Machine Learning

LI Tao

March 27, 2014

Contents

1 Probability Theory

1.1 Expectations and covariances

1.1.1 Expectation

The average value of some function $f(x)$ under a probability distribution $p(x)$ is called the *expectation* of $f(x)$ and will be denoted by $\mathbb{E}[f]$, for discrete:

$$\mathbb{E}[f] = \sum_x p(x)f(x) \tag{1.1}$$

In the case of continuous variables:

$$\mathbb{E}[f] = \int p(x)f(x)dx \tag{1.2}$$

Approximation:

$$\mathbb{E}[f] \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n) \tag{1.3}$$

Sometimes we will be considering expectations of functions of several variables, in which case we can use a subscript to indicate which variable is being averaged over:

$$\mathbb{E}_x[f(x,y)] \tag{1.4}$$

The average of the function $f(x,y)$ average with respect to the distribution of x, it will be a function of y.

The conditional expectation:

$$\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x) \tag{1.5}$$

1.1.2 Variance

The *variance* of $f(x)$:

$$\text{var}[f] = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])^2\right] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2 \tag{1.6}$$

In particular:

$$\text{var}[x] = \mathbb{E}[x^2] - \mathbb{E}[f(x)]^2 \tag{1.7}$$

For the two variables x and y the *covariance* is defined by:

$$\text{cov}[x, y] = \mathbb{E}_{x,y}\left[\{x - \mathbb{E}[x]\}\{y - \mathbb{E}[y]\}\right] \tag{1.8}$$

$$= \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] \tag{1.9}$$

Express the extent to which x and y **vary together**. If x and y are independent, then their covariance vanishes.

1.2 Bayesian Probabilities

View probabilities in terms of hte frequencies of random, repeatable events: *classical* or *frequentist*

Now we turn to Bayesian view, in which probabilities provide a quantification of uncertainty.

We capture out assumptions about $\mathbf{w}$, *before* observing the data, in the form of a prior probability distribution $p(\precapprox)$

The effect of the observed data $\mathcal{D}$ is expressed through the conditional probability $p(calD|\mathbf{w})p(\mathbf{w})$, then the Bayesian's theorem:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \tag{1.10}$$

Evaluate the uncertainty in $\mathbf{w}$ *after* we have observed $\mathcal{D}$

Likelihood function: $p(\mathcal{D}|\mathbf{w})$: evaluated for the observed data set $\mathcal{D}$ and can be viewed as a function of $\mathbf{w}$. It expresses how probable the observed data set is for different settings of the parameter.

In both Bayesian and frequentist, likelihood function $p(\mathcal{D}|\mathbf{w})$ plays a central role. However, they are different in: E

- In a frequentist setting, $\mathbf{w}$ is considered to be a **fixed** parameter, whose value is determined by some form of "estimator", and error bars on this estimate are obtained by considering the distribution of possible data sets $\mathcal{D}$.

- From the Bayesian view point there is only a single data set $\mathcal{D}$ and the uncertainty in th eparameters is expressed through a probability distribution over $\mathbf{w}$

2 Probability Distributions

Density estimation: model the probability distribution $p(\mathbf{x})$ of a random variable $\mathbf{x}$, given a finite set $\mathbf{x}_1 \dots \mathbf{x}_N$ of observation.

Assumption: data points are independent and identically distributed (i.i.d)

- *binomial* and *multinomial* distribution for discrete
- *Gaussian distribution* for continuous random variables.

These are specific examples of *parametric* distribution because they are governed by a samll number of adaptive parameters.

- Frequentist treatment: choose specific value for the parameters by optimizing some criterion, such as likelihood function
- Bayesian treatment: **introduce prior distributions over the parameter**, and then use Baye's theorem to compute the corresponding posterior distribution given the observed data.

2.1 Binary Variables

(Flipping coins) $x \in \{0, 1\}$, $p(x = 1|\mu) = \mu$, $p(x = 0|\mu) = 1-\mu$, where μ is the parameter, the probability distribution:

$$\text{Bern}(x|\mu) = \mu^x(1 - \mu)^{1-x}$$

It's the *Bernoulli* distribution.

The mean and variance are given by:

$$\mathbb{E}[x] = \mu$$
$$\text{var}[x] = \mu(1 - \mu)$$

Now dataset $\mathcal{D} = \{x_1, \dots, x_N\}$. We can construct the **likelihood function** (function of μ).

The likelihood function is the probability of drawing this dataset.

Assume that the data set is drawn from $p(x|\mu)$, the likelihood function:

$$p(\mathcal{D}|\mu) = \prod_{n=1}^{N} p(x_n|\mu) = \prod_{n=1}^{N} \mu^{x_n}(1-\mu)^{1-x_n}$$

In a frequentist setting, we can estimate a value for μ by maximizing the likelihood function. Or the log likelihood:

$$\ln p(\mathcal{D}|\mu) = \sum_{n=1}^{N} \ln p(x_n|\mu) = \sum_{n=1}^{N} \{x_n \ln \mu + (1-x_n)\ln(1-\mu)\}$$

If we set the derivative of $\ln p(\mathcal{D}|\mu)$ w.r.t. μ equal to zero, we have an maximum likelihood estimator:

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

It's the *sample mean*

Binomial distribution: number m of observations of $x = 1$ given that the data set has size N:

$$\mathrm{Bin}(m|N,\mu) = \binom{N}{m}\mu^m(1-\mu)^{N-m}$$

$$\mathbb{E}[m] = \sum_{m=1}^{N} \mathrm{mbox}(m|N,\mu) = N\mu$$

$$\mathrm{var}[m] = \sum_{m=0}^{N} (m-\mathbb{E}[m])^2 \mathrm{Bin}(m|N,\mu) = N\mu(1-\mu)$$

2.1.1 The beta distribution

Prior distribution $p(\mu)$ over the parameter μ

Take the form of the products of factors of the form $\mu^x(1-\mu)^{1-x}$

Choose a prior to be proportional to power of μ and $(1-\mu)$ then the posterior will have the same functional form as the prior.

Posterior and prior have the same form.

The prior is *beta* distribution:

$$\text{Beta}(\mu|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\mu^{a-1}(1-\mu)^{b-1}$$

$$\mathbb{E}[\mu] = \frac{a}{a+b}$$

$$\text{var}[\mu] = \frac{ab}{(a+b)^2(a+b+1)}$$

a and b are hyperparameters (parameters of parameters)

Then the posterior:

$$p(\mu|m,l,a,b) = \frac{\Gamma(m+a+l+b)}{\Gamma(m+a)\Gamma(l+b)}\mu^{m+a-1}(1-\mu)^{l+b-1}$$

Simple interpretation of a and b: effective number of observations of $x = 1$ and $x = 0$. (They add to the counts of samples).

Sequential approach to learning:

1. Taking observations one at a time
2. After each observation updating the current posterior. Posterior is gotten by Multiplying by the likelihood ($P(\mathcal{D}|\theta)$) for the new observation, and then normalizing to obtain new posterior.
3. At each stage, the posterior is a beta distribution with some total number of (prior plus actual) observed values for $x = 1$ and $x = 0$, given by the parameter a and b.
4. Incorporation of an additional observation of $x = 1$ simply corresponds to incrementing the value of a by 1.

Sequential methods make use of observations one at a time, or in small batches, and then discard them befroe the next observations are used.

To predict:

$$p(x=1|\mathcal{D}) = \int_0^1 p(x=1|\mu)p(\mu|\mathcal{D})du = \int_0^1 \mu p(\mu|\mathcal{D})du = \mathbb{E}[\mu|\mathcal{D}]$$

Using the result of the mean of beta distribution:

$$p(x=1|\mathcal{D}) = \frac{m+a}{m+a+l+b} \tag{2.1}$$

The total fraction of observations (real obsercations and fictitious prior observations) that correspond to $x = 1$.

Note that in the limit of an infinitely large $m, l \to \infty$ the result reduces to the maximum likelihood result.

2.2 Multinomial Variables

Variables ca take on one of K possible mutually exclusive states.

Represented by a K-D vector $\mathbf{x}$ in which one of $x_k = 1$. Denote the probability of $x_k = 1$ by parameter μ_k, then the distribution of $\mathbf{x}$:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k} \tag{2.2}$$

The likelihood

$$P(\mathcal{D}|\boldsymbol{\mu}) = \prod_{n=1}^{K} \prod_{k=1}^{K} \mu_k^{x_{nk}} = \prod_{k=1}^{K} \mu_k^{\sum_n x_{nk}} = \prod_{k=1}^{K} \mu_k^{m_k} \tag{2.3}$$

The estimation can be achieved with the help of Lagrange:

$$\mu_k^{\mathrm{ML}} = \frac{m_k}{N} \tag{2.4}$$

Multinominal distribution, the joint distribution of $m_1, \ldots, m_K$ conditioned on the parameters $\boldsymbol{\mu}$ and N

$$\mathrm{Mult}(m_1, m_2, \ldots, m_K|\boldsymbol{\mu}, N) = \binom{N}{m_1 m_2 \ldots m_K} \prod_{k=1}^{K} \mu_k^{m_k} \tag{2.5}$$

Where

$$\binom{N}{m_1 m_2 \ldots m_K} = \frac{N!}{m_1! m_2! \ldots m_K!} \tag{2.6}$$

2.2.1 The Dirichlet distribution

The conjugate prior:

$$p(\boldsymbol{\mu}|\boldsymbol{\alpha}) \propto \prod_{k=1}^{K} {\mu_k}^{\alpha_k - 1} \tag{2.7}$$

The normalized form:

$$\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_K)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1} \tag{2.8}$$

Where $\Gamma(x)$ is the gamma function and $a_0 = \sum_{k=1}^{K} \alpha_k$.

2.3 Gaussian Distribution

The D-dimensional vector $\mathbf{x}$, the multivariate Gaussian distribution takes the form:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \tag{2.9}$$

Where μ is a D-dimensional mean vector, $\boldsymbol{\Sigma}$ is a $D \times D$ covariance matrix, and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$.

2.3.1 Partitioned Gaussians

Given a joint Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Lambda} \equiv \boldsymbol{\Sigma}^{-1}$ and

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix}, \quad \boldsymbol{\mu} = (\boldsymbol{\mu}_a, \boldsymbol{\mu}_b) \tag{2.10}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{pmatrix} \tag{2.11}$$

Conditional distribution:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Lambda}_{aa}^{-1}) \tag{2.12}$$

$$\boldsymbol{\mu}_{a|b} = \mu_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b) \tag{2.13}$$

2.4 The Exponential Family

Members of the *exponential family* have many important properties in common, and it is illuminating to discuss these properties in some generality.

The exponential family of distributions over $\mathbf{x}$, given parameter $\boldsymbol{\eta}$, is defined to be the set of distributions of the from:

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta}) \exp\left\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\right\} \tag{2.14}$$

Here the $\boldsymbol{\eta}$ are called the *natural parameters* of the distribution, and $\mathbf{u}(\mathbf{x})$ is some function of $\mathbf{x}$.

The function $g(\boldsymbol{\eta})$ can be interpreted as the coefficient that ensures that the distribution is normalized and therefore satisfies

$$g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\right\} d\mathbf{x} = 1 \tag{2.15}$$

Maximum likelihood and sufficient statistics

Problem: estimating the parameter vector $\boldsymbol{\eta}$ in the general exponential family distribution 2.14 using the technique of maximum likelihood.

Taking the gradient of both sides of 2.15:

$$\nabla g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\right\} dx + g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\right\} \mathbf{u}(\mathbf{x}) d\mathbf{x} = 0 \tag{2.16}$$

Rearrange and make use again of 2.15

$$-\frac{1}{g(\boldsymbol{\eta})} \nabla g(\boldsymbol{\eta}) = g(\boldsymbol{\eta}) \int \int \exp\left\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\right\} \mathbf{u}(\mathbf{x}) d\mathbf{x} = \mathbb{E}[\mathbf{u}(\mathbf{x})] \tag{2.17}$$

We therefore have

$$-\nabla \ln g(\boldsymbol{\eta}) = \mathbb{E}[\mathbf{u}(\mathbf{x})] \tag{2.18}$$

Now a dataset $\mathbf{X} = \{\mathbf{x}_1, \ldots \mathbf{x}_n\}$, the likelihood:

$$p(\mathbf{X}|\boldsymbol{\eta}) = \left(\prod_{n=1}^{N} h(\mathbf{x}_n) \right) g(\boldsymbol{\eta})^N \exp\left\{ \boldsymbol{\eta}^T \sum_{n=1}^{N} \mathbf{u}(\mathbf{x}_n) \right\} \tag{2.19}$$

Setting the gradient of $\ln p(\mathbf{X}|\boldsymbol{\eta})$ w.r.t $\boldsymbol{\eta}$ to zero, we get the following condition to be satisfied by MLE $\boldsymbol{\eta}_{ML}$

$$-\nabla \ln g(\boldsymbol{\eta}_{ML}) v = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\mu}(\mathbf{x}_n) \tag{2.20}$$

The solution for the maximum likelihood estimator depends on the data only through $\sum_n \mathbf{u}(\mathbf{x}_n)$, which is therefore called the *sufficient statistic* of the distribution.

We do not need to store the entire data set itself but only the value of sufficient statistic.

3 Bayesian Decision Theory

3.1 Bernoulli

Bernoulli:

$$P(X) = p_0^X (1 - p_0)^{1-X}$$

, Where p_0 is the parameter

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

$$\hat{p_0} = \frac{\#\text{heads}}{\#\text{tosses}} = \frac{\sum_{l=1}^N x^{(l)}}{N}$$

Predict outcome = head if $P_0 > 1/2$, tail otherwise. The prediction always output the result with greatest possibility.

3.2 Baye's Rule

Baye's Rule:

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

Baye's for $K > 2$ classes:

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

3.3 Optimal decision

Choose C_i if

$$P(C_i|\mathbf{x}) = \max_k P(C_k|\mathbf{x})$$

Namely, choose C_i if the occurrence of C_i has the highest probability giving instances $\mathbf{x}$

Losses and Risks :
$R(\alpha_i|\mathbf{x}) = \sum_{k=1}^{K} \lambda_{ik} P(C_k|\mathbf{x})$,
α_i the action assigned to class C_i, λ_{ik} loss for α_i if C_k

Optimal:α_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:

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

λ is the loss for choosing reject.

Expected risk:

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

Optimal Decision: choose C_i if
$R(\alpha_i|x) = \min_{1 \leq k \leq K} R(\alpha_i|\mathbf{x}) < R(\alpha_{K+1}|\mathbf{x})$,
Reject otherwise.

Discriminant Functions : choose C_i if

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

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

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:

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

4 Parameter Estimation

4.1 Max Likelihood Estimation

4.1.1 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 $p(\mathbf{x}|\theta)$. θ is the parameter (possibly a vector).

The goal: Estimate θ 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

4.2 MLE

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

Likelihood:

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

That is: the probability of drawing $\mathcal{X}$ (training data) giving the parameter θ

Log likelihood:

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

Max Likelihood estimate (MLE)

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

In words, find a parameter θ such that, if we sample some data point follows this distribution with parameter θ, then it is most likely that we will get our learning set.

4.2.1 Example: Assume the data follow Bernoulli

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

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

That is, the probability of drawing x given p_0 (the parameter)

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.

$$\begin{aligned}
\frac{d\mathcal{L}(p_0|\mathcal{X})}{dp_0} &= \sum_{l}^{N} \left[\frac{x^{(l)}}{p_0} - \frac{1 - x^{(l)}}{1 - p_0} \right] \\
&= \sum_{l=1}^{N} \frac{x^l - p_0}{p_0(1 - p_0)} = 0 \\
&\Leftrightarrow \sum_{l=1}^{N} x^{(l)} - Np_0 = 0 \\
&\Leftrightarrow p_0 = \frac{1}{N} \sum_{l=1}^{N} x^{(l)}
\end{aligned}$$

4.2.2 Example: Multinomial Ran Variable

$\mathbf{x}$ 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

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

Log likelihood

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

ML estimate

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

4.2.3 Example Normal

p. d. f. :

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

Log likelihood

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

ML Estimates

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

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

4.2.4 Multivariable Normal

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

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

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

ML estimates

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

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

4.3 Bias and Variance

Bias:

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

Variance:

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

(d is estimator of param θ)

Mean Squared error

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

4.4 Bayesian Estimation

Treat θ as a Random Variable, that is, the parameter θ 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 $y = g(x|\theta)$:

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

4.5 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 μ, σ and σ_0
$x^{(l)} \sim \mathcal{N}(\theta, \sigma_0^2)$, $\theta \sim \mathcal{N}(\mu, \sigma^2)$
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: $p(x|C_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right]$

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

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

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: $r = f(x) + \epsilon$

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: $\mathcal{X} = \left\{(x^{(l)}, r^{(l)})\right\}$:

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

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

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

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

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

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

Bias and Variance Expected squared error of sample $\mathcal{X}$ $E\left[(r - g(x))^2|x\right] = (E\left[r|x\right] - g(x))^2 +$ $E\left[(r - E\left[r|x\right])^2|x\right]$ = squared err + noise
Average over $\mathcal{X}$: $E_{\mathcal{X}} = \left[(E[r|x] - g(x))^2|x\right] = (E[r|x] - E_{\mathcal{X}}[g(x)])^2 + E_{\mathcal{X}}\left[(g(x) - E_{\mathcal{X}}[g(x)])^2\right]$ = bias + variance

5 Multivariate Method

Multivariate Data N i.i.d. instances:

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

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

Covariance of x_i and x_j:

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

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

Covariance matrix:

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

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

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

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

Bivariate Normal Distribution Covariance matrix:

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

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

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

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

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

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

$\mathbf{m}_i = \frac{\sum_l r_i^{(l)} \mathbf{x}^{(l)}}{\sum_l r_i^{(l)}}$

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

Quadratic Discriminant Functions

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

where $\mathbf{W}_i = -\frac{1}{2}\mathbf{S}_i^{-1}$

$\mathbf{w}_i = \mathbf{S}_i^{-1}\mathbf{m}_i$

$w_{i0} = -\frac{1}{2}\mathbf{m}_i^T \mathbf{S}_i^{-1}\mathbf{m}_i - \frac{1}{2}\log|\mathbf{S}_i| + \log\hat{P}(C_i)$

6 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:

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

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

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.

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

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

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

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

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: ϵ_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
$Var(x_i) = \sum_{j=1}^k v_{ij}^2 Var(z_j) + Var(\epsilon_i) = \sum_{j=1}^k v_{ij}^2 + \Psi_i$
Covariance Matrix: $\boldsymbol{\Sigma} = Cov(\mathbf{V}\mathbf{z} + \boldsymbol{\epsilon}) = \mathbf{V}\mathbf{V}^T + \boldsymbol{\Psi}$
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:

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

$b_{rs} = \sum_{j=1}^d x_j^{(r)} x_j^{(s)}$, or matrix form $\mathbf{B} = \mathbf{X}\mathbf{X}^T$
Constraint: $\sum_l^N x_j^{(l)} = 0, \forall j$, define: $T = \sum_{l=1}^N b_{ll}$
Then $\sum_r d_{rs}^2 = T + Nb_{ss}$, $\sum_r \sum_s = d_{rs}^2 = 2NT$
defining:
$d_{*s}^2 = \frac{1}{N}\sum_r d_{rs}^2$, $d_{r*} = \frac{1}{N}\sum_s d_{rs}^2$, $d_{**}^2 = \frac{1}{N^2}\sum_r \sum_s d_{rs}^2$
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 λ_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: $J(\mathbf{W}) = \frac{Tr(\mathbf{W}^T\mathbf{S}_B\mathbf{W})}{Tr(\mathbf{W}^T\mathbf{S}_W\mathbf{W})}$ Corresponds to eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$

7 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 $p(X)$: $\hat{p}(x) = \frac{1}{h}[\frac{\#\{x^{(l)}+h \leq x\} - \#\{x^{(l)} \leq x\}}{N}]$, 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

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

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

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

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

Kernel Estimator Kernel $K(u) = \frac{1}{2\pi}\exp(-\frac{u^2}{2})$,
Estimator:$\hat{p} = \frac{1}{Nh}\sum_{l=1}^{N} K(\frac{x-x^{(l)}}{h})$

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

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

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

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

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

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

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

Regularized cost function balance bias and variance:

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

8 Decision Trees

8.1 Constructing Decision Trees

8.1.1 Basic alogrithm

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.

8.1.2 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 noo
- 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 zer, 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 raeio is derived by taking into account the number and size of of daughter nodes into which an attribute splits the dataset, disregarding any information about the class.

8.2 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.

8.2.1 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 stimate 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 $\hat{P}(C_i|\mathbf{x}, m) = \frac{N_m^i}{N_m} = p_m^j$, pure if $p_m^j = 1$

Entropy $\mathcal{T}_m = -\sum_{i=1}^{K} p_m^j \log_2 p_m^j$, 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 o$[1/2, 1]$

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: $\hat{P}(C_i|\mathbf{x}, m, j) = p^i_{mj} = \frac{N^i_{mj}}{N_{mj}}$,total impurity after split: $\mathcal{T}'_m = -\sum_{i=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log p^i_{mj}$

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 $E'_m = \frac{1}{N_m} \sum_j \sum_l (y^{(l)} - g^2_{mj} b_{mj}(\mathbf{x}^{(l)})$
Best split: split that resuts 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.

9 Linear Model

9.1 Linear Discriminant functions

A discriminant is a function that take an input vector $\mathbf{x}$ and assignes 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 $g_i(\mathbf{x}) = \sum_{j=1}^{k} w_j\phi_{ij}(\mathbf{x})$

9.1.1 Two classes

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

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

9.1.2 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 $r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$

9.1.3 Multi classes

K discriminant:

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

Linear separable:

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

if $\mathbf{x} \in C_i$

Choose C_i if

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

9.1.4 Pairwise Separation

Discriminant function for class i and j:

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

9.2 Logistic Discrimination

9.2.1 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;

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

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

Rearranging terms:

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

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:

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

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

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

$$w_i = w_i + \Delta w_i$$

η 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 $y^{(l)} = p(C_1|\mathbf{x}^{(l)})$:

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

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

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

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

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

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

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

and we get the following update equations:

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

9.2.2 Multiple Classes

Generalization of sigmoid

Take one of the classes C_k, as reference class and assume that:
$\log \frac{p(\mathbf{x}|C_i)}{p(\mathbf{x}|C_K)} = \mathbf{w}_i^T \mathbf{x} + w_{i0}^o$, $i = 1, \ldots, K-1$
Then we have:

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

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

Summing over i we can deduce:

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

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

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

Softmax

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

Learning

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

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

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

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

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

Learning $\Delta\mathbf{w} = \eta\sum_l (r^{(l)} - y^{(l)})y^{(l)}(1-y^{(l)})\mathbf{x}^{(l)}$,
$\Delta w_0 = \eta\sum_l (r^{(l)} - y^{(l)})y^{(l)}(1-y^{(l)})$

$K > 2$ **classes** $\mathbf{r}^{(l)} = \mathbf{y}^{(l)} + \boldsymbol{\epsilon}$,
$\epsilon \sim \mathcal{N}_K(0, \sigma^2\mathbf{I}_K)$, $y_i^{(l)} = \frac{1}{1+\exp\left[-(\mathbf{w}_i^T\mathbf{x}^{(l)}+w_{i0})\right]}$,
Likelihood:
$L(\{\mathbf{w}_i, w_{i0}\}_i|\mathcal{X}) = \prod_l \frac{1}{(2\pi)^{K/2}|\mathbf{\Sigma}|^{1/2}} \exp\left[-\frac{\|\mathbf{r}^{(l)}-\mathbf{y}^{(l)}\|^2}{2\sigma^2}\right]$
Error Func: $E(\{\mathbf{w}_i, w_{i0}\}|\mathcal{X}) = \frac{1}{2}\sum_l \|\mathbf{r}^{(l)} - \mathbf{y}^{(l)}\|^2$
Learning $\Delta\mathbf{w}_i = \eta\sum_l (r_i^{(l)} - y_i^{(l)})y_i^{(l)}(1-y_i^{(l)})\mathbf{x}^{(l)}$,
$\Delta w_0 = \eta\sum_l (r_i^{(l)} - y_i^{(l)})y_i^{(l)}(1-y_i^{(l)})$

9.3 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**

10 Multilayer Perceptrons

10.1 Perceptron

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

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

where x_0 is a special *bias unit* with $x_0 = 1$ and $\mathbf{w} = (x_0, w_1, \ldots, w_d)^T$ 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:

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

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

10.1.1 $K > 2$ 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:

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

10.1.2 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 η 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:

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

10.2 Multilayer Perception

MLP has a hidden layer between the input and output.

Input to hidden:

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

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:

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

10.2.1 MLP for Nonlinear Regression(Multi output)

outputs:

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

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:

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

10.2.2 MLP for NonLinear Multi-class Discrimination

Outputs:

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

,

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

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.

11 Support Vector Machine

11.1 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 $|\mathbf{w}^T\mathbf{x} + w_0| = 1 \Rightarrow$ **canoical 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 $\mathbf{w}^T(x^{(1)} - \mathbf{w}^{(2)}) = 2$, 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

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

Lagrangian

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

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 duel 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:

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

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:
$y^{(l)}(\mathbf{w}^T\mathbf{x}^{(l)} + w_0) \geq 1 - \zeta_l$,
minimize: $C\sum_l \zeta_l + \frac{1}{2}\|\mathbf{w}\|^2$
Lagrangian(Primal):
$L_p = \frac{1}{2}\|\mathbf{w}\|^2 +$
$C\sum_l \zeta_l - \sum_{l=1}^{N} \alpha_l \left[y^{(l)}(\mathbf{w}^T\mathbf{x}^{(l)} + w_0) - 1 + \zeta_l \right] - \sum \mu_l \zeta_l$,
C is the regularization parameter, μ_l is the new Lagrange multipiler to guarantee that $\zeta \geq 0$
Dual: Max $\sum_l \alpha_l - \frac{1}{2}\sum_l \sum_{l'} \alpha_l \alpha_{l'} y^{(l)} y^{(l')} (\mathbf{x}^{(l)})^T \mathbf{x}^{(l')}$
subject to: $\sum_l \alpha_l y^{(l)} = 0$ and $0 \leq \alpha_l \leq C, \forall l$

Kernel Functions Dual Problem:
Max: $\sum_l \alpha_l - \frac{1}{2}\sum_l \sum_{l'} \alpha_l \alpha_{l'} y^{(l)} y^{(l')} \phi(\mathbf{x}^{(l)})^T \phi(\mathbf{x}^{(l')})$
subject to: $\sum_l y^{(l)} = 0$ and $0 \leq \alpha_l \leq C, \forall l$
Kernel: $K(\mathbf{x}^{(l)}, \mathbf{x}^{(l')} = \phi(\mathbf{x}^{(l)})^T \phi(\mathbf{x}^{(l')})$

$\epsilon - InsensitiveLoss$

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

Primal: Max $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_l (\zeta_l^+ + \zeta_l^-)$
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: ζ_l^+ such that $y^{(l)} - (\mathbf{w}^T \mathbf{x}^{(l)} + w_0) > \epsilon$, ζ_l^- such that $(\mathbf{w}^T \mathbf{x}^{(l)} + w_0) - y^{(l)} > \epsilon$

12 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 $\mathcal{X}_1, \ldots, \mathcal{X}_K$ 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: $\mathcal{T}_1 = \mathcal{X}_1^{(1)}, \mathcal{V}_1 = \mathcal{X}_1^{(2)}, \mathcal{T}_2 = \mathcal{X}_1^{(2)}, \mathcal{V}_2 = \mathcal{X}_1^{(1)}, \mathcal{T}_3 = \mathcal{X}_2^{(1)}, \mathcal{V}_3 = \mathcal{X}_2^{(2)}, \mathcal{T}_4 = \mathcal{X}_2^{(2)}, \mathcal{V}_4 = \mathcal{X}_2^{(1)}, \ldots$

Bootstrapping Generates new samples, each of size N, by drawing randomly with replacement. Multiple bootstrap sammples 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 θ, intervale specifies an interval within which θ lines with a certain degree of confidence.

Confidence Interval Define z_α,
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 σ^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 $\mathcal{N}(\mu, \sigma^2)$, σ known, μ 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 μ_0. Accept H_0 with level of significance α if μ_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, α 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 σ^2 is not known, $H_0 : \mu = \mu_0$, $H_1 : \mu \neq \mu_0$, accept at α 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 B ernoulli 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, $p_i, i = 1, \ldots, K$ 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 significane level α.

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

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 constuct 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 μ

$H_0 = \mu = 1$, $H_1 = \mu \neq 0$. 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 α if $\sqrt{K}m/S \in -t_(\alpha/2, K-1, t_{\alpha/2,K-1})$

13 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:

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

, with $w_j \geq 0$ and $\sum_{j=1}^{L} w_j = 1$, $\mathbf{\Phi} = (w_1, \ldots, w_L)^T$ 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: $P(C_i|x) = \sum_{models \mathcal{M}_j} P(C_i|x, \mathcal{M}_j)P(\mathcal{M}_|)$, 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 func-

Training:
For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize
For all base-learners $j = 1, \ldots,$
Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$
Train d_j using $\mathcal{X}_j$
For each (x^t, r^t), calculate
Calculate error rate: $\epsilon_j \leftarrow$
If $\epsilon_j > 1/2$, then $L \leftarrow j-1$;
$\beta_j \leftarrow \epsilon_j/(1-\epsilon_j)$
For each (x^t, r^t), decrease
If $y_j^t = r^t$ $p_{j+1}^t \leftarrow \beta_j p_j^t$ E
Normalize probabilities:
$Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow$
Testing:
Given x, calculate $d_j(x), j = 1,$
Calculate class outputs, $i = 1,$
$y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$

tion of the error of the previous base learner.

13.1 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 $Tr(\mathbf{AB}) = Tr(\mathbf{BA})$, $Tr(\mathbf{ABC}) = Tr(\mathbf{CBA}) = Tr(\mathbf{BCA})$, $|\mathbf{A}^{-1}| = \frac{1}{|\mathbf{A}|}$, $\mathbf{a}^T\mathbf{Aa} = Tr(\mathbf{Aaa}^T)$

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

14 Convolutional Neural Networks

14.1 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.

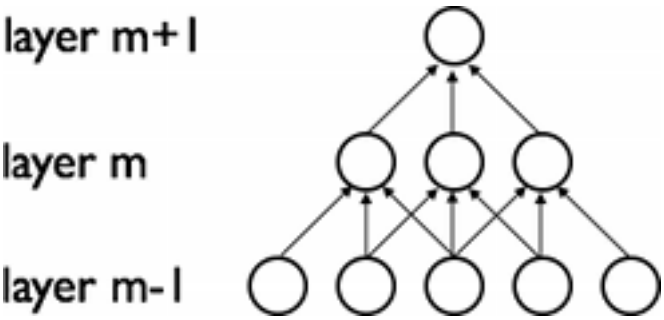
Two basic cell types: Simple cells (S) and complex cells (C).

- 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.

14.2 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 tot he 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.

14.3 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

14.4 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)

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

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

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

14.5 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.

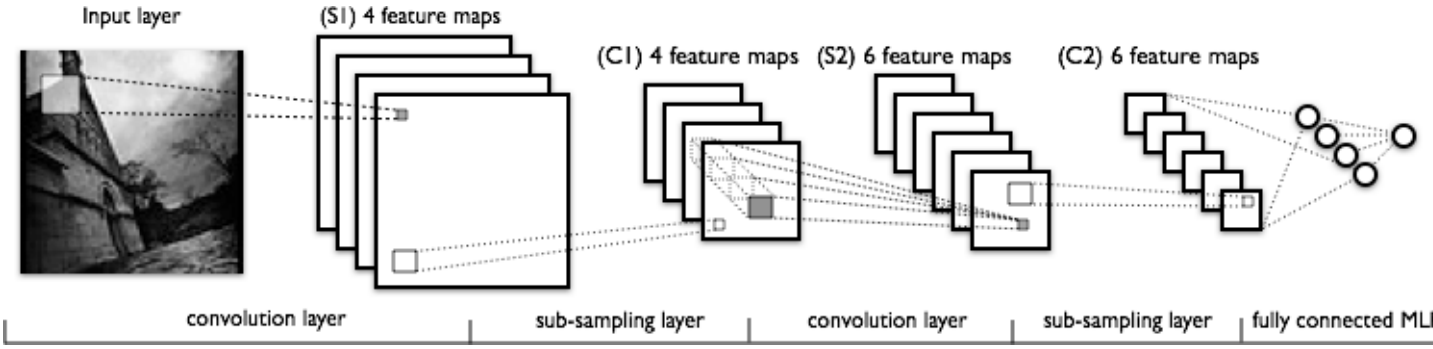
Useful for two reasons:

- 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 2×2 region, 3 out of these 8 possible configurations will produce exactly the same output at the convolutional layer. For max-pooling over a 3×3

14.6 The Full Model: LeNet

Graphical depiction:



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

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

15 Denoising Autoencoders

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

15.1 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 $\mathbf{y} \in [\mathbf{0},\mathbf{1}]^{\mathbf{d}'}$ 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 decorder) 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 $L(\mathbf{x},\mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|$

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.

15.2 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.

16 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.

16.1 Multi-Label Classification

16.1.1 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, μ_n as the disjunction of the labels contained in the node. A training example can be considered annotated with meta-label μ_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 μ_c is among the prediction of $h_{\text{parent}(c)}$
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 hte proposed approach in this case.

Training

1. Assume the existence of a set $D = \{(\mathbf{x}_i, \mathbf{Y}_i) | i = 1 \cdots |D|\}$, 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 eexample 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.

16.1.2 RAKEL

Let $L = \{\lambda_i\}, i = 1 \dots |L|$ be the set of labels. A set $Y \subseteq L$ with $k = |Y|$ is called k-labelset.

16.1.3 MLkNN

Preliminaries

Notations

domain of instances $\mathcal{X}$

Finite set of labels $\mathcal{Y} = \{(x_1, Y_1), (x_2, Y_2), \dots, (x_m, Y_m)\}, (x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y})$

Learning system will produce a real-valued function of the form:

$$f : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}$$

A successful learning system will tend to output larger values for labels in Y_i than those not in Y_i. That is

$$f(x_i, y_1) > f(x_i, y_2), \forall y_1 \in Y_i, \text{and} y_2 \notin Y_i$$

The corresponding multi-label classifier $h(\cdot)$:

$$h(x_i) = \{y | f(x_i, y) > t(x_i), y \in \mathcal{Y}\}$$

Where $t(\cdot)$ is a threshold function which is usually set to be the zero constant.

16.1.4 ML-KNN

Given an instance x and its associated label set $Y \in \mathcal{Y}$

Let $\vec{y_x}$ be the category vector for x:

$$\vec{y_x}(l) = \begin{cases} 1 & l \in Y \\ 0 & \text{otherwise} \end{cases}$$

Let $N(x)$ denote the set of the label sets of these neighbors, a *membership counting vector* can be defined as:

$$\vec{C}(x) = \sum_{a \in N(x)} \vec{y_a}(l), l \in \mathcal{Y}$$

$\vec{C_x}(l)$ counts the number of neighbors of x belonging to the lth class.

- For each test instance t, ML-KNN firstly identifies its KNNs $N(t)$ in the training set.
- $N(t)$ — neighbors of t
- $\vec{C}_t(l)$ — membership counting vector. Number of element in $N(t)$ that has label l.
- H_b^l, $b \in 0, 1$ — t (the testing instance) has label l if $b = 1$, otherwise if $b = 0$
- $\vec{y}_x(l)$ — 1 if instance x has label l
- $E_j^l, j \in \{0, 1, \ldots, K\}$ — Among $N(t)$ exactly j instances have label l.

Based on the membership counting vector $\vec{C}_t$, the category vector $\vec{y}_t$ is determined using the following MAP principle:

$$\vec{y}_t(l) = \arg\max_{b \in \{0,\}} P(H_b^l | E^l_{\vec{C}_t(l)}), l \in \mathcal{Y}$$

Using the Bayesian rule:

$$\begin{aligned}\vec{y}_t(l) &= \arg\max_{b \in \{0,1\}} \frac{P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l)}{P(E^l_{\vec{C}_t(l)} | H_b^l)} \\ &= \arg\max_{b \in \{0,1\}} P(H_b^l) P(E^l_{\vec{C}_t(l)} | H_b^l)\end{aligned}$$

Algorithm

1. For each label $l \in \mathcal{Y}$, calculate the prior:

$$P(H_1^l) = \frac{(s + \sum_{i=1}^m y_{x_i}(l))}{s \times 2 + m}$$

The probability that any instance has label l. s is smoothing parameter. It is basically $\frac{\#\text{instance has } l}{\#\text{total training instances}}$.

$$P(H_0^l) = 1 - P(H_1^l)$$

2. Identify the KNN $N(x_i), i \in \{1, 2, \ldots, m\}$ (m testing instances)
3. for each label $l \in \mathcal{Y}$, do the following:

   a) For $j \in \{0, 1, \ldots, K\}$ (the possible count of labels) initialize the counting $c[j] = 0$ and $c'[j] = 0$

b) For each of the training instances $i \in \{1, 2, \dots m\}$, the number of label l in the neighbors of x_i:

$$\delta = C_{x_i}^{(} l) \leftarrow \sum_{a \in N(x_i) \vec{y}_a(l)}$$

If $\vec{y}_{x_i}(l) = 1$, that is, if instance x_i has label l, then $c[\delta] \leftarrow c[\delta] + 1$, else $c'[\delta] \leftarrow c'[\delta] + 1$.

So $c[\delta]$ is the cases that the training instance x_i has label l and among its neighbors, there are exactly δ instances also have label l.

c) Calculate the posterior probabilities $P(E_j^l | H_b^l)$, probability that produce E_j^l (j label l in the neighbor) given the instance H_b^l (has or does not have label l) The probability that l has j instances

$$P(E_j^l | H_1^l) = \frac{s + c[j]}{s \times K + 1 + \sum_{p=0}^{k} c[p]}$$

In words, it is (ignoring the smoothing terms):

$$P(j \text{ neighbors with label } l) = \frac{\#\text{num instances with } l \text{ has } j \text{ neighbors labelled } l \text{ neighbors}}{\#\text{total number of } l \text{ instances whose neighbors has label } l}$$

17 Appendix

17.1 Gradients and Directional Derivatives

Definition 17.1.1 (Gradient) *At any point (x, y) where the first partial derivatives of the function $f(x, y)$ exists, we define the* gradient vector $\nabla f(x, y) = \mathbf{grad} f(x, y)$ *by*

$$\nabla f(x, y) = \mathbf{grad} f(x, y) = f_1(x, y)\mathbf{i} + f_2(x, y)\mathbf{j} \tag{17.1}$$

The symbol ∇ is a vector differential operator

$$\nabla = \mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} \tag{17.2}$$

Theorem 17.1.1 *If $f(x, y)$ is differentiable at the point (a, b) and $\nabla f(a, b) \neq 0$, then $\nabla f(a, b)$ is a normal vector to the level curve of f 5that passes through (a, b)*

Definition 17.1.2 (Directional Derivative) *Let $\mathbf{u} = u\mathbf{i} + v\mathbf{j}$ be a unit vector. The* ***directional derivative*** *of $f(x, y)$ at (a, b) in the direction of $\mathbf{u}$ is the rate of change of $f(x, y)$ with respect to distance measured at (a, b) along a ray in the direction of $\mathbf{u}$ in the xy-plane. Given by*

$$D_{\mathbf{u}} f(a, b) = \lim_{h \to 0+} \frac{f(a + hu, b + hv) - f(a, b)}{h} \tag{17.3}$$

$$= D_{\mathbf{u}} f(a, b) = \frac{d}{dt} f(a + tu, b + tv)\Big|_{t=0} \tag{17.4}$$

Theorem 17.1.2

$$D_{\mathbf{u}} f(a, b) = \mathbf{u} \cdot \nabla f(a, b) \tag{17.5}$$

Definition 17.1.3 (Chain Rule)

$$\frac{dz}{dt} = \frac{\partial z}{\partial t}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt} \tag{17.6}$$