Chap 1 Overview

Types of problems in ML
- ↗ supervised learning
- → unsupervised learning
- ↘ reenforcement learning

Description of supervised Learning

Task `Learn' a mapping $f: X \to Y$: $y = f(x)$

- input: feature $\in \mathbb{R}^D$
- output: label: $y \in \{1, 2, .. C\}$ ↗ Classification
- or $y \in \mathbb{R} \Rightarrow$ Regression

given a set of input-output pairs $D = \{(x_n, y_n)_{n=1..N}\}$ `Experience' `Training set'

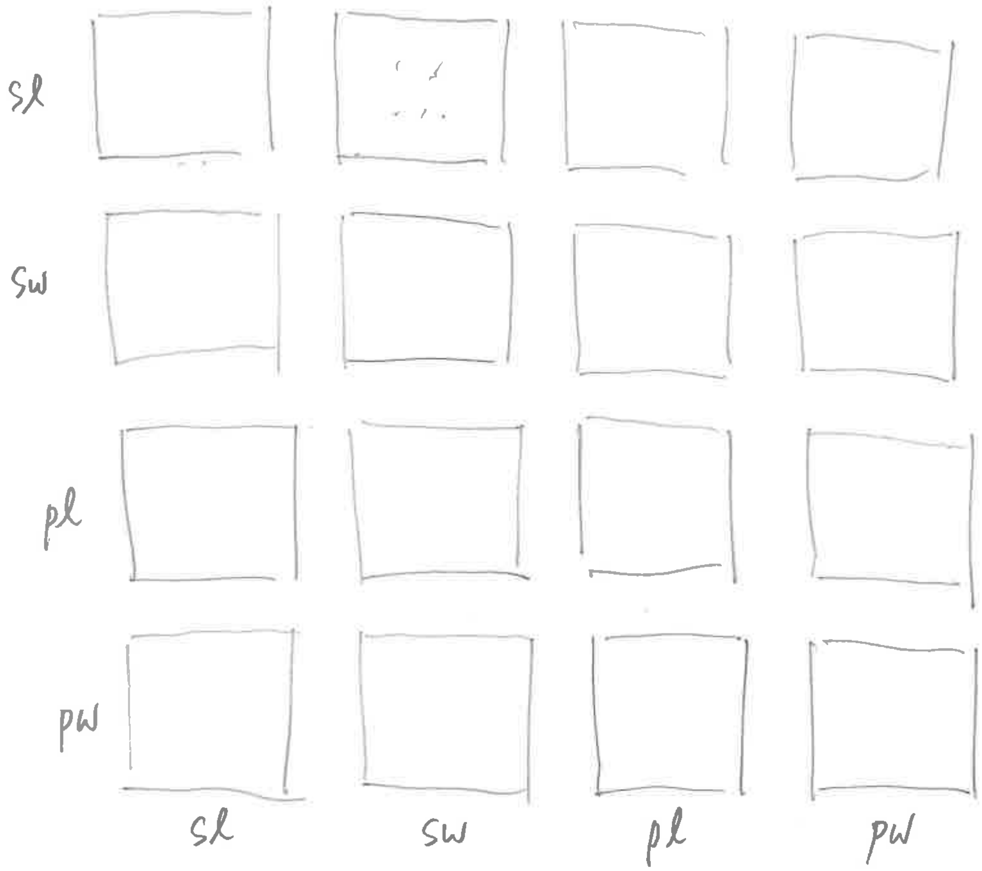
Exaple: Iris flower data

$\underline{x} = (sw, sl, pw, pl) \in \mathbb{R}^4$

sw = sepal width
sl = sepal length
pw = petal width
pl = petal length

Botanists found out that with these dimensions it can be determined whether a pusticular plant is one of the three species 1 = Setosa, 2 = Versicolor. 3 = Virginica. They also determined the training set of the measurements of n diffrent plants with their classification into a species. Thus we have an $N \times 4$ matrix and a column vector where the n-th row contain the data for the n-th plant in the training set.

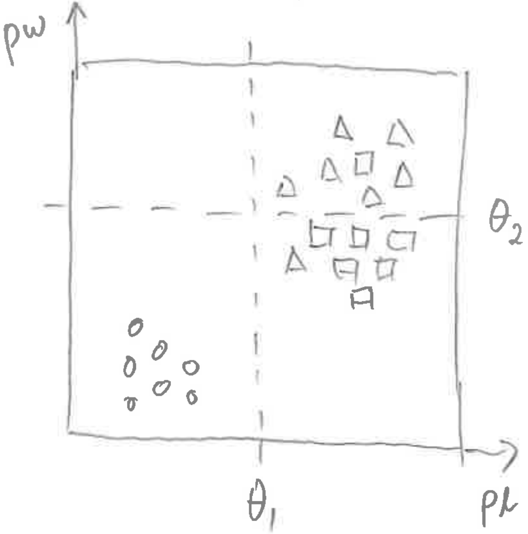
Since $D = 4$ it is hard to visualize the data, thus we can resort to a pairwise scatter plot:



o = 1
□ = 2
△ = 3

The diagonals display the marginal distributions for each species. Only plots below the diagonal display unique information.

Consider the pw-pl plot in more detail:



this suggest the following decision rule

```
def f(pl, pw, $\theta_1$, $\theta_2$):
 if pl < $\theta_1$:
     return 1
 else:
     if pw < $\theta_2$:
        return 2
     else:
        return 3
```

This is an example of a possible 'learned' mapping $y = f(x, \theta)$, albeit a very crude one, as it does not include any sl, wl dependence.

Description of Unsupervised Learning

Algorithms learn patterns from unlabelled data.

Example: Iris set. The training set consists of the dimensions (sl_n, sw_n, pl_n, pw_n)

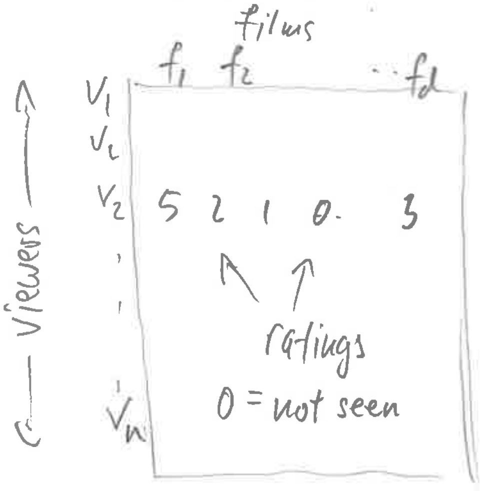
the task is to determine: • how many species are there, i.e. $C=?$

• Which specimens fall into which categories?

this is an example of clustering.

We could also ask the question: are there fewer factors that determine the species? This is an exaple of dimension reduction.

Example Netfix Prize



Very large sparse matrix

goal: make recommendations to viewers about movies they have not seen.

§2.1 . Intro Probability

Example roll a die $\Rightarrow$ get a number $\in \{1,2,3,4,5,6\}$
if the die is fair, then any number comes up with the same frequency (= probability) if the die is uneven, then the probabilities are different.

Terminology
1.) X = sample space
2.) $\mathcal{F}$ = Set of events
3.) $P: \mathcal{F} \to [0,1]$ probability function
} probability triple

Exmple (die) $X = \{1,2,..6\}$
exmples of events $\{1\}$ $\{2,4,6\}$ $\{1,2,3\}$ etc
$p(\{k\}) = \frac{1}{6}$ $p(\{2,4,6\}) = \frac{1}{2}$
↑ means: Probability of an even number is $\frac{1}{2}$

Properties of $\mathcal{F}$ (= sigma algebra):
a) $X \in \mathcal{F}$
b.) $A \in \mathcal{F} \Rightarrow X \setminus A \in \mathcal{F}$
c.) $A, B \in \mathcal{F} \Rightarrow A \cup B \in \mathcal{F}$ (this implies $A \cap B \in \mathcal{F}$ also)
also, countable unions are in $\mathcal{F}$.

Properties of P:
- if A and B are disjoint then $P(A \cup B) = P(A) + P(B)$
- $P(X) = 1$

We say two events are <u>independent</u> if $P(A \cap B) = p(A) \cdot p(B)$

Important: not all events are independent

Example: $A = \{2,4,6\}$ "even" $B = \{1,2,3,4,5\}$ "not 6"

$A \cap B = \{2,4\}$

$p(A) = \frac{1}{2}$ $p(B) = \frac{5}{6}$ $p(A \cap B) = \frac{1}{3} \neq \frac{1}{2} \cdot \frac{5}{6}$

Conditional Probability

$p(A|B)$ probability of event A if event B already happened

Example: $A =$ even , $B =$ not 6

$$p(A|B) = \frac{2}{5}$$

← 2 even possibilities <6

← 5 possibilities for not 6

product rule of probability

$$p(A \cap B) = p(B) \cdot p(A|B)$$

↑ both events happen

↑ B must happen

↑ A must happen knowing B happened already

Example $\frac{1}{3} = \frac{5}{6} \cdot \frac{2}{5}$ ✓

§2.2 Random Variables

recall: probability triple $(\mathcal{X}, \mathcal{F}, \Pr)$

A random variable is a mapping $\mathcal{X} \to \mathbb{R}$ (in general a measure space)

Discrete random variable

Ex-ple Roll a die $\mathcal{X} = \{$ ⚀ ⚁ … ⚅ $\}$

⚀ ↓ $X=1$ ⚅ ↓ $X=6$

every event x has probability $p(x) = \Pr(X=x)$ "probability mass fcn"

obviously $\sum_{x \in \mathcal{X}} p(x) = 1$

Continuous random variables

Ex-ple X = max temperature in Chennai on Jan 19

$\Pr(X=28)$ is not a good way of expressing what we want to say.

Instead, use the cumulative distribution function

$$P(x) = \Pr(X \leq x)$$

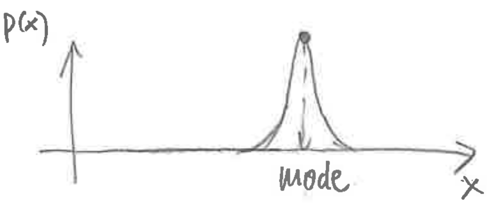
Then the probability that the temperature is between 27.5 and 28.5 is

$$\Pr(27.5 \leq X \leq 28.5) = P(28.5) - P(27.5)$$

We can also do smaller intervals and scale to size => Probability density fcn

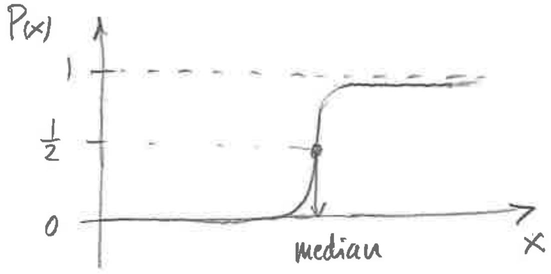
$$p(x) = P'(x) = \lim_{h \to 0} \frac{P(x+h) - P(x)}{h}$$

Example: Temp. of Chennai on Jan 19



prob. density fcn

mode may not be unique



Cumulative prob. density fcn

(§2.4 pa

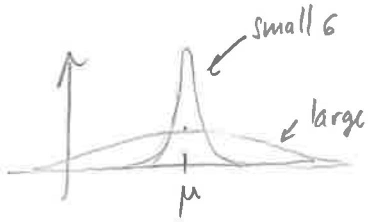
Expected Value (=mean)

- discrete r.v. : $\mathbb{E}(X) = \sum_{x \in X} x\, p(x)$

  ex: die $E(X) = \frac{1}{6}(1+2+3+4+5+6) = 3.5$

- cont r.v. $\mathbb{E}(X) = \int_{-\infty}^{\infty} x\, p(x)\, dx$

Example Normal distribution (Gaussian)



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

$$\mathbb{E}(\mathcal{N}) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \int_{-\infty}^{\infty} x \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx$$

$$= \frac{1}{\sqrt{2\pi}\sigma} \cdot \left[\mu \cdot \int_{-\infty}^{\infty} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx + \int_{-\infty}^{\infty} (x-\mu) \underbrace{\exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right) dx}_{=0}\right]$$

$t = \frac{x-\mu}{\sqrt{2}\sigma}$ $dt = \frac{1}{\sqrt{2}\sigma}$

$$= \frac{\mu}{\sqrt{\pi}} \underbrace{\int_{-\infty}^{\infty} \exp(-t^2)\, dt}_{=\sqrt{\pi}}$$

$$= \mu$$

Variance

$$\mathbb{V}(X) = \mathbb{E}\big((X-\overset{\mu = \mathbb{E}(X)}{\mu})^2\big) = \sigma^2$$ Expected value of how far X deviates from μ

discrete $\mathbb{V}(x) = \sum_{x \in X} (x-\mu)^2 p(x)$

contin. $\mathbb{V}(x) = \int_{-\infty}^{\infty} (x-\mu)^2 p(x)\, dx$

useful formula:

$$\sigma^2 = \mathbb{E}((X-\mu)^2) = \mathbb{E}(X^2 - 2\mu X + \mu^2) = \mathbb{E}(X^2) - 2\mu \overset{=\mu}{\widetilde{\mathbb{E}(X)}} + 1 \cdot \mu^2$$

$$\Rightarrow \mathbb{E}(X^2) = \sigma^2 + \mu^2$$

check at home $\mathbb{V}(\mathcal{N}) = \sigma^2$

Terminology $\sigma = \sqrt{\mathbb{V}(X)}$ is the standard deviation.

SL05 p1

2.3 Bayes' Rule

recall product rule: $p(H \cap Y) = p(Y) \cdot p(H|Y)$
H, Y are events $\quad\quad = p(H) \cdot p(Y|H)$

$$\left.\begin{aligned} p(H \cap Y) &= p(Y)\cdot p(H|Y) \\ &= p(H)\cdot p(Y|H) \end{aligned}\right\} \Rightarrow \boxed{p(H|Y) = \frac{p(H)\cdot p(Y|H)}{p(Y)}}$$

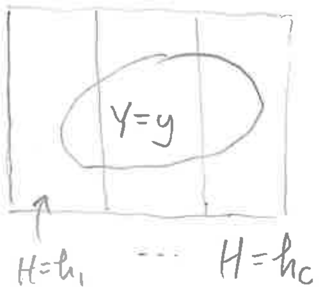
Bayes' rule

Suppose that H and Y are random variables and $\mathcal{H} = \{h_1, \dots, h_C\}$.

Since $H = h_k$ are disjoint events

$$p(Y=y) = p\left(\bigcup_{k=1}^{C} \{Y=y\} \cap \{H=h_k\}\right)$$

$$= \sum_{k=1}^{C} p(H=h_k) \cdot p(Y=y \mid H=h_k)$$



Substitution into Bayes' rule gives:

$$p(H=h \mid Y=y) = \frac{p(H=h)\, p(Y=y \mid H=h)}{\sum_{k=1}^{C} p(H=h_k) \cdot p(Y=y \mid H=h_k)}$$

We use this formula in this context

$$\left.\begin{array}{l} Y \text{ is an observed quantity} \\ H \text{ is a hidden quantity} \end{array}\right\} \begin{array}{l} \text{determine the likelihood that } H=h_k \text{ if} \\ \text{we know that } Y \text{ has happened.} \end{array}$$

$$\left.\begin{array}{l} p(H) \text{ is the prior distribution} \\ p(Y \mid H) \text{ are called likelihoods} \end{array}\right\} \text{usually known a-priori}$$

Example: covid testing.

- prior: H covid status of a random person in the general population
  $p(H=n) = 0.9$, $p(H=p) = 0.1$ (covid positive or negative)
- observed: Y result of a covid test. $Y \in \{p, n\}$
- likelihoods: The covid test has uncertainties. There can be false positives or false negatives

$$\left.\begin{array}{c} \text{test result} = Y \\ \text{actual} = H \quad \begin{array}{c|cc} & n & p \\ \hline n & 0.975 & 0.025 \\ p & 0.025 & 0.975 \end{array} \end{array}\right\} = \begin{bmatrix} p(Y=n \mid H=n) & p(Y=p \mid H=n) \\ p(Y=n \mid H=p) & p(Y=p \mid H=p) \end{bmatrix}$$

Do the covid test for a random person. It comes out positive.
Find the probability that this person is actually positive:

First, we need $P(Y=p) = p(H=p)\cdot p(Y=p|H=p) + p(H=n)\cdot p(Y=p|H=n)$

$$= 0.1 \cdot 0.875 + 0.9 \cdot 0.025 = 0.11$$

It asks for $p(H=p|Y=p) = \dfrac{p(H=p)\cdot p(Y=p|H=p)}{p(Y)} = \dfrac{0.1 \cdot 0.875}{0.11} \approx 0.795$

§2.4 Bernoulli / Binomial Distribution

- Tossing a coin:

$Y=1$ head probability $= \theta$ $\quad 0 \le \theta \le 1$

$Y=0$ tail " $= 1-\theta$

$$\mathrm{Ber}(y|\theta) = \theta^y (1-\theta)^{1-y} \qquad y \in \{0,1\}$$

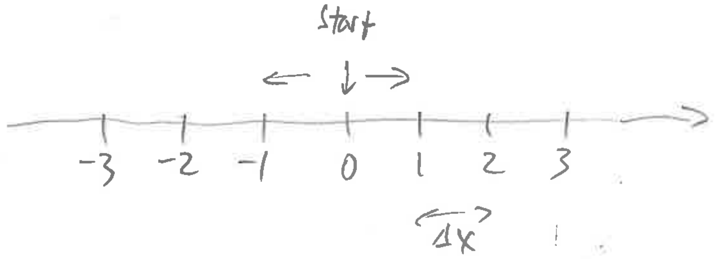
- Tossing a coin N times $\quad X = \{0, 1, \dots, N\}$ $\quad X = s$ means "s times head"

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

$\uparrow$ # of possibilities to get s heads in N tosses

$$\binom{N}{s} = \frac{N!}{(N-s)!\,s!}$$

Random Walk A random walker goes with prob. $\theta = \frac{1}{2}$ either left or right.



probability that the walker is at $x = s$ after N moves follows a binomial distribution.

If we let $\Delta X \to 0$ and $N \to \infty$ we obtain a normal distribution

(§2.6)

More properties of the Normal (Gauss) Distribution

Recall $\mathcal{N}(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) =: p(x)$ probability density fcn

Find the cumulative density. For that we need the error function

$$\operatorname{erf}(x) := \frac{2}{\sqrt{\pi}}\int_0^x \exp(-t^2)\,dt \qquad \text{also} \quad \operatorname{erf}(\infty) = \frac{2}{\sqrt{\pi}}\int_0^\infty \exp(-t^2)\,dt = 1$$

Now integrate $p(x)$

$$P(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\left(\frac{t-\mu}{\sqrt{2}\,\sigma}\right)^2\right) \cdot dt \;\underset{\uparrow}{=}\; \frac{1}{\sqrt{\pi}}\int_{-\infty}^{z} \exp(-u^2)\,du$$

$$u = \frac{t-\mu}{\sqrt{2}\,\sigma} \qquad z := \frac{x-\mu}{\sqrt{2}\,\sigma}$$

$$du = \frac{1}{\sqrt{2}\,\sigma}\,dt$$

$$= \frac{1}{2}\left[\frac{2}{\sqrt{\pi}}\int_{-\infty}^{0} \exp(-u^2)\,du + \frac{2}{\sqrt{\pi}}\int_0^z \exp(-u^2)\,du\right]$$

$$= \frac{1}{2}\left[1 + \operatorname{erf}(z)\right]$$

conclusion: $P(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\,\sigma}\right)\right]$ is the cdf of $\mathcal{N}(x|\mu,\sigma)$

We know that $\int_{-\infty}^{\infty} \mathcal{N}(x|\sigma,\mu)\,dx = 1$ for any $\sigma > 0$

also for $x \neq \mu$: $\lim_{\sigma\to 0} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) = 0$

this implies that

$$\lim_{\sigma\to 0} \mathcal{N}(x|\sigma,\mu) = \delta(x-\mu)$$

conclusion: In the limit as $\sigma \to 0$

$$\Pr(a \le X \le b) = \begin{cases} 1 & \text{if } \mu \in [a,b] \\ 0 & \text{if } \mu \notin [a,b] \end{cases}$$

Chap 3 Multivariate Models

two random variables may related.

Example: X = max temperature for a given day of the year
Y = total rainfall for the same day

Since it tends to be cooler when it rains we say they have a negative covariance. This is made precice in the following definition

Covariance: $\text{Cov}(X,Y) = \mathbb{E}\left[(X-\mathbb{E}(X))\cdot(Y-\mathbb{E}(Y))\right]$

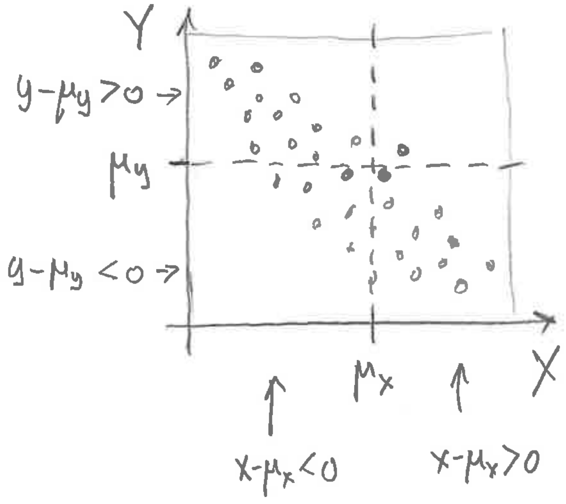
discrete r.v: $\text{Cov}(X,Y) = \sum_{\substack{x\in X\\ y\in Y}} (x-\mu_x)(y-\mu_y)\cdot p(x,y)$

cont. r.v: $\text{Cov}(X,Y) = \int_{\mathbb{R}}\int_{\mathbb{R}} (x-\mu_x)\cdot(x-\mu_y)\cdot p(x,y)\,dy\,dx$

here, $\mu_x = \mathbb{E}(X)$
$\mu_y = \mathbb{E}(Y)$

Correlation $\text{corr}(X,Y) = \dfrac{\text{Cov}(X,Y)}{\sqrt{\mathbb{V}(X)\,\mathbb{V}(Y)}}$

Scatterplot for Rainfall / Temperature



most data points are in regions where the product $(x-\mu_x)\cdot(y-\mu_y)$ is negative contributing to a negavive covariance.

Note $\text{Cov}(X,X) = \mathbb{V}(X)$

§ 3.1 Properties of the covariance

- Suppose we have D random variables $(D \geq 1)$. Then $\underline{x} = [X_1 \dots X_D]$,

$$\mathrm{Cov}(\underline{x}) = \begin{bmatrix} \mathrm{Cov}(X_1, X_1) & \dots & \mathrm{Cov}(X_1, X_n) \\ \vdots & & \\ \mathrm{Cov}(X_n, X_1) & \dots & \mathrm{Cov}(X_n, X_n) \end{bmatrix} \in \mathbb{R}^{D \times D}$$

symmetric matrix denoted by Σ

- independent => uncorrelated

independent means $p(x,y) = p(x) \cdot p(y)$. Then

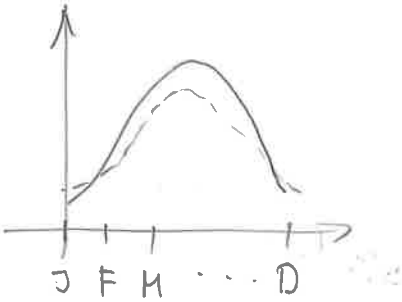
$$\mathrm{Cor}(X,Y) = \int_{\mathbb{R}}\int_{\mathbb{R}} (x-\mu_x)\cdot(y-\mu_y)\, p(x)\cdot p(y)\, dy\, dx = \int_{\mathbb{R}} (x-\mu_x)\, p(x)\, dx \cdot \int_{\mathbb{R}} (y-\mu_y)\, p(y)\, dy$$

$$= \left(\mathbb{E}(X) - \mu_x\right)\left(\mathbb{E}(Y) - \mu_y\right) = 0$$

Note: uncorreladed does not imply independence:

- Correlation does not imply causation:

Exaple Ice cream Sales and motor cycle accidents are correlated in the US (both go up in the summer) but there is no causal relation.



§ 3.2 Multivariable Gauss Distribution

$\Sigma \in \mathbb{R}^{D \times D}$ is a $\overset{\text{symmetric}}{\text{positive}}$ definite matrix

- Recall some facts from linear algebra (see § 7.4)

We can diagonalize Σ, i.e. we can write

$\Sigma = U \Lambda U^T$ where $U = [\underline{u}_1, \dots, \underline{u}_D] \in \mathbb{R}^{D \times D}$ is an orthonormal matrix

$\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_D \end{bmatrix} \in \mathbb{R}^{D \times D}$ is a diagonal matrix

further, $\underline{u}_k, \lambda_k$ are the eigenvectors, values and $\lambda_k > 0$

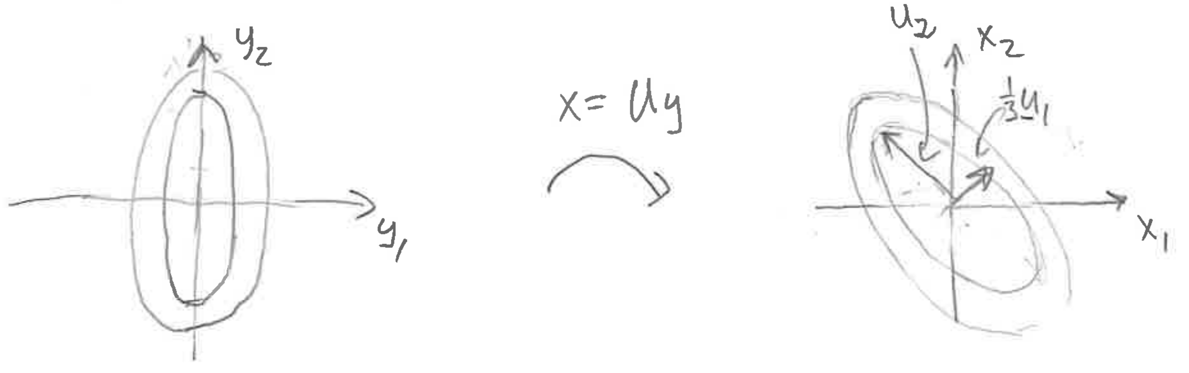
also $\Sigma = \sum_{k=1}^{D} \lambda_k \underline{u}_k \underline{u}_k^T$ (Σ is a sum of outer products, don't confuse the matrix Σ with the summation sign)

Quadratic forms: $f(\underline{x}) = x^T \Sigma x$ is called a quadratic form. Diagonalize:

$$f(x) = x^T U \Lambda U^T x \underset{\substack{\uparrow \\ \text{change variables } y = U^T x \Rightarrow y^T = x^T U}}{=} y^T \Lambda y = \sum_{k=1}^{D} \lambda_k y_k^2$$

Example $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ $U = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ +1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$ $\Lambda = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$

$$f(x) = \lambda_1 y_1^2 + \lambda_2 y_2^2 = 3y_1^2 + y_2^2$$



contour lines of $f(x)$

• The multivariable Gauss (normal) distribution is defined as follows

$\underline{y} \in \mathbb{R}^D$, $\underline{\mu} \in \mathbb{R}^D$ center ; $\Sigma \in \mathbb{R}^{D\times D}$ Covariance matrix SPD

$$\mathcal{N}(y \mid \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \cdot \exp\left[-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(y-\mu)\right]$$

where

$$|\Sigma|^{1/2} := (\lambda_1 \cdot .. \cdot \lambda_D)^{1/2}$$

The normalization constant is chosen such that $\int_{\mathbb{R}^D} \mathcal{N}(y \mid \mu, \Sigma)\, dy = 1$.

Check: $$\int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(\underline{y}-\underline{\mu})\right]\, dy \underset{\substack{\nearrow \\ \underline{z} = \underline{y}-\underline{\mu} \\ d\underline{z} = d\underline{y}}}{=} \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}\underline{z}^T \Sigma^{-1} \underline{z}\right) dz$$

$$= \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2} z^T U \Lambda^{-1} U^T z\right) dz \underset{\substack{\nearrow \\ x = U^T z \\ dx = |\det(U)| \cdot dz = dz}}{=} \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2} x^T \Lambda^{-1} x\right) dx$$

$$= \int_{\mathbb{R}} \exp\left(-\frac{1}{2\lambda_1} x_1^2\right) dx_1 \cdot \int_{\mathbb{R}} \exp\left(-\frac{1}{2\lambda_2} x_2^2\right) dx_2 \cdot .. \cdot \int_{\mathbb{R}} \exp\left(-\frac{1}{2\lambda_D} x_D^2\right) dx_D$$

$$\underset{\substack{\nearrow \\ \text{use}}}{=} \sqrt{2\pi\lambda_1} \cdot .. \cdot \sqrt{2\pi\lambda_D} = (2\pi)^{D/2} \cdot |\Sigma|^{1/2}$$

use

$$\int_{\mathbb{R}} \exp(-ax^2)\, dx = \frac{\sqrt{\pi}}{\sqrt{a}}$$

Using a similar argument, we can show that

$$\operatorname{Cov}(\underline{y}) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \cdot \int_{\mathbb{R}^D} \underbrace{(\underline{y}-\underline{\mu})(\underline{y}-\underline{\mu})^T}_{\substack{\text{outer product} \Rightarrow \text{matrix} \\ \text{integral is over each component of the matrix.}}} \exp\left[-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(\underline{y}-\underline{\mu})\right] dy = \Sigma$$

§ 4 Statistics

probability vs. statistics:

probability: random variable density function + depend on parameters θ $\Rightarrow$ likelihood that a certain event happens

statistics: random variables density function + data $\Rightarrow$ find approximations for parameters $\hat{\theta}$

§ 4.2 Maximum Likelihood Estimations (MLE)

Example N coin tosses, got $D = \{y_1, y_2, .. y_n\}$ $y_k \in \{\overset{\text{tail}}{0}, \overset{\text{head}}{1}\}$

find an estimate $\hat{\theta}$ for the likelihood θ of head.

Coin tosses are independent and identically distributed (iid)

let $\mathcal{L}(\theta)$ = likelihood of getting $\mathcal{D}$

$$= \prod_{n=1}^{N} \theta^{y_n}(1-\theta)^{1-y_n}$$

maximize $\mathcal{L}(\theta) \Leftrightarrow$ minimize $NLL(\theta) = -\ln \mathcal{L}(\theta)$ (neg. log. likelihood)

$$NLL(\theta) = -\sum_{n=1}^{N}\left[y_n \cdot \ln(\theta) + (1-y_n)\ln(1-\theta)\right]$$

$$= -\left(\sum_{n:y_n=1} 1\right)\cdot \ln(\theta) - \left(\sum_{n:y_n=0} 1\right)\cdot \ln(1-\theta)$$

s = #heads

$$= -s\cdot \ln(\theta) - (N-s)\ln(1-\theta)$$

$$\frac{d}{d\theta} NLL(\theta) = 0 \Rightarrow -\frac{s}{\theta} + \frac{N-s}{1-\theta} = 0 \Rightarrow \frac{N-s}{1-\theta} = \frac{s}{\theta} \Rightarrow \frac{1-\theta}{N-s} = \frac{\theta}{s}$$

$$\Rightarrow \theta\left(\underbrace{\frac{1}{s} + \frac{1}{N-s}}_{\frac{N\cancel{-s}\cancel{+s}}{s(N-s)}}\right) = \frac{1}{N-s} \Rightarrow \hat{\theta} = \frac{s(N-s)}{N}\,\frac{1}{N-s} = \frac{s}{N}$$

$\hat{\theta} = \frac{s}{N}$ (not so surprising)

This process is called MLE

Example: Single-variable Gauss Distribution

given $D = \{y_1, \dots, y_N\}$

and $p(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{1}{2}\frac{(y-\mu)^2}{\sigma^2}\right)$

find σ, μ that best fit the data, using MLE

$$\mathcal{L}(\mu,\sigma) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(y_n-\mu)^2}{\sigma^2}\right)$$

$$NLL(\mu,\sigma) = -\frac{N}{2}\cdot \ln\left(\frac{1}{2\pi\sigma^2}\right) + \frac{1}{2}\sum_{n=1}^{N}\left(\frac{y_n-\mu}{\sigma}\right)^2$$

$$= \frac{N}{2}\ln(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n-\mu)^2$$

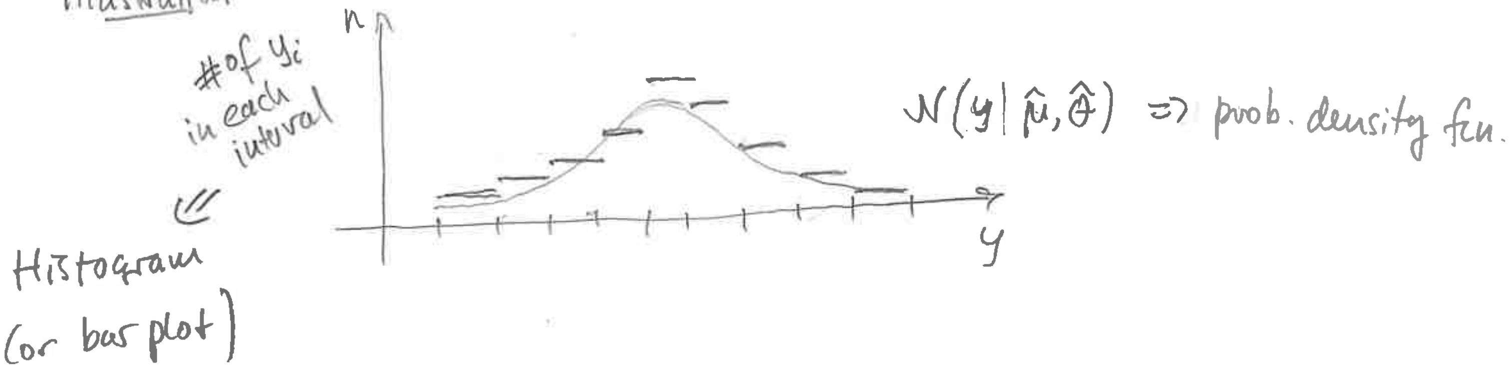
$$\frac{\partial}{\partial\mu} NLL(\mu,\sigma) = \frac{-2}{2\sigma^2}\sum_{n=1}^{N}(y_n-\mu) = 0 \Rightarrow \sum_{n=1}^{N} y_n = N\cdot\mu \Rightarrow \hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} y_n$$

$$\frac{\partial}{\partial\sigma^2} NLL(\mu,\sigma) = \frac{N}{2}\cdot\frac{1}{\sigma^2} - \frac{1}{2\sigma^4}\sum_{n=1}^{N}(y_n-\hat{\mu})^2 = 0$$

$$\Rightarrow \frac{N}{\cancel{2}} - \frac{1}{\cancel{2}\sigma^2}\sum_{n=1}^{N}(y_n-\hat{\mu})^2 = 0 \Rightarrow \hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n-\hat{\mu})^2$$

$\hat{\mu}$ is the average, $\hat{\sigma}$ is the variance of D.

Illustration



Exaple: multi-variable Gauss Distribution

given $D=\{\underline{y}_1,\dots,\underline{y}_N\}$ $\quad \underline{y}_k \in \mathbb{R}^D$

and $p(\underline{y}\,|\,\underline{\mu}\,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}}\cdot \exp\left(-\frac{1}{2}(\underline{y}-\underline{\mu})\Sigma^{-1}(\underline{y}-\underline{\mu})\right)$

then $NLL(\mu,\Sigma) = \frac{N}{2}\log|\Lambda| - \frac{1}{2}\sum_{n=1}^{N}(y_n-\underline{\mu})^T\Lambda(y_n-\underline{\mu}) \qquad \Lambda=\Sigma^{-1}$

find μ: let $z=y_n-\mu$, then

$$\frac{\partial}{\partial\mu_i}\left(z^T\Lambda z\right) = \frac{\partial}{\partial z_i}\left(z^T\Lambda z\right)\frac{\partial z_i}{\partial \mu_i} = \frac{\partial}{\partial z_i}\sum_{k,l}\overset{\text{coeffs of }\Lambda}{\lambda_{kl}}\cdot z_k z_l\cdot(-1)$$

$$= -\sum_{k,l}\lambda_{kl}\frac{\partial}{\partial z_i}\left(z_k z_l\right) = -\sum_{k,l}\lambda_{kl}\left(\delta_{ik}z_l+\delta_{il}z_k\right)$$

$$= -\sum_{l}\lambda_{il}z_l - \sum_k \lambda_{ki}z_k = -\left[\Lambda z+\Lambda^T z\right]_i$$

$$\Rightarrow \nabla_\mu NLL(\mu,\Sigma) = +\left(\Lambda+\Lambda^T\right)\cdot\sum_{n=1}^{N}(\underline{y}_n-\underline{\mu})$$

minimum occurs when $\nabla_\mu(\dots)=0 \underset{\overset{\uparrow}{\Lambda+\Lambda^T\text{ is SPD}}}{\Rightarrow} \sum_{n=1}^{N}(\underline{y}_n-\underline{\mu})=0 \Rightarrow \boxed{\hat{\mu}=\frac{1}{N}\sum_{n=1}^{N}y_n}$

A similar, albeit more complicated computation shows that

$$\hat{\Sigma} = \frac{1}{N}\sum_{n=1}^{N}\left(\underline{y}_n-\hat{\underline{\mu}}\right)\cdot\left(y_n-\hat{\mu}\right)^T$$

see, § 4.2.6.2.

4.5 Regularization

Fundamental problem of MLE: We may not have enough data to reliably predict the parameters. This gets more pronounced as $\#\theta$ grows large $\Rightarrow$ overfitting (i.e., too many parameters).

But even if $\#\theta = 1$ this can be a problem. E.g. in the coin-toss excple, if we toss the coin three times with result `head' then the MLE estimate is $\hat{\theta} = 1$, while it is quite possible to get three head with a fair coin $\theta = \frac{1}{2}$.

The idea of regularization is to add a term that penalizes extreme values of θ, i.e., instead of $NLL(\theta)$ minimize

$$NLL(\theta, \lambda) = NLL(\theta) + \lambda C(\theta)$$

where λ and C depend on the situation. For the coin-toss one could add

$$NLL(\theta, \lambda) = -\sum_{n=1}^{N} y_n \ln(\theta) + (1-y_n)\ln(1-\theta) + \lambda[\ln\theta + \ln(1-\theta)]$$

same calculation as before leads to

$$\hat{\theta} = \frac{S+\lambda}{N+2\lambda}$$

if $S = N = 3$ we get $\hat{\theta} = \frac{3+\lambda}{3+2\lambda}$

For the multivariable Gaussian one can use

$$\hat{\Sigma}_\lambda = \lambda\hat{\Sigma} + (1-\lambda)\,\mathrm{diag}\,\hat{\Sigma}$$

This reduces the off-diagonal entries, called `Shrinkage Estimate'

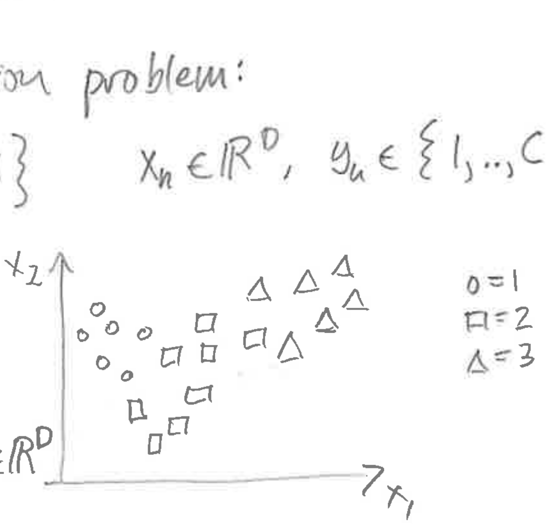
Chap 9 Gaussian Discriminant Analysis

We now turn to the classification problem:

given data $D = \{(x_n, y_n) : n = 1..N\}$ $\quad x_n \in \mathbb{R}^D, \; y_n \in \{1,..,C\}$

find $f : \mathbb{R}^D \to \{1,..,C\}$

that gives the most likely class at the (unknown) point $x \in \mathbb{R}^D$



to that end, we need the conditional probability $p(Y=c \mid X=x)$. Here, Y and X are random variables. Think of the iris-set. The data come from a random selection of plants. If we know $p(Y=c \mid X=x)$ then $f(x) = \max\limits_{c \in \{1,..C\}} p(Y=c \mid X=x)$. However, this conditional probability is not what is directly accessable. Thus, use Bayes' rule as in §2.3

$$p(Y=c \mid X=x) = \frac{p(X=x \mid Y=c) \cdot p(Y=c)}{\sum\limits_{k=1}^{C} p(X=x \mid Y=k) \cdot p(Y=k)} \qquad (9.1)$$

It is easier to make assumtions about the probabilities on the right hand side. For $p(Y=k)$ we can simply set $p(Y=k) = \dfrac{\#\{y_n = k\}}{N} =: \pi_k$.

Classification $\quad f(x) = \operatorname{argmax}\limits_{c} p(Y=c \mid X=x)$

§ 9.2 Gaussian Discriminant Analysis

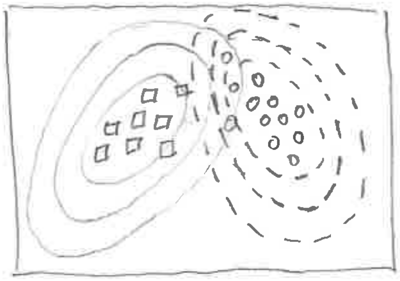
assume $p(X=x \mid y=c) = \mathcal{N}(x \mid \mu_c, \Sigma_c)$ $\qquad$ multivariate Gaussian as in § 3.2

in the iris example, this simply says that the sepal and petal dimensions are normally distributed for a given species.

The parameters μ_c and Σ_c depend on the class c, and are usually not known a-priori.

The parameters μ_c, Σ_c have to be estimated. This is called Model fitting, which will be described later.

2D-illustration; $C=2$



□ = 1 , o = 2

solid lines = contour lines of $p(X=x|y=1)$

dashed lines = contour lines of $p(X=x|y=2)$

Decision boundaries $\Leftrightarrow$ $p(Y=c_1|X=x) = p(Y=c_2|X=x)$

$\Leftrightarrow$ boundaries of regions with the same maximizing c

When determining decision boundaries we can neglect the denominator in (9.1) because it is the same for all $c \in \{1,2,..C\}$. Thus

$$p(X=x|Y=1)\pi_1 = p(X=x|Y=2)\pi_2$$

use multivariate Gaussian and take logarithms:

$$\ln \frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} - \frac{1}{2}(\underline{x}-\underline{\mu}_1)^T\Sigma_1^{-1}(\underline{x}-\underline{\mu}_1) = \ln \frac{\pi_2}{(2\pi)^{p/2}|\Sigma_2|^{1/2}} - \frac{1}{2}(\underline{x}-\underline{\mu}_2)\Sigma_2^{-1}(x-\mu_2)$$

we can rearrange this equation to this form

$$\frac{1}{2}x^TAx + b^Tx + c = 0$$

where $A = \Sigma_1^{-1} - \Sigma_2^{-1}$

$b = \dots$

$c = \dots$

Thus, when $\Sigma_1 \neq \Sigma_2$ we get a quadratic decision boundary =QDA

" $\Sigma_1 = \Sigma_2 \Rightarrow A = 0$ so we get a linear decision bdry. =LDA

9p5

Model Fitting (§9.2.4)

- The most straight-forward approach to approximate μ_c, Σ_c is the MLE as described in Chap 4. Each class c in the data $D = \{(x_n, y_n)\}_{n \in N}$ is considered separately. Thus

$$\hat{\mu}_c = \frac{1}{N_c} \sum_{n: y_n = c} x_n$$

$$c \in \{1, .., C\}$$

$$\hat{\Sigma}_c = \frac{1}{N_c} \sum_{n: y_n = C} (x_n - \hat{\mu}_c)(x_n - \hat{\mu}_c)^T$$

- Note that the number of free parameters is $D + \frac{(D+1)D}{2} \cdot C$ $(\mu \& \Sigma)$ if the dataset is small compared to this, then one often reduces the number of parameters, by using the same Σ for all classes c, to avoid overfitting. Then

$$\hat{\mu}_c = \frac{1}{N_c} \sum_{n: y_n = c} x_n$$

$$\hat{\Sigma}_c = \hat{\Sigma} = \frac{1}{N} \sum_{c=1}^{C} \sum_{n: y_n = c} (x_n - \hat{\mu}_c)(x_n - \hat{\mu}_c)^T \quad \text{(tied covariances)}$$

- Assume that Σ_c is diagonal ⇐ more on that later.

- Use shrinkage (§4.5) $\hat{\Sigma}_\lambda = \lambda \hat{\Sigma}_c + (1-\lambda)\,\mathrm{diag}(\hat{\Sigma}_c)$

(9 p4)

Naive Bayes

recall Bayes $$p(Y=c|X=x) = \frac{p(X=x|Y=c)\ \overset{=:\pi_c}{p(Y=c)}}{\sum_K p(X=x|Y=k)\cdot p(Y=k)} \qquad (*)$$

Naive Bayes: assume $p(X=x|Y=c)$ is independent, i.e., $X=(X_1 .. X_D)$ and
$$p(X=x|Y=c) = \prod_{d=1}^{D} \underbrace{p(X_d=x_d|Y=c)}_{f_{dc}(x)} \cdot \pi_c$$

If $p(X=x|Y=c)$ is a multivariate Gaussian, then this amounts to a diagonal $\Sigma_c = \text{diag}(\sigma_{1c}, .., \sigma_{Dc})$

and $$f_{dc}(x_d) = \frac{1}{(2\pi\sigma_{dc})^{1/2}} \exp\left(-\frac{1}{2}\frac{(x_d - \mu_{dc})^2}{\sigma_{dc}}\right)$$

instead of a gaussian, we can also approximate $f_{dc}(x_d)$ by a histogram (= non parametric model)

Nearest Centroid:

Assume tied covariances $(\Sigma_c = \Sigma)$ and tied priors $(\pi_c = \pi)$

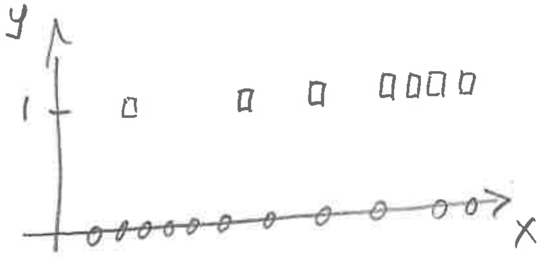
then from $(*)$ $p(Y=c|X=x) \sim p(X=x|Y=c) \sim \exp\left(-\frac{1}{2}(x-\mu_c)^T \Sigma^{-1}(x-\mu_c)\right)$

then: $$\underset{c}{\text{argmax}}\ p(Y=c|X=x) = \underset{c}{\text{argmin}} \underbrace{(x-\mu_c)^T \Sigma (x-\mu_c)}_{=: d^2(x,\mu_c)}$$

Mahalanobis distance.

i.e. the classifier is the nearest centroid in the $d(x,\mu)$ metric.

Chap 10: Logistic Regression



Example A bank wants to predict the risk of a loan default. They have data of the form $D = \{(x_n, y_n)\}_{n=1}^{N}$

where X = loan amount

$y \in \{0,1\}$ default Yes=1, No=0

goal: predict $P(Y=1 \mid X=x)$. For the Bayes model we would write

$$p(Y=1 \mid X=x) = \frac{P(X=x \mid Y=1) \cdot P(Y=1)}{p(X=x)}$$

$$p(Y=0 \mid X=x) = \frac{P(X=x \mid Y=0) \cdot P(Y=0)}{P(X=x)}$$ (see chap 9)

However, this does not appear to be advantageous, because it is not obvious how to obtain a model for $P(X=x \mid Y=c)$, $c \in \{0,1\}$. These don't look like Gaussians at all. Thus we use a model for $p(Y=1, X=x)$.

To that end, consider the sigmoid function, defined by

$$\sigma(t) = \frac{e^t}{1+e^t} = \frac{1}{1+e^{-t}}$$

properties:

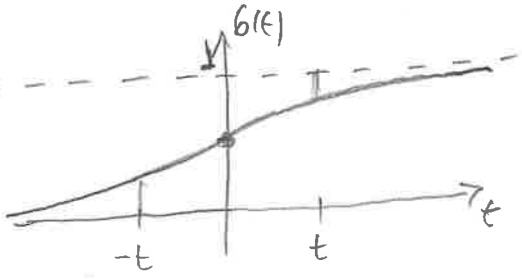
σ is monotonically increasing

$\sigma(-\infty) = 0$

$\sigma(\infty) = 1$

$\sigma(0) = 1/2$

$\sigma(-t) = 1 - \sigma(t)$



for the loan default model we set

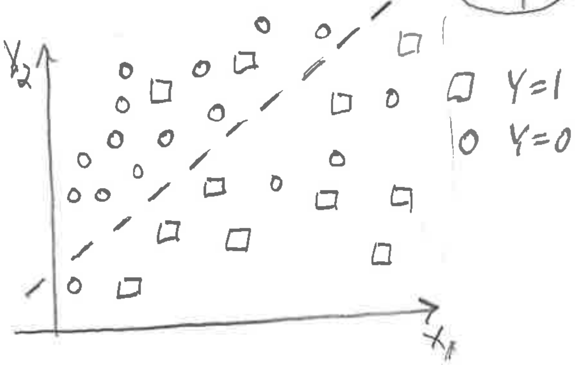
$$p(Y=1 \mid X=x) = \sigma(b + wx)$$

where b, w are parameters that we fit to the data.

Example: A better loan default model, that is based on X_1 = loan amount
X_2 = income

i.e., $D=2$

data:



In this case, the loan default model is

$$p(Y=1 \mid X=x) = \sigma(b + \underline{w}^T \underline{x})$$

where $b \in \mathbb{R}$ and $\underline{w} \in \mathbb{R}^D$ are the parameters.

Recall: $Ber(y \mid \theta) = \theta^y (1-\theta)^{(1-y)}$ Bernoulli distribution $y \in \{0,1\}$.

Hence we can write

$$p(Y=y \mid X=x) = Ber(y \mid \sigma(b+w^T x))$$

This is the binary logistic regression model.

The function $f(x)$ returns the more likely value of y. Thus

$$\begin{aligned} f(x) &= \mathbb{I}(p(Y=1 \mid X=x) > p(Y=0 \mid X=x)) \\ &= \mathbb{I}\left(\ln \frac{p(Y=1 \mid X=x)}{p(Y=0 \mid X=x)} > 0\right) \\ &= \mathbb{I}\left(\ln \frac{1}{1+e^{-a}} \Big/ \left(1-\frac{1}{1+e^{-a}}\right) > 0\right) = \mathbb{I}(\ln e^{+a} > 0) = \mathbb{I}(a > 0) \end{aligned}$$

$a := b + w^T x$

The line $a = 0 \iff b + w^T x = 0$ is the decision boundary. This is a line in $\mathbb{R}^2$ (dashed line in the figure), a plane in $\mathbb{R}^3$ or a D-1 dimensional hyper plane in $\mathbb{R}^D$.

With nonlinear models, other decision boundaries are possible. E.g., $p(Y=1 \mid X=x) = \sigma(x_1^2 + x_2^2 - R^2)$ will result in the circle rad = R center = origin.

Determine the parameters b and $\underline{w}$ $\Rightarrow$ MLE (§ 10.2.3) in the book: μ_n

$$\mathcal{L}(b,w) = p(D \mid b,w) = \prod_{n=1}^{N} \mathrm{Ber}(y_n \mid \theta_n) \qquad \text{where } \theta_n = \sigma(b+w^T x_n)$$

recall: $D = \{(x_n, y_n)\}_n$ and $\mathrm{Ber}(y,\theta) = \theta^y (1-\theta)^{1-y}$ $\quad y \in \{0,1\}$

$$\Rightarrow \mathrm{NLL}(w) = -\sum_{n=1}^{N} \{ y_n \ln(\theta_n) + (1-y_n) \ln(1-\theta_n) \}$$

$$= -\sum_{n: y_n = 1} \ln(\theta_n) - \sum_{n: y_n = 0} \ln(1-\theta_n)$$

for notational convenience write $b + w^T x_n = \underline{w}^T \underline{x}_n$ where $\underline{w} = [b, w]^T$ $\underline{x}_n = [1, x_n]^T$

also, write $\tilde{y}_n = \begin{cases} 1 & \text{when } y_n = 1 \\ -1 & \text{when } y_n = 0 \end{cases}$ and $a_n = b + w^T x_n = \underline{w}^T \underline{x}_n$

then

$$\mathrm{NLL}(w) = -\sum_{n: \tilde{y}_n = 1} \ln(\sigma(a_n)) - \sum_{n: \tilde{y}_n = -1} \ln(1 - \sigma(a_n))$$

$$= -\sum_{n: \tilde{y}_n = 1} \ln(\sigma(a_n)) - \sum_{n: \tilde{y}_n = -1} \ln(\sigma(-a_n)) \qquad \Leftarrow \text{last property of sigmoid fcn}$$

$$= -\sum_{n} \ln(\sigma(\tilde{y}_n a_n))$$

$$= \sum_{n} \ln(1 + \exp(-\tilde{y}_n a_n)) \qquad \Leftarrow \text{properties of ln and sigmoid}$$

To determine the minimum, we have to find $\nabla \mathrm{NLL}(w) = \left[\frac{\partial}{\partial b} \mathrm{NLL}, \frac{\partial}{\partial w} \mathrm{NLL} \right] = 0$

Note $a_n = b + w^T x_n \Rightarrow \frac{\partial a_n}{\partial b} = 1$ and $\frac{\partial a_n}{\partial w_i} = x_{ni}$ (ith component of the n-th data point)

Using the notations $\nabla = \begin{bmatrix} \frac{\partial}{\partial b} \\ \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial}{\partial w_D} \end{bmatrix}$ $\quad \underline{x}_n = \begin{bmatrix} 1 \\ x_{n1} \\ \vdots \\ x_{nD} \end{bmatrix}$ $\quad \underline{w} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix}$

we get $\nabla a_n = \underline{x}_n$

Now compute

$$\nabla NLL(w) = \nabla \sum_n \ln\left(1+\exp(-\tilde{y}_n a_n)\right)$$

$$= \sum_n \frac{d}{da_n} \ln\left(1+\exp(-\tilde{y}_n a_n)\right) \cdot \nabla a_n$$

$$= \sum_n \frac{\exp(-\tilde{y}_n a_n)}{1+\exp(-\tilde{y}_n a_n)} (-\tilde{y}_n) \cdot \underline{x}_n$$

$$= -\sum_n \sigma(-\tilde{y}_n a_n)\, \tilde{y}_n \underline{x}_n = \sum_{n:\tilde{y}_n=-1} \sigma(a_n) \underline{x}_n - \sum_{n:\tilde{y}_n=1} \sigma(-a_n) \underline{x}_n$$

$$= \sum_{n:y_n=0} \sigma(a_n) \underline{x}_n - \sum_{n:y_n=1} \left(1-\sigma(a_n)\right) \underline{x}_n$$

$$= \sum_{n=1}^{N} \sigma(a_n) \underline{x}_n - \sum_{n:y_n=1} \underline{x}_n$$

$$= \sum_{n=1}^{N} \left(\sigma(a_n) - y_n\right) \underline{x}_n$$

$$= \sum_{n=1}^{N} \left(\sigma(\underline{w}^T \underline{x}_n) - y_n\right) \underline{x}_n = \underline{0} \qquad (*)$$

At the minimum the gradient vanishes. (*) is a system of D+1 equations and D+1 unknowns.

There is no closed form solution for (*), hence one has to use numerical methods to solve either max $NLL(\underline{w})$ or $\nabla NLL(w) = \underline{0}$

We will discuss these soon.

§10.3 Multinomial Log Regression

so far, we talked about binary regression $C=2$, $y \in \{0,1\}$.

we had
$$p(Y=1\mid X=x) = \sigma(w^T x) = \frac{e^a}{1+e^a} \qquad a = \underline{w}^T \underline{x}$$
$$P(Y=0\mid X=x) = 1-\sigma(w^T x) = \frac{1}{1+e^a}$$

generalization of this to multiple classes:
$$p(Y=c\mid X=x) = \frac{e^{a_c}}{\sum_{k=1}^{C} e^{a_k}} \qquad a_c = \underline{w}_c^T \underline{x} \qquad c \in \{1,..,C\}$$

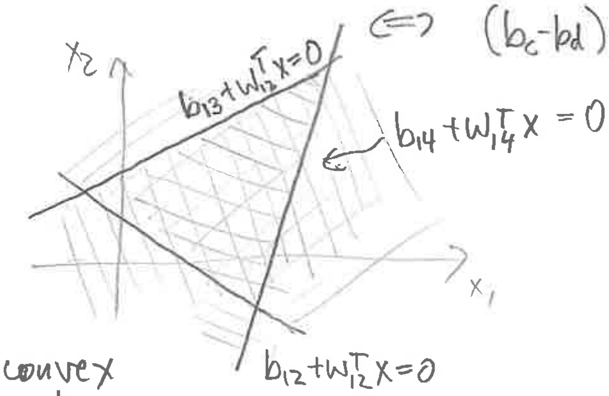
"Softmax function". Note $\sum_{k=1}^{C} p(Y=k\mid X=x) = 1$ as it should!

Also, $w_c \in \mathbb{R}^{D+1}$, so there are $(D+1)\cdot C$ parameters. To make this compatible to binary regression one sets $\underset{\uparrow\ \text{cap-}C}{\underline{w}_C} = \underline{0}$. Then there are $D \cdot C$ parameters.

To understand decision boundaries we set $p(Y=c\mid X=x) = p(Y=d\mid X=x)$ for $c \neq d$, similar to what we did for Gaussian decision analysis (Chap 9)

thus $e^{a_c} = e^{a_d} \iff a_c = a_d \iff b_c + w_c^T x = b_d + w_d^T x$ → hyperplanes
$$\iff (b_c - b_d) + (w_c^T - w_d^T)\,x = 0 \iff b_{cd} - w_{cd}^T x = 0$$

e.g. find the region where $c=1$ has max probability:



intersection of half-spaces } convex polytope.

The determination of the parameters $\underline{W} = \begin{bmatrix} -\ w_1^T\ - \\ \vdots \\ -\ w_{C-1}^T\ - \end{bmatrix}$ must be done numerically. Details are discussed in §10.3.2 and not covered here.

Chap 8: Optimization

$$\mathrm{NLL}(\theta) = -\ln \mathcal{L}(\theta)$$

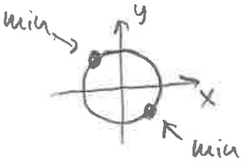
maximum likelihood: $\theta^* = \operatorname{argmin} \mathcal{L}(\theta) \overset{\downarrow}{=} \operatorname{argmax} \mathrm{NLL}(\theta)$

write $\theta \mapsto x$
$\mathcal{L} \mapsto f$

unconstrained: $\min\limits_{x\in\mathbb{R}^n} f(x)$ constrained: $\min\limits_{x \in C} f(x)$ domain in $\mathbb{R}^n$

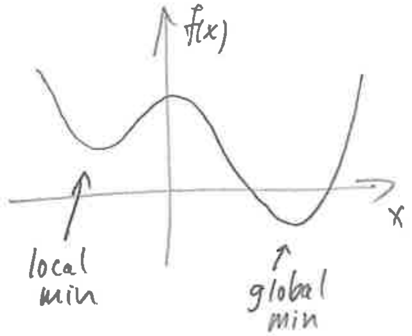
often: $C = \{x \in \mathbb{R}^n : g_j(x) \le 0 \quad j = 1..J\}$

example $\min x \cdot y$
$x^2 + y^2 \le 1$



Local minimum: $\|x - x^*\| \le \delta \Rightarrow f(x^*) \le f(x)$

Global minimum: $x \in C$ (or $\mathbb{R}^n$) $\Rightarrow f(x^*) \le f(x)$



Multivariable Taylor series:

single variable $\varphi(t) = \varphi(0) + t\varphi'(0) + \frac{t^2}{2}\varphi''(0) + O(t^3)$ $\varphi: \mathbb{R} \to \mathbb{R}$

fix $x, h \in \mathbb{R}^n$ for $f: \mathbb{R}^n \to \mathbb{R}$ define $\varphi(t) := f(x + th)$, by the chain rule

$$\varphi'(t) = \sum_{k=1}^n \frac{\partial f}{\partial x_k}(x+th)\cdot h_k \quad \Rightarrow \varphi'(0) = \sum_{k=1}^n \frac{\partial f}{\partial x_k}(x) h_k = \nabla f(x)\cdot h$$

$$\varphi''(t) = \sum_{k=1}^n \sum_{l=1}^n \frac{\partial^2 f}{\partial x_k \partial x_l}(x+th) h_k h_l \quad \Rightarrow \varphi''(0) = \sum_{k=1}^n \sum_{l=1}^n \frac{\partial^2 f}{\partial x_k \partial x_l}(x) h_k h_l$$

$$= h^T Hf(x) h$$

where $\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$ $Hf(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_k \partial x_l} \end{bmatrix}$

gradient Hesse matrix

set $t=1$ and $h = y - x$ then $\varphi(1) = f(x + 1(y-x)) = f(y)$ $\varphi(0) = f(x)$ and from the single-variable Taylor series it follows that

$$f(y) = f(x) + \nabla f(x)\cdot(y-x) + \frac{1}{2}(y-x)^T Hf(x)(y-x) + O(\|y-x\|^2)$$

1st order conditions for extremum:

x^* is an extremum $\Rightarrow$ $\nabla f(x^*) = 0$

2nd order conditions:

$\nabla f(x^*) = 0$ and $Hf(x^*)$ is SPD $\Rightarrow$ x^* is a local minimum.

The proof of this follows from the Taylor series.

Important: The stated conditions only apply for unconstrained optimization problems. For constrained problems they are generalized by the Kuhn - Tucker conditions.

Convexity

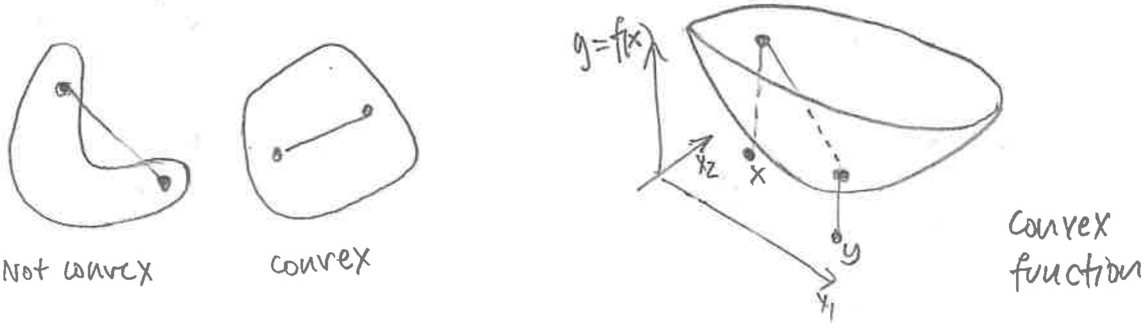
- A subset $C \subset \mathbb{R}^n$ is convex if for two points in C the line segment between the points is in C, i.e.,

$$x \in C,\ y \in C \Rightarrow \text{for any } 0 \le \lambda \le 1 \quad \lambda x + (1-\lambda) y \in C$$

- A function $f: \mathbb{R}^n \to \mathbb{R}$ is convex if for two points on the graph of f the line segment between the two points lies above the graph of f

$$f(\lambda x + (1-\lambda) y) \le \lambda f(x) + (1-\lambda) f(y) \quad \text{for any } 0 \le \lambda \le 1$$

Illustration:



- A function is strictly convex if for $x \neq y$ and $0 < \lambda < 1$

$$f(\lambda x + (1-\lambda) y) < \lambda f(x) + (1-\lambda) \cdot f(y)$$ ← missing in the notes

holds

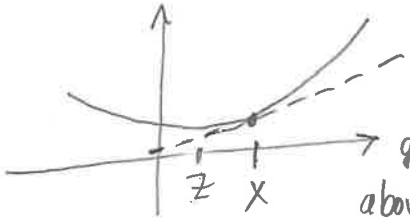
Theorem: f is strictly convex $\Leftarrow$ $Hf(x)$ is SPD for all x

f is convex $\Leftrightarrow$ $Hf(x)$ is positive semidefinite for all x

Subgradient

for a smooth and convex function we have

$$f(z) \geq f(x) + \nabla f(x)^T (z-x)$$



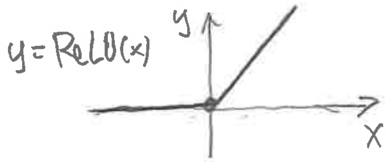
the graph is above the tangent plane

We call $g \in \mathbb{R}^n$ a subgradient if

$$f(z) \geq f(x) + g^T(z-x)$$

Note: If f is smooth at x then the only subgradient is $g = \nabla f(x)$.

This concept is more interesting if f is not smooth at x. Then there are more possibilities. Example: $f(x) = \text{ReLU}(x)$



at $x=0$ any $g \in [0,1]$ is a subgradient

Notation $\partial \text{ReLU}(x) = \begin{cases} \{0\} & x<0 \\ [0,1] & x=0 \\ \{1\} & x>0 \end{cases}$

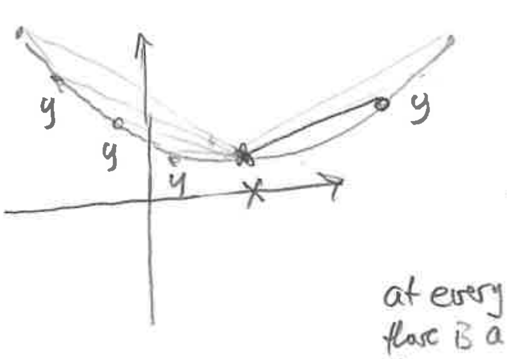
Lipschitz Regularity

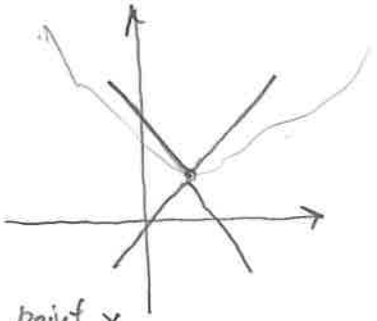
A function is Lipschitz if there is a constant $L>0$ such that

$$|f(x) - f(y)| \leq L \cdot |x-y| \quad \text{for all } x, y \text{ in the domain.}$$

Graphical illustration

$$\frac{|f(x) - f(y)|}{|x-y|} \leq L$$





at every point x there is a cone that does not contain the graph

Lipschitz ⇒ continuous but not necessarily differentiable

ReLU is an example of a convex, Lipschitz continuous function

§ 8.2 1st order Methods

Goal Solve: $\min\limits_{x} f(x)$

General idea: Start with an initial guess x_0, then follow a descent direction d_0, for a certain distance, then repeat.

$\underline{d}$ = descent direction at x: There is $s_{max} > 0$: $f(x+sd) < f(x)$ for all $0 < s < s_{max}$

If f is smooth then $\nabla f(x) \cdot d < 0$

since $\nabla f(x) \cdot d = |\nabla f(x)| \cdot |d| \cdot \cos\theta$ $\quad d := -\nabla f(x)$ is the direction of steepest descent, b/c $\theta = -\pi$

Descent Algorithm:

```
    $x_0$ = initial guess
    for $t = 0, 1, 2, 3, \dots$
        select a descent direction $d_t$
        select a step size $s_t$
        $x_{t+1} = x_t + s_t d_t$
    end
```

← s_t is also called learning rate

Descent algorithms differ on how to choose d_t and s_t.

If $\nabla f(x)$ can be computed easily, then $d_t = -\nabla f(x_t)$ is obvious (but not always the best).

For the step size s_t = constant is usually not recommended. A better, but much more expensive way is to do a line search:

$$s_t = \operatorname*{argmin}_{s>0} f(x_t + s d_t)$$

i.e., seek the minimum of f in the direction of d_t.

Example $f(x) = \frac{1}{2} x^T A x - b^T x + c$ (8 p5

Suppose A is SPD $\Rightarrow$ f is convex

gradient $\nabla f(x) = Ax - b$

Note that by the second order conditions:

$$\left.\begin{array}{l} \nabla f(x) = 0 \Rightarrow Ax^* = b \\ Hf(x) = A \end{array}\right\} \qquad \min_{x \in \mathbb{R}^n} f(x) \iff \text{solve } Ax = b$$

The steepest descent algorithm is an iterative method to solve the linear system.

Suppose x_t is the current iterate
then $d_t = b - Ax_t$ is the descent direction.

Find the learning rate (drop subscripts t)

$$\min_{\rho > 0} f(x + \rho d) = \min_{\rho > 0} \frac{1}{2}(x + \rho d)^T A (x + \rho d) - b^T(x + \rho d) + c$$

$$= \min_{\rho > 0} \frac{1}{2} x^T A x + b^T x + c + \rho\, d^T \underbrace{(Ax - b)}_{= -d} + \frac{1}{2} \rho^2 d^T A d$$

equation of a parabola, for the min: $\rho_t d^T A d - d^T d = 0 \Rightarrow \rho_t = \dfrac{d^T d}{d^T A d}$

Steepest descent algorithm for $f(x) = \frac{1}{2} x^T A x - b^T x + c$

```
$x_0$ = initial guess
for $t = 0, 1, 2, \dots$
    $d_t = b - Ax_t$
    $\rho_t = \dfrac{d_t^T d_t}{d_t^T A d_t}$
    $x_{t+1} = x_t + \rho_t d_t$
end
```

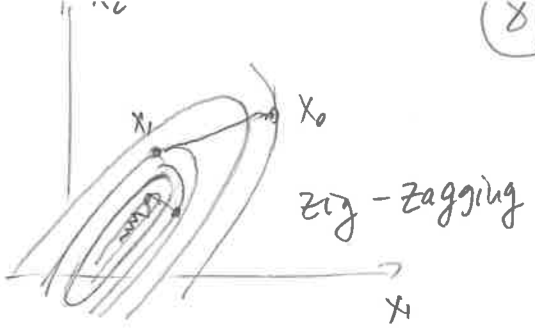
It can be shown that for smooth $f(x)$ the iterates satisfy

$$|f(x_{t+1}) - f(x^*)| \le \mu\, |f(x_t) - f(x^*)| \qquad \text{for some } 0 < \mu < 1.$$

This is called linear convergence. If μ is close to 1 the convergence rate is slow. for $f(x) = \frac{1}{2} x^T A x - bx + c$ one can

Show that $\mu = \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2$

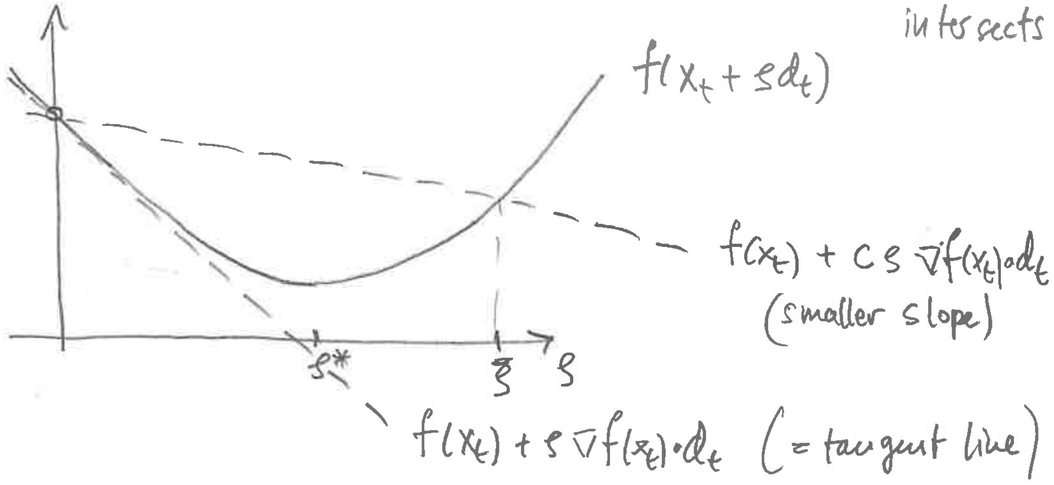
Hence, if $\lambda_{max} \gg \lambda_{min}$ then $\mu \approx 1$



Armijo Rule

It is not necessary to find the exact minimum of the line search. An approximation will suffice.

We know the tangent line is below the graph of a convex function also, a line with a smaller slope is initially above the graph, until it intersects the graph at $\bar{\xi}$.



$f(x_t + \xi d_t)$
$f(x_t) + c\xi \nabla f(x_t) \cdot d_t$
(smaller slope)
ξ^*
$\bar{\xi}$
ξ
$f(x_t) + \xi \nabla f(x_t) \cdot d_t$ (= tangent line)

$c < 1$
often $c \ll 1$

If c is small enough, then the minimum ξ^* satisfies $\xi^* < \bar{\xi}$.

This suggests the following algorithm:

```
choose $0 < c \ll 1$ and $0 < \beta < 1$ and $\xi > 0$
if $f(x_t + \xi d_t) < f(x_t) + c\xi \nabla f(x_t) \cdot d_t$   stop   $\xi = \xi_t$
else   set $\xi = \beta\xi$ and repeat the test.
```

(skip momentum methods)

§ 8.3 Second Order Methods

The methods in § 8.2 are called 1st order because they involve only first-order derivatives. Now we consider methods with 2nd derivatives.

Newton

This method is more commonly described to find solutions of systems of nonlinear equations. I.e., $\underset{\text{vector!}}{F} : \mathbb{R}^N \to \mathbb{R}^N : \; F(x) = 0$.

In the case of finding the minimum of a scalar function $\min_x f(x)$ it follows from the 1st order conditions that $\nabla f(x) = 0$, which is a system of nonlinear equations with $F(x) = \nabla f(x)$.

Newtons method is based on linearization. If x_t is the current guess then d in $x_t + d = x^*$ is the corrector. To approximate Δx write:

$$0 = F(x^*) = F(x_t + d) \approx F(x_t) + F'(x_t) \cdot d \overset{!}{=} 0$$

Thus the Newton update is $d_t = -\left[F'(x_t)\right]^{-1} \cdot F(x_t)$

note that $F'(x)$ is the Jacobian matrix. If $F(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix}$

then $F'(x) = \left[\frac{\partial f_k}{\partial x_\ell}\right]_{k\ell} = \begin{bmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_n(x) \end{bmatrix}$

If we solve $F(x) = \nabla f(x) = 0$ then $f_k(x) = \frac{\partial f}{\partial x_k} \Rightarrow F'(x) = \left[\frac{\partial^2 f}{\partial x_k \partial x_\ell}\right]_{k,\ell}$

$$= Hf(x)$$

The Newton method for solving $\min_x f(x)$ is summarited as follows

```
$x_0$: initial guess
for $t = 0, 1, 2, \dots$
    compute $\nabla f(x_t)$, $Hf(x_t)$
    solve $Hf(x_t) \cdot d_t = -\nabla f(x_t)$
    update $x_{t+1} = x_t + d_t$
end
```

← damped Newton: $x_{t+1} = x_t + \rho_t d_t$
where $\rho_t \approx \arg\min f(x_t + \rho_t d_t)$
e.g., use the Armijo rule.

The rate of convergence is much faster than 1st order methods. One can show that $|x_{t+1} - x^*| \leq c|x_t - x^*|^2$

The downsides of Newton are:

1.) method often does not converge, when the initial guess is far from x^*

2.) calculation of H and solving the linear system is often cost prohibitive.

To address 2.) one can use methods that approximate $Hf(x_t)$ using $\nabla f(x_0), \dots \nabla f(x_t)$ these are called quasi-Newton methods

To address 1.) one can use trust region methods and regularization e.g. Levenberg Marquard.

We don't discuss this in this course.

§8.4 Stochastic Gradient Descent

Problem: want to minimize a function of the form $f(x) = \frac{1}{N}\sum_{n=1}^{N} f_n(x)$

We get this problem, for instance, when minimizing $NLL(\theta, D)$ where $|D| = N$ is a large data set.

The cost of evaluating $f(x)$ and $\nabla f(x)$ may be high if N is large.

Idea: replace $f(x)$ by $f(x,z)$ where z is a random variable and $f(x,z)$ is cheaper to evaluate than $f(x)$.

and $\mathbb{E}\, f(x,z) = f(x)$

Example z = random number in $\{1,..,N\}$ with probability $p(z=n) = \frac{1}{N}$ for all n.

then $\mathbb{E}(f(x,z)) = \sum_{n=1}^{N} f(x, z=n) \cdot \frac{1}{N} = \frac{1}{N}\sum_{n=1}^{N} f_n(x) = f(x)$

Improvements:

- Instead of picking one number at random one could use a "minibatch" $B = \{n_1 .. n_{\#B}\}$ $n_b \in \{1 .. n\}$ and let

$$f(x, z = \{n_1 .. n_b\}) = \frac{1}{\#B} \cdot \sum_{b=1}^{\#B} f_{n_b}(x)$$

- Learning Rate
  - Armijo rule
  - full line search
  - Heuristics: $\varrho_t \to 0$ but such that $\frac{\sum_t \varrho_t^2}{\sum_t \varrho_t} \to 0$ (eg. $\varrho_t = \frac{1}{t}$)

- Averaging

$$\bar{\theta}_t := \frac{1}{t}\sum_{k=1}^{t} \theta_k = \frac{1}{t}\theta_k + \frac{t-1}{t}\bar{\theta}_{t-1}$$

Example 1: for stochastic gradient

Linear Regression (we'll get into that soon)

Recall from linear algebra:

given data points $x_1 \dots x_N$ $\quad x_n \in \mathbb{R}^D$
measurements $y_1 \dots y_N$ $\quad y_n \in \mathbb{R}$

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1D} \\ \vdots & & \vdots \\ x_{N1} & \cdots & x_{ND} \end{bmatrix} \qquad \underline{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

fit a linear model to the data: $\quad y = \underline{x}^T \underline{\theta}$

where $\underline{\theta} \in \mathbb{R}^D$ are the parameters that are determined from the data points:

Loss-function $\quad \mathcal{L}(\theta) = \frac{1}{2N} \sum_{n=1}^{N} (x_n^T \theta - y_n)^2 = \frac{1}{2N} \| X^T \underline{\theta} - \underline{y} \|^2$

this is a least squares problem. The direct solution of $\min_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta)$ can be done, e.g., with the QR-factorization, but this method does not scale well in N and D.

We can also use steepest descent of § 8.2. In this case the gradient is:

$$\frac{\partial}{\partial \theta_k} \mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} (\underline{x}_n^T \underline{\theta} - y_n) \cdot x_{nk} \Rightarrow \nabla \mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \underbrace{(\underline{x}_n^T \underline{\theta} - y_n)}_{\text{scalar}} \cdot \underbrace{\underline{x}_n}_{\text{vector in } \mathbb{R}^D}$$

But the evaluation of $\nabla \mathcal{L}(\theta)$ scales like $D \cdot N$ which is expensive.

For the stochastic gradient descent method we define

$$\mathcal{L}(\theta, n) = \frac{1}{2} (x_n^T \theta - y_n)^2 \quad \text{where } n \text{ is random with } p(n) = \frac{1}{N}$$

Thus $\mathbb{E}(\mathcal{L}(\theta, n)) = \mathcal{L}(\theta)$ and the gradient is

$$\nabla \mathcal{L}(\theta, n) = (x_n^T \theta - y_n) \cdot \underline{x}_n$$

The evaluation of $\nabla \mathcal{L}(\theta, n)$ scales like D which is much cheaper.

Example 2 for Stochastic Gradient Descent

(8 p11)

Application to binary logistic regression (§ 10.2.4)

recall from Chap 10: Model for binary classification $Y \in \{0,1\}$

$$P(Y=1 \mid X=x) = \sigma(\underline{w}^T \underline{x})$$

where $\sigma(t) = \frac{e^t}{1+e^t}$ $\underline{w} = \begin{bmatrix} b \\ w \end{bmatrix}$ $\underline{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$ $\underline{w}^T \underline{x} = b + w^T x$

$\underline{w} \in \mathbb{R}^{D+1}$ is the set of parameters, that is fit to the data $x_1 \cdots x_n$ $y_1 \cdots y_n$

by minimizing

$$NLL(\underline{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + \exp\left(-\tilde{y}_n \underline{x}_n^T \underline{w}\right)\right)$$

(See chap 10)

$$\nabla NLL(w) = \frac{1}{N} \sum_{n=1}^{N} \left(\sigma(\underline{w}^T \underline{x}_n) - y_n\right) \cdot \underline{x}_n$$

The cost of ∇NLL scales like $D \cdot N$. For stochastic gradient, we define

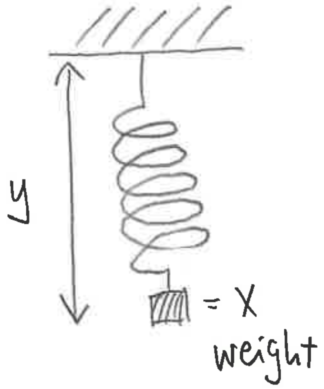
$$NLL(w,n) = \ln\left(1 + \exp(-\tilde{y}_n \underline{x}_n^T \circ \underline{w}\right)$$

$$\nabla NLL(W,n) = \left(\sigma(\underline{w}^T \underline{x}_n) - y_n\right) \cdot \underline{x}_n$$

again the cost of ∇NLL scales like D

Chapter 11 : Linear Regression

Motivating Exple



Hooke's Law

$$y = w_0 + w_1 X$$

$\uparrow$ length of spring with no load

$\uparrow$ extension is proportional to weight. w_1 = spring constant.

Suppose we want to measure w_0 and w_1. Do measurements of y for different weights. Thus we obtain a data set $D = \{(x_1,y_1), \dots, (x_N, y_N)\}$. When we do measurements we cannot expect that every data point (x_n, y_n) satisfies Hooke's law because of errors of measurements. We can assume that the errors are normally distributed,

$$p(y \mid x, w) = \mathcal{N}(y \mid w_0 + w_1 x, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{(y - w_0 - w_1 x)^2}{2\sigma^2}\right)$$

i.e., the mean is the exact Hooke's law, but the measurements are distributed around the mean with a certain variance $\sigma > 0$.

To determine the unknown parameters w_0, w_1 we do a MLE:

$$\mathcal{L}(w) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_n - (w_0 + w_1 x_n))^2}{2\sigma^2}\right)$$

$$\Rightarrow \quad NLL(w) = -N \cdot \ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left[y_n - (w_0 + w_1 x_n)\right]^2$$

No we could compute $\nabla NLL(w) = 0$ and thus solve for w_0, w_1. Instead, we do some linear algebra to bring this problem into a familiar form

set $X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_N \end{bmatrix}$ and $\underline{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ and $\underline{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$ then

$$NLL(\underline{w}) = -N \cdot \ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \frac{1}{2\sigma^2} \| X w - y \|^2$$

minimizing NLL(w) is equivalent to minimizing RSS(w), defined by

$$RSS(w) = \|X\underline{w} - \underline{y}\|^2$$

This is a least squares problem. Note that $\underline{w}$ is independent of σ.

More generally, if $x \in \mathbb{R}^{D'}$ and $y \in \mathbb{R}$ satisfy a linear relationship

$$y = \underline{w}^T \underline{x}$$

and if $y = \mathcal{N}(y \mid \underline{w}^T \underline{x})$

then the MLE estimate for a given data set $\{x_n, y_n\}$ $x_n \in \mathbb{R}^D$

solves the least square system

$$\min_{w} \|X\underline{w} - y\|^2$$

where $X = \begin{bmatrix} - x_1^T - \\ \vdots \\ - x_N^T - \end{bmatrix} \in \mathbb{R}^{N\times D}$ $\quad w = \begin{bmatrix} w_1 \\ \vdots \\ w_D \end{bmatrix} \in \mathbb{R}$ $\quad \underline{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N$

Note: we need $N \gg D$

Normal Equations

$$RSS(w) = \tfrac{1}{2}\|Xw - y\|^2 = \tfrac{1}{2}(X\underline{w} - \underline{y})^T(X\underline{w} - \underline{y}) = \tfrac{1}{2} w^T X^T X w - w^T X^T y$$

$$\nabla RSS(w) = X^T X w - X^T y = 0$$

$$\iff X^T X w = X^T y$$

This is a linear system $Aw = b$ with $A = X^T X \in \mathbb{R}^{D\times D}$ & $b \in \mathbb{R}^D$

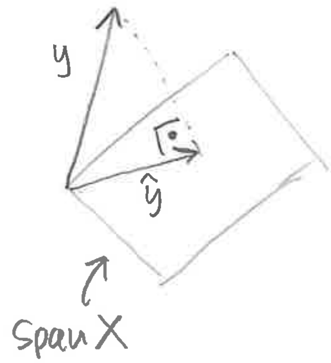
note that if the columns of X are independent, then A is SPD.

We could find w by solving this system, but this is computationally expensive when N and D are large.

Geometry of the Least-squares problem

we saw $\min_{w}\|Xw-y\| \iff X^TXw = X^Ty$

if $y \notin \text{span}(X)$ then $\|Xw-y\|>0$ for all $w\in\mathbb{R}^D$



$\hat{y}\in \text{span}\,X$ is the closest vector to y if $\hat{y}-y \perp \text{span}\,X$

$\hat{y}$ is called the orthogonal projection of y into span$X$

it satisfies $(\hat{y}-y)^T\cdot X\hat{w}=0$ for all $\hat{w}\in\mathbb{R}^D$

$$\left.\begin{aligned} &\iff X^T(\hat{y}-y)=0 \\ &\text{also } \hat{y}\in\text{span}(X) \Rightarrow \hat{y}=Xw \end{aligned}\right\} X^T(Xw-y)=0$$

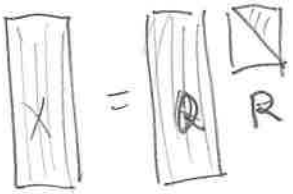
$$\iff X^TXw = X^Ty$$

normal equations.

IF N, D are moderately sized (say, less than 5,000) the best way to solve a least squares problem is by the QR-factorization:

$X=QR$ $\quad Q$ = matrix with orthogonal columns $\in\mathbb{R}^{N\times D}$

R = upper triangular $\in\mathbb{R}^{D\times D}$ ← invertible if columns of X are independent



Note $Q^TQ=I$

but $QQ^T\neq I$

then $X^TXw = X^Ty$

$\iff R^T\underbrace{Q^TQ}_{I}Rw = R^TQ^Ty \Rightarrow Rw=Q^Ty \Rightarrow w=R^{-1}Q^Ty$

Algorithm for $\hat{w} = \arg\min \| Xw - y \|$

1.) $Q, R = qr(X)$, i.e, compute the QR factorization

2.) $z = Q^T y$

3.) solve $Rw = z$ using backward elimination

Recall (1) $\hat{w} = (X^T X)^{-1} X^T y = R^{-1} Q^T y$ Solution of least squares problem

(2) $\hat{y} = Xw = X(X^T X)^{-1} X^T y = \dots = QQ^T y$ orthogonal projection of y into W

Recall $\hat{w}$ is the approximation of the linear function $f(x) = w^T x$ in (1) we see that $\hat{w}$ strongly depends on $(X^T X)^{-1}$ which in turn depends on the data.

We can ask the question: How does the variance σ^2 for y affect the variance of $\hat{w}$? To that end, note that for $\underline{y} = \begin{bmatrix} y_1 \\ y_n \end{bmatrix}$ the covariance matrix is $Cov[\underline{y}] = \sigma^2 I$ (components are iid)

Thus, by (1) the covariance of $\hat{w}$ is $Cov[\hat{w}] = Cov[Ay]$, $A = (X^T X)^{-1} X^T$

From exercise 3.4 #1 we know

$$Cov[\hat{w}] = A\, cov[y]\, A^T = (X^T X)^{-1} X^T \sigma^2 I\, X (X^T X)^{-1}$$
$$= \sigma^2 (X^T X)^{-1} X^T X (X^T X)^{-1}$$
$$= \sigma^2 (X^T X)^{-1}$$

also $\mathbb{E}[\hat{w}] = w$

Conclusion: Formula (1) for approximating the true parameter is unbiased (i.e., the expected value = exact value) but the covariance will be large if $(X^T X)^{-1}$ has large eigenvalues.

This is equivalent to: $X^T X$ close to singular $\Rightarrow$ large $Cov[\hat{w}]$.

Shrinkage methods are aimed at reducing Cov[$\hat{w}$] ↗ §11.3: Ridge ↘ §11.4: Lasso

The idea is simply to add a term to the NLL-function. We already discussed this in §4.5.

§11.3 Ridge Regression

$$\hat{w} = \text{argmin} \frac{1}{2\sigma^2}\underbrace{(y-Xw)^T(y-Xw)}_{\to RSS} + \frac{1}{2\tau^2}\|w\|_2^2 \qquad (1)$$

This simply 'penalizes' that w gets large, as this is a result of $(X^TX)^{-1}$ having large eigenvalues.

$\lambda = \frac{\sigma^2}{\tau^2}$ regularization parameter.

Now we minimize $J(w) = \frac{1}{2}(y-Xw)^T(y-Xw) + \frac{\lambda}{2}\|w\|^2$

$$\nabla J(w) = 0 \Rightarrow X^TXw - X^Ty + \lambda w = 0$$

$$\Leftrightarrow (X^TX + \lambda I)w = X^Ty \Leftrightarrow w = (X^TX+\lambda I)^{-1}X^Ty \qquad (2)$$

We do not use (2) for numerical purposes. Instead, we write (1) in the equivalent form

$$(1) \Leftrightarrow \hat{w} = \text{argmin} \frac{1}{2}\left\|\frac{1}{\sigma}Xw - \frac{1}{\sigma}y\right\|^2 + \frac{1}{2}\left\|\frac{w}{\tau}\right\|^2$$

$$= \text{argmin} \frac{1}{2}\left\| \underbrace{\begin{bmatrix}\frac{1}{\sigma}X \\ \frac{1}{\tau}I\end{bmatrix}}_{\tilde{X}} w - \underbrace{\begin{bmatrix} y/\sigma \\ 0 \end{bmatrix}}_{\tilde{y}} \right\|^2 = \text{argmin} \frac{1}{2}\|\tilde{X}w - \tilde{y}\|^2$$

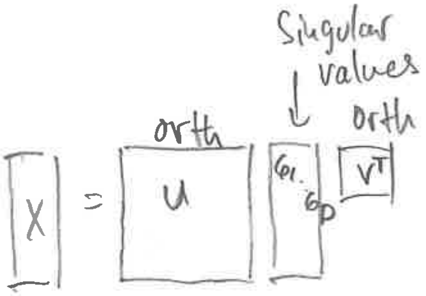
Now use the QR-factorization of $\tilde{X}$ to solve the least squares problem as discussed before.

Analysis of ridge regression using the SVD

we have seen: $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$

consider the SVD of $X = U S V^T$ ⟺

(see § 7.5)



$U^T = U^{-1}$
$V^T = V^{-1}$

Thus $\hat{w} = (V S^T \underbrace{U^T U}_{=I} S V^T + \lambda I)^{-1} V S^T \underbrace{U^T y}$

⟺ $\hat{w} = (V S_2 V^T + \lambda V V^T)^{-1} V S^T \hat{y}$

$S_2 = S^T S = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_D^2 \end{bmatrix} \in \mathbb{R}^{D \times D}$

$= V (S_2 + \lambda I)^{-1} \underbrace{V^T V}_{I} S^T \hat{y}$

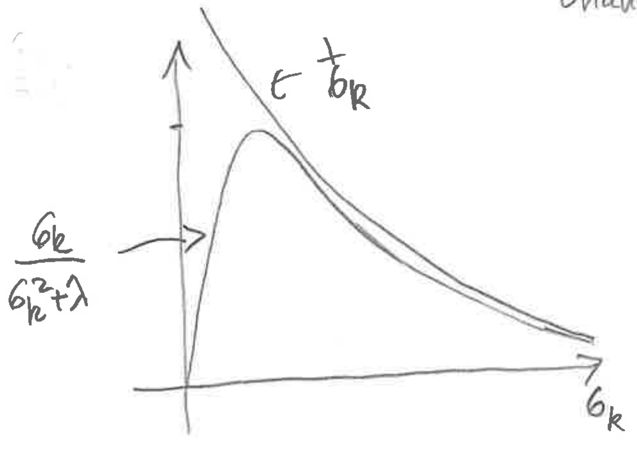
Set $\tilde{w} = V^T \hat{w}$ then

$\tilde{w} = \underbrace{(S_2 + \lambda I)^{-1}}_{\text{diagonal}} S^T \hat{y}$ ⟹ $\tilde{w}_k = \frac{\sigma_k}{\sigma_k^2 + \lambda} y_k$

components

when $\lambda = 0$ $\tilde{w}_k = \frac{1}{\sigma_k} y_k$

The factor $\lambda > 0$ switches off components with small σ_k but does not change components with large σ_k.



Lasso Regression

Simple change from ridge regression: Replace the $\|W\|_2^2$ regularizer by $\|W\|_1$.

Recall: $\|W\|_p = \sum_{d=1}^{D} |W_d^p|^{1/p}$ is a norm when $p \geq 1$

$p=2$: Euclidean norm

$p=1$: $\|w\|_1 = \sum_{d=1}^{D} |W_d|$

$p=\infty$: $\|W\|_\infty = \max_d |W_d|$

Thus the Lasso (Stands for Least Absolute Shrinkage and Selection Operator) is

$$\hat{w} = \operatorname{argmin} \frac{1}{2}\|y - Xw\|_2^2 + \lambda \|w\|_1 \qquad (*)$$

Interpretation as MLE estimator:

Recall that we had:

$$\mathcal{L}(w) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(y_n - W^T x_n)^2}{2\sigma^2}\right) \quad \to \max$$

To penalize large w's we can multiply with a Laplace prior, i.e.,

$$\mathcal{L}(w) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(y_n - W^T x_n)^2}{2\sigma^2}\right) \cdot \prod_{d=1}^{D} \exp(-\lambda w_d)$$

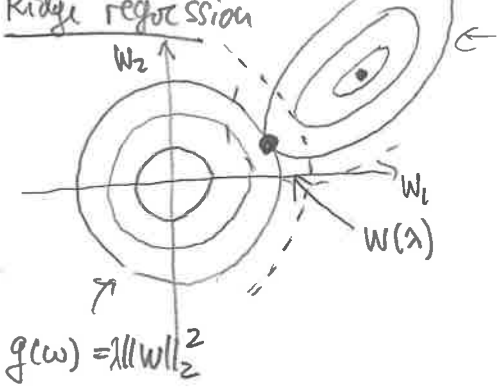
leads to $(*)$.

To understand the differences between Ride and Lasso note that for general $f(w)$ ad $g(w)$ the first order conditions imply that at a min of $L_\lambda(w) = f(w) + \lambda g(w)$ we get

$$\nabla L_\lambda(w) = 0 \quad \Rightarrow \quad \nabla f(w) = -\lambda \nabla g(w)$$

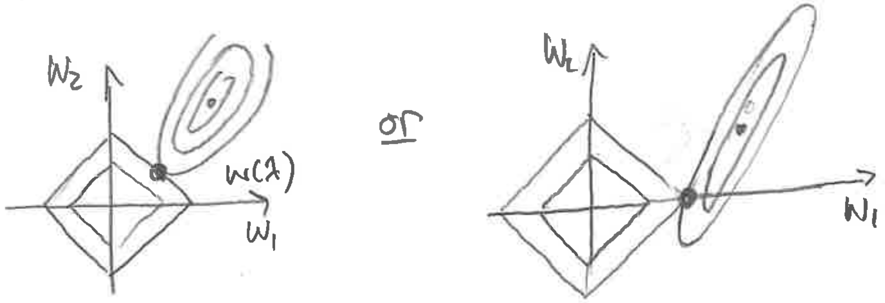
i.e. gradients are parallel or anti parallel at extremum. For ridge regression we have $f(w) = \frac{1}{2}\|Xw - y\|^2$, contour lines are ellipses and $g(w) = \lambda \|w\|^2$, contour lines are circles. At a min. contour lines must touch each other b/c the gradient is perpendicular. See figure.

Ridge regression


W_2
W_1
$w(\lambda)$
$\leftarrow f(w) = \frac{1}{2}\|Xw - y\|_2^2$
$g(w) = \lambda\|w\|_2^2$

The optimal solution for $\lambda = 0$ is the Center of the ellipse. The optimal Solution for $\lambda > 0$ is $w(\lambda)$.

For Lasso, the contour lines of $g(w)$ are diamond-shaped, and not smooth on the coordinate axes. Here, it is possible that the contour lines meet at a smooth point, with gradients of $f(w)$ and $g(w)$ going in opposite directions. However it is possible that the min occurs at a coordinate axis



Thus Lasso has the ability to identify variables with a vanishing w-factor, that is it can identify variables that do not affect the classifier, and can therefore be eliminated. This is called feature selection.

Gradient of Lasso-objective fcn

$$\mathcal{L}(w) = \tfrac{1}{2}\| y - Xw\|_2^2 + \lambda \|w\|_1$$

$$NLL(w) = \tfrac{1}{2}\| y - X\underline{w}\|_2^2 = \tfrac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{l=1}^{D} x_{nl} \cdot w_l\right)^2$$

$$\frac{\partial}{\partial w_d} NLL(w) = \sum_{n=1}^{N} -\left(y_n - \sum_{l=1}^{D} x_{nl} w_l\right) \cdot x_{nd} \overset{\text{"extract" } l=d}{=} \underbrace{-\sum_{n=1}^{N}\left(y_n - \sum_{\substack{l=1 \\ l\neq d}}^{D} x_{nl} w_l\right) x_{nd}}_{c_d} + \underbrace{\sum_{n=1}^{N} x_{nd}^2}_{=:a_d} w_d$$

Set $\underline{w}_{-d} := \begin{bmatrix} w_1 \\ \leftarrow \\ w_D \end{bmatrix}$ ← w_d deleted $\quad w \in \mathbb{R}^D \Rightarrow w_{-d} \in \mathbb{R}^{D-1}$

then $\frac{\partial}{\partial w_d} NLL(w) = a_d w_d - c_d$

where $a_d = \sum_{n=1}^{N} x_{nd}^2 = \| X(n,:)\|^2$

and $c_d = \sum_{n=1}^{N} (y_n - w_{-d}^T \underline{x}_{n,-d}) \cdot x_{nd}$

include $\lambda\|w\|_1$: not differentiable, need the subgradient.

$$\partial\mathcal{L}(w) = a_d w_d - c_d + \lambda \cdot \begin{cases} -1 & \text{if } w_d < 0 \\ [-1,1] & \text{if } w_d = 0 \\ +1 & \text{if } w_d > 0 \end{cases}$$

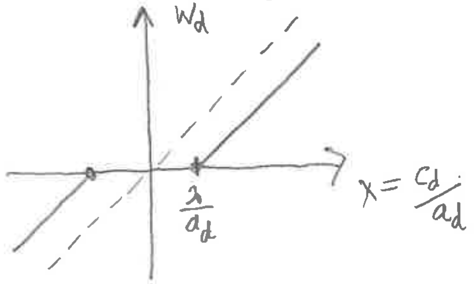
$\partial\mathcal{L}(w) = 0$ gives

$w_d = \frac{c_d + \lambda}{a_d}$ when $w_d < 0 \Leftrightarrow c_d < -\lambda$

$w_d = 0$ when $w_d = 0 \Leftrightarrow -\lambda \le c_d < \lambda$

$w_d = \frac{c_d - \lambda}{a_d}$ when $w_d > 0 \Leftrightarrow c_d > \lambda$

Mind that c_d/a_d = soln without regularization, $w_d = w_d(\lambda)$ includes regularization

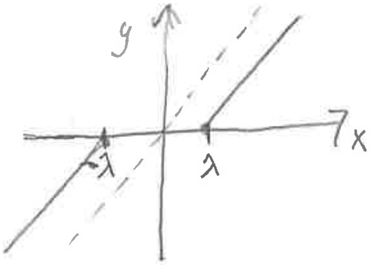


we see $|w(\lambda)| < \frac{c_d}{a_d}$

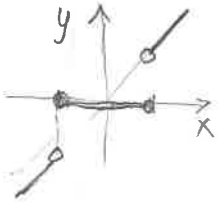
⇒ shrinkage

"soft thresholding"

Soft Threshold function $\quad \text{SoftThrsh}(x,\lambda) = \begin{cases} x+\lambda, & x<-\lambda \\ 0, & -\lambda\le x\le\lambda \\ x+\lambda, & x>\lambda \end{cases}$



Remark $\quad \text{HardThrsh}(x,\lambda) = \begin{cases} x & x<-\lambda \\ 0 & -\lambda\le x\le\lambda \\ x & x>\lambda \end{cases}$



Returning to Lasso: $\quad w_d = \text{SoftThrsh}\left(\frac{c_d}{a_d}, \frac{\lambda}{a_d}\right)$

Coordinate Descent Algorithm for Lasso (§ 11.4.9.1)

we solve $\nabla\left\{\frac{1}{2}\|X\underline{w}-y\|_2^2+\lambda\|w\|_1\right\}=0$ (in the sense of subgradients) to find the optimal parameter w. Instead of solving the entire system we solve one equation at a time and iterate. This is the same idea as the Jacobi Method for linear systems. We have seen that the solution of the d-th equation is given by the Softmax function:

```
Initial guess:
    $w = (X^TX + \lambda I)^{-1} \cdot X^Ty$        (= solution of ridge regression)
    repeat
        for d=1..D
        |   $a_d = ..$  }
        |   $c_d = ..$  }  from prev. page
        |
        |   $w_d = \text{SoftThrsh}\left(\frac{c_d}{a_d}, \frac{\lambda}{a_d}\right)$
        end
    until converged.
```

(15pt)

Chap 13 Deep Neural Networks

Feature Transformation:

Example: Hooke's law $f(x,w) = w_0 + w_1 x$ is only valid for small x. To better fit the actual physics we need to add nonlinear terms.

$$f(x,w) = w_0 + w_1 x + w_2 x^2 + .. + w_p x^p$$
$$= \underline{w} \circ \Phi(x) \quad \text{where } w = \begin{bmatrix} w_0 \\ \vdots \\ w_p \end{bmatrix} \text{ and } \Phi(x) = \begin{bmatrix} 1 \\ x \\ \vdots \\ x^p \end{bmatrix} \quad \text{"feature transformation"}$$

Even though $\phi(x)$ is nonlinear, this still leads to a linear least squares problem for the parameters $\underline{w}$.

For the data $(x_n, y_n)_{n=1..N}$ we get

$$\min_{w} \frac{1}{2} \left\| \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{bmatrix} \begin{bmatrix} w_0 \\ \vdots \\ w_p \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2$$

This is very ill-conditioned when p is large. The general form is

$$(*) \qquad y = f(x,\theta) = W\phi(x) + b$$

where $\phi: \mathbb{R}^p \to \mathbb{R}^K$, $W \in \mathbb{R}^{K \times C}$, $b \in \mathbb{R}^K$ and $\theta = [W, b]$ are the parameters.

Instead of building "large" feature transformations one can iterate such functions, i.e., in the RHS of $(*)$ replace x by $f_1(x,\theta_1)$, then

$$y = f_2(f_1(x,\theta_1), \theta_2)$$

and repeating this

$$y = f_L(f_{L-1}(\dots f_1(x,\theta_1), \theta_2), \dots \theta_{L-1}), \theta_L)$$

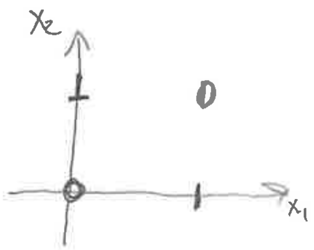
Deep Neural Nets (=DNNs) are based on this idea.

Single vs. Multilayer Perceptrons

Example Exclusive OR = XOR function

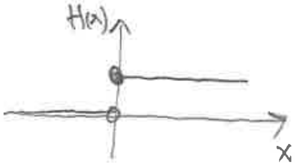
0 = False 1 = True

x_1	x_2	$XOR(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0



Heavy side fn

$$H(x) = \begin{cases} 0 & x<0 \\ 1 & x \geq 0 \end{cases}$$

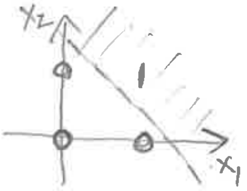


- linear (or single layer) perceptron: $f(x) = H(w^T x)$ $x, w \in \mathbb{R}^2$
  will not match the data. Cannot draw a str. line such that all $f(x)=1$ values are on one side.

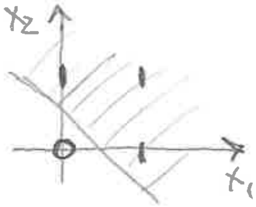
- Idea: Stack two perceptrons: h_1 = AND function
  h_2 = OR function

x_1	x_2	h_1	h_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

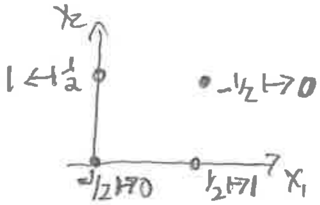
we can write h_1 and h_2 as linear perceptrons, e.g.



$h_1(x) = H(x_1 + x_2 - \frac{3}{2})$

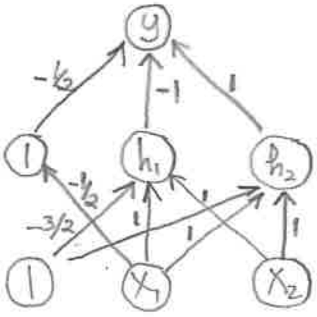


$h_2(x) = H(x_1 + x_2 - \frac{1}{2})$ $\Rightarrow$



$f(x) = H(h_2 - h_1 - \frac{1}{2})$

Draw it as a network:



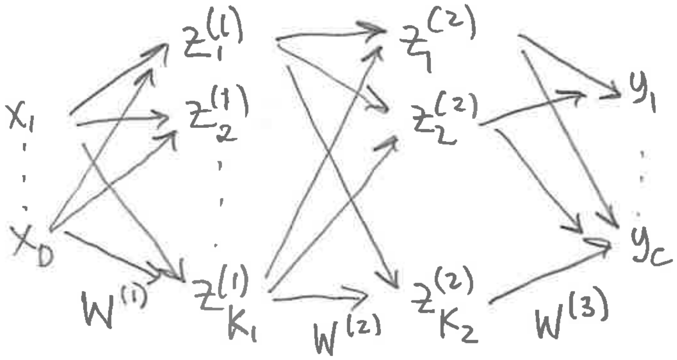
Activation function

In the previous example we used H as an activation function, but this is difficult to train, better replace by a continuous or a smooth function,

e.g. $\phi(a) = \sigma(a) = \frac{1}{1+e^{-a}}$ sigmoid fcn

or $\phi(a) = \mathrm{ReLU}(a) = (a)_+ = \begin{cases} 0 & a<0 \\ a & a\geq 0 \end{cases}$ rectified linear unit fcn.

Topology of a DNN $y = f(x,\theta)$



Shown $L=3$
2 hidden layers

$$\underline{z}^{(0)} = \underline{x}$$

$$\underline{z}^{(\ell)} = \varphi_\ell\left(\underline{b}^{(\ell)} + W^{(\ell)}\underline{z}^{(\ell-1)}\right) \qquad \ell = 1,2,\dots,L$$

$$\underline{y} = z^{(L)}$$

Componentwise: $$z_k^{(\ell)} = \varphi_\ell\Big(\underbrace{b_k^{(\ell)} + \sum_{j=1}^{K_\ell} W_{kj}^{(\ell)} z_j^{(\ell-1)}}_{a_k^{(\ell)}}\Big)$$ pre-activation

Need of an activation function

suppose we omit the activation function i.e. $\varphi_\ell(x) = x$ (or anything linear)

then
$$
\begin{aligned}
z^{(\ell)} &= b^{(\ell)} + W^{(\ell)} z^{(\ell-1)} \\
&= b^{(\ell)} + W^{(\ell)}\left(b^{(\ell-1)} + W^{(\ell-1)} z^{(\ell-2)}\right) \\
&= \hat{b}^{(\ell)} + \hat{W}^{(\ell)} z^{(\ell-2)} \qquad \hat{b}^{(\ell)} = b^{(\ell)} + W^{(\ell)} b^{(\ell-1)} \\
& \qquad\qquad\qquad\qquad\quad \hat{W}^{(\ell)} = W^{(\ell)} \cdot W^{(\ell-1)}
\end{aligned}
$$

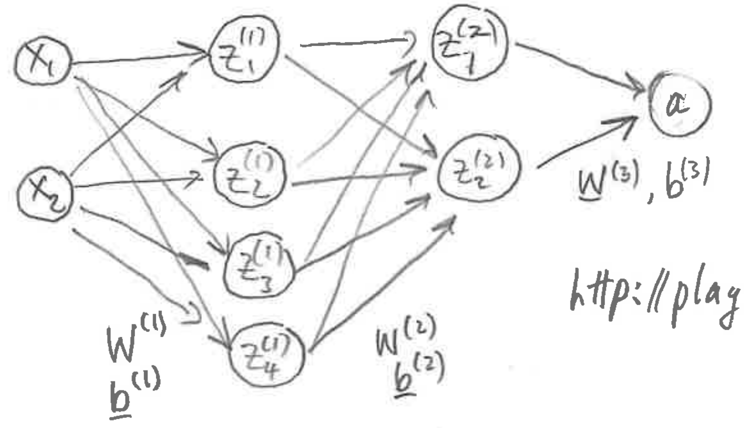
we see that all $z^{(\ell)}$'s have a linear relation, and thus there is a matrix W and a vector b such that

$$y = Wx + b$$

i.e., the model becomes linear.

Example models of DNNs

- classify 2D-data into two categories (using two hidden layers)



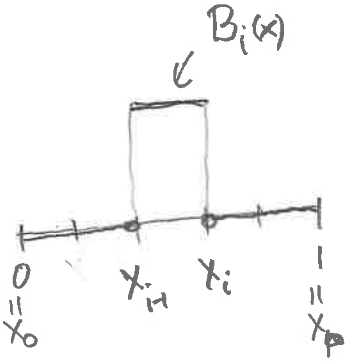
$$p(Y=y \mid X=\underline{x}) = \mathrm{Ber}(y, \sigma(a))$$

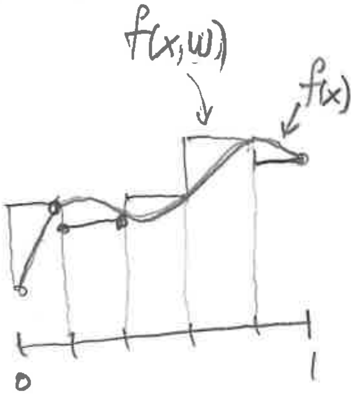
http://playground.tensorflow.org

- piecewise constant splines:

approximation of a Function $f: [0,1] \to \mathbb{R}$ by pcw. constant splines:

$$B_i(x) = \begin{cases} 1 & x_{i-1} \le x \le x_i \\ 0 & \text{else} \end{cases}$$





$$f(x,u) = \sum_{i=1}^{p} B_i(x) \cdot w_i$$

How can we choose the w_i ?

(a) Interpolation: Let $w_i = f(x_i^*)$ where x_i^* is a point $x_{i-1} \le x_i^* \le x_i$

We do this sort of thing When we derive the Riemann integral.

(b) solve a least squares problem:

choose data points $x_n \in [0,1]$ $n=1..N$ and $y_n = f(x_n)$ $n=1..N$.

Here, N is much bigger than p (= number of intervals).

Then minimize the RSS (residual square sum)

$$W = \operatorname{argmin} \sum_{n=1}^{N} \left(\sum_{j=1}^{p} B_j(x_n) W_j - y_n \right)^2$$

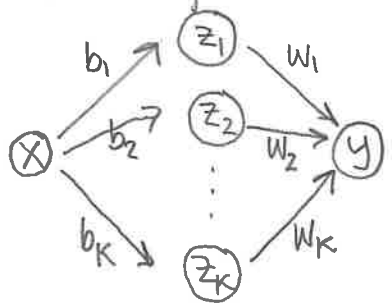
(C) Neural Network:

Note that $B_i(x) = H(x-x_{i-1}) - H(x-x_i)$ $\qquad H(\cdot)$ = Heavyside fcn.

Hence, we can think of a piecewise constant spline function as a weighted sum of shifted H-functions

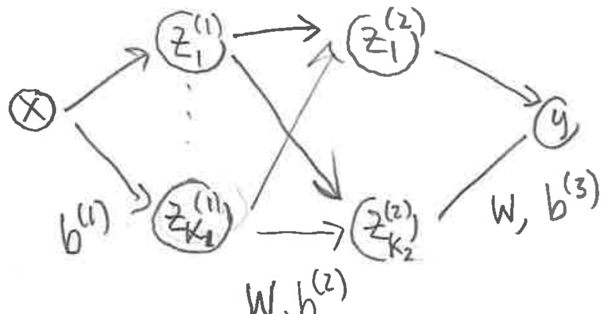
$$y = f(x,w) = \sum_{k=1}^{K} w_k H(x-b_k)$$

The evaluation of this function can be done by a neural net with one hidden layer



where $z_k = H(x-b_k)$

Now add one, or possibly more hidden layers:

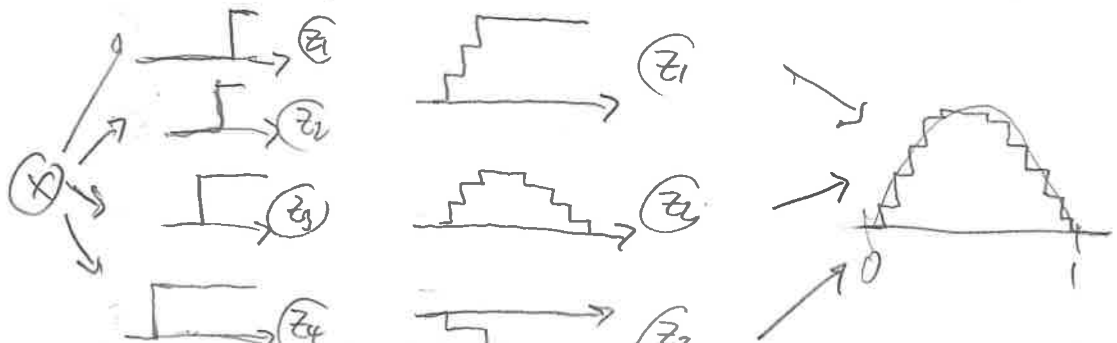


$$z_k^{(1)} = H(x - b_k^{(1)})$$

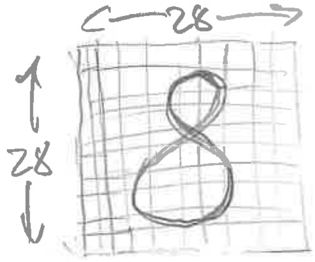
$$z_k^{(2)} = H\left(\sum_{l=1}^{K_1} W_{kl} z_l^{(1)} - b_l^{(2)}\right)$$

$$y = H\left(\sum w_l z_l^{(2)} - b_l^{(3)}\right)$$

The function $y = f(x,\theta)$ is again a pcw constant spline function. The second layer can pick up larger scale features of the function. e.g. if $f(x) = \sin(\pi x)$ then $z_k^{(1)}$ are simple step functions while $z_k^{(2)}$'s may contain the increasing and decreasing parts, and the max:



- DNN's for image classification

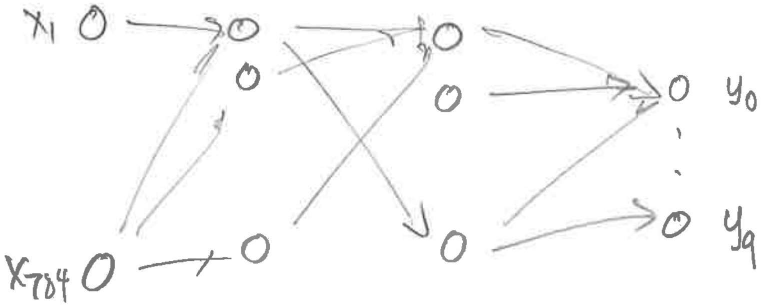


28 x 28 gray scale values

flatten into one vector:

$x = [x_{1,1} \dots x_{1,28}, x_{2,1} \dots x_{2,28}, \dots, x_{28,1} \dots x_{28,28}]$

$x \in \mathbb{R}^{784}$



(There are better ways to do this => convolutional neural networks)

§13.5 Backpropagation

Goal in this section: Compute the gradient of a loss function from a DNN.

First, we need to review some basic multivariable calculus:

Let $f: \mathbb{R}^n \to \mathbb{R}^m$, i.e. $\underline{f}(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix}$ where $x \in \mathbb{R}^n$ and $f_i(x)$ is a scalar.

↗ "Vector-valued function"

Gradient (for scalar function) $\nabla f_i(x) = \begin{bmatrix} \frac{\partial f_i}{\partial x_1} \\ \vdots \\ \frac{\partial f_i}{\partial x_n} \end{bmatrix}$ column vector with all partial derivatives of f_i

also $Df_i(x) = \nabla f_i(x)^T = \left[\frac{\partial f_i}{\partial x_1}, \dots, \frac{\partial f_i}{\partial x_n} \right]$ row vector

Jacobian $J_f(x) = \left[\frac{\partial f_i}{\partial x_j} \right]_{i,j} \in \mathbb{R}^{m \times n} = \begin{bmatrix} Df_1(x) \\ \vdots \\ Df_m(x) \end{bmatrix}$

Composition of vector valued functions, chain rule:

Let $g: \mathbb{R}^N \to \mathbb{R}^K$ and $f: \mathbb{R}^K \to \mathbb{R}^M$

then $(f \circ g)(x) = f(g(x))$

By the multivariable chain rule

$$\frac{\partial}{\partial x_j} f_i(g(x)) = \sum_{k=1}^{K} \frac{\partial f_i}{\partial y_k} \frac{\partial g_k}{\partial x_j} = \sum_{k=1}^{K} [J_f]_{ik} [J_g]_{kj}$$

$$\Rightarrow \quad J_{f \circ g} = J_f \cdot J_g$$

By induction we see that if $f = f_L \circ \dots \circ f_2 \circ f_1$

$$J_f = J_{f_L} \cdot \dots \cdot J_{f_1}$$

The matrix J_f can be either computed column-wise or row-wise.

Recall that for any matrix $A = [a_1 \cdots a_N]$ $\quad a_j = A\underline{e}_j$ and thus $A = [Ae_1, \ldots, Ae_N]$

like wise $A = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix}$ $\quad a_j^T = e_j^T A$ and thus $A = \begin{bmatrix} e_1^T A \\ \vdots \\ e_M^T A \end{bmatrix}$

<u>Forward mode differentiation:</u> (= column wise evaluation of J_f)

```
for $j = 1:n$
    $V_1^{(j)} = \underline{e}_j$
    $x_1 = x$
    for $\ell = 1:L$
        $x_{\ell+1} = f_\ell(x_\ell)$
        $V_{\ell+1}^{(j)} = J_\ell(x_\ell) \cdot V_\ell^{(j)}$
    end
end
```

$$J_f = \left[V_{L+1}^{(1)} \cdots V_{L+1}^{(n)} \right]$$

<u>Reverse mode differentiation</u>

```
$x_1 = x$
for $\ell = 1:L$
    $x_{\ell+1} = f_\ell(x_\ell)$
end

for $j = 1:m$
    $u_1^{(j)} = e_j^T$
    for $\ell = L:1$ do
        $u_{\ell+1}^{(j)} = u_\ell^{(j)} J_\ell(x_\ell)$
    end
end
```

$$J_f = \begin{bmatrix} u_{L+1}^{(1)} \\ \vdots \\ u_{L+1}^{(m)} \end{bmatrix}$$

Both algorithms accomplish the same task. The numerical cost depends on n and m ($\Rightarrow$ homework)

When training DNN's there are two types of variables X = independent variable and θ = set of parameters.

Note: in DNN's the weights between layers are matrices, but θ is always organized as a vector, by concatenating the columns of W, i.e.,

$$\theta = [\underbrace{W_{11} \cdots W_{M1}}_{\text{first col}},\ \underbrace{W_{12} \cdots W_{M2}}_{\text{2nd col}},\ \cdots,\ \underbrace{W_{1N} \cdots W_{MN}}_{\text{last col}},\ \underbrace{b_1 \cdots b_M}_{\text{bias}}]$$

With the loss function we have (say, $L=4$)

$$\mathcal{L} = \tfrac{1}{2}\|y - x_4\|_2^2$$

$$y = x_4 = f_3(x_3, \theta_3)$$

$$x_3 = f_2(x_2, \theta_2)$$

$$x_2 = f_1(x_1, \theta_1)$$

$$x_1 = x$$

$\uparrow$ evaluation of $\mathcal{L}$ goes up

differentiate with respect to the parameters

$$\frac{\partial \mathcal{L}}{\partial \theta_3} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial x_4}{\partial \theta_3} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial \theta_3}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_2} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial x_4}{\partial \theta_2} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial x_3} \cdot \frac{\partial x_3}{\partial \theta_2} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial x_3} \cdot \frac{\partial f_2}{\partial \theta_2}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial x_4}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial x_3} \cdot \frac{\partial x_3}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial x_3} \cdot \frac{\partial f_2}{\partial x_2} \cdot \frac{\partial x_2}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial x_3} \frac{\partial f_2}{\partial x_2} \frac{\partial f_1}{\partial \theta_1}$$

Mind that $\frac{\partial \mathcal{L}}{\partial x_4}$ is a row-vector and $\frac{\partial f_\ell}{\partial x_\ell}$ and $\frac{\partial f_\ell}{\partial \theta_\ell}$ are matrices.

On the right-hand sides we see that we can re-use a lot of computations.

For instance for $\frac{\partial \mathcal{L}}{\partial \theta_2}$ we set $\frac{\partial \mathcal{L}}{\partial \theta_2} = u_2 \cdot \frac{\partial f_2}{\partial \theta_2}$ where $u_2 = \frac{\partial \mathcal{L}}{\partial x_4} \cdot \frac{\partial f_3}{\partial x_3}$ (row vector times matrix)

then we can evaluate $\frac{\partial \mathcal{L}}{\partial \theta_1}$ as follows

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = u_1 \cdot \frac{\partial f_1}{\partial \theta_1} \quad \text{where} \quad u_1 = u_2 \cdot \frac{\partial f_2}{\partial x_2}$$

This explains the famous backpropagation algorithm:

```
Input $x$ and $\theta_1 .. \theta_L$
// Forward pass
$x_1 = x$
for $\ell = 1:L$
    $x_{\ell+1} = f_\ell(x_\ell, \theta_\ell)$
end
// Backward pass
$u_{L+1} = \frac{\partial \mathcal{L}}{\partial x_{L+1}}$          (typo in the book)
for $\ell = L:1$
    $g_\ell = u_{\ell+1} \frac{\partial f_\ell}{\partial \theta_\ell}(x_\ell, \theta_\ell)$
    $u_\ell = u_{\ell+1} \frac{\partial f_\ell}{\partial x_\ell}(x_\ell, \theta_\ell)$
end
```

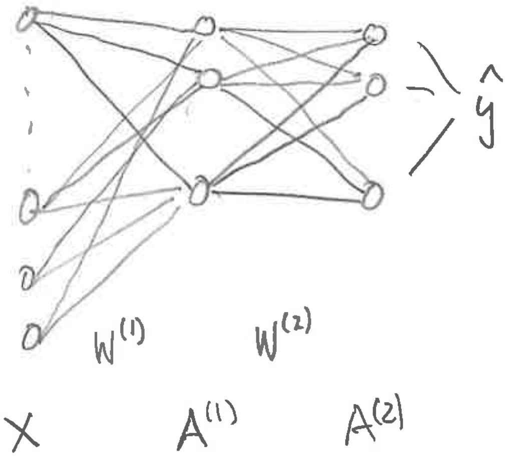
Output: $\mathcal{L} = x_{L+1}$; $D_x \mathcal{L} = u_1$; $D_{\theta_\ell} \mathcal{L} = g_\ell$ $\ell = 1:L$

13.11

Building a neural network from scratch

Youtube Samson Zhang

digit recognizer



Data

n-th image

$$X = \begin{bmatrix} & \big\downarrow & \\ \cdots & & \cdots \end{bmatrix} \quad 784 \times N$$

$$Y = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad 10 \times N$$

one-hot-vector

forward Propagation

$$Z^{(1)} = W^{(1)}X + b^{(1)} \qquad Z^{(1)} \in \mathbb{R}^{10 \times N} \qquad W^{(1)} \in \mathbb{R}^{10 \times 784}$$

$$A^{(1)} = \text{ReLU}(Z^{(1)}) \qquad A^{(1)} \in \mathbb{R}^{10 \times N}$$

$$Z^{(2)} = W^{(2)}A^{(1)} + b^{(2)} \qquad Z^{(2)} \in \mathbb{R}^{10 \times N} \qquad W^{(2)} \in \mathbb{R}^{10 \times 10}$$

$$A^{(2)} = \text{softmax}(Z^{(2)}) \qquad A^{(2)} \in \mathbb{R}^{10 \times N}$$

goal minimize

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_n \sum_k y_{kn} \log\Big(\text{Softmax}\big(\overset{\text{n-th col}}{Z_n^{(2)}}\big)\Big)$$

$$= \frac{1}{N}\sum_n \mathcal{L}_n(Z_n^{(2)})$$

where

$$\mathcal{L}_n(z) = -\sum_k y_{kn} \log\big(\text{Softmax}(z)\big)$$

Recall

$$\text{Sofmax}(z) = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}} = \frac{\exp(z_k - z_{max})}{\sum_{k'} \exp(z_{k'} - z_{max})}$$

this avoids overflow.

1.3.a

$$\mathcal{L}_n(z) = -\sum_k y_{kn} \log\left(\frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}\right)$$

$$= +\sum_k y_{kn}\left(\log \sum_{k'} e^{z_{k'}} - z_k\right)$$

$$\Rightarrow \frac{\partial \mathcal{L}_n}{\partial z_j} = \sum_k y_{jn} \frac{\partial}{\partial z_j}\left(\log \sum_{k'} e^{z_{k'}} - z_k\right)$$

$$= \sum_k y_{jn}\left(\underbrace{\frac{e^{z_j}}{\sum_{k'} e^{z_k}}}_{A^{(2)}_{jn}} - \delta_{jk}\right) = A_{jn} - y_{jn}$$

put into a matrix

$$dz^{(2)} = \left[\frac{\partial \mathcal{L}_n}{\partial z_j}\right]_{jn} = A^{(2)} - Y \qquad \in \mathbb{R}^{10 \times N}$$

Now find $\frac{\partial \mathcal{L}}{\partial W^{(2)}_{k\ell}}$:

a.) $$z^{(2)}_{jn} = \sum_i W^{(2)}_{ji} A^{(1)}_{in} + b_j$$

$$\Rightarrow \frac{\partial z^{(2)}_j}{\partial W^{(2)}_{k\ell}} = \sum_i \delta_{jk}\delta_{i\ell} A^{(1)}_{in} = \delta_{jk} A^{(1)}_{\ell n}$$

b.) $$\frac{\partial \mathcal{L}}{\partial W^{(2)}_{k\ell}} = \frac{1}{N}\sum_n \frac{\partial \mathcal{L}_n}{\partial W_{k\ell}} \overset{\substack{\text{Chain rule}\\ \downarrow}}{=} \frac{1}{N}\sum_{n,j} \frac{\partial \mathcal{L}_n}{\partial z^{(2)}_j} \frac{\partial z^{(2)}_j}{\partial W_{k\ell}}$$

$$= \frac{1}{N}\sum_{n,j} \left[dz^{(2)}\right]_{jn} \cdot \delta_{jk} A^{(1)}_{\ell n}$$

$$= \frac{1}{N}\sum_n \left[dz^{(2)}\right]_{kn} A^{(1)}_{\ell n}$$

$$\Rightarrow \underset{10\times 10}{dW^{(2)}} = \frac{1}{N} \underset{10\times N}{dz^{(2)}} \cdot \underset{N\times 10}{A^{(1)T}}$$

Now find $\frac{\partial \mathcal{L}}{\partial b_k}$

a.) $z_{jn}^{(2)} = \sum_i W_{ji}^{(2)} A_{in}^{(1)} + b_j^{(2)}$

$$\Rightarrow \frac{\partial z_{jn}}{\partial b_k^{(2)}} = \delta_{kj}$$

b.) $\frac{\partial \mathcal{L}}{\partial b_k^{(2)}} = \frac{1}{N} \sum_n \frac{\partial \mathcal{L}_n}{\partial b_k^{(2)}} \overset{\substack{\text{chain rule} \\ \downarrow}}{=} \frac{1}{N} \sum_{nj} \frac{\partial \mathcal{L}_n}{\partial z_j^{(2)}} \cdot \frac{\partial z_j^{(2)}}{\partial b_k^{(2)}}$

$$= \frac{1}{N} \sum_{nj} \frac{\partial \mathcal{L}_n}{\partial z_j^{(2)}} \cdot \delta_{kj}$$

$$= \frac{1}{N} \sum_n \frac{\partial \mathcal{L}}{\partial z_k^{(2)}}$$

$$\Rightarrow db^{(2)} = \frac{1}{N} \sum_n \underset{\uparrow \text{ nth col of } dz^{(2)}}{dz_n^{(2)}}$$

Likewise: $dz^{(1)} = W^{(2)T} \cdot dz^{(2)} .* g'(z^{(1)})$

$$dW^{(1)} = \frac{1}{N} dz^{(1)} \cdot X$$

$$db^{(1)} = \frac{1}{N} \sum_n dz_n^{(1)}$$

Chap 16 Non Parametric Methods

- parametric methods: $y = f(x, \theta)$ regression and classification

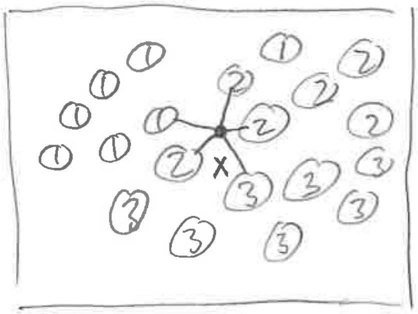
optimize θ such that there is a "best fit" $y_n \approx f(x_n, \theta)$ where $D = (x_n, y_n)$ is the data.

- non parametric method $y = f(x, D)$ There are no parameters.

f is directly constructed from the data.

16.1 K nearest neighbor classification (KNN)

Datapoints and labels in $\mathbb{R}^2$



$K = 5$

for $x \in \mathbb{R}^2$ find the 5 nearest data points, then count how many neighbors are class 1, 2, or 3

in this case $$\underline{P}(x, D) = \frac{1}{5}\begin{bmatrix} 1+0+0+0+0 \\ 0+1+1+1+0 \\ 0+0+0+0+1 \end{bmatrix} = \begin{bmatrix} 1/5 \\ 3/5 \\ 1/5 \end{bmatrix}$$ ← probabilities of class 1, 2, 3

one-hot-vectors

in general the KNN classification is

$$p(Y=c \mid x, D) = \frac{1}{K} \sum_{n \in N_K(x)} \mathbb{I}(y_n = c)$$

where $N_K(x)$ is the neighborhood of x.

$N_K(x)$ = indices of the K nearest datapoints

Distance metrics:

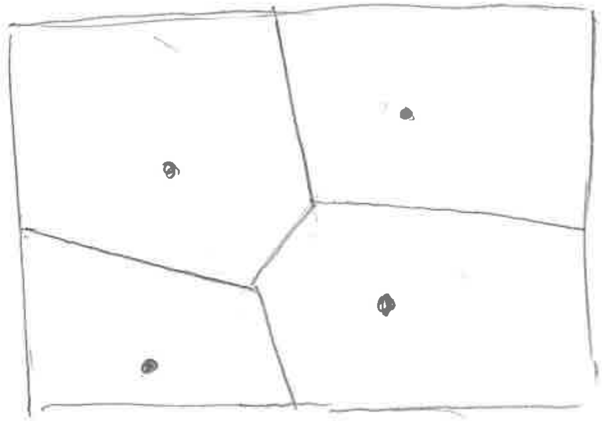
(a) $d(x_n, x) = [(x_n - x)^T M (x_n - x)]^{1/2}$ $M \in \mathbb{R}^{D \times D}$ is SPD

Mahalanobis distance

$M = I$ Euclidean distance

(b.) $d(x_n, x) = \|x_n - x\|_1$ city block distance.

An interesting case is $K=1$, and $M=I$. Here the regions that that are closest to a given data point have the same classification as the data point. Ex-ple in $\mathbb{R}^2$



Voronoi tesselation all regions are convex polytopes.

curse of dimensionality: when D is large the space gets more empty (100 points in the unit square are much denser than 100 points in the unit cube). There fore the K nearest neighbors will have much larger distances in higher dimensions.

§16.3 Kernel Density Estimation

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

gives a probability density function.

more general 1.) $K : \mathbb{R} \to \mathbb{R}^+$

2.) $\int_{\mathbb{R}} K(x)\,dx = 1$

3.) $K(x) = K(-x)$

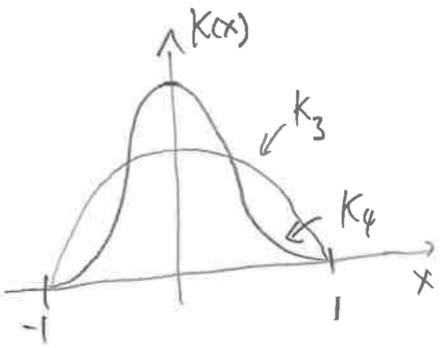
Bandwidth: $h \leftrightarrow \sigma$ $\qquad K_h(x) = \frac{1}{h} K\left(\frac{x}{h}\right)$

Examples: a.) $K_1(x) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-x^2}{2}\right)$ gives the Gaussian kernel.

b.) $K_2(x) = \frac{1}{2}\mathbb{I}(|x| \le 1)$ Box car kernel

c.) $K_3(x) = \frac{3}{4}(1-x^2)\,\mathbb{I}(|x| \le 1)$

d.) $K_4(x) = \frac{70}{81}(1-|x|^3)\,\mathbb{I}(|x| \le 1)$



Kernels are used to estimate PDF's:

Motivation: in Chap 9, (Linear Discriminant analysis) we used classification models of the form

$$p(y=c|x,\theta) = \frac{p(x|y=c,\theta)\cdot P_c}{\sum_k p(x|y=k,\theta)\cdot P_k}$$

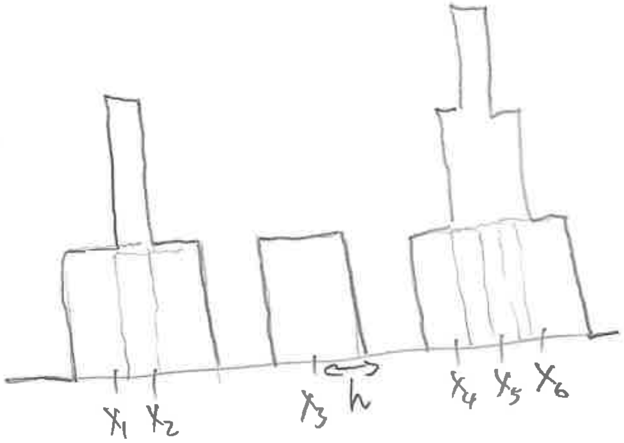
for $p(x|y=c,\theta)$ we used Gaussian distributions, where $\theta = [\mu, \Sigma]$ were approximated from the data D.

Instead of a parametric model for $p(x \mid y=c, \theta)$ we can alternatively use non-parametric PDF's.

For instance $p(x \mid D) = \frac{1}{N} \sum_{n=1}^{N} K_h(x - x_n)$

Parzen estimator or kernel density estimator. (KDE)

Examples



Box car kernel

Properties

a.) $\int_{\mathbb{R}} x K(x)\, dx = 0$ b/c parity of integrand

b.) $\int_{\mathbb{R}} x_n K(x)\, dx = x_n \cdot \int_{\mathbb{R}} K(x)\, dx = x_n$

c.) $\int_{\mathbb{R}} K_h(x)\, dx = \frac{1}{h} \int_{\mathbb{R}} K\left(\frac{x}{h}\right) dx \underset{\substack{\uparrow \\ y = \frac{x}{h} \\ dy = \frac{dx}{h}}}{=} \int_{\mathbb{R}} K(y)\, dy = 1$

KDE:

d) $\int_{\mathbb{R}} p(x, D)\, dx = \frac{1}{N} \sum_{n=1}^{N} \int_{\mathbb{R}} K_h(x - x_n)\, dx \underset{\substack{\uparrow \\ y = x - x_n}}{=} \frac{1}{N} \sum_{n=1}^{N} \underbrace{\int_{\mathbb{R}} K_h(y)\, dy}_{=1} = 1$

So KDE is a PDF.

Generalize to $\mathbb{R}^D$

$K(\underline{x}) = c_d K(\|x\|)$

Chap 21 Clustering

Recall the difference between supervised and unsupervised learning:

Supervised: given $\{X_n, y_n\}_{n=1}^{N}$ data. Find $y = f(x, \theta)$
or $y = f(x, D)$

unsupervised: find patterns in unlabeled data $D = \{X_1 \dots X_N\}$

Clustering is a form of unsupervised learning. Here we group the X_n's in such a way that objects in the same group are more similar than objects in different groups.

Examples of unlabeled data:

- Iris data set is labeled because a botanist assigned a species to each plant based on the sepal and petal dimensions. This is an example of clustering.
- Handwritten digits: Suppose we are unaware of arabic numerals and want to make sense out of the images with 0-9.

Dissimilarity metric (or proximity matrix)

$d_{ij} \geq 0$ measures the dissimilarity of data x_i and x_j.

$d_{ii} = 0$, $d_{ij} = d_{ji}$

Examples: a.) $d_{ij} = \|X_i - X_j\|_p$ when $X_i \in \mathbb{R}^D$ ($p=2$ Euclidean, $p=1$ Taxicab, $p=\infty$ or even $p=0$)

b.) For non-numeric data other metrics are possible. Eg if X_n's are strings one could set

d_{ij} = # operations to convert string X_i into X_j

Where an operation is:
- delete a character
- add a character
- replace a character

d1 p4

c.) Categorical values

$x_i \in \{$ 'motor cycle', 'passenger vehicle', 'truck', 'bus' ... $\}$

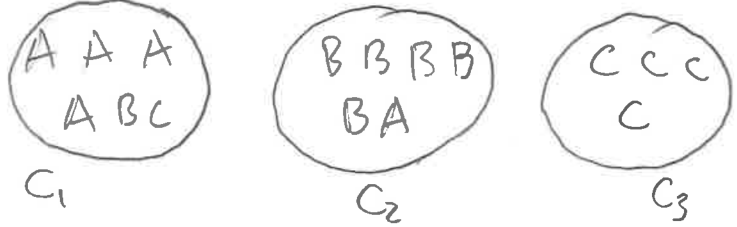
$$d_{ij} = \begin{cases} 0 & \text{if } x_i = x_j \\ 1 & \text{if } x_i \neq x_j \end{cases}$$

§ 21.1 Evaluating the output of clustering methods

To assess the quality of an clustering algorithm we can apply it to a labeled set and compare the output of the algorithm with the clustering obtained by putting the same labels into the same clusters

a.) Purity:

Exmple: labels are A, B, C; the output of the clustering algorithm is



N_{ij} = number of objects in cluster i that belong to class j

$$N = \begin{array}{c|ccc|} & A & B & C \\ C_1 & 4 & 1 & 1 \\ C_2 & 1 & 5 & 0 \\ C_3 & 0 & 0 & 4 \end{array} \begin{array}{c} \\ \longrightarrow \\ \longrightarrow \\ \longrightarrow \end{array} \begin{array}{cc} N_i & N_i/N \\ 6 & 3/8 \\ 6 & 3/8 \\ \underline{4} & 1/4 \\ 16 & \end{array}$$

N_i = number of objects in class i (=rowsum)

$P_{ij} = N_{ij}/N_i$ cond. probabilities

$$P = \begin{bmatrix} 2/3 & 1/6 & 1/6 \\ 1/6 & 5/6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{array}{c} P_i \\ 2/3 \\ 5/6 \\ 1 \end{array}$$

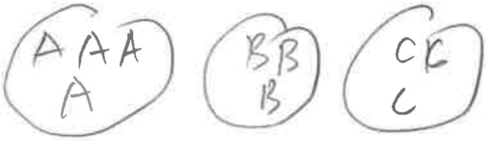
$$P_i = \max_j P_{ij}$$

$$\text{Purity} = \sum_i \frac{N_i}{N} \cdot P_i = \frac{3}{8}\frac{2}{3} + \frac{3}{8}\frac{5}{6} + \frac{1}{4} 1 = \frac{1}{4} + \frac{5}{16} + \frac{1}{4}$$

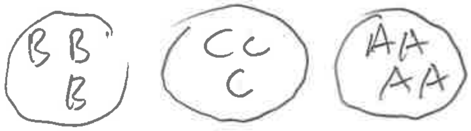
$$= \frac{13}{16}$$

$$0 < \text{purity} \leq 1$$

Exaples of purity

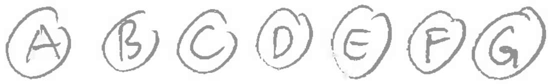


→ purity = 1



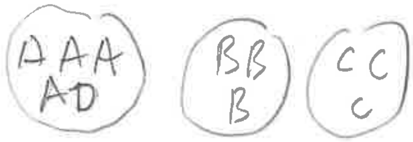
→ purity = 1

doesn't depend on the numbering of C_i



→ purity = 1

doesn't account for # clusters



→ definition of purity makes sense when #Classes ≠ #Clusters.

b.) Rand index:

let $\mathcal{U} = \{u_1 \dots u_R\}$ $V = \{v_1 \dots v_c\}$ be two partitions of the same data points. $D = \{x_1 \dots x_N\}$

for all pairs (x_i, x_j), where $i \neq j$, count

TP = # of pairs that are in the same cluster in $\mathcal{U}$ as well as V (= 1)

TN = # of pairs that are in different clusters in $\mathcal{U}$ as well as V (= 1)

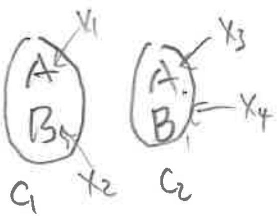
FP = # of pairs that are in the same cluster of $\mathcal{U}$, but in different clusters of V (= 0)

FN = # of pairs that are in different clusters of $\mathcal{U}$, but same cluster of V. (= 0)

Rand index $R = \dfrac{TP+TN}{TP+TN+FP+FN} = \dfrac{TP+TN}{\binom{N}{2}}$

$0 \leq R \leq 1$

Exaple



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & - & 0 & 0 & 1 \\ 2 & 0 & - & 1 & 0 \\ 3 & 0 & 1 & - & 0 \\ 4 & 1 & 0 & 0 & - \end{array}$$

1 = TP or TN
0 = else

$R = \dfrac{2}{6}$ ← only the upper half

$= \frac{1}{}$

21.2 Hieradical Agglomeration Clustering HAC

so far we have defined the dissimilarity of two data points = d_{ij}

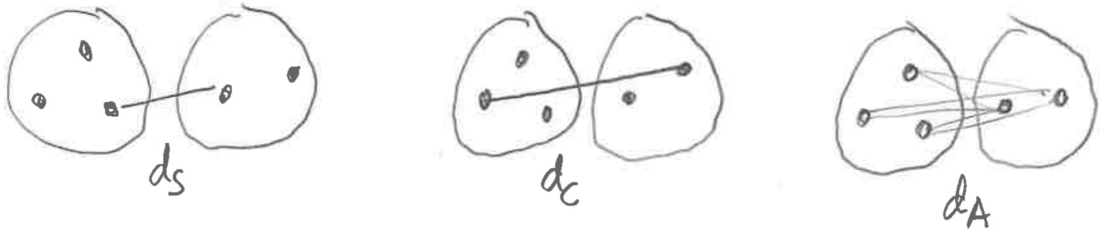
now define the dissimilarity between two clusters G, H

a.) Single link $d_s(G,H) = \min\limits_{\substack{i\in G\\ j\in H}} d_{ij}$

b.) complete link $d_c(G,H) = \max\limits_{\substack{i\in G\\ j\in H}} d_{ij}$

c.) average link $d_A(G,H) = \frac{1}{\#G} \cdot \frac{1}{\#H} \sum\limits_{\substack{i\in G\\ j\in H}} d_{ij}$

Clearly $d_s(G,H) \le d_A(G,H) \le d_c(G,H)$



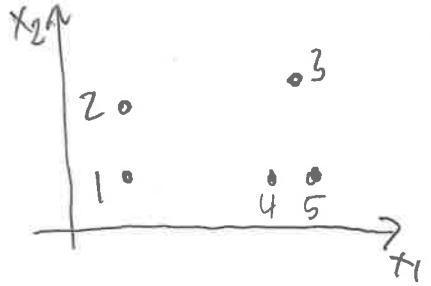
so choose one $\Rightarrow d(G,H)$

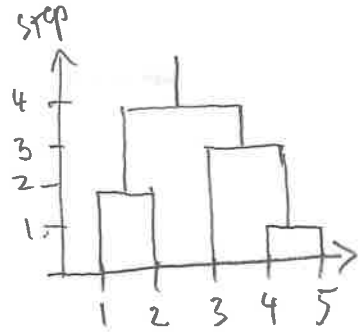
if $\#G = \#H = 1$, then $d(G,H) = d_{ij}$

Agglomerative Clustering Algorithm:

```
$C_i = \{i\}$   $i = 1..N$                          # initialize
$S = \{1,\dots,N\}$
while $\#S > 1$ :
    $(j,k) = \arg\min_{i\in S, j\in S, i\neq j} d(C_i, C_j)$        # find the two nearest clusters
    $C_j \leftarrow C_j \cup C_k$                            # merge clusters
    $S \leftarrow S \setminus \{k\}$                              # delete the 2nd Cluster
```

Ex.ple





dendro gram

at every step we obtain a clustering of the data.

In general, single link clustering merges smaller clusters first.

Complexity of hierarchical Clustering:

Count the number of comparisons. In the k-th step we have N-k clusters (here k=0 is the first step). The number of comparisons in $(j,k) = \operatorname{argmin} \dots$ is $(N-k)\cdot(N-k-1)/2 \approx (N-k)^2/2$, since we can neglect lower order terms. Adding up all steps $k=0 \dots N-1$

$$\#\text{Comparisons} \approx \frac{1}{2}\sum_{k=1}^{N-1}(N-k)^2 = \frac{1}{2}\sum_{k=1}^{N-1}k^2 \approx \frac{1}{2}\,\frac{2N^3}{6} = \frac{N^3}{6}$$

$$\uparrow$$

$$\sum_{k=1}^{N}k^2 = \frac{(2N+1)(N+1)N}{6}$$

Thus the algorithm grows rapidly with the number of data points.

§21.3 K-means Clustering

A different approach to obtain clustering of data

Definition Encoder function $Z : \underbrace{\{1, \dots, N\}}_{\text{datapoints}} \rightarrow \underbrace{\{1, \dots, K\}}_{\text{clusters}}$ (# clusters is predetermined)

where $Z(n) = k$ means the n-th datapoint belongs to cluster k

The number of different encoders = number of ways to assign datapoints to clusters grows rapidly with N and K (but is less than N^K)

Example $N = 3$, $K = 2$:

Cluster 1	Cluster 2
$\{1, 2, 3\}$	$\emptyset$
$\{1\}$	$\{2\ 3\}$
$\{2\}$	$\{1\ 3\}$
$\{3\}$	$\{1\ 2\}$

4 different ways $< 3^2 = 9$

K-means Clustering Algorithm

```
Choose cluster centers $\{\mu_k\}_{k=1..K}$          % initialization of cluster centers

while :
    $Z(n) = \underset{k \in \{1..K\}}{\arg\min} \|X_n - \mu_k\|_2$ ; $n \in \{1..N\}$     % find nearest cluster center

    $\mu_k = \frac{1}{N_k} \sum_{n: Z(n) = k} x_n$          % new cluster center = mean of points in the center
```

in Step 2 we minimize:

$$\mu_k = \arg\min \sum_{n:\, z(n)=k} \|x_n - \mu\|^2$$

because $f(\mu) = \sum_n \|x_n - \mu\|^2 = \sum_n \|x_n\|^2 - 2\mu \cdot \sum_n x_n + \|\mu\|^2 \cdot N_k$

$$\Rightarrow \nabla f(\mu) = -2\sum_n x_n + 2\mu \cdot \quad \Rightarrow \mu = \frac{1}{N_k}\sum_n x_n$$

Hence, define the distortion:

$$J(M,Z) = \frac{1}{2}\sum_{n=1}^{N} \|x_n - \mu_{z(n)}\|^2$$

where $M = [\mu_1 \ldots \mu_K] \in \mathbb{R}^{D\times K}$

Z is an encoder fcn.

The optimal clustering is the minimum of J.

Let $M^{(k)}$ and $Z^{(k)}$ be the centers and encoders in the k-th step of the K-means algorithm. Then it follows easily that

$$J(M^{(k)}, Z^{(k)}) \underset{\substack{\uparrow \\ \text{Step 1:} \\ \text{nearest} \\ \text{cluster ctr.}}}{\geq} J(M^{(k)}, Z^{(k+1)}) \underset{\substack{\uparrow \\ \text{step 2.} \\ \text{New means}}}{\geq} J(M^{(k+1)}, Z^{(k+1)})$$

So the $[M^{(k)}, Z^{(k)}]$'s are a descending sequence. This does <u>not</u> imply that the $[M^{(k)}, Z^{(k)}]$'s always converge to the optimal solution.

Stopping criterium for K-means: $Z^{(k+1)} = Z^{(k)}$

K-medoids algorithm

if we do not have the euclidean distance as the dissimilarity metric the K-medoids algorithm can be used, which only needs some d_{ij}.

Definition: The medoid of a cluster = point in cluster whose average dissimilarity with the other cluster points is minimized:

$$n = \arg\min_{i} \sum_{j} d_{ij}$$

Example

$$[d_{ij}] = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 4 \\ 2 & 4 & 0 \end{bmatrix} \begin{matrix} \to 3/3 \\ \to 5/3 \\ \to 6/3 \end{matrix}$$

so the medoid is the 1st point.

K-medoid Algorithm

```
choose $m_k \in \{1..N\}$                                        % initialize
while:
    $z(n) = \arg\min_k d_{n m_k}$    for $n \in \{1..N\}$               % find nearest medoid
    $m_k = \arg\min_{i: z(i)=k} \sum_{j: z(j)=k} d_{ij}$    for $k \in \{1..K\}$   % new cluster center = medoid of the cluster.
```