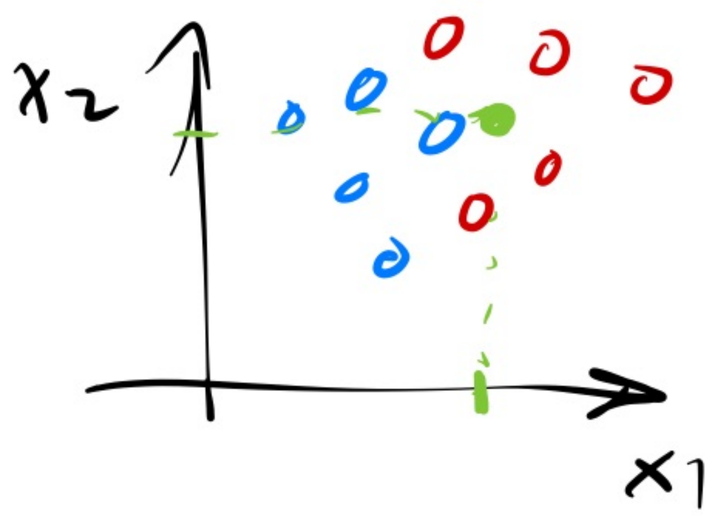
Classification

Task



} Give prediction for the class of $\vec{x}_{new}$

3 ways to do this:

1) Discriminative functions

↳ just caring about the decision boundary

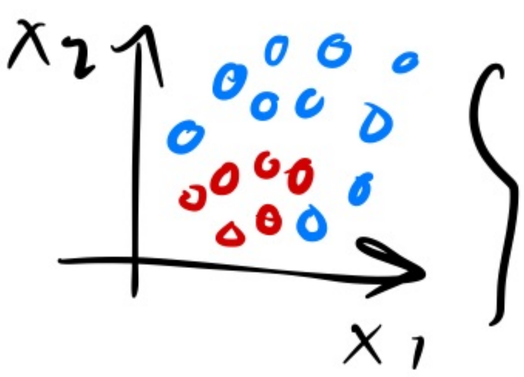
2) Probabilistic discriminative models

3) " generative models

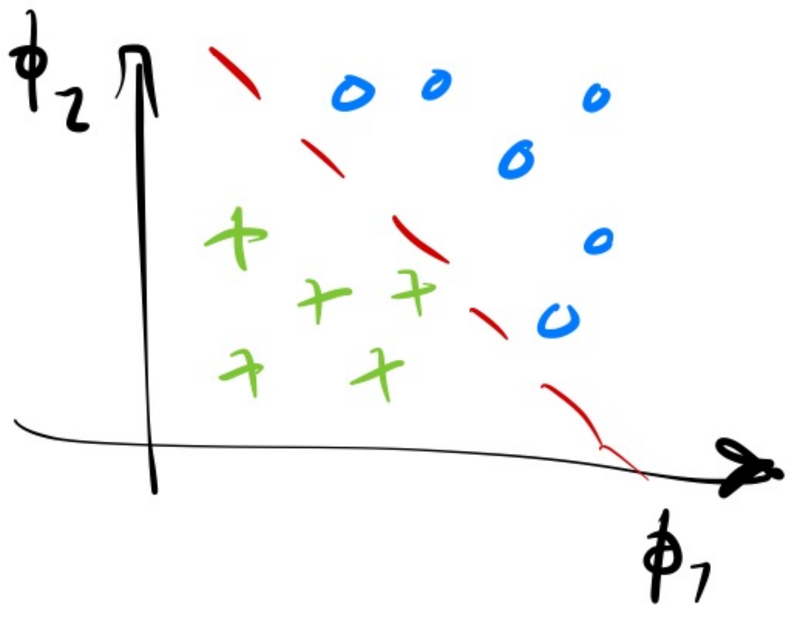
} Inference on the prob. of a given class given some $\vec{x}$

1) Discriminative functions

$$f(\vec{x}, \vec{w}) = \vec{w}^T \cdot \vec{\phi}(\vec{x})$$



} doesn't look like linearly separable



Then decide for the class with a rule like

$$C(\vec{x}) = \begin{cases} c_1 & \text{if } f(\vec{x}, \vec{w}) \geq 0 \\ c_2 & \text{otherwise} \end{cases}$$

If more than 2 classes

$$f_k(\vec{x}, \vec{w_k}) = \vec{w_k}^T \cdot \vec{x} + w_{k0}$$

for class $k = 1, \ldots, k$

s.t.

$$C(\vec{x}) = c_k \quad \text{if} \quad f_k > f_j \quad \forall j \neq k$$

How to optimise the weights? $\vec{w}$?

↳ least squares fitting?

↳ Cost function

$$\mathcal{E} = \frac{1}{2} \sum_{i=1}^{N} (\vec{t}_i - \vec{f}_i)^T (\vec{t}_i - \vec{f}_i)$$

$\vec{t}_i$ is the 1-to-K scheme for point i

↳ label e.g. (0,1,0) if point is class=2 and K=3

$$\vec{f} = W^T \cdot \vec{\phi}(\vec{x})$$

$$K * 1 = [(D+1) \times K] \times [(D+1) \times 1]$$

by minimising E wrt W:

$$W^* = (\Phi^T \Phi)^{-1} \Phi^T \cdot T$$

matrix of labels

$N \times K$

↓

in practice not giving good results in General

↓

MSE is assuming that the output is Gaussian

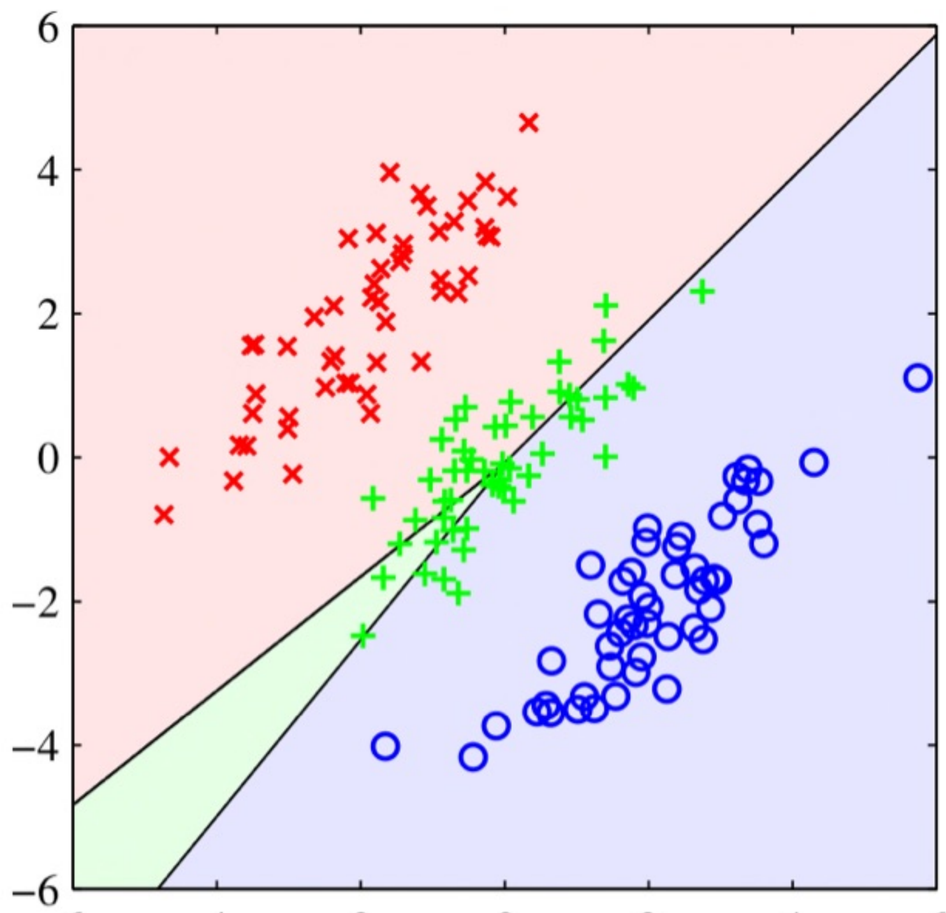


Figure 4.5 Example of a synthetic data set compri

↑

categorical variables don't follow a Gaussian

↳ (Fisher discriminant, Perceptron)

↳ better models

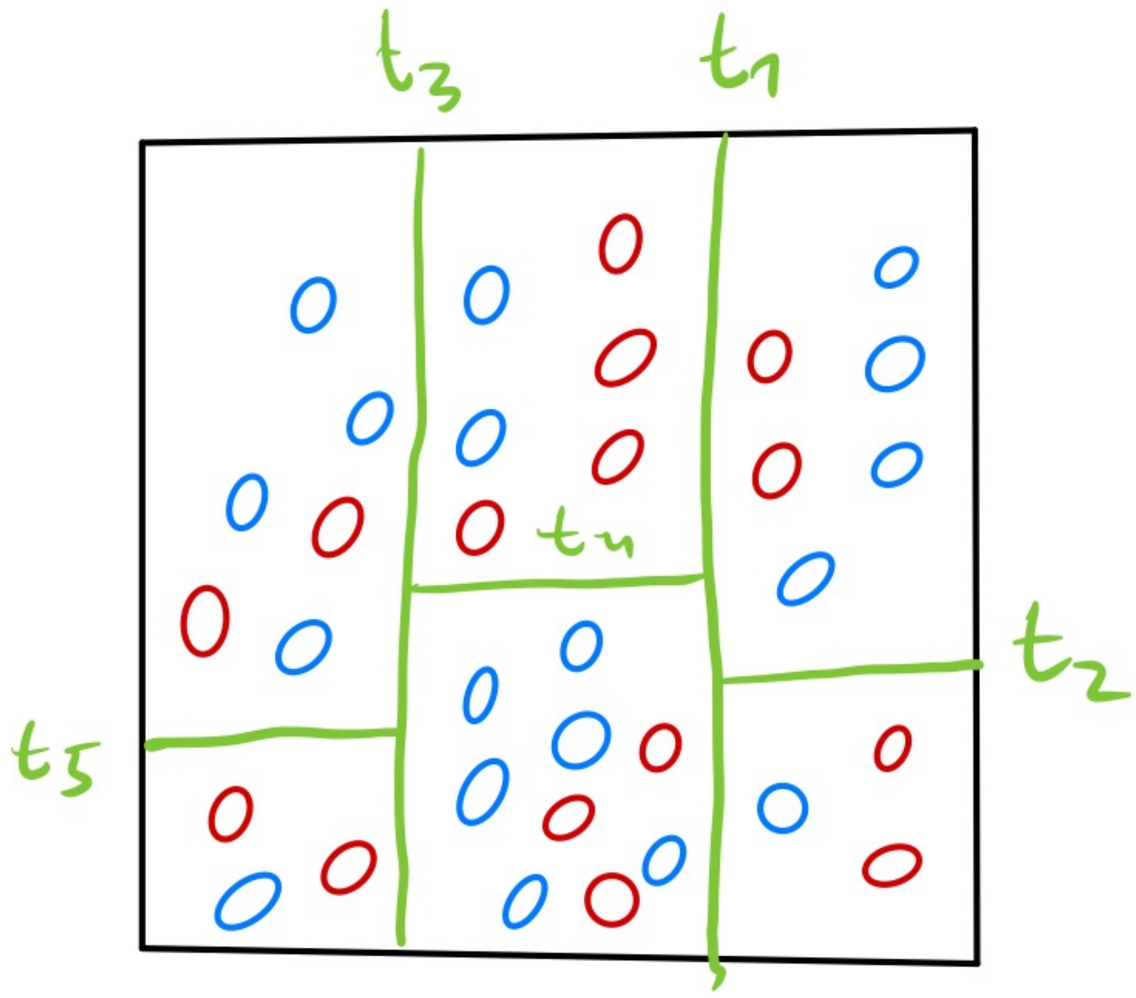
Decision Trees

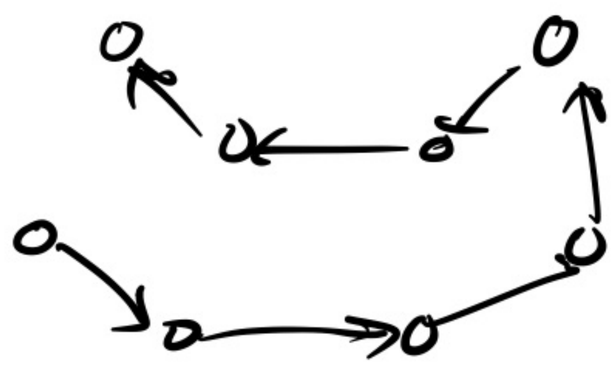
Idea: Split input space in a smart way

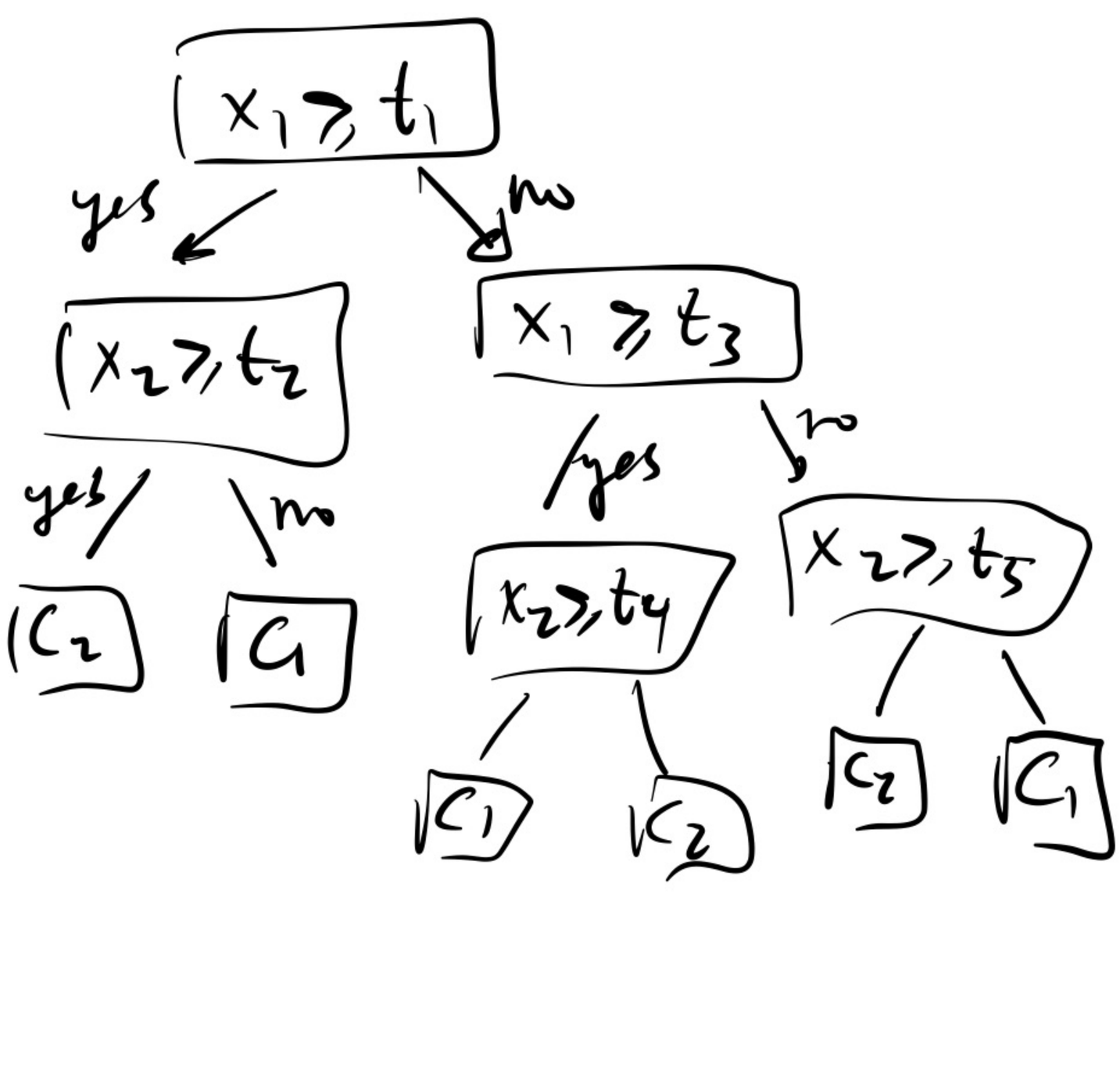
∴ in a greedy manner

x2

e.g. Travelling salesman problem







The splitting criterion has to do with some measure of "impurity"

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

↓ proportion of class k in R_m

Different ways to use it:

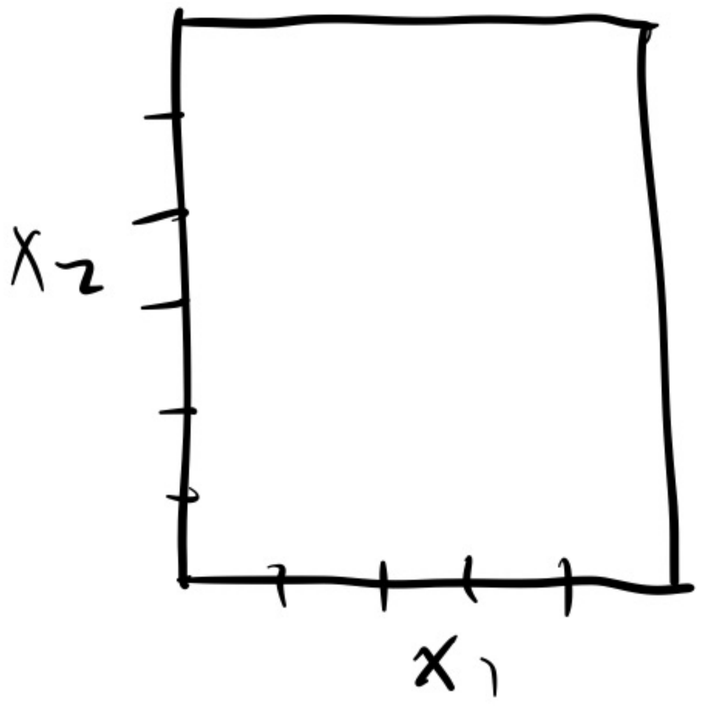
* Misclassification error

$$1 - p_{m\hat{k}}$$

$$\hat{k} = \operatorname{argmax} p_{mk}$$

* Gini-index: $\sum_{k=1}^{K} p_{mk}(1 - p_{mk})$

* Cross-entropy: $-\sum_{k=1}^{K} p_{mk} \log p_{mk}$



for 1st split:
evaluate Gini for all the possible points of the grid

Advantages ⇒ - Interpretability

- Data doesn't need to be linearly separable

Disadvantages ⇒ makes splits are orthogonal to the axes

(hard assignments of class

& (today this is improved by making probabilistic assignments)

- A single tree is likely to do overfitting

⇓

solution is to rely = ensemble methods"

(Decisions Forests e.g.)

∴ Situations where Discriminative Functions may not be convenient

1) inherently interested in prob. of given class

2) want to penalise more misclassifications in class A than in class B

⇓

design a particular loss function

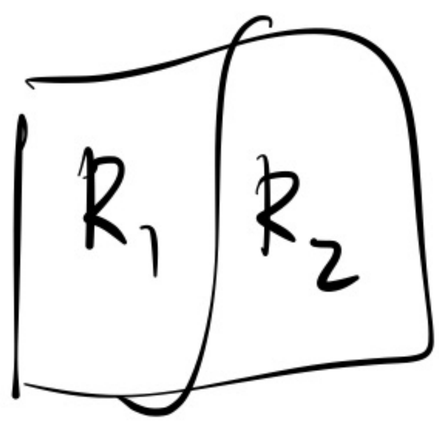
$$L : \text{true} \begin{cases} A \\ B \end{cases} \overbrace{\begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix}}^{\substack{A \quad B \\ \text{predicted class}}}$$

Minimising

$$\mathbb{E}[L] = \sum_k \sum_j \int_{R_j} L_{kj}\, p(\vec{x}, C_k)\, d\vec{x}$$

look for

$$R_j = \text{argmin}\ \mathbb{E}[L]$$



⇓

$$\hat{j} = \text{argmin} \sum_k L_{kj} \overbrace{p(C_k|\vec{x})}^{\substack{\text{object you are} \\ \text{after}}}$$

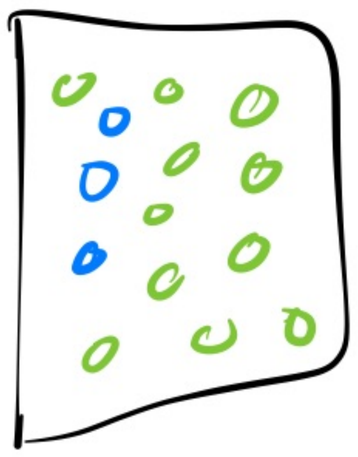
you don't need to train again if L changes

training is

- Compensate for unbalanced data

Assign priors

$p(C_k) \Rightarrow$ Fraction of points of each k



99%

1%

- Balance the data

- estimate $p(C_k|\vec{x}) \downarrow \alpha$

- correct with $p(C_k)$

Probabilistic Approach

Random binary variable t

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

(μ prob for $t=1$)

Categorical variable $\vec{t} = \{t_k\}$

$$gBern(\vec{t}|\vec{\mu}) = \prod_{k=1}^{K} \mu_k^{t_k}$$

Idea is to infer μ_k

estimate $p(C_k|\vec{x}) \Leftarrow \mu_k$

Suppose dataset $\mathcal{D} = \{\vec{x}_i, \vec{t}_i\}_{i=1}^{N}$

likelihood

$$p(\mathcal{D}|\vec{w}) = \prod_{i=1}^{N} \prod_{k=1}^{K} p(C_k|\vec{x}, \vec{w})^{t_{ik}}$$

Classif. model is a proposition for

Prob. Generative approach

$$p(C_k|\vec{x}) \leftarrow p(\vec{x}|C_k)$$

↳ likelihood of point $\vec{x}$ given C_k

⇓ via Bayes theorem

$$p(C_k|x) = \frac{p(\vec{x}|C_k)\,p(C_k)}{p(\vec{x})}$$

Consider 2 classes:

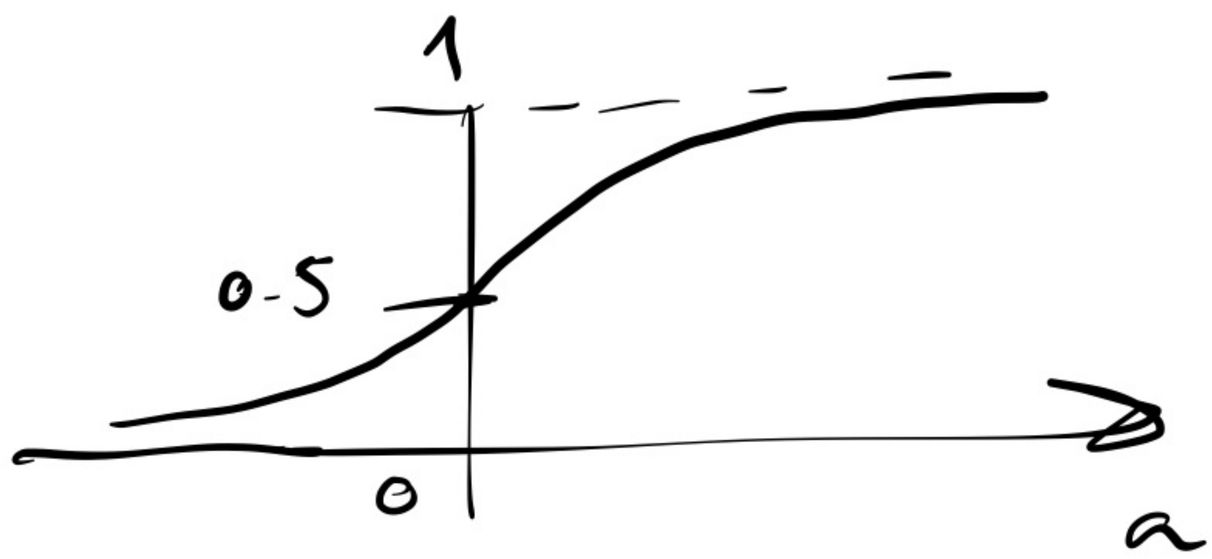
$$p(C_1|x) = \frac{p(x|C_1)\,p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)\,p(C_2)}$$

$$= \frac{1}{1 + \frac{p(x|C_2)\,p(C_2)}{p(x|C_1)\,p(C_1)}}$$

$$a \equiv \ln\left[\frac{p(x|C_1)\,p(C_1)}{p(x|C_2)\,p(C_2)}\right]$$

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

⇓ sigmoid function



For $K > 2$ classes

$$p(C_k|x) = \frac{p(x|C_k)\,p(C_k)}{\sum_j p(x|C_j)\,p(C_j)}$$

$$a_k \equiv \ln p(x|C_k)\,p(C_k)$$

$$= \frac{e^{a_k}}{\sum_j e^{a_j}} \equiv \sigma_k(\vec{a})$$

↑ softmax function

e.g. suppose $\vec{x}$ is Gaussian

$$p(\vec{x}|C_k) \propto \exp\left\{-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma^{-1} (\vec{x}-\vec{\mu}_k)\right\}$$

$\vec{\mu}_k$: characterises the class k

Σ^{-1}: assume the same $\forall k$

e.g. 2 classes

$$a = \ln p(\vec{x}|C_1) - \ln p(\vec{x}|C_2) + \ln\left[p(C_1)/p(C_2)\right]$$

$$= (\vec{\mu}_1 - \vec{\mu}_2)^T \Sigma^{-1} \vec{x} - \frac{1}{2}\vec{\mu}_1^T \Sigma^{-1} \vec{\mu}_1 + \frac{1}{2}\vec{\mu}_2^T \Sigma^{-1} \vec{\mu}_2 + \ln p(C_1)/p(C_2)$$

$$= \vec{w}^T \cdot \vec{x} + w_0$$

$$p(C_k|\vec{x}) = \sigma(a_k)$$

$$= \sigma(\vec{w}_k^T \vec{x} + w_{k0})$$

↑ implies a linear decision boundary

Now you need to find the optimum $\vec{w}$

Example of a Generative model:

Naïve Bayes classifier

Assume that, given a class C_k, each component of $\vec{x} = \{x_d\}_{d=1}^{D}$ are indep. from each other

$$p(\vec{x}|C_k) = \prod_{d=1}^{D} p(x_d|C_k)$$

↑ conditional independence assumption

$$p(C_k|\vec{x}) = \left[\prod_{j=1}^{D} p(x_j|C_k)\right] p(C_k)$$

↑ compute this for each k

for a new point $\vec{x}_{new}$ we choose the class that maximises $p(C_k|\vec{x}_{new})$

$$C(\vec{x}_{new}) = \underset{k}{\operatorname{argmax}}\ p(C_k|\vec{x}_{new})$$

Probabilistic Discriminative Models

directly inferring $p(C_k|x)$

without defining $p(x|C_k)$ first

Logistic Regression

$$p(C_k|\vec{x}) = \sigma(\underbrace{\vec{w}_k^T \vec{x} + w_0^k}_{\text{it is a linear classifier}})$$

$$p(C_k|\vec{x}) = \sigma(\vec{w}_k^T \cdot \vec{\phi}(\vec{x}))$$

To train it: minimize Cross-entropy:

$$C = -\log p(T|W)$$

↓ matrix of labels

$$= -\sum_{i=1}^{N}\sum_{k=1}^{K} t_{ik} \log \sigma(\vec{w}_k^T \vec{\phi}_i(\vec{x}))$$

$$\frac{dC}{d\vec{w}_j} = -\sum_i \sum_k t_{ik} \cdot \frac{1}{y_{ik}} \frac{dy_{ik}}{da_j} \frac{da_j}{d\vec{w}_j}$$

17W * please do it

$$= \sum_i (y_{ij} - t_{ij}) \vec{\phi}_i$$

$$= \sum_i [\sigma(\vec{w}_j^T \cdot \vec{\phi}_i) - t_{ij}] \vec{\phi}_i = 0$$

optimum $\vec{w}_j^*$ will not be analytical

$$\vec{w}_j^{\,*} = \underset{w_j}{\operatorname{argmin}}\, C(\vec{w}_j)$$ numerically

Log. Regression $\Leftarrow$ Neural network

(without hidden layer)