



SAPIENZA
UNIVERSITÀ DI ROMA

METODI DI INTELLIGENZA ARTIFICIALE E MACHINE LEARNING IN FISICA

S. Giagu - AA 2019/2020
Lezione 6: 12.3.2020

CLASSIFICATORI BAYESIANI: STIMA DELLE PDF

- con l'approccio bayesiano è possibile in teoria costruire il classificatore ottimale, pur di conoscere esattamente:
 - le probabilità a priori $P(\omega_i)$
 - le densità di probabilità condizionate alla classe $P(x|\omega_i)$
- informazioni che all'atto pratico non sono mai disponibili (in modo esatto)
- per costruire il classificatore dovremo quindi basarci su un insieme di esempi dai quali stimare le informazioni necessarie
 - stima delle $P(\omega_i)$ → normalmente facile
 - stima delle distribuzioni condizionate → più complicata, fortemente sensibile alla dimensione statistica dei training set → fattore limitante ultimo di un dato classificatore bayesiano
- esistono due approcci per la stima delle PDF:
 - **approccio parametrico:** si assume una forma particolare per le pdf e si “fittano” i parametri dal training set
 - **approccio non parametrico:** non si assume alcuna conoscenza sulla forma analitica delle pdf, che verranno stimate dai dati stessi del training set



STIMA DELLE PDF: STIMA PARAMETRICA

- si assume nota la forma delle densità condizionali
 - $P(x|\omega_i) = P(x|\theta_i)$ dove $\theta_i = \theta_i(D_i)$ è una stima del parametro θ_i basata sul campione T_i
 - spesso gli algoritmi automatici di stima assumono pdf unimodali gaussiane: $p(x|\omega_i) \sim G(\mu_i, \Sigma_i)$
 - se non lo sono è possibile in ogni caso “gaussianizzare” le distribuzioni (pre-processing) per semplificare/uniformare la realizzazione dei differenti algoritmi di classificazione (devono però sempre essere unimodali)
- tecniche più usate per la stima dei parametri:
 - Maximum-Likelihood (ML)
 - Stima Bayesiana
- NOTA: logicamente differenti, ma portano a risultati quasi identici nella maggior parte dei casi pratici



STIMA MAXIMUM LIKELIHOOD

- i valori ottimali dei parametri sono ottenuti attraverso la massimizzazione della probabilità totale di ottenere i campioni osservati
- la stima ha buone proprietà di convergenza al crescere della dimensione dell'insieme dei campioni
- concettualmente semplice (rispetto ad altre tecniche)
- principi generali:
 - assumiamo di avere C classi, con: $P(x | \omega_j) \equiv P(x | \omega_j, \theta_j)$
 - per ogni classe ω_i abbiamo un insieme di campioni T_i
 - supponiamo, per semplificare, che i campioni in T_i non diano informazioni su θ_j , assumiamo cioè che i parametri delle differenti classi siano funzionalmente indipendenti, così da poter lavorare separatamente con ciascuna classe, e quindi nel seguito omettere dalla notazione l'indice di classe
 - con tale assunzione abbiamo quindi C problemi separati della forma:
 - usare un training set T contenente n campioni x_1, x_2, \dots, x_n estratti indipendentemente in accordo alla densità di probabilità $p(x|\theta)$ per stimare il vettore di parametri incogniti θ
 - gli n campioni sono indipendenti → la probabilità totale di ottenere l'insieme T dato θ è quindi:

$$P(T | \theta) = \prod_{k=1}^n P(x_k | \theta)$$

- per definizione la stima ML di θ è il valore che massimizza $P(T | \theta)$: i.e. è il valore di θ che meglio si accorda con i campioni di training effettivamente osservati

$$\hat{\theta} = \arg \max_{\theta} [P(T | \theta)]$$

NOTA:

$$\arg \max_x f(x) := \{x \mid \forall y : f(y) \leq f(x)\}$$



STIMA MAXIMUM LIKELIHOOD

- poichè:

$$\hat{\theta} = \arg \max_{\theta} [P(T|\theta)] = \arg \max_{\theta} [\log P(T|\theta)]$$

NOTA:

$$\arg \max_x f(x) := \{x \mid \forall y : f(y) \leq f(x)\}$$

è spesso utile considerare il problema equivalente in termini di LogLikelihood:

$$\hat{\theta} = \arg \max_{\theta} \left[\log \left(\prod_{k=1}^n p(x_k | \theta) \right) \right] = \arg \max_{\theta} \left[\sum_{k=1}^n \log(p(x_k | \theta)) \right]$$

in cui si devono trattare derivate di somme invece che di prodotti

- Esempio (elementare) che già tutti conoscete dai corsi introduttivi di statistica:

- supponiamo che T contenga n campioni provenienti da una pdf gaussiana $p(x)=N(\mu, \sigma)$ con σ nota
- quale è la stima ML di μ ?

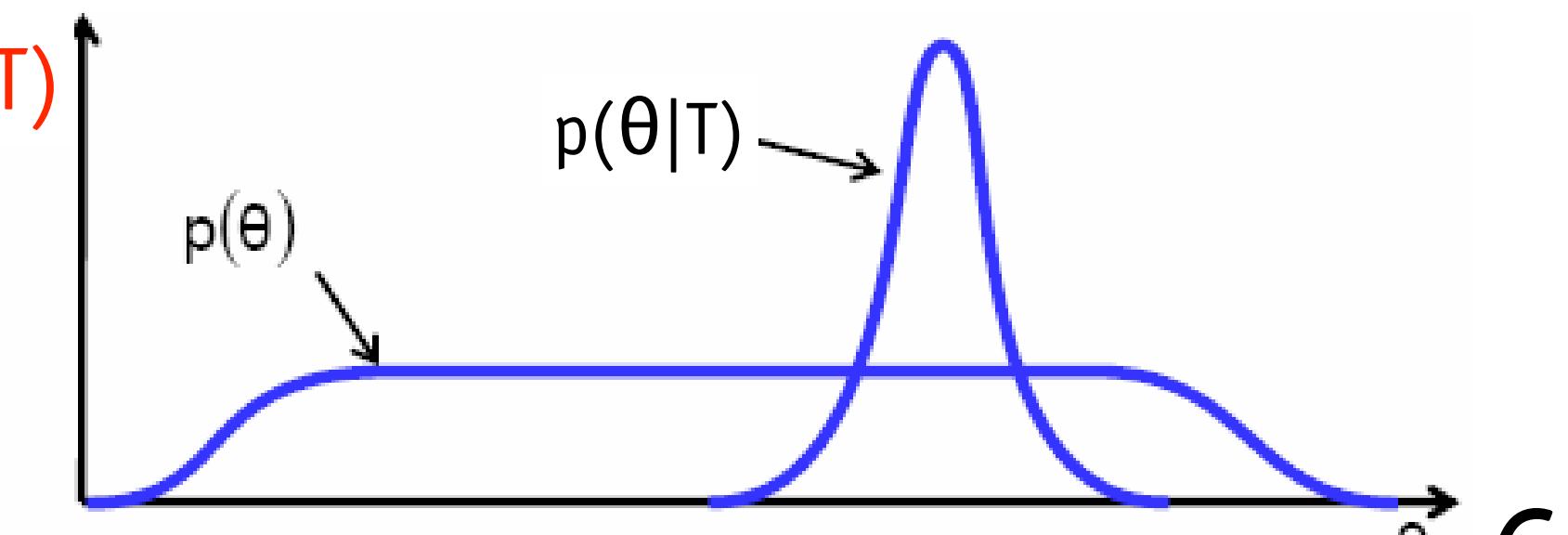
$$\begin{aligned}\hat{\theta} &= \arg \max_{\theta} \sum_{k=1}^n \log(p(x_k | \theta)) = \arg \max_{\theta} \sum_{k=1}^n \log \left(\frac{1}{\sqrt{2\pi}\sigma} \exp \left(-\frac{(x_k - \theta)^2}{2\sigma^2} \right) \right) = \\ &= \arg \max_{\theta} \sum_{k=1}^n \left[\log \left(\frac{1}{\sqrt{2\pi}\sigma} \right) - \frac{(x_k - \theta)^2}{2\sigma^2} \right] = \arg \max_{\theta} \sum_{k=1}^n \left[-\frac{(x_k - \theta)^2}{2\sigma^2} \right]\end{aligned}$$

- usuale calcolo del valore estremale: $\frac{d}{d\theta} \sum_{k=1}^n \left[-\frac{(x_k - \theta)^2}{2\sigma^2} \right] = \sum_{k=1}^n \left[\frac{x_k - \theta}{\sigma^2} \right] = 0 \Rightarrow \hat{\mu} = \hat{\theta} = \frac{1}{n} \sum_{k=1}^n x_k$



STIMA BAYESIANA

- anche nella stima bayesiana la forma di $p(x|\theta)$ è assunta nota, con θ non noto
- con una importante differenza concettuale:
 - nella stima ML il vettore dei parametri veri θ è non noto ma fisso
 - nella stima bayesiana θ è considerato una variabile aleatoria di densità nota $p(\theta)$, che può essere convertita in una probabilità a posteriori $p(\theta|T)$
 - tramite l'utilizzo delle informazioni contenute nel training sample è possibile migliorare la conoscenza della $p(\theta|T)$ tendendo a farla “addensare” intorno al valore vero dei parametri (bayesian learning).
- prima dell'osservazione dei dati, $p(\theta)$ può essere descritta per esempio da una densità a priori con un supporto molto ampio, al fine di rappresentare per esempio la scarsa conoscenza sul suo vero valore
- una volta noti i dati, sfruttando il teorema di Bayes si determina la densità a posteriori $p(\theta|T)$, ottenendo una migliore conoscenza sul parametro θ
- questa procedura permette di sfruttare in modo più efficiente l'informazione contenuta nei dati stessi
- lo scatto da pagare è una procedura più complessa e computazionalmente intensiva di quella ML



STIMA BAYESIANA

- come nella stima di ML, si suppone che $p(x|\theta)$ sia completamente specificata dati i parametri θ
- ora però θ è una variabile aleatoria con prior $p(\theta) \rightarrow p(x|\theta)$ diventa una probabilità condizionata
- quindi nella stima bayesiana la singola stima $p(x|\omega_i) = p(x|\theta_i)$ dove $\theta_i = \theta_i(T_i)$ del metodo ML viene rimpiazzata dalla stima bayesiana:

$$p(x|\omega) \doteq \int p(x, \theta|T)d\theta = \int p(x|\theta, T)p(\theta|T)d\theta = \int p(x|\theta)p(\theta|T)d\theta$$

- in cui:
 - $p(\theta|T)$ è la densità di probabilità a posteriori per θ
 - la probabilità congiunta $p(x, \theta|T) = p(x|\theta, T)p(\theta|T)$ per definizione di probabilità condizionata
 - $p(x|\theta, T)$ è indipendente da T , perché una volta noto θ la distribuzione di x è completamente nota $\rightarrow p(x|\theta, T) = p(x|\theta)$
- la probabilità a priori per θ : $p(\theta|T)$ è esprimibile tramite il teorema di Bayes come:

$$p(\theta|T) = \frac{p(T|\theta)p(\theta)}{p(T)} = \frac{p(T|\theta)p(\theta)}{\int p(T|\theta)p(\theta)d\theta} \quad \text{prob. a priori di } \theta$$

funzione di likelihood, cioè la densità di probabilità del campione T condizionata al valore dei parametri θ , che può essere calcolata assumendo gli eventi di T statisticamente indipendenti come:

$$p(T|\theta) = \prod_{k=1}^N p(x_k|\theta) \quad \{k=1, \dots, N\} \text{ su gli } N \text{ campioni di } T \text{ usati per la stima}$$



STIMA BAYESIANA VS MLE

1. in favore del MLE:

- nella stima bayesiana in principio $p(x|\omega)$ potrebbe essere calcolato analiticamente come:

$$p(x|\omega) = \int p(x|\theta) \frac{\prod_k p(x_k|\theta)p(\theta)}{\int \prod_k p(x_k|\theta)p(\theta)} d\theta$$

- in pratica l'integrazione analitica risulta quasi sempre troppo difficile e si deve ricorrere a metodi numerici
 - nel metodo MLE invece la soluzione richiede come abbiamo visto la differenziazione della funzione di likelihood e non l'integrazione, e come noto la differenziazione è un'operazione molto più semplice che può sempre essere fatta analiticamente

2. in favore della stima bayesiana:

- $p(x|\omega)$ nella stima bayesiana può essere visto come la media pesata del modello proposto su tutti i possibili valori di θ :

$$p(x|\omega) = \int p(x|\theta)p(\theta|D)d\theta$$

modello proposto peso che il modello riceve dai dati

- in contrasto con il metodo MLE in cui la soluzione ci fornisce sempre un unico modello $p(x|\theta)$
 - quando abbiamo molte possibili soluzioni, prendere la loro media pesata sembra sicuramente una scelta migliore di quella di sceglierne un'unica tra tutte ...

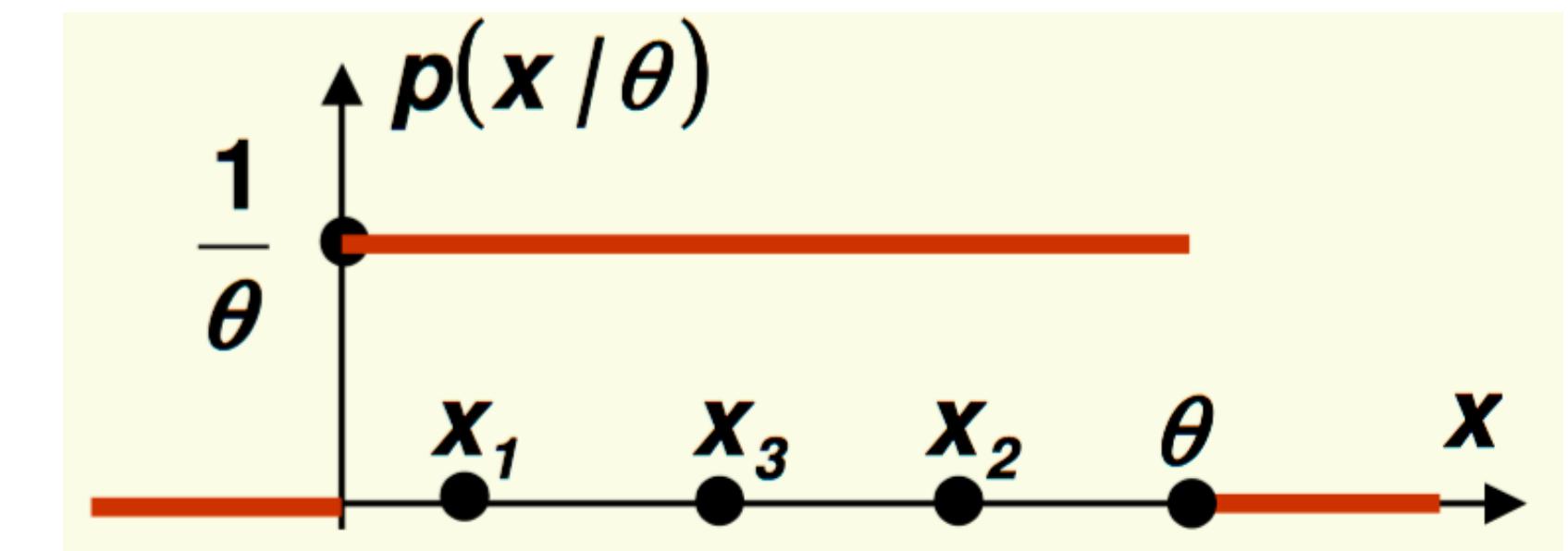


STIMA BAYESIANA VS MLE: ESEMPIO DISTRIBUZIONE UNIFORME

- supponiamo di avere una distribuzione uniforme $U[0, \theta]$

$$p(x | \theta) = \frac{1}{\theta} \text{ in } [0, \theta]$$

0 altrimenti



- calcoliamo la likelihood:

$$\begin{aligned} L(\theta) &= \prod p(x_k | \theta) = 1/\theta^n \text{ se } \theta \geq \max\{x_1, \dots, x_n\} \\ &= 0 \quad \text{altrimenti} \end{aligned}$$

- otteniamo quindi nella **stima MLE**: $\hat{\theta} = \operatorname{argmax}\{L(\theta)\} = \max\{x_1, \dots, x_n\}$



non è una stima molto attraente visto che sicuramente θ dovrebbe essere più grande di qualunque x osservato!

STIMA BAYESIANA VS MLE: ESEMPIO DISTRIBUZIONE UNIFORME

- vediamo come il problema viene trattato nella **stima bayesiana**:
 - assumiamo un **prior** $p(\theta) = U[0,10]$ che è un prior ragionevole se non conosciamo nulla su θ a parte il range di valori
 - dobbiamo calcolare $p(x|T) = \int p(x|\theta)p(\theta|T)d\theta$
- con: $p(\theta|T) = \frac{p(T|\theta)p(\theta)}{p(T)} = \frac{p(T|\theta)p(\theta)}{\int p(T|\theta)p(\theta)d\theta}$ e $p(T|\theta) = \prod_k p(x_k|\theta)$
- nell'MLE abbiamo visto che $L(\theta) = p(T|\theta) = 1/\theta^n$ se $\theta \geq \max\{x_1, \dots, x_n\}$; 0 altrimenti \Rightarrow

- $p(T|\theta) = A/\theta^n$ per $\max\{x_1, \dots, x_n\} \leq \theta \leq 10$; 0 altrimenti, con

$$A = \frac{1}{\int_{\max\{x_1, \dots, x_n\}}^{10} \frac{d\theta}{\theta^n}}$$

costante di normalizzazione

- ora abbiamo due possibilità:

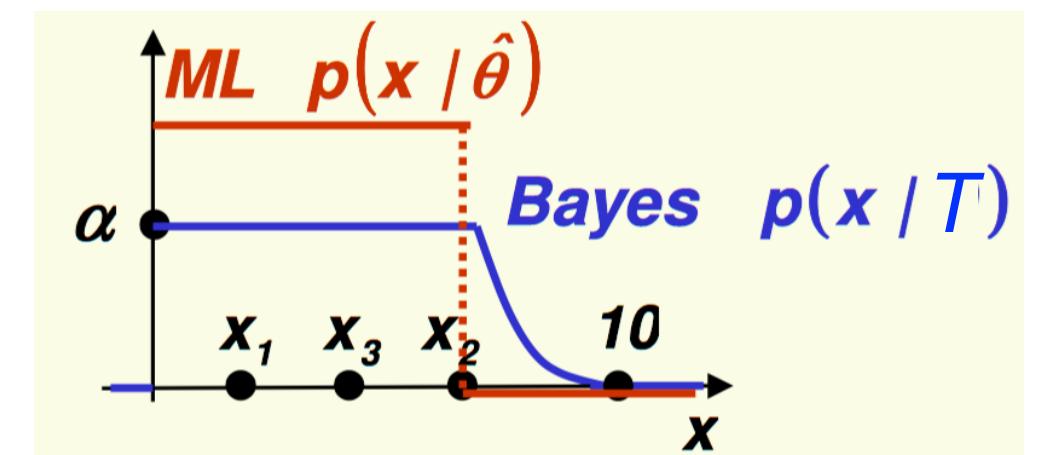
- $x < \max\{x_1, \dots, x_n\}$ $p(x|T) = \int_{\max\{x_1, \dots, x_n\}}^{10} p(x|\theta)p(\theta|T)p(\theta)d\theta = \int_{\max\{x_1, \dots, x_n\}}^{10} A \frac{d\theta}{\theta^{n+1}} = \left. \frac{A}{-n\theta^n} \right|_{\max\{x_1, \dots, x_n\}}^{10} = \frac{A}{n \max\{x_1, \dots, x_n\}^n} - \frac{A}{n 10^n} = \alpha$

- $x > \max\{x_1, \dots, x_n\}$ $p(x|T) = \int_x^{10} p(x|\theta)p(\theta|T)p(\theta)d\theta = \int_x^{10} A \frac{d\theta}{\theta^{n+1}} = \left. \frac{A}{-n\theta^n} \right|_x^{10} = \frac{A}{nx^n} - \frac{A}{n 10^n}$



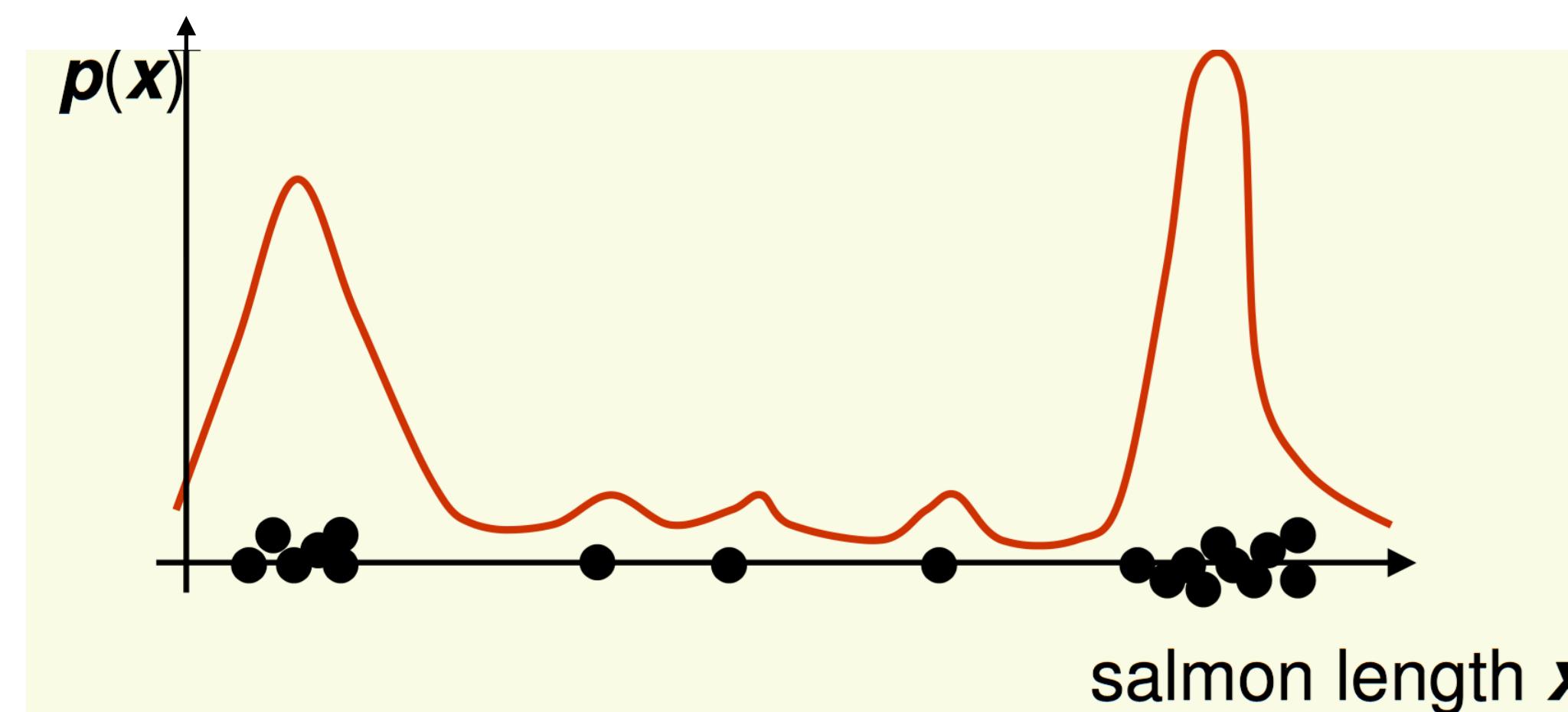
- anche oltre $x > \max\{x_1, \dots, x_n\}$ la densità bayesiana non è nulla, e questo sembra molto sensato!

- fatto curioso: la densità bayesiana non è uniforme! Non ha la forma funzionale che abbiamo assunto per il prior!



STIMA DELLE PDF: APPROCCIO NON PARAMETRICO

- le implementazioni “automatiche” dell’approccio parametrico richiedono che tutte le densità siano uni-modali (un singolo massimo locale)
 - questo non è vero in molti problemi pratici
 - nell’ approccio non parametrico si rimuove l’assunzione di conoscenza della forma delle densità di probabilità, per cui si può lavorare con distribuzioni di forma arbitraria (uni e multi-modali)
- unica assunzione: si conosce la classe di appartenenza di ciascun campione di dati usato per stimare le PDF
- L’idea di base della maggior parte dei metodi non parametrici è concettualmente molto semplice:
 - più dati si addensano in una certa regione, più grande è la densità di probabilità in quella regione

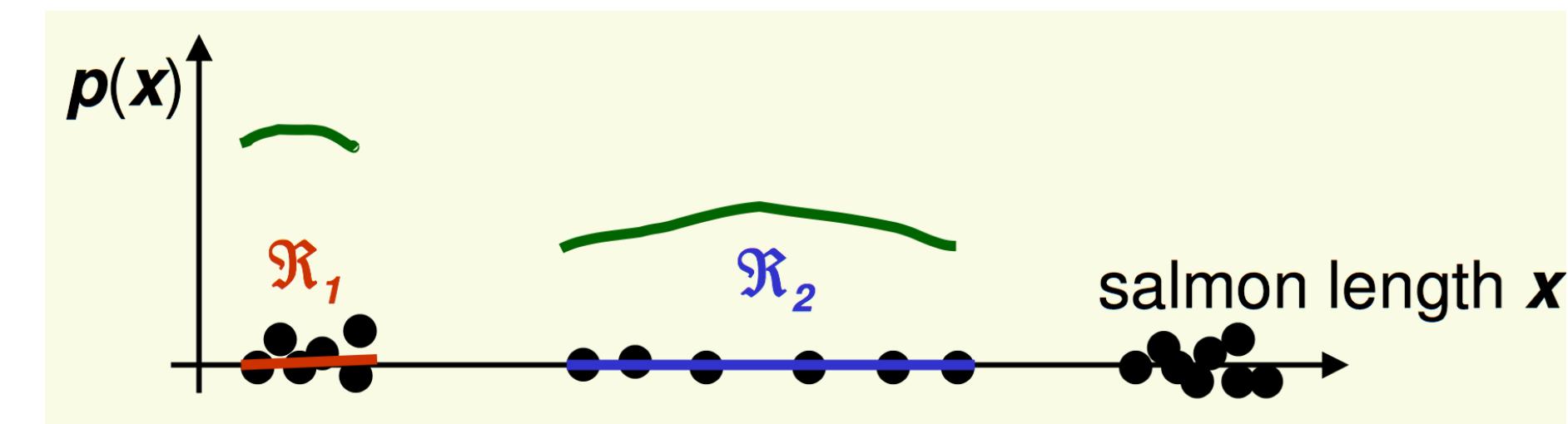


$$P(x \in R) = \int_R p(\xi) d\xi$$

P = probabilità che $x \in R$
→ media di $p(x)$ su R

STIMA DELLE PDF: APPROCCIO NON PARAMETRICO

- come possiamo stimare $P(x \in R_1)$ e $P(x \in R_2)$?



- contando il numero di eventi nel campione in R_1 e R_2 : $P(x \in R_1) \sim 6/20$, $P(x \in R_2) \sim 6/20$

- dal punto di vista delle densità di probabilità in R_1 e R_2 , ci aspettiamo abbiano la stessa altezza?
 - ovviamente no, visto che la regione R_1 è più piccola di R_2 :

$$P(x \in R_1) = \int_{R_1} p(\xi) d\xi \simeq \int_{R_2} p(\xi) d\xi = P(x \in R_2)$$

- per stimare la densità è quindi necessario normalizzare per la dimensione della regione

- assumendo $p(x)$ circa piatta in R possiamo approssimare la pdf $p(x)$ nel punto x in R come:

$$\frac{\# \text{eventi in } R}{\# \text{totale eventi}} \simeq P(x \in R) = \int_R p(y) dy \sim p(x) \times \text{Volume}(R)$$

$$p(x) \sim \frac{\# \text{eventi in } R}{\# \text{totale eventi}} \frac{1}{\text{Volume}(R)}$$



DERIVAZIONE FORMALE

- supponiamo di estrarre n campioni statisticamente indipendenti dalla distribuzione $p(x)$: x_1, \dots, x_n , la probabilità che k di questi n campioni siano contenuti in R è data dalla legge binomiale:

$$P_k = \binom{n}{k} P^k (1-P)^{n-k}$$

⇒ il valore atteso per k sarà: $E[k] = nP$ e la varianza $\text{Var}[k] = nP(1-P)$

- la distribuzione binomiale per k è fortemente piccata intorno al valore medio, per cui possiamo assumere che il rapporto k/n fornisca una buona stima della probabilità P , stima che inoltre migliorerà sempre più al crescere di n

- dim:

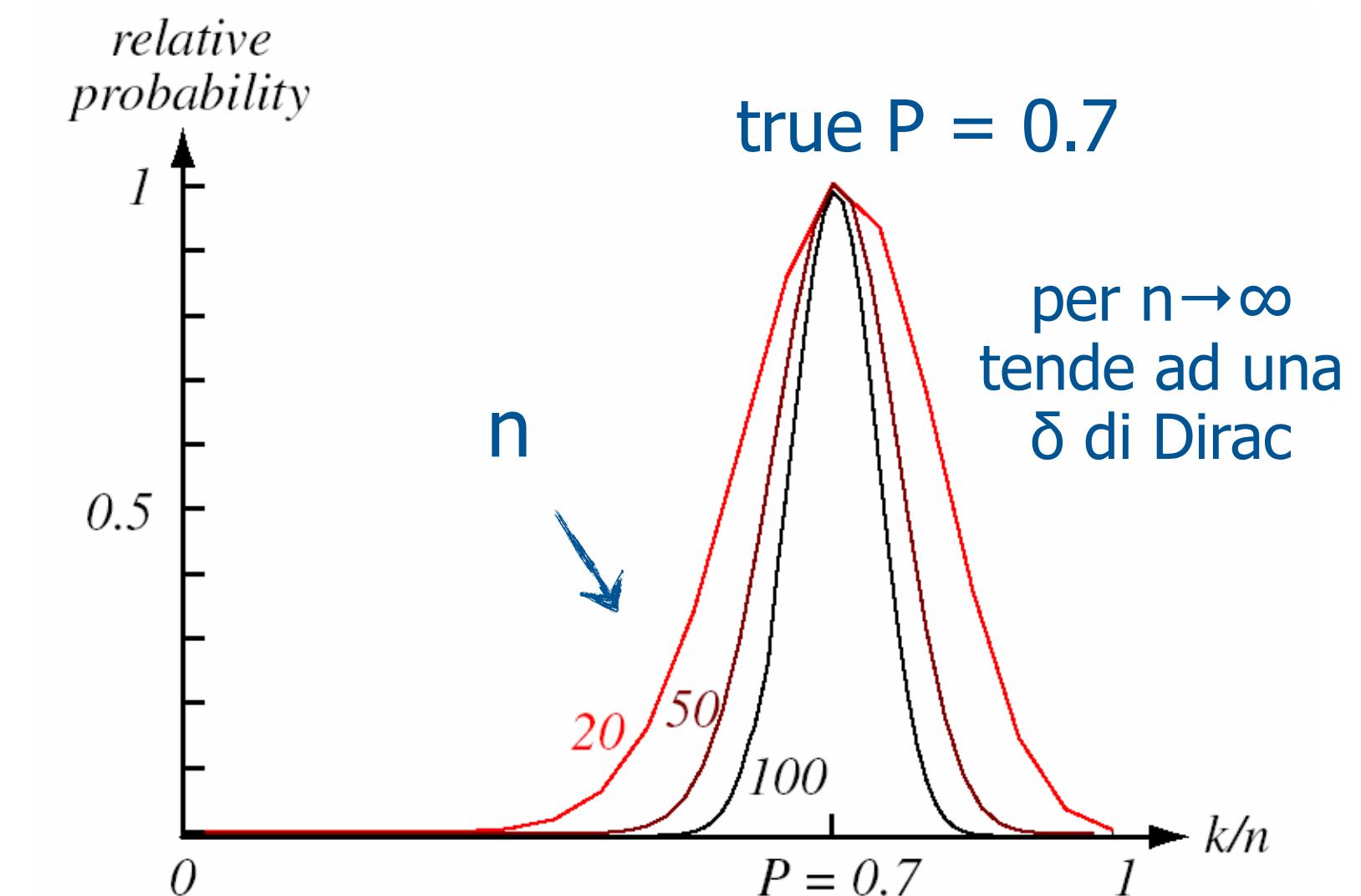
- $E[k/n] = nP/n = P$ $\text{Var}[k/n] = E[(k/n - P)^2] = P(1-P)/n$

- se assumiamo ora $p(x)$ continua e R sufficientemente piccola per cui p non vari apprezzabilmente in essa:

$$P = \int_R p(\xi) d\xi \cong p(x) \int_R d\xi = p(x)V$$

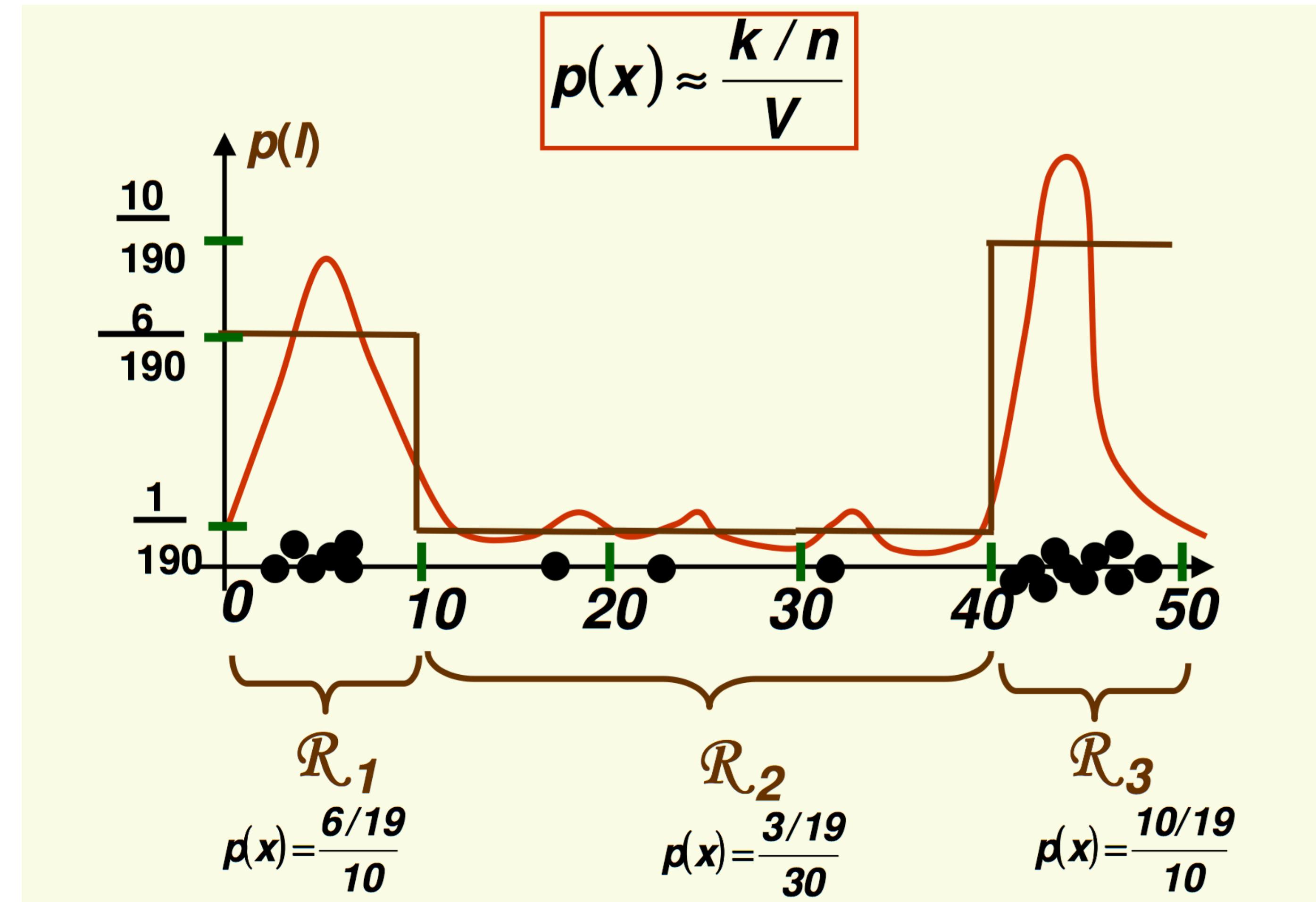
- in cui V è il volume (iper-volume) racchiuso da R
- in questo modo, otteniamo la stima di $p(x)$ data da:

$p(x) \sim (k/n) / V$



questa è la stessa formula trovata prima!

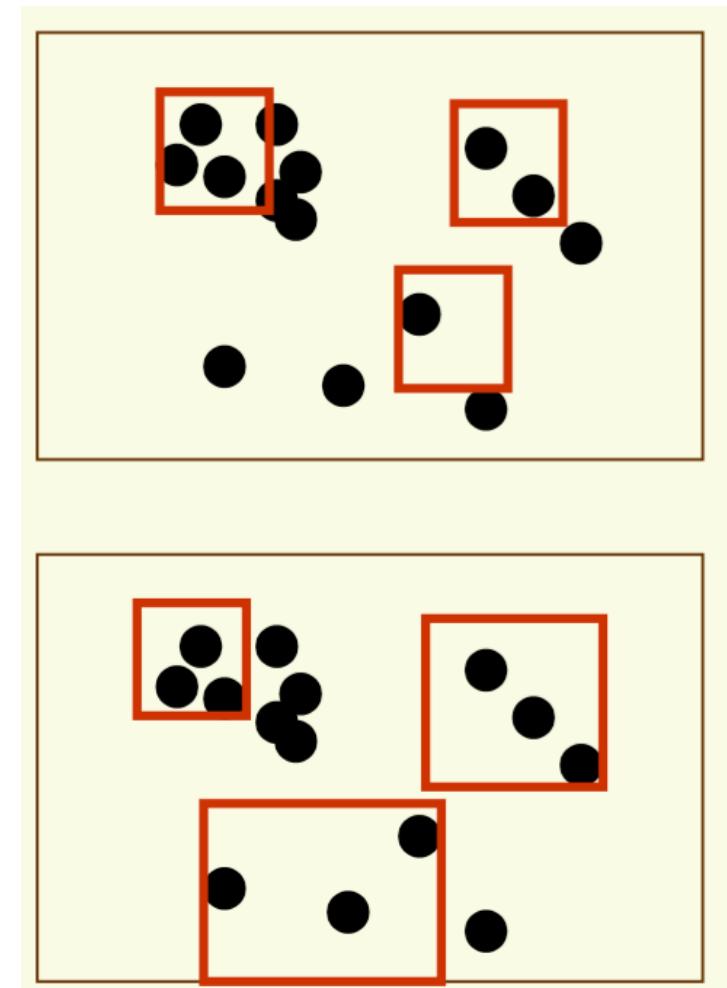
STIMA PDF NON PARAMETRICA: ESEMPIO



NOTA: se le regioni R_i non si sovrappongono, quello che otteniamo come stima di $p(x)$ è sostanzialmente un istogramma ...

LIMITAZIONI NELLA STIMA NON PARAMETRICA

- quanto è accurata la stima di $p(x)$ ottenuta in questo modo?
 - in pratica stiamo ottenendo una media di $p(x)$ sul volume della regione R a cui x appartiene
 - se vogliamo ottenere $p(x)$ invece che la media, dobbiamo prepararci a considerare un volume V che tende a zero
 - tuttavia per n fissato, se facciamo tendere V a zero, la regione R diviene così piccola da arrivare a non contenere nessun punto del campione $\Rightarrow k=0 \Rightarrow p(x)=0$ con una stima quindi inutile
- visto che i campioni di training hanno sempre dimensione finita V non potrà diventare mai arbitrariamente piccolo
- dovremo accettare l'idea di avere un certo grado di varianza nel rapporto k/n ed un certo livello di "media" nella stima di $p(x)$
- da un punto di vista teorico è possibile dimostrare che esistono due metodi distinti che convergono alla vera $p(x)$ nel limite di campioni infiniti:
 - **metodo dei Kernel (KDE algorithm):**
 - si sceglie un volume $V(n)$ fissato e si determina il corrispondente valore di k dai dati
 - **metodo della stima a k -vicini (k -nearest neighbors algorithm):**
 - si sceglie un valore di $k(n)$ fissato e si determina il corrispondente volume V dai dati



STIMA DELLE PDF: METODO DEI KERNEL

$p(x)$: pdf da stimare con $x=(x_1, \dots, x_N)$: insieme di osservazioni sperimentali di p

distribuzione cumulativa di p :

$$F(x) = P(X \leq x) = \int_{-\infty}^x p(y)dy$$

una stima della probabilità F è ottenibile contando il numero di punti $x_i : x_i \leq x$:

$$F(x) \sim \hat{F}(x) \doteq \frac{1}{N} \#\{x_i | x_i \leq x\}$$

poiché $p(x) = F'(x) \Rightarrow \exists h > 0 \in \mathbb{R}$:

$$p(x) \sim \frac{1}{2h} (\hat{F}(x+h) - \hat{F}(x-h)) = \frac{1}{2Nh} \#\{x_i | -h < x_i - x \leq h\} \doteq \hat{p}(x)$$

sia ora per esempio K è la funzione di distribuzione uniforme nell'intervallo $x \in [0,1]$:

$$K(x) = \begin{cases} \frac{1}{2} & -1 \leq x < 1 \\ 0 & \text{altrimenti} \end{cases}$$

possiamo riscrivere la stima di $p(x)$ come:

$$\hat{p}(x) = \frac{1}{Nh} \sum_{i=1}^N K\left(\frac{x - x_i}{h}\right)$$

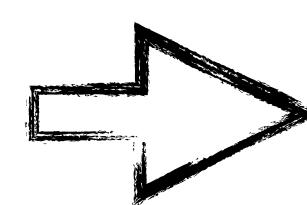
stima basata sul kernel K della pdf $p(x)$ con parametro di smoothing h



STIMA DELLE PDF: METODO DEI KERNEL

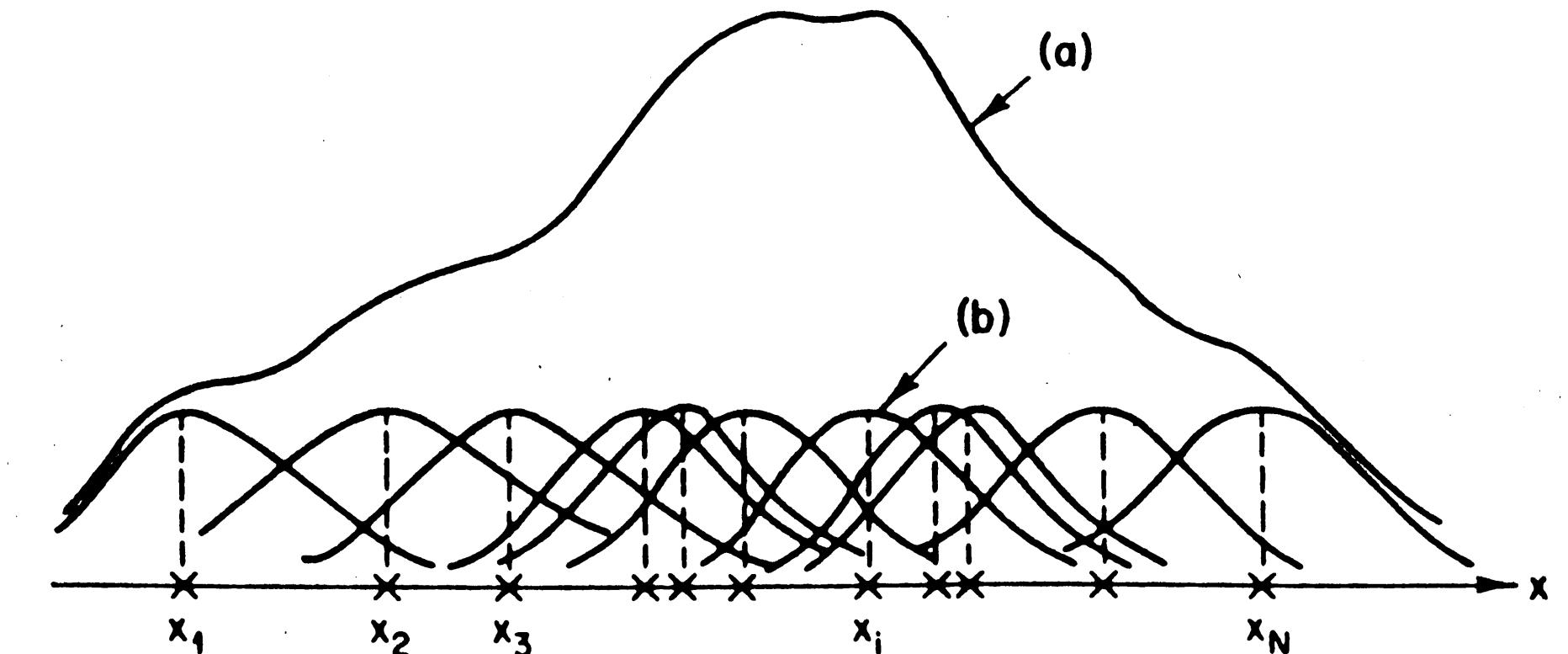
Nel caso generale K può essere una qualunque funzione, la cui scelta dipende dalle caratteristiche del problema che si vuole risolvere: ex. kernel gaussiano:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$



$$\hat{p}(x) = \frac{1}{Nh} \sum_{i=1}^N K\left(\frac{x - x_i}{h}\right)$$

stima basata sul kernel K della pdf $p(x)$ con parametro di smoothing h



l'ottimizzazione della stima consiste nella scelta del parametro di smoothing ottimale:

- h troppo piccolo \rightarrow si creano "features" inesistenti,
- h troppo grande \rightarrow si perdono dettagli (i.e. potere) della distribuzione

soluzione: **smoothing adattivo**: $h=h(x)=h/\sqrt{p(x_i)}$ \rightarrow gaussiane più larghe sulle code della pdf

STIMA DELLE PDF: KERNEL MULTIDIMENSIONALI

generalizzazione del caso 1-D

d = dimensione vettore features

$$\hat{p}(\underline{x}) = \frac{1}{Nh_1 \dots h_d} \sum_{i=1}^N \left\{ \prod_{j=1}^d K \left(\frac{x_j - x_{ji}}{h_j} \right) \right\}$$

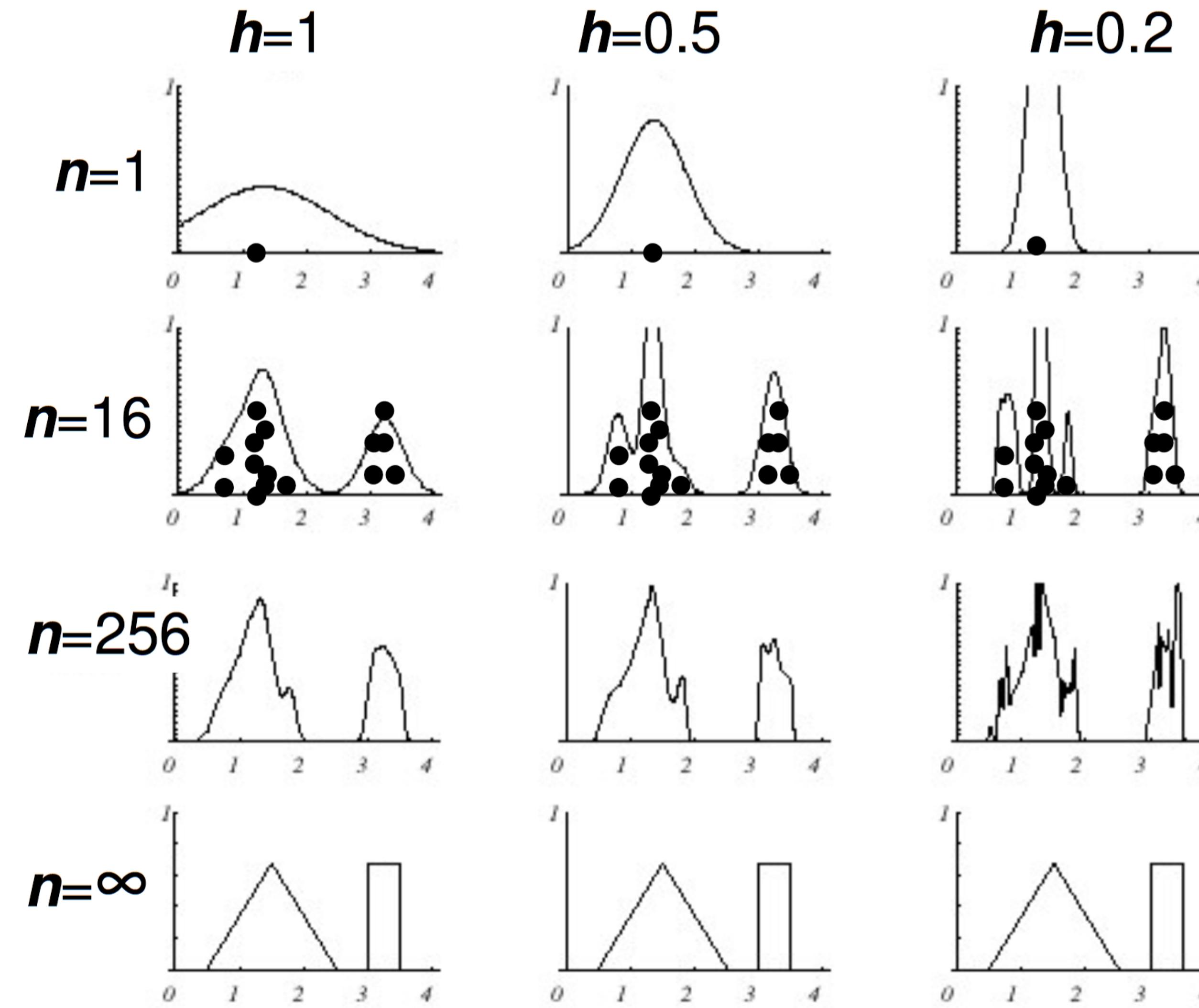
- per esempio un prodotto di distribuzioni gaussiane (ma è possibile usare kernel diversi per ciascuna variabile al prezzo di una maggiore complessità computazionale)

NOTA: questo metodo trascura le correlazioni tra le osservabili!



STIMA DELLE PDF: METODO DEI KERNEL

Esempio: pdf vera = mistura di distribuzione uniforme + dist. triangolare



NAIVE BAYES CLASSIFIER

il più semplice algoritmo di classificazione basato sulla stima (parametrica o non parametrica) delle pdf assume non vi sia nessuna correlazione tra le features

- stima della pdf per ogni singola variabile: p_k
- $p_{\text{tot}} = \prod p_k \leftarrow \text{ignora le correlazioni}$
- combina $p_{\text{tot}}(S)$ e $p_{\text{tot}}(B)$ in un discriminante tipo LikelihoodRatio (LR)

comportamento ottimale solo in caso di correlazione zero o lineare (decorrelazione)

$$y_L(i_{\text{event}}) = \frac{\prod_{k \in \{\text{variables}\}} p_k^{\text{signal}}(x_k(i_{\text{event}}))}{\sum_{U \in \{\text{species}\}} \left(\prod_{k \in \{\text{variables}\}} p_k^U(x_k(i_{\text{event}})) \right)}$$

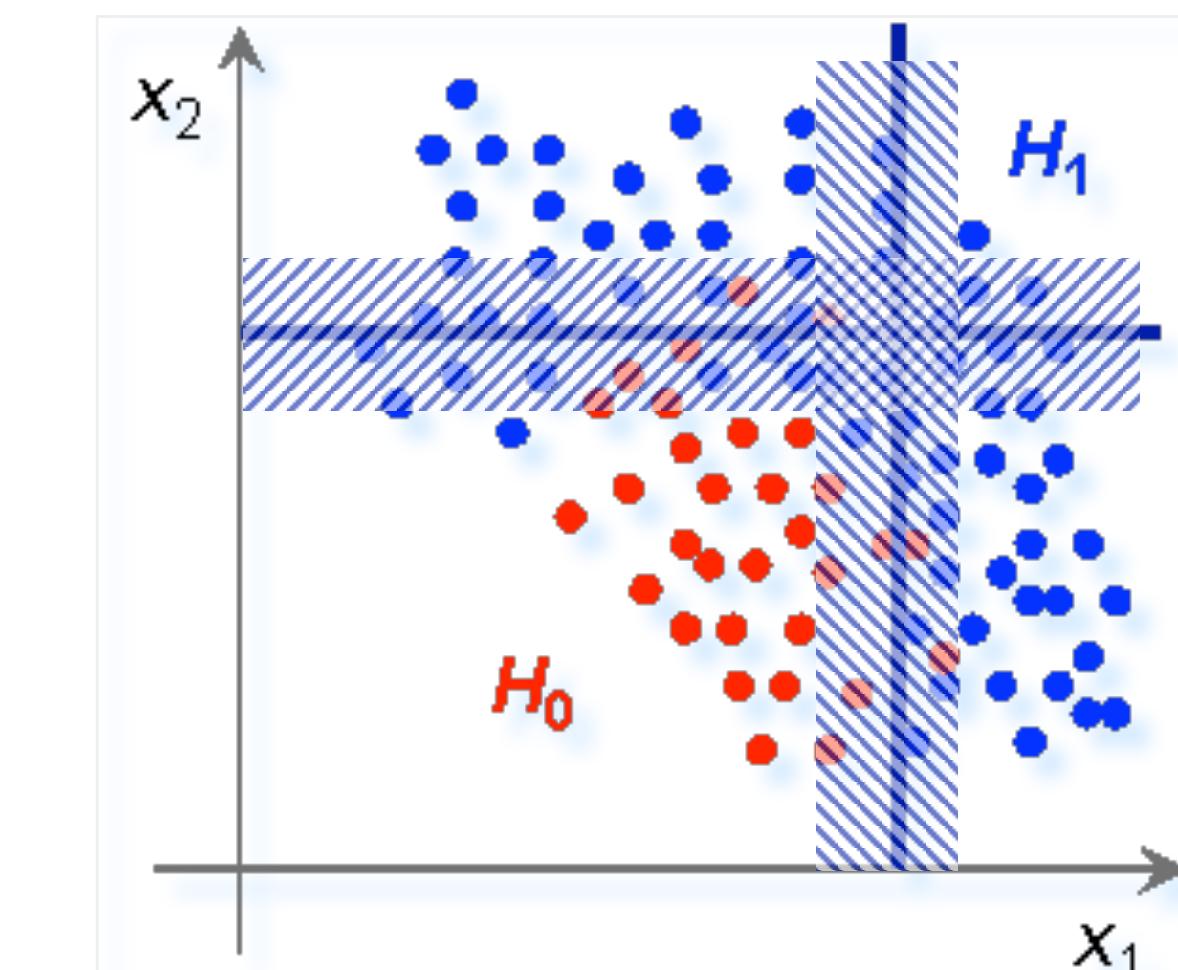
Likelihood ratio for event i_{event}

PDFs

discriminating variables

Species: signal, background types

2-classi: $C_1 \equiv S$ e $C_2 \equiv B$



NBD introduce una fuzzy logic nella definizione delle regioni di decisione

nel caso di stima delle pdf basata su kernel, l'ottimizzazione corrisponde a scegliere il parametro di smoothing che fornisce per esempio la migliore separazione S/B o la più bassa rate di errori di classificazione etc...

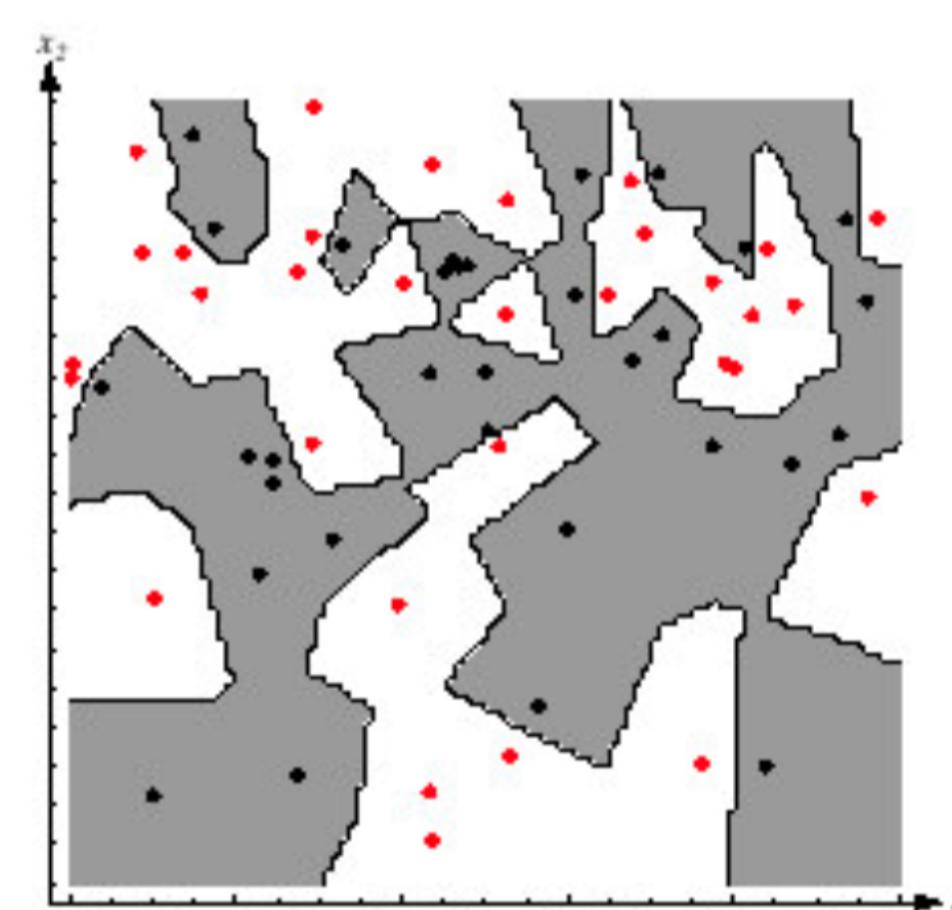


NAIVE BAYES: VANTAGGI E SVANTAGGI

- Vantaggi:
 - può essere applicata a dati provenienti da qualsiasi distribuzione
 - semplice e intuitivo: non è una black-box
- Svantaggi:
 - per campioni di dimensioni limitate (e multidimensionali) → **diventa difficile scegliere h (over/under fitting)**

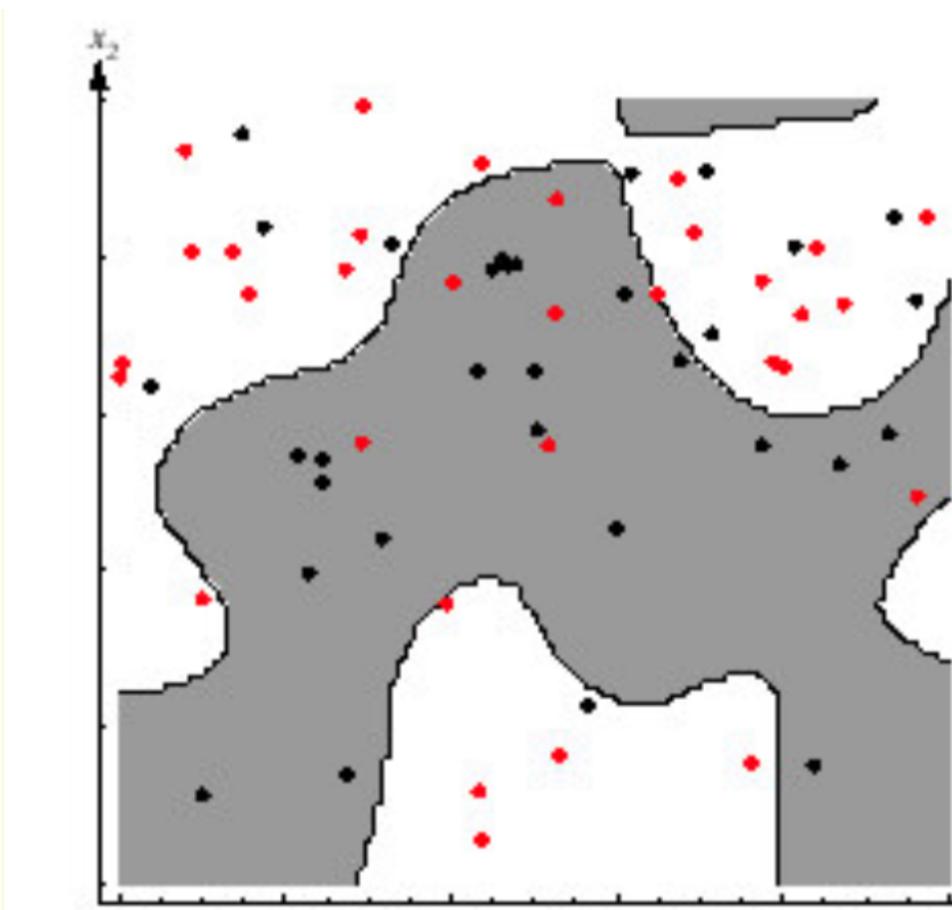
h piccolo

classificazione perfetta ma le superfici di separazione diventano complesse e il classificatore perde in generalità



h grande

classificazione non perfetta ma buona generalizzazione



- molto pesante dal punto di vista computazionale: per classificare un evento bisogna calcolare una funzione che potenzialmente dipende da tutti sample di training
- perde rapidamente potenza quando le feature presentano correlazioni non lineari

NaiveBayes CLASSIFICATION EXAMPLE IN SCIKIT-LEARN

https://www.dropbox.com/s/gbix4v231rfwzhv/NaiveBayes_classification_example.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import make_moons, make_circles, make_classification
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split

# create a toy dataset X data, y target (label)
# redundant features: generated as random linear combinations of the informative features
X, y = make_classification(n_samples=500, n_features=2, n_redundant=0, n_informative=1,
                           random_state=1, n_classes=2, n_clusters_per_class=1, class_sep=1.0)
```

```
print (X[0])
print(y[0])
```

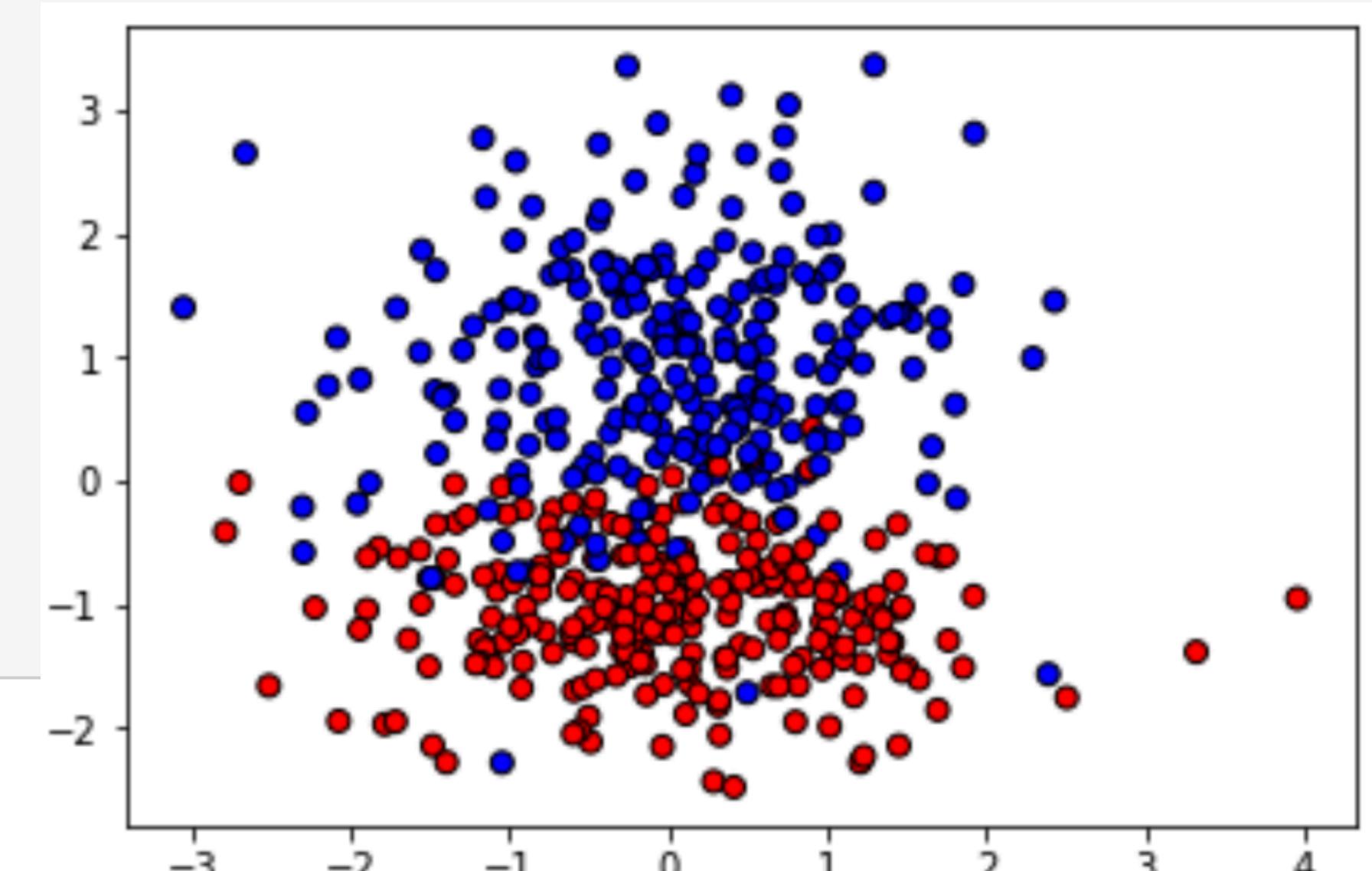


```
[-0.9612636  2.58773317]
1
```

```
print (X.shape)
print (y.shape)
```

```
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cm_bright, edgecolors='k')
plt.show()
```

Generazione di un campione TOY



```
x_train, x_test, y_train, y_test = \
    train_test_split(X, y, test_size=.4, random_state=21, shuffle=True)    split sample in training and test
```

```
print(x_train.shape)      → (300, 2)
print(x_test.shape)       → (200, 2)
```

```
model = GaussianNB()
model.fit(x_train, y_train)
score_train = model.score(x_train, y_train)
score = model.score(x_test, y_test)

print('train accuracy:', score_train)
print('test accuracy:', score)
```

```
train accuracy: 0.9233333333333333
test accuracy: 0.915
```

GaussianNB Model:

the likelihood of the features is assumed to be Gaussian

$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

↓

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

- define the model: **model = GaussianNB()**
- fit the model to data (training): **model.fit(X,y)**
- get the average accuracy: **model.score(X,y)**

equivalent to:

```
accuracy = (np.size(y_test) - np.count_nonzero(predicted-y_test)) / np.size(y_test)
print ('accuracy: ', accuracy)
```

```
accuracy: 0.915
```



- perform classification on a given input: `model.predict(X)`

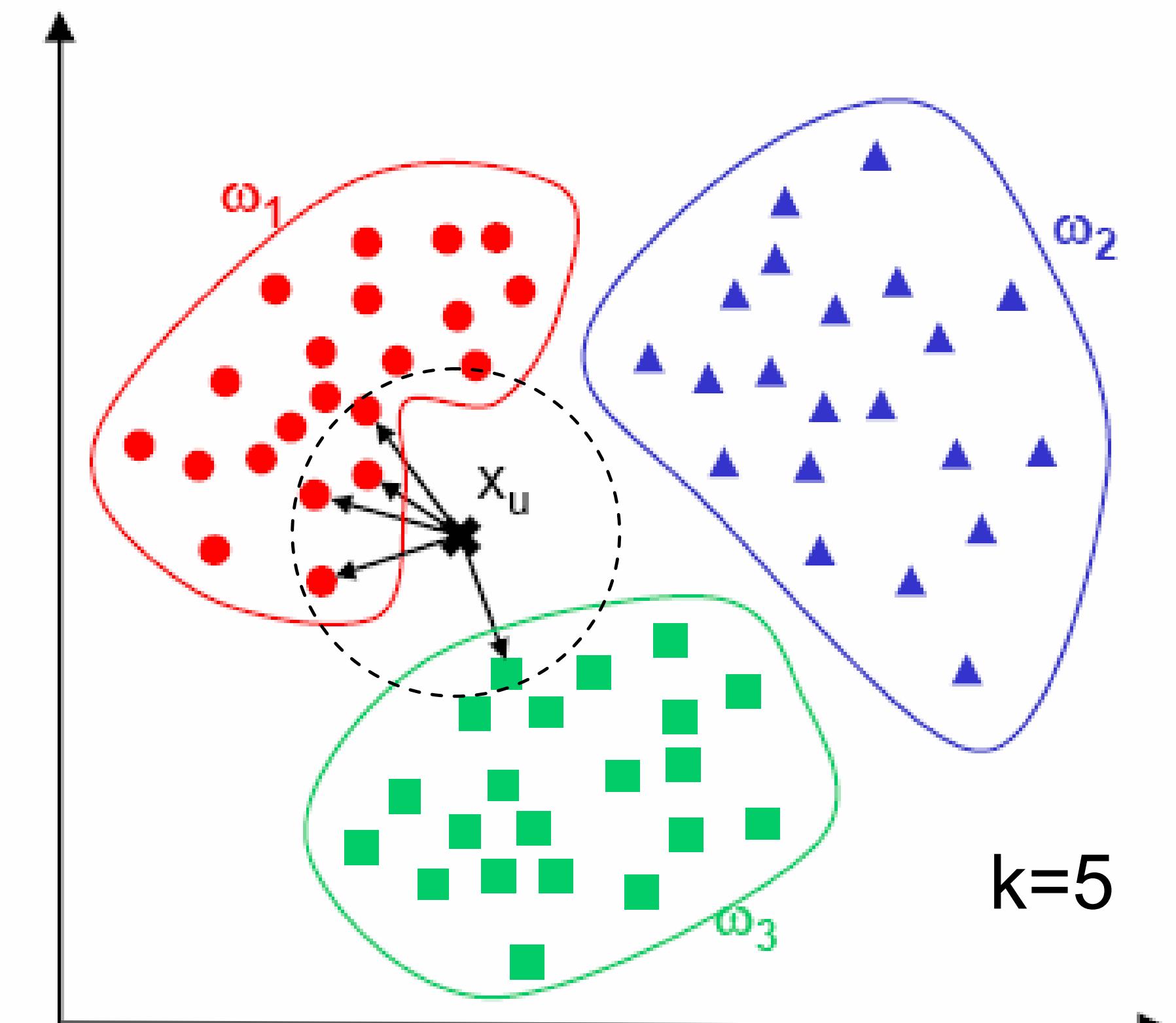
```
# predict output
predicted = model.predict(x_test)

print('predicted: ', predicted)
print('target: ', y_test)
```



CLASSIFICATORE K-NEAREST NEIGHBORS

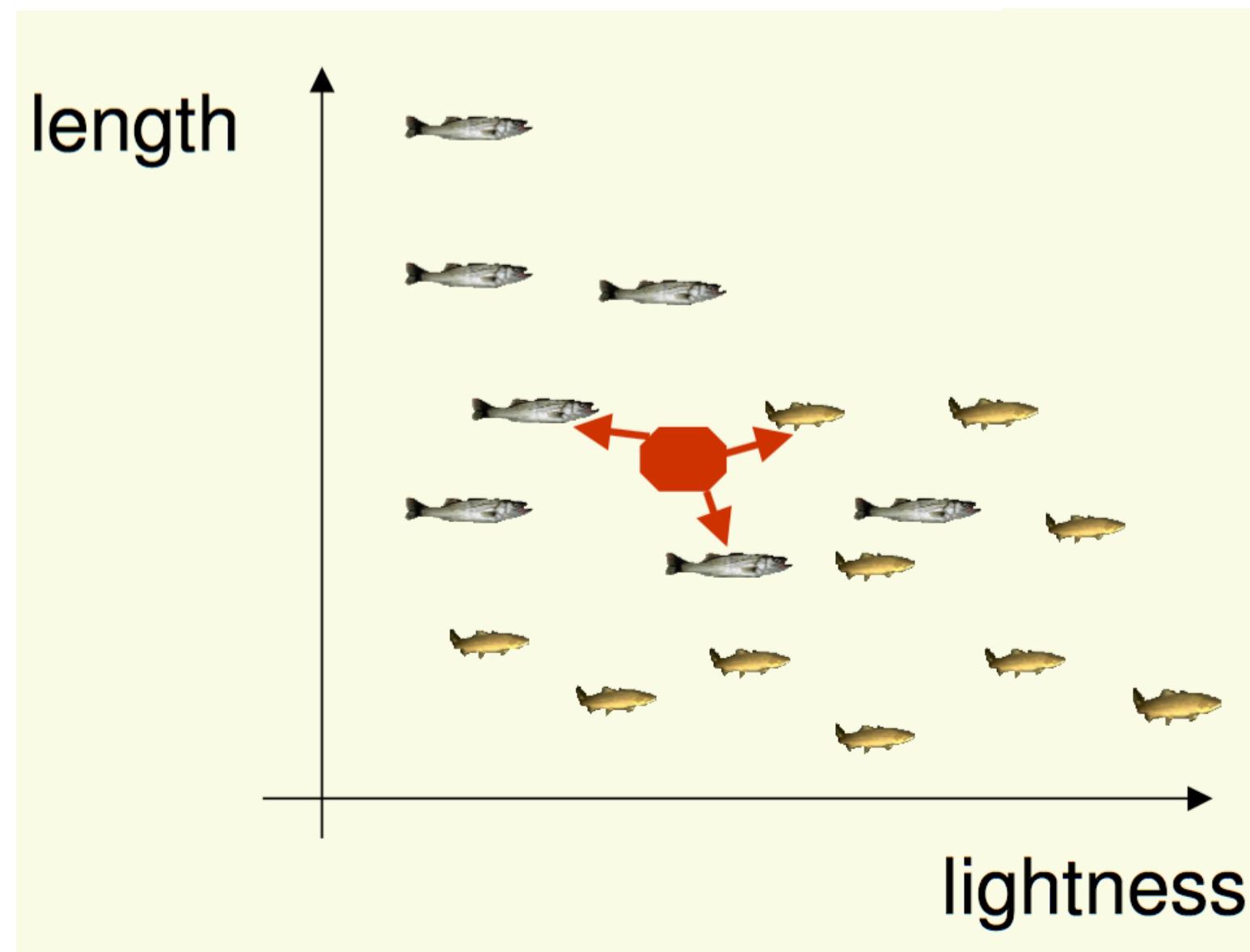
- il classificatore k-NN è un classificatore non parametrico che classifica campioni sulla base della loro somiglianza con gli esemplari del training set T
- è molto semplice: per definire il classificatore è unicamente necessario:
 - scegliere un valore k (il numero di primi vicini)
 - un insieme di campioni di training
 - una metrica per definire la “vicinanza” tra i campioni del training set
- il classificatore è sub-ottimale, nel senso che non garantisce la probabilità di errore minima esibita dal classificatore bayesiano
- è però possibile dimostrare che:
 - anche per $k=1$ per $n \rightarrow \infty$ la rate di errore è sempre < di 2 volte la rate di errore del TLR
 - per $n \rightarrow \infty$, la probabilità di errore P del classificatore k-NN si avvicina alla probabilità di errore del classificatore bayesiano (P^*) se $k \rightarrow \infty$



CLASSIFICATORE K-NN: ESEMPIO TOY

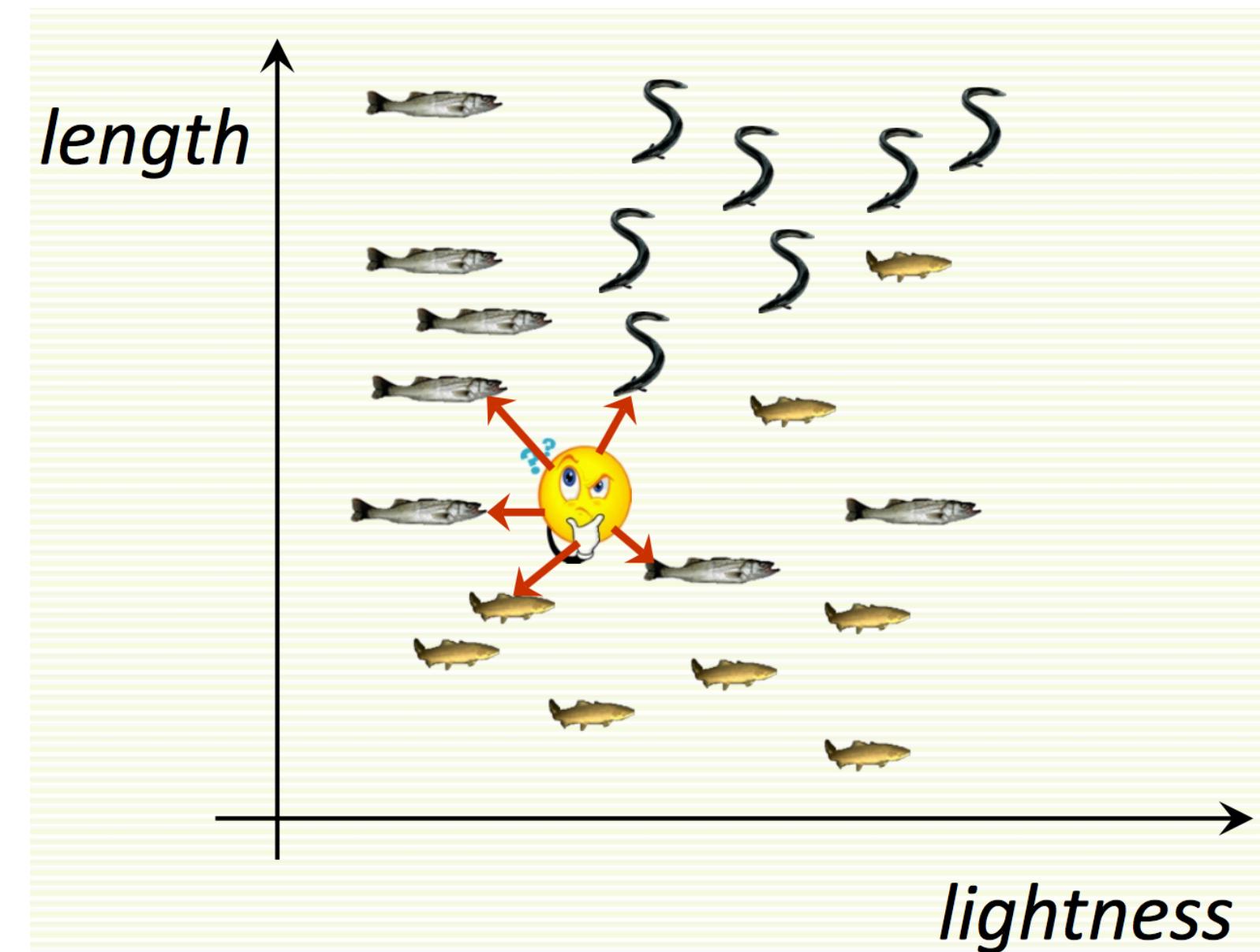
Vogliamo classificare un pesce incognito sulla base di un feature vector (lightness, length) indicato dal punto rosso

supponiamo di scegliere $k=3$



dato il punto rosso, i 3 primi vicini sono:

2 spigole e 1 salmone → il punto è
classificato come spigola



si generalizza facilmente a classi multiple:

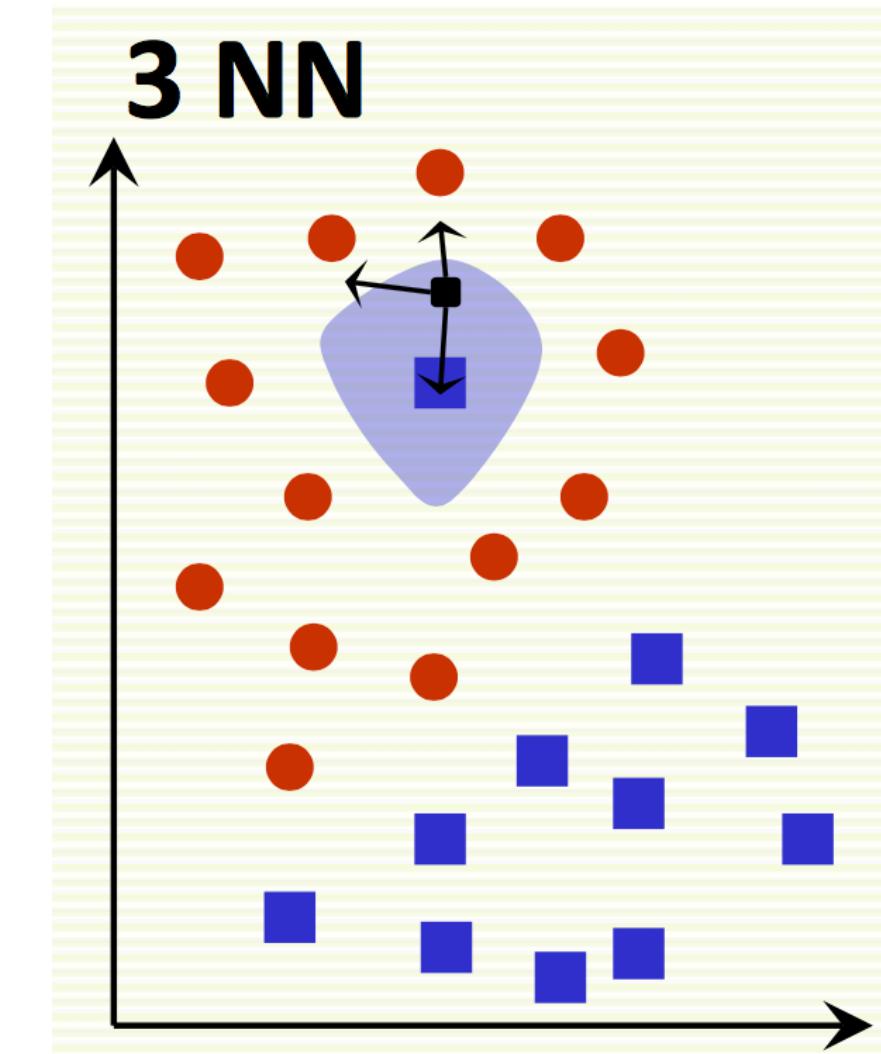
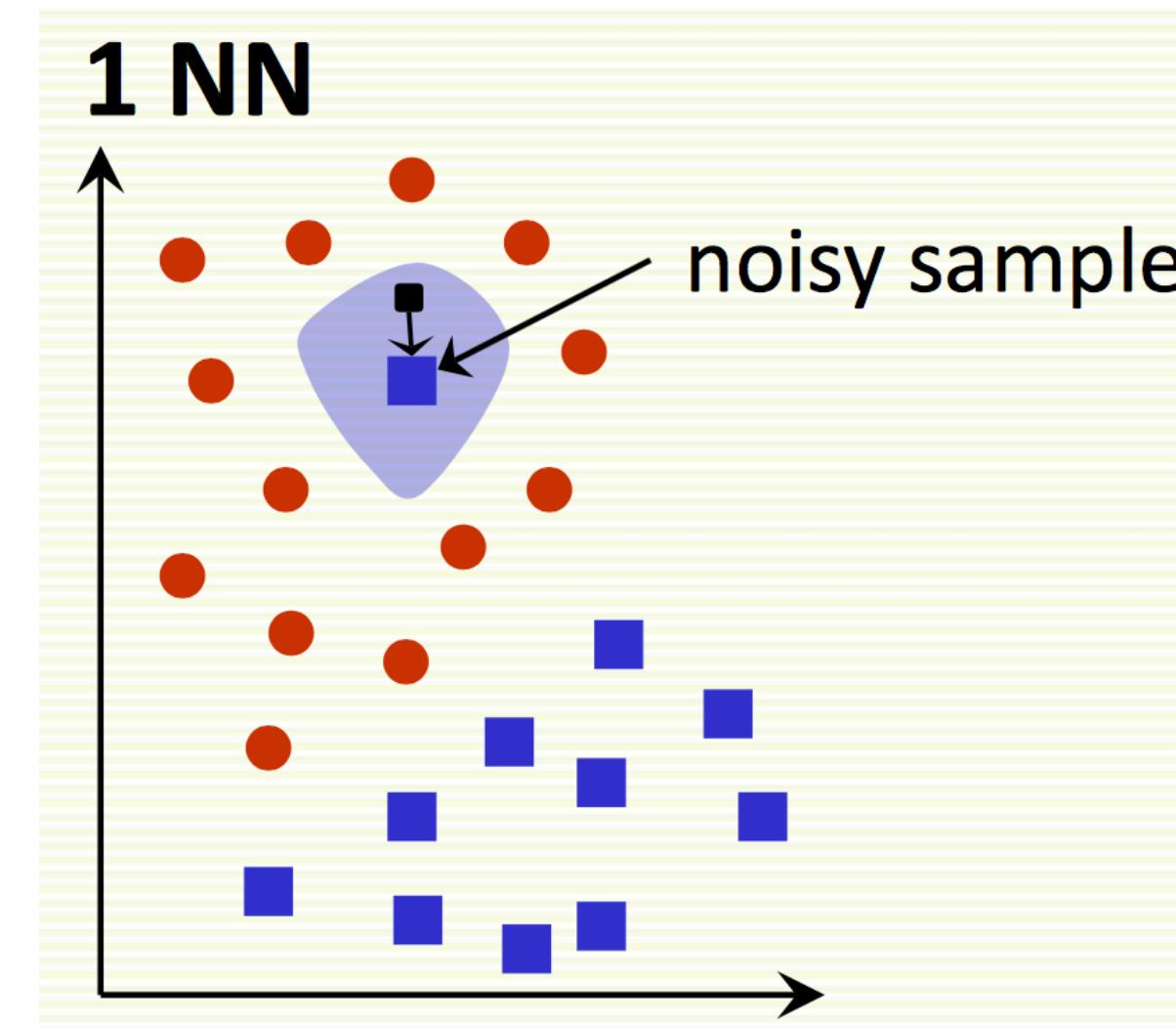
nell'esempio scegliendo $k=5$:

3 spigole, 1 salmone, 1 anguilla → il punto è classificato come
spigola

k-NN fa parte degli algoritmi tipo “lazy learning”, in cui la funzione viene approssimata solo localmente e tutti i calcoli sono rimandati al momento della classificazione

K-NN: SCELTA DEGLI IPERPARAMETRI

- la migliore scelta per k dipende dalla statistica disponibile. Con statistica infinita, più è grande k migliore è la classificazione
- per campioni di dimensione finita: k grande implica bassa efficienza (devono essere disponibili k campioni vicini) e rende i confini tra le classi meno distinti
- k=1 è la scelta più efficiente ma è molto sensibile al rumore



- **regola standard:** $k(n) = \sqrt{n}$ per varie proprietà teoriche (generalizzazione, assenza di bias per $n \rightarrow \infty$)
- spesso però k viene scelta ed ottimizzata empiricamente sulla base del problema specifico (**ottimizzazione degli iperparametri**)

CLASSIFICATORE K-NN: SCELTA DELLA METRICA

- scelta più usata: distanza euclidea
 - tratta tutte le feature democraticamente
 - questo può creare problemi in caso di:
- feature con diversi intervalli dinamici → normalizzazione delle feature in input

$$f_{new} = \frac{f_{old} - f_{old}^{\min}}{f_{old}^{\max} - f_{old}^{\min}} \rightarrow [0,1]$$

$$f_{new} = \frac{f_{old} - \mu}{\sigma} \rightarrow N(0,1)$$

- feature irrilevanti → distanza pesata

$$D(a, b) = \sqrt{\sum_k (a_k - b_k)^2} = \sqrt{\sum_i (a_i - b_i)^2 + \sum_j (a_j - b_j)^2}$$

feature con potere
di separazione

feature irrilevanti
(noise)



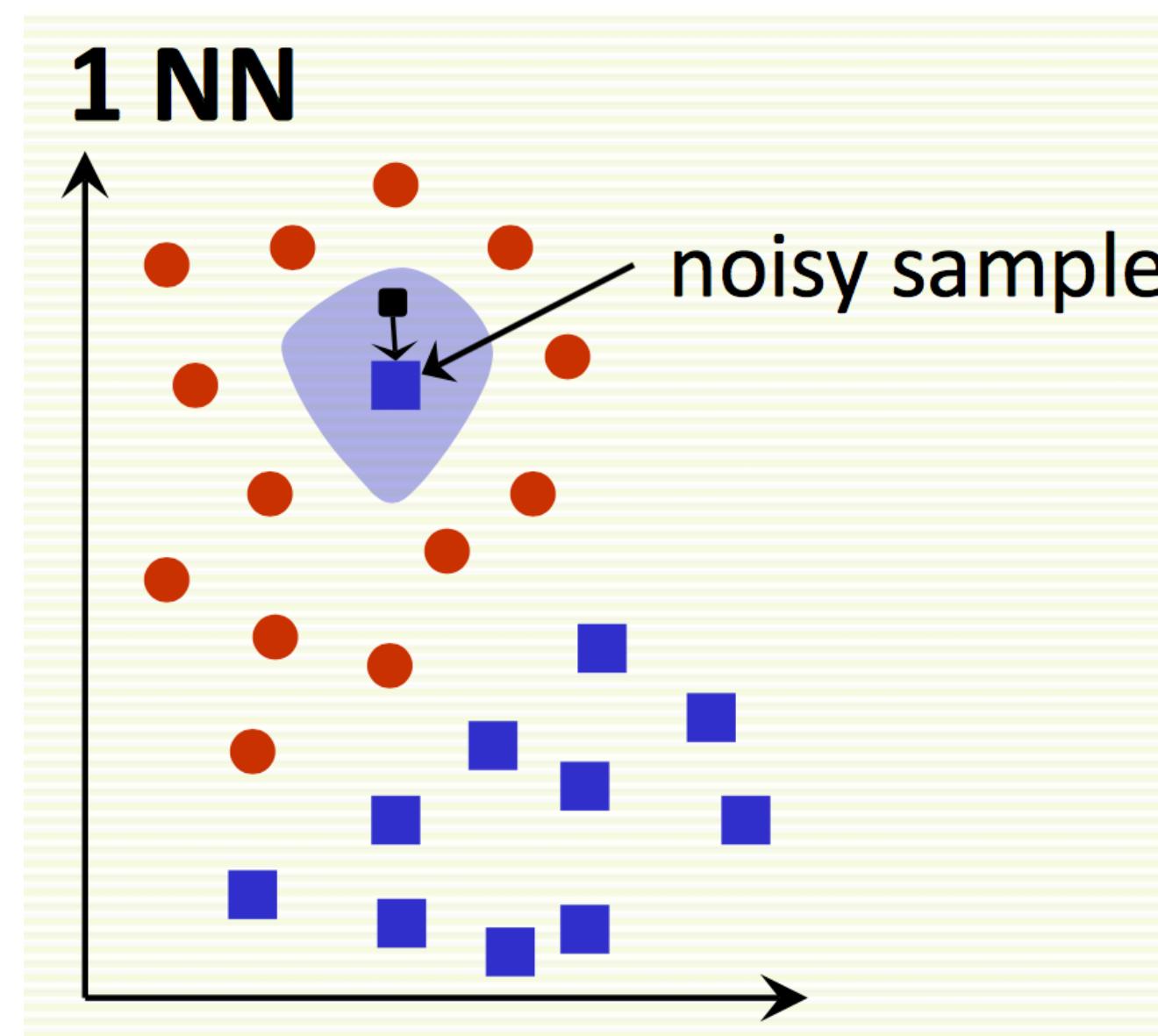
$$D(a, b) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$

w_i scelti o su informazioni a priori
o utilizzando indicatori del potere
di separazione di ciascuna feature

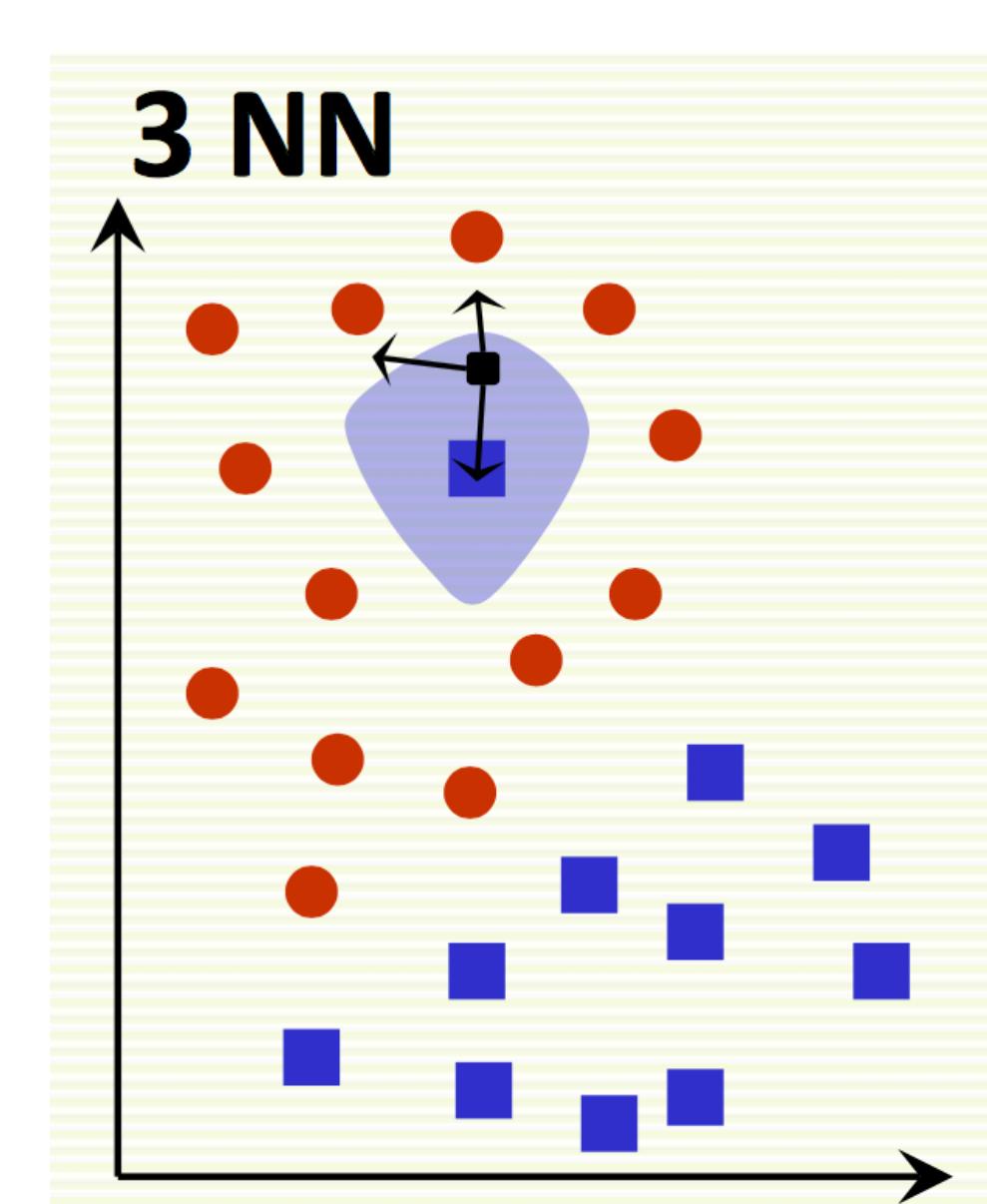


CLASSIFICATORE K-NN: SCELTA DI K

- in teoria: con statistica infinita, più è grande k migliore è la classificazione
- per campioni di dimensione finita: k grande implica bassa efficienza (devono essere disponibili k campioni vicini)
- k=1 è la scelta più efficiente ma è molto sensibile al rumore



ogni evento nella
regione magenta
verrebbe mis-
classificato come
appartenente alla
classe blu



- **regola standard:** $k(n) = \sqrt{n}$ per varie proprietà teoriche (generalizzazione)
- spesso però k viene scelta ed ottimizzata empiricamente sulla base del problema specifico (**ottimizzazione degli iperparametri**)

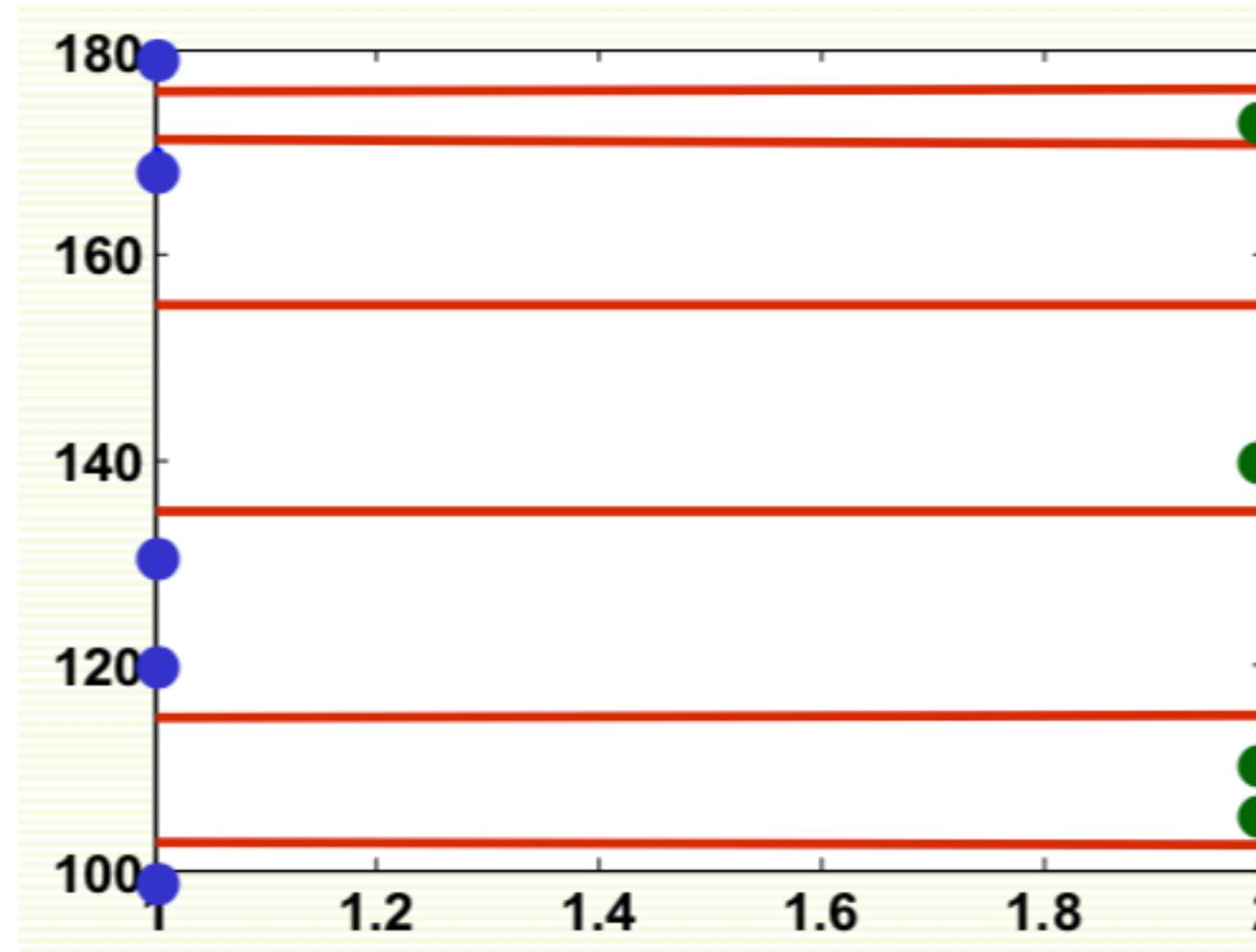
ESEMPIO

- Esempio limite: problema a 2 classi descritto da 2 feature
 - x_1 : ha potere discriminante e assume i valori discreti 1 o 2
 - x_2 : nessun potere discriminante: numero uniforme tra 100 e 200
- se andiamo a vedere le regioni di decisione chiaramente risultano inadeguate, a causa della scala di x_2 che domina il calcolo della distanza

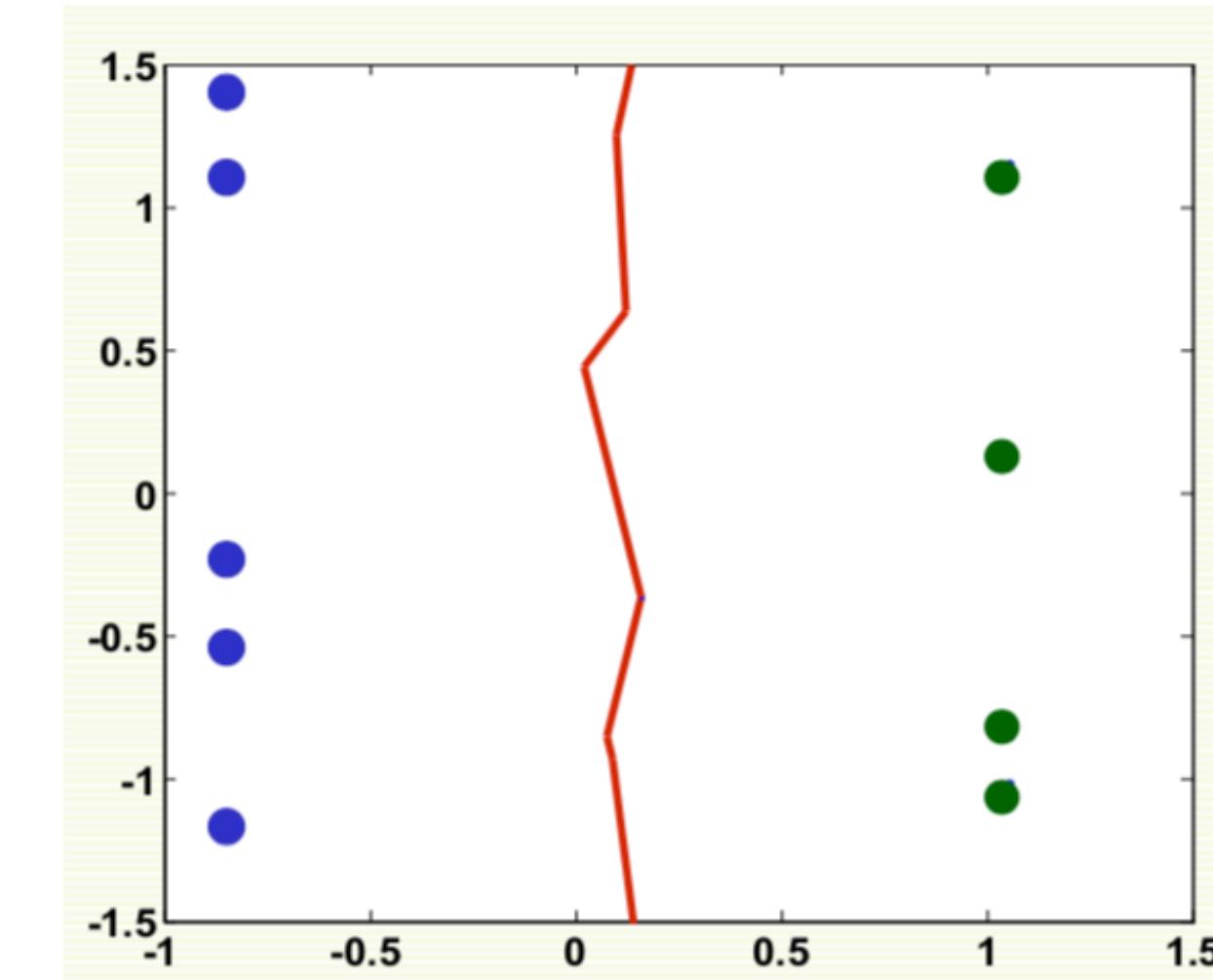
dataset: [1 150]
[2 110]
classify [1 100]

$$D\left(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 1 \\ 150 \end{bmatrix}\right) = \sqrt{(1-1)^2 + (100-150)^2} = 50$$
$$D\left(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 2 \\ 110 \end{bmatrix}\right) = \sqrt{(1-2)^2 + (100-110)^2} = 10.5$$

sbagliato!

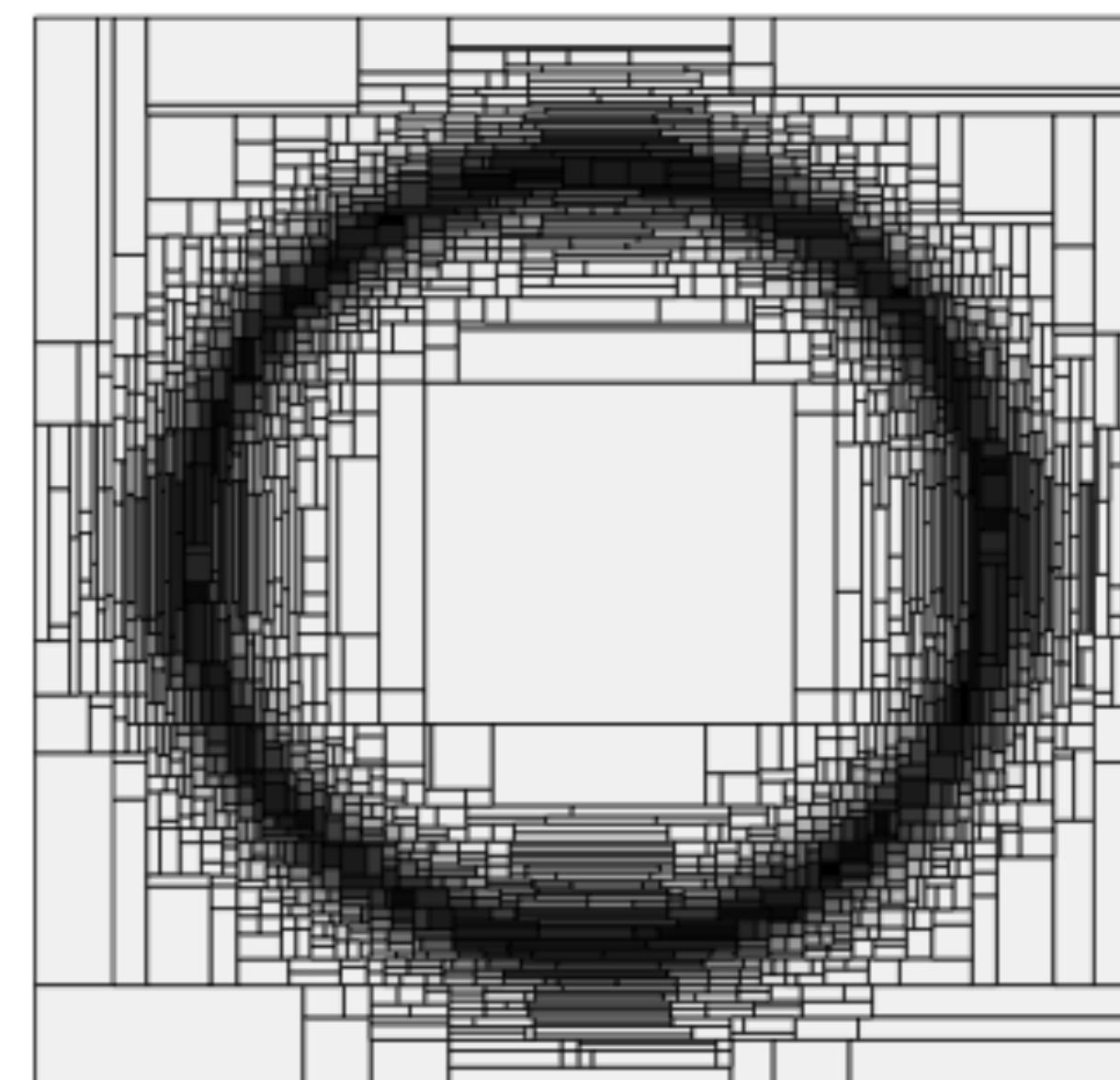
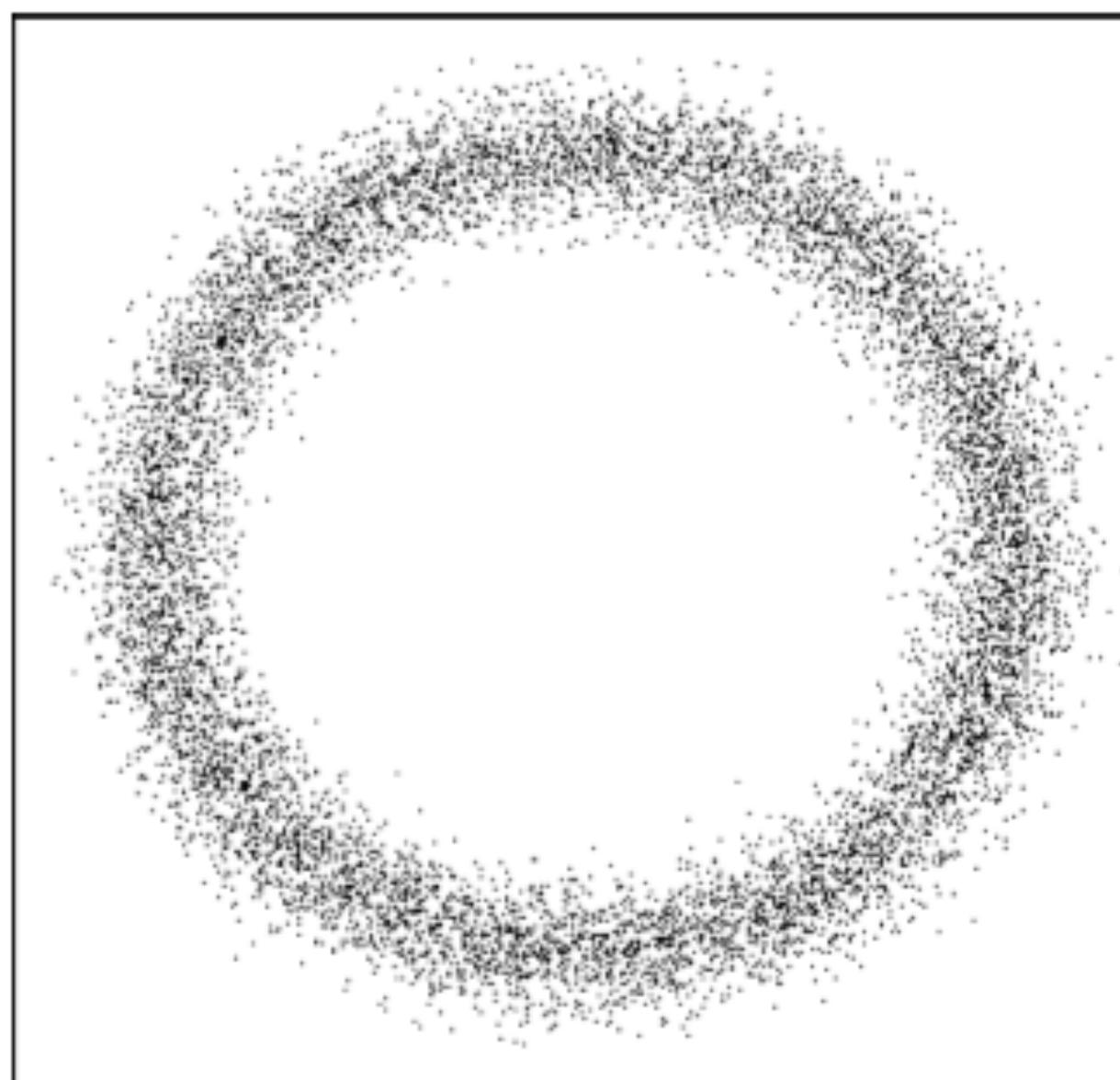


normalizzazione
delle feature

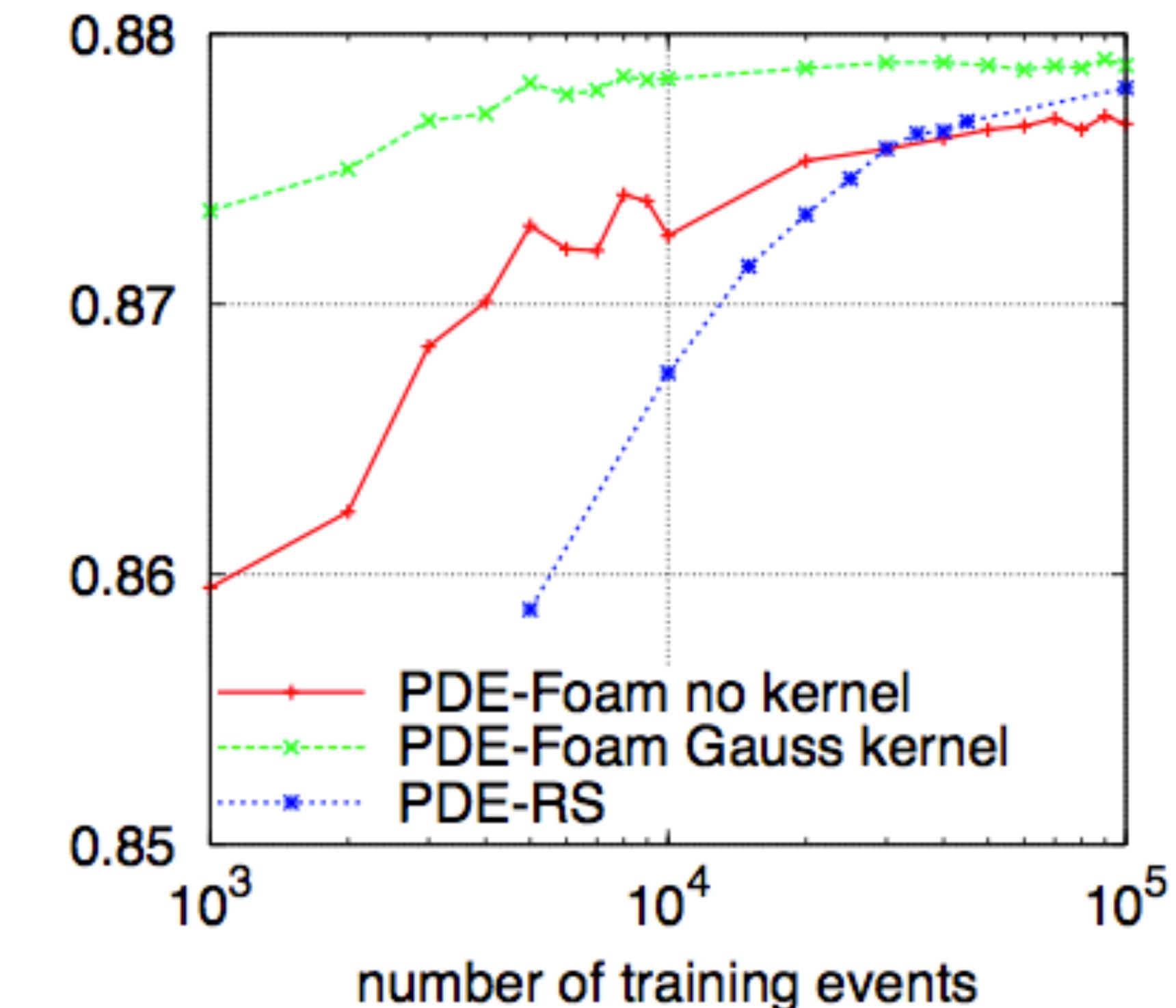


PDE-FOAM / HYPERBALL CLASSIFIERS

- evoluzione dei classificatori basati sulla stima della probabilità che preserva il vantaggio di k-NN di considerare le correlazioni tra osservabili senza richiedere enormi campioni di training per evitare le fluttuazioni statiche associate
- usa algoritmi adattivi che modificano dinamicamente la dimensione/forma dei volumi con i quali si campiona l'iperspazio delle features, in modo tale da minimizzare la varianza della stima della pdf



AUC: ROC Area
ROC area



K-NN CLASSIFICATION EXAMPLE IN SCIKIT-LEARN

https://www.dropbox.com/s/2n3ge1ykjdj1g31s/kNN_classification_example.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

IRIS dataset available in scikit-learn

```
# import example data: iris dataset
# consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal
# length, stored in a 150x4 numpy.ndarray
# rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
iris = datasets.load_iris()
```

```
# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
```

```
x = iris.data[:, :2]
```

```
y = iris.target
```

```
print (x[0])
```

```
print(y[0])
```

```
print (x.size)
```

[5.1 3.5]

0

300



```

n_neighbors = 15
h = .02 # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    # weight = uniform --> All points in each neighborhood are weighted equally
    # weight = distance --> weight points by the inverse of their distance (closer neighbors
    # of a query point will have a greater influence)
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

```



```

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

#np.arange: Return evenly spaced values within a given interval
# np.meshgrid: Return coordinate matrices from coordinate vector
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# np.c_ : translates slice objects to concatenation along the second axis
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

print(Z.shape)

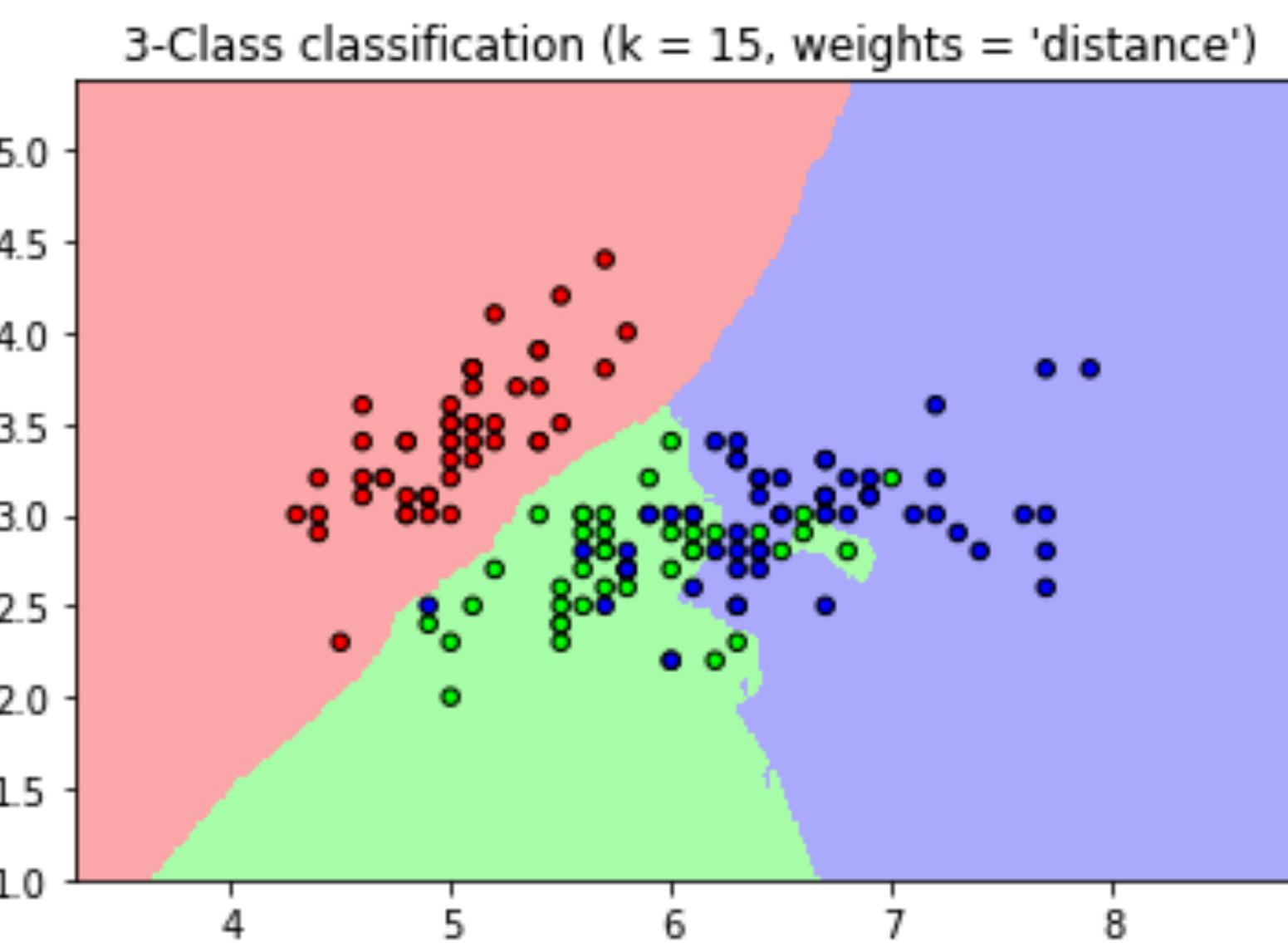
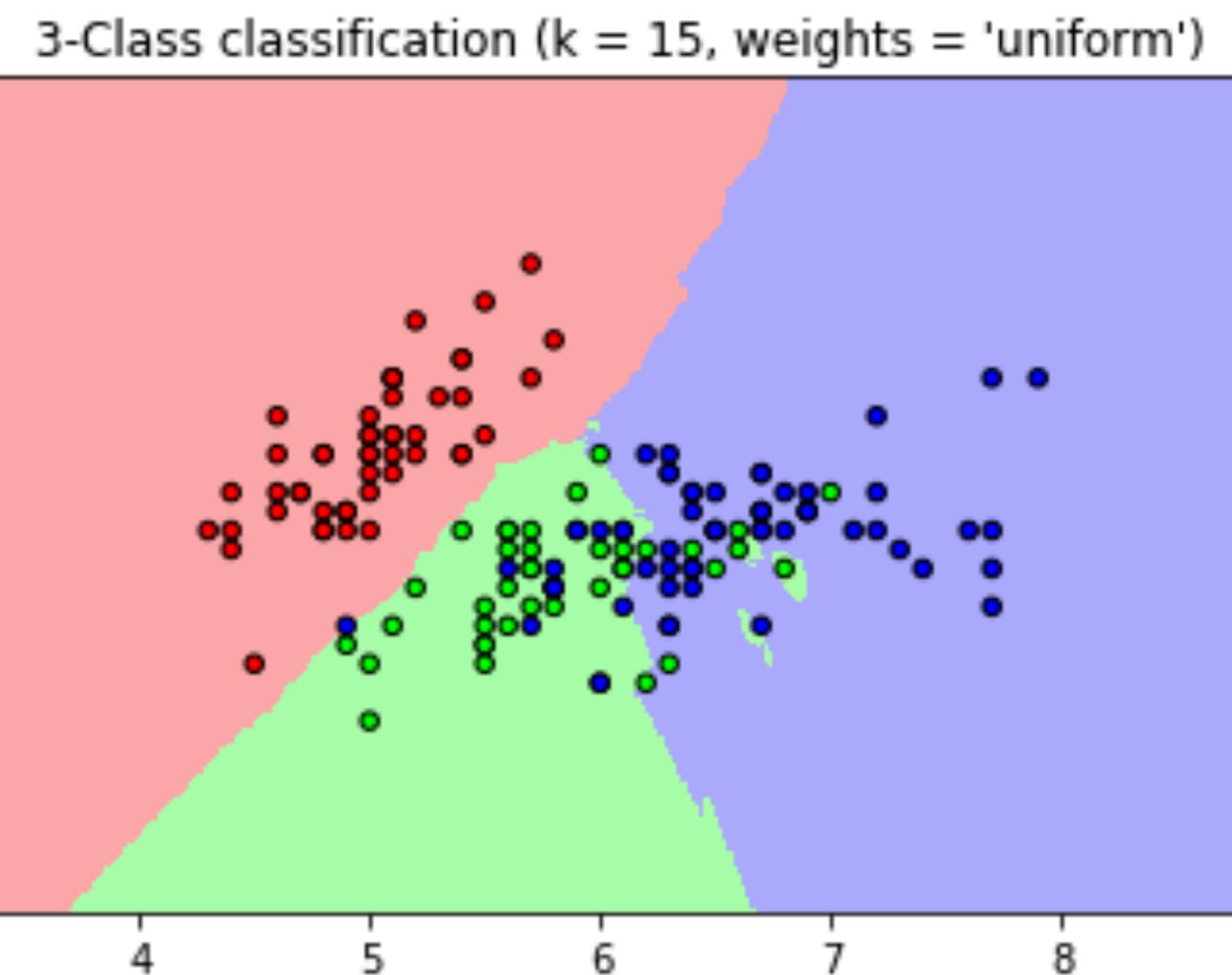
# Put the result into a color plot
Z = Z.reshape(xx.shape)
print(Z.shape)

plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
            edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = %i, weights = '%s')"
          % (n_neighbors, weights))

plt.show()

```



```
# predict output
n_neighbors = 2
clf = neighbors.KNeighborsClassifier(n_neighbors, weights='uniform')
clf.fit(X, y)

predicted = clf.predict(X)

print('predicted: ', predicted, ' vs target:', y)
```

```
score = clf.score(X, y)
print('accuracy: ', score)
```

accuracy: 0.8733333333333333

