\Delta w_j^{(l)} = \eta(r^{(l)} - y^{(l)})x_j^{(l)}$$

$K > 2$

$$y_i^{(l)} = \frac{\exp(\mathbf{w}_i^T \mathbf{x}^{(l)})}{\sum_k \exp(\mathbf{w}_k^T \mathbf{x}^{(l)})}$$

Likelihood:

$$L = \prod_i (y_i^{(l)})^{r_i^{(l)}}$$

Cross Entropy:

$$E^{(l)}(\{\mathbf{w}_i\}|\mathbf{x}^{(l)}, \mathbf{r}^{(l)}) = -\sum_i r_i^{(l)} \log y_i^{(l)}$$

Online update rule:

$$\Delta w_{ij}^{(l)} = \eta(r_i^{(l)} - y_i^{(l)})x_j^{(l)}$$

## 8.2 Multilayer Perception

MLP has a hidden layer between the input and output.

Input to hidden:

$$z_h = \text{sigmoid}(\mathbf{w}_h^T \mathbf{x}) = \frac{1}{1 + \exp \left[ -(\sum_{j=1}^d w_{hj} x_j + w_{h0}) \right]}$$

Hidden to output:

$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^H v_{ih} z_h + v_{i0}$$

Backward, hidden-to-output weight: treating hidden unit as input

Input-to-hidden, chain rule:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$

### 8.2.1 MLP for Nonlinear Regression(Multi output)

outputs:

$$\begin{aligned} y_i^{(l)} &= \sum_{h=1}^H v_{ih} z_h^{(l)} + v_{i0} \\ z_h^{(l)} &= \text{sigmoid}(\mathbf{w}_h^T \mathbf{x}^{(l)}) \end{aligned}$$

Error function:

$$E(\mathbf{W}, \mathbf{v} | \mathcal{X}) = \frac{1}{2} \sum_l \sum_i (r_i^{(l)} - y_i^{(l)})^2$$

Update rule for second layer:

$$\Delta v_{ih} = \eta \sum_l (r_i^{(l)} - y_i^{(l)}) z_h^{(l)}$$

Update rule for first layer:

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = \eta \sum_l \left[ \sum_i (r_i^{(l)} - y_i^{(l)}) v_{ih} \right] z_h^{(l)} (1 - z_h^{(l)}) x_j^{(l)}$$

### 8.2.2 MLP for NonLinear Multi-class Discrimination

Outputs:

$$y_i^{(l)} = \frac{\exp(o_i^{(l)})}{\sum_k \exp(o_k^{(l)})}$$

,

$$o_i^{(l)} = \sum_h v_{ih} z_h^{(l)} + v_{i0}$$

Error Function:

$$E(\mathbf{W}, \mathbf{V} | \mathcal{X}) = - \sum_l \sum_i r_i^{(l)} \log y_i^{(l)}$$

, update rules are the same us regression.





# 9 Support Vector Machine

## 9.1 Preliminary

**Margin of a separating hyperplane** distance from hyperplane to the nearest data point. (largest margin generalizes best)

**Hard-margin case** points are **linearly separable** with proper scaling of  $\mathbf{w}$  and  $w_0$ , the points closest to the hyper plane satisfy  $|\mathbf{w}^T \mathbf{x} + w_0| = 1 \Rightarrow$  **canonical separating hyperplane**.

The one max the margin: **canonical optimal separating hyperplane**

$\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$  be two closest point on each side:

$$\mathbf{w}^T \mathbf{x}^{(1)} + w_0 = +1$$

$$\mathbf{w}^T \mathbf{x}^{(2)} + w_0 = -1$$

So  $\mathbf{w}^T(\mathbf{x}^{(1)} - \mathbf{x}^{(2)}) = 2$ , the margin is given by:

$$\gamma = \frac{1}{2} \frac{\mathbf{w}^T(\mathbf{x}^{(1)} - \mathbf{x}^{(2)})}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

Maximizing the margin is equivalent to minimizing  $\|\mathbf{w}\|$

**Inequality constraint**

$$\mathbf{w}^T \mathbf{x}^{(l)} + w_0 \begin{cases} \geq +1 & \text{if } y^{(l)} = +1 \\ \leq -1 & \text{if } y^{(l)} = -1 \end{cases}$$

Equivalent to:

$$y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) \geq 1$$

**Primal Optimization**

$$\begin{aligned} &\text{Minimize} && \frac{1}{2} \|\mathbf{w}\|^2 \\ &\text{Subject to} && y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) \geq 1, \forall l \end{aligned}$$

**Lagrangian**

$$\begin{aligned} L_p(\mathbf{w}, w, \alpha_l) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{l=1}^N \alpha_l \left[ y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) - 1 \right] \\ &= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \mathbf{w}^T \sum_l \alpha_l y^{(l)} \mathbf{x}^{(l)} - w_0 \sum_l \alpha_l y^{(l)} + \sum_l \alpha_l \end{aligned}$$

**Eliminating Primal Var**

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_l \alpha_l y^{(l)} \mathbf{x}^{(l)}$$

$$\frac{\partial L_p}{\partial w_0} = 0 \Rightarrow \sum_l \alpha_l y^{(l)} = 0$$

**Dual Optimization Problem** Setting gradient of  $L_p$  w.r.t.  $\mathbf{w}$  and  $w_0$  to 0, then plugging, we get dual optimization problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_l \alpha_l - \frac{1}{2} \sum_l \sum_{l'} \alpha_l \alpha_{l'} y^{(l)} y^{(l')} (\mathbf{x}^{(l)})^T \mathbf{x}^{(l')} \\ \text{subject to} \quad & \sum_l \alpha_l y^{(l)} = 0 \text{ and } \alpha_l \geq 0, \forall l \end{aligned}$$

**Support Vector** Most of the Dual Variables vanish with  $\alpha_l = 0$ . They are points lying beyond the margin, Support vectors:  $\mathbf{x}^{(l)}$  with  $\alpha_l > 0$ .

Computation of primal variables:  $\mathbf{w} = \sum_{l=1}^N \alpha_l y^{(l)} \mathbf{x}^{(l)} = \sum_{\mathbf{x}^{(l)} \in SV} \alpha_l y^{(l)} \mathbf{x}^{(l)}$

Support vector on margin:  $y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) = 1$  or  $w_0 = y^{(l)} - \mathbf{w}^T \mathbf{x}^{(l)}$ , Then:

$$w_0 = \frac{1}{|SV|} \sum_{\mathbf{x}^{(l)} \in SV} (y^{(l)} - \mathbf{w}^T \mathbf{x}^{(l)})$$

**Discriminant Function**  $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$  (plug in  $\mathbf{w}$  and  $w_0$  above), choose  $C_1$  if  $g(\mathbf{x}) > 0$

$K > 2$  An SVM  $g_i(\mathbf{x})$  is learned for each two-class problem. Choose  $C_j$  if  $j = \arg \max_k g_k(\mathbf{x})$

**Slack Variables** Relaxed constraint:

$$y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) \geq 1 - \zeta_l,$$

$$\text{minimize: } C \sum_l \zeta_l + \frac{1}{2} \|\mathbf{w}\|^2$$

Lagrangian(Primal):

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 +$$

$$C \sum_l \zeta_l - \sum_{l=1}^N \alpha_l [y^{(l)}(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) - 1 + \zeta_l] - \sum \mu_l \zeta_l,$$

$C$  is the regularization parameter,  $\mu_l$  is the new Lagrange multiplier to guarantee that  $\zeta \geq 0$

$$\text{Dual: Max } \sum_l \alpha_l - \frac{1}{2} \sum_l \sum_{l'} \alpha_l \alpha_{l'} y^{(l)} y^{(l')} (\mathbf{x}^{(l)})^T \mathbf{x}^{(l')}$$

$$\text{subject to: } \sum_l \alpha_l y^{(l)} = 0 \text{ and } 0 \leq \alpha_l \leq C, \forall l$$

**Kernel Functions** Dual Problem:

$$\text{Max: } \sum_l \alpha_l - \frac{1}{2} \sum_l \sum_{l'} \alpha_l \alpha_{l'} y^{(l)} y^{(l')} \phi(\mathbf{x}^{(l)})^T \phi(\mathbf{x}^{(l')})$$

$$\text{subject to: } \sum_l y^{(l)} = 0 \text{ and } 0 \leq \alpha_l \leq C, \forall l$$

$$\text{Kernel: } K(\mathbf{x}^{(l)}, \mathbf{x}^{(l')}) = \phi(\mathbf{x}^{(l)})^T \phi(\mathbf{x}^{(l')})$$

$\epsilon$  – *Insensitive Loss*

$$\varepsilon_\epsilon(y^{(l)}, f(\mathbf{x}^{(l)})) = \begin{cases} 0 & \text{if } |y^{(l)} - f(\mathbf{x}^{(l)})| \leq \epsilon \\ |y^{(l)} - f(\mathbf{x}^{(l)})| - \epsilon & \text{otherwise} \end{cases}$$

$$\text{Primal: Max } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_l (\zeta_l^+ + \zeta_l^-)$$

$$\text{Subject to: } y^{(l)} - (\mathbf{w}^T \mathbf{x}^{(l)} + w_0) \leq \epsilon + \zeta_l^+, \forall l$$

$$(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) - y^{(l)} \leq \epsilon + \zeta_l^-, \forall l, \zeta_l^+, \zeta_l^- \geq 0, \forall l$$

Two slack variables:  $\zeta_l^+$  such that  $y^{(l)} - (\mathbf{w}^T \mathbf{x}^{(l)} + w_0) > \epsilon$ ,  $\zeta_l^-$  such that  $(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) - y^{(l)} > \epsilon$

# 10 Performance Evaluation and Comparison

**K-Fold Cross Validation** The data set  $\mathcal{X}$  is randomly partitioned into  $K$  equal-sized subsets  $\mathcal{X}_i$ , **Stratification** the class distribution different subsets are kept roughly the same.

Use  $\mathcal{X}_1, \dots, \mathcal{X}_K$  as validation sets, and the remaining as training respectively.

If  $N$  is small,  $K$  should be large to allow large enough training sets.

Leave one out: one instance if left out as validation and  $N - 1$  for training.

**5 × 2 Cross Validation** For each fold  $i$ , Split into two equal-sized parts,  $\mathcal{X}_i^{(1)}, \mathcal{X}_i^{(2)}$ , 10 training/validation set pairs:  $\mathcal{T}_1 = \mathcal{X}_1^{(1)}, \mathcal{V}_1 = \mathcal{X}_1^{(2)}, \mathcal{T}_2 = \mathcal{X}_1^{(2)}, \mathcal{V}_2 = \mathcal{X}_1^{(1)}, \mathcal{T}_3 = \mathcal{X}_2^{(1)}, \mathcal{V}_3 = \mathcal{X}_2^{(2)}, \mathcal{T}_4 = \mathcal{X}_2^{(2)}, \mathcal{V}_4 = \mathcal{X}_2^{(1)}, \dots$

**Bootstrapping** Generates new samples, each of size  $N$ , by drawing randomly with replacement. Multiple bootstrap samples are used to max the change that the system is trained on all instances.

**Error Measure** TP: True positive, FP: False postivie, FN:False Negative, TN: True Negative, Error rate:  $\frac{|FN|+|FP|}{N}$

**Receiver Operating characteristics curve** Hit rate:  $\frac{|TP|}{|TP|+|FN|}$  False alarm rate:  $\frac{|FP|}{|FP|+|TN|}$ , area under the curve is often used.

**Point vs Interval estimator** Point specifies a value for  $\theta$ , intervale specifies an interval within which  $\theta$  lines with a certain degree of confidence.

**Confidence Interval** Define  $z_\alpha$ ,

Two sided:  $P(\mathcal{Z} > z_\alpha) = \alpha$ , then for level of confidence  $1 - \alpha$ ,

$$P(-z_{\alpha/2} < \sqrt{N} \frac{m - \mu}{\sigma} < z_{\alpha/2}) = 1 - \alpha$$

One-sided  $P(m - z_{\alpha} \frac{\sigma}{\sqrt{N}} < \mu) = 1 - \alpha$

**t Distribution**  $\sigma^2$  replaced by sample var  $S^2 = \frac{\sum_l (x^{(l)} - m)^2}{N-1}$ ,  $\sqrt{N}(m - \mu)/S$  follows a t distribution with  $N - 1$  degree of freedom:  $\sqrt{N} \frac{m - \mu}{S} \sim t_{N-1}$  For  $\alpha \in (0, 1/2)$ ,

$$P(m - t_{\alpha/2, N-1} \frac{S}{\sqrt{N}} < \mu < m + t_{\alpha/2, N-1} \frac{S}{\sqrt{N}}) = 1 - \alpha$$

**Hypothesis Testing** Test some hypothesis concerning the parameters.

1. Define a statistics obey a certain distribution
2. Random sample is consistent with the hypothesis under consideration, accept(not reject)
3. Otherwise reject

Given normal  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma$  known,  $\mu$  unknown. **Null Hypothesis:**  $H_0 : \mu = \mu_0$ ,  $H_1 : \mu \neq \mu_0$  Accept  $H_0$  if the sample mean is not too far from  $\mu_0$ . Accept  $H_0$  with level of significance  $\alpha$  if  $\mu_0$  lies in the  $100(1 - \alpha)$ ,

$$\sqrt{N} \frac{m - \mu_0}{\sigma} \in (-z_{\alpha/2}, z_{\alpha/2})$$

**Error** Type I, rejected when it is correct,  $\alpha$  defines how much type I can tolerate. Type II accepted when incorrect.

**One Sided**  $H_0 : \mu \leq \mu_0, H_1 : \mu > \mu_0$ , accept  $\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in (-\infty, z_\alpha)$

**t-test** If  $\sigma^2$  is not known,  $H_0 : \mu = \mu_0, H_1 : \mu \neq \mu_0$ , accept at  $\alpha$  if

$$\sqrt{N} \frac{m - \mu_0}{S} \in (-t_{\alpha, N-1}, t_{\alpha, N-1})$$

**Binomial Test** Single test/validation: Classifier is trained on a training set  $\mathcal{T}$  and tested on validation set  $\mathcal{V}$ .  $p$  be the probability that the classifier makes a misclassification error.. Define  $x^{(l)}$  as a Bernoulli variable to denote the correctness,  $x^{(l)} = 1$  with probability  $p$  and  $x^{(l)} = 0$  with probability  $1 - p$ , point estimation:  $\hat{p} = \frac{\sum_l x^{(l)}}{|\mathcal{V}|} = \frac{\sum_l x^{(l)}}{N}$

Hypothesis test:  $H_0 : p \leq p_0$  vs.  $H_1 : p > p_0$ ,  $X = \sum_{l=1}^N x^{(l)}$  denote the number of errors on  $\mathcal{V}$ .,

$$P(X = j) = \binom{N}{j} p^j (1-p)^{N-j}$$

Under the null hypothesis  $p \leq p_0$ , so the probability that there are  $e$  errors or less is:

$$P(X \leq e) = \sum_{j=1}^e \binom{N}{j} p_0^j (1-p_0)^{N-j}$$

Binomial test: accept  $H_0$  if  $P(X \leq e) < 1 - \alpha$

**Paired t Test** Multiple training/validation set pairs: run the algorithm  $K$  times on  $K$  T/V set pairs, we get  $K$  error probabilities,  $p_i, i = 1, \dots, K$  on  $K$  validation sets.

Paired t test to determine whether to accept the null hypothesis  $H_0$  that the classifier has error probability  $p_0$  or less at significance level  $\alpha$ .

$$x_i^{(l)} = \begin{cases} 1 & \text{if classifier trained on } \mathcal{V}_i \text{ makes an error on instance } l \text{ of } \mathcal{V}_i \\ 0 & \text{otherwise} \end{cases}$$

Test statistic:  $\sqrt{K} \frac{(m-p_0)}{S} \sim t_{K-1}$ , where  $p_i = \frac{\sum_{l=1}^N x_i^{(l)}}{N}$ ,  $m = \frac{\sum_{i=1}^K p_i}{K}$ ,  $S^2 = \frac{\sum_{i=1}^K (p_i - m)^2}{K-1}$

**K-Fold Cross-Validated Paired t Test** Given two classification algorithms and a data set, we want to compare and test whether the two algorithms construct classifiers that have the same expected error rate on a new instance.

K-fold cross validation is used to obtain  $K$  training/validation set pairs  $\{(\mathcal{T}_i, \mathcal{V}_i)\}_{i=1}^K$ , run two algorithms, error probabilities  $p_i^1$  and  $p_i^2$ , define  $p_i = p_i^1 - p_i^2$ ,  $p_i$  with mean  $\mu$

$H_0 = \mu = 0, H_1 = \mu \neq 0$ . Test statistic:  $\sqrt{K} \frac{(m-0)}{S} \sim t_{K-1}$ , where  $m = \frac{\sum_{i=1}^K p_i}{K}$ ,  $S^2 = \frac{\sum_{i=1}^K (p_i - m)^2}{K-1}$ , accept  $H_0$  at significance level  $\alpha$  if  $\sqrt{K} m / S \in (-t_{\alpha/2, K-1}, t_{\alpha/2, K-1})$

# 11 Ensemble Learning

**Combining Base Learners** Multiexpert combination method: base learners work in parallel, give decision and combined to give a final.

Multistage combinations: Base learners work serially, sorted increasing complexity, complex is used when simple is not confident.

**Voting** Takes a convex combination of the base learners:

$$y = f(d_1, \dots, d_L | \Phi) = \sum_{j=1}^L w_j d_j(\mathbf{x})$$

, with  $w_j \geq 0$  and  $\sum_{j=1}^L w_j = 1$ ,  $\Phi = (w_1, \dots, w_L)^T$  are the parameters and  $y$  is the final prediction.

**Voting for Classification** for class  $C_i$   $y_i = \sum_{j=1}^L w_j d_{ij}(\mathbf{x})$ , where  $d_{ij}$  is the vote of learner  $j$  for  $C_i$

Simple voting  $w_j = \frac{1}{L}$

Bayesian model combination:  $P(C_i|x) = \sum_{models \mathcal{M}_j} P(C_i|x, \mathcal{M}_j)P(\mathcal{M}_j)$ , weight  $w_j$  can be seen as approximation of the prior  $P(\mathcal{M}_j)$ , Analysis, as  $L$  increase, bias does not change but the variance decreases.

**Bagging** bootstrap aggregation, a voting method whereby the base learners are made different by training on slightly different training sets.

**AdaBoost** Modifies the probabilities of drawing instances for classifier training as a function of the error of

**Training:**

For all  $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$ , initialize  $p_1^t = 1/N$

For all base-learners  $j = 1, \dots, L$

Randomly draw  $\mathcal{X}_j$  from  $\mathcal{X}$  with probabilities  $p_j^t$

Train  $d_j$  using  $\mathcal{X}_j$

For each  $(x^t, r^t)$ , calculate  $y_j^t \leftarrow d_j(x^t)$

Calculate error rate:  $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If  $\epsilon_j > 1/2$ , then  $L \leftarrow j - 1$ ; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each  $(x^t, r^t)$ , decrease probabilities if correct

If  $y_j^t = r^t$   $p_{j+1}^t \leftarrow \beta_j p_j^t$  Else  $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$Z_j \leftarrow \sum_t p_{j+1}^t$ ;  $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

**Testing:**

Given  $x$ , calculate  $d_j(x), j = 1, \dots, L$

Calculate class outputs,  $i = 1, \dots, K$ :

$$y_i = \sum_{j=1}^L \left( \log \frac{1}{\beta_j} \right) d_{ji}(x)$$

the previous base learner.

## 11.1 Matrices Properties

**Basic Matrix**  $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$ ,  $(\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ ,  $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$ ,  $\mathbf{P}^{-1} + \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B})^{-1} \mathbf{B}^T \mathbf{R}^{-1} = \mathbf{PB}^T(\mathbf{BPB}^T + \mathbf{R})^{-1}$ ,

**Traces and Determinants**  $Tr(\mathbf{AB}) = Tr(\mathbf{BA})$ ,  $Tr(\mathbf{ABC}) = Tr(\mathbf{CBA}) = Tr(\mathbf{BCA})$ ,  $|\mathbf{A}^{-1}| = \frac{1}{|\mathbf{A}|}$ ,  $\mathbf{a}^T \mathbf{A} \mathbf{a} = Tr(\mathbf{A} \mathbf{a} \mathbf{a}^T)$

**Matrix Derivatives**  $\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}^T \mathbf{a}) = \frac{\partial}{\partial \mathbf{x}}(\mathbf{a}^T \mathbf{x}) = \mathbf{a}$ ,  $\frac{\partial}{\partial \mathbf{x}}(\mathbf{AB}) = \frac{\partial \mathbf{A}}{\partial \mathbf{x}} \mathbf{B} + \mathbf{A} \frac{\partial \mathbf{B}}{\partial \mathbf{x}}$ ,  $\frac{\partial}{\partial x}(\mathbf{A}^{-1}) = -\mathbf{A}^{-1} \frac{\partial \mathbf{A}}{\partial x} \mathbf{A}^{-1}$ ,  $\frac{\partial}{\partial x} \ln |\mathbf{A}| = Tr(\mathbf{A}^{-1} \frac{\partial \mathbf{A}}{\partial x})$ ,  $\frac{\partial}{\partial A_{ij}} Tr(\mathbf{AB}) = B_{ji}$ ,  $\frac{\partial}{\partial \mathbf{A}} Tr(\mathbf{AB}) = \mathbf{B}^T$ ,  $\frac{\partial}{\partial \mathbf{A}} Tr(\mathbf{A}) = \mathbf{I}$ ,  $\frac{\partial}{\partial \mathbf{A}} Tr(\mathbf{ABA}^T) = \mathbf{A}(\mathbf{B} + \mathbf{B}^T)$ ,  $\frac{\partial}{\partial \mathbf{A}} \ln |\mathbf{A}| = (\mathbf{A}^{-1})^T$

# 12 Convolutional Neural Networks

## 12.1 Motivation

There exist a complex arrangement of cells within the visual cortex. These cells are sensitive to small sub-regions of the input space, called a **receptive field**, and are tiled in such a way as to cover the entire visual field. These filters are local in input space and are thus better suited to exploit the strong spatially local correlation present in natural images.

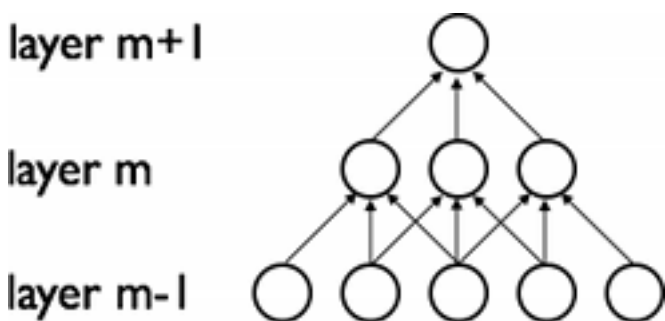
Two basic cell types: Simple cells (S) and complex cells (C).

- Simple Cells(S) respond maximally to specific edge-like stimulus patterns within their receptive field.
- Complex cells(C) have larger receptive fields and are locally invariant to the exact position of stimulus.

## 12.2 Sparse Connectivity

CNNs exploit spatially local correlation by **enforcing a local connectivity pattern between neurons of adjacent layers**.

Input hidden units in the  $m$ -th layer are connected to a local subset of units in the  $(m - 1)$ -th layer, which have spatially contiguous receptive fields.



- Layer  $m - 1$  is the input retina.
- Units in layer  $m$  have receptive fields of width 3. Thus only connected to 3 adjacent neurons in the  $(m - 1)$ -th layer.
- Layer  $m$  have a similar connectivity with the layer below.  $(m + 1)$

We say that their receptive field with respect to the layer below is 3, but their receptive field with respect to the input is larger (it is 5). The architecture thus confines the learnt “filters” to the spatially local pattern.

## 12.3 Shared Weights

In CNNs, each sparse filter  $h_i$  is additionally replicated across the entire visual field. These “replicated” units form a **feature map**, which share the same parameterization. (the receptive fields in the same feature map share the same weight and bias for the sensitive region)

Gradient descent can still be used to learn such shared parameters, and requires only a small change to the original algorithm. The gradient of a shared weight is simply the sum of the gradients of the parameters being shared.

Replicating units in this way allows for feature to be detected regardless of their position in the visual field. Additionally, weight sharing offers a very efficient way to do this, since it greatly reduces the number of free parameter

## 12.4 Details and notation

The  $k$ -th feature map at a given layer as  $h^k$ , whose filter with weights  $W^k$  and bias  $b_k$ , then the feature map  $h^k$  is obtained as follows (for  $\tanh$  non-linearities)

$$h_{ij}^k = \tanh \left( \left( W^k * x \right)_{ij} \right) + b_k$$

To form a richer representation of the data, hidden layers are composed of a set of multiple feature maps  $\{h^{(k)}, k = 0 \dots K\}$

The weights of this layer can be parameterized as a 4D tensor (Tensors are geometric objects that describe linear relations between vectors, scalars, and other tensors. Elementary examples of such relations include the dot product, the cross product and linear maps, vectors and scalars themselves are also tensors.)

- Destination feature map index
- Source feature map index
- Source vertical position index
- Source horizontal position index

## 12.5 MaxPooling

MaxPooling is a form of non-linear down-sampling. MaxPooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value.

Useful for two reasons:

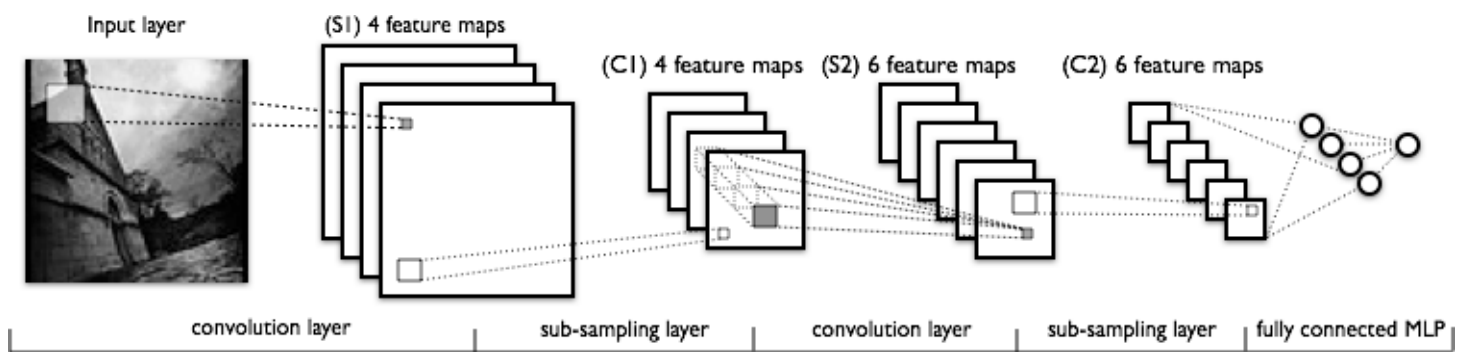
- Reduces the computational complexity for upper layers
- Provides a form of translation invariance

Why it works: Imagine cascading a max-pooling layer with a convolutional layer. There are 8 directions in which one can translate the input image into a single pixel. If max-pooling is done over a  $2 \times 2$  region, 3 out of these 8 possible configurations will produce exactly the same output at the convolutional layer. For max-pooling over a  $3 \times 3$



## 12.6 The Full Model: LeNet

Graphical depiction:



The lower layers are composed to alternating convolution and max-pooling layers.

The upper layers however are fully-connected and correspond to a traditional MLP



# 13 Denoising Autoencoders

An extension of a classical autoencoder and it was introduced as a building block for deep networks.

## 13.1 Autoencoders

An auto-encoder is trained to encode the input  $\mathbf{x}$  into some representation  $\mathbf{c}(\mathbf{x})$  so that the input can be reconstructed from that representation. Hence the target output of the auto-encoder is the auto-encoder input itself.

An autoencoder takes an input  $\mathbf{x} \in [0, 1]^d$

First maps it with an encoder to a hidden representation  $\mathbf{y} \in [0, 1]^{d'}$  through a deterministic mapping:

$$\mathbf{y} = s(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Where  $s$  is a non-linearity such as the sigmoid.

The latent representation  $\mathbf{y}$ , or **code** is then mapped back (with a decoder) into a **reconstruction**  $\mathbf{z}$  of same shape as  $\mathbf{x}$  though a similar transformation:

$$\mathbf{z} = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$$

. Where  $'$  does NOT indicate transpose, and  $\mathbf{z}$  should be seen as prediction of  $\mathbf{x}$ , given the code  $\mathbf{y}$

The parameter of the model  $W$  are optimized such that **the average reconstruction error is minimized**.

Measure the reconstruction error using the traditional squared error  $L(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2$

If the input is interpreted as either bit vectors or vectors of bit probability by the reconstruction cross-entropy defined as (if  $\mathbf{x}|\mathbf{y}$  is in Gaussian):

$$L_H(\mathbf{x}, \mathbf{z}) = -\log P(\mathbf{x}|\mathbf{y}) = -\sum_{k=1}^d [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)]$$

The hope:  $\mathbf{y}$  is a distributed representation that captures the coordinates along the main factors of variation in the data.

## 13.2 Denoising Autoencoder

The idea behind denoising autoencoders is that in order to force the hidden layer to discover more robust features and prevent it from simply learning the identity (copy the input to output), we train the autoencoder to reconstruct the input from a corrupted version of it.

The denoising auto-encoder is a stochastic version of the auto encoder. It does two things:

- Try to encode the input (preserve the information)
- Try to undo the effect of a corruption process stochastically applied to the input of the auto-encoder.

The stochastic corruption process consists in randomly setting some of the inputs (as many as half of them) to zero. Hence the denoising auto-encoder is trying to **predict corrupted values from the uncorrupted values**, for randomly selected subsets of missing patterns. Note how being able to predict any subset of variables from the rest is a sufficient condition for completely capturing the joint distribution between a set of variables.

To convert the autodecoder class into a denoising autoencoder class, all we need to do is to add a stochastic corruption step operating on the input.

# 14 Stacked Denoising Autoencoders(SDA)

The denoising autoencoders can be stacked into form a deep network by **feeding the latent representation(output code)** of the denoising autoencoder found on the layer below as input to the current layer.

The unsupervised pre-training of such an architecture is done one layer at a time.

Each layer is trained as a denoising auto-encoder by minimizing the reconstruction of its input (output code of the previous layer).

Once the first  $k$ -layers are trained, we can train the  $(k + 1)$ -th layer because we can now compute the code (latent representation) from the layer below.

Once all layers are pre-trained, the network goes through a second stage of training called **fine-tuning**.

**Supervised fine-tuning** Minimize the prediction error on a supervised task.

1. Add a logistic regression layer on top of the network (the out put code of the output layer)
2. Train the entire network as we would train a multilayer perceptron.
3. At this point, we only consider the encoding parts of each auto-encoder.

## 14.1 Multi-Label Classification

### 14.1.1 HOMER Algorithm

#### Training

General idea: transform large set of labels  $L$  into a tree-shaped hierarchy of simpler multi-label classification tasks.

- Each node  $n$  of the tree contains  $L_n \subseteq L$
- There are  $|L|$  leaves, each one containing single distinct label  $\lambda_i \in L$
- Each node contains the union of the label set of its children.  $L_{root} = L$

*Meta-label:* of a node  $n$ ,  $\mu_n$  as the disjunction of the labels contained in the node. A training example can be considered annotated with meta-label  $\mu_n$  if it is annotated with at least one of the labels in  $L_n$

Each internal node  $n$  of the hierarchy also contains a multilabel classifier  $h_n$ .

Task of  $h_n$ : the prediction of one or more of the labels of its children. Set of labels for  $h_n$

$$M_n = \{\mu_c | c \in \text{children}(n)\}$$

**Prediction**

1. Starts with  $h_{root}$
2. Follows a recursive process forwarding  $x$  to the multilabel classifier  $h_c$  of a child node  $c$  only if  $\mu_c$  is among the prediction of  $h_{parent(c)}$
3. Eventually, this process may lead to the prediction of one or more single-labels by the multi-label classifiers.
4. The union of these predicted single-labels is the output of the proposed approach in this case.

**Training**

1. Assume the existence of a set  $D = \{(\mathbf{x}_i, \mathbf{Y}_i) | i = 1 \dots |D|\}$ , each one consists of a feature vector  $\mathbf{x}_i$  and set of labels  $\mathbf{Y}_i \subseteq L$ .
2. Recursively in a top-down depth-first fashion starting with the root
3. At each  $n$ ,  $k$  children nodes are first created, unless  $|L_n| < k$ , in which the number of children is  $|L_n|$ .
4. Each such child  $n$  filters the data of its parent, keeping only the examples that are annotated with at least one of its own labels.
5. Two main processes are then sequentially executed:
  - a) labels of the current node are distributed into  $k$  disjoint subsets, one for each child of the current node
  - b) A multilabel classifier is trained for the prediction of the meta-labels of its children.
6. The approach recurses into each child node that contains no more than a single label.

Grouping: balanced k-means.

**14.1.2 RAKEL**