

# Machine Learning: Basi e Sue Applicazioni

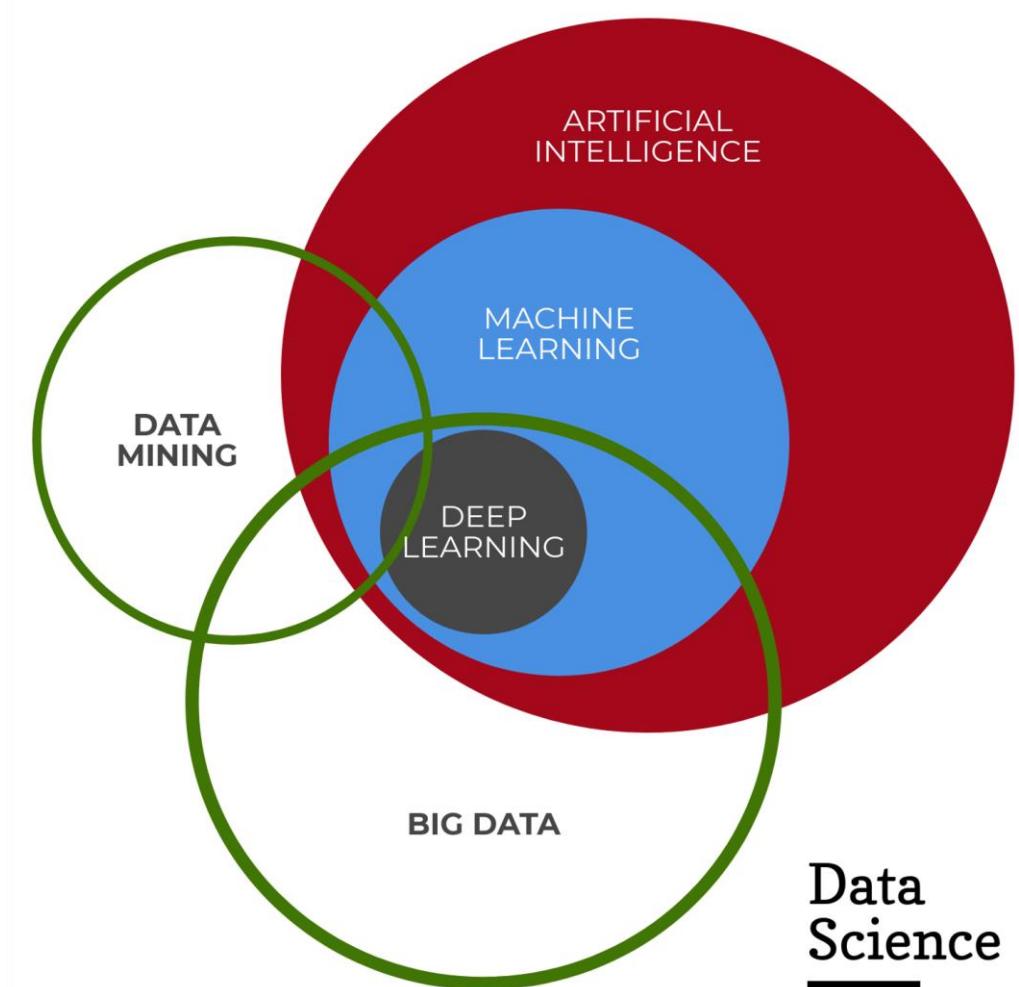
Christian Salvatore  
Scuola Universitaria Superiore IUSS Pavia

# Machine learning

Machine learning is the subfield  
of **computer science** that gives  
**"computers the ability to learn  
without being explicitly programmed"**

---

**Arthur Samuel**, 1959



## How is ML different from classical statistics?

*It is focused on  
**classification** rather than  
inference*

---

*Distribution-free approach*

---

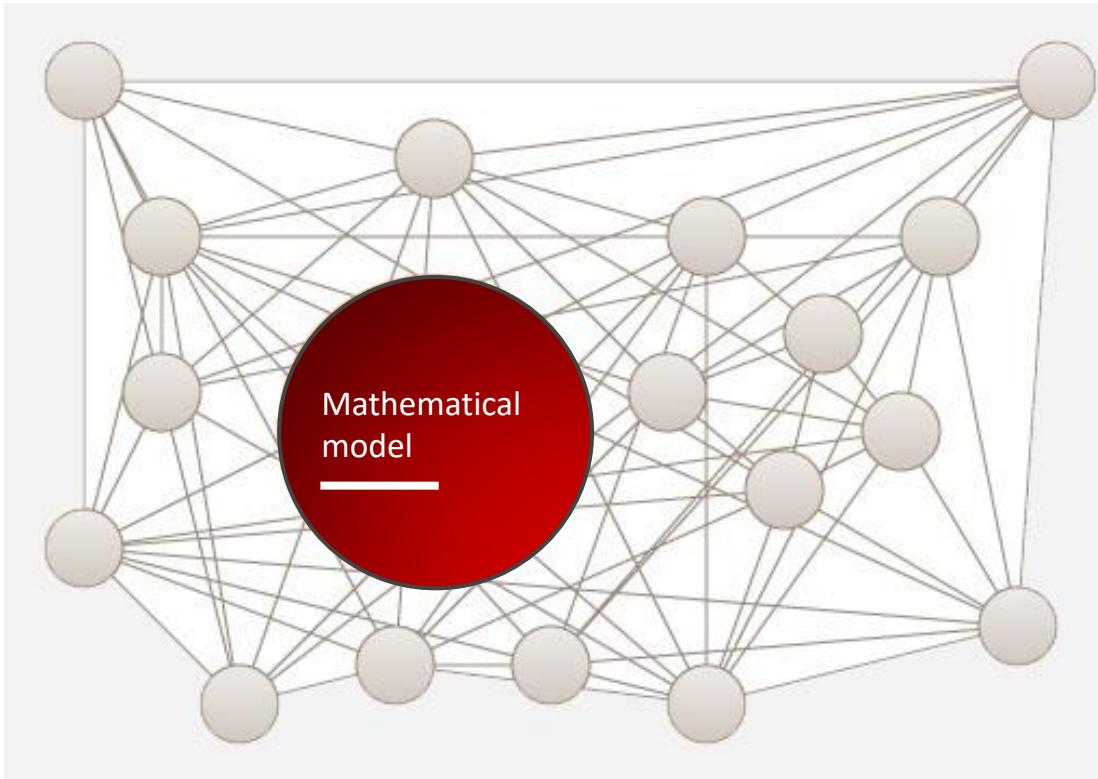
*High-dimensional  
problems*

---

*Algorithmic and  
computational aspects play a  
central role*

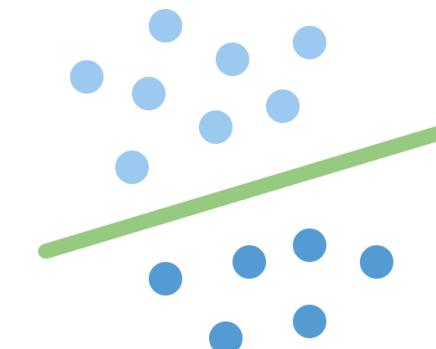
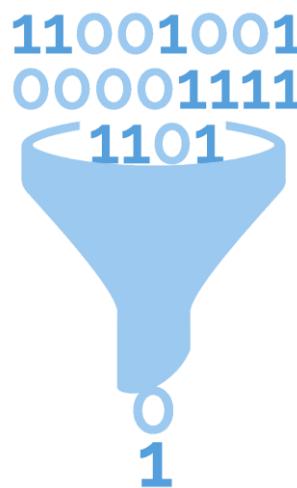
---

# Machine learning



Designing mathematical models able to

# Machine learning



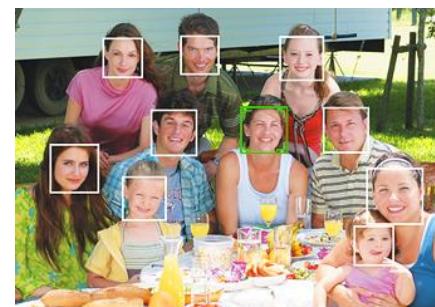
1. identify hidden patterns in data
2. handle and summarize a great quantity of data into a model
3. use that model to perform automatic classification

# Machine learning

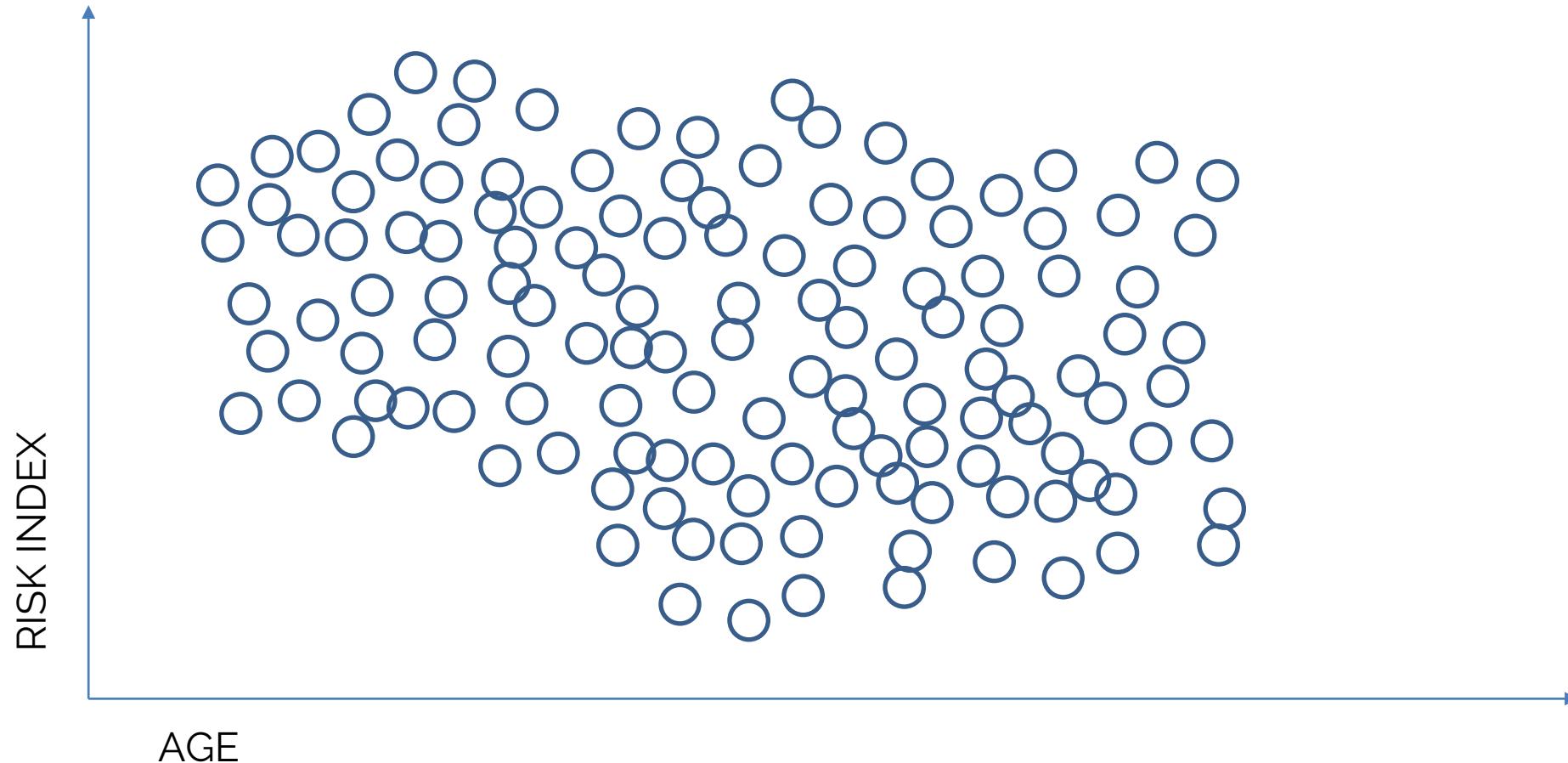
Training data → Target variable

From the trend registered during the last years      PREDICT      Future market performance

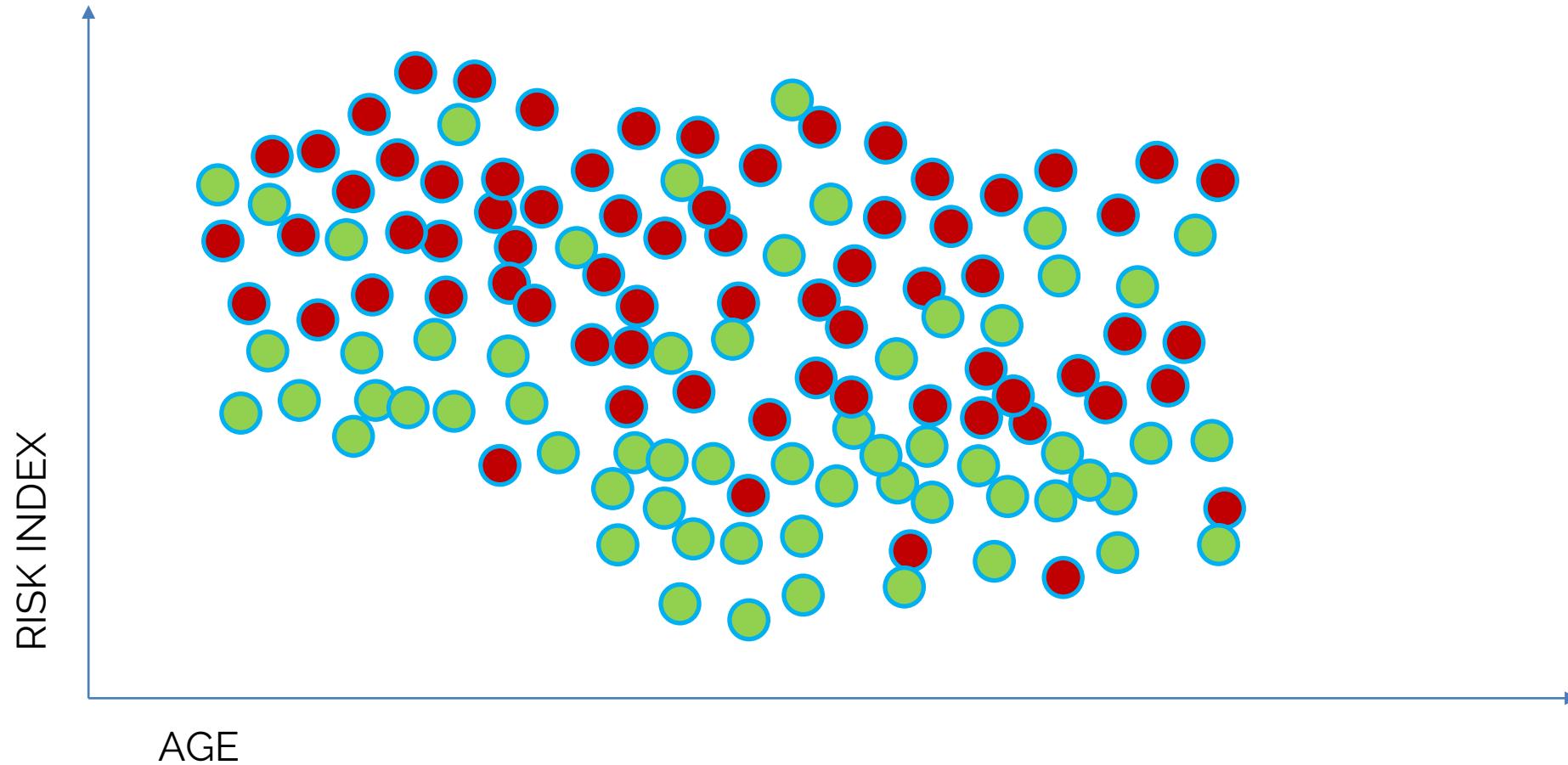
From a set of images      DETECT      Faces (in new images)



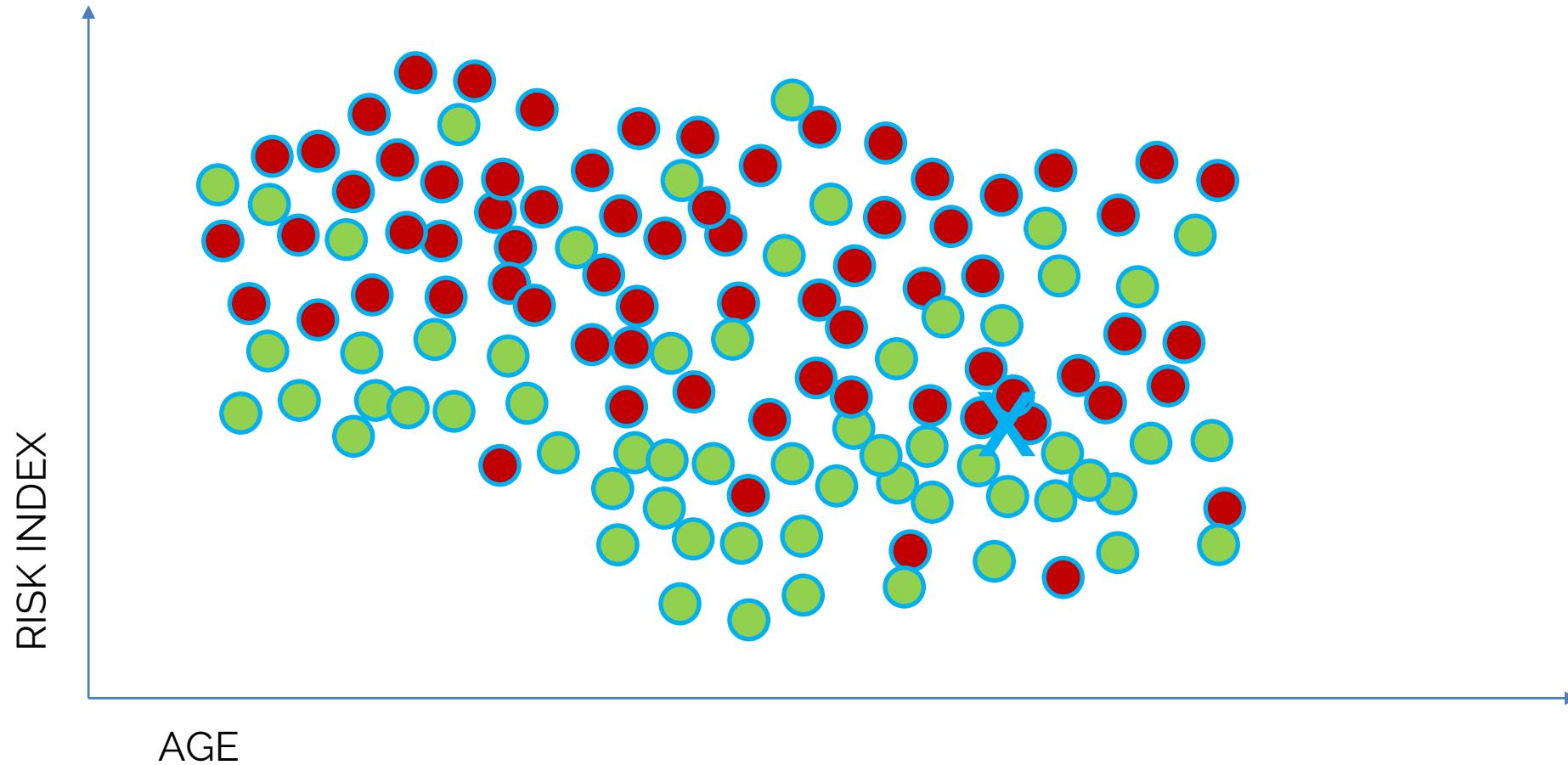
# Classification



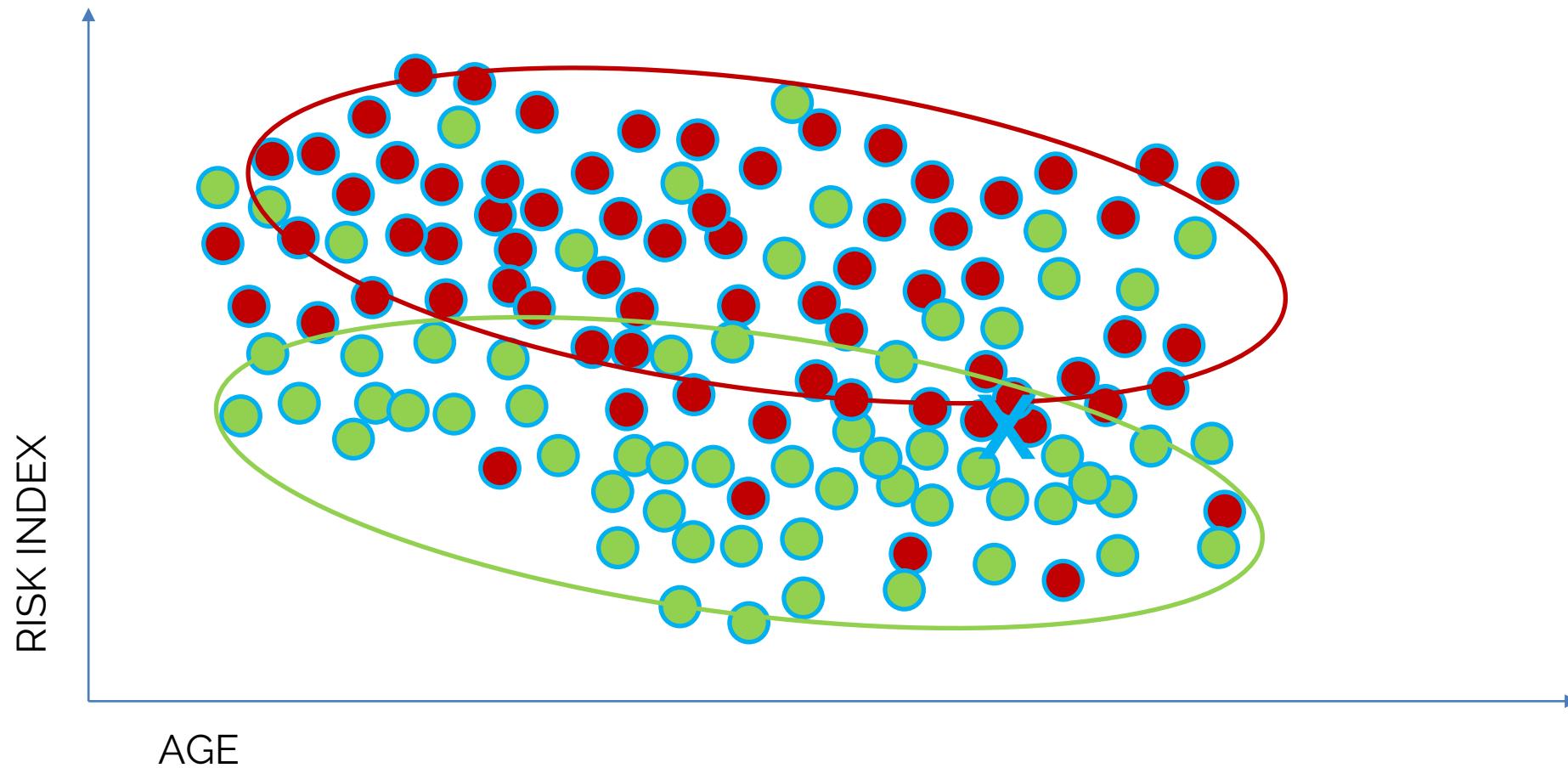
# Classification



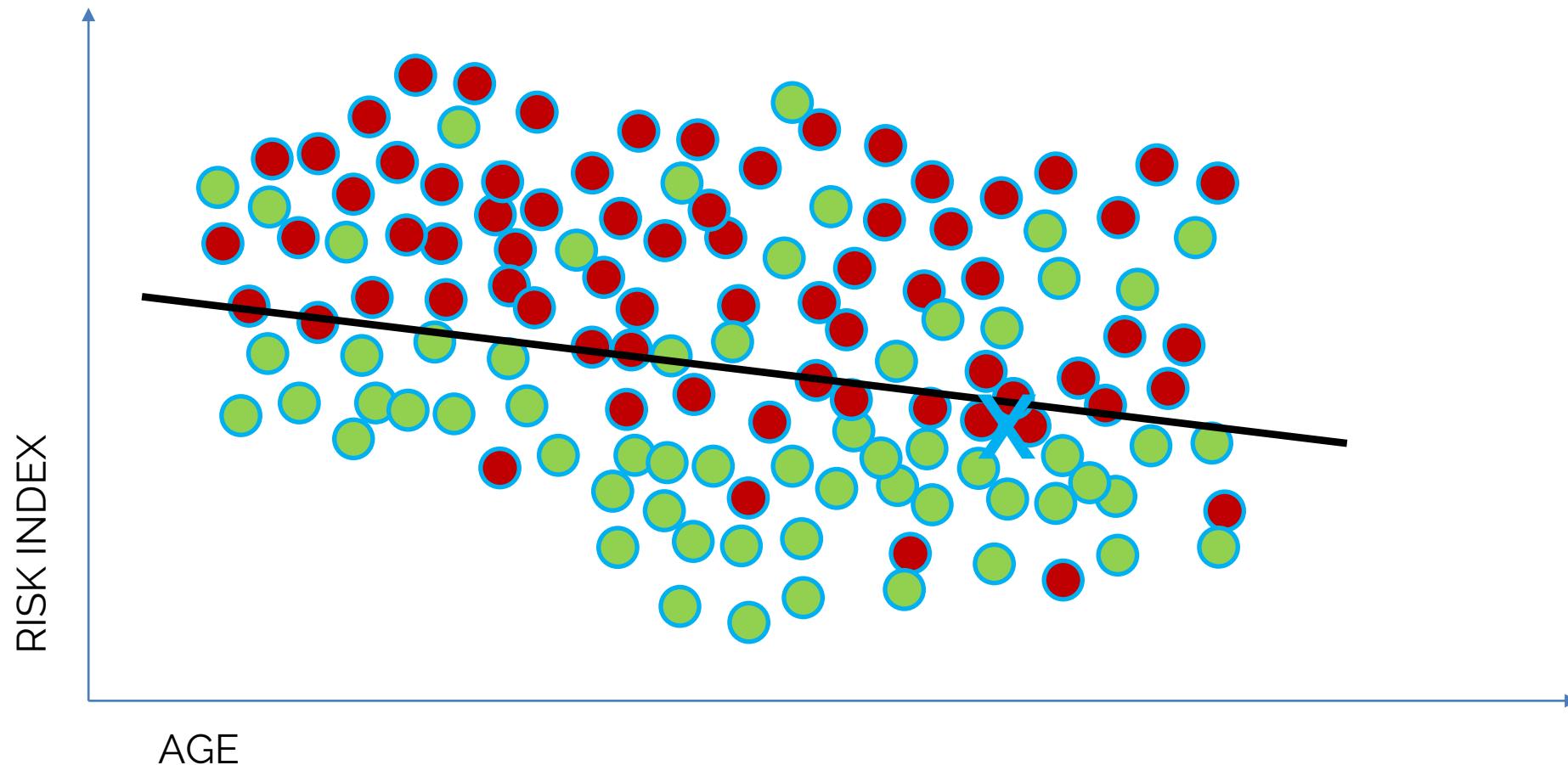
# Classification



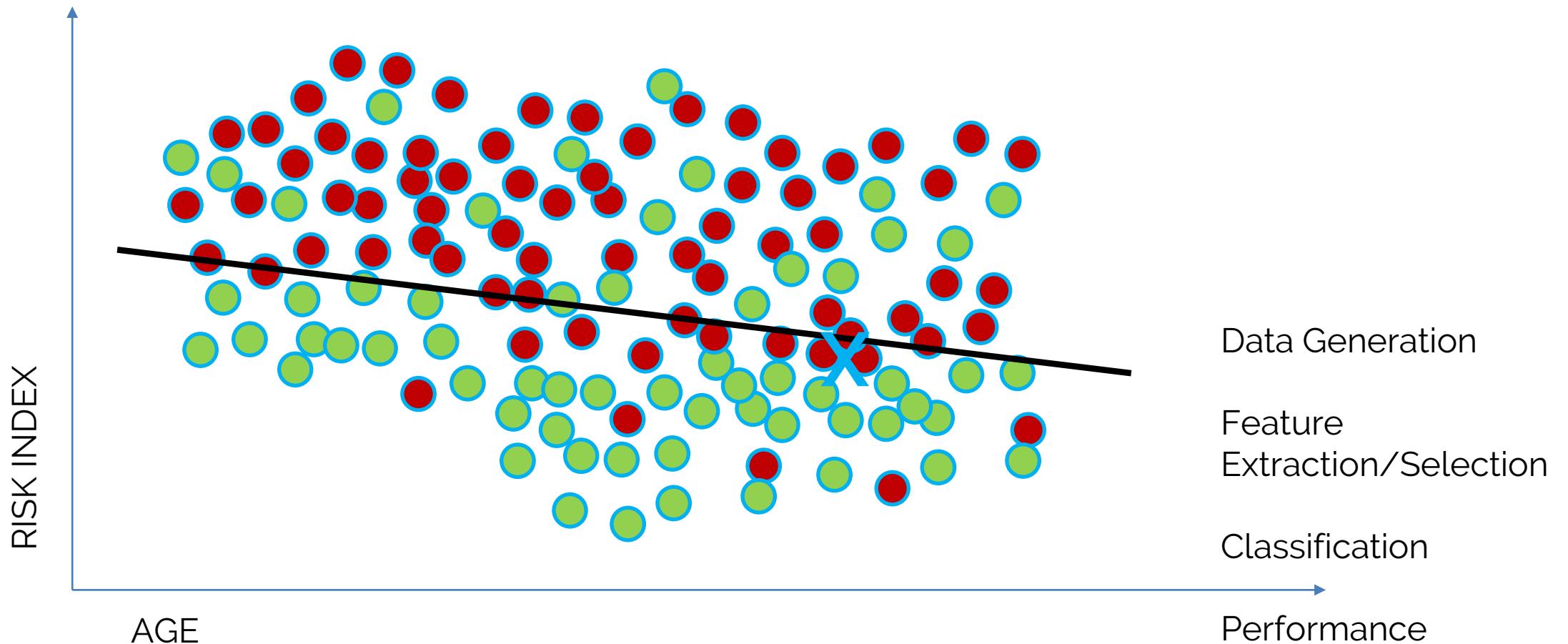
# Classification



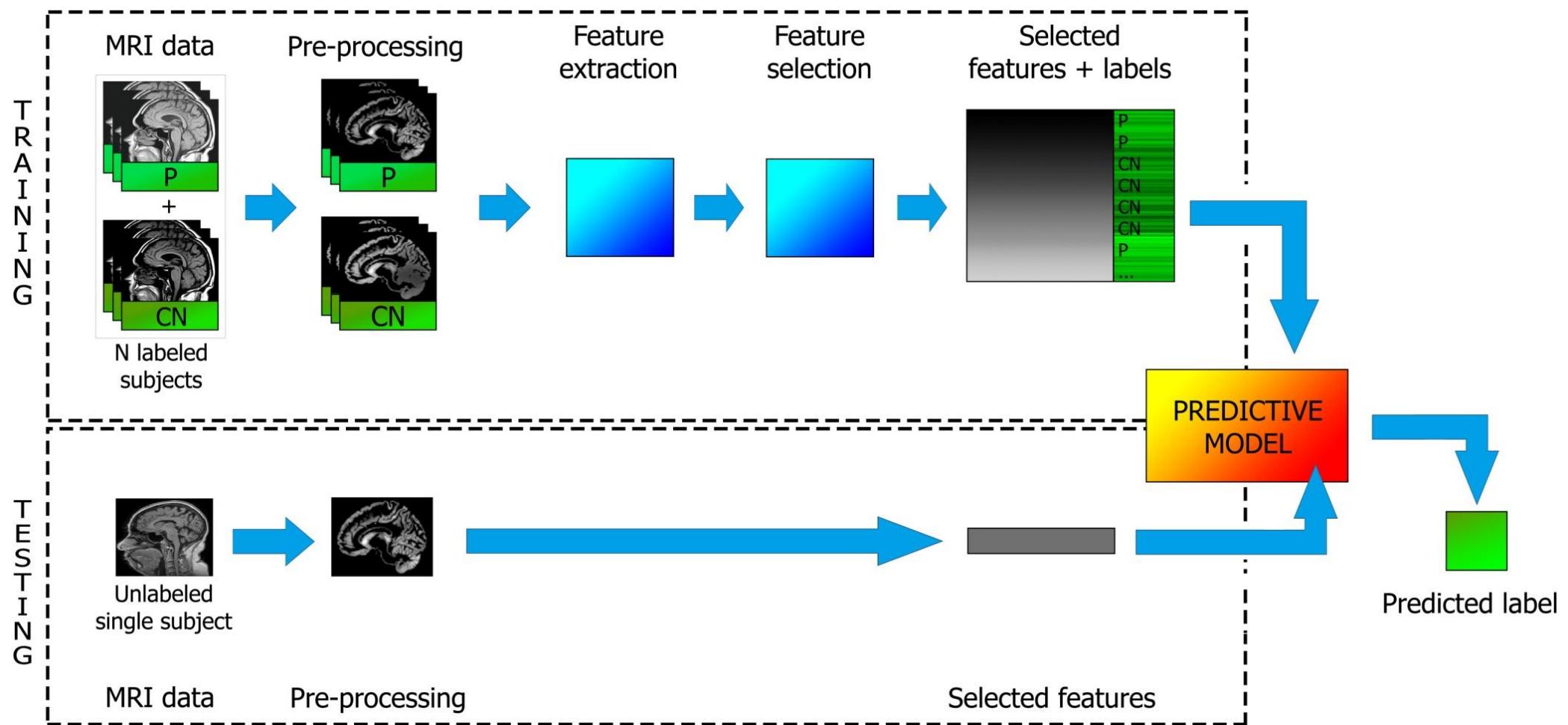
# Classification



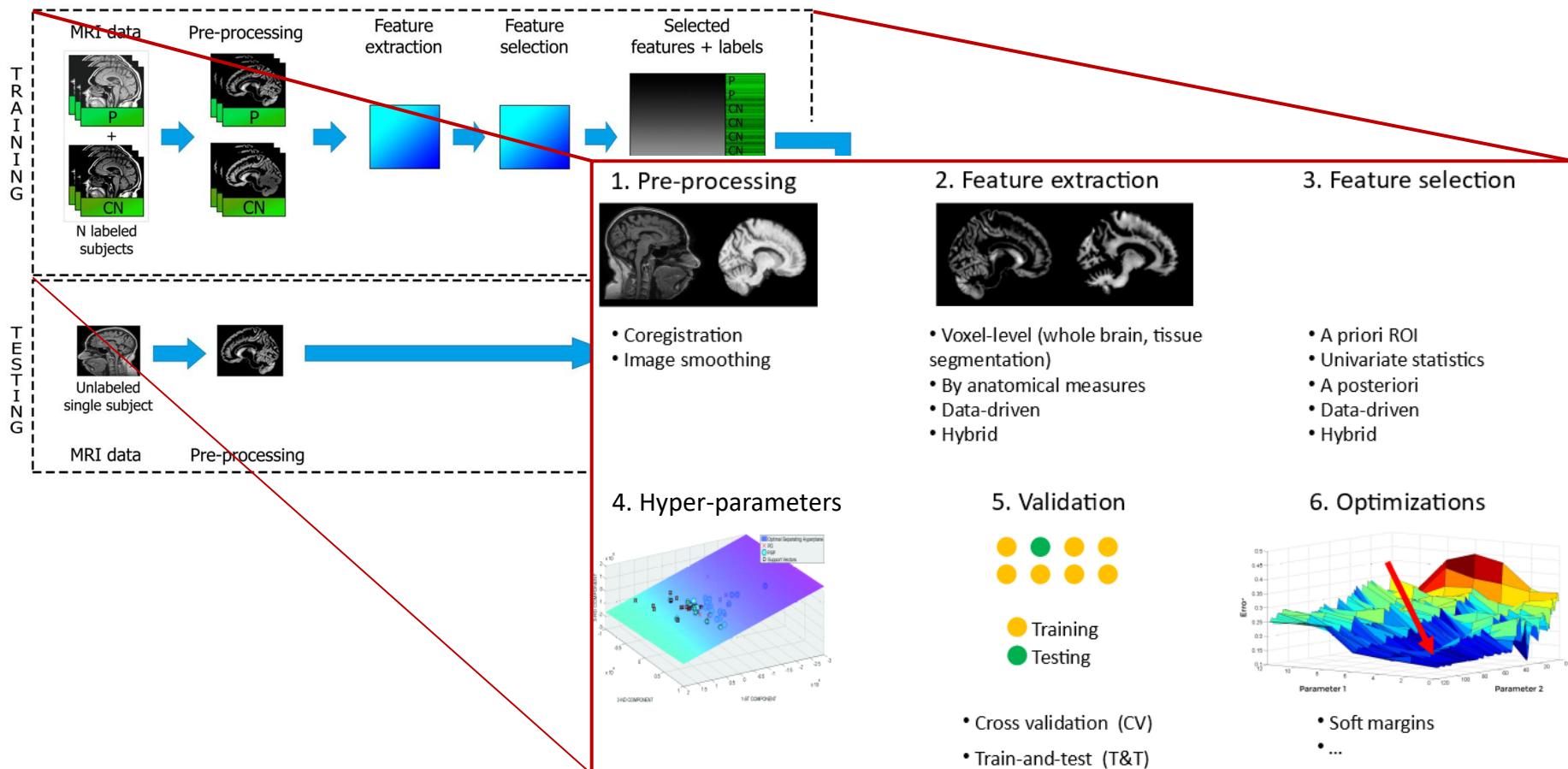
# Classification



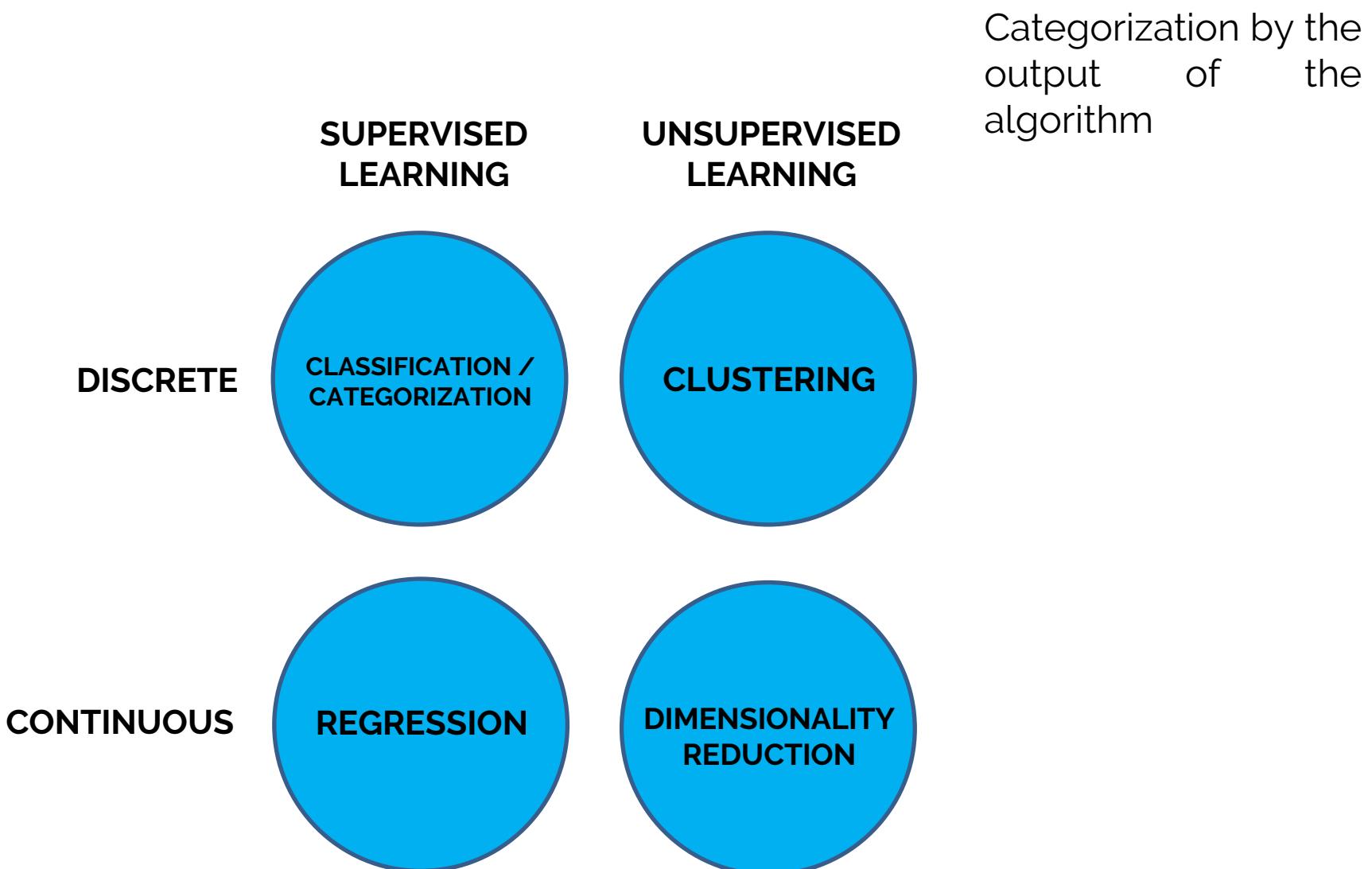
# Machine Learning Applied to Medical Data



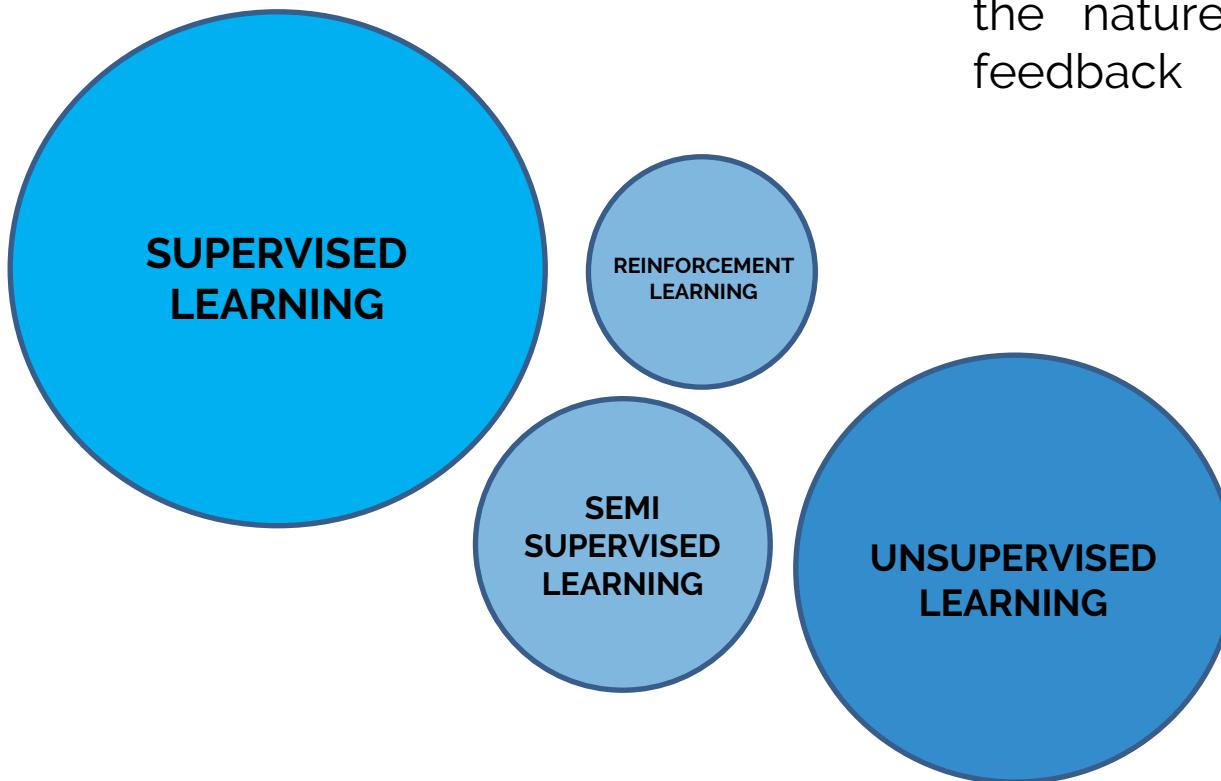
# Machine Learning Applied to Medical Data



# Machine learning



# Machine learning



Categorization by  
the nature of the  
feedback

# FEATURE EXTRACTION

# Feature Extraction | Obiettivo

Eseguire un mapping dallo spazio iniziale a uno spazio di dimensione inferiore  $R^k$ , con  $k < d$

Operare in spazi a dimensionalità inferiore

- rende più semplice addestrare algoritmi di machine learning (richiede meno dati per l'addestramento)
- scartando dati ridondanti (informazioni correlate) e rumorosi si migliorano anche le prestazioni e le si rendono più robuste

Obiettivo è scartare le informazioni non rilevanti o meno rilevanti per il problema di interesse

Ridurre la dimensionalità non significa ridurre alcune dimensioni e salvarne altre, ma combinare le dimensioni in modo opportuno.

# Feature Extraction | Main Techniques

Analisi delle Componenti Principali - Principal Component Analysis (PCA)

- trasformazione non-supervisionata che esegue un mapping lineare delle dimensioni con l'obiettivo di preservare al massimo l'informazione dei pattern (nota anche come Karhunen-Loeve -KL- transform)

(FEATURE SELECTION)

Analisi delle Discriminanti Lineari, Linear Discriminant Analysis (LDA)

- trasformazione supervisionata di mapping lineare

# Feature Extraction | Principal Components Analysis

Dato un training set  $\mathbf{x}_i \in \mathbb{R}^d$ ,  $i = 1 \dots n$ , siano

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1 \dots n} \mathbf{x}_i \quad \text{il vettore medio} \in \mathbb{R}^d$$

$$\Sigma = \frac{1}{n-1} \sum_{i=1 \dots n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^t \quad \text{la matrice di covarianza} \in \mathbb{R}^{d \times d}$$

allora per un dato  $k$  ( $k < d$ ,  $k < n$ ,  $k > 0$ ), lo spazio  $k$  dimensionale  $(Sx, \Phi_k)$

è univocamente definito dal vettore medio e dalla matrice di proiezione  $\Phi_k \in \mathbb{R}^{d \times k}$

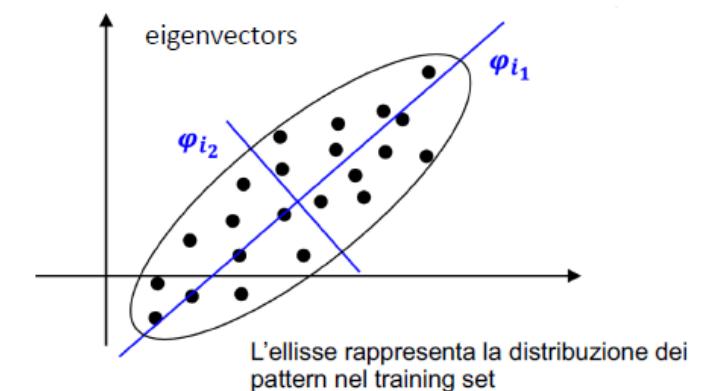
le cui colonne sono costituite dagli autovettori di  $\Sigma$  corrispondenti ai  $k$  più grandi autovalori

$$\Phi_k = [\varphi_{i_1}, \varphi_{i_2} \dots \varphi_{i_k}] \text{ con } \lambda_{i_1} \geq \lambda_{i_2} \geq \dots \lambda_{i_k} \geq \dots \lambda_{i_d}$$

$\varphi_{ir}$  autovettore di  $\Sigma$  corrispondente all'autovalore  $\lambda_{ir}$   $r = 1 \dots d$

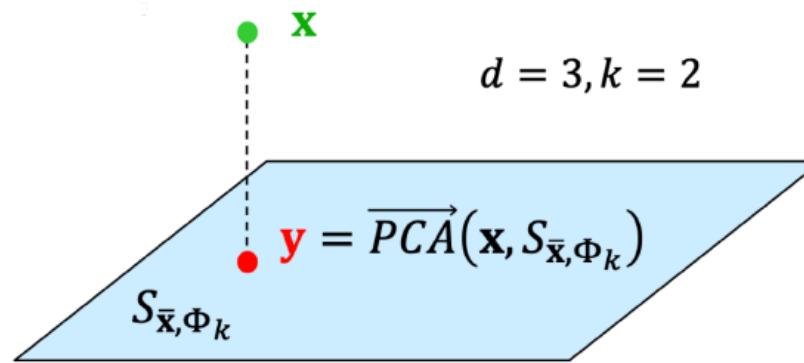
**I primi  $k$  autovettori sono detti componenti principali (PC)**

$\varphi_{i1}$  indica la direzione di maggior varianza nel training set



# Feature Extraction | Principal Components Analysis

$\Re_d \rightarrow \Re_k$



**Proiezione** Una volta determinato lo spazio PCA, la proiezione di un pattern  $x$  su tale spazio è semplicemente la proiezione geometrica del vettore  $x$  sull'iperpiano che definisce lo spazio. In realtà la vera proiezione geometrica è un vettore che ha la stessa dimensionalità del vettore originale mentre in questo contesto indichiamo con proiezione il vettore (ridotto) nello spazio PCA. Matematicamente questa operazione è eseguita come prodotto della matrice di proiezione trasposta per il pattern  $x$  al quale è preventivamente sottratta la media.

$$\overrightarrow{PCA}(\bar{x}, S_{\bar{x}, \Phi_k}) = \Phi_k^t(x - \bar{x})$$

$\Re_k \rightarrow \Re_d$

**Retroproiezione** Dato un vettore  $y$  nello spazio PCA, la sua retro-proiezione verso lo spazio originale si ottiene moltiplicando il vettore per la matrice di proiezione e sommando il vettore medio. Questa trasformazione non sposta spazialmente il vettore, che giace ancora sullo spazio PCA, ma opera un cambiamento di coordinate che ne permette la codifica in termini delle  $d$  componenti dello spazio originale.

$$\overleftarrow{PCA}(y, S_{\bar{x}, \Phi_k}) = \Phi_k y + \bar{x}$$

# Feature Extraction | Principal Components Analysis

Se l'obiettivo è quello di scartare informazione inutile e dati correlati mantenendo gran parte del contenuto informativo si può scegliere  $k$  nel modo seguente:

Fissata una percentuale  $t$  del contenuto informativo che si vuole preservare (es.  $t = 95\%$ ) si sceglie il minimo valore di  $k$  per cui la somma dei più grandi  $k$  autovalori è maggiore o uguale a  $t$  rispetto alla somma di tutti gli autovalori

Considerando gli autovalori ordinati in ordine decrescente:

$$k = \arg \min_z \left\{ \frac{\sum_{i=1 \dots z} \lambda_i}{\sum_{i=1 \dots d} \lambda_i} \geq t \right\}$$

Poiché gli autovalori denotano la varianza lungo i diversi assi, il rapporto nella formula indica la varianza conservata rispetto alla varianza totale

La scelta di  $k$  è obbligata ad esempio per la visualizzazione 2D o 3D dei dati ( $k=2, 3$ )

# Feature Extraction | Principal Components Analysis

Per  $d$  elevato (tipico nel caso di immagini, audio, ecc.) la matrice di covarianza può essere molto grande

Esempio per  $d = 16384$ ,  $\Sigma \in \mathbb{R}^{16384 \times 16384}$  oltre 268 milioni di valori !

E' più conveniente calcolare la matrice di proiezione attraverso la decomposizione ai valori singolari (Single Value Decomposition, SVD) della matrice rettangolare degli  $n$  pattern centralizzati  $\mathbf{X} \in \mathbb{R}^{d \times n}$ ,  $n \ll d$  senza passare per la matrice di covarianza

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \mathbf{x}_2 - \bar{\mathbf{x}} & \cdots & \mathbf{x}_n - \bar{\mathbf{x}} \end{bmatrix}$$

SVD per  $n \ll d$ :  $\mathbf{X} = \mathbf{U}\Gamma\mathbf{V}^t$ , con  $\mathbf{U} \in \mathbb{R}^{d \times n}$  ortonormale unitaria,  $\Gamma \in \mathbb{R}^{n \times n}$  diagonale,  $\mathbf{V} \in \mathbb{R}^{n \times n}$  ortonormale

Gli elementi di  $\Gamma$  sono i valori singolari di  $\mathbf{X}$ , gli elementi di  $\mathbf{U}$  sono i vettori singolari sinistri di  $\mathbf{X}$  e gli elementi di  $\mathbf{V}$  sono i vettori singolari destri di  $\mathbf{X}$ . Si verifica che:

I vettori singolari di sinistra di  $\mathbf{X}$  sono gli autovettori di  $\mathbf{X}\mathbf{X}^t$ , i vettori singolari di destra di  $\mathbf{X}$  sono gli autovettori di  $\mathbf{X}^t\mathbf{X}$ , i vettori singolari non nulli di  $\mathbf{X}$  (che si trovano sulla diagonale principale di  $\Gamma$ ) sono le radici quadrate degli autovalori non nulli di  $\mathbf{X}\mathbf{X}^t$  e  $\mathbf{X}^t\mathbf{X}$

$$\Sigma = \frac{1}{n} \mathbf{X}\mathbf{X}^t = \frac{1}{n} \mathbf{U}\Gamma\mathbf{V}^t\mathbf{V}\Gamma\mathbf{U}^t = \frac{1}{n} \mathbf{U}\Gamma^2\mathbf{U}^t$$

Gli autovettori e gli autovalori di  $\Sigma$  possono dunque essere ottenuti dalle colonne di  $\mathbf{U}$  (vettori singolari sinistri di  $\mathbf{X}$ ) e corrispondenti elementi diagonali di  $\Gamma^2$  (valori singolari al quadrato di  $\mathbf{X}$ ).

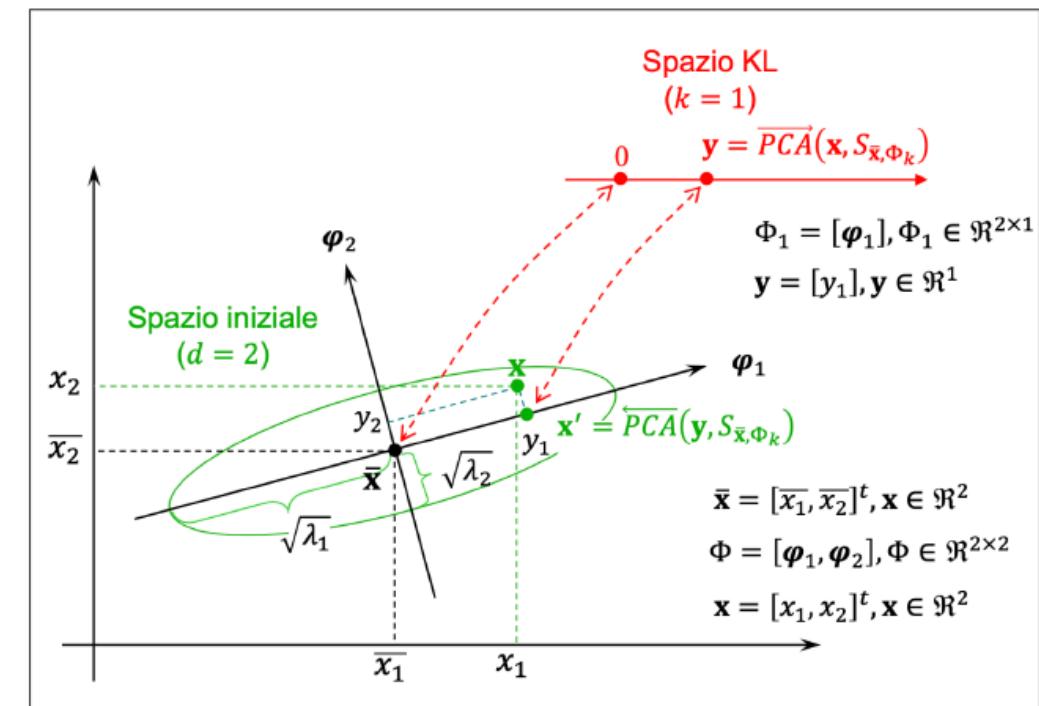
# Feature Extraction | Principal Components Analysis

L'ellisse rappresenta la distribuzione dei pattern nel training set

$\varphi_1$  e  $\varphi_2$  sono gli autovettori della matrice di covarianza

Gli autovalori  $\lambda_1$  e  $\lambda_2$  sono le varianze della distribuzione lungo gli assi  $\varphi_1$  e  $\varphi_2$ .

$y_1$  e  $y_2$  sono le proiezioni di  $x$  sugli assi  $\varphi_1$  e  $\varphi_2$ .

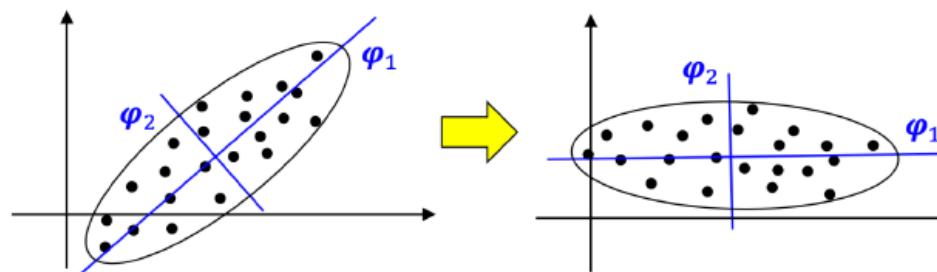


se  $\lambda_2$  è piccolo,  $x$  può essere approssimato con  $x'$  (retroproiezione di  $y$ ) senza perdite significative di informazione

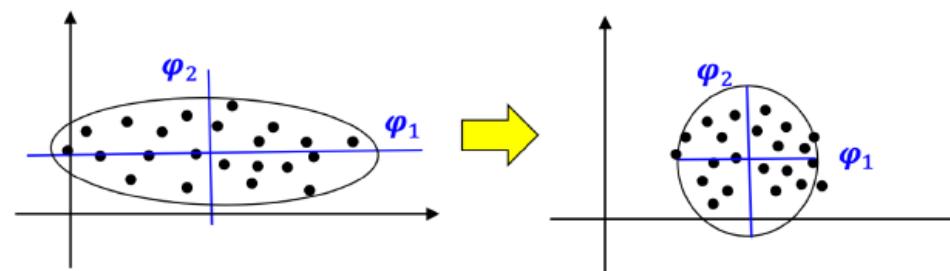
# Feature Extraction | Principal Components Analysis

È una tecnica di pre-normalizzazione dei dati, che:

- Rimuove le correlazioni tra le dimensioni, ruotando la nuvola di punti per allineare gli assi di variazione principale dei dati (autovettori) agli assi cartesiani.



- Sferizza l'ellissoide, uniformando le varianze (denotate dagli autovalori) a 1 lungo tutti gli assi



- Dopo aver proiettato i pattern sullo spazio PCA (definito dai primi  $k$  autovettori) è sufficiente dividere ogni dimensione per la radice quadrata dell'autovalore corrispondente (deviazione standard).

*La matrice di covarianza dei dati normalizzati è l'identità*

# Feature Extraction | Principal Components Analysis

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{n}$$

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

$$s = \sqrt{\frac{\sum_{i=1}^n (X_i - \bar{X})^2}{(n-1)}}$$

**Eigenvectors**

**Eigenvalues**

$$s^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{(n-1)}$$

$$var(X) = \frac{\sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

$$cov(X, Y) = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

**Step 5: Choosing components and forming a feature vector**

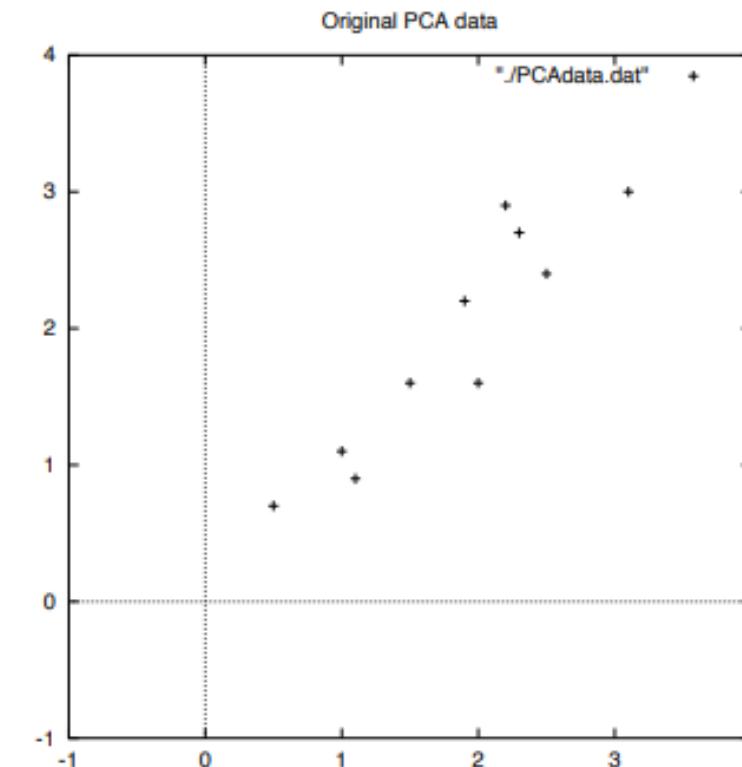
**Step 5: Deriving the new data set**

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

## Step 1: Get some data

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
Data = 3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

x	y	x	y
2.5	2.4	.69	.49
0.5	0.7	-1.31	-1.21
2.2	2.9	.39	.99
1.9	2.2	.09	.29
Data =	3.1	1.29	1.09
	2.3	.49	.79
	2	.19	-.31
	1	-.81	-.81
	1.5	-.31	-.31
	1.1	-.71	-1.01

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

	<i>x</i>	<i>y</i>
	.69	.49
	-1.31	-1.21
	.39	.99
	.09	.29
DataAdjust =	1.29	1.09
	.49	.79
	.19	-.31
	-.81	-.81
	-.31	-.31
	-.71	-1.01

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

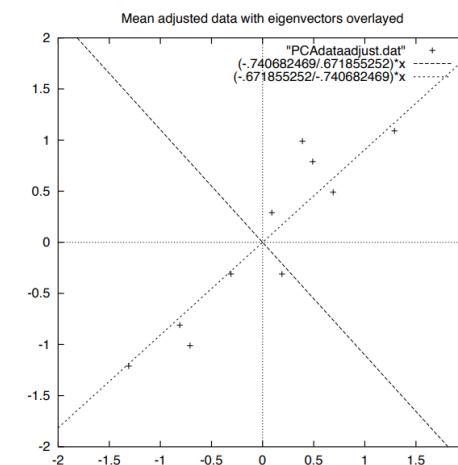


Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlaid on top.

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

	x	y
	.69	.49
	-1.31	-1.21
	.39	.99
	.09	.29
DataAdjust =	1.29	1.09
	.49	.79
	.19	-.31
	-.81	-.81
	-.31	-.31
	-.71	-1.01

$$\text{eigenvalues} = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$\text{eigenvectors} = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

$$\text{cov} = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

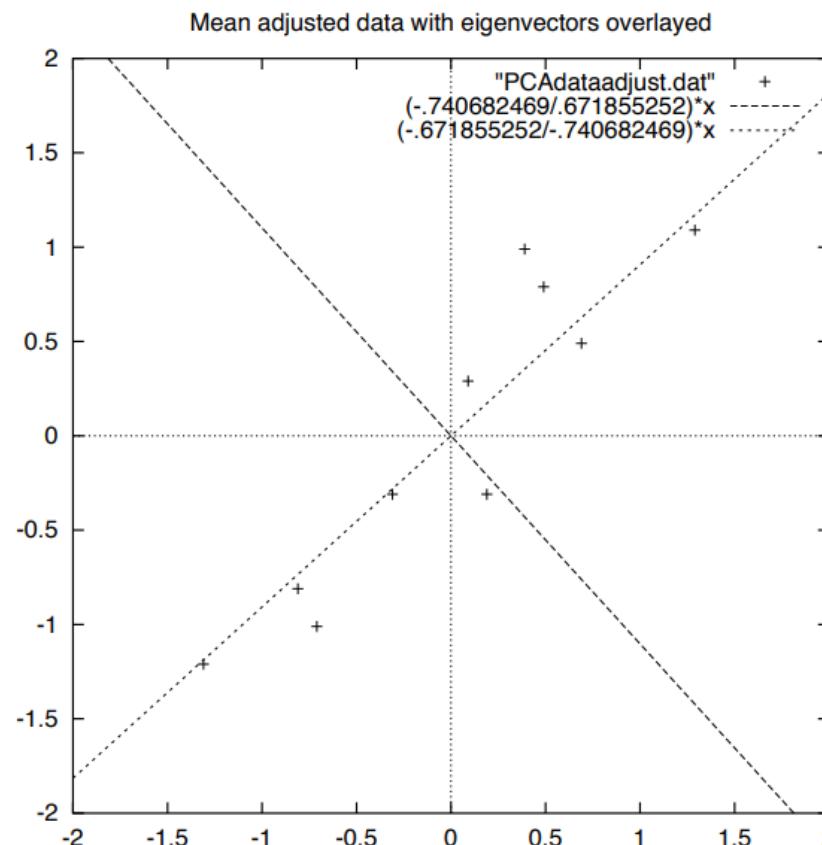


Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlayed on top.

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

$$\text{FeatureVector} = (eig_1 \ eig_2 \ eig_3 \ \dots \ eig_n)$$

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

**Step 5: Choosing components and forming a feature vector**

	x	y
	.69	.49
	-1.31	-1.21
	.39	.99
	.09	.29
DataAdjust =	1.29	1.09
	.49	.79
	.19	-.31
	-.81	-.81
	-.31	-.31
	-.71	-1.01

$$\text{eigenvalues} = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$\text{eigenvectors} = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

$$\text{cov} = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

**Step 5: Choosing components and forming a feature vector**

**Step 5: Deriving the new data set**

$$\text{FinalData} = \text{RowFeatureVector} \times \text{RowDataAdjust},$$

where *RowFeatureVector* is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and *RowDataAdjust* is the mean-adjusted data *transposed*, ie. the data items are in each column, with each row holding a separate dimension.

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

**Step 5: Choosing components and forming a feature vector**

**Step 5: Deriving the new data set**

$$\text{FinalData} = \text{RowFeatureVector} \times \text{RowDataAdjust},$$

where *RowFeatureVector* is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and *RowDataAdjust* is the mean-adjusted data *transposed*, ie. the data items are in each column, with each row holding a separate dimension.

	x	y
	-.827970186	-.175115307
	1.77758033	.142857227
	-.992197494	.384374989
	-.274210416	.130417207
Transformed Data=	-1.67580142	-.209498461
	-.912949103	.175282444
	.0991094375	-.349824698
	1.14457216	.0464172582
	.438046137	.0177646297
	1.22382056	-.162675287

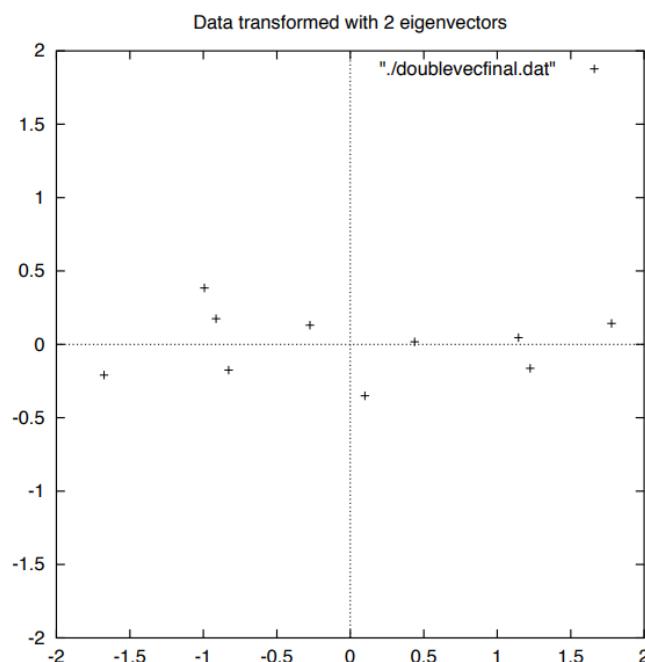


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

**Step 5: Choosing components and forming a feature vector**

**Step 5: Deriving the new data set**

Transformed Data (Single eigenvector)

$x$
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056

$$\text{FinalData} = \text{RowFeatureVector} \times \text{RowDataAdjust},$$

where *RowFeatureVector* is the matrix with the eigenvectors in the columns *transposed* so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and *RowDataAdjust* is the mean-adjusted data *transposed*, ie. the data items are in each column, with each row holding a separate dimension.

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

$$FinalData = RowFeatureVector \times RowDataAdjust$$

**Step 2: Subtract the mean**

$$RowDataAdjust = RowFeatureVector^{-1} \times FinalData$$

**Step 3: Calculate the covariance matrix**

$$RowDataAdjust = RowFeatureVector^T \times FinalData$$

**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

only true if the elements of the matrix are all unit eigenvectors

**Step 5: Choosing components and forming a feature vector**

$$RowDataAdjust = RowFeatureVector^T \times FinalData$$

**Getting the old data back**

$$RowOriginalData = (RowFeatureVector^T \times FinalData) + OriginalMean$$

A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis

**Step 1: Get some data**

**Step 2: Subtract the mean**

**Step 3: Calculate the covariance matrix**

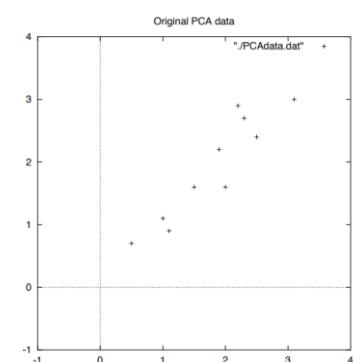
**Step 4: Calculate the eigenvectors and eigenvalues of the covariance matrix**

**Step 5: Choosing components and forming a feature vector**

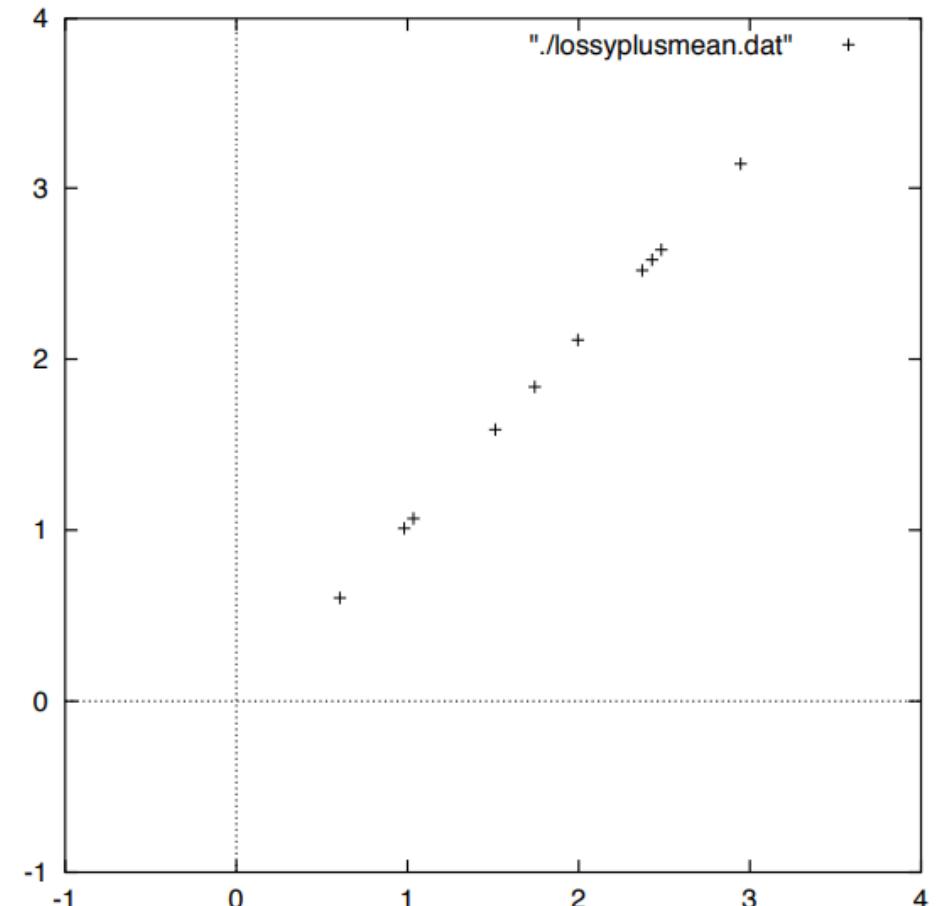
**Step 5: Deriving the new data set**

**Getting the old data back**

$$\text{RowOriginalData} = (\text{RowFeatureVector}^T \times \text{FinalData}) + \text{OriginalMean}$$

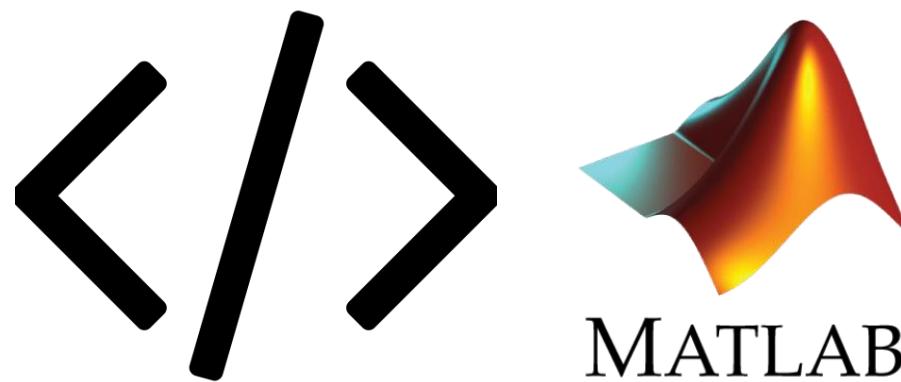


Original data restored using only a single eigenvector



A tutorial on Principal Components Analysis by Lindsay I. Smith

# Feature Extraction | Principal Components Analysis



# Feature Extraction

- Accuracy improvements.
- Overfitting risk reduction.
- Speed up in training.
- Improved Data Visualization.
- Increase in explainability of our model.

Many other  
techniques...

# FEATURE SELECTION

# Linear Discriminant Analysis

Riduzione di dimensionalità lineare e supervisionata il cui obiettivo è massimizzare la separazione tra le classi.

Per formulare il criterio di ottimizzazione di massima separazione tra le classi sono definite le seguenti matrici di scattering (sparpagliamento):

*within-class*

indica come i vettori sono scattered rispetto al centro delle classi (ciascuno rispetto alla propria classe).

*between-class*

indica come i centri delle classi sono scattered rispetto al centro generale della distribuzione (ovvero quanto le classi sono scattered).

# Linear Discriminant Analysis

Dato un training set contenente  $n$  pattern  $(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n)$ ,

dove  $\mathbf{x}_i \in \Re^d$  sono i pattern multidimensionali e  $y_i \in [1 \dots s]$  le etichette delle  $s$  classi.

Siano  $n_i$  e  $\bar{\mathbf{x}}$  il numero di pattern e il vettore medio della classe  $i$ -esima.

Allora le matrici di scattering sono definite come:

*within-class*

$$\mathbf{S}_w = \sum_{i=1 \dots s} \mathbf{S}_i, \quad \mathbf{S}_i = \sum_{\mathbf{x}_j | y_j=i} (\mathbf{x}_j - \bar{\mathbf{x}}_i)(\mathbf{x}_j - \bar{\mathbf{x}}_i)^t$$

matrice di covarianza senza  
normalizzare per il numero di pattern

*pattern della classe  $i$  – esima*

|

*between-class*

$$\mathbf{S}_b = \sum_{i=1 \dots s} n_i \cdot (\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_0)(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_0)^t, \quad \bar{\mathbf{x}}_0 = \frac{1}{n} \sum_{i=1 \dots s} n_i \cdot \bar{\mathbf{x}}_i$$

*pattern della classe  $i$  – esima*

*media globale*

# Linear Discriminant Analysis

Il criterio per la soluzione ottimale e' intuitivo in quanto cerca di massimizzare lo scattering tra le classi (between class  $S_b$ ) minimizzando al contempo quello all'interno di ogni classe (within class,  $S_w$ )

Il che equivale a massimizzare la quantità:

$$J_1 = \text{tr}(\mathbf{S}_w^{-1} \mathbf{S}_b) = \sum_{i=1 \dots d} \lambda_i$$

ove  $\text{tr}$  è la traccia (somma degli autovalori) della matrice.

Si dimostra che per massimizzare  $J_1$  lo spazio LDA e' definito dagli autovettori relativi ai primi  $k$  autovalori della matrice  $\mathbf{S}_w^{-1} \mathbf{S}_b$  ( $k < n$ ,  $k < s$ ,  $k < d$ ) (analogia con PCA)

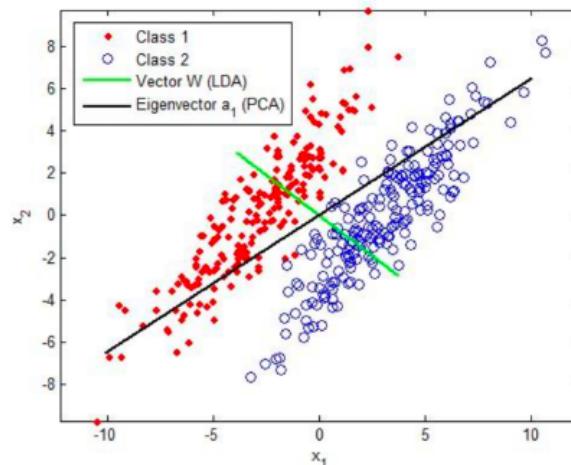


Valore massimo di  $k = s - 1$

# Linear Discriminant Analysis vs. Principal Components Analysis

Riduzione di dimensionalità da  $d = 2$  a  $k = 1$

Sono eseguiti mapping lineari  $\mathbb{R}^2 \rightarrow \mathbb{R}^1$  ma la soluzione (retta) è profondamente diversa.



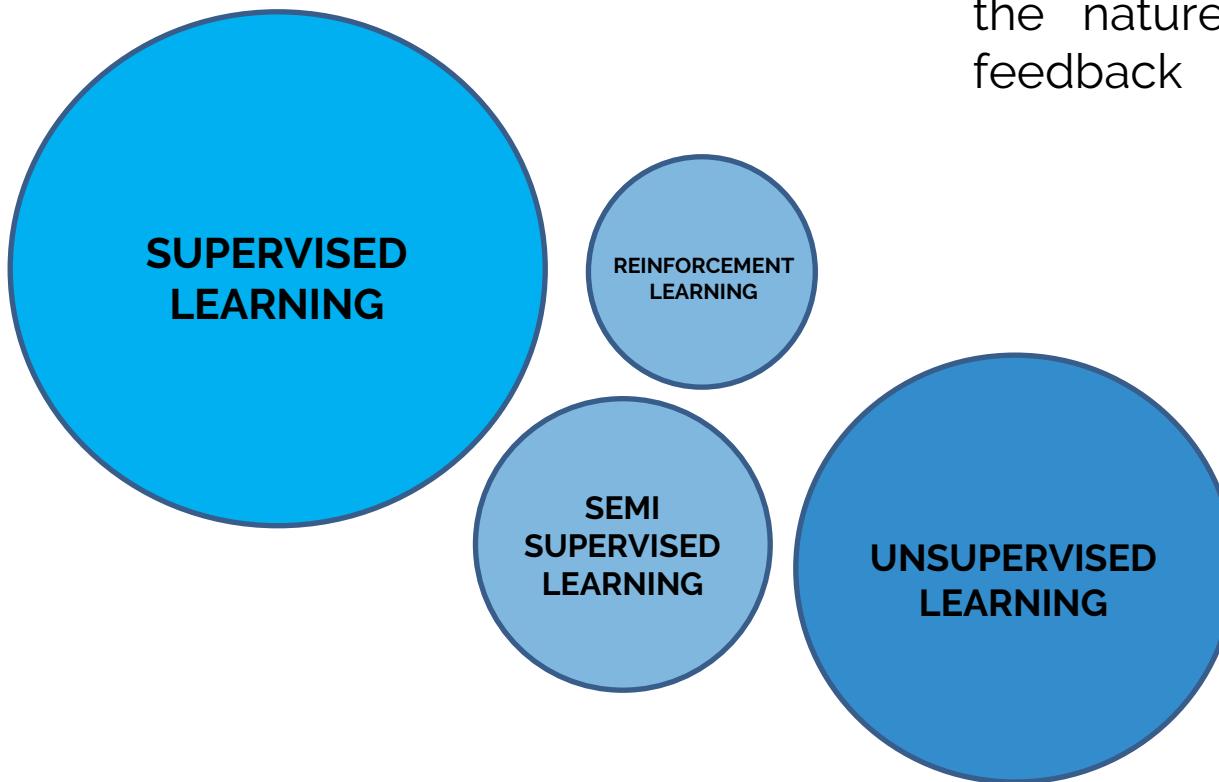
Il segmento nero che identifica la soluzione PCA è l'iperpiano sul quale proiettando i pattern (indipendentemente dalla loro classe) conserviamo al massimo l'informazione.

Il segmento verde che identifica la soluzione LDA è l'iperpiano sul quale proiettando i pattern siamo in grado di discriminare al meglio le due classi

*Mentre PCA privilegia le dimensioni che rappresentano al meglio i pattern, LDA privilegia le dimensioni che discriminano al meglio i pattern del TS.*

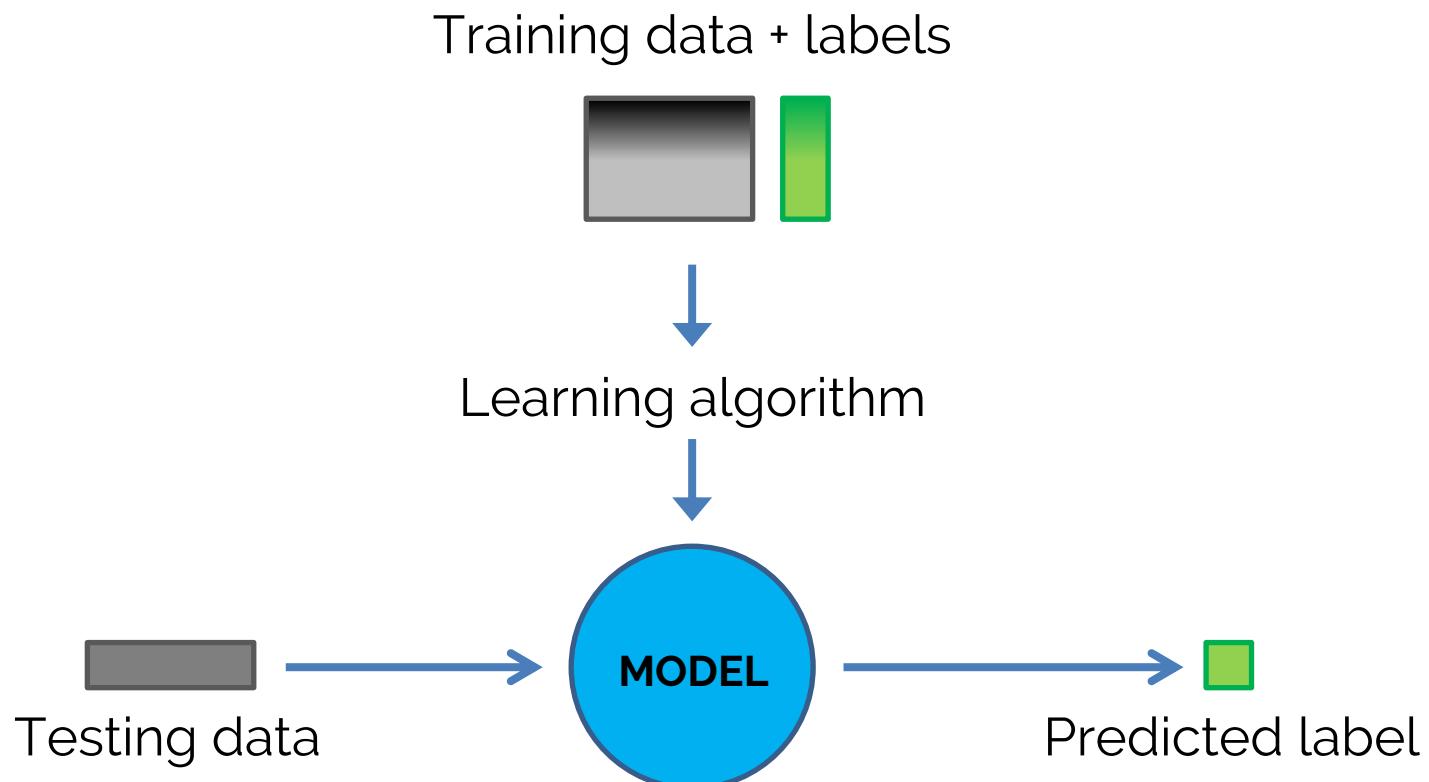
# CLASSIFICATION

# Machine learning

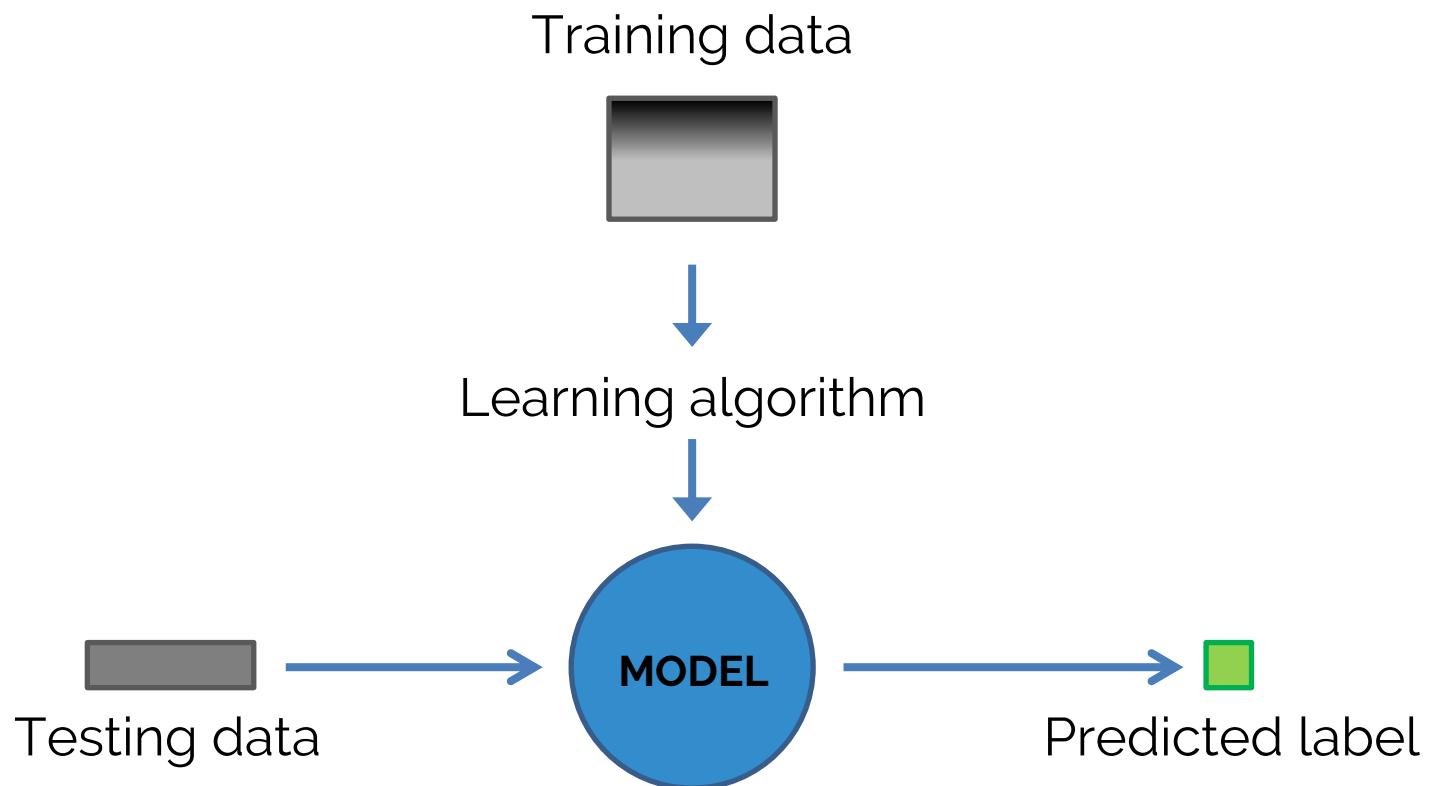


Categorization by  
the nature of the  
feedback

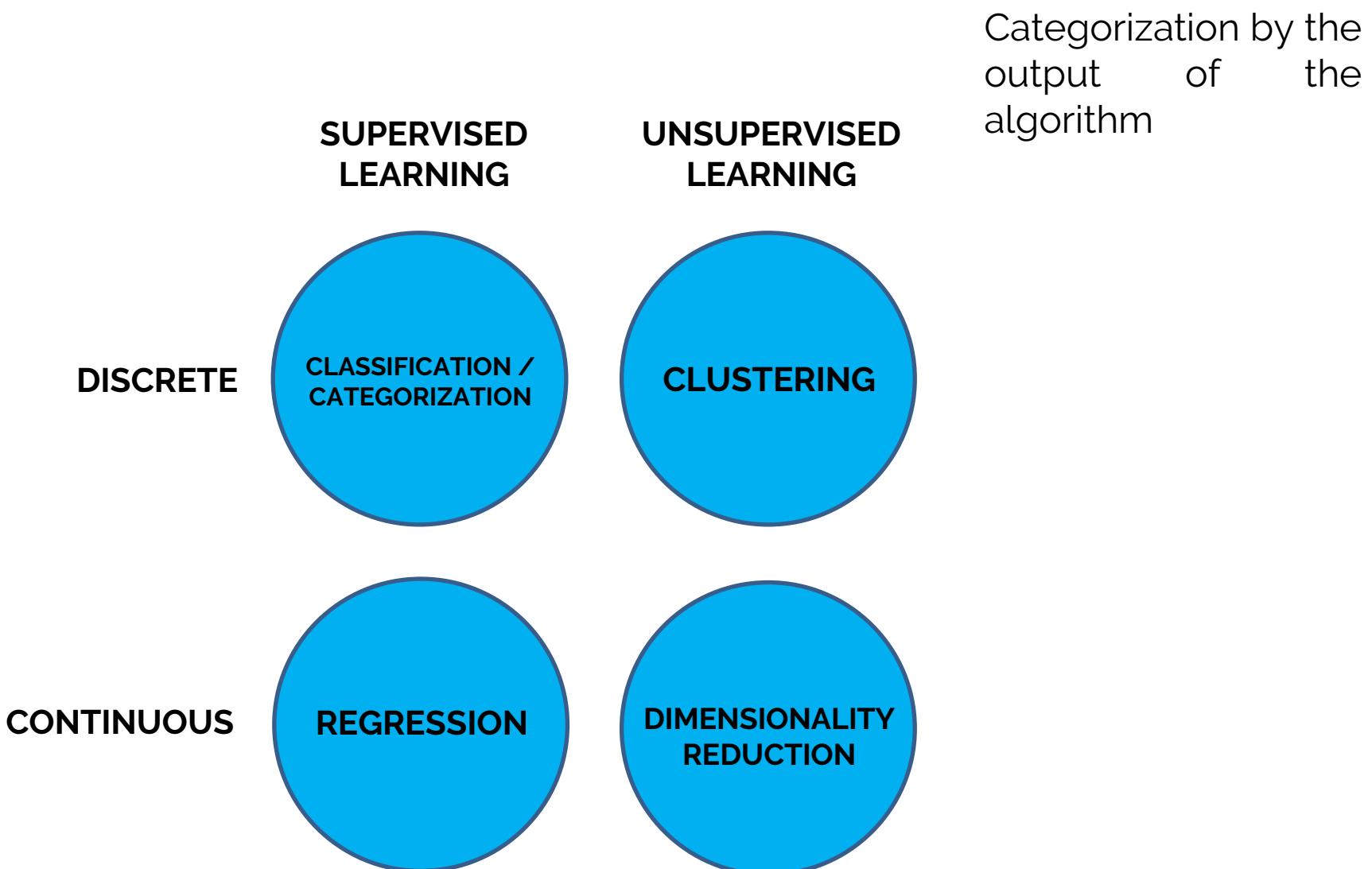
# Supervised learning



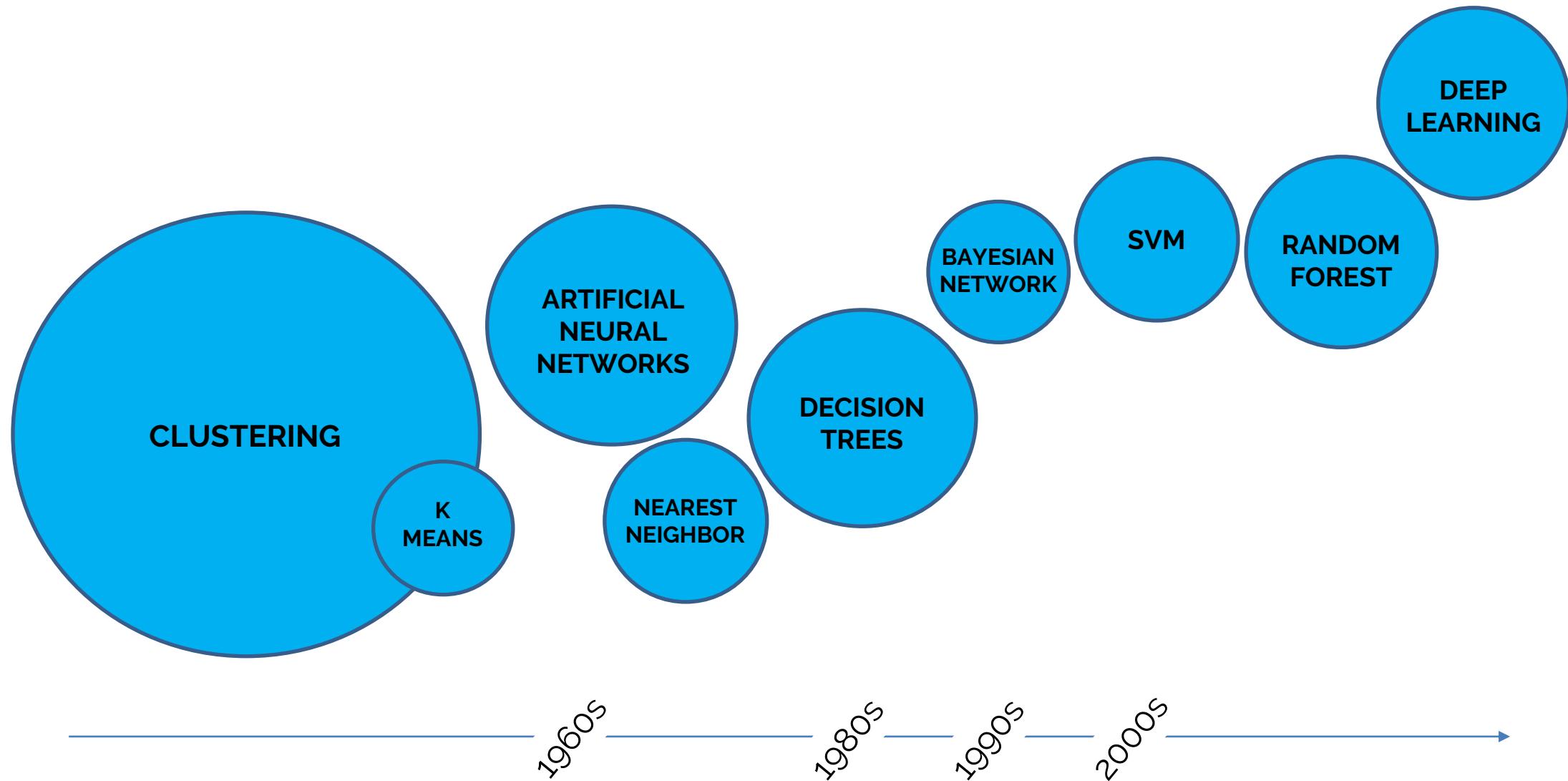
# Unsupervised learning



# Machine learning



# Machine learning



# UNSUPERVISED CLASSIFICATION

## CLUSTERING

# Definizioni

- Definizioni, Criteri, Algoritmi
- Centroid-based Clustering
  - K-means
  - Fuzzy k-means
  - Expectation-Maximization (Gaussian Mixture)

Clustering (raggruppamento), Robert Tyron 1939

Famiglia di metodi non supervisionati in grado di individuare raggruppamenti intrinseci (cluster) di pattern nello spazio multidimensionale, e (opzionalmente) di definire in corrispondenza di tali raggruppamenti le classi (incognite).

Il problema è molto complesso: determinare la soluzione ottimale con ricerca esaustiva è possibile solo nei casi di pattern limitati

# Dimensionalità

Nel caso  $c = 2$ , dati 4 pattern {A, B, C, D} e  $d = 2$ , il numero di soluzioni possibili è 7

- (A) (B,C,D)
- (B) (A,C,D)
- (C) (A,B,D)
- (D) (A,B,C)
- (A,B) (C,D)
- (A,C) (B,D)
- (A,D) (B,C)

# Dimensionalità

Dati  $n$  pattern, assumendo di conoscere  $c$ , il numero di soluzioni è dell'ordine di  $c^n/c!$  (approssimazione numero di Stirling di seconda specie\*).

Esempio: per  $n = 100$ ,  $c = 5$ , il numero di soluzioni è 1067.

Se  $c$  non è fissato a priori, il numero di soluzioni è dato dal numero di Bell, ottenibile sommando tutti casi per tutti valori di  $c$  da 1 a  $n$ .

\*il numero di Stirling di seconda specie è il numero di modi in cui  $n$  oggetti distinguibili possono essere suddivisi tra  $k$  sottoinsiemi disgiunti e non vuoti.

La maggior parte dei criteri di clustering sono definiti sulla base delle due osservazioni seguenti:

- 1) i cluster sono costituiti da nuvole di punti a densità relativamente elevata separate da zone dove la densità è più bassa;
- 2) i pattern all'interno dello stesso cluster devono essere tra loro più simili rispetto a pattern appartenenti a cluster diversi

- Minimizzazione delle distanze dai centroidi

Minimizza la somma dei quadrati delle distanze dei pattern  $x$  dai centroidi (i.e. baricentri) delle classi

$$J_e = \sum_{i=1..s} \sum_{x \in C_i} \|x - \bar{x}_i\|^2, \quad \bar{x}_i = \frac{1}{n_i} \sum_{x \in C_i} x$$

dove  $C_i$  è l' $i$ -esimo cluster,  $n_i$  il numero di pattern che il cluster contiene e  $\bar{x}_i$  il suo centroide (media).

È un buon criterio per cluster a simmetria radiale (i.e., circolari), ma penalizza forme allungate o cluster innestati (i.e. un cluster a forma di anello con all'interno un altro cluster)

- Minimizzazione distanze intra-classe

$$J_e = \sum_{i=1..s} n_i \cdot \bar{s}_i, \quad \bar{s}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in C_i} f_s(\mathbf{x}, C_i)$$

dove  $f_s$  è una misura di distanza tra  $\mathbf{x}$  e il cluster a cui appartiene. Ad esempio:

$$f_s(\mathbf{x}, C_i) = \frac{1}{n_i} \sum_{\mathbf{x}' \in C_i} \|\mathbf{x} - \mathbf{x}'\|^2 \quad \text{criterio simile alla minimizzazione distanze dai centroidi}$$

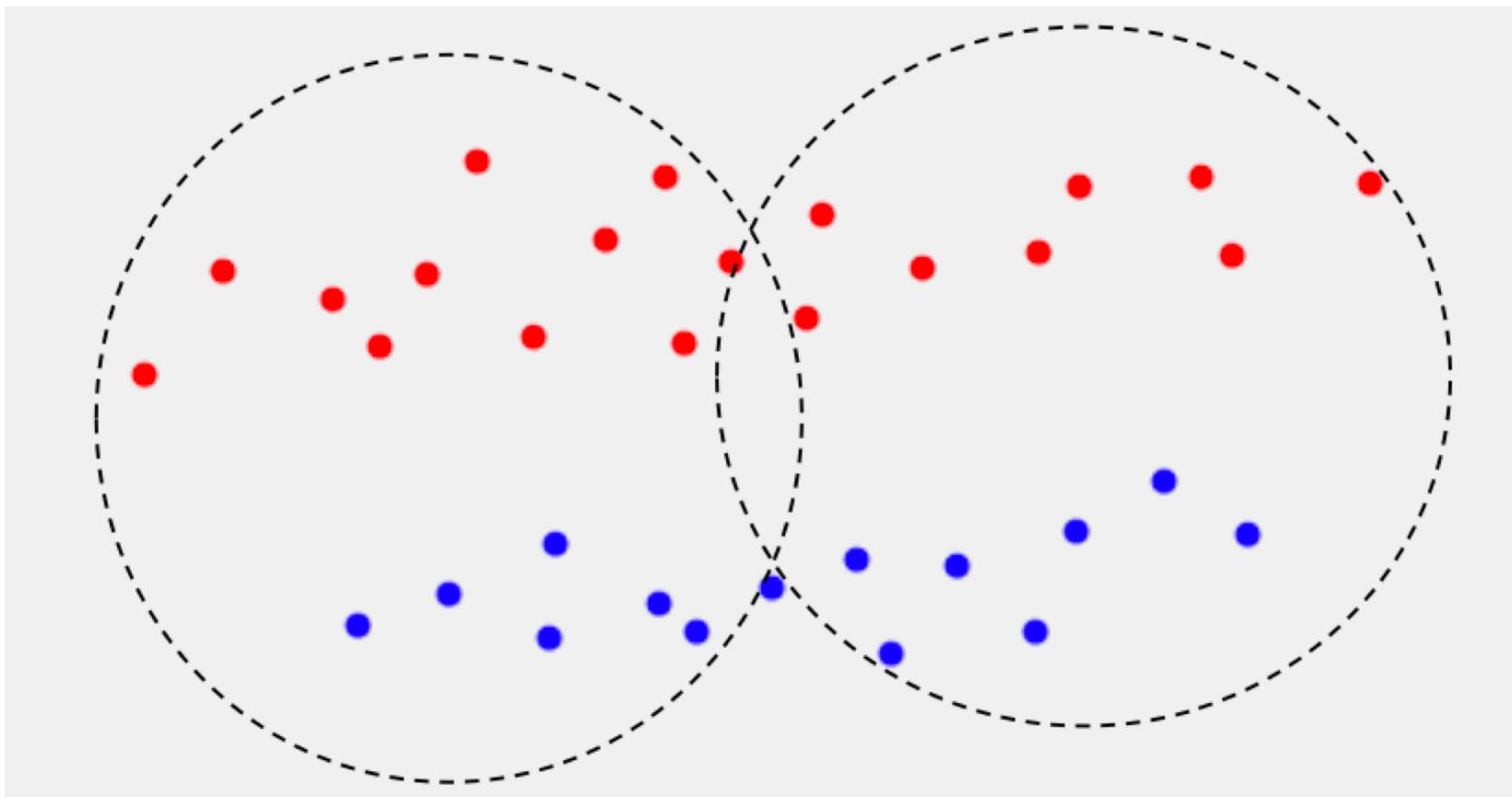
$$f_s(\mathbf{x}, C_i) = \min_{\substack{\mathbf{x}' \in C_i \\ \mathbf{x}' \neq \mathbf{x}}} \|\mathbf{x} - \mathbf{x}'\|^2$$

non penalizza cluster allungati affinché  $f_s$  assuma valore ridotto, non è necessario che tutti (o gran parte de) i pattern di  $C$  siano vicini a  $x$ , ma è sufficiente un vicino.

# Criteri

$C=2$

- 1) Minimizzazione distanze tra i centroidi: aggregazione come da cerchi tratteggiati
- 2) Minimizzazione della distanza dei vicini: come da colori rosso e blu



# Algoritmi

- Clustering basato su centroidi

Attraverso successive iterazioni, si individuano i cluster cercando di minimizzare la distanza dei pattern dai centroidi dei cluster cui appartengono:

- K-means
- Fuzzy K-means
- Expectation – Maximization (Gaussian Mixture)

- Clustering gerarchico

Attraverso operazioni che aggregano pattern in base a una misura di distanza, si organizzano i dati in struttura ad albero (dendogramma)

- Clustering basato sulla densità:

I cluster individuati sono regioni connesse in aree ad elevata densità

# Appartenenza

- Clustering hard (esclusivo)

Un pattern è assegnato (in modo esclusivo) a un solo cluster

- Clustering soft (fuzzy)

I pattern appartengono ai diversi cluster con un certo grado di appartenenza (es. tra 0 e 1)

È più efficace nel gestire pattern vicino al bordo di due o più cluster e outlier.

L'assegnazione può diventare esclusiva scegliendo, per ogni pattern, il cluster verso cui il grado di appartenenza è massimo

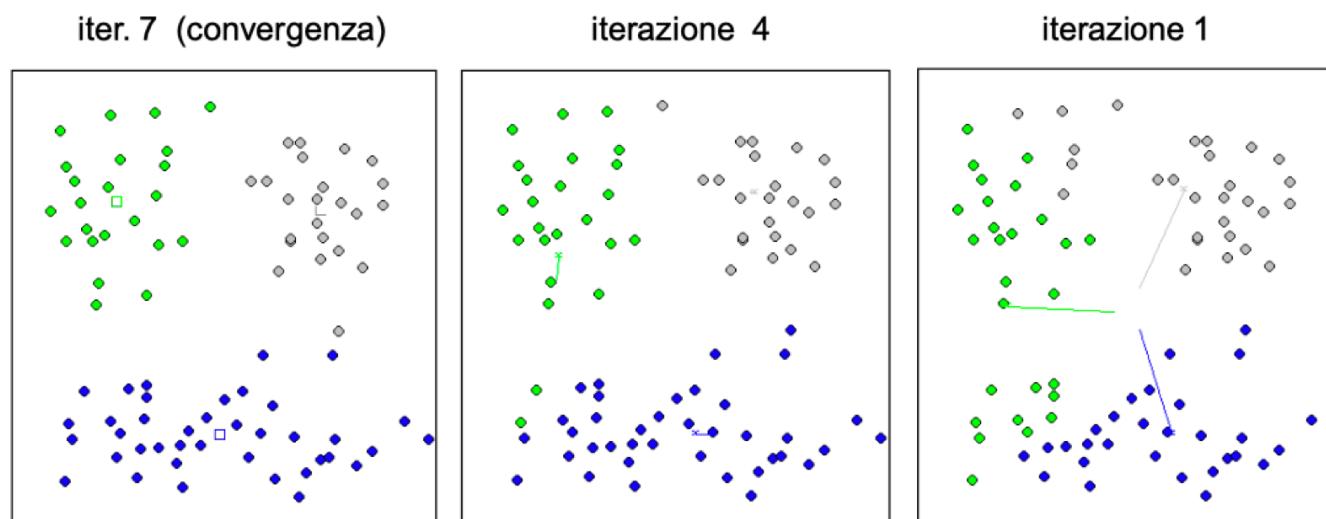
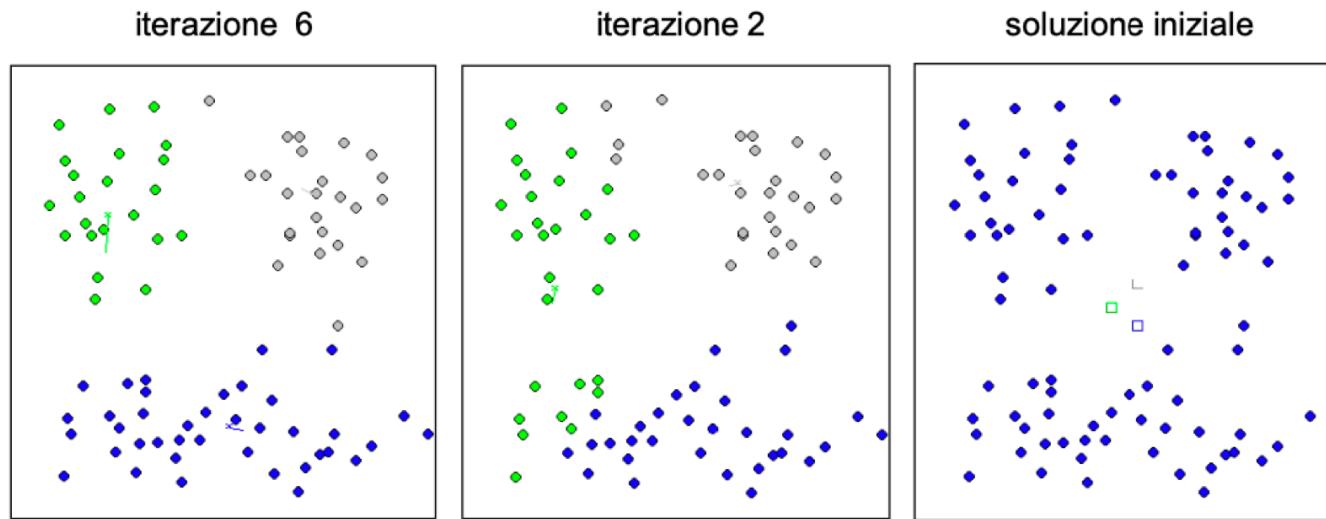
# K-means (hard)

Clustering basato sui centroidi (hard)

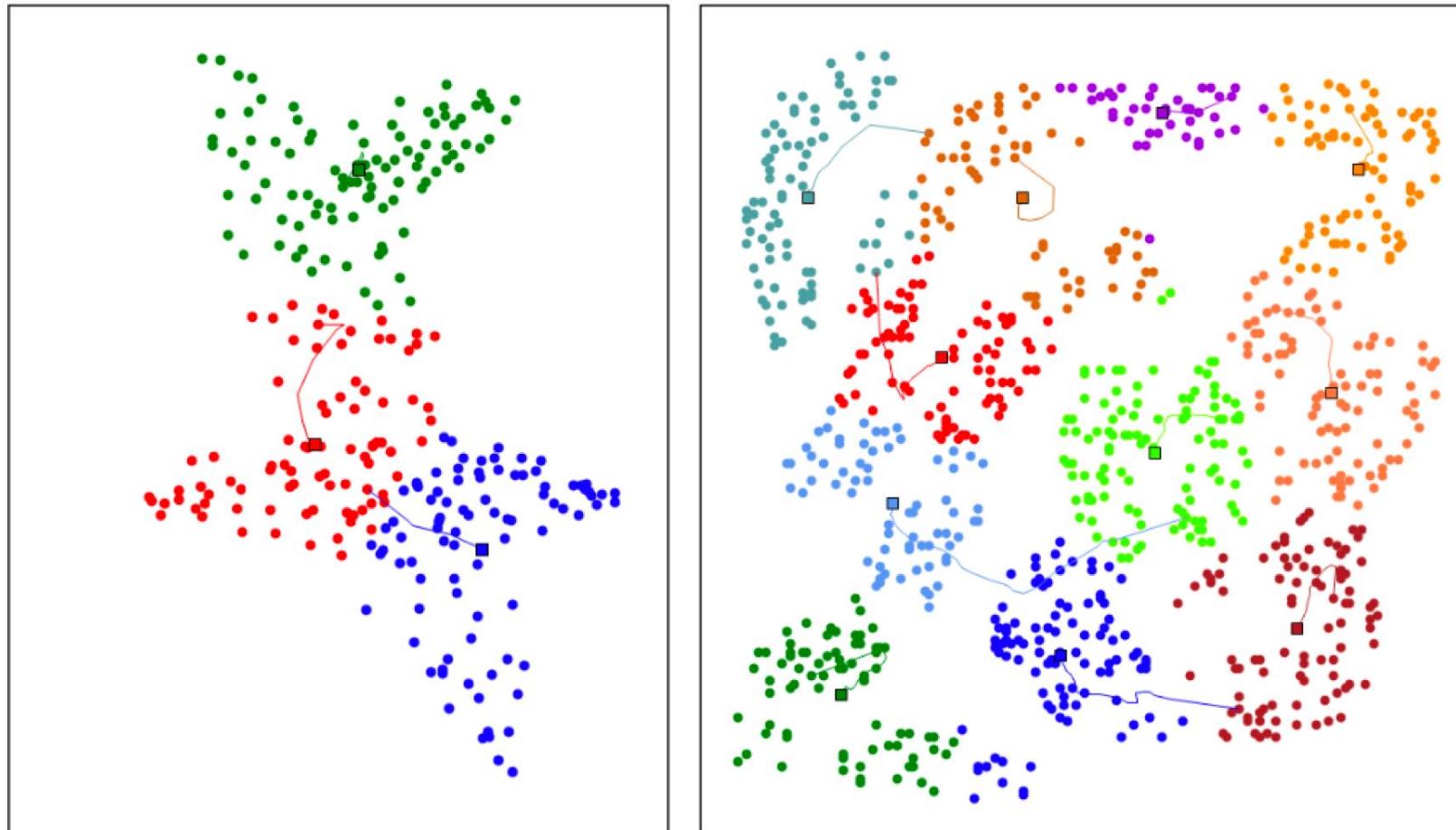
- metodo computazionalmente molto semplice e altrettanto semplice da implementare (spesso la prima scelta per risolvere problemi di clustering)
- minimizza le distanze dai centroidi
- richiede in input il numero di cluster  $c$  e una soluzione iniziale
- assegna ogni pattern al cluster per cui è minima la distanza dal centroide
- i cluster sono modificati iterativamente a seguito del ricalcolo del loro centroide.
- l'algoritmo termina (converge) quando i centroidi sono stabili e quindi le partizioni non cambiano.
- la convergenza si ottiene solitamente in pochi passi ( $< 10$ ).
- Il tipo di ottimizzazione è iterativa e locale, pertanto il metodo può convergere a massimi locali della soluzione.

Identifica cluster iper-sferici nel caso in cui venga utilizzata la distanza euclidea come misura di distanza tra i pattern o cluster iper-ellisoidali nel caso di distanza di Mahalanobis.

# K-means (hard)



## K-means (hard)



Minimizzando le distanze dai centroidi, K-means non è in grado di identificare cluster dalla forma non sferica

# K-means (hard) | Soluzione iniziale e validazione

Diverse varianti per generare buone soluzioni iniziali e di determinare il numero di classi (clustering validation).

Per minimizzare il rischio di convergenza verso minimi locali, l'algoritmo può essere eseguito più volte a partire da soluzioni iniziali diverse:

Es:

- random
- prodotte da un (diverso) euristico

Le tecniche di validazione tendono a valutare a posteriori la bontà delle soluzioni prodotte per diversi valori di  $c$ , e a sceglierne una sulla base di un criterio di validazione che tenga conto sia della bontà della soluzione sia della sua complessità

Problema: considerando il criterio di minimizzazione distanze dai centroidi e lanciando K-means con diversi valori di  $c$ , è molto più facile ottenere valori ridotti di pattern per valori elevati di  $s$ .  
Può essere utile provare a penalizzare le soluzioni con molti cluster, ad esempio introducendo penalità sul numero di pattern.

# Fuzzy K-means (soft)

La variante Fuzzy del K-means consente a un pattern di appartenere con un certo grado di probabilità a diverse classi. Il criterio di ottimizzazione in questo caso è:

$$J_{fuz} = \sum_{i=1..s} \sum_{j=1..n} [P(C_i | \mathbf{x}_j, \Theta)]^m \cdot \|\mathbf{x}_j - \boldsymbol{\theta}_i\|^2$$

dove P è la probabilità che dato il pattern  $x_j$  e l'insieme  $\Theta$  di centroidi che definiscono la soluzione attuale, il cluster di appartenenza sia  $C_i$

## Fuzzy K-means (soft)

I centrodi invece che come semplice media dei pattern si calcolano come media pesata rispetto alle probabilità di appartenenza:

$$\boldsymbol{\theta}_i = \frac{\sum_{j=1..n} P(C_i | \mathbf{x}_j, \Theta) \cdot \mathbf{x}_j}{\sum_{j=1..n} P(C_i | \mathbf{x}_j, \Theta)}$$

# Fuzzy K-means (soft)

Le probabilità di appartenenza si calcolano come:

$$P(C_i | \mathbf{x}_j, \Theta) = \frac{1}{\sum_{k=1..s} \left( \frac{\|\mathbf{x}_j - \boldsymbol{\theta}_i\|}{\|\mathbf{x}_j - \boldsymbol{\theta}_k\|} \right)^{\frac{2}{m-1}}}$$

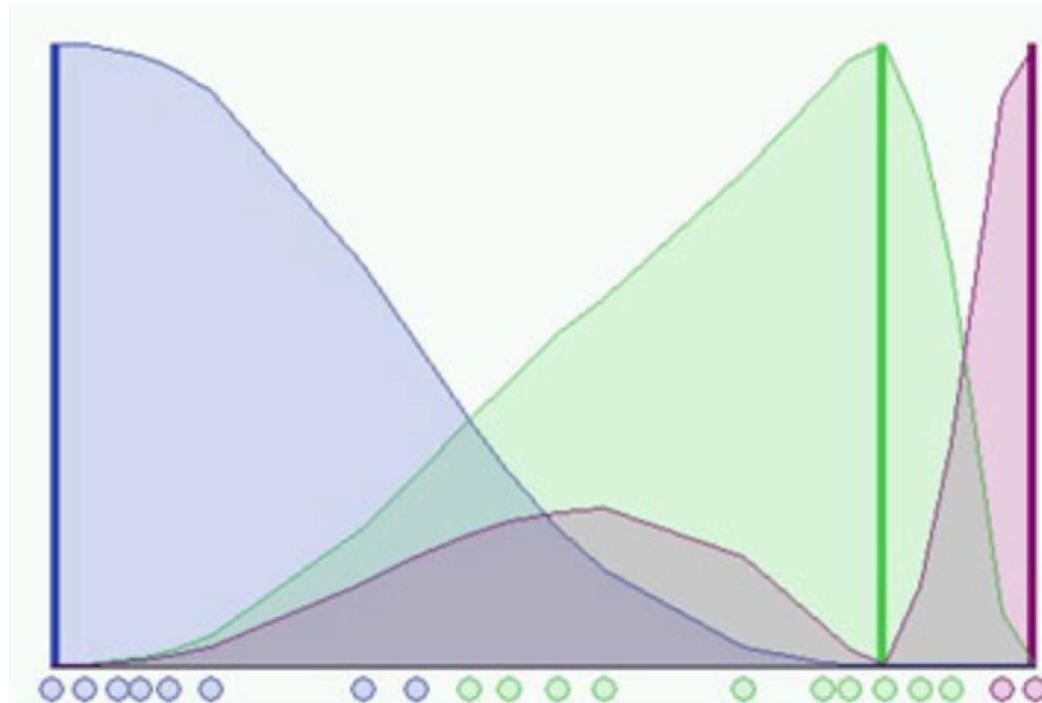
$m > 1$  è un parametro che definisce quanto l'appartenenza ai diversi cluster debba essere sfumata. Valore tipico  $m = 2$ .

# Fuzzy K-means (soft)

Caso di un'applicazione monodimensionale

$N = 20$ ,  $c = 3$  per inizializzare l'algoritmo

Le figure seguenti mostrano il valore dell'appartenenza per ciascun dato e per ciascun cluster. Il colore dei dati è quello del cluster più vicino in base alla funzione di appartenenza.



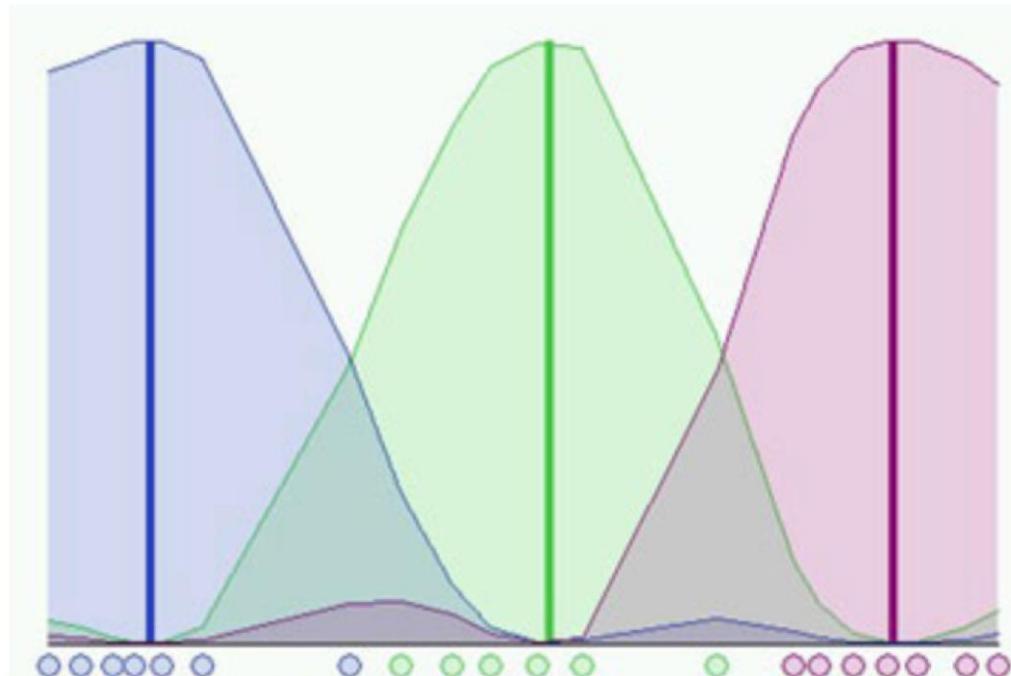
Condizione iniziale in cui la distribuzione fuzzy dipende dalla particolare posizione dei cluster

# Fuzzy K-means (soft)

Caso di un'applicazione monodimensionale

$N = 20$ ,  $c = 3$  per inizializzare l'algoritmo

Le figure seguenti mostrano il valore dell'appartenenza per ciascun dato e per ciascun cluster. Il colore dei dati è quello del cluster più vicino in base alla funzione di appartenenza.



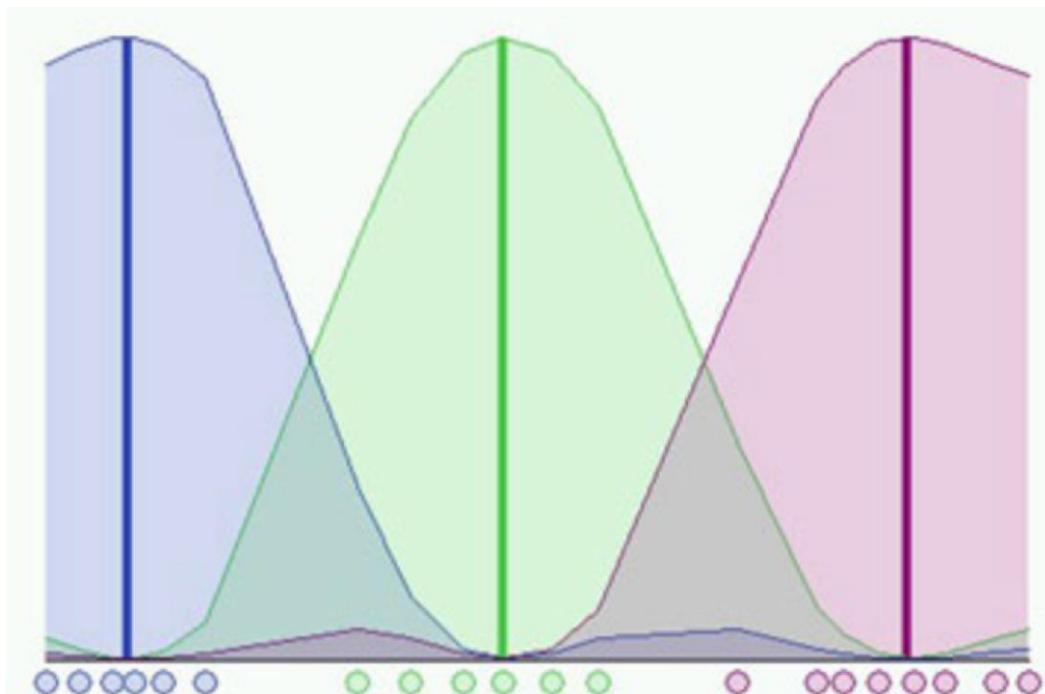
Condizione finale raggiunta  
dopo 8 iterazioni ( $m = 2$ )

# Fuzzy K-means (soft)

Caso di un'applicazione monodimensionale

$N = 20$ ,  $c = 3$  per inizializzare l'algoritmo

Le figure seguenti mostrano il valore dell'appartenenza per ciascun dato e per ciascun cluster. Il colore dei dati è quello del cluster più vicino in base alla funzione di appartenenza.



Condizione finale raggiunta  
dopo 37 iterazioni ( $m = 2$ )

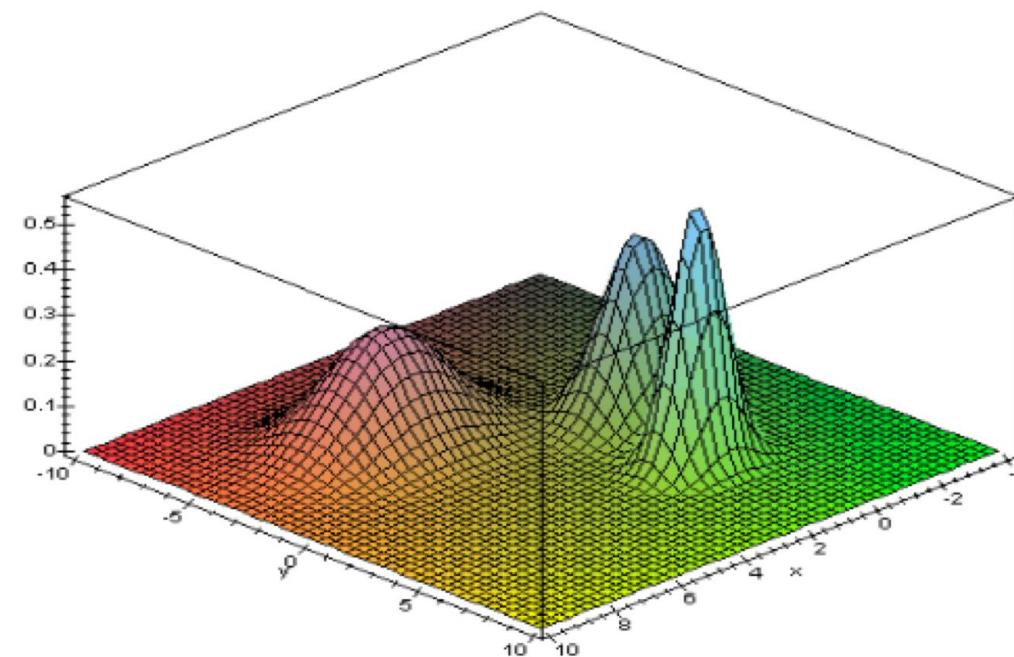
# Fuzzy K-means | Vantaggi / Svantaggi

Rispetto al K-means, la variante fuzzy fornisce a volte una convergenza più robusta verso la soluzione finale.

D'altro canto uno svantaggio è legato al fatto che l'appartenenza di un pattern a un cluster dipende implicitamente dal numero di cluster, e se questo è specificato in modo incorretto si possono ottenere soluzioni non ottimali.

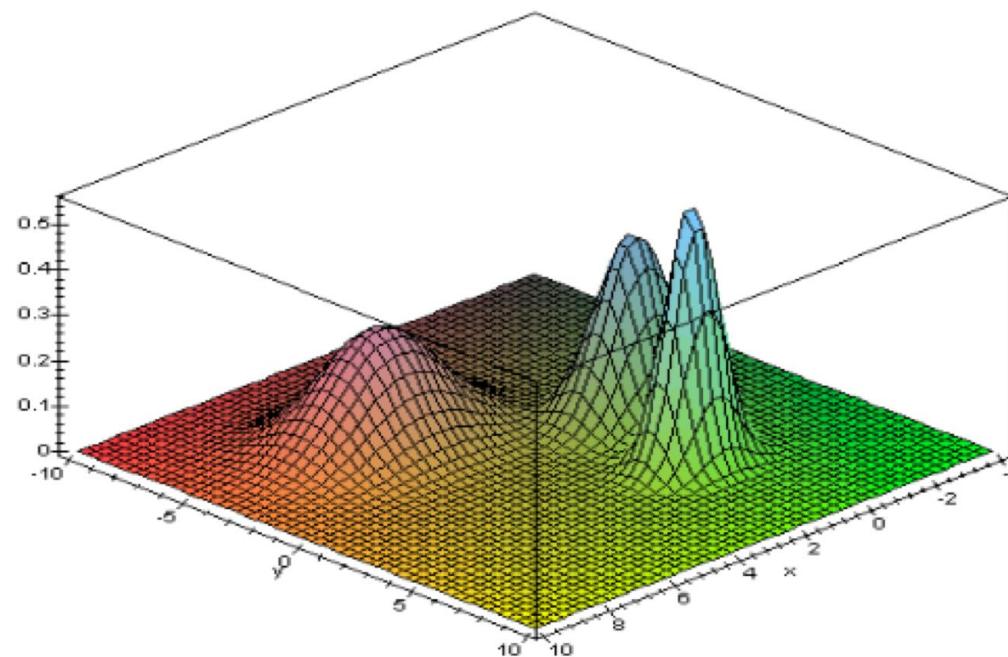
# Expectation-Minimization (EM)

Si ipotizza che i pattern siano stati generati da un mix di distribuzioni. Ogni classe ha generato dati in accordo con una specifica distribuzione, ma al termine della generazione i pattern appaiono come prodotti da un'unica distribuzione multimodale.



# Expectation-Minimization (EM)

Obiettivo del clustering con EM è risalire (a partire dai pattern del training set) ai parametri delle singole distribuzioni che li hanno generati



# Gaussian mixture

A tal fine si ipotizza nota la forma delle distribuzioni e si assume, per semplicità, che esse siano tutte dello stesso tipo.

Il caso più frequente è quello di mix di  $s$  distribuzioni multinormali (gaussiane), di cui si vogliono stimare i parametri di definizione ( $s$  vettori medi +  $s$  matrici di covarianza +  $s$  coefficienti alfa)

$$p(\mathbf{x}|\boldsymbol{\Theta}) = \sum_{i=1..s} \alpha_i \cdot p_i(\mathbf{x}|\boldsymbol{\theta}_i) \quad \boldsymbol{\theta}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$$

$\boldsymbol{\Theta} = \alpha_1, \alpha_2 \dots \alpha_s, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \dots \boldsymbol{\theta}_s$  è il vettore di parametri che definisce il mix di distribuzioni,  
 $p_i(\mathbf{x}|\boldsymbol{\theta}_i)$  è una multinormale di parametri  $\boldsymbol{\theta}_i$

# Funzione obiettivo

La stima dei parametri avviene secondo il criterio generale del maximum likelihood (massima verosimiglianza).

Il likelihood  $L$  dei parametri  $\Theta$ , dati i pattern  $x$ , corrisponde alla probabilità di aver ottenuto i pattern  $x$ , dati i parametri  $\Theta$ , (inversione):

$$\mathcal{L}(\Theta | \mathcal{X}) = p(\mathcal{X} | \Theta)$$

# Funzione obiettivo

Per semplicità, al posto della likelihood, si massimizza il suo logaritmo. Considerando i pattern indipendenti, la probabilità di avere ottenuto i pattern del training set è il prodotto delle probabilità delle singole generazioni:

$$\log p(\mathcal{X}|\boldsymbol{\theta}) = \log \prod_{j=1 \dots n} p(\mathbf{x}_j|\boldsymbol{\theta}) = \sum_{j=1 \dots n} \log \left( \sum_{i=1 \dots s} \alpha_i \cdot p_i(\mathbf{x}_j|\boldsymbol{\theta}_i) \right)$$

Purtroppo questa massimizzazione è molto difficile a causa della sommatoria dentro al logaritmo.

Per eliminare la sommatoria, ci servirebbe sapere, per ogni pattern  $x_j$  del training set da quale componente  $p_i$  della mixture è stato generato.

# EM: soluzione al problema di ottimizzazione della funzione obiettivo

$$\log p(\mathcal{X}|\boldsymbol{\Theta}) = \log \prod_{j=1 \dots n} p(\mathbf{x}_j|\boldsymbol{\Theta}) = \sum_{j=1 \dots n} \log \left( \sum_{i=1 \dots s} \alpha_i \cdot p_i(\mathbf{x}_j|\boldsymbol{\theta}_i) \right)$$

dove  $p(\mathbf{x}|\boldsymbol{\Theta}) = \sum_{i=1 \dots s} \alpha_i \cdot p_i(\mathbf{x}|\boldsymbol{\theta}_i)$   $\boldsymbol{\theta}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$

A tal fine si utilizza EM, che è un approccio iterativo nato per il calcolo del maximum likelihood nel caso in cui i dati a disposizione  $X = x_1, x_2 \dots x_n$  siano incompleti per la mancanza di alcuni valori  $y$ .

Ogni pattern completo  $z_j = x_j, y_j$ , con  $j=1 \dots n$  è costituito da due parti di cui solo la prima è nota.

Nel caso di interesse (derivare i parametri di una gaussian mixture) i dati sono completi ma le potenzialità di EM sono utilizzate per considerare le componenti più ignote che hanno generato i singoli pattern come valori  $y$  non noti di dati a disposizione completi.

## EM: caso generale

Obiettivo di EM è dunque la massimizzazione del log-likelihood dei dati completi o log-likelihood-completo così definito:

$$\log \mathcal{L}(\Theta | \mathcal{Z}) = \log \mathcal{L}(\Theta | X, y) = p(X, y | \Theta)$$

Per risolvere vengono eseguiti iterativamente (fino a convergenza) due passi, Expectation e Maximization.

Expectation: viene calcolato il valore atteso  $E(\cdot)$  del log-likelihood-completo, dato il training set  $X$  e una stima dei parametri  $\Theta_g$  ottenuti all'iterazione precedente:

$$Q(\Theta | \Theta^g) = E(\log p(X, y | \Theta) | X, \Theta^g)$$

## EM: caso generale

Il valore atteso (media) è calcolato rispetto alla variabile  $y$ , governata dalla distribuzione  $f$ :

$$E(\log p(\mathcal{X}, y|\Theta) | \mathcal{X}, \Theta^g) = \int_{\mathbf{y} \in \Psi} \log p(\mathcal{X}, y|\Theta) \cdot f(\mathbf{y}|\mathcal{X}, \Theta^g) d\mathbf{y}$$

Maximization: viene calcolato il valore dei parametri che massimizzano il valore atteso calcolato al passo di Expectation:

$$\Theta^{g+1} = \underset{\Theta}{\operatorname{argmax}} Q(\Theta|\Theta^g)$$

## EM: caso specifico

Si assume l'esistenza di una componente nascosta  $y$  (per ogni pattern) che indica quale delle  $s_j$  distribuzioni gaussiane ha generato il pattern  $x_j$ .

Il log-likelihood-completo assume ora la forma:

$$\log \mathcal{L}(\Theta | \mathcal{X}, \mathbf{y}) = \sum_{j=1 \dots n} \log \left( \alpha_{y_j} \cdot p_{y_j} (\mathbf{x}_j | \Theta_{y_j}) \right)$$

sebbene gli  $y$  non siano noti, una loro stima (o meglio una stima  $j$  della loro distribuzione) può essere derivata (teorema di Bayes) dai parametri  $\Theta^g$  disponibili all'iterazione ( $g$ ) corrente.

# EM: caso specifico

Per un singolo  $y_j$  (e corrispondente  $x_j$ ):

$$P(y_j = k | \mathbf{x}_j, \Theta^g) = P(k | \mathbf{x}_j, \Theta^g) = \frac{\alpha_k^g \cdot p_k(\mathbf{x}_j | \Theta_k^g)}{\sum_{i=1 \dots s} \alpha_i^g \cdot p_i(\mathbf{x}_j | \Theta_i^g)}$$

Per un generico vettore  $\mathbf{y}$  di osservazioni:

$$P(\mathbf{y} | \mathcal{X}, \Theta^g) = \prod_{j=1 \dots n} P(y_j | \mathbf{x}_j, \Theta^g)$$

# EM: caso specifico

Il valore atteso Q delle slide precedenti può essere scritto come:

$$\begin{aligned} Q(\Theta|\Theta^g) &= \sum_{\mathbf{y} \in \Psi} \log \mathcal{L}(\Theta|\mathcal{X}, \mathbf{y}) \cdot P(\mathbf{y}|\mathcal{X}, \Theta^g) = \\ &= \sum_{\mathbf{y} \in \Psi} \left( \sum_{j=1 \dots n} \log \left( \alpha_{y_j} \cdot p_{y_j}(\mathbf{x}_j | \Theta_{y_j}) \right) \cdot \prod_{j=1 \dots n} P(y_j | \mathbf{x}_j, \Theta^g) \right) \end{aligned}$$

## EM: formule incrementali

A seguito di alcuni passaggi (non banali) durante i quali l'ottimizzazione è eseguita eguagliando a zero le derivate parziali rispetto ai parametri incogniti, si ottengono le seguenti formule per l'aggiornamento incrementale dei parametri:

$$P(C_k | \mathbf{x}_j, \Theta^g) = \frac{\alpha_k^g \cdot p(\mathbf{x}_j | \Theta_k^g)}{\sum_{i=1 \dots s} \alpha_i^g \cdot p(\mathbf{x}_j | \Theta_i^g)} \quad \text{eq (1)}$$

$$\alpha_k^{g+1} = \frac{1}{n} \sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g) \quad \text{eq (2)}$$

$$\boldsymbol{\mu}_k^{g+1} = \frac{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g) \cdot \mathbf{x}_j}{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g)} \quad \text{eq 3)$$

$$\boldsymbol{\Sigma}_k^{g+1} = \frac{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g) (\mathbf{x}_j - \boldsymbol{\mu}_k^{g+1}) (\mathbf{x}_j - \boldsymbol{\mu}_k^{g+1})^t}{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g)} \quad \text{eq (4)}$$

# EM: formule incrementali

Intuitivamente, trattando le componenti della mixture come cluster:

Eq (1), stessa delle slide precedenti: indica la probabilità di appartenenza di  $x_j$  al cluster  $C_k$ . Calcolata sfruttando l'inversione di Bayes  $p(\cdot | \Theta_k)$  è una multinormale con parametri  $\Theta_k$

Eq (2). Calcola il peso del cluster C in base alla somma delle k appartenenze a C di tutti i pattern del training set

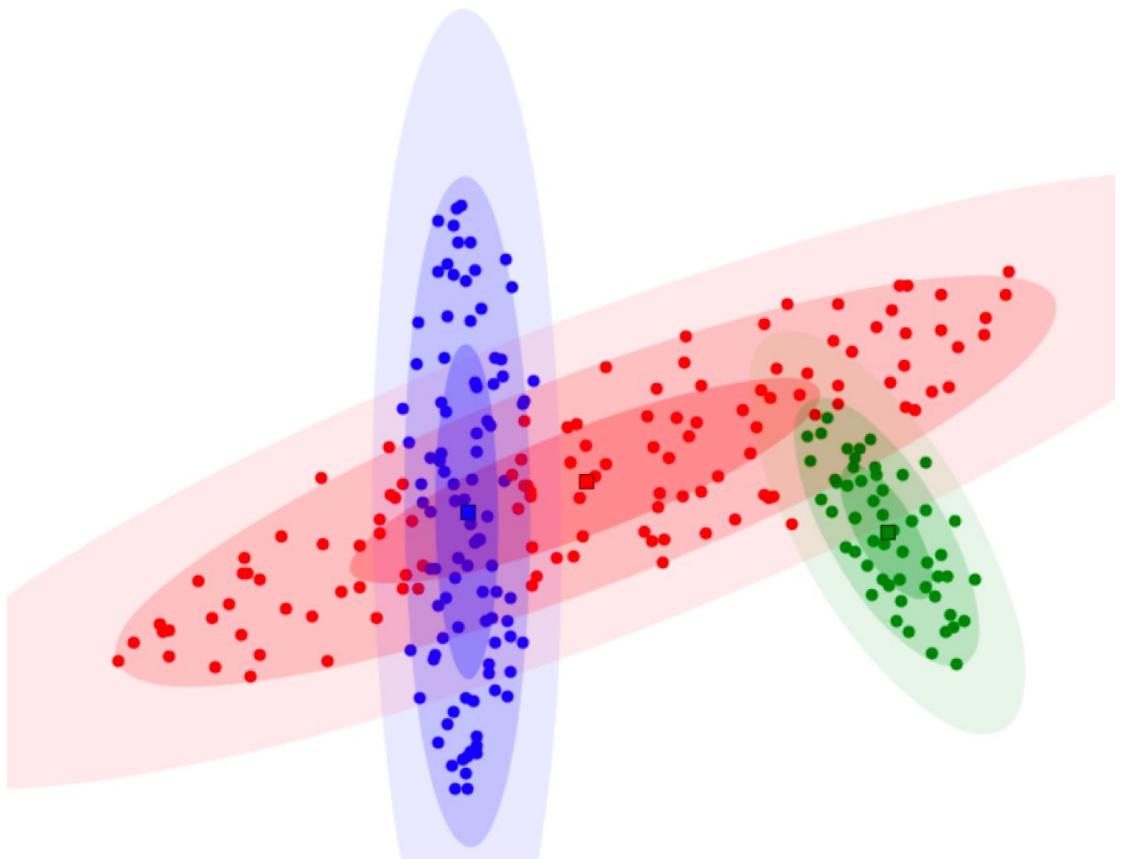
Eq (3) e (4). Calcolano media e matrice di covarianza del cluster C pesando i pattern rispetto al loro grado di appartenenza a C

$$P(C_k | \mathbf{x}_j, \Theta^g) = \frac{\alpha_k^g \cdot p(\mathbf{x}_j | \Theta_k^g)}{\sum_{i=1 \dots n} \alpha_i^g \cdot p(\mathbf{x}_j | \Theta_i^g)} \quad \text{eq (1)}$$

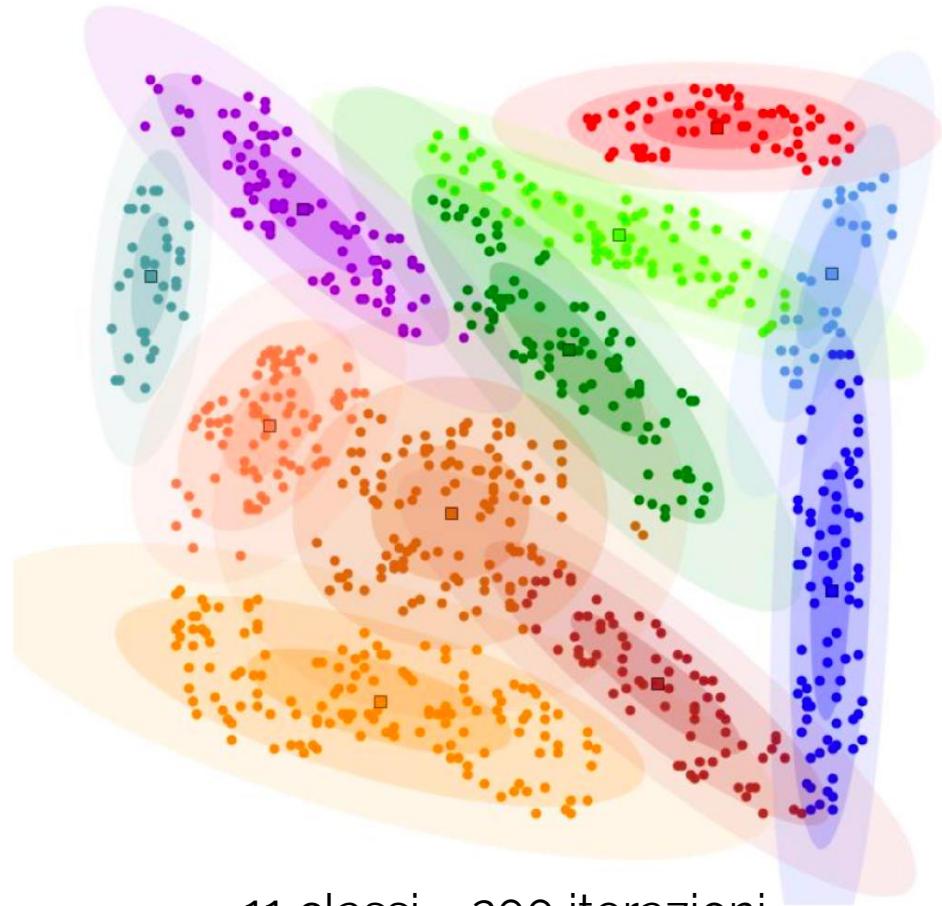
$$\alpha_k^{g+1} = \frac{1}{n} \sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g) \quad \text{eq (2)}$$

$$\boldsymbol{\mu}_k^{g+1} = \frac{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g) \cdot \mathbf{x}_j}{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g)} \quad \text{eq 3)$$

$$\boldsymbol{\Sigma}_k^{g+1} = \frac{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g) (\mathbf{x}_j - \boldsymbol{\mu}_k^{g+1}) (\mathbf{x}_j - \boldsymbol{\mu}_k^{g+1})^t}{\sum_{j=1 \dots n} P(C_k | \mathbf{x}_j, \Theta_k^g)} \quad \text{eq (4)}$$



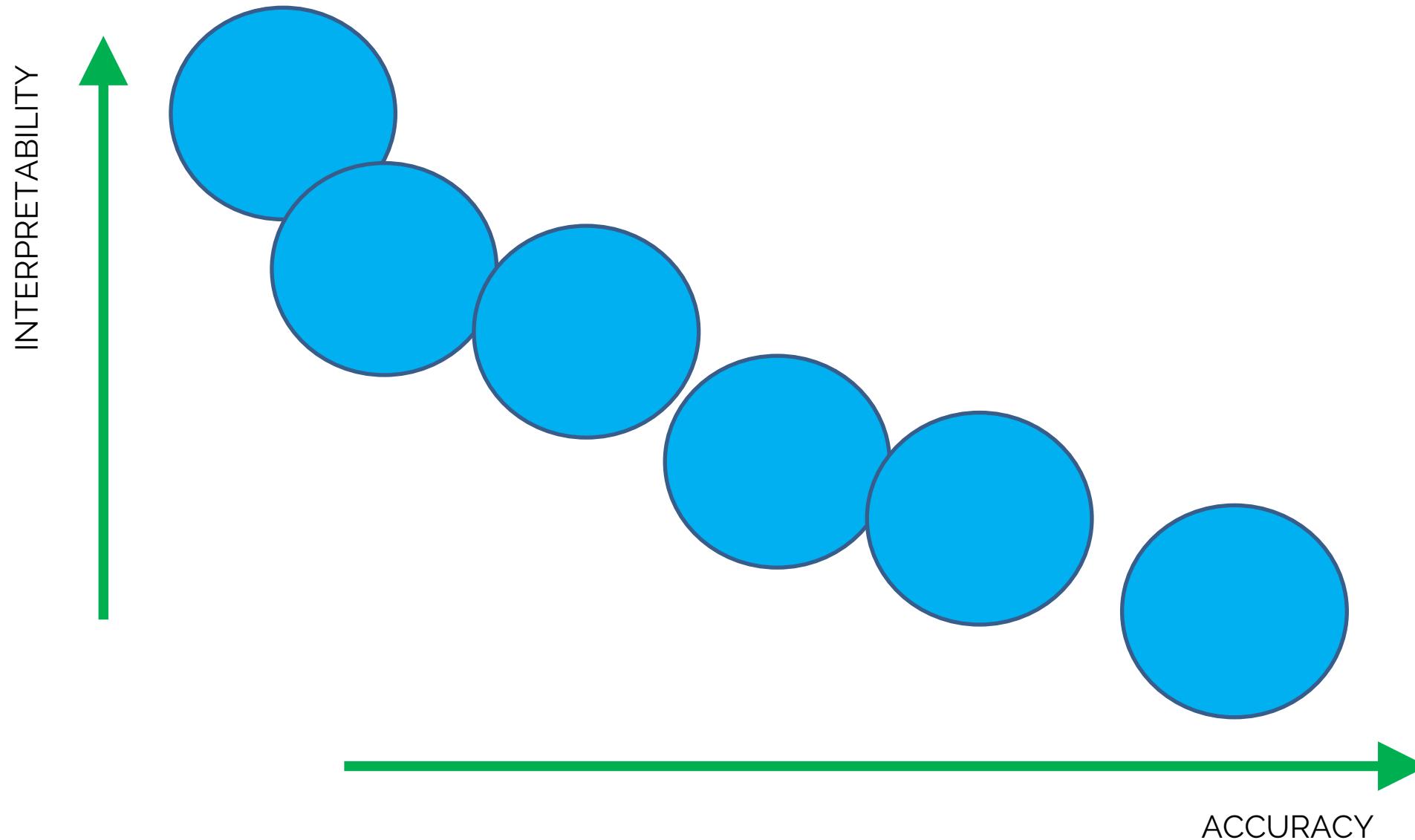
3 classi – 40 iterazioni



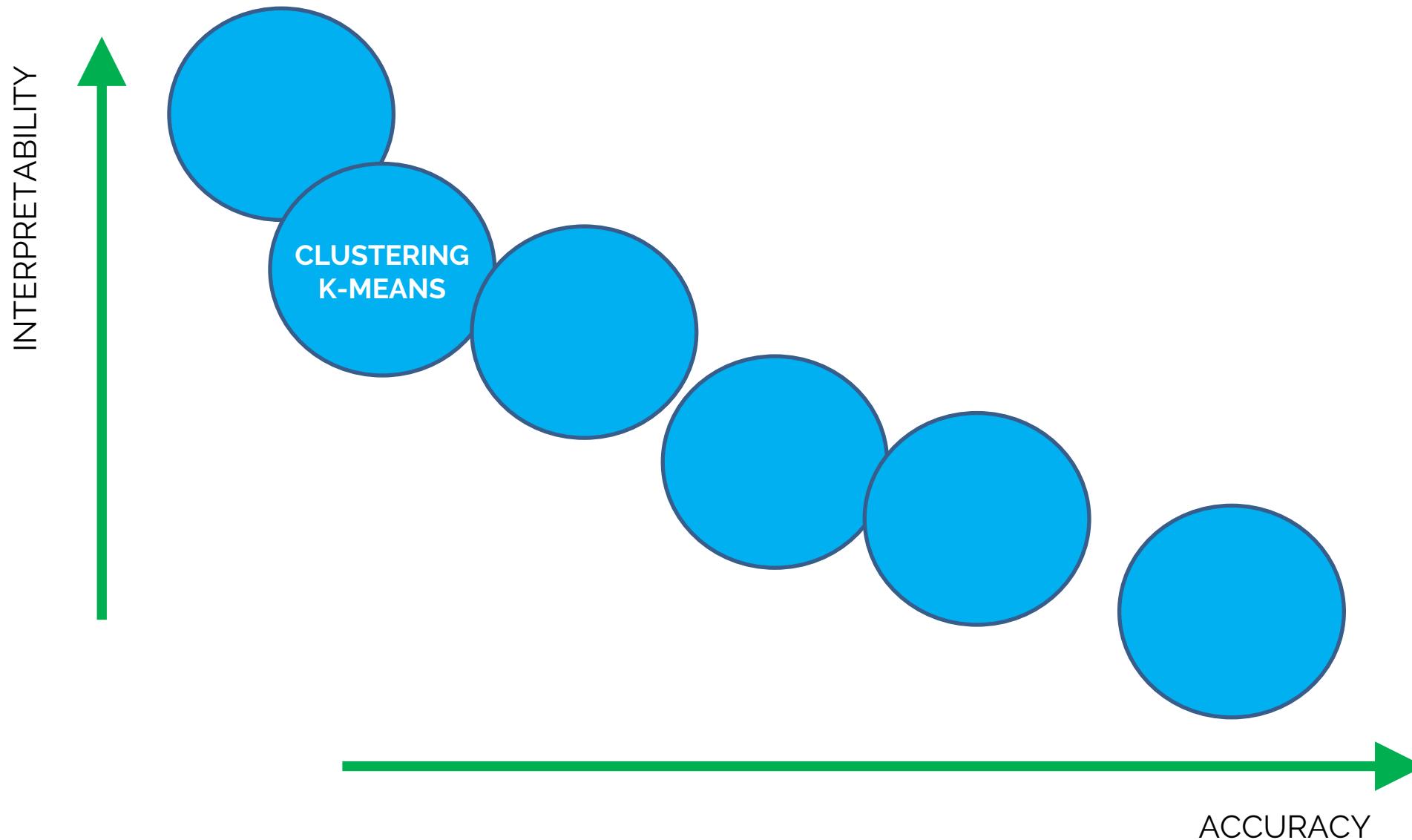
11 classi – 200 iterazioni

EM è in grado di identificare nuvole di punti con forme ellissoidali e capsule, a differenza di K-means e fuzzy K-means

# Interpretability-Accuracy TRADEOFF



# Interpretability-Accuracy TRADEOFF



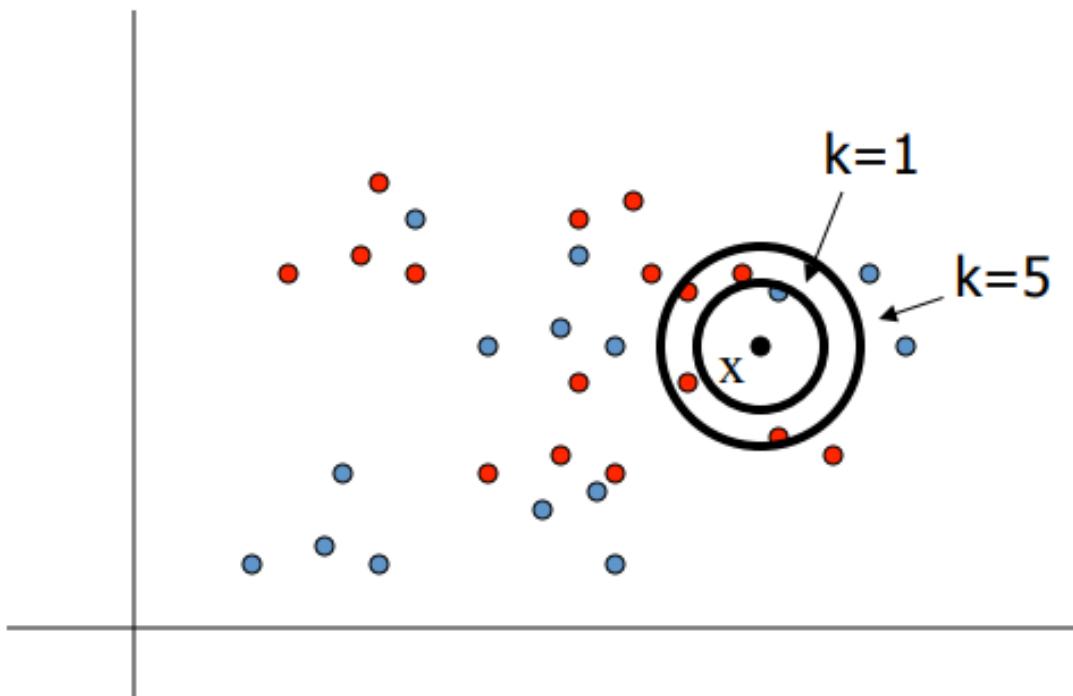
# CLASSIFICATION

## Nearest Neighbors

WITH SOME MATERIALS FROM DAVID SONTAG, VIBHAV GOGATE, CARLOS  
GUESTRIN, MEHRYAR MOHRI, LUKE ZETTLEMOYER, DHILIP SUBRAMANIAN

# Nearest Neighbors

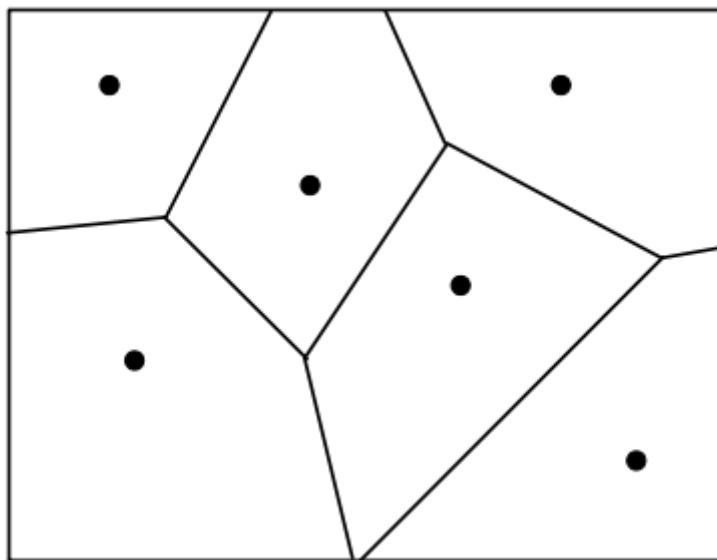
To classify a new input vector  $x$ , examine the  $k$ -closest training data points to  $x$  and assign the object to the most frequently occurring class



Common values for  $k = 3, 5$

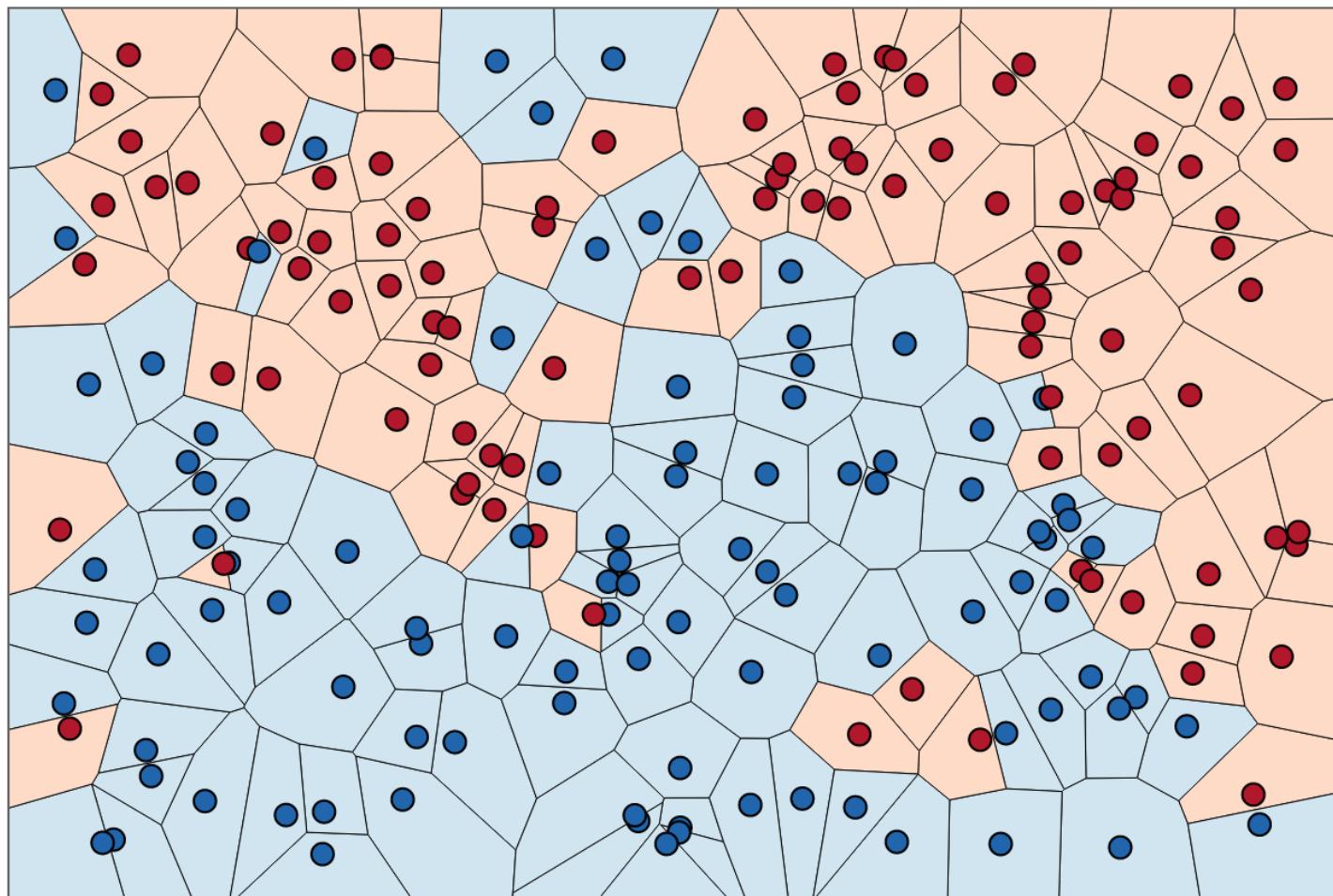
# Nearest Neighbors

The nearest neighbor algorithm does not explicitly compute decision boundaries.



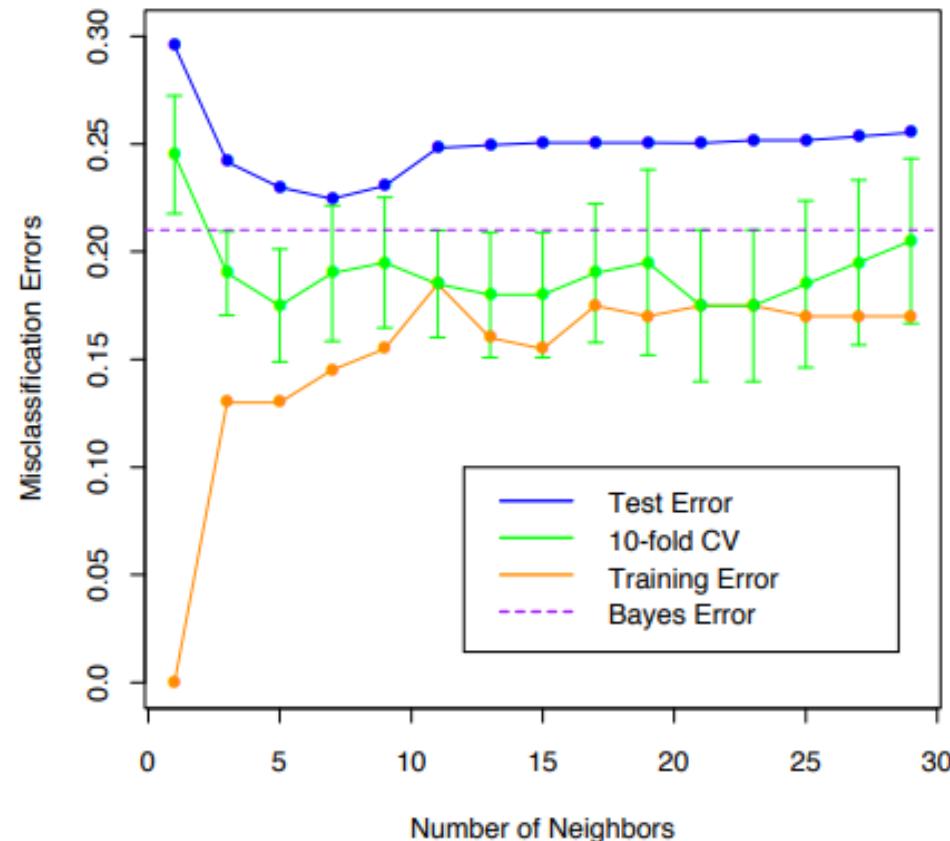
The decision boundaries form a subset of the Voronoi diagram for the training data.

# Nearest Neighbors

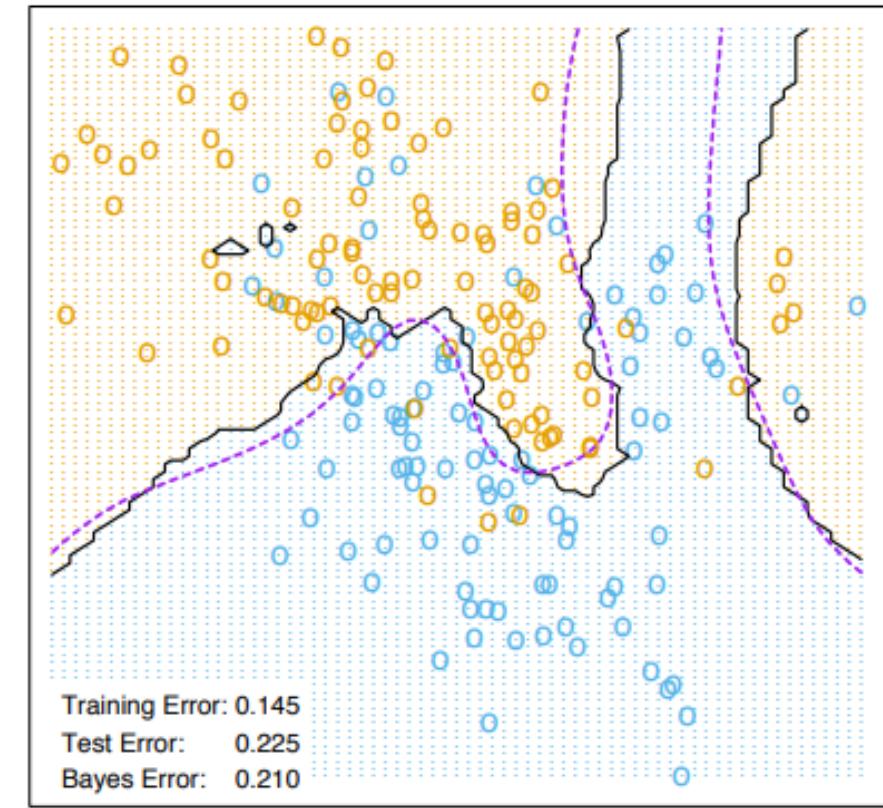


The more examples that are stored,  
the more complex the decision boundaries can become

# Nearest Neighbors



7-Nearest Neighbors



HASTIE & TIBSHIRANI

# Nearest Neighbors

## CONSIDERATIONS

- Instance map to points in  $R^n$
- Less than 20 attributes per instance
- Lots of training data

### Advantages

- Training is very fast
- Learn complex target functions
- Do not lose information

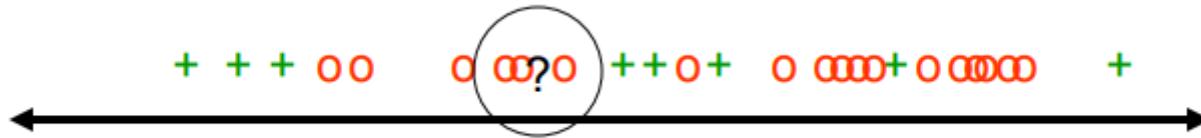
### Disadvantages

- Slow at classification
- Easily fooled by irrelevant attributes  
(see next slides)

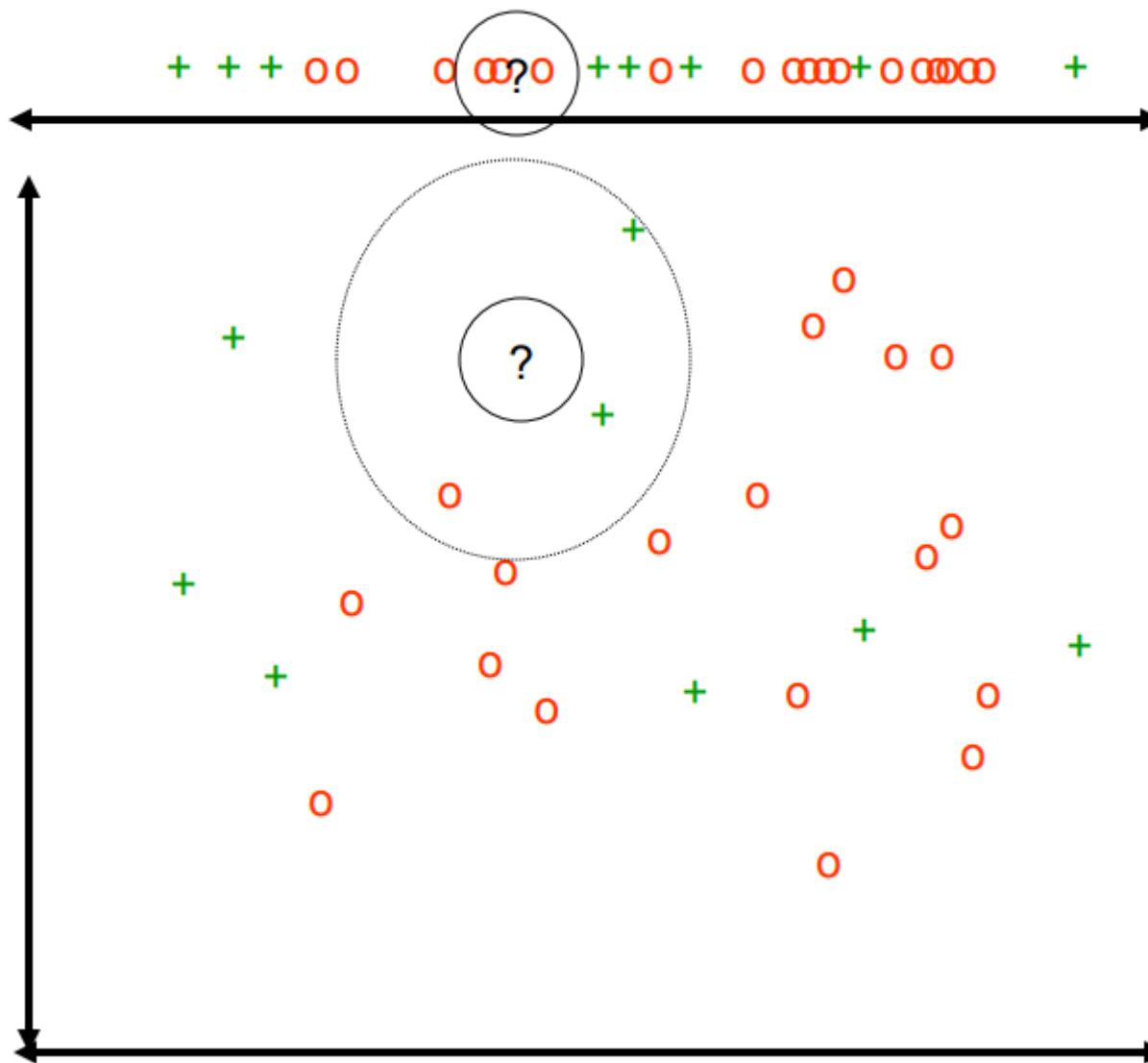
## ISSUES

- Distance measure – Most common: Euclidean
- Choosing  $k$  – Increasing  $k$  reduces variance, increases bias
- For high dimensional space, the nearest neighbor may be very distant
- Memory-based technique. Must make a pass through the data for each classification (think about big data!)

# Nearest Neighbors



# Nearest Neighbors



# Nearest Neighbors

Notation: object with p measurements

$$\mathbf{x}^i = (x_1^i, x_2^i, \dots, x_p^i)$$

Most common distance metric is the Euclidean Distance:

$$d_E(\mathbf{x}^i, \mathbf{x}^j) = \left( \sum_{k=1}^p (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

The use of the Euclidean Distance is appropriate when different measurements are commensurate; each is variable measured in the same units. If the measurements are not comparable, this may affect training/classification results.

# Nearest Neighbors

## STANDARDIZATION

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation.

The estimate for the standard deviation of  $x_k$  is

$$\hat{\sigma}_k = \left( \frac{1}{n} \sum_{i=1}^n (x_k^i - \bar{x}_k)^2 \right)^{\frac{1}{2}}$$

where  $\bar{x}_k$  is the sample mean

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^n x_k^i$$

# Nearest Neighbors

## WEIGHTED EUCLIDEAN DISTANCE

If we have a-priori knowledge of a relative importance of each variable, we can weight them by a factor

$$d_{WE}(i, j) = \left( \sum_{k=1}^p w_k (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

# Nearest Neighbors

Nearest neighbor breaks down in high dimensional spaces because the "neighborhood" becomes very large.

- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5 nearest neighbor algorithm.
- Suppose our query point is at the origin.
  - 1D
  - we must go at a distance of  $5/5000 = 0.001$  on average to capture the 5 nearest neighbors
- 2D
  - we must go  $\sqrt{0.001}$  to get a square that contains 0.001 of the volume
- Generic D
  - we must go  $(0.001)^{1/D}$

... CURSE OF DIMENSIONALITY

# Nearest Neighbors

No structured method to find the best value for “K”.

- Trial and error approach, assuming that training data is unknown.
- Choosing smaller values for K can be noisy and will have a higher influence on the result.
- Larger values of K will have smoother decision boundaries which mean lower variance but increased bias.
- Larger values of K are computationally expensive.
- Through cross-validation (K can be treated as a hyperparameter...)
- GENERAL PRACTICE: choosing the value of k is  $k = \sqrt{N}$  where N stands for the number of samples in your training dataset.
- ODD!

# Nearest Neighbors

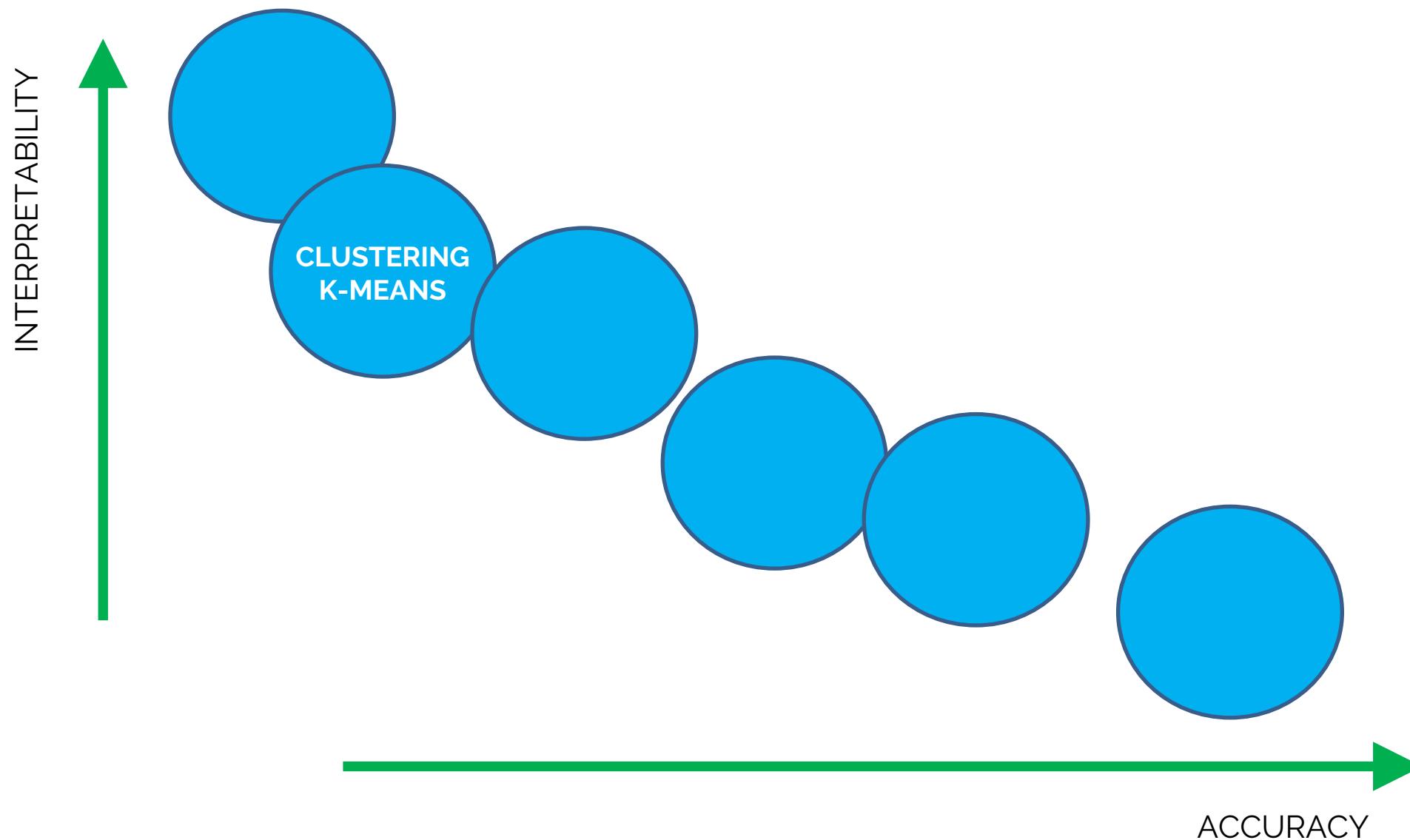
Advantages:

- Simple to implement
- Flexible to feature/distance choices
- Naturally handles multi-class cases
- Can do well in practice with enough representative data (comparable to more complex models)
- No optimization or training required

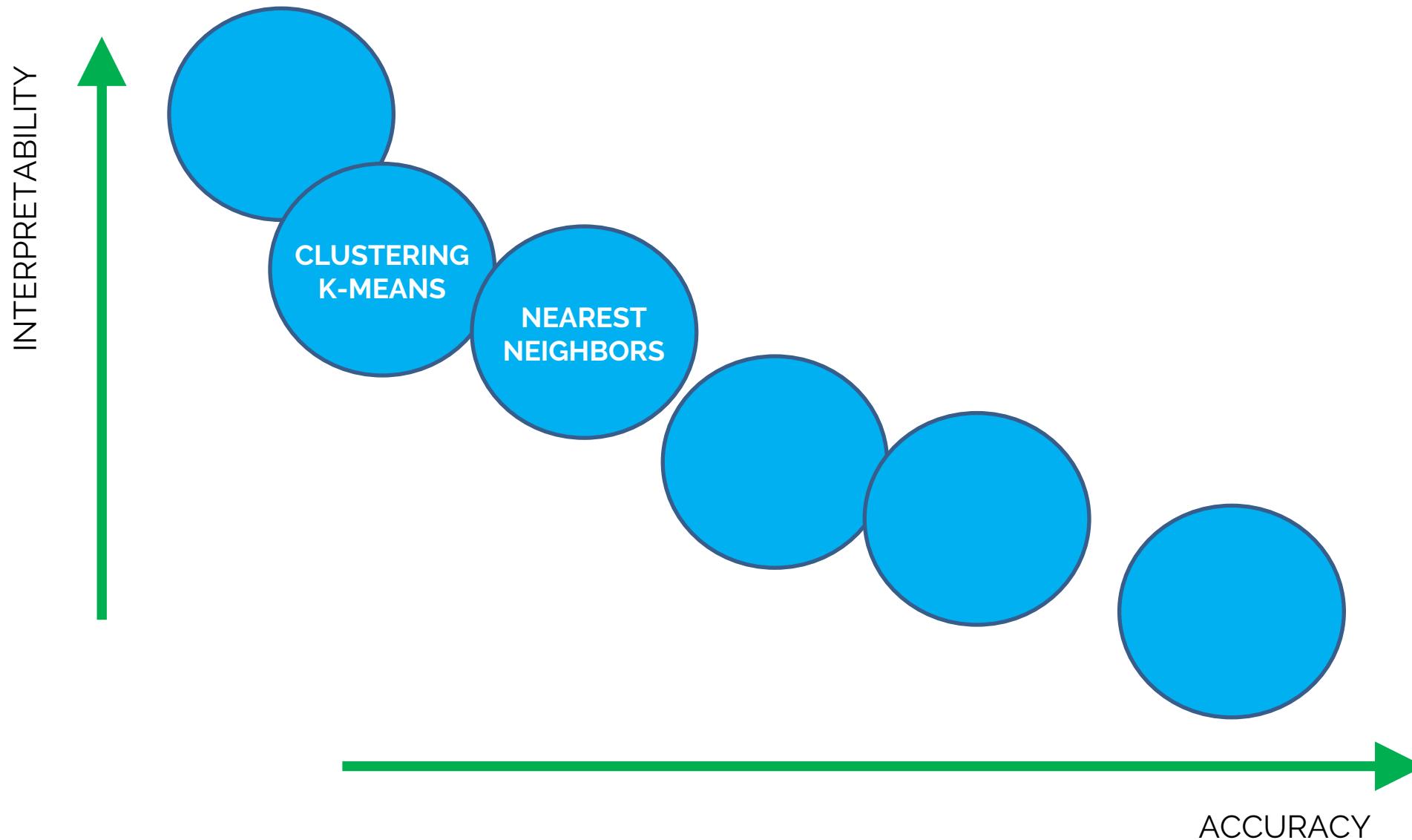
Drawbacks:

- Need to determine the value of parameter  $K$  (number of nearest neighbors)
- Computation cost is quite high because we need to compute the distance of each query instance to all training samples.
- Storage of data
- Choose a meaningful distance function.

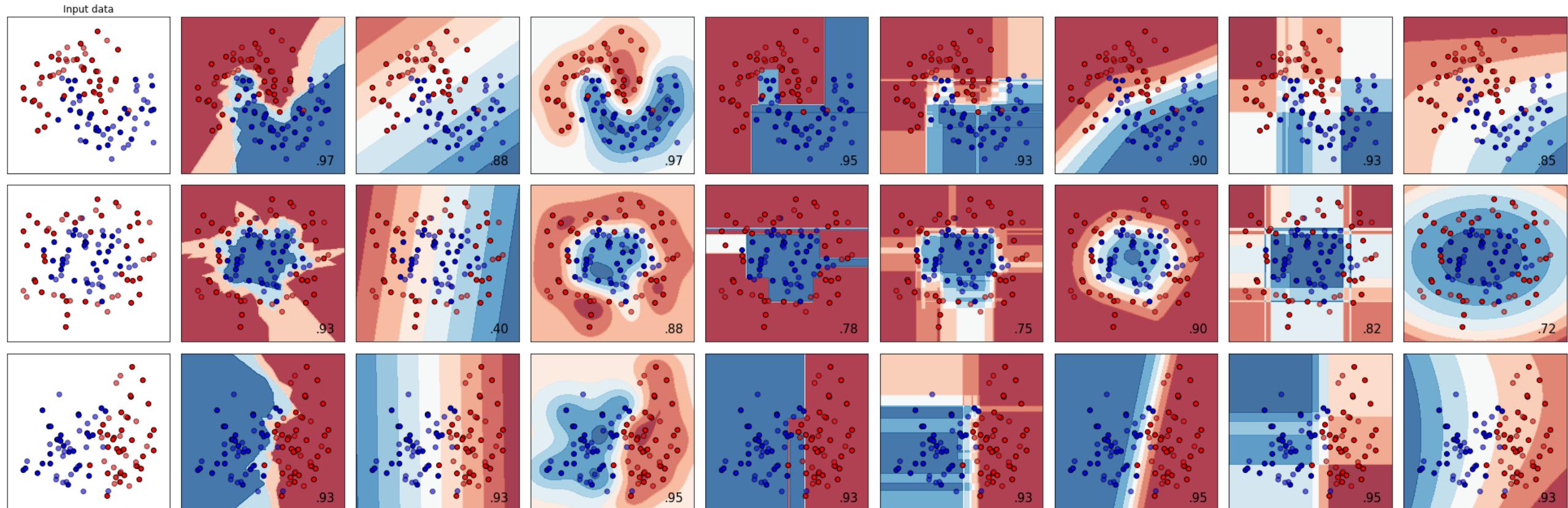
# Interpretability-Accuracy TRADEOFF



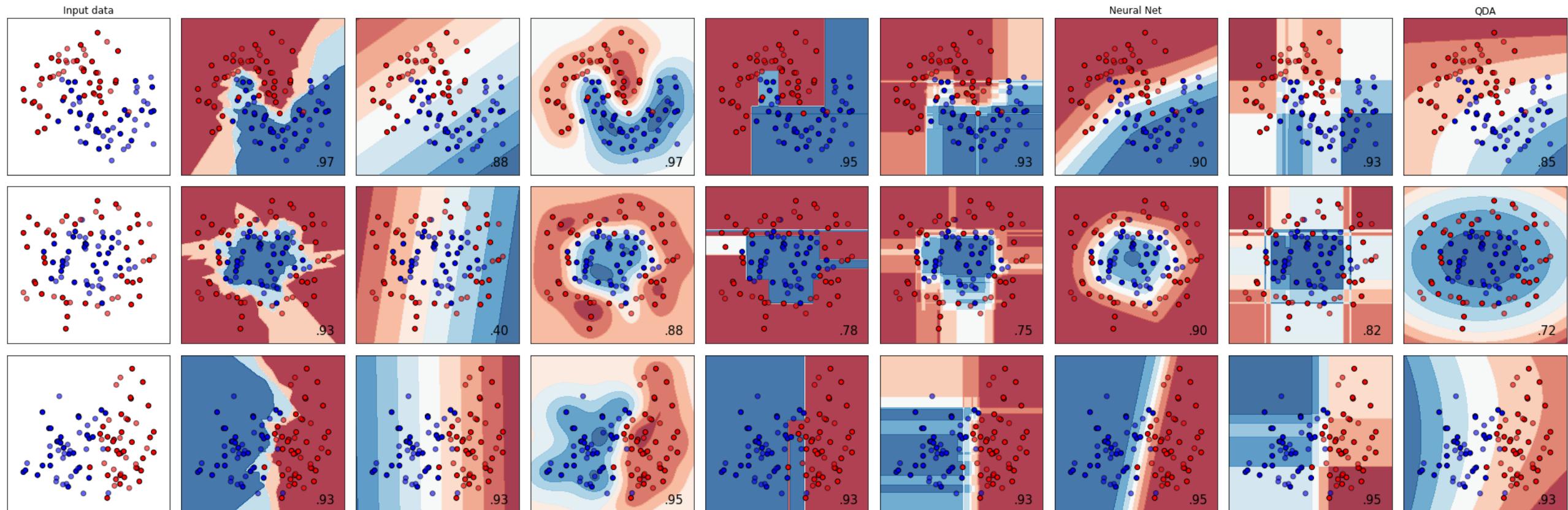
# Interpretability-Accuracy TRADEOFF



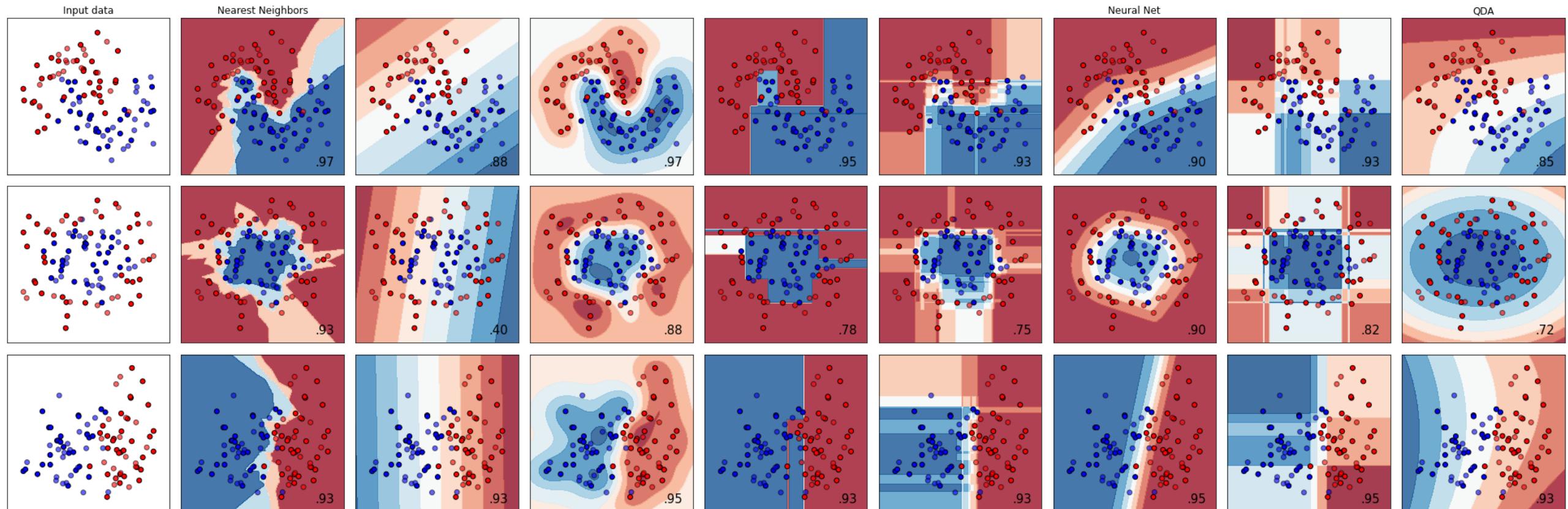
# Which is which | Decision Function



# Which is which | Decision Function



# Which is which | Decision Function



# CLASSIFICATION

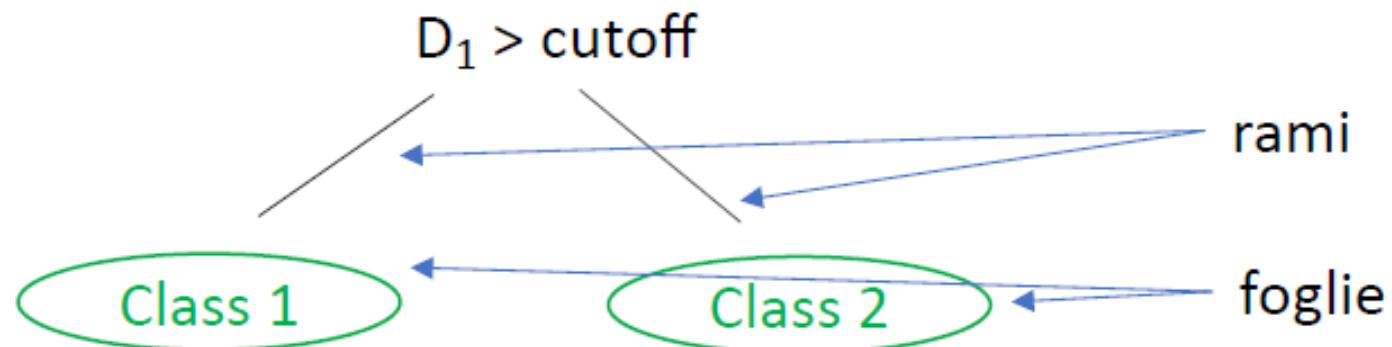
## DECISION TREE

# Decision Tree

Un decision tree o classification tree è un albero binario in cui ogni nodo divide i pattern sulla base di un criterio su una singola feature (o dimensione).

In queste strutture ad albero, i nodi a foglia rappresentano le etichette delle classi e i rami rappresentano l'insieme delle caratteristiche che portano a quelle classificazioni.

Algoritmi ad albero in cui la variabile target può assumere un insieme discreto di valori sono chiamati alberi di classificazione, quelli in cui la variabile target può assumere valori continui (in genere numeri reali) sono chiamati alberi di regressione.

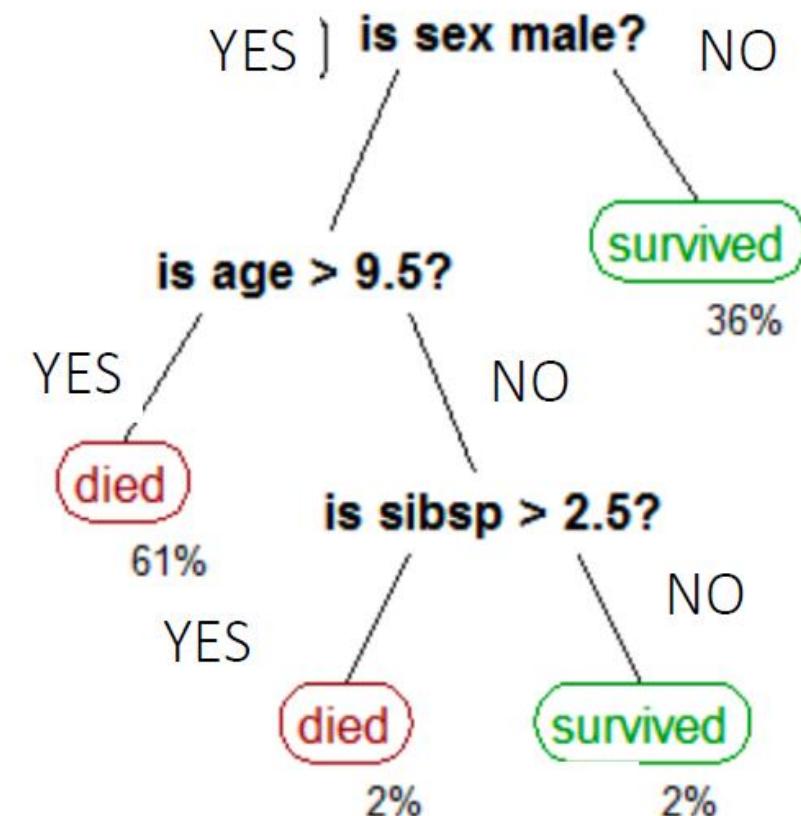


# Decision Tree

Sopravvivenza passeggeri del Titanic

Classi: died e survived

I numeri sotto le foglie indicano la percentuale di campioni nella foglia (classe)



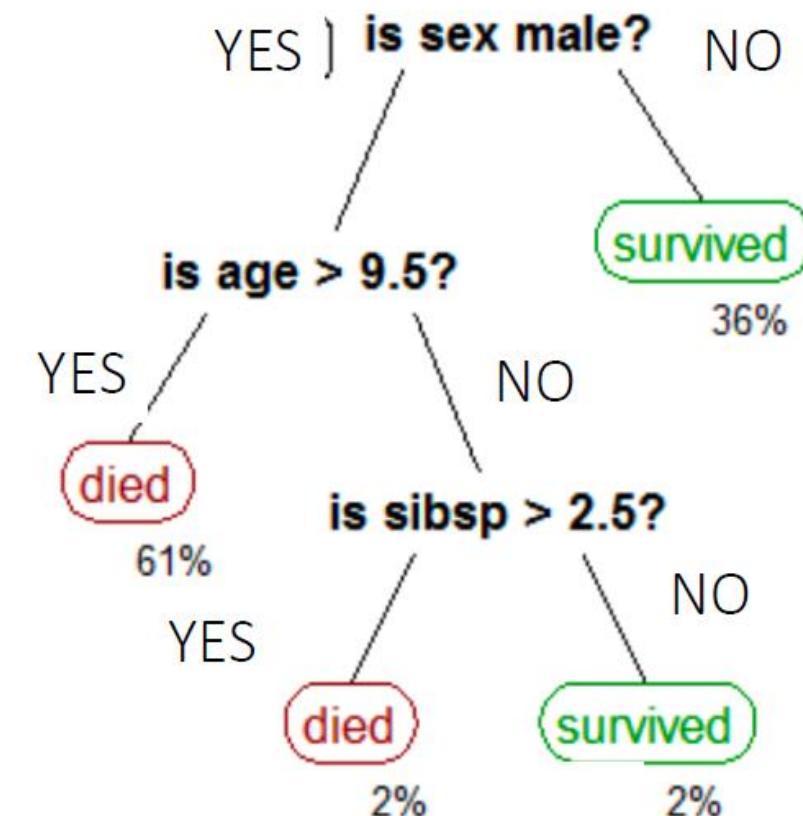
*Le possibilità di sopravvivenza erano buone se il soggetto era  
(i) una femmina o (ii) un maschio di età inferiore ai 9,5 anni  
con meno di 2,5 fratelli*

# Decision Tree

Ogni nodo interno risulta essere una macro-classe costituita dall'unione delle classi associate ai suoi nodi figli.

Il predicato che si associa ad ogni nodo interno (sulla base del quale avviene la ripartizione dei dati) è chiamato condizione di split.

E' utile fissare un minimo numero di campioni richiesti per effettuare lo split.



*Le possibilità di sopravvivenza erano buone se il soggetto era (i) una femmina o (ii) un maschio di età inferiore ai 9,5 anni con meno di 2,5 fratelli*

# Decision Tree | Objective Function

In un buon albero di classificazione, i nodi foglia dovrebbero essere il più possibile puri (ovvero contenere solo dati che appartengono ad una sola classe).

Un parametro che definisce il criterio di impurità è l'indice di Gini\* (impurità di Gini), che può quindi essere considerata come una funzione obiettivo.

Si stima il guadagno informativo come la differenza fra l'impurità del nodo genitore e la somma delle impurità dei nodi figli: minore è l'impurità dei nodi figli, maggiore è il guadagno informativo (decrease in Gini index).

L'indice di Gini raggiunge il suo minimo (zero) quando il nodo appartiene ad una singola categoria. Intuitivamente, l'impurità di Gini può essere considerata come un criterio per minimizzare la probabilità di un'errata classificazione.

$$I_G(i) = 1 - \sum_{j=1}^m f(i, j)^2 \quad \text{f rappresenta la frequenza del valore j nel nodo i.}$$

\*misura della diseguaglianza di una distribuzione, valori bassi indicano una distribuzione abbastanza omogenea (0-1)

# Decision Tree | Objective Function

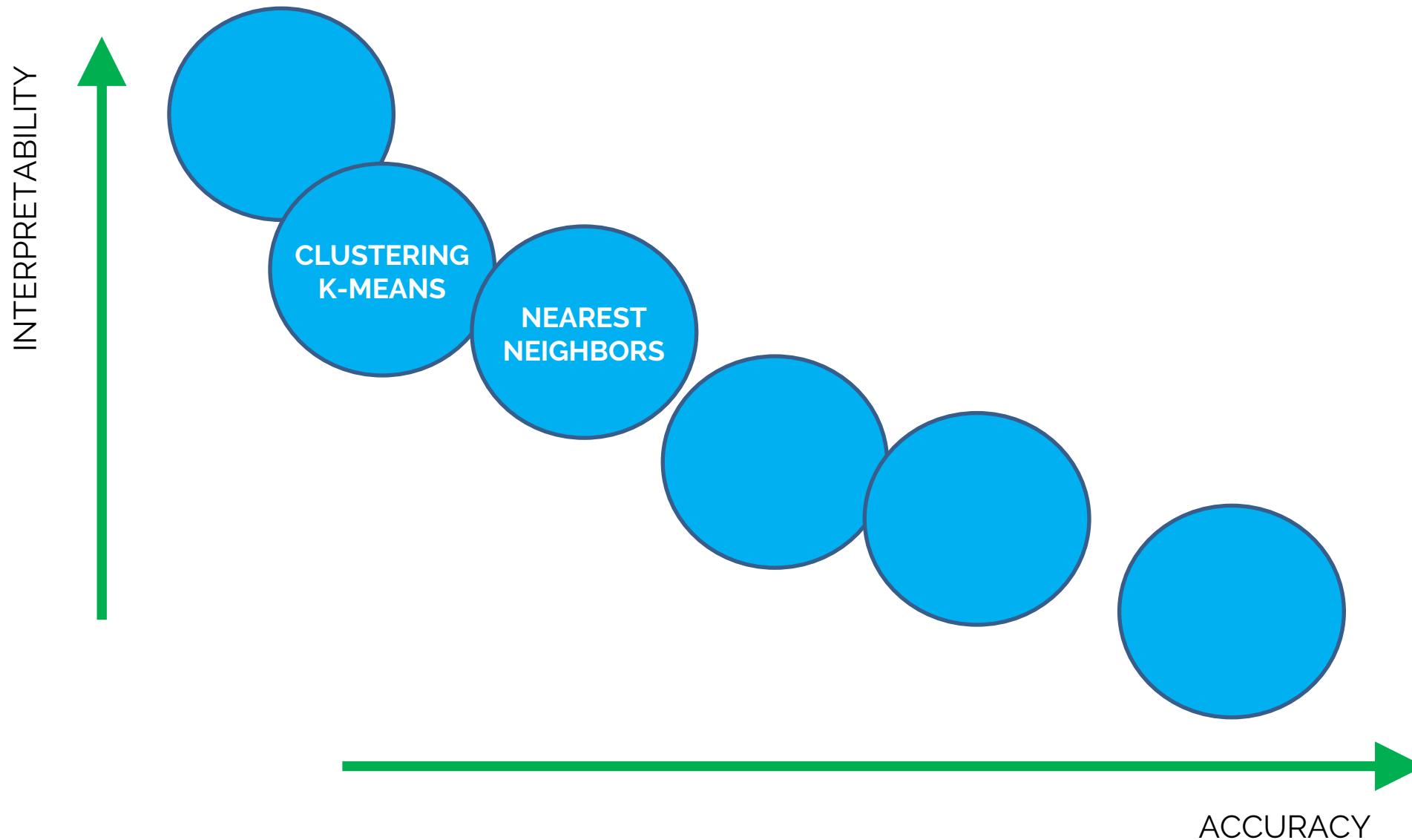
Un altro parametro che definisce il criterio di impurità è l'indice di entropia

$$I_E(i) = - \sum_{j=1}^m f(i, j) \log f(i, j)$$

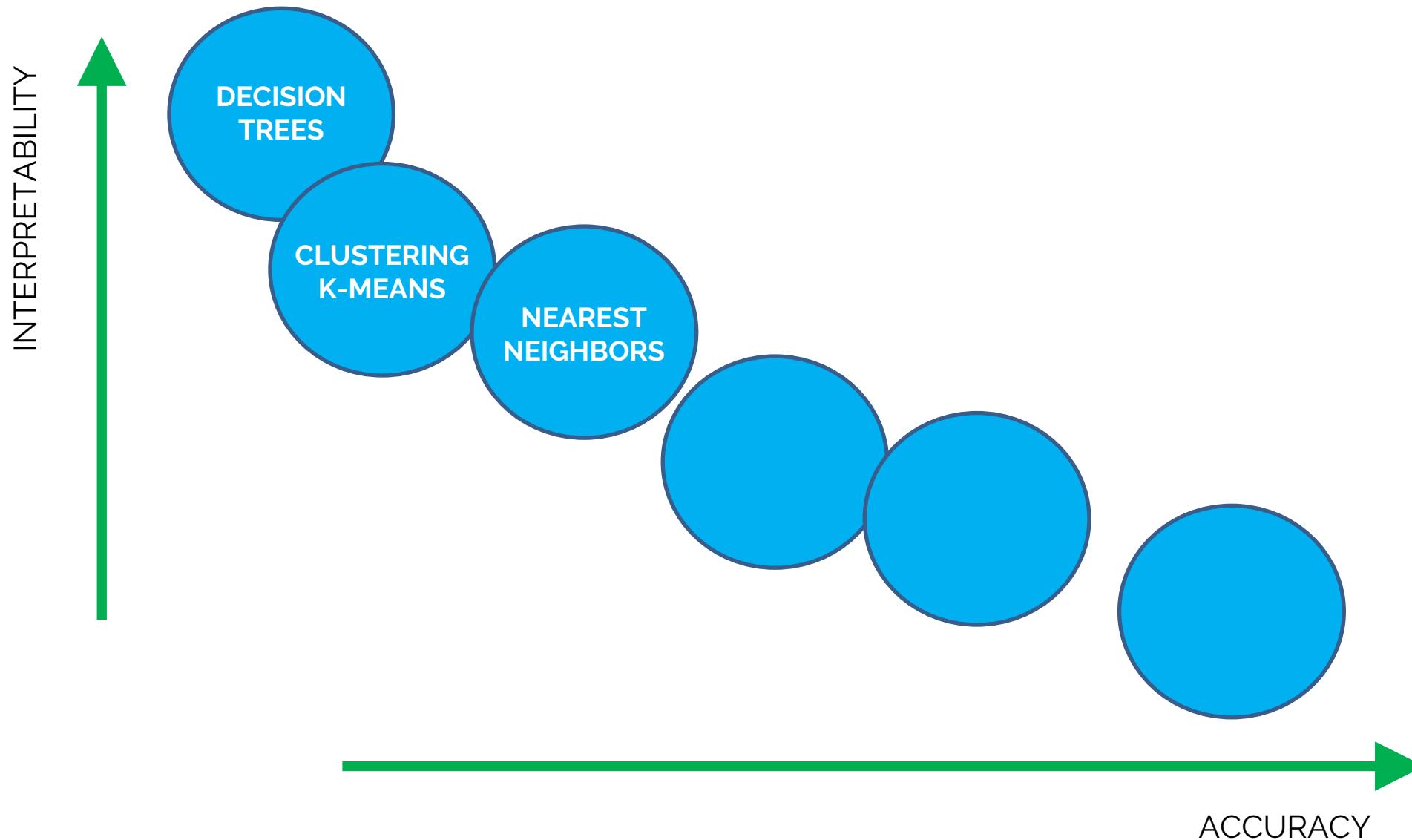
L'indice di Gini e l'indice di entropia sono i parametri che vengono usualmente utilizzati per guidare la costruzione dell'albero.

Il tasso di errore di classificazione che viene utilizzato per effettuare una ottimizzazione dell'albero nota come processo di arresto (halting) o di pruning («potatura» dei nodi superflui), al fine di determinarne la profondità massima dell'albero (max depth). Il crescere della profondità di un albero (ovvero della sua dimensione) non influisce direttamente sulla bontà del modello. Infatti, una crescita eccessiva della dimensione dell'albero potrebbe portare solo ad aumento sproporzionato della complessità computazionale rispetto ai benefici riguardanti l'accuratezza delle previsioni/classificazioni.

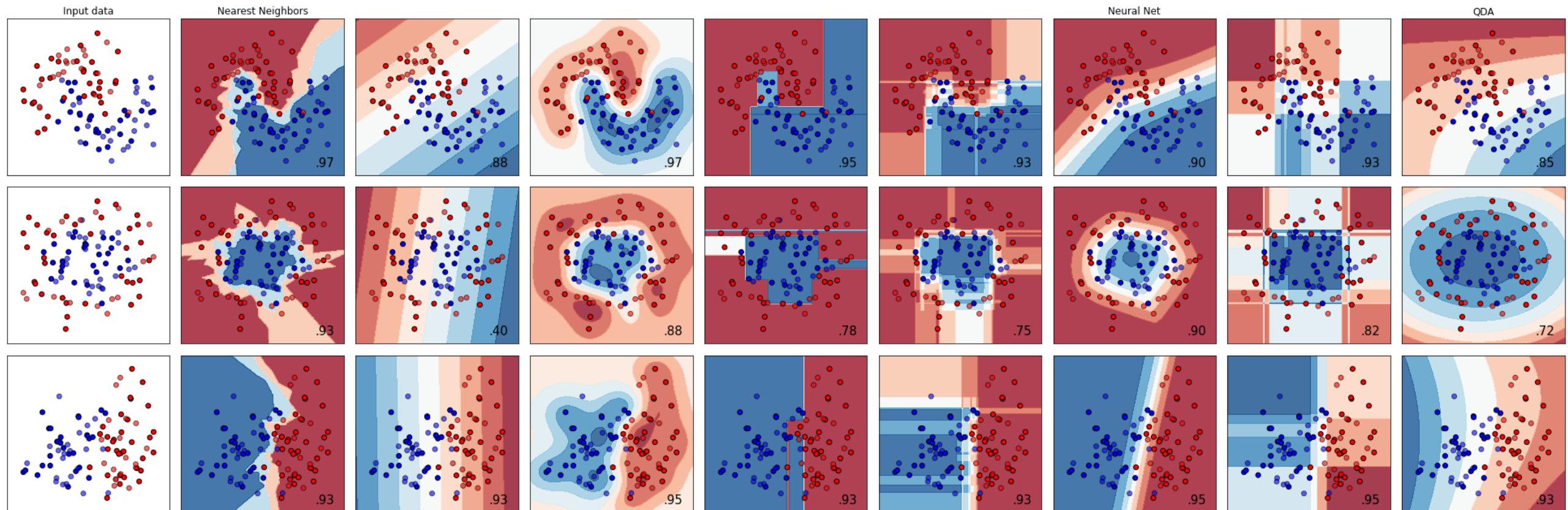
# Interpretability-Accuracy TRADEOFF



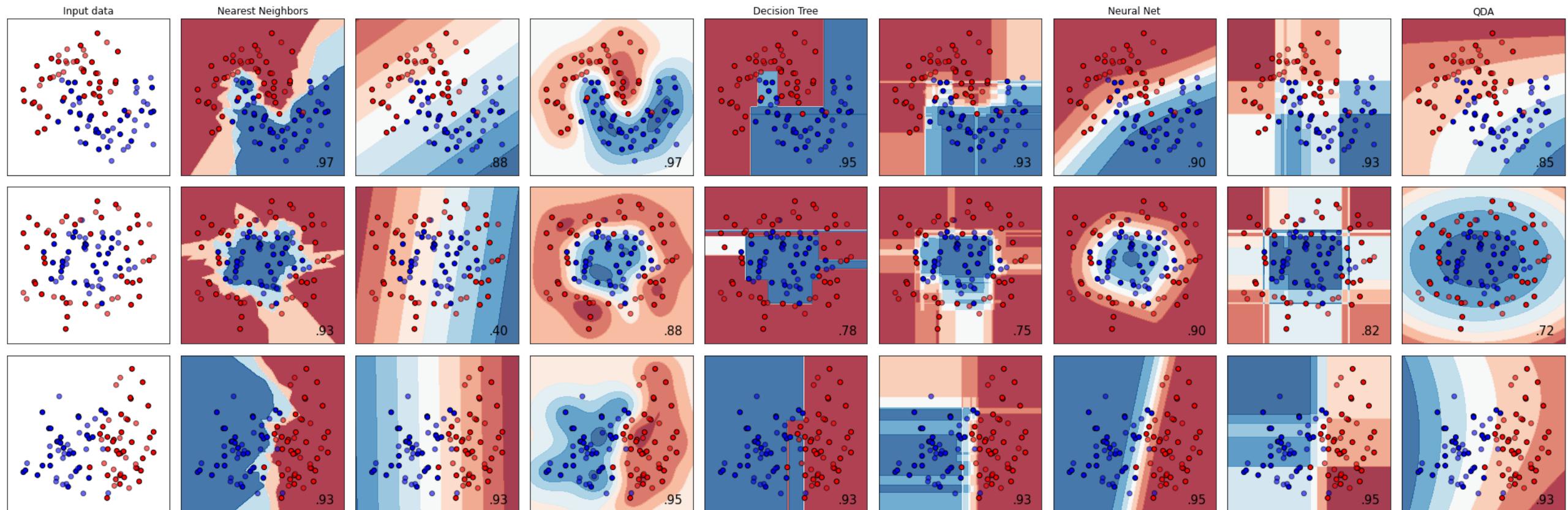
# Interpretability-Accuracy TRADEOFF



# Which is which | Decision Function



# Which is which | Decision Function



# CLASSIFICATION

## SUPPORT VECTOR MACHINE

# Support Vector Machine

Machine Learning, 20, 273–297 (1995)

© 1995 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

## Support-Vector Networks

CORINNA CORTES

VLADIMIR VAPNIK

AT&T Bell Labs., Holmdel, NJ 07733, USA

[corinna@neural.att.com](mailto:corinna@neural.att.com)

[vlad@neural.att.com](mailto:vlad@neural.att.com)

**Editor:** Lorenza Saitta

**Abstract.** The *support-vector network* is a new learning machine for two-group classification problems. The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensure high generalization ability of the learning machine. The idea behind the support-vector network was previously implemented for the restricted case where the training data can be separated without errors. We here extend this result to non-separable training data.

SVM is a binary classifier which aims at generating a predictive model for the discrimination of new samples.

# Support Vector Machine

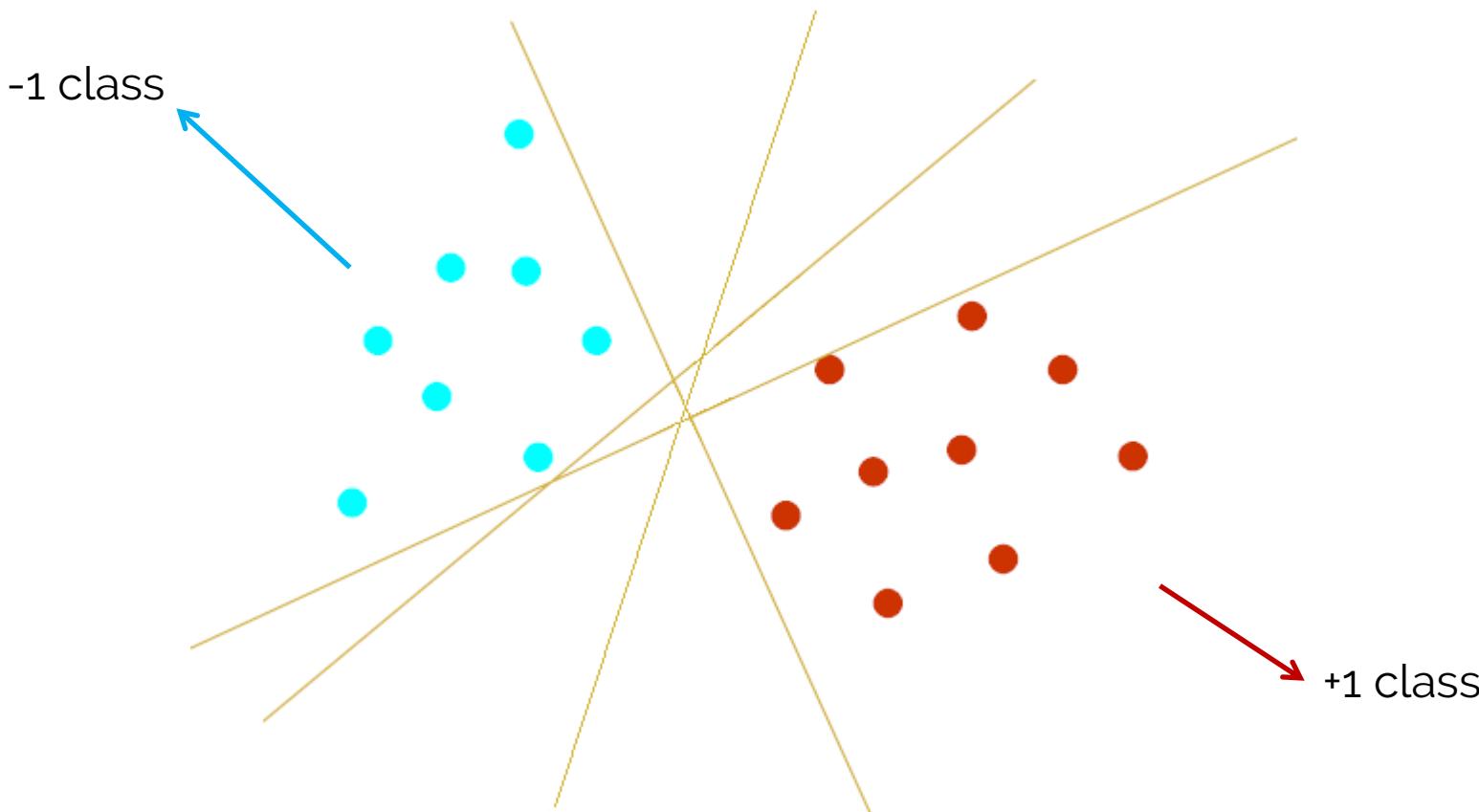
Teoria introdotta da Vapnik (statistical learning theory) nel 1965.  
Perfezionata più tardi dallo stesso Vapnik e altri nel 1995.

Vapnik suggerisce di determinare superfici decisionali tra le classi (classification boundaries) per separarle.

SVM nasce come classificatore binario (2 classi), con gradi diversi di complessità:

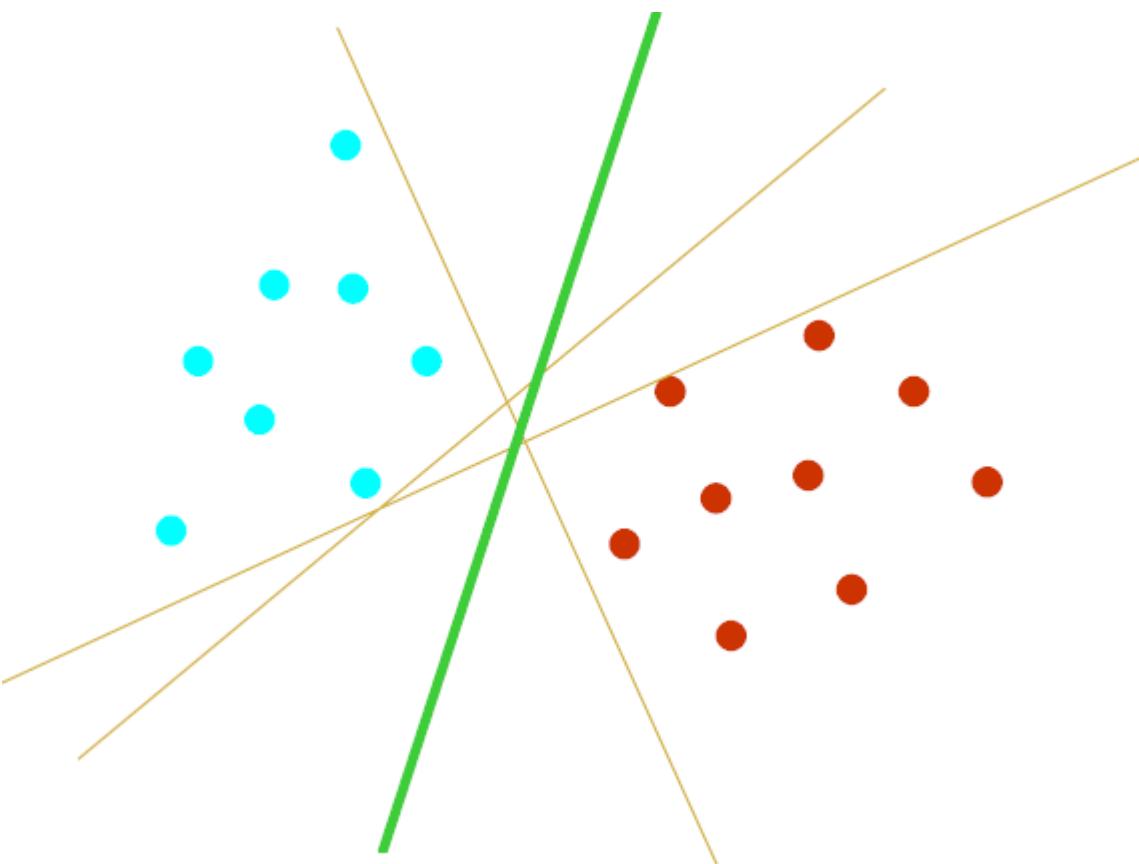
- SVM lineare (i.e., la superficie di separazione è un iperpiano) e i pattern del training set sono linearmente separabili (i.e., esiste per ipotesi almeno un iperpiano in grado di separarli).
- SVM lineare e pattern non linearmente separabili. Errori di classificazione non esistendo alcun iperpiano in grado di separare i pattern.
- SVM non lineare (i.e., superficie di separazione complessa) senza ipotesi sulla separabilità dei pattern.
- Estensione SVM a multi-classe.

# Support Vector Machine



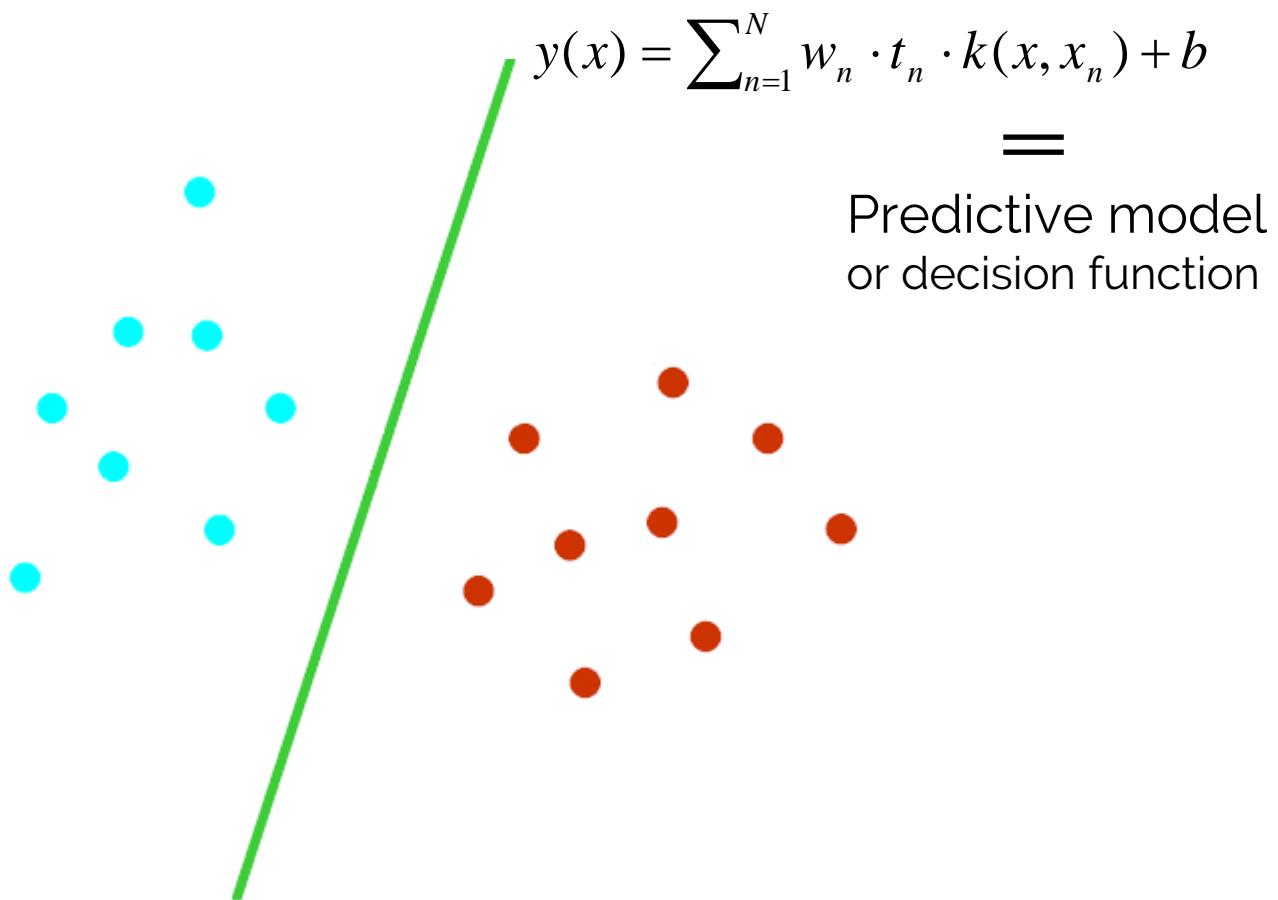
Let us suppose to have a set of training data consisting of a vector of  $N$  samples belonging to two classes and the corresponding vector of class labels (e.g. -1 and +1 for control and patient class, respectively)

# Support Vector Machine



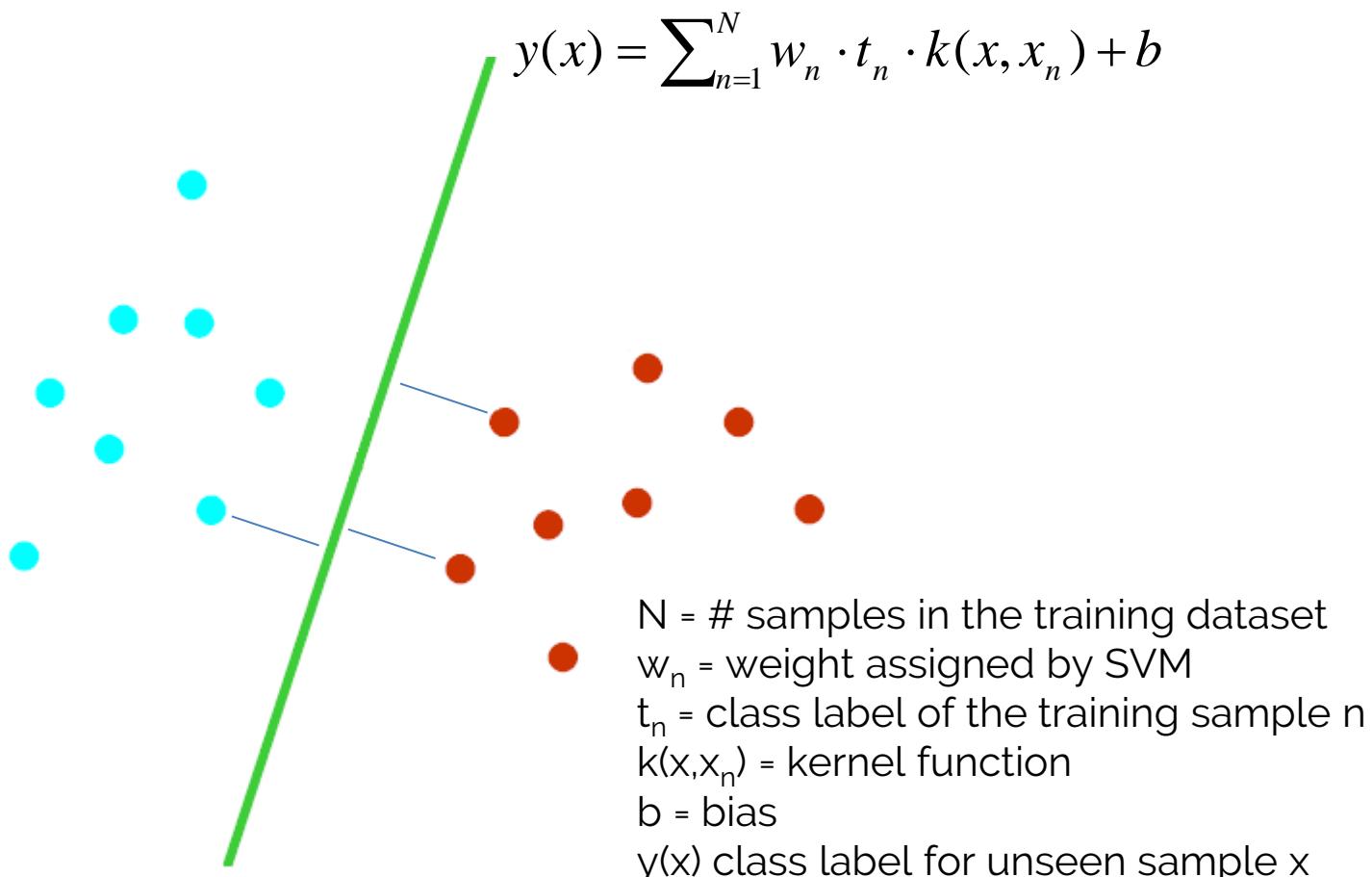
SVM generates a hyper-plane able to discriminate between the two classes of training dataset

# Support Vector Machine



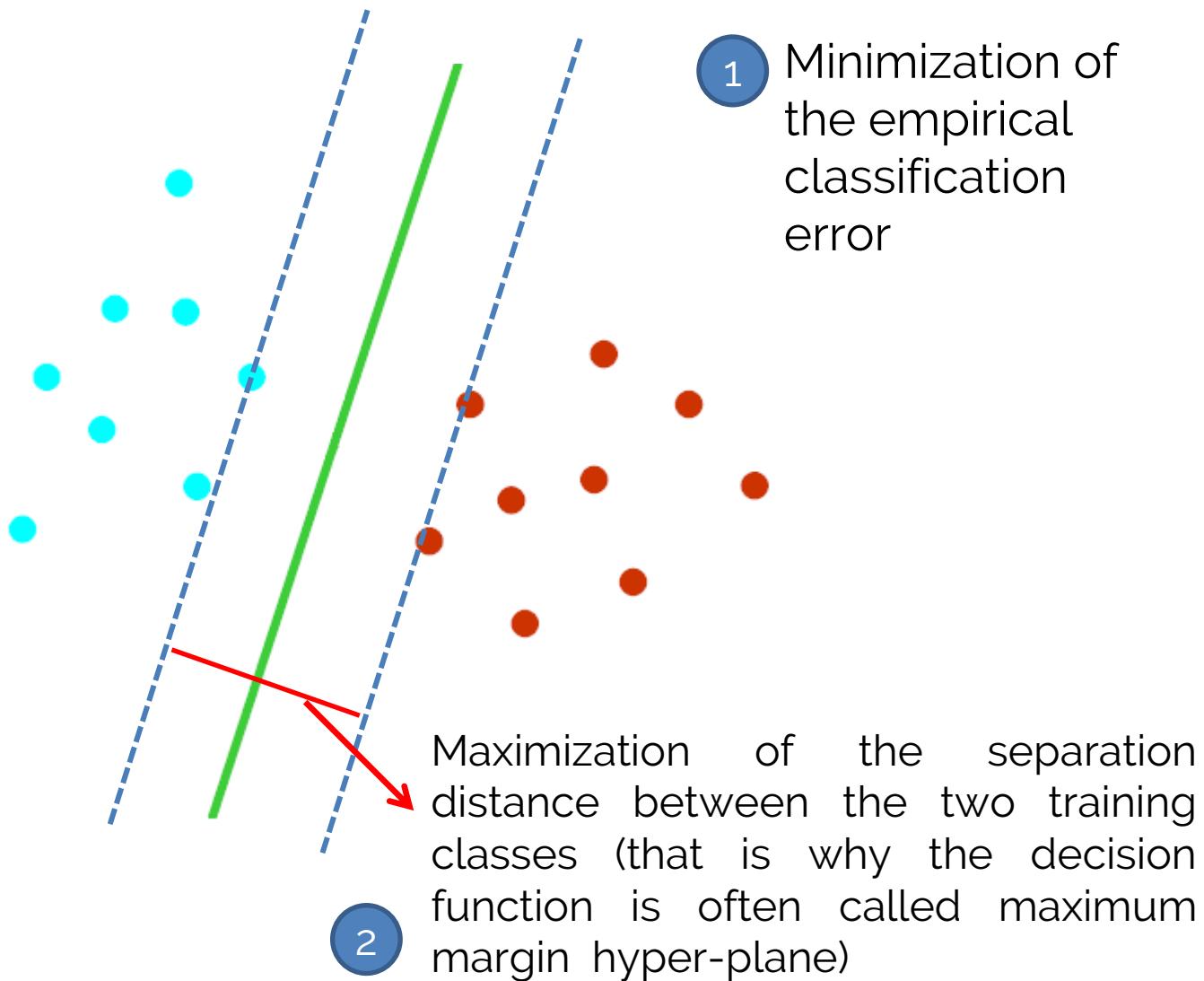
SVM generates a hyper-plane able to discriminate between the two classes of training dataset

# Support Vector Machine

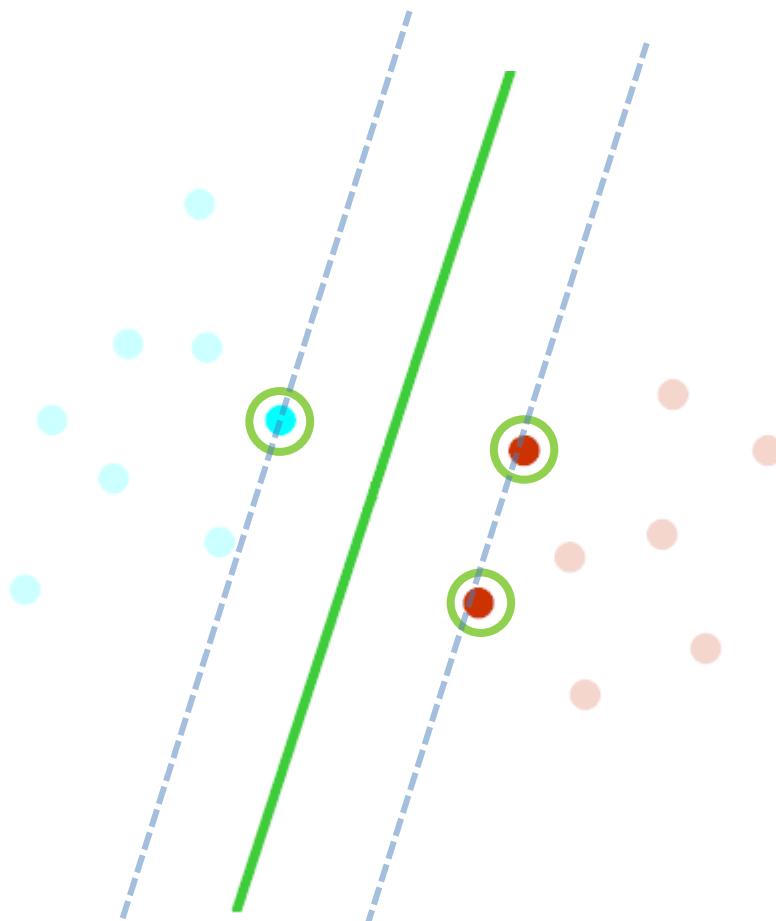


The weight assigned by SVM during the training phase to each training sample reflects its importance for classes discrimination

# Support Vector Machine

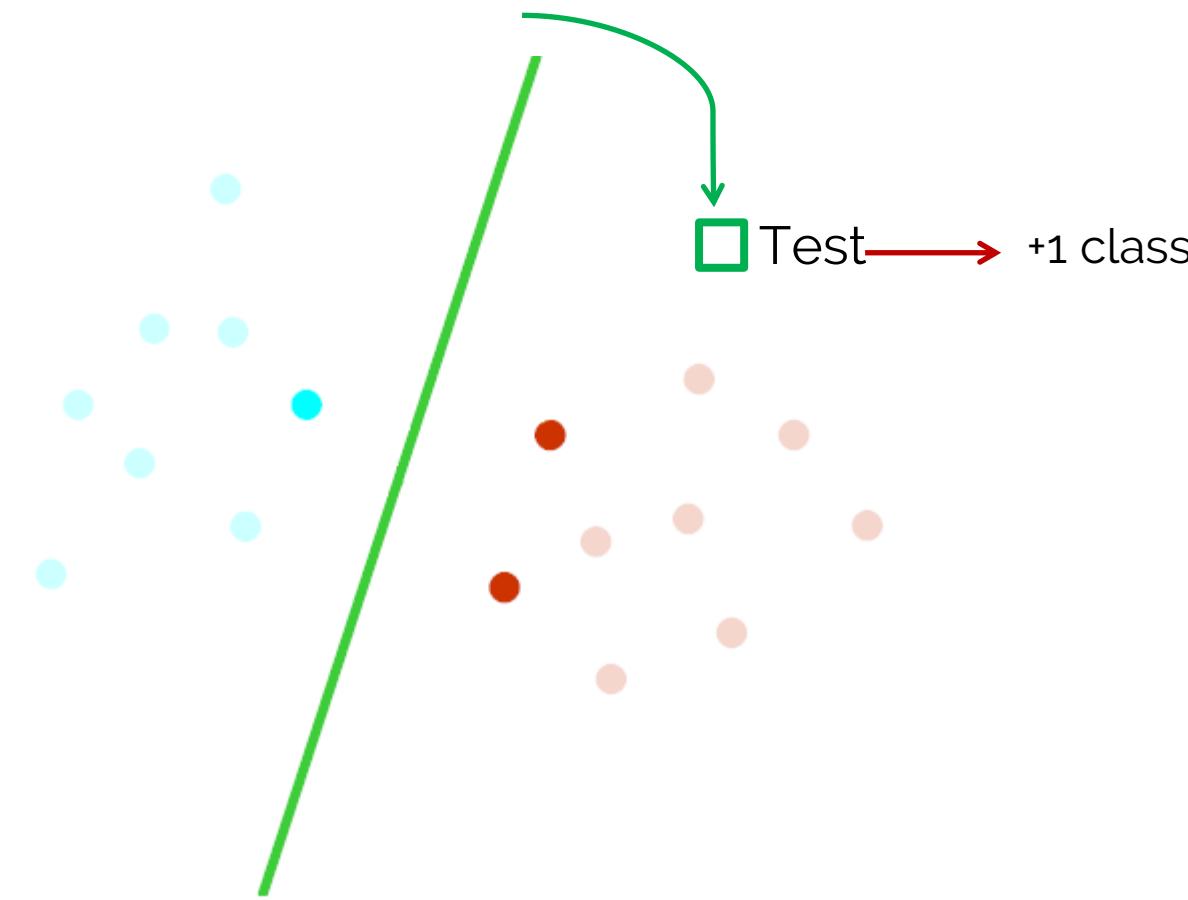


# Support Vector Machine



Samples of the two classes which lie on the margin of the hyper-plane are called support vector, and they are the only samples for which the assigned weight  $w$  is non-zero.

# Support Vector Machine

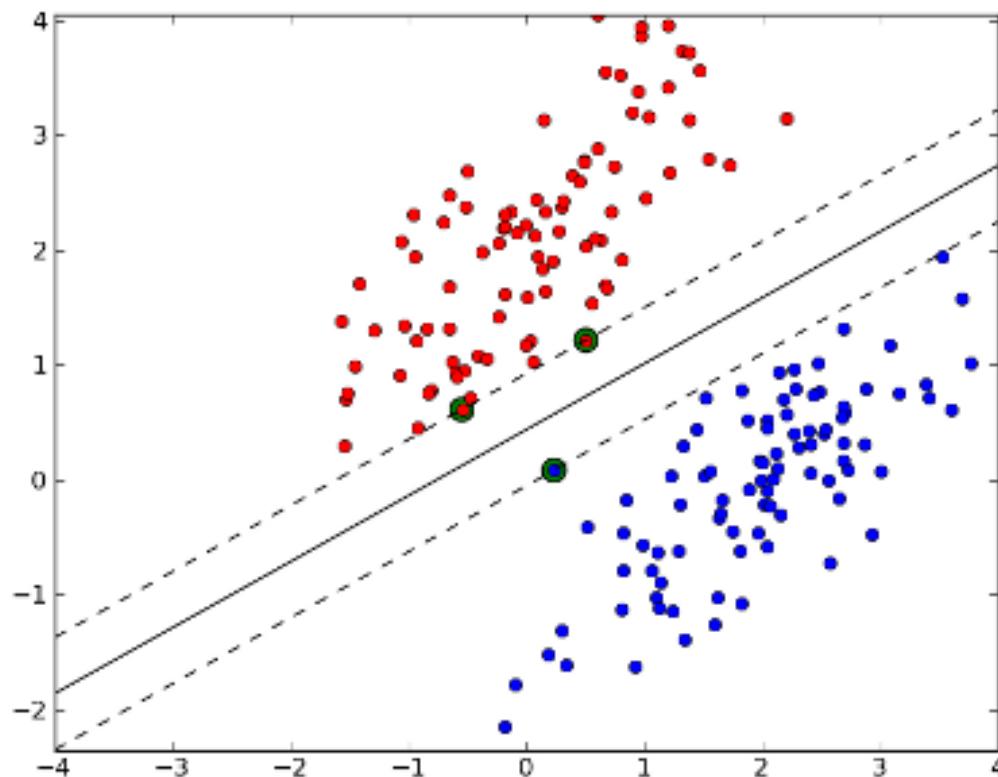


Once the hyper-plane is defined, the model is able to predict the class label for unseen samples.

# Support Vector Machine

The main parameter to be set in a SVM classifier is the kernel function, which is mostly set as linear, polynomial or Gaussian Radial Basis Function (RBF). Linear kernels are defined as

$$k(x_i, x_j) = (x_i \cdot x_j)$$



# Support Vector Machine | Proprietà degli Iperpiani

$$D(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + b$$

$\mathbf{w}$ : vettore normale all'iperpiano

$b / \|\mathbf{w}\|$  : distanza dall'origine

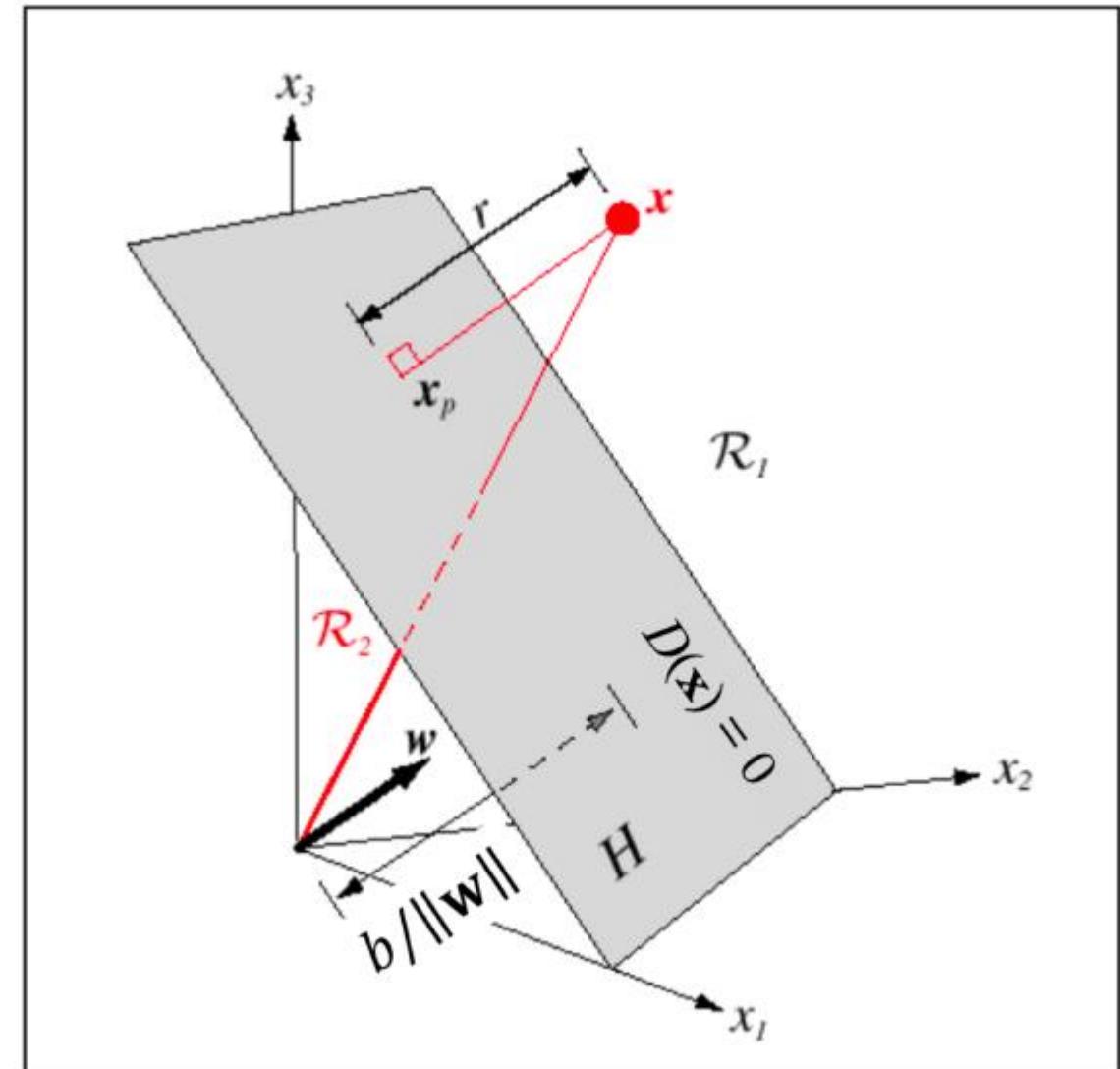
$D(\mathbf{x}) = 0$ : luogo dei vettori sull'iperpiano

$$D(\mathbf{x}_p) = 0$$

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

$$D(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + b = r \|\mathbf{w}\|$$

distanza di un vettore  $\mathbf{x}$  dall'iperpiano  $r = D(\mathbf{x}) / \|\mathbf{w}\|$

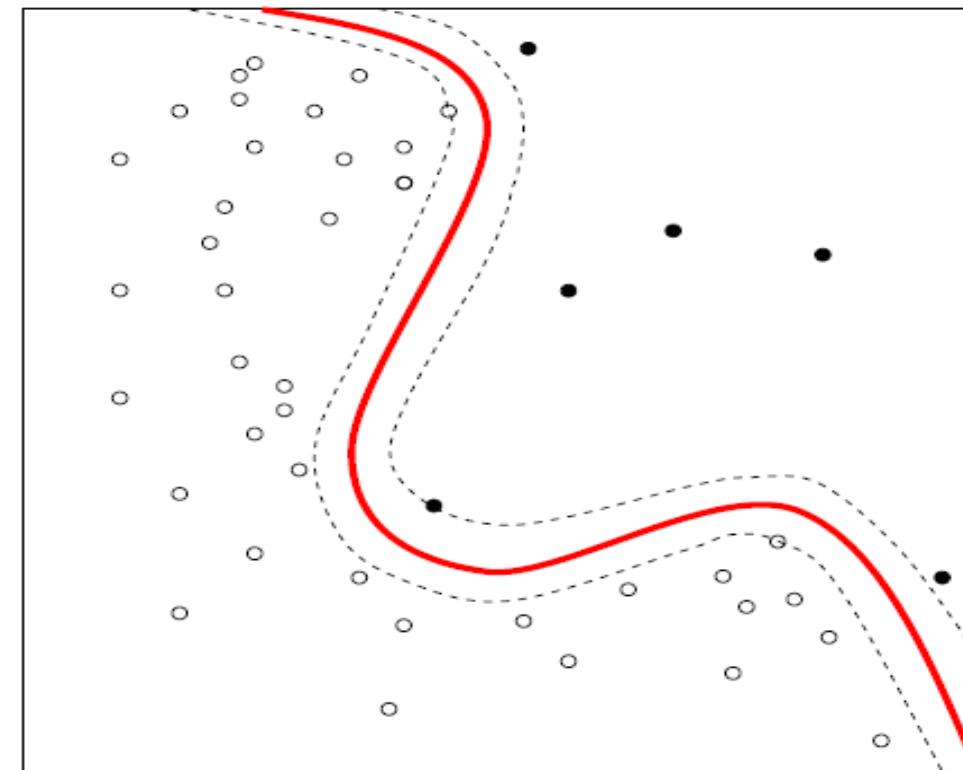


# Support Vector Machine

while (homogeneous and inhomogeneous) polynomial kernels are given by

$$k(x_i, x_j) = (x_i \cdot x_j)^d$$

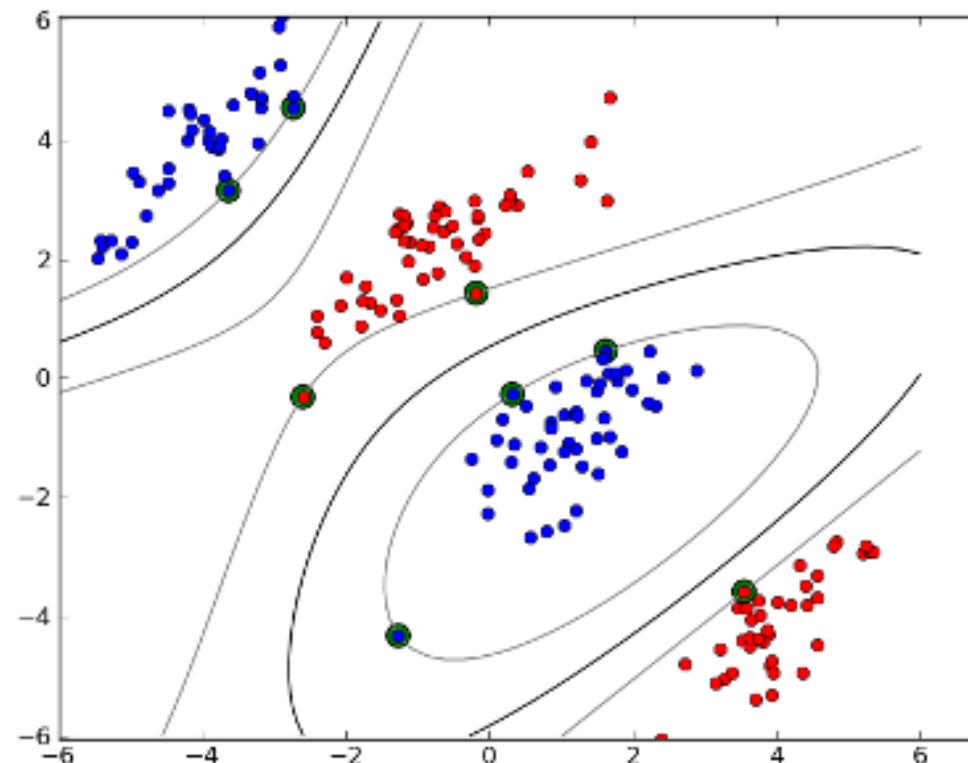
$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d$$



# Support Vector Machine

and Gaussian RBF kernels  
have the following form

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \quad \gamma > 0$$



# Support Vector Machine

Imagine we are given  $N$  observations, each one consisting of a pair: an input vector  $x_n \in R^N$ ,  $n = 1, \dots, N$  and the corresponding target value  $t_n \in \{\pm 1\}$ , given to us by a trusted source. Data are assumed to be *iid*, i.e., independently drawn from an unknown probability distribution  $P(x; t)$  and identically distributed. In our example,  $x_n \in R^N$  might be a vector of pixel values (representing images of patients) and  $t_n \in \{\pm 1\}$  would be +1 for images belonging to the feature class, -1 for images belonging to the normal class. The aim is to estimate a function that will correctly classify unseen examples  $(x; t)$ , i.e., we want  $f(x) = t$  for samples  $(x; t)$  that were also generated from  $P(x; t)$ . This problem reduces to the goal of separating the two classes by a function which is induced from available examples, in order to produce a classifier, based on input-output data training, which will work well on unseen samples. Let us now make the further assumption that the training data set is linearly separable in feature space and consider the example in Figure 1.3.1: here there are many possible linear classifiers that can separate the data, but there is only one that maximizes the margin (i.e., the distance between it and the nearest data point of each class). This classifier is termed the optimal separating hyper-plane. Intuitively, we would expect this boundary to generalize well as opposed to the other possible boundaries (Figure 1.3.1).

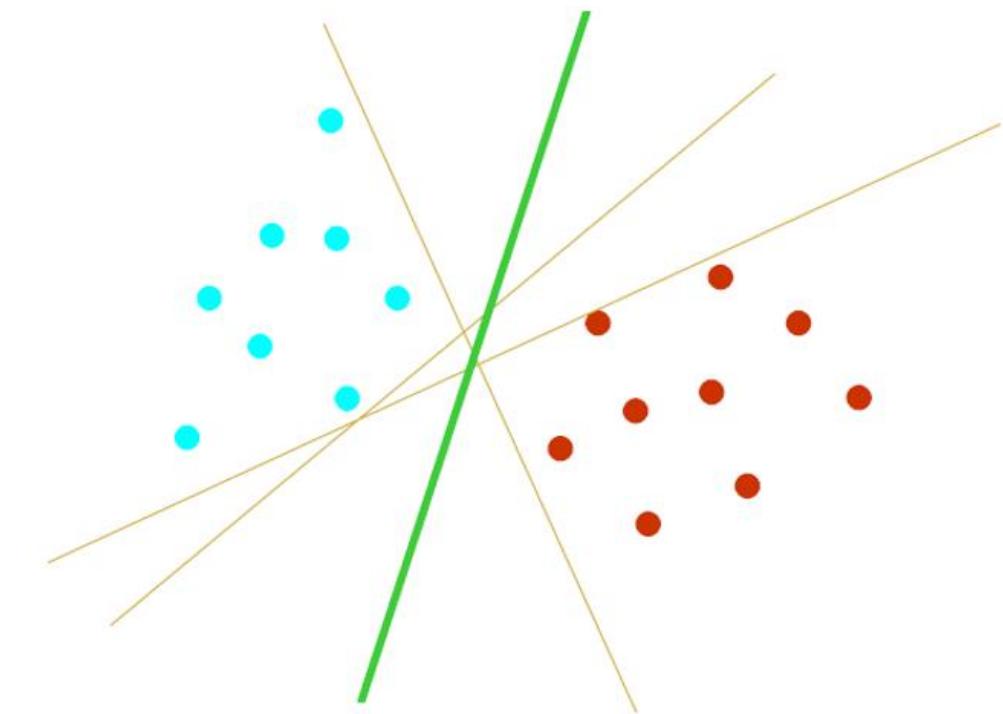


Figure 1.3.1 Optimal separating hyper-plane

# Support Vector Machine

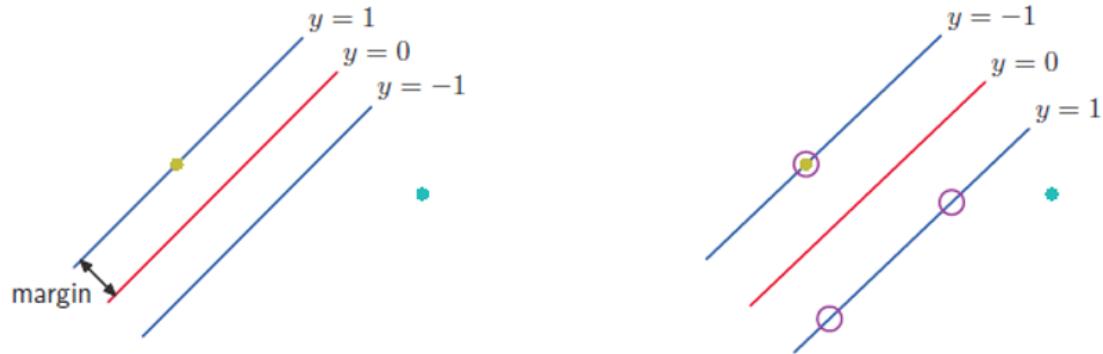


Figure 1.3.2 Margin and support vectors

If we consider the following class of hyper-planes (linear model)

$$y(x) = \omega^T \varphi(x) + b \quad (1.3.1)$$

Where  $\varphi(x)$  denotes a fixed feature-space transformation and the explicit parameter  $b$  is called bias parameter, then new data points  $x$  can be classified according to the sign of  $y(x)$ . In fact, as we assumed the training data set to be linearly separable in feature space, by definition there exists at least one choice of the parameters  $w$  and  $b$  such that a function of the form 1.3.1 satisfies  $y(x_n) > 0$  for points having  $t_n = +1$  and  $y(x_n) < 0$  for points having  $t_n = -1$ . Furthermore,  $t_n \cdot y(x_n) > 0$  for all training data points. As we said before, there may exist many such solutions that separate the classes exactly. This problem can be solved by introducing the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples, as illustrated in Figure 1.3.2.

In SVM, the decision boundary is chosen to be the one for which the margin is maximized. The maximum margin solution can be motivated using computational learning theory, also known as statistical learning theory or VC (Vapnik-Chervonenkis) theory.

# Support Vector Machine

It must be noted that the margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left part of Figure 1.3.2. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right part of Figure 1.3.2. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles. The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left part of Figure 1.3.2. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right of Figure 1.3.2. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles.

In general, the perpendicular distance of a point  $x$  from a hyper-plane defined by  $y(x) = 0$ , where  $y(x)$  takes the form specified in equation 1.3.1, is given by  $|y(x)|/\|w\|$ . Furthermore, in this case we are only interested in solutions for which all data points are correctly classified, so that  $t_n \cdot y(x_n) > 0$  for all  $n$ . Thus the distance of a point  $x_n$  to the decision surface is given by

$$\frac{t_n \cdot y(x_n)}{\|w\|} = \frac{t_n \cdot (\omega^T \varphi(x) + b)}{\|w\|} \quad (1.3.2)$$

# Support Vector Machine

The margin is given by the perpendicular distance to the closest point  $x_n$  from the data set, and we wish to optimize the parameters  $w$  and  $b$  in order to maximize this distance. Thus the maximum margin solution is found by solving

$$\operatorname{argmax}_{W,b} \left\{ \frac{1}{\|w\|} \min_n [t_n \cdot (\omega^T \varphi(x) + b)] \right\} \quad (1.3.3)$$

where we have taken the factor  $\frac{1}{\|w\|}$  outside the optimization over  $n$  because  $w$  does not depend on  $n$ . Solving this optimization problem simply reduces to the requirement of minimizing  $\|w\|^2$

$$\operatorname{argmax}_{W,b} \left\{ \frac{1}{2} \|w\|^2 \right\} \quad (1.3.4)$$

(the factor of  $\frac{1}{2}$  is included for later convenience), subject to the constraints

$$t_n \cdot (\omega^T \varphi(x) + b) \geq 1, n = 1, \dots, N \quad (1.3.5)$$

# Support Vector Machine

It appears that the bias parameter  $b$  has disappeared from the optimization. However, it is determined implicitly via the constraints, because these require that changes to  $\|w\|$  be compensated by changes to  $b$ . This constrained optimization problem is dealt with by introducing Lagrange multipliers  $a_n > 0$ , with one multiplier  $a_n$  for each of the constraints in equation 1.3.5, giving the Lagrangian function

$$L(\omega, b, a) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^N a_n \{t_n \cdot (\omega^T \varphi(x) + b) - 1\} \quad (1.3.6)$$

where  $a = (a_1, \dots, a_N)^T$ . The negative sign in front of the Lagrange multiplier term is a consequence of the fact that minimization is performed with respect to  $w$  and  $b$ , and maximization is performed with respect to  $a$ . By setting the derivatives of  $L(\omega, b, a)$  with respect to  $w$  and  $b$  equal to zero, the two following conditions can be obtained:

$$\omega = \sum_{n=1}^N a_n \cdot t_n \cdot \varphi(x) \quad (1.3.7)$$

and

$$\sum_{n=1}^N a_n \cdot t_n = 0 \quad (1.3.8)$$

# Support Vector Machine

Eliminating  $w$  and  $b$  from  $L(\omega, b, a)$  using these conditions, then, gives the dual representation of the maximum margin problem in which we maximize

$$\tilde{L}(a) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N a_m a_n t_m t_n k(x_n, x_m) \quad (1.3.9)$$

with respect to the constraints

$$a_n \geq 0, n = 1, \dots, N \quad (1.3.10)$$

$$\sum_{n=1}^N a_n \cdot t_n = 0 \quad (1.3.11)$$

Here, the kernel function is defined by  $k(x, x') = \varphi(x)^T \cdot \varphi(x')$ : this kernel formulation makes clear the role of the constraint that the kernel function  $k(x, x')$  be positive definite, because this ensures that the Lagrangian function  $\tilde{L}(a)$  is bounded below, giving rise to a well-defined optimization problem. In order to classify new data points using the trained model, we evaluate the sign of  $y(x)$  defined in equation 1.3.1. This can be expressed in terms of the parameters  $a_n$  and the kernel function by substituting for  $\omega$  using 1.3.7 to give the hyper-plane decision function

$$y(x) = \sum_{n=1}^N a_n \cdot t_n \cdot k(x, x_n) + b \quad (1.3.12)$$

# Support Vector Machine

It can be shown that a constrained optimization of this form satisfies the Karush-Kuhn-Tucker (KKT) conditions, which, in this case, require that the following three properties hold:

$$a_n \geq 0 \quad (1.3.13)$$

$$t_n \cdot y(x_n) - 1 \geq 0 \quad (1.3.14)$$

$$a_n \{t_n \cdot y(x_n) - 1\} = 0 \quad (1.3.15)$$

Thus for every data point, either  $a_n = 0$  or  $t_n \cdot y(x_n) = 1$ . Any data point for which  $a_n = 0$  will not appear in the sum in 1.3.12 and, hence, they will play no role in making predictions for new data points. The remaining data points are called support vectors, and because they satisfy  $t_n \cdot y(x_n) = 1$ , they correspond to points that lie on the maximum margin hyper-planes in feature space, as illustrated in Figure 1.3.2. This property is central to the practical applicability of SVM. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained. Having solved the quadratic programming problem and found a value for  $a$ , we can then determine the value of the threshold parameter  $b$  by noting that any support vector  $x_n$  satisfies  $t_n \cdot y(x_n) = 1$ .

# Support Vector Machine

Using 1.3.12, this gives

$$t_n(\sum_{m \in S} a_m t_m k(x_n, x_m) + b) = 1 \quad (1.3.16)$$

where  $S$  denotes the set of indices of the support vectors. Although we can solve this equation for  $b$  using an arbitrarily chosen support vector  $x_n$ , a numerically more stable solution is obtained by first multiplying through by  $t_n$ , making use of  $(t_n)^2 = 1$ , and, then, averaging these equations over all support vectors and solving for  $b$  to give

$$b = \frac{1}{N_S} \sum_{n \in S} (t_n - \sum_{m \in S} a_m t_m k(x_n, x_m)) \quad (1.3.17)$$

where  $N_S$  is the total number of support vectors.

# Support Vector Machine

In practice...

# Support Vector Machine

In practice, a separating hyper-plane may not exist, e.g., if a high noise level causes a large overlap of the class-conditional distribution. In order to overcome this limitation, Cortes and Vapnik (1995) proposed a modified version of SVM introducing the idea of soft margin, which is useful when training classes cannot be sharply discriminated. Specifically, the soft margin approach allows to misclassify a fraction of training samples, while preserving the ability of the hyper-plane to maximizing its distance from the nearest samples of the two classes.

In order to allow for the possibility of violating 1.3.14, the general approach has to be modified so that data points are allowed to be on the *wrong side* of the margin boundary, but with a penalty that increases with the distance from that boundary. For the subsequent optimization problem, it is convenient to make this penalty a linear function of this distance; to do this, we introduce slack variables

$$\xi_n \geq 0, n = 1, \dots N \tag{1.3.18}$$

with one slack variable for each training data point. These are defined by  $\xi_n = 0$  for data points that are on or inside the correct margin boundary and  $\xi_n = |t_n - y(x_n)|$  for other points.

# Support Vector Machine

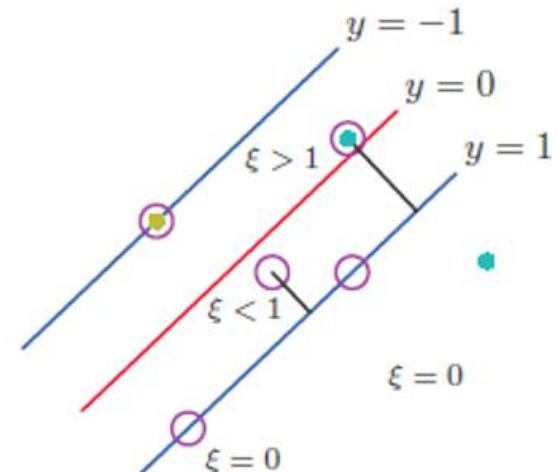
Thus, a data point that is on the decision boundary  $y(x_n) = 0$  will have  $\xi_n = 1$  and points with  $\xi_n > 1$  will be misclassified. In this case, classification constraints relax to

$$t_n \cdot y(x_n) \geq 1 - \xi_n, n = 1, \dots, N \quad (1.3.19)$$

in which the slack variables are constrained to satisfy  $\xi_n \geq 0$ . Data points for which  $\xi_n = 0$  are correctly classified and are either on the margin or on the correct side of the margin. Points for which  $0 < \xi_n \leq 1$  lie inside the margin, but on the correct side of the decision boundary, and those data points for which  $\xi_n \geq 1$  lie on the wrong side of the decision boundary and are misclassified, as illustrated in Figure 1.3.3. This is sometimes described as relaxing the hard margin constraint to give a soft margin and it allows some of the training set data points to be misclassified. Note that while slack variables allow for overlapping class distributions, this framework is still sensitive to outliers, because the penalty for misclassification increases linearly with  $\xi$ . In order to realize a soft margin classifier, we now have to minimize

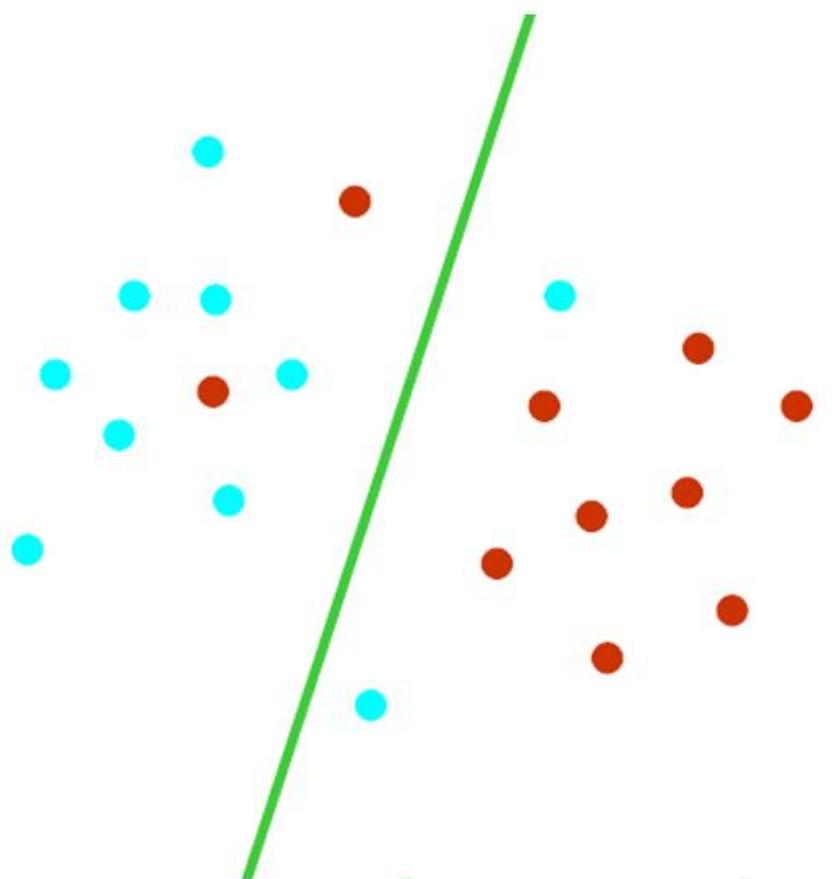
$$\frac{1}{2} \|w\|^2 + C \sum_{n=1}^N \xi_n \quad (1.3.20)$$

subject to the constraints 1.3.18 and 1.3.19; here, the parameter  $C > 0$  controls the trade-off between the slack variable penalty and the margin.



**Figure 1.3.3** Slack variables in support vector classifiers objective function.  
Data points with circles around them are support vectors.

# Support Vector Machine



**Figure 1.3.4 Generalized optimal separating hyper-plane**

Incorporating kernels and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing 1.3.9, subject to the constraints

$$0 \leq a_n \leq C \quad (1.3.21)$$

$$\sum_{n=1}^N a_n t_n = 0 \quad (1.3.22)$$

for  $n = 1, \dots, N$  (constraints in 1.3.21 are known as box constraints).

The only difference from the separable case is the upper bound  $C$  on the Lagrange multipliers  $a_n$ . In this way, the influence of the individual patterns (which could be outliers) gets limited. As above, the solution shows that predictions for new data points are again made by using 1.3.12. The threshold  $b$  can be computed by exploiting the fact that for all support vectors  $x_n$ , with  $0 \leq a_n \leq C$ , the slack variable  $\xi_n = 0$  and, hence, will satisfy 1.3.16. As before, a subset of the data points may have  $a_n = 0$ , in which case it does not contribute to the predictive model 1.3.12. The remaining data points constitute the support vectors.

# Support Vector Machine

In practice, when dealing with a SVM-based classifier, the main parameter to be set for a study is the kernel function. The most popular kernels in literature are the linear, the polynomial and the Gaussian Radial Basis Function (RBF) kernels:

$$k(x_i, x_j) = (x_i \cdot x_j) \quad (1.3.23)$$

$$k(x_i, x_j) = (x_i \cdot x_j)^d \quad (1.3.24)$$

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d \quad (1.3.25)$$

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \quad \gamma > 0 \quad (1.3.26)$$

where relation 1.3.23 is associated to linear kernels, relations 1.3.24-25 are associated to (homogeneous and inhomogeneous) polynomial kernels and relation 1.3.26 is associated to Gaussian RBF kernels, respectively.

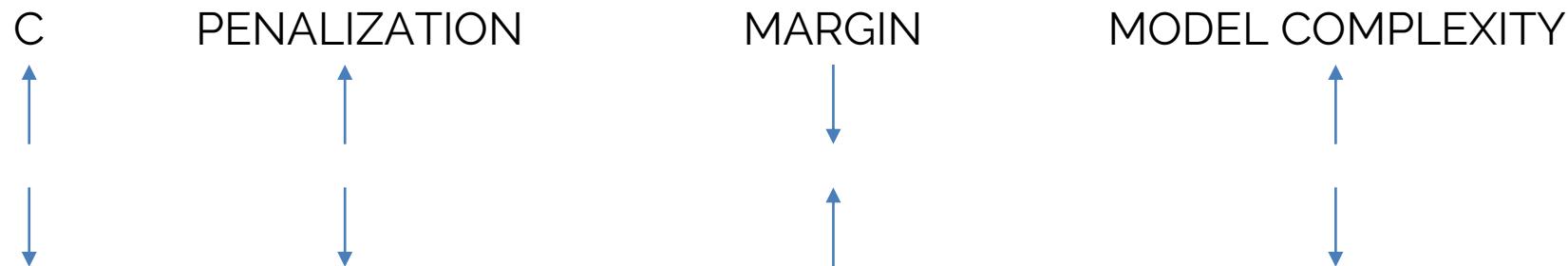
# Support Vector Machine | C HYPERPARAMETER

**The C parameter** is related to “how much you want to avoid misclassification” of each training example

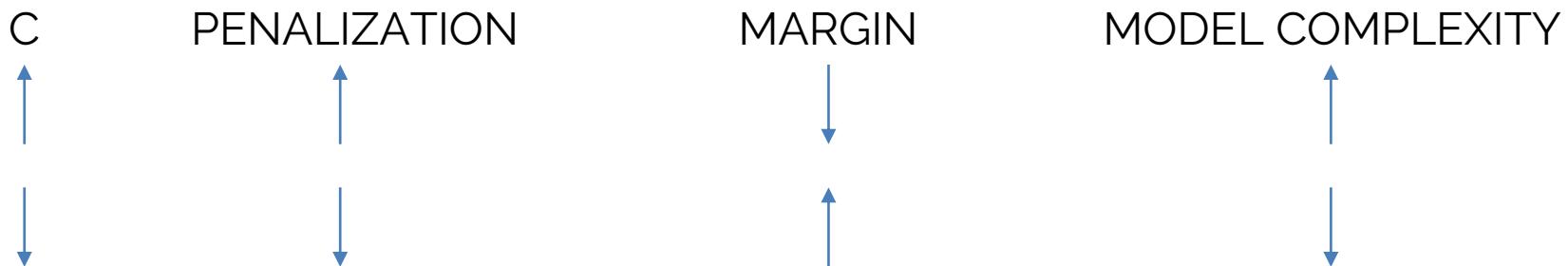
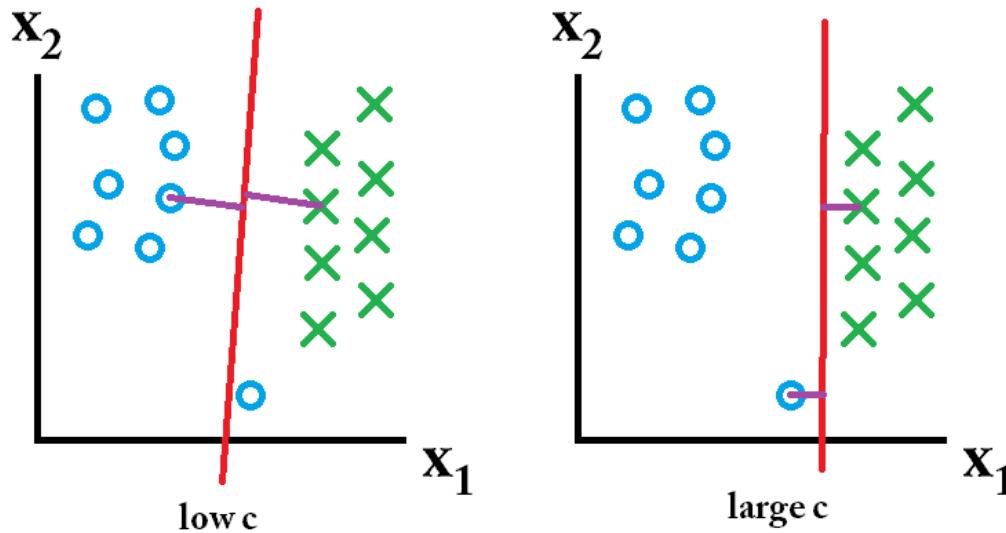
LARGE VALUES of C correspond to a smaller-margin separating hyperplane if that hyperplane correctly classifies all training data

SMALL VALUES of C correspond to a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

CONSIDER THAT an optimal C parameter should be larger when you scale down your data, smaller when data are not scaled

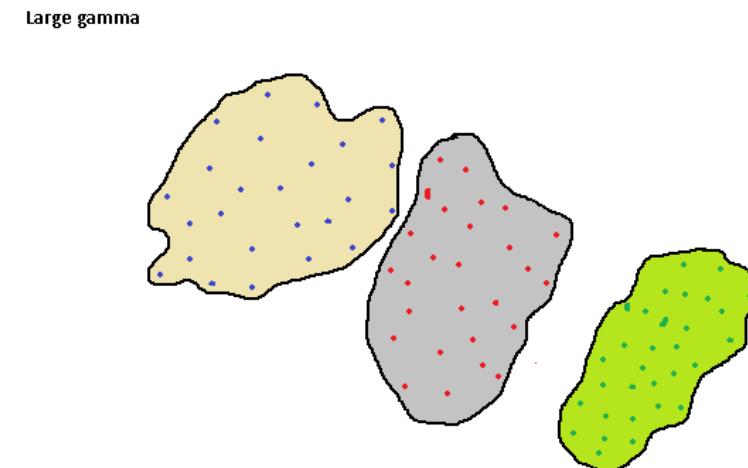
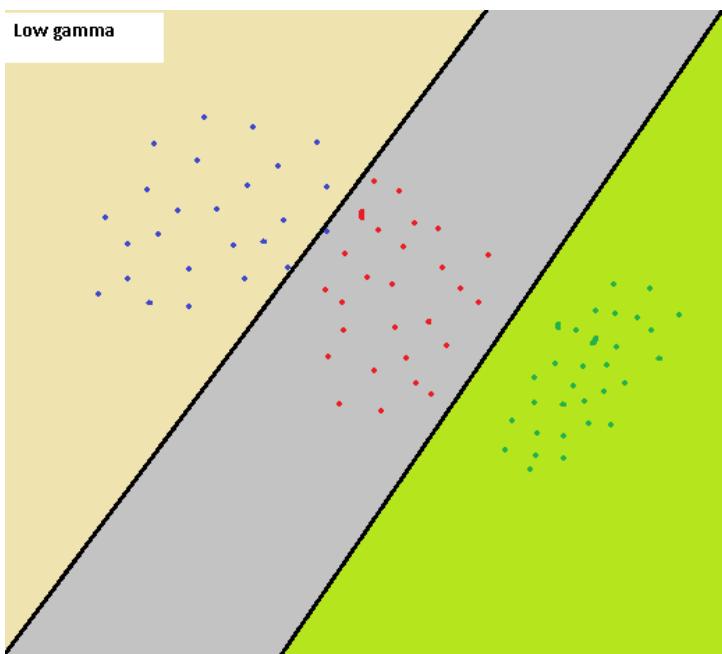


# Support Vector Machine | C HYPERPARAMETER



# Support Vector Machine | GAMMA HYPERPARAMETER

"Gamma parameter of RBF controls the distance of influence of a single training point. Low values of gamma indicates a large similarity radius which results in more points being grouped together. For high values of gamma, the points need to be very close to each other in order to be considered in the same group (or class). Therefore, models with very large gamma values tend to overfit."



If gamma is large, the effect of  $c$  becomes negligible.  
If gamma is small,  $c$  affects the model just like how it  
affects a linear model.

$$0.0001 < \text{gamma} < 10$$

$$0.1 < c < 100$$

# Support Vector Machine

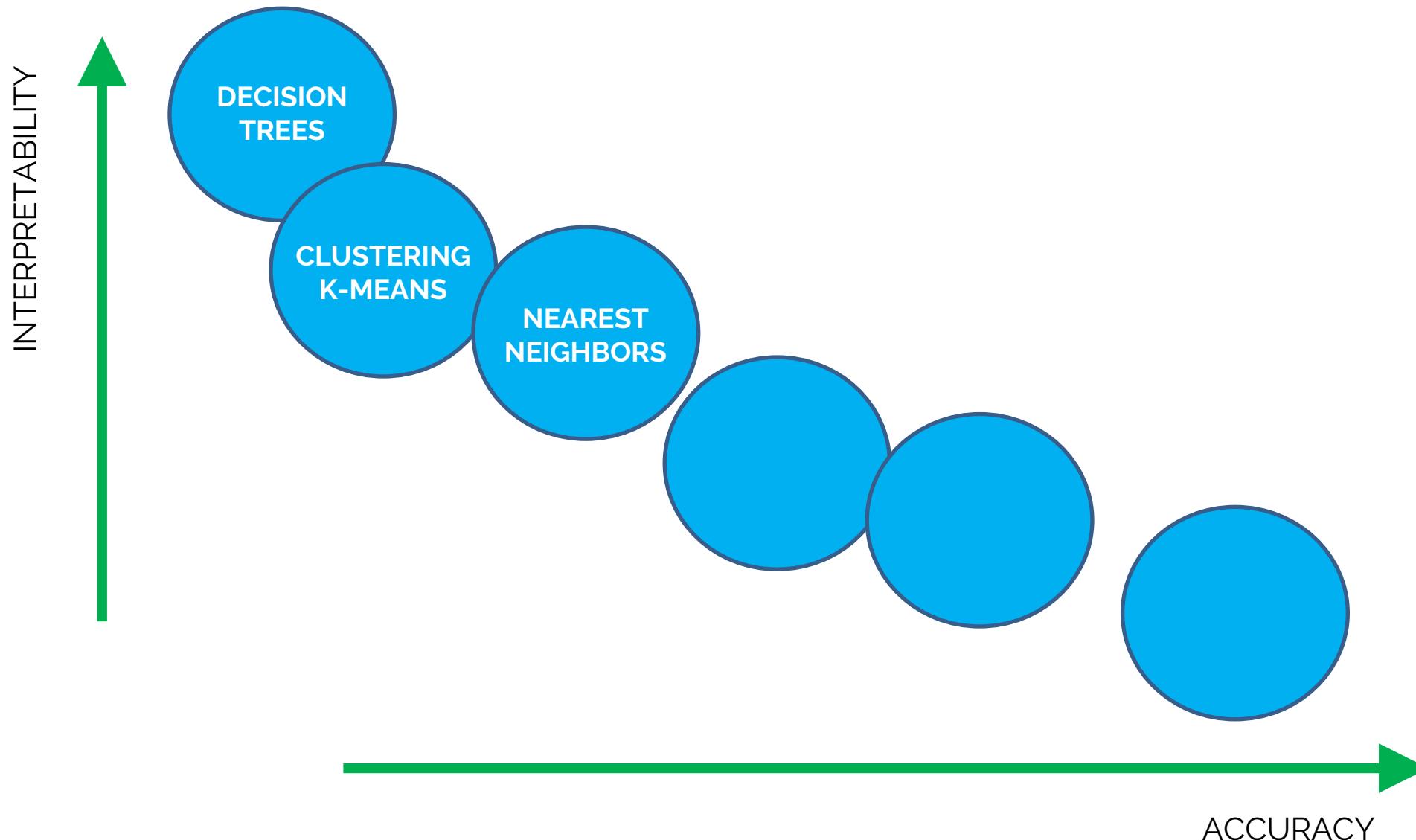
Advantages:

- Very powerful framework
- SVMs work well in practice (even with small datasets)

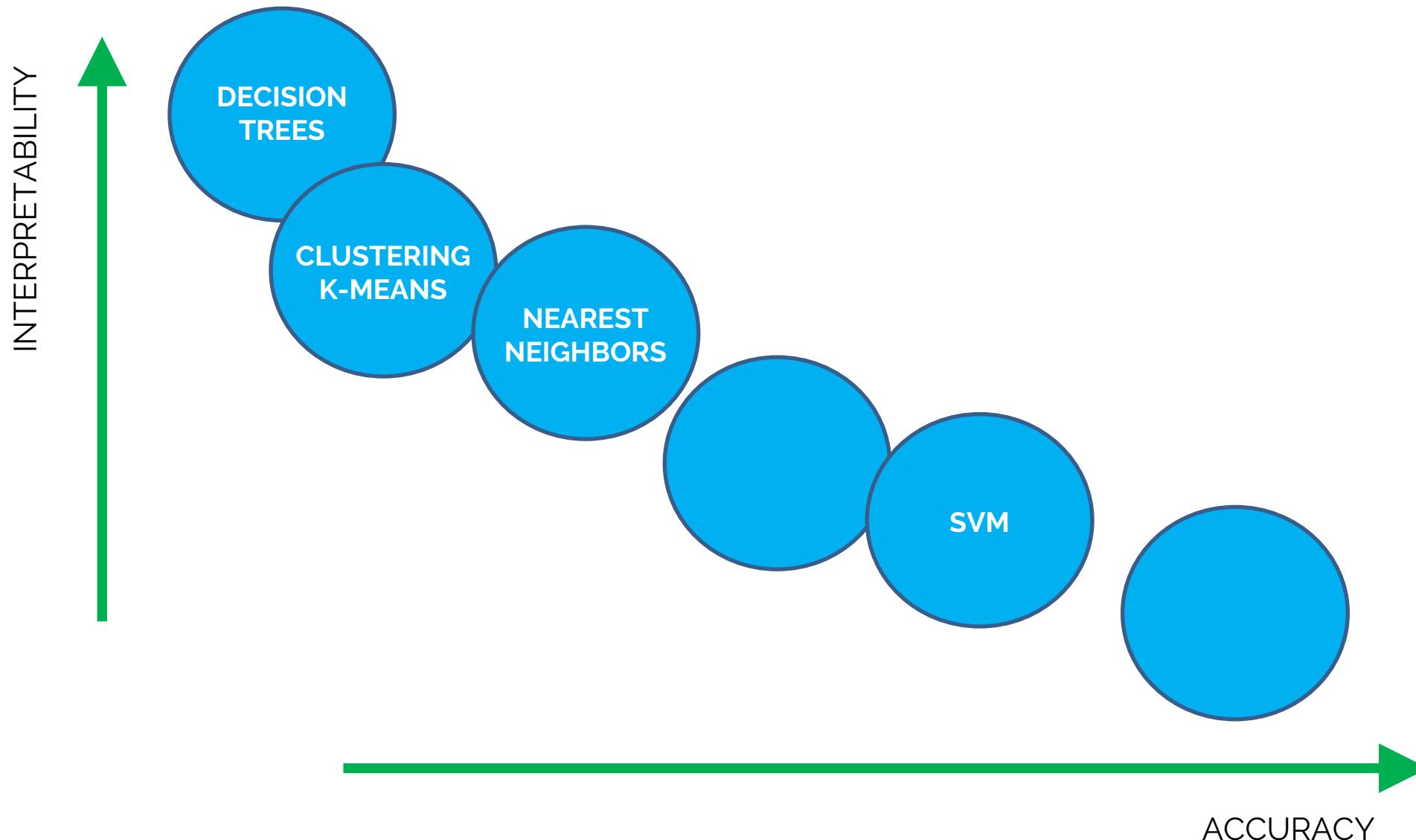
Drawbacks:

- High computational costs for training
- SVMs only allow direct binary classification
- Classification [+1/-1] is the only output (no probability-of-classification is given)

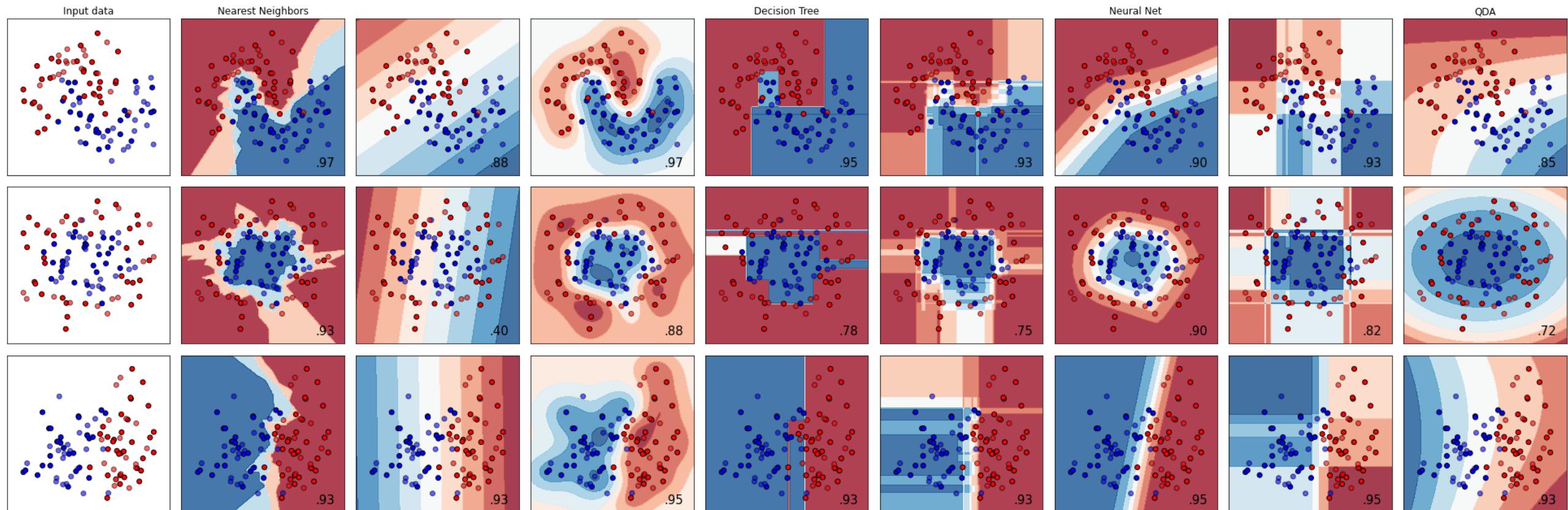
# Interpretability-Accuracy TRADEOFF



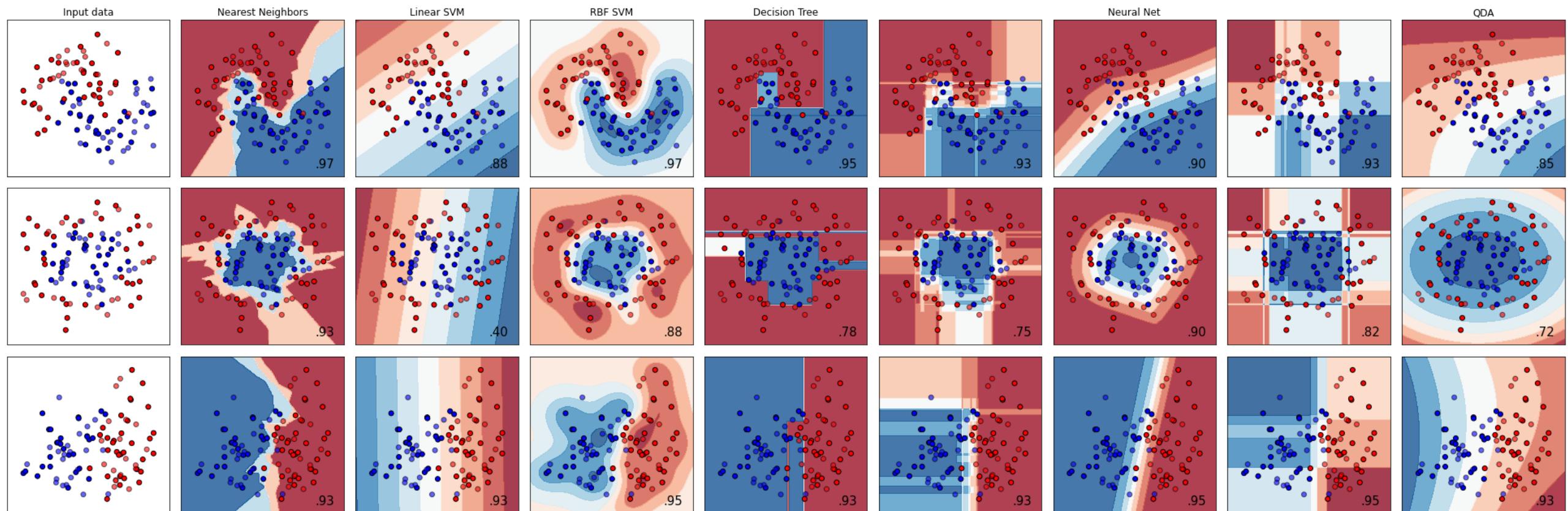
# Interpretability-Accuracy TRADEOFF



# Which is which | Decision Function



# Which is which | Decision Function



# Support Vector Machine

- Consider the following generalization of the  $k$ -NN algorithm (specialized to binary classification):

$$\hat{y}(\vec{x}) \leftarrow \text{sign} \left( \sum_{i=1}^N y_i d(\vec{x}_i, \vec{x}) \right) \quad \text{with } d(\vec{x}_i, \vec{x}) = \frac{1}{\|\vec{x}_i - \vec{x}\|_2^2} \text{ or... } d(\vec{x}_i, \vec{x}) = \exp \left( -\frac{\|\vec{x}_i - \vec{x}\|_2^2}{2\sigma^2} \right)$$

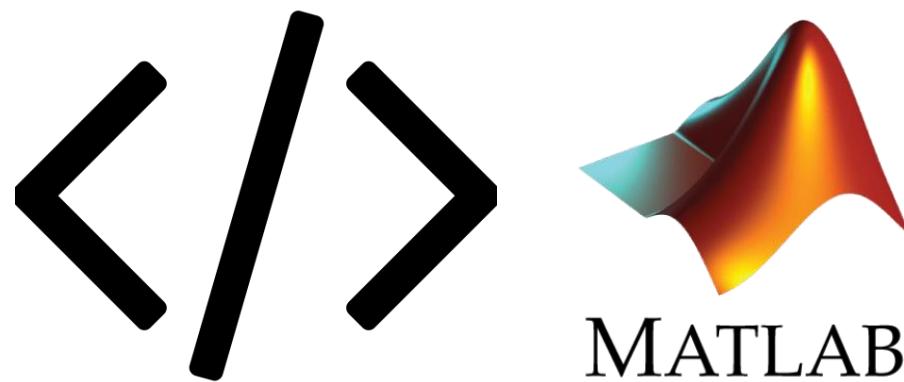
- Looks at *all* training points (i.e.,  $k=N$ ), but weights the  $i$ 'th training point's label by how far  $\mathbf{x}_i$  is from  $\mathbf{x}$
- Now compare this to classification with SVM and a Gaussian kernel:

$$\hat{y}(\vec{x}) \leftarrow \text{sign} \left( \sum_{i=1}^N \alpha_i y_i K(\vec{x}_i, \vec{x}) \right) \quad K(\vec{u}, \vec{v}) = \exp \left( -\frac{\|\vec{u} - \vec{v}\|_2^2}{2\sigma^2} \right) \quad 0 \leq \alpha_i \leq C$$

- The discriminant functions are nearly identical! The SVM has parameters  $\alpha_i$  that can be learned to maximize predictive accuracy

FROM DAVID SONTAG

# Support Vector Machine



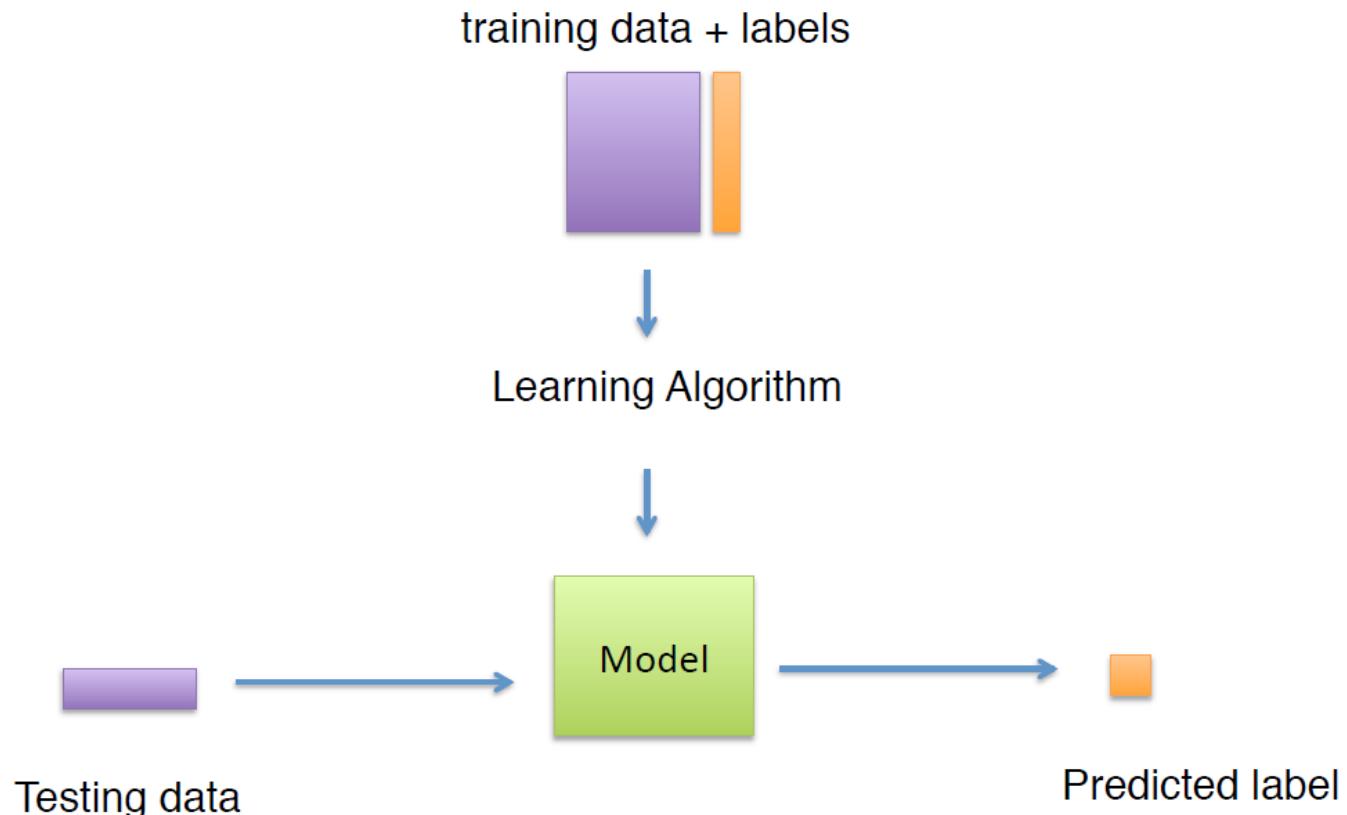
# CLASSIFICATION

## ENSEMBLES OF CLASSIFIERS

WITH SOME SLIDES FROM PROF. GAVIN BROWN

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



## Neural Network Ensembles

LARS KAI HANSEN AND PETER SALAMON

**Abstract**—We propose several means for improving the performance and training of neural networks for classification. We use crossvalidation as a tool for optimizing network parameters and architecture. We show further that the remaining residual “generalization” error can be reduced by invoking ensembles of similar networks.

**Index Terms**—Crossvalidation, fault tolerant computing, neural networks, *N*-version programming.

### I. INTRODUCTION

RECENT schemes for training neural networks involving hidden neurons have caused a resurgence of interest in nonalgorithmic supervised learning. A supervised learning scheme is implemented using a *database* which consists of a set of input patterns (a sample from the set of possible inputs) together with the corresponding targets (classifications). The objective of the training is to

optimize the performance of the network, we can optimize such performance by varying network characteristics and architecture.

A residual error will typically remain even after optimizing all available network characteristics [6]. To further reduce this error we propose to use a device from fault tolerant computing [7]. We run not a single network but an *ensemble* of networks, each of which have been trained on the same database. The basic idea is to classify a given input pattern by obtaining a classification from each copy of the network and then using a consensus scheme to decide the collective classification by vote.

### II. CROSSVALIDATION FOR NETWORK OPTIMIZATION

For supervised learning we employ a *database* as described above including a representative sample of the

International Journal of Forecasting 5 (1989) 559–583  
North-Holland

559

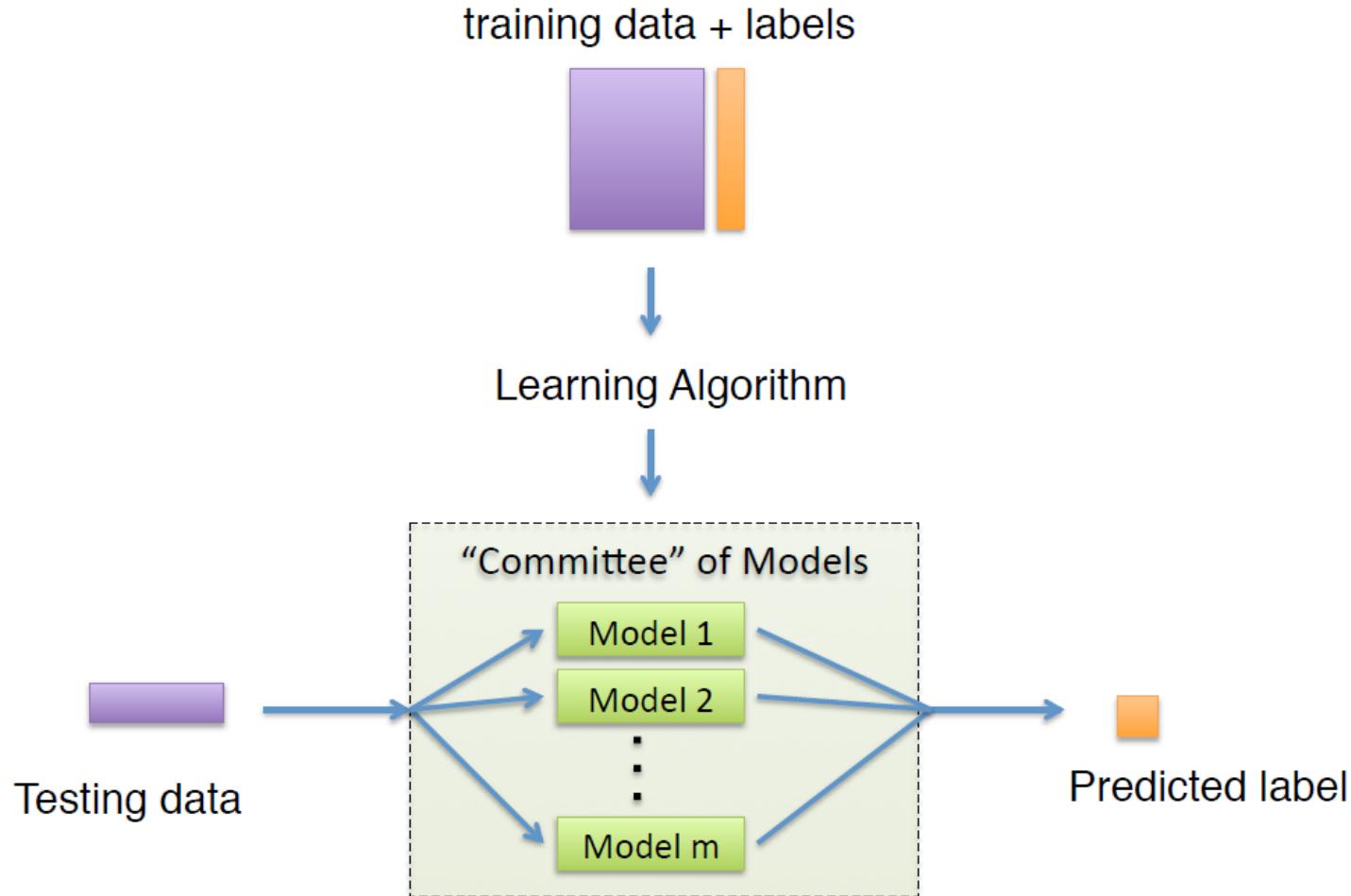
### Combining forecasts: A review and annotated bibliography

Robert T. CLEMEN \*  
College of Business Administration, University of Oregon, Eugene, OR 97403-1208, USA

**Abstract:** Considerable literature has accumulated over the years regarding the combination of forecasts. The primary conclusion of this line of research is that forecast accuracy can be substantially improved through the combination of multiple individual forecasts. Furthermore, simple combination methods often work reasonably well relative to more complex combinations. This paper provides a review and annotated bibliography of that literature, including contributions from the forecasting, psychology, statistics, and management science literatures. The objectives are to provide a guide to the literature for students and researchers and to help researchers locate contributions in specific areas, both theoretical and applied. Suggestions for future research directions include (1) examination of simple combining approaches to determine reasons for their robustness, (2) development of alternative uses of multiple forecasts in order to

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

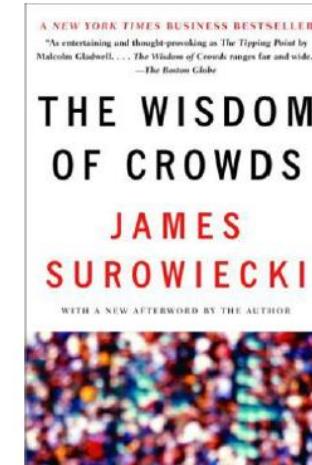


## Combining general predictions?

1906 ... county fair in Cornwall, England.

Competition: guess the weight of the cow!

Francis Galton recorded some statistics on the day....



# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

NATURE

[MARCH 7, 1907]

mean of the ire for month	Distribution of the estimates of the dressed weight of a particular living ox, made by 787 different persons.				
	Degrees of the length of Array 0—100	Estimates in lbs.	* Centiles		
			Observed deviates from 1207 lbs.	Normal p.e = 37	Excess of Observed over Normal
1 year- Both years. Bulletin contains station ults of ions in 151 and y in a ation is and is l applica- o with tempera- maximum absolute absolute ll rain- nt was Most tember. it being J. D.	5	1074	- 133	- .90	+ .43
	10	1109	- 98	- .70	+ .28
	15	1126	- 81	- .57	+ .24
	20	1148	- 59	- .46	+ .13
	$q_1$ 25	1162	- 45	- .37	+ .8
	30	1174	- 33	- .29	+ .4
	35	1181	- 26	- .21	+ .5
	40	1188	- 19	- .14	+ .5
	45	1197	- 10	- .07	+ .3
	$m$ 50	1207	0	0	0
	55	1214	+ 7	+ .7	0
	60	1219	+ 12	+ .14	- .2
	65	1225	+ 18	+ .21	- .3
	70	1230	+ 23	+ .29	- .6
	$q_3$ 75	1236	+ 29	+ .37	- .8
	80	1243	+ 36	+ .46	- 10
	85	1254	+ 47	+ .57	- 10
	90	1267	+ 52	+ .70	- 18
	95	1293	+ 86	+ .90	- 4

$q_1$ ,  $q_3$ , the first and third quartiles, stand at 25° and 75° respectively.  
 $m$ , the median or middlemost value, stands at 50°.  
The dressed weight proved to be 1198 lbs.



787 guesses.

Truth      1198 lb  
(~543kg)

Median    1207 lb  
Mean     1197 lb

## Combining Votes

In 1786 Nicolas de Condorcet (political theorist) asked how do parliaments behave when voting?

...assuming M independent voters...

If a single voter has a probability  $\epsilon$  of making an error, then

$$p(\text{exactly } k \text{ errors}) = \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$

and...

$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$



Marquis de  
Condorcet

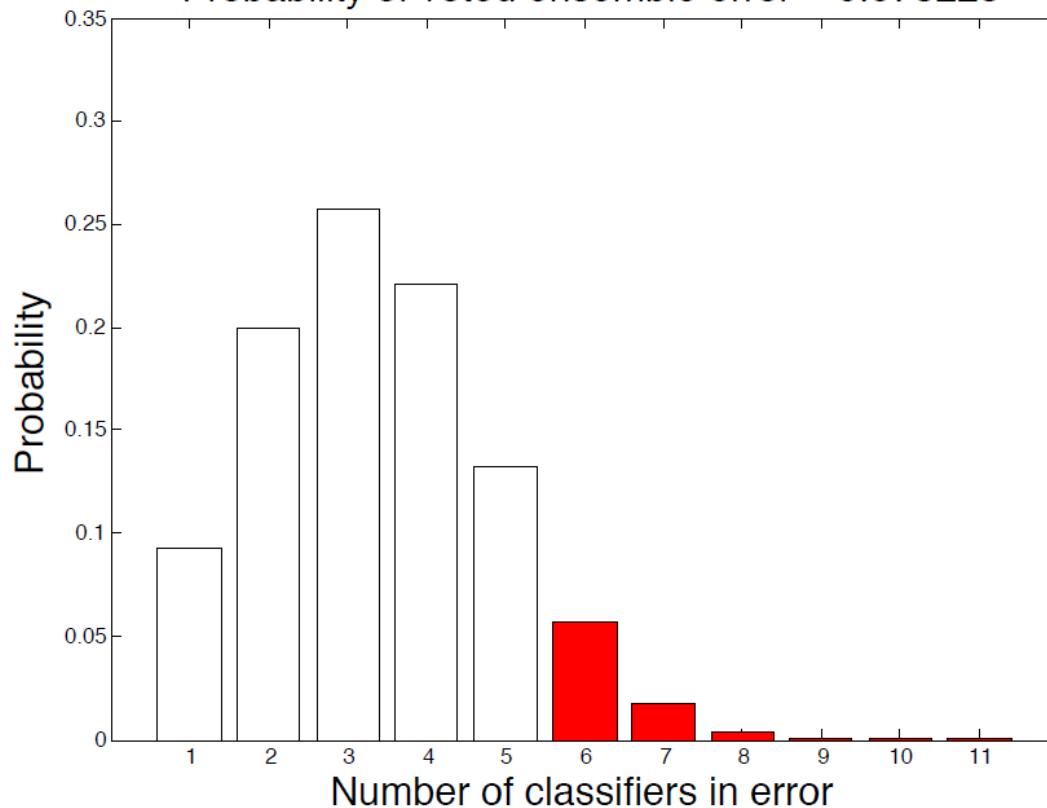


# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil}^M \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$

11 classifiers. Individual error probability = 0.3  
Probability of voted ensemble error = 0.078225

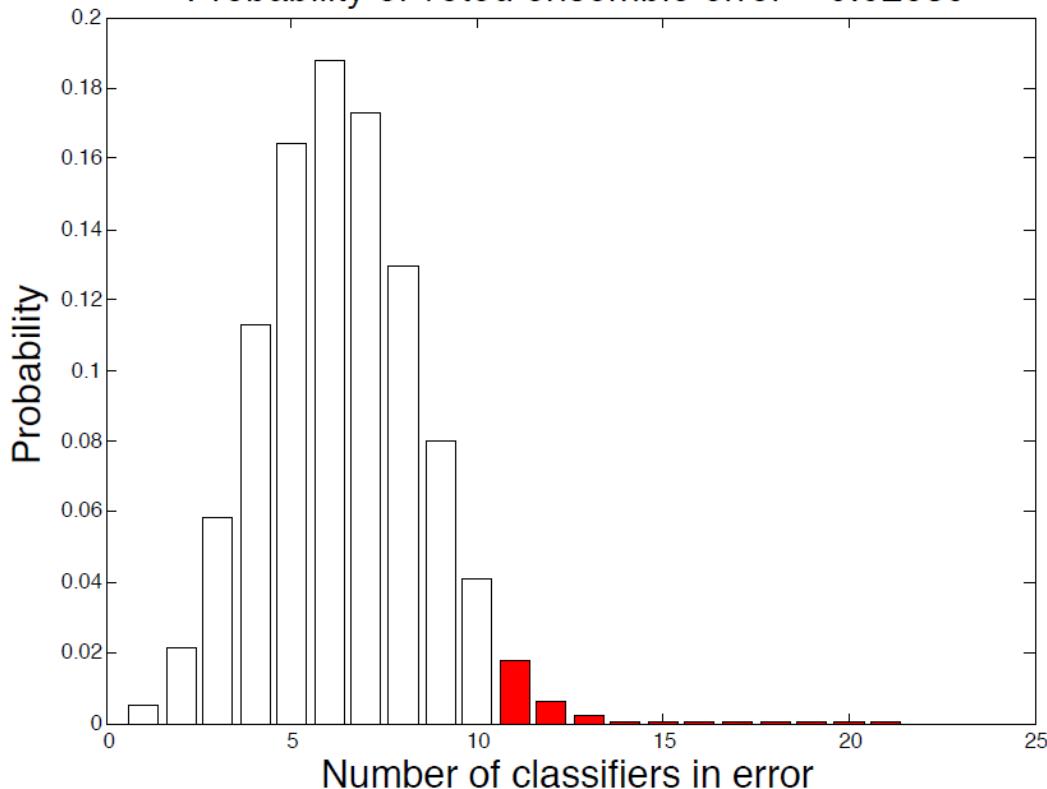


# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$

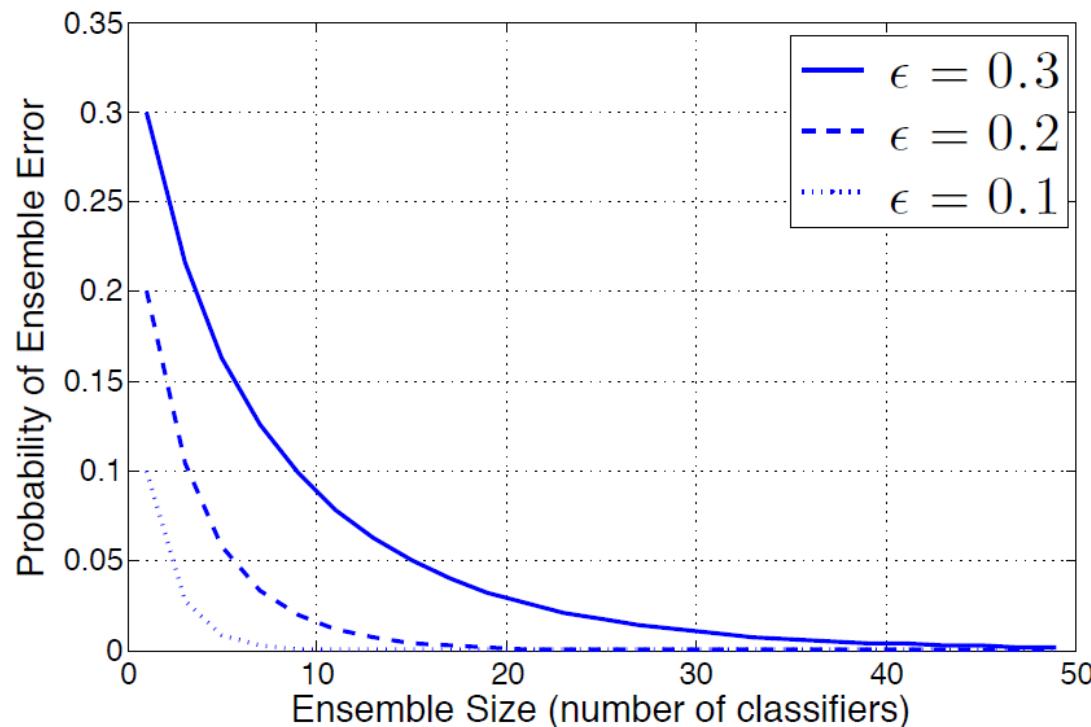
21 classifiers. Individual error probability = 0.3  
Probability of voted ensemble error = 0.02639



# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil}^M \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$



Virtually ZERO error by  $M = 50 !!$

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

$$\varphi(E[X]) \leq E[\varphi(X)]$$

$$\phi\left(\sum_{i=1}^M d_i x_i\right) \leq \sum_{i=1}^M d_i \phi(x_i) \quad \sum_i d_i = 1$$

se...  $\phi(z) = (z-t)^2$

e  $d_i = \frac{1}{M}$  per tutti i valori di  $i$

→

$$\left(\sum_i d_i x_i - t\right)^2 \leq \sum_i d_i (x_i - t)^2$$

$$\left(\sum_i \frac{x_i}{M} - t\right)^2 \leq \sum_i \frac{(x_i - t)^2}{M}$$

$$\left(\frac{1}{M} \sum_i x_i - t\right)^2 \leq \frac{1}{M} \sum_i (x_i - t)^2$$

The squared error of a linear combination of predictions...

....is guaranteed to be less than or equal to...

...the average squared error of the individual predictions.

→

$$\left( \sum_i d_i x_i - t \right)^2 \leq \sum_i d_i (x_i - t)^2$$

$$\left( \sum_i \frac{x_i}{M} - t \right)^2 \leq \sum_i \frac{(x_i - t)^2}{M}$$

$$\left( \frac{1}{M} \sum_i x_i - t \right)^2 \leq \frac{1}{M} \sum_i (x_i - t)^2$$

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Training *different* classifiers . . .

but how "different" the single classifiers should be?

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Training *different* classifiers . . .

but how "different" the single classifiers should be?

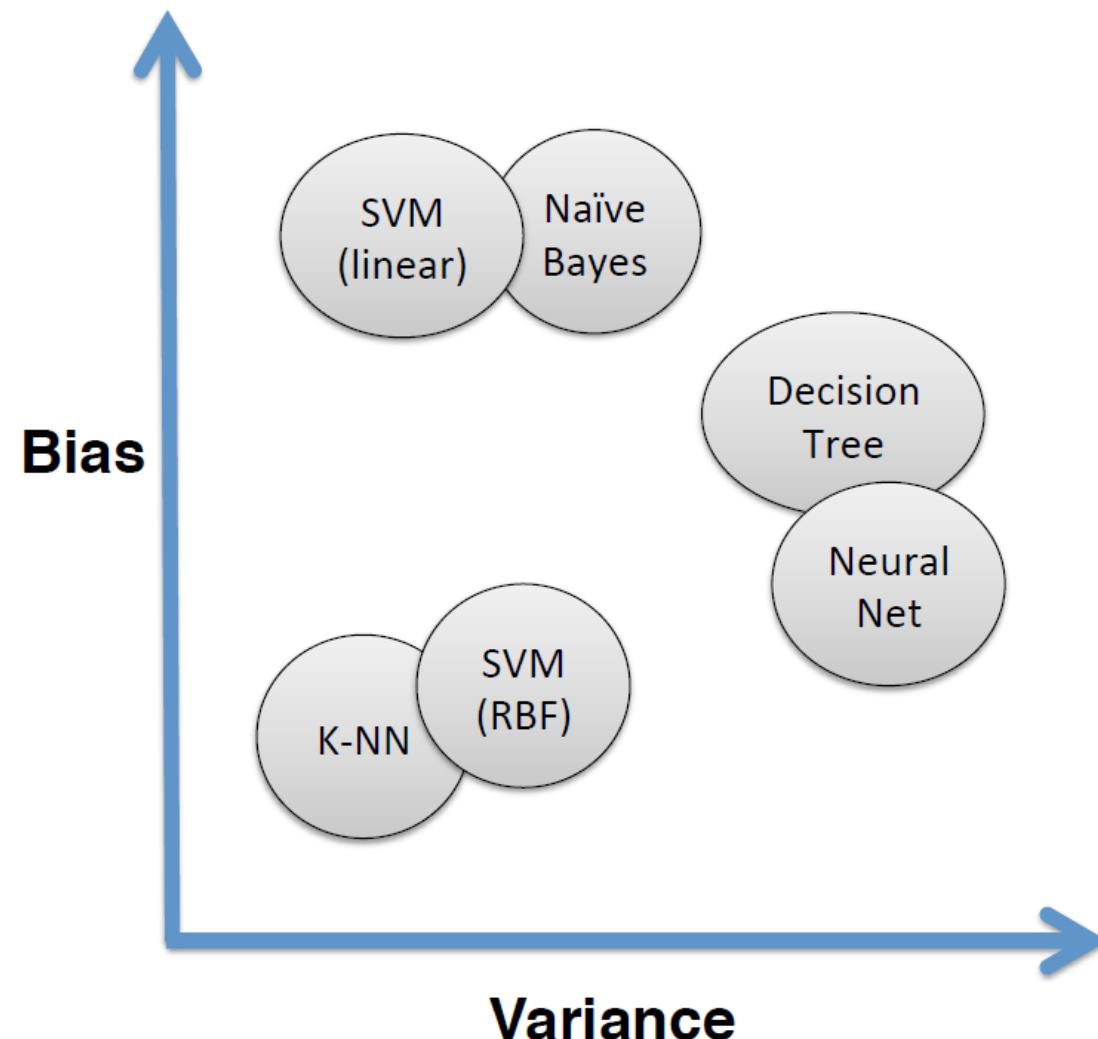
DIFFERENT  
(?)

ACCURATE  
(?)

INDEPENDENT  
(?)

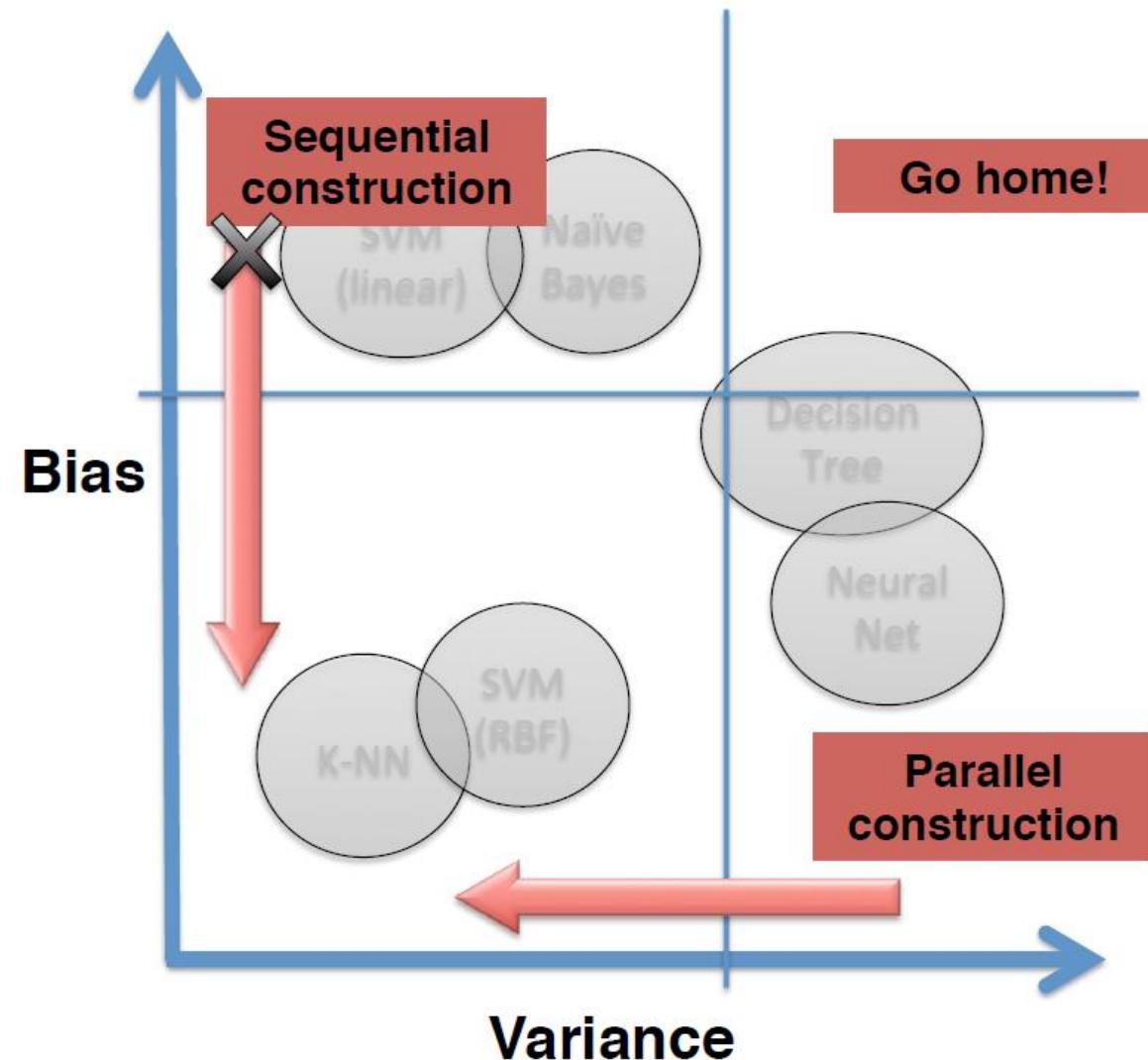
# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



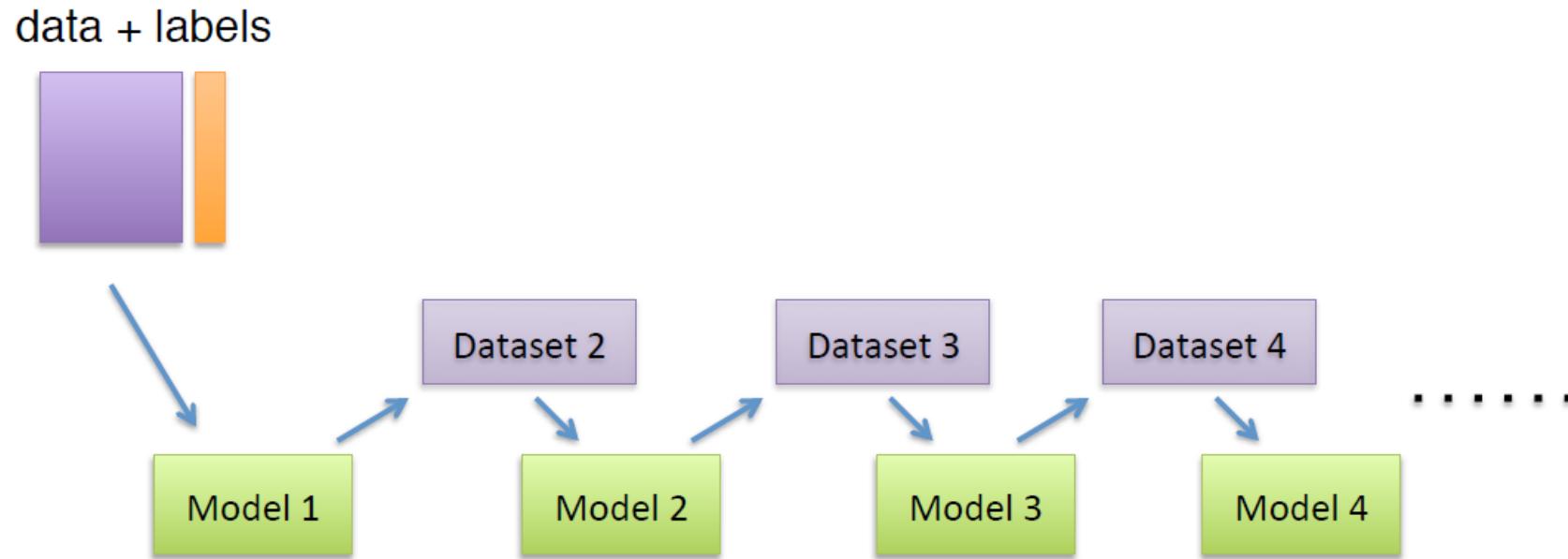
# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers

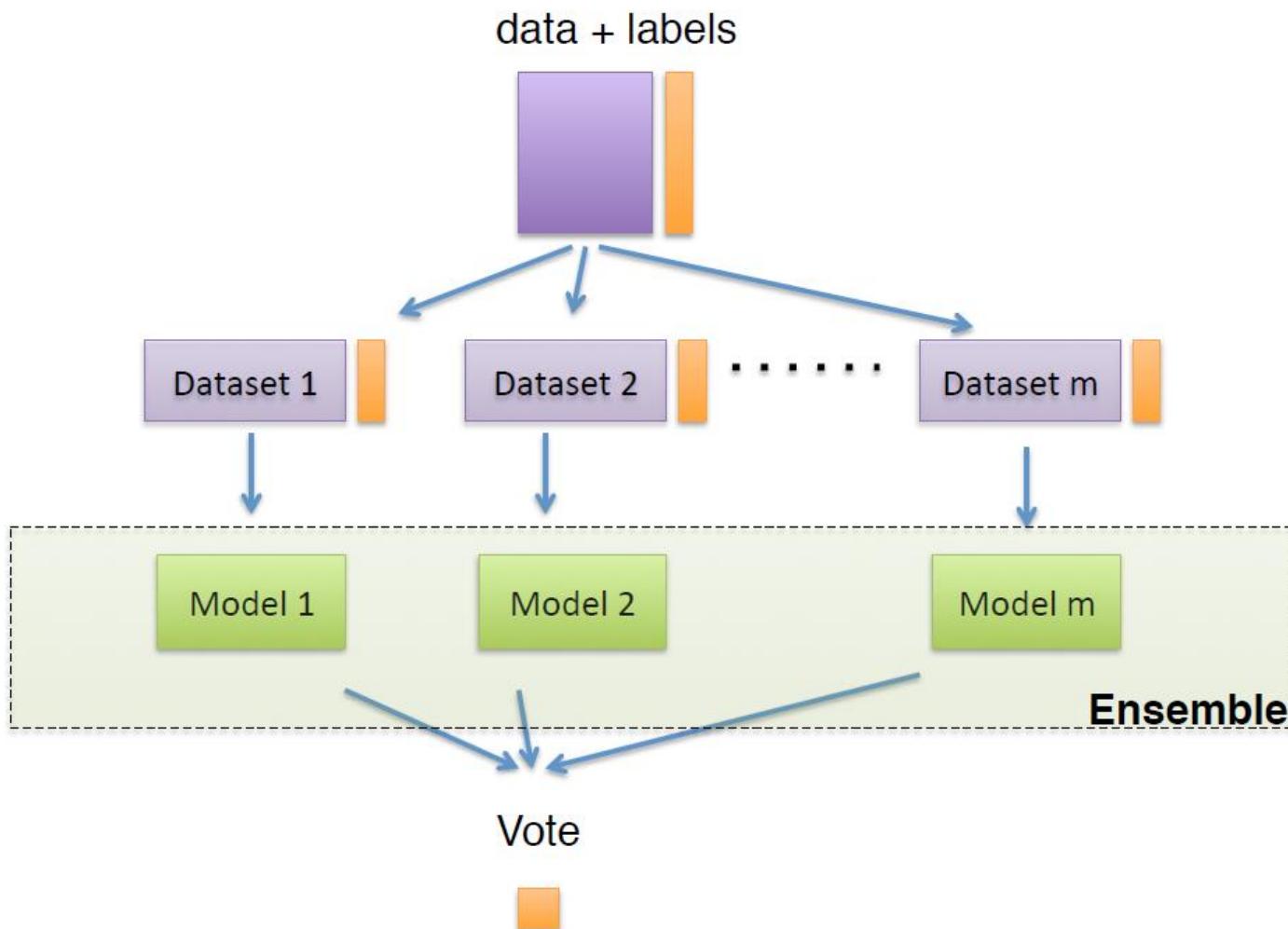
WITH SOME SLIDES FROM PROF. GAVIN BROWN



Each model corrects the mistakes of its predecessor.

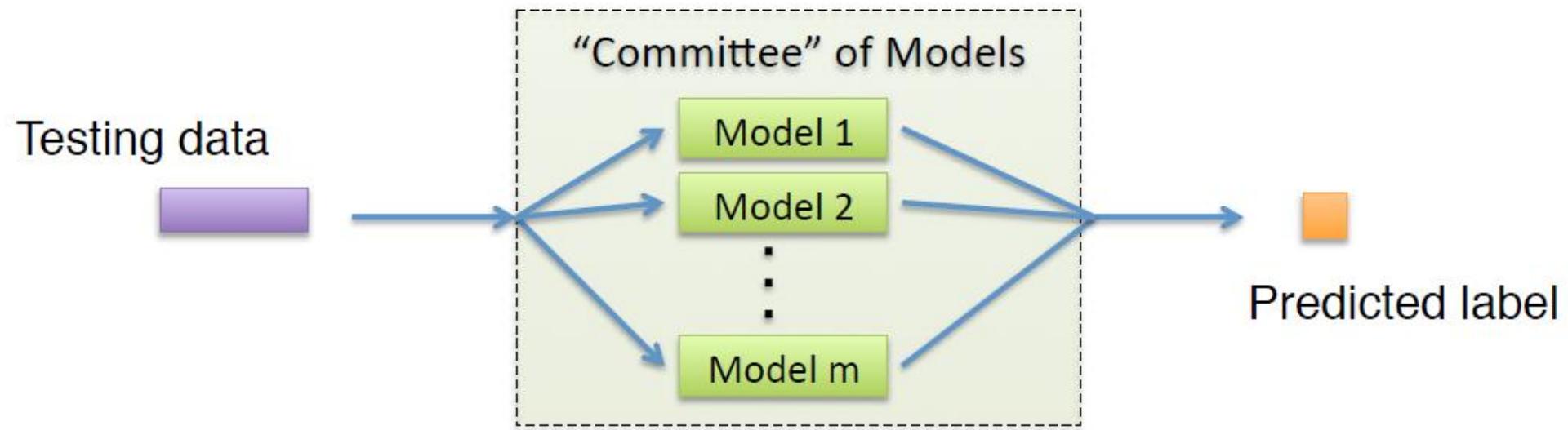
# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



Decisions of individuals combined.

At testing phase is the same...

Which are the most popular methods to build  
“ensemble” classifiers?

BAGGING

BOOSTING

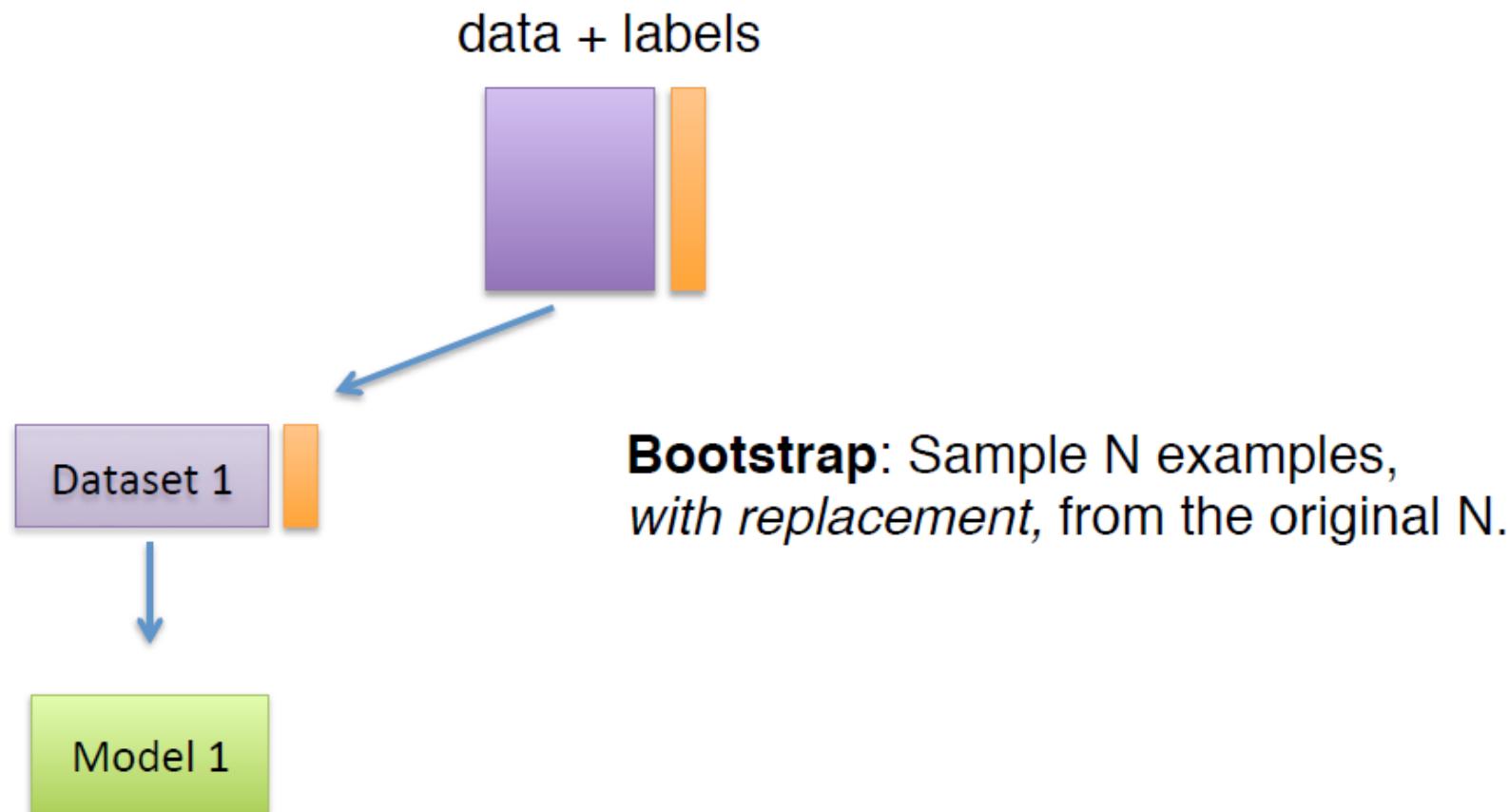
STACKING

BLENDING

# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

BAGGING = BOOTSTRAP AGGREGATING



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Original Dataset

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$
-------	-------	-------	-------	-------	-------	-------	-------	-------	----------

Bootstrap 1

$x_8$	$x_6$	$x_2$	$x_9$	$x_5$	$x_8$	$x_1$	$x_4$	$x_8$	$x_2$
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

$x_3$	$x_7$	$x_{10}$
-------	-------	----------

Bootstrap 2

$x_{10}$	$x_1$	$x_3$	$x_5$	$x_1$	$x_7$	$x_4$	$x_2$	$x_1$	$x_8$
----------	-------	-------	-------	-------	-------	-------	-------	-------	-------

$x_6$	$x_9$
-------	-------

Bootstrap 3

$x_6$	$x_5$	$x_4$	$x_1$	$x_2$	$x_4$	$x_2$	$x_6$	$x_9$	$x_2$
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

$x_3$	$x_7$	$x_8$	$x_{10}$
-------	-------	-------	----------

Training Sets

Test Sets



This work by Sebastian Raschka is licensed under a  
Creative Commons Attribution 4.0 International License.

# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Probability of including any given example in a bootstrap:

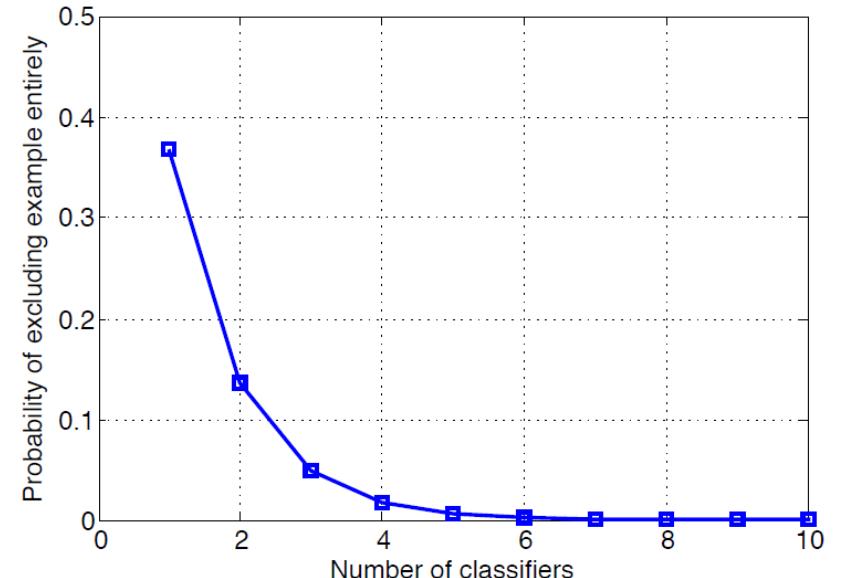
$$p = 1 - \left(1 - \frac{1}{N}\right)^N$$

$$\lim_{N \rightarrow \infty} \left\{ 1 - \left(1 - \frac{1}{N}\right)^N \right\} = 1 - e^{-1} \approx 0.6321$$

~63% of the data included.

... 37% left out!?

PROBABILITY OF EXCLUDING A SAMPLE  
FROM THE WHOLE ENSEMBLE



---

**Bagging** (input training data+labels  $T$ , number of models  $M$ )

---

**for**  $j = 1$  to  $M$  **do**

    Take a *bootstrap* sample  $T'$  from  $T$

    Build a model using  $T'$ .

    Add the model to the set.

**end for**

**return** set of models

---

For a test point  $\mathbf{x}$ , get a response from each model, and take a majority vote.

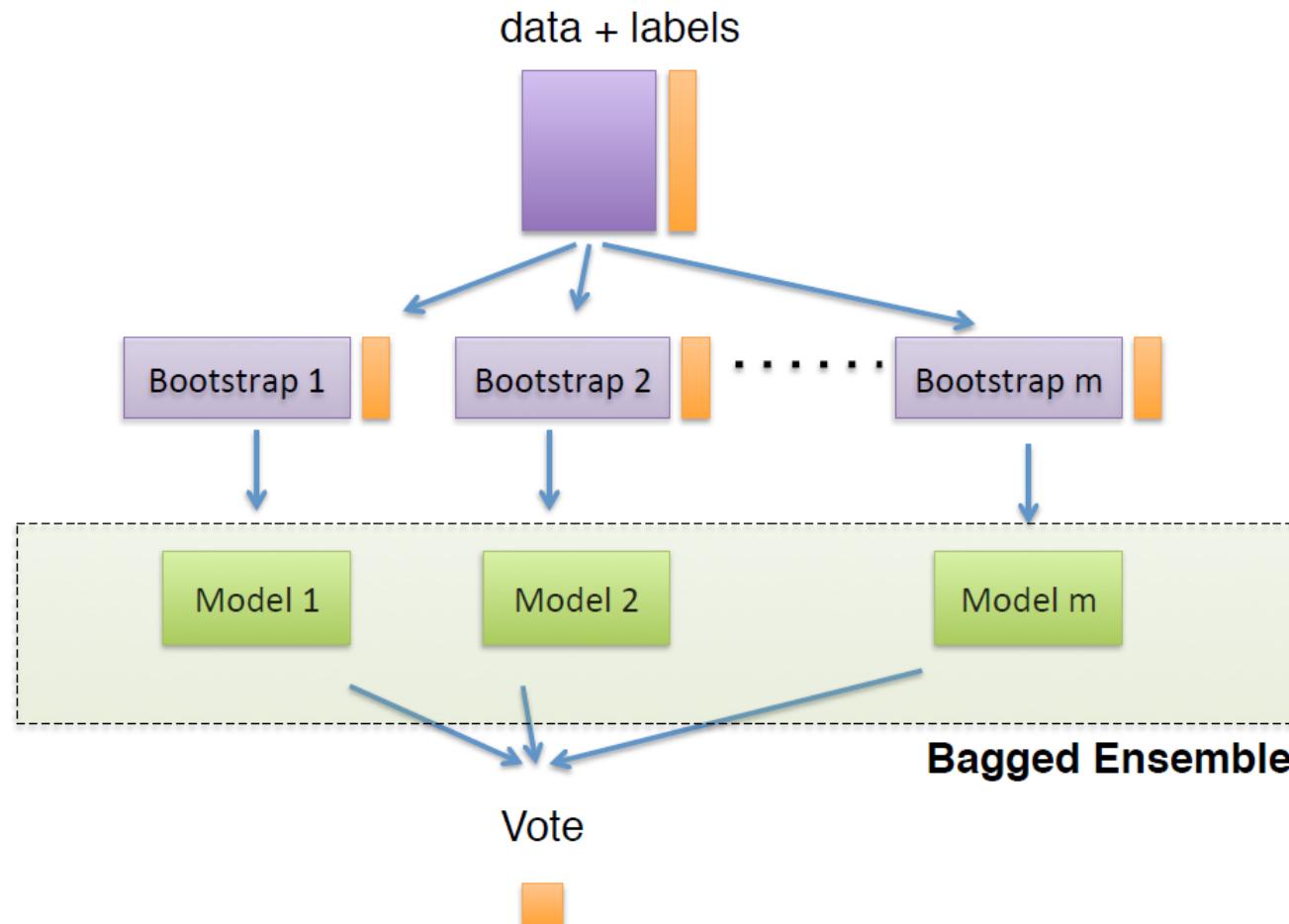
---

Breiman, 1996

(who will implement random forests 4 years later...)

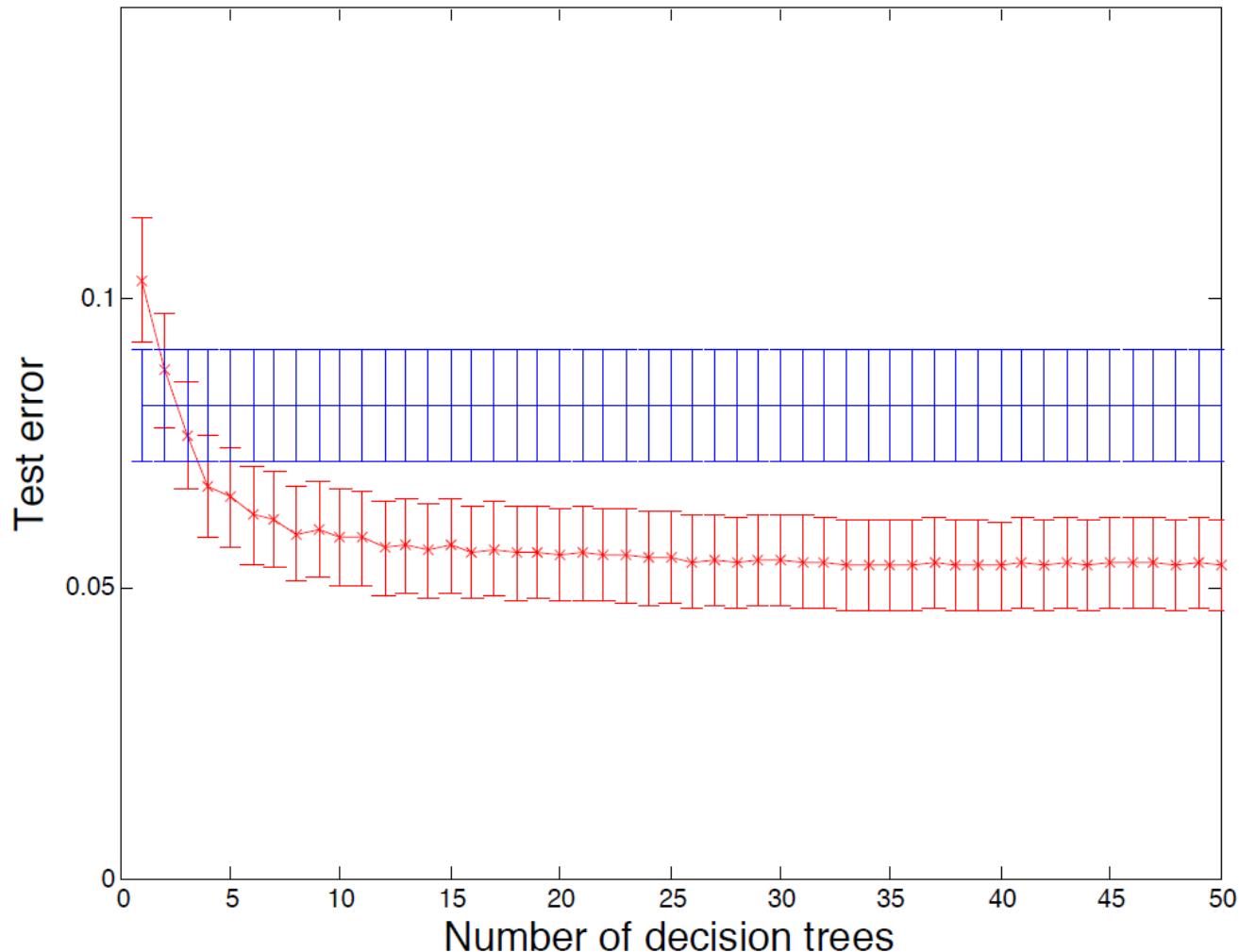
# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

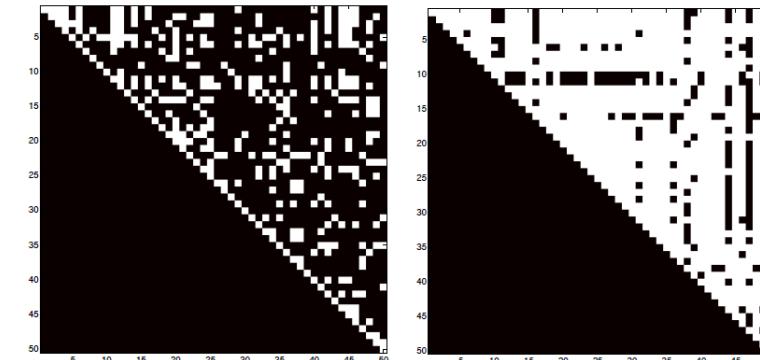
Training *different* classifiers . . .

but how “different” the single classifiers should be?

DIFFERENT  
(!)

ACCURATE  
(~)

INDEPENDENT  
(?)



Dependencies between 50 bagged classifiers – CART (left), Gaussian Naive Bayes (right).

$(\chi^2$  test,  $\alpha = 0.05)$

# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

A note on model stability

Some models are almost completely unaffected by bootstrapping (similar to each other on test data)

STABLE MODELS are not the first choice  
for parallel ensemble methods

Neural Networks

Logistic Regression

kNN

SVM

Decision Trees



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

## A note on model stability

Some models are almost completely unaffected by bootstrapping (similar to each other on test data)

STABLE MODELS are not the first choice  
for parallel ensemble methods

kNN  
SVM  
Decision Trees

Logistic Regression

Neural Networks



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

## A note on model stability

Some models are almost completely unaffected by bootstrapping (similar to each other on test data)

STABLE MODELS are not the first choice  
for parallel ensemble methods



## Stacking or Stacked Generalization

“Given multiple machine learning models that are skillful on a problem, but in different ways, how do you choose which model to use (trust)?”

Use a second-level (meta) machine-learning model that learns when to use (trust) each model in the ensemble

Architecture of a stacking model:

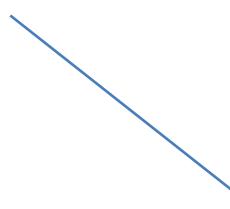
- two or more base models (often referred to as LEVEL-0 models), which learn to discriminate training data at a single-classifier level and whose predictions are computed and recorded (to be used at level 1)
- one meta-model (LEVEL 1) that combines the predictions of the base models (referred to as a LEVEL-1 model) by learning how to best combine LEVEL-0 models (or their predictions)

# Ensembles of Classifiers | Stacking

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Architecture of a stacking model:

- two or more base models (often referred to as LEVEL-0 models), which learn to discriminate training data at a single-classifier level and whose predictions are computed and recorded (to be used at level 1)
- one meta-model (LEVEL 1) that combines the predictions of the base models (referred to as a LEVEL-1 model) by learning how to best combine LEVEL-0 models (or their predictions)



SHOULD BE:

1. sufficiently accurate
2. Sufficiently different (assumptions about the predictive task)
3. sufficiently independent from each other (uncorrelated in their predictions)

# Ensembles of Classifiers | Stacking

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Architecture of a stacking model:

- two or more base models (often referred to as LEVEL-0 models), which learn to discriminate training data at a single-classifier level and whose predictions are computed and recorded (to be used at level 1)
- one meta-model (LEVEL 1) that combines the predictions of the base models (referred to as a LEVEL-1 model) by learning how to best combine LEVEL-0 models (or their predictions)

LEARN classifiers' combination  
through cross validation (or similar)

SHOULD BE:

1. sufficiently accurate
2. Sufficiently different (assumptions about the predictive task)
3. sufficiently independent from each other (uncorrelated in their predictions)

# Ensembles of Classifiers | Stacking

WITH SOME SLIDES FROM PROF. GAVIN BROWN



*“Blending is a word introduced by the Netflix winners. It is very close to stacked generalization, but a bit simpler and less risk of an information leak. [...] With blending, instead of creating out-of-fold predictions for the train set, you create a small holdout set of say 10% of the train set. The stacker model then trains on this holdout set only.*

— Kaggle Ensemble Guide, MLWave, 2015.

## The BellKor 2008 Solution to the Netflix Prize

Robert M. Bell  
AT&T Labs - Research  
Florham Park, NJ

Yehuda Koren  
Yahoo! Research  
Haifa, Israel

Chris Volinsky  
AT&T Labs - Research  
Florham Park, NJ

[BellKor@research.att.com](mailto:BellKor@research.att.com)

### 1. Introduction

Our RMSE=0.8643<sup>2</sup> solution is a linear blend of over 100 results. Some of them are new to this year, whereas many others belong to the set that was reported a year ago in our 2007 Progress Prize report [3]. This report is structured accordingly. In Section 2 we detail methods new to this year. In general, our view is that those newer methods deliver a superior performance compared to the methods we used a year ago. Throughout the description of the methods, we highlight the specific predictors that participated in the final blended solution. Nonetheless, the older methods still play a role in the blend, and

- **Blending:** Stacking-type ensemble where the meta-model is trained on predictions made on a holdout dataset.
- **Stacking:** Stacking-type ensemble where the meta-model is trained on out-of-fold predictions made during k-fold cross-validation.

# Ensembles of Classifiers | Boosting

WITH SOME SLIDES FROM PROF. GAVIN BROWN

---

Define a distribution over the training set,  $D_1(i) = \frac{1}{N}, \forall i.$

**for**  $t = 1$  to  $T$  **do**

    Build a model  $h_t$  from the training set, using distribution  $D_t$ .

    Update  $D_{t+1}$  from  $D_t$ :

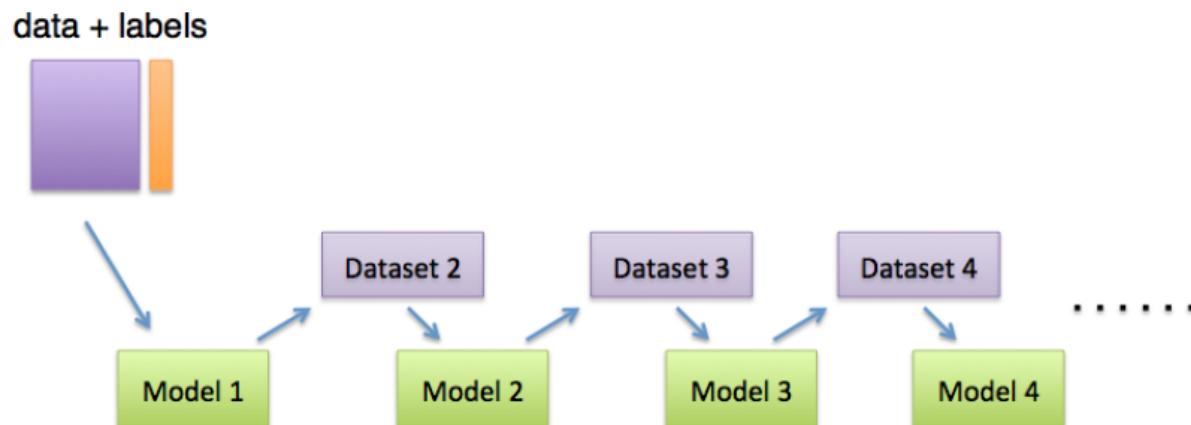
        Increase the weight on examples that  $h_t$  incorrectly classifies.

        Decrease the weight on examples that  $h_t$  correctly classifies.

**end for**

For a new testing point  $(\mathbf{x}', y')$ , we take a weighted majority vote from  $\{h_1, \dots, h_T\}$ .

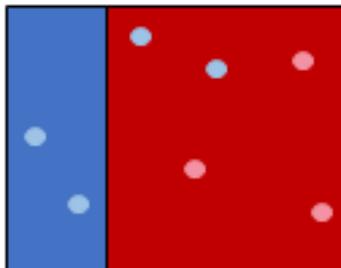
---



## ADABOOST

### AdaBoost:

- Combining **weak learners** (decision trees)
- Assigning **weights to incorrect values**
- **Sequential tree growing** considering past mistakes



Results of  
tree 1

## ADABOOST

### AdaBoost:

- Combining **weak learners** (decision trees)
- Assigning **weights to incorrect values**
- **Sequential tree growing** considering past mistakes

The diagram shows two decision trees being combined. On the left, a blue square labeled "Results of tree 1" contains three blue dots. To its right is a plus sign (+). To the right of the plus sign is another blue square labeled "Results of tree 2". This second square contains four blue dots in the top half and three pink dots in the bottom half. This visualizes how two weak learners are combined sequentially to form a stronger classifier.

## ADABOOST

### AdaBoost:

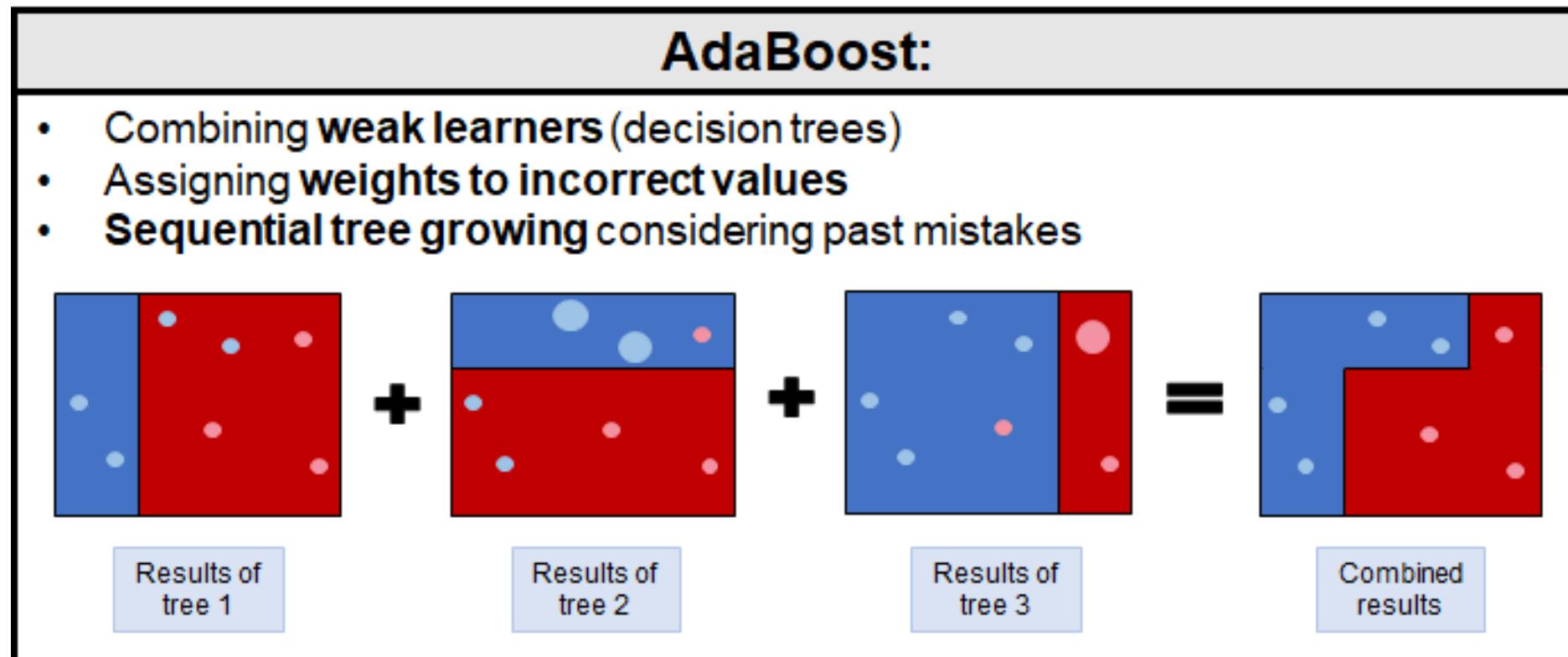
- Combining **weak learners** (decision trees)
- Assigning **weights to incorrect values**
- Sequential tree growing** considering past mistakes

The diagram illustrates the AdaBoost process through three stages of decision tree results:

- Results of tree 1:** A 2x3 grid where the left column is blue and the right column is red. Blue dots are in the top-left and bottom-left cells; red dots are in the top-right and bottom-right cells.
- Results of tree 2:** A 2x3 grid where the top row is blue and the bottom row is red. Blue dots are in the top-left and middle-left cells; red dots are in the top-right and bottom-right cells.
- Results of tree 3:** A 2x3 grid where the left column is blue and the right column is red. Blue dots are in the top-left, middle-left, and bottom-left cells; red dots are in the top-right and bottom-right cells.

Plus signs between the stages indicate the sequential addition of each tree's results to form the final ensemble classifier.

## ADABOOST



## ADABOOST

Define a distribution over the training set,  $D_1(i) = \frac{1}{N}, \forall i.$

**for**  $t = 1$  to  $T$  **do**

    Build a classifier  $h_t$  from the training set, using distribution  $D_t$ .

    Set  $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$

    Update  $D_{t+1}$  from  $D_t$  :

        Set  $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$

**end for**

$$H(x') = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x') \right)$$

# Ensembles of Classifiers | Boosting

WITH SOME SLIDES FROM PROF. GAVIN BROWN

## ADABOOST

Define a  
**for**  $t = 1$

### Weight Update

Build a classifier  $h_t$  from the tra

$$\text{Set } \alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$$

Update  $D_{t+1}$  from  $D_t$  :

$$\text{Set } D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

**end for**

$$H(x') = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x') \right)$$

$$E_1 = \frac{1}{N} \sum_{i=1}^N e^{-y_i h_1(x_i)}$$

$$E_2 = \underbrace{\sum_{i=1}^N \frac{1}{N} e^{-y_i \alpha_1 h_1(x_i)} e^{-y_i \alpha_2 h_2(x_i)}}_{\text{constant}}$$

$$E_2 = \sum_{i=1}^N w_2(i) e^{-y_i \alpha_2 h_2(x_i)}$$

# Ensembles of Classifiers | Boosting

WITH SOME SLIDES FROM PROF. GAVIN BROWN

## ADABOOST

Define a  
**for**  $t = 1$

### Weight Update

Build a classifier  $h_t$  from the tra

$$\text{Set } \alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$$

Update  $D_{t+1}$  from  $D_t$  :

$$\text{where } Z_t = \sum_i D_t(i) e^{-y_i \alpha_t h_t(x_i)}$$

$$H(x') = sign \left( \sum_{t=1}^T \alpha_t h_t(x') \right)$$

$$E_1 = \frac{1}{N} \sum_{i=1}^N e^{-y_i h_1(x_i)}$$

$$E_2 = \underbrace{\sum_{i=1}^N \frac{1}{N} e^{-y_i \alpha_1 h_1(x_i)} e^{-y_i \alpha_2 h_2(x_i)}}_{\text{constant}}$$

$$E_2 = \sum_{i=1}^N w_2(i) e^{-y_i \alpha_2 h_2(x_i)}$$

## ADABOOST

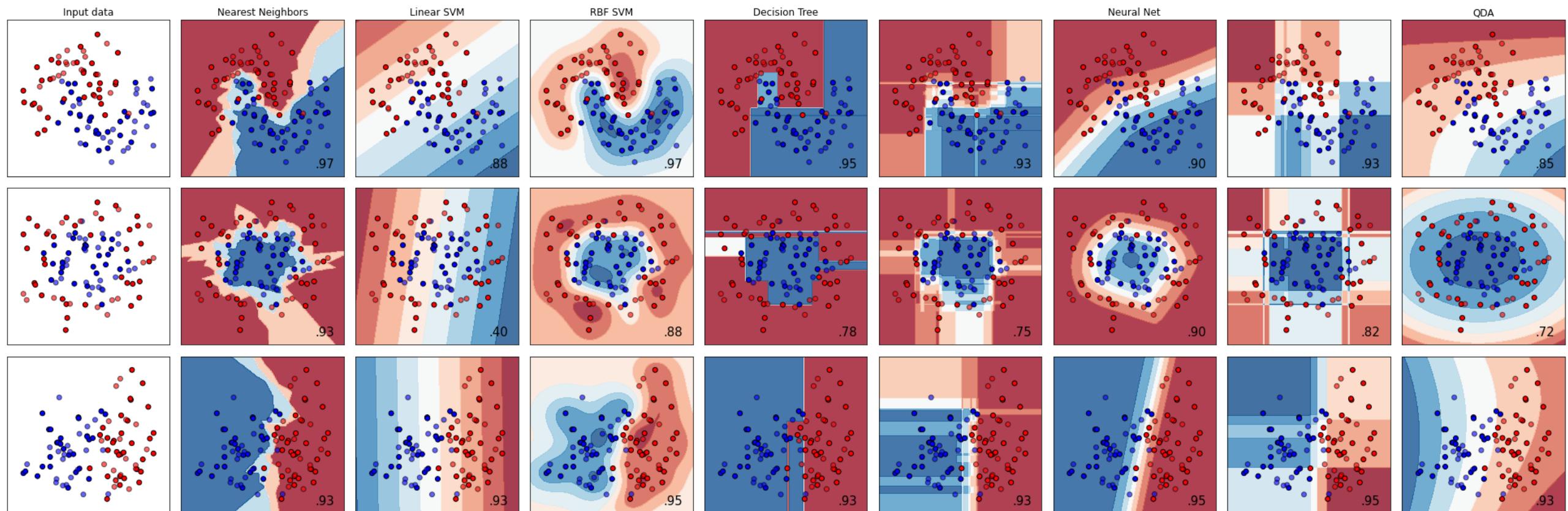
Good performance with non-noisy datasets

Controversial results with noisy data

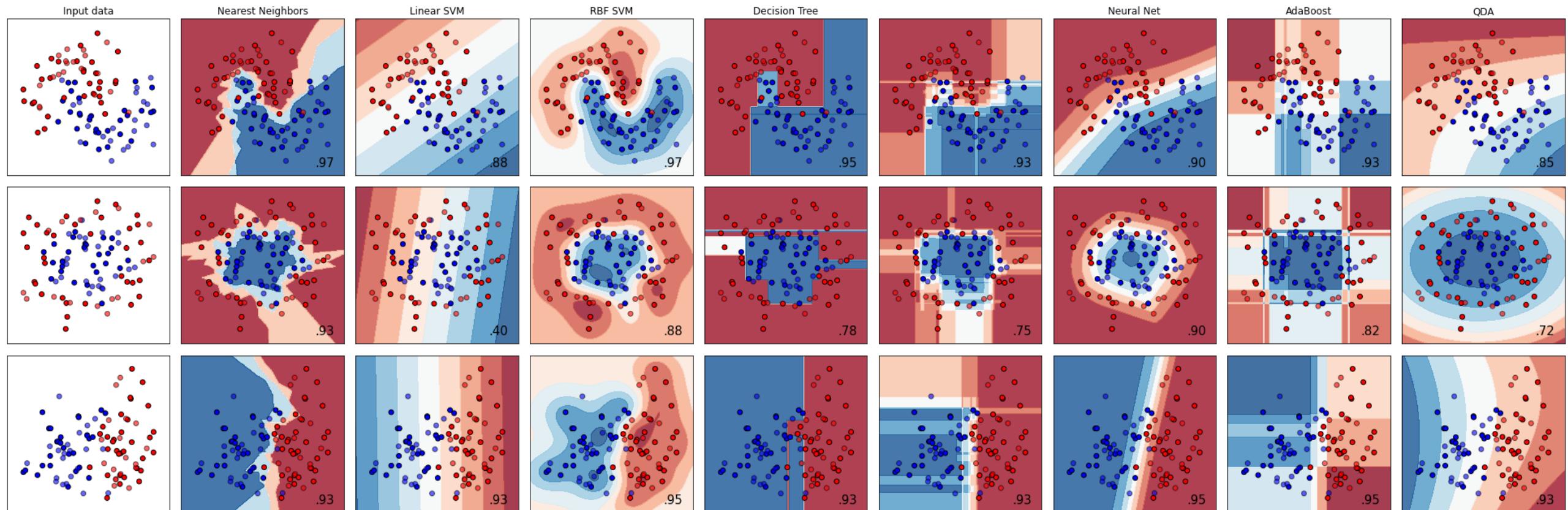
Low number of hyperparameters

Not optimized for computationa costs in terms of time

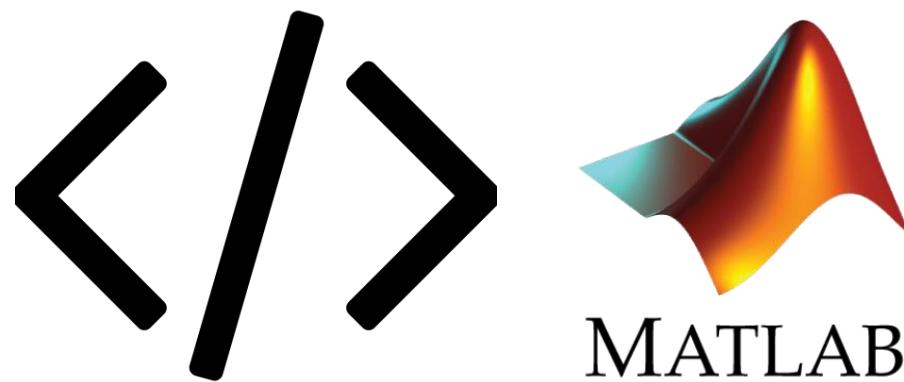
# Which is which | Decision Function



# Which is which | Decision Function



# Support Vector Machine



## To Ensemble or Not Ensemble: When does End-To-End Training Fail?

Andrew Webb<sup>1</sup>, Charles Reynolds<sup>1</sup>, Wenlin Chen<sup>1</sup>, Henry Reeve<sup>2</sup>, Dan Iliescu<sup>3</sup>,  
Mikel Luján<sup>1</sup>, and Gavin Brown<sup>1</sup>

<sup>1</sup> University of Manchester, UK

<sup>2</sup> University of Bristol, UK

<sup>3</sup> University of Cambridge, UK

**Abstract.** End-to-End training (E2E) is becoming more and more popular to train complex Deep Network architectures. An interesting question is whether this trend will continue—are there any clear failure cases for E2E training? We study this question in depth, for the specific case of E2E training an *ensemble* of networks. Our strategy is to blend the gradient smoothly in between two extremes: from independent training of the networks, up to to full E2E training. We find clear failure cases, where overparameterized models *cannot be trained E2E*. A surprising result is that the optimum can sometimes lie in between the two, neither an ensemble or an E2E system. The work also uncovers links to Dropout, and raises questions around the nature of ensemble diversity and multi-branch networks.

# CLASSIFICATION

## RANDOM FOREST

# Random Forest

Ideato da Leo Breiman nel 2001, è un classificatore multiplo in cui i singoli classificatori sono classification tree (alberi di classificazione) (estimatori)

- Genera alberi multipli ma a partire sempre dallo stesso training set
- Per la crescita dell'albero si sceglie ad ogni livello la feature che meglio separa le classi (foglie) e si determina la soglia di suddivisione
- Per la classificazione di un nuovo pattern (test set) si visita l'albero, e una volta giunti a una foglia, si classifica il pattern sulla base della classe più comune nel nodo tra i pattern del training set
- La decisione avviene mediante fusione dei classificatori (e.g. majority vote rule)

# Random Forest | Bagging

- Per ogni nodo la scelta della feature migliore su cui partizionare non è fatta sull'intero insieme delle d feature (dimensionalità dei pattern), ma su un sottoinsieme random di d' feature:

Valore tipico  $d' = \sqrt{d}$

-In assenza di questo accorgimento (noto anche come feature bagging) molti alberi sceglierrebbero con elevata probabilità le stesse variabili (quelle più discriminanti).

-Allo stesso tempo anche il training non è fatto sull'intero insieme degli n pattern, ma su un sottoinsieme random di n' pattern.

Valore tipico  $n' = 2/3 n$

Bagging = Bootstrap AGGREGatING

# Random Forest | Bagging

- Random Forest opera simultaneamente due tipi di bagging: uno sui pattern del training set e uno sulle features.
- Le prestazioni possono essere stimate con validation set separato (k-fold cross validation) o, grazie al bagging, con tecnica Out-Of-Bag (OOB) che non richiede validation set separato. Infatti ciascun pattern x può essere utilizzato per stimare le prestazioni a partire dai soli tree nel cui training x non è stato coinvolto.

# Random Forest | Importance

Mean decrease in the Gini index.

Misura analoga a quella usata negli alberi di decisione: per ogni albero e per ogni variabile  $i$ , si valuta ogni nodo in cui lo split coinvolge tale variabile, calcolando la differenza dell'indice di Gini (impurità) prima e dopo la suddivisione (nodo madre e nodi figli), pesandola per il numero di campioni nel nodo. Quindi si fa la media sugli alberi.

Più alti i valori, maggiore è la rilevanza della feature.

# Random Forest

---

**Random Forests** (input training data+labels  $T$ , number of trees  $M$ )

---

**for**  $j = 1$  to  $M$  **do**

    Take a bootstrap sample  $T'$  from  $T$

    Build a decision tree using  $T'$ , but, at every split point:

- Choose a random fraction  $K$  of the remaining features,
- Pick the best feature (minimising cost) from that subset.

        Add the tree to the set, *without pruning*

**end for**

**return** set of trees

For a test point  $\mathbf{x}$ , get a response from each tree, and take a majority vote.

---

PROF. GAVIN BROWN

# Random Forest | Vantaggi / Svantaggi

- Migliora le prestazioni di un singolo albero di decisione appreso su tutti i dati
- Perde la facile interpretabilità degli alberi e scala meno bene
- Aumenta in maniera contenuta la complessità temporale rispetto agli alberi, dato che ogni singolo albero è appreso solo su un sottoinsieme dei dati e a ogni split solo un sottoinsieme dei predittori viene preso in considerazione
- Può gestire dati anche di grande taglia prima di offrire rallentamenti evidenti
- Non presenta problemi di overfitting all'aumentare del numero di alberi. Questo è dovuto al fatto che solo una piccola porzione dei predittori è usata ad ogni split, oltre al fatto che RF predice mediante Aggregazione
- È più stabile alle variazioni dei dati di input, grazie al bagging
- Gli alberi appresi sono indipendenti
- Questo rende il metodo facilmente parallelizzabile con hardware appropriato (e.g. più core o processori)

# Random Forest vs. (simple) Bagging

*Bagging*      *vs.*      *Random Forest*

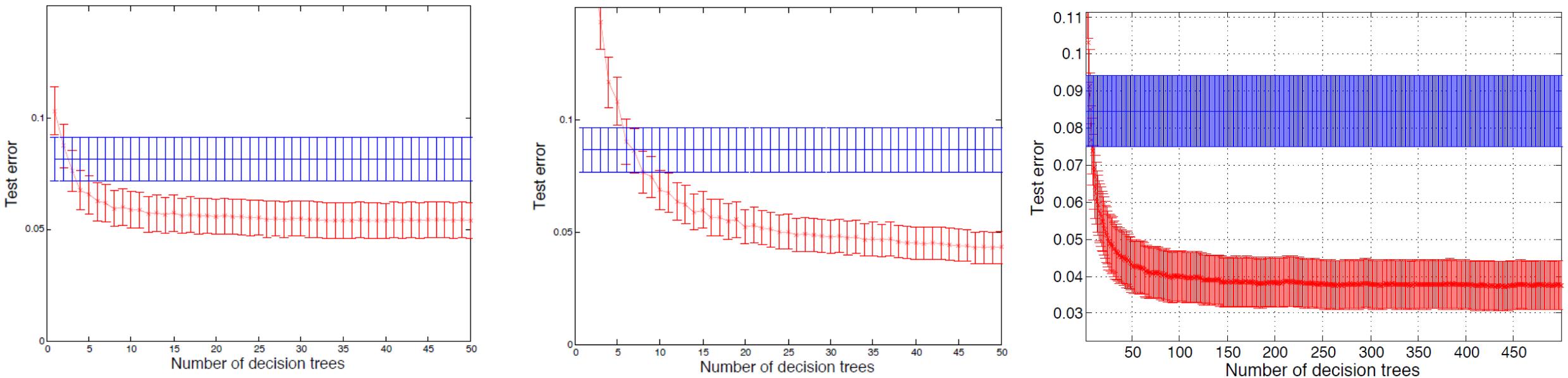
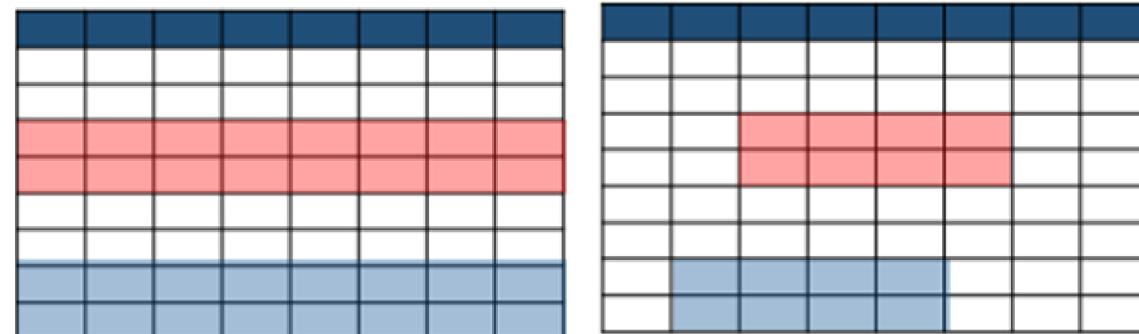
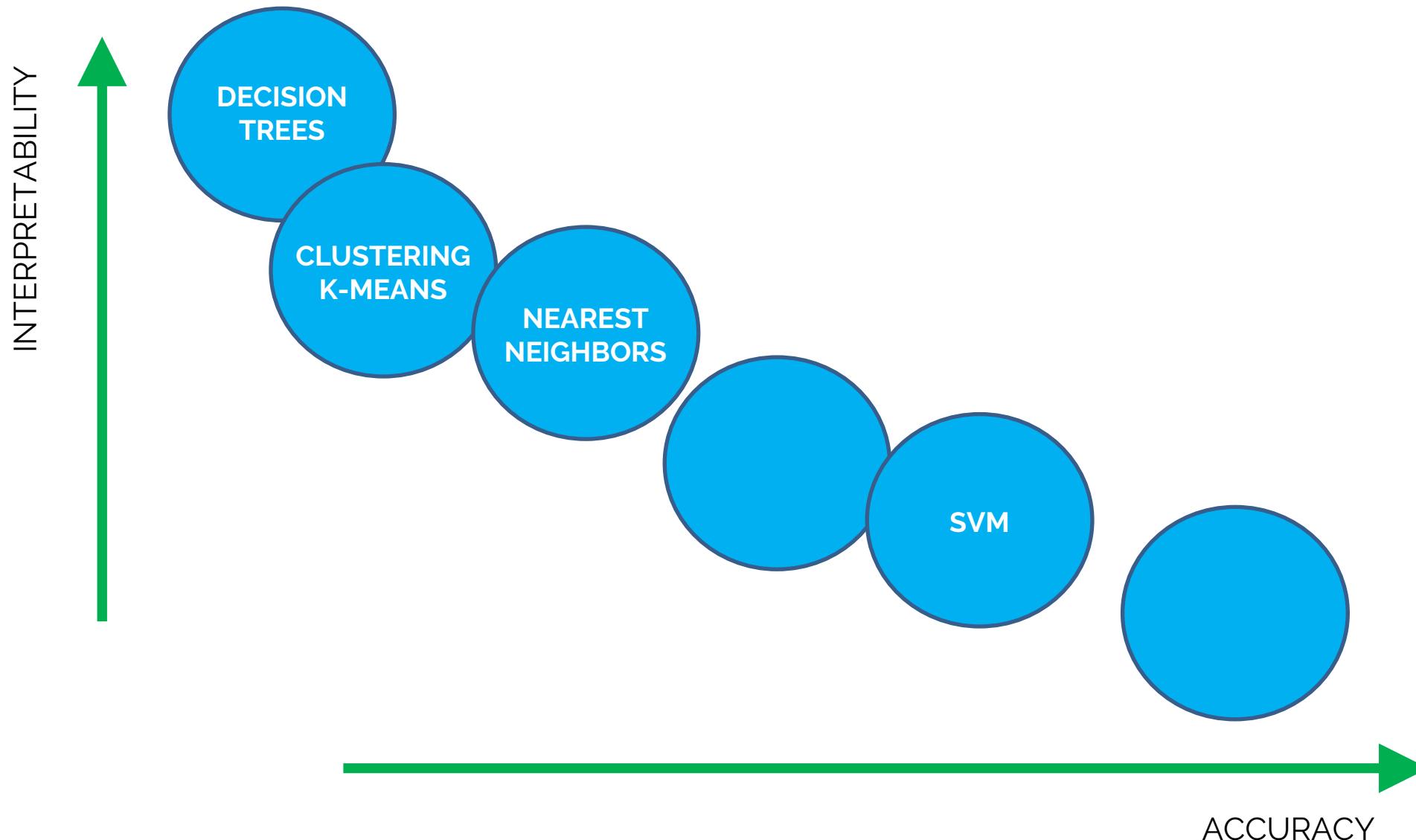
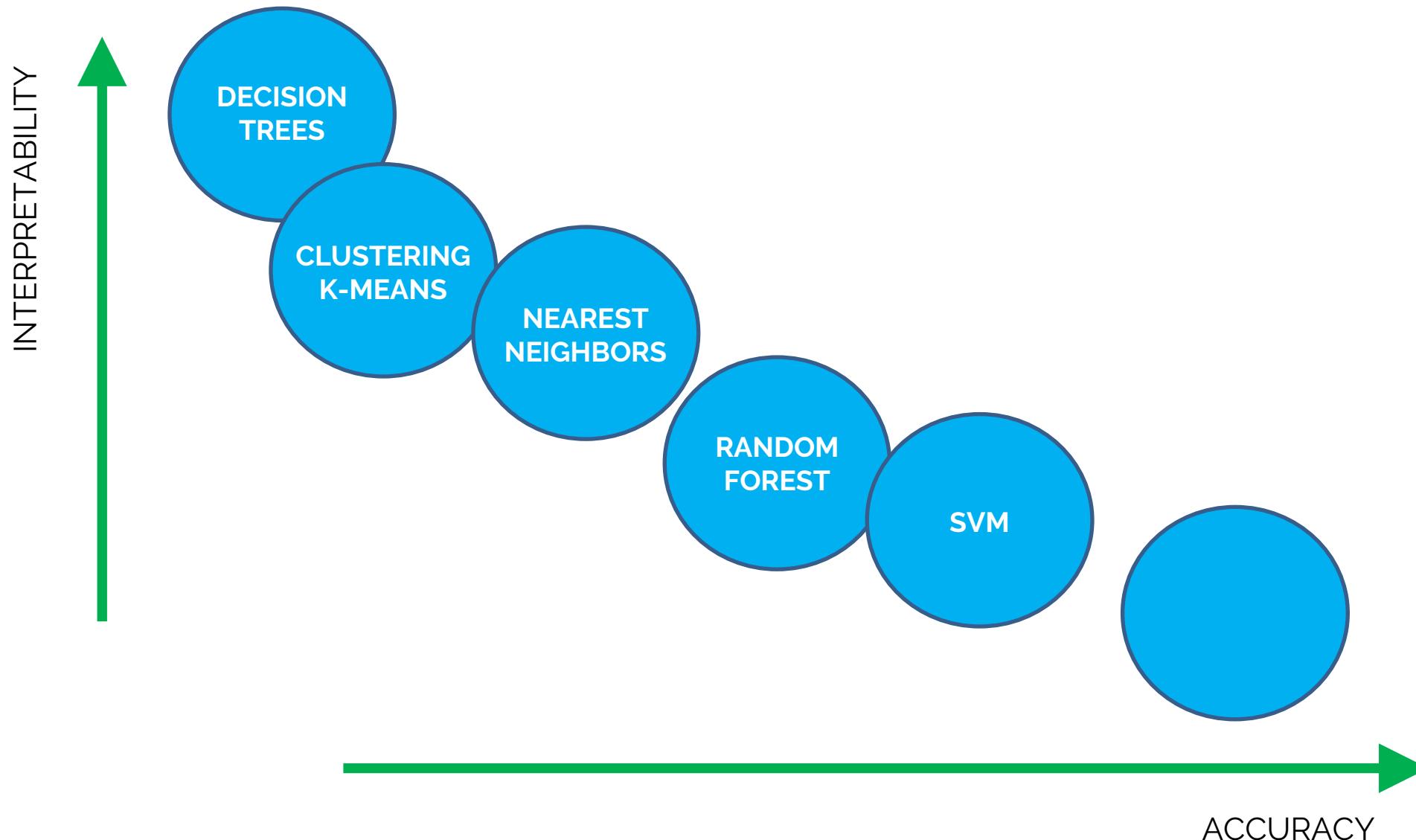


Figure 13: Bagging (LEFT) vs Random Forests (RIGHT) on the Splice dataset.

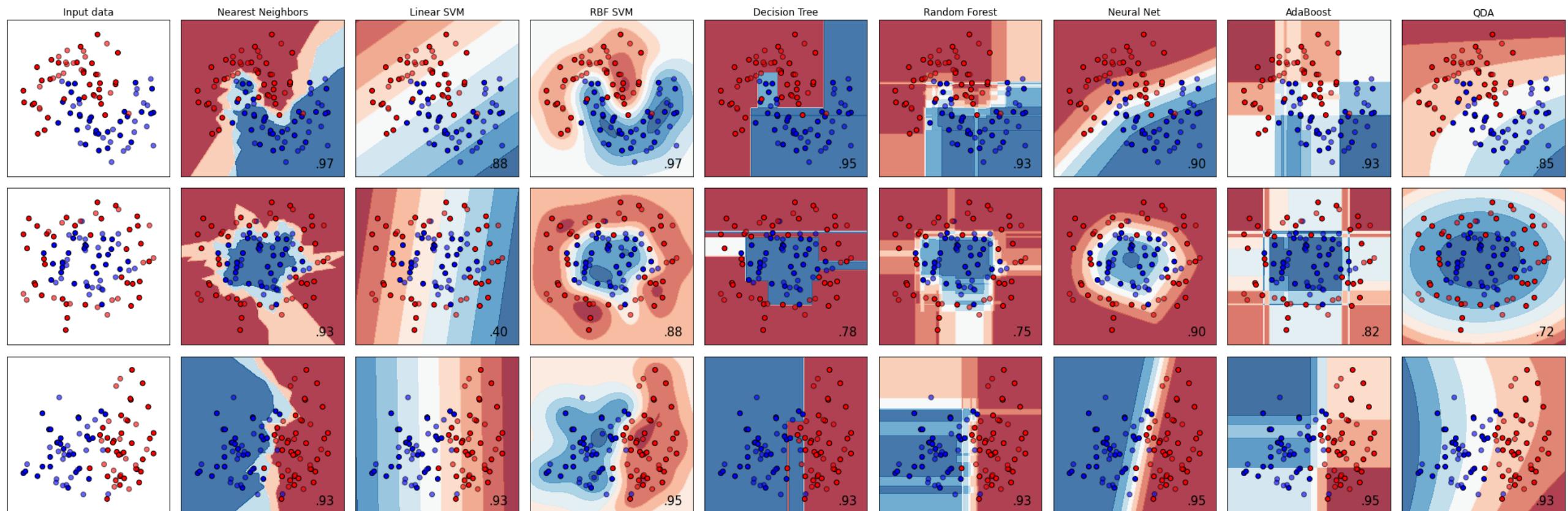
# Interpretability-Accuracy TRADEOFF



# Interpretability-Accuracy TRADEOFF



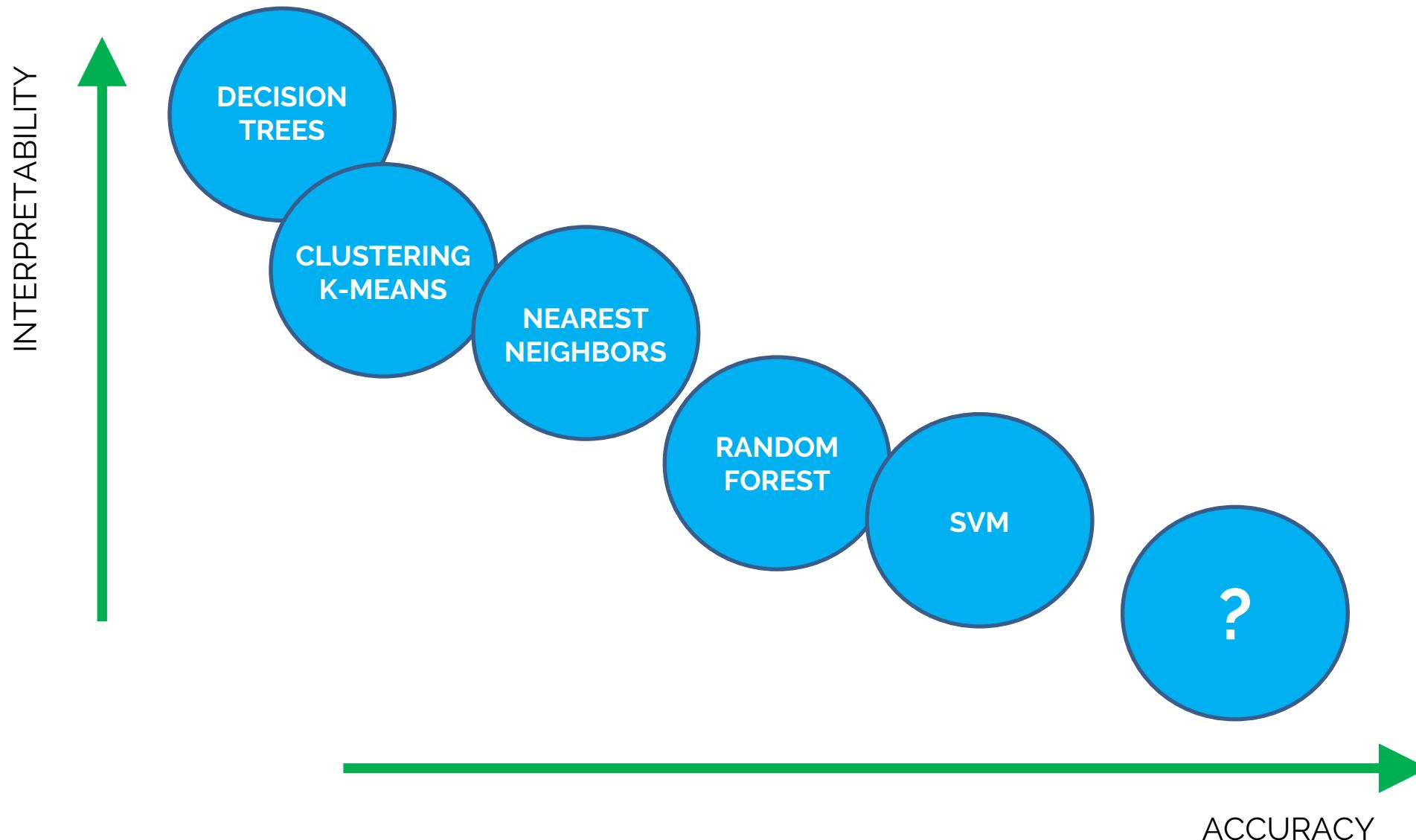
# Which is which | Decision Function



# Random Forest



# Interpretability-Accuracy TRADEOFF



# VALIDATION & PERFORMANCE EVALUATION

# How to test a machine-learning classifier?

A good validation process allows to obtain a **minimally biased estimate** of the true diagnostic performance of the classifier



- Correct quantification of the discriminatory power of a given model (model evaluation)
- Possibility to compare classification techniques based on different approaches (model selection)

# How to test a machine-learning classifier?

For example, if parameter selection, training of the predictive model and validation are performed using the same dataset, the generated classifier will show limited generalization ability when classifying unseen samples



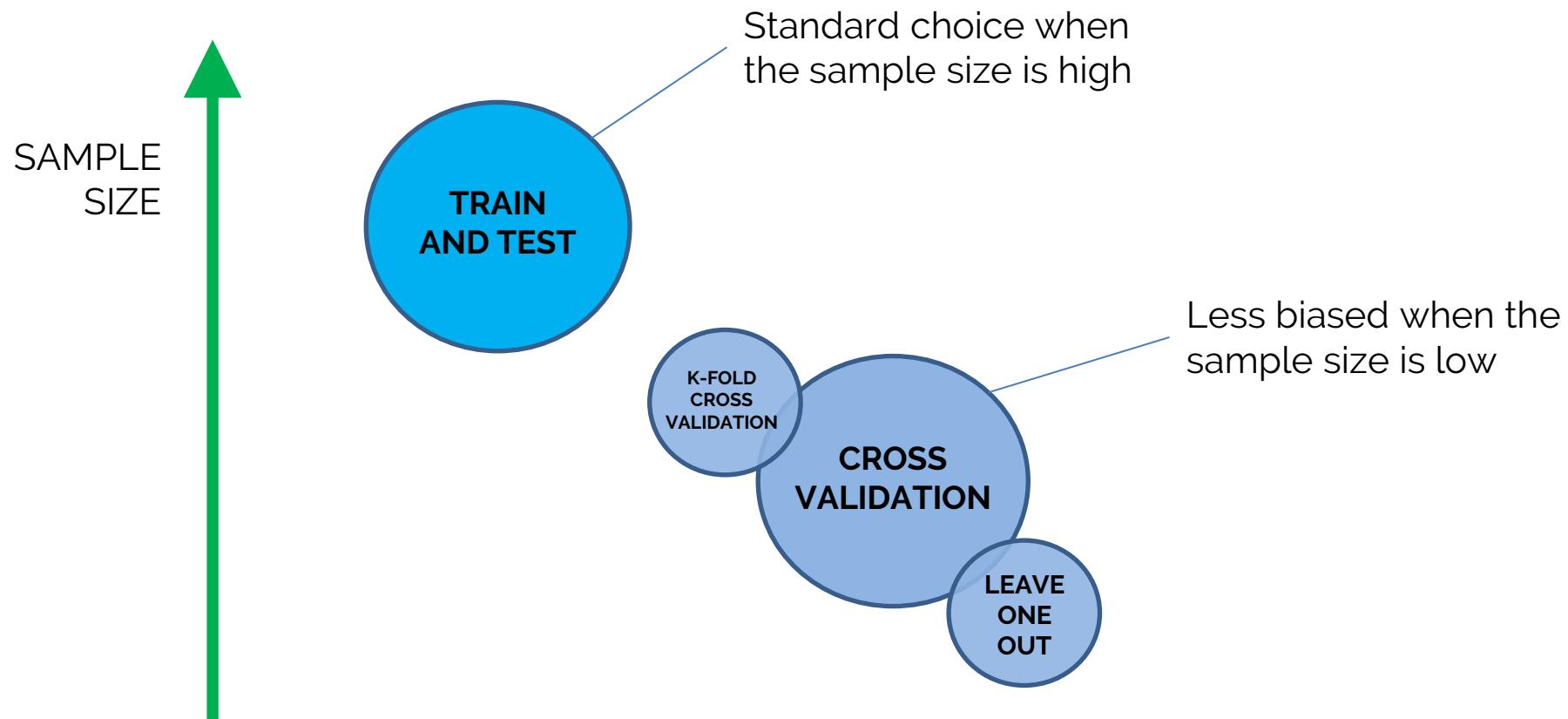
LOW  
TRAINING ERROR



HIGH  
TESTING ERROR  
(low generalization  
ability)

OVERFITTING

# Which validation approach?

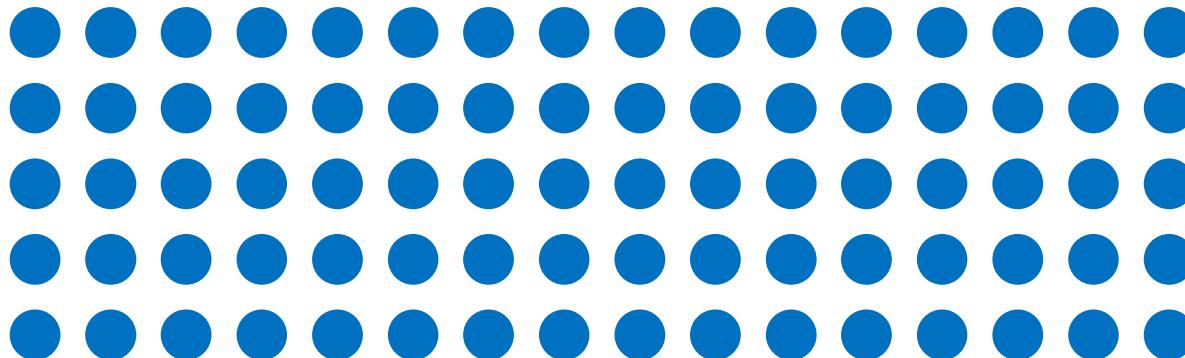


## Train-and-test

This kind of procedure is used when the number of samples in the original dataset is high enough to allow its splitting into two subsets including different samples, which can be used to train and test the classifier.

# Train-and-test

This kind of procedure is used when the number of samples in the original dataset is high enough to allow its splitting into two subsets including different samples, which can be used to train and test the classifier.

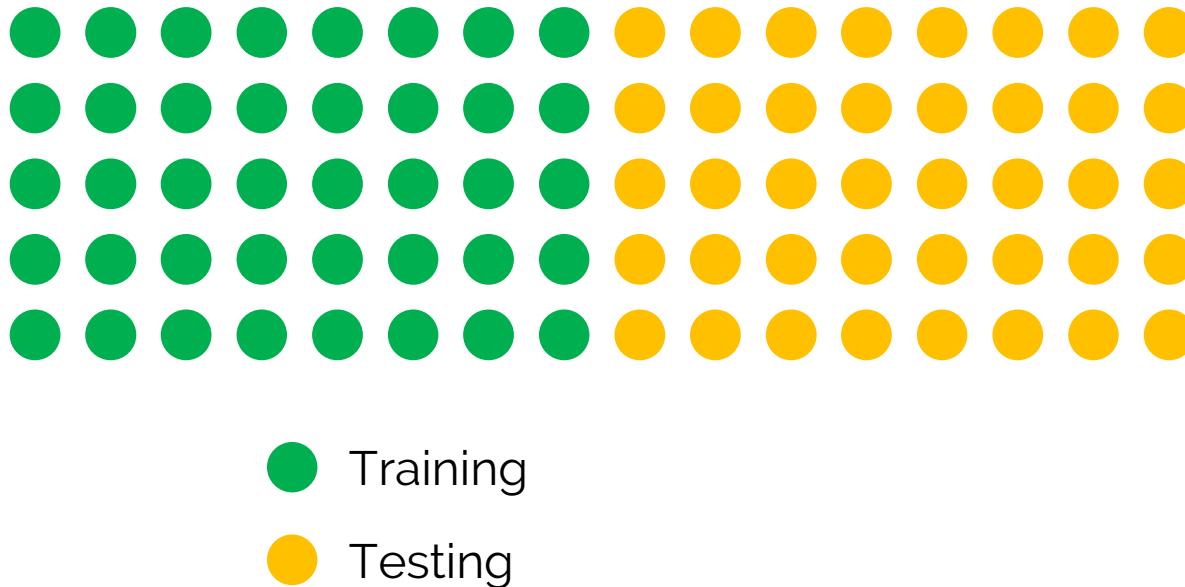


The original dataset is partitioned into 2 complementary subsets, the TRAINING set and the TESTING set.

The TRAINING set is used to train the classifier

The TESTING set is used for validation

# Train-and-test



# Train-and-test

Advantages:

- Over-training problems are reduced, because the training and testing sets are completely independent

Drawbacks:

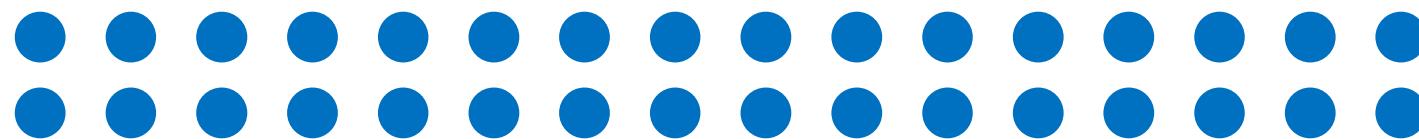
- Results could be related to the particular choice of the partition subsets

# Leave-one-out cross validation

Leave-One-Out (LOO) CV can be considered a particular form of k-fold CV in which

$k = \text{number of samples in the original dataset}$

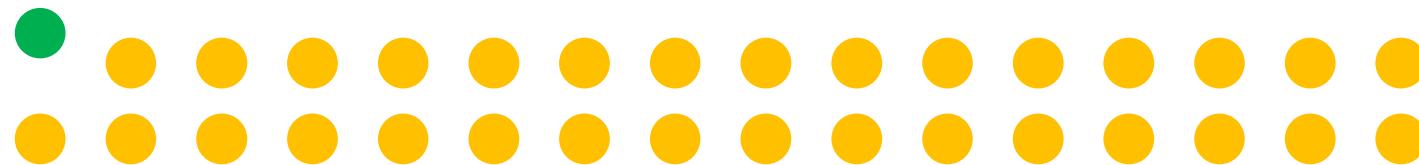
# Leave-one-out cross validation



TRAINING of the classifier is performed using  $n-1$  samples of the original dataset

TESTING is performed using the remaining sample ( $n$  being the total number of samples in the original dataset)

# Leave-one-out cross validation

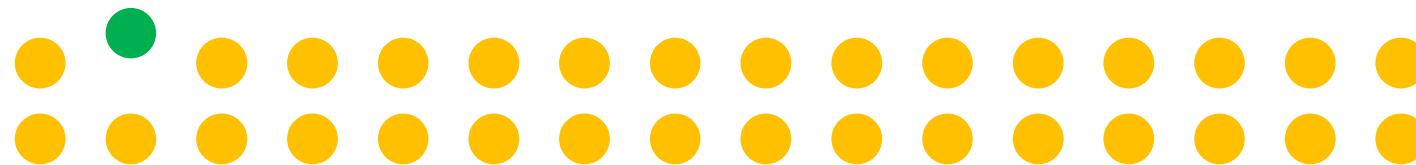


● Training

● Testing

The procedure is then repeated  $n$  times, until all samples are used once for validation.

# Leave-one-out cross validation

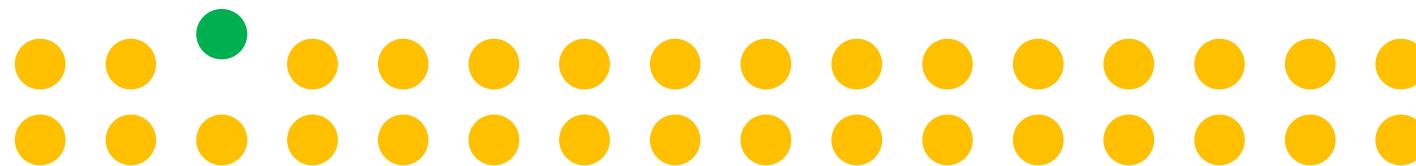


● Training

● Testing

The procedure is then repeated  $n$  times, until all samples are used once for validation.

# Leave-one-out cross validation

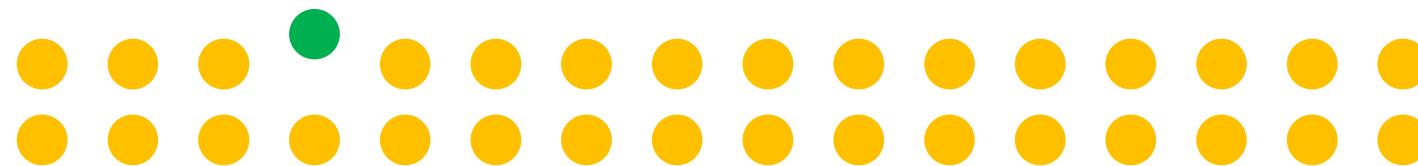


● Training

● Testing

The procedure is then repeated  $n$  times, until all samples are used once for validation.

# Leave-one-out cross validation



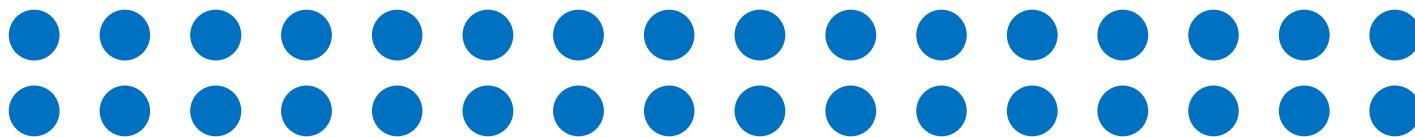
● Training

● Testing

and so on...

# Cross validation

Quantification of the discriminatory power of a predictive model even if the size of the dataset is small



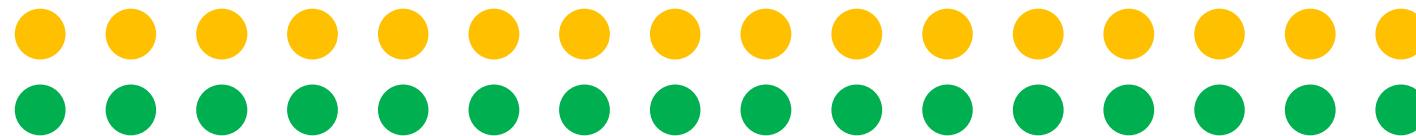
CV involves partitioning the original dataset into complementary subsets, the training set and the testing set

The TRAINING set is used to train the classifier

The TESTING set is used to validate the generated predictive model

# Cross validation

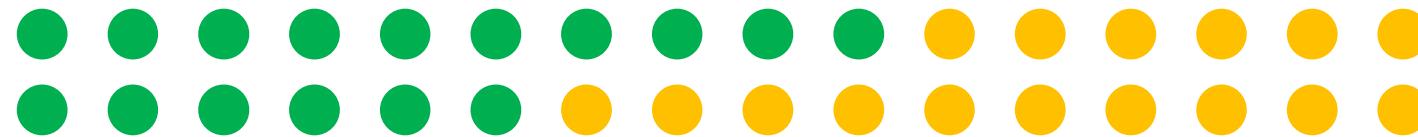
- Training
- Testing



By using different partitions of the original dataset, multiple rounds of CV can be performed, which can aid reducing the variability of the partitioned subsets.

# Cross validation

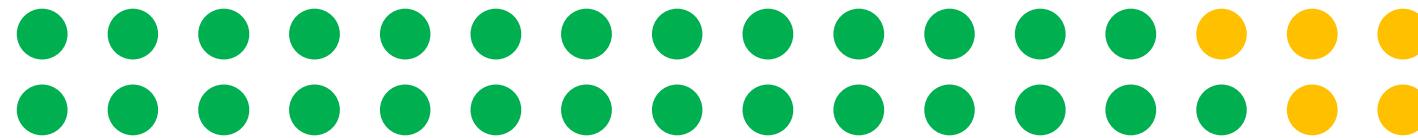
- Training
- Testing



By using different partitions of the original dataset, multiple rounds of CV can be performed, which can aid reducing the variability of the partitioned subsets.

# Cross validation

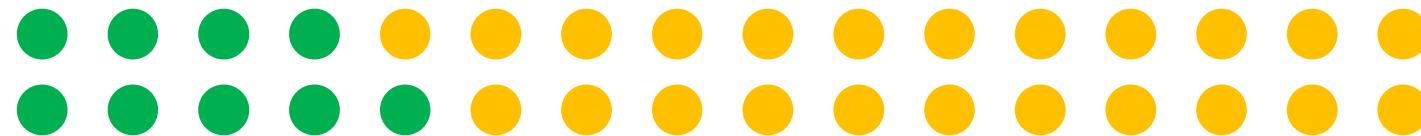
- Training
- Testing



By using different partitions of the original dataset, multiple rounds of CV can be performed, which can aid reducing the variability of the partitioned subsets.

# Cross validation

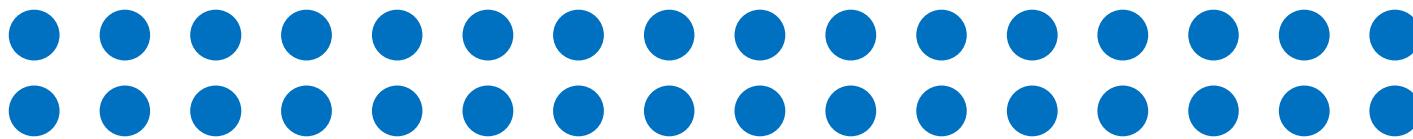
- Training
- Testing



and so on...

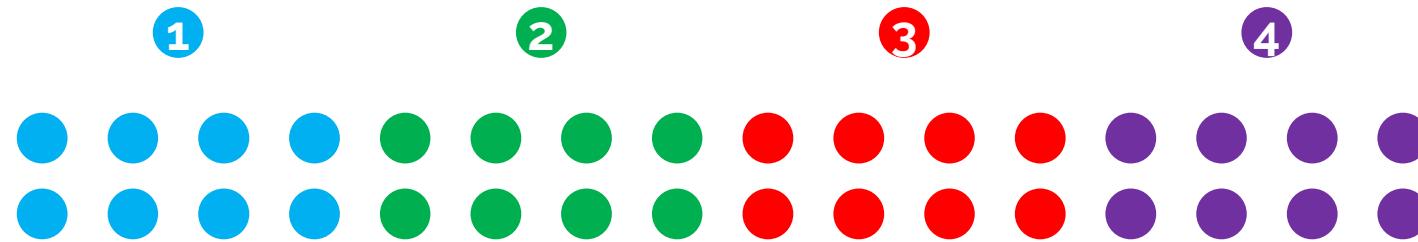
Results obtained from multiple rounds can be averaged in order to obtain a quantification of the performance of the classifier.

# K-fold cross validation



The original dataset is randomly partitioned into  $k$  subsets of equal size.

# K-fold cross validation

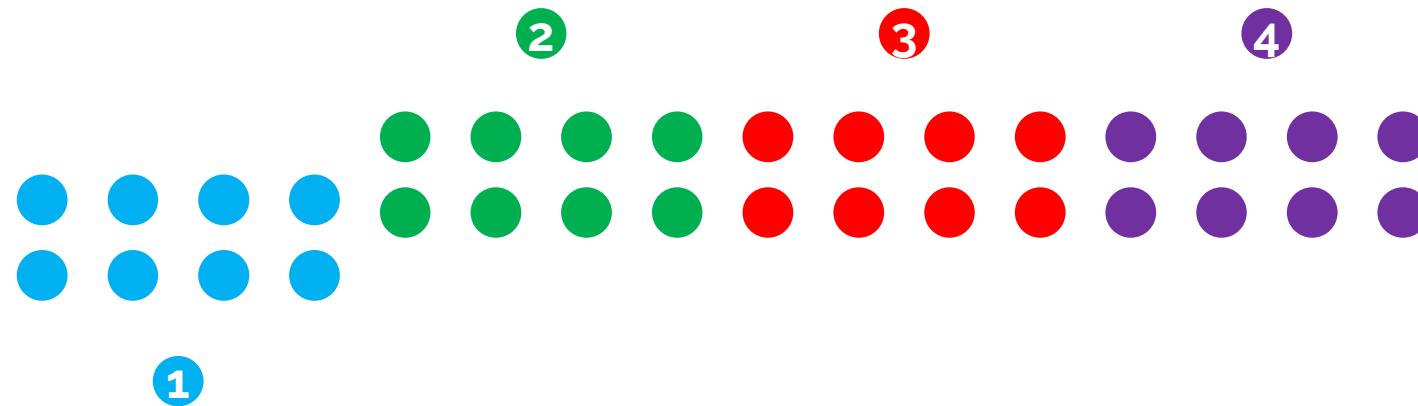


The original dataset is randomly partitioned into  $k$  subsets of equal size

TRAINING of the classifier is performed using  $k-1$  subsets

TESTING is performed using the remaining subset

# K-fold cross validation

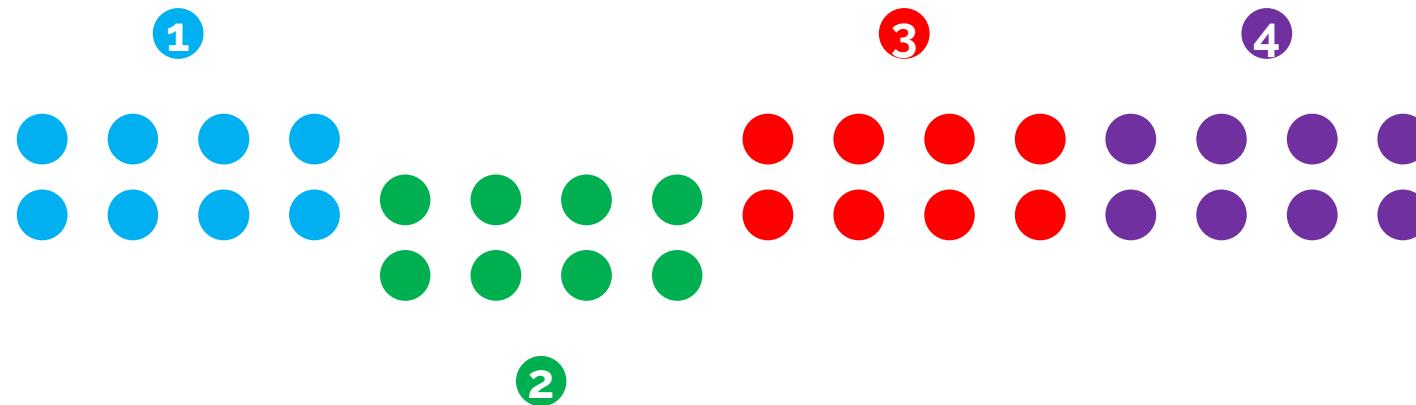


2 3 4 Training

1 Testing

The procedure is then repeated k times, until all subsets are used once as testing set.

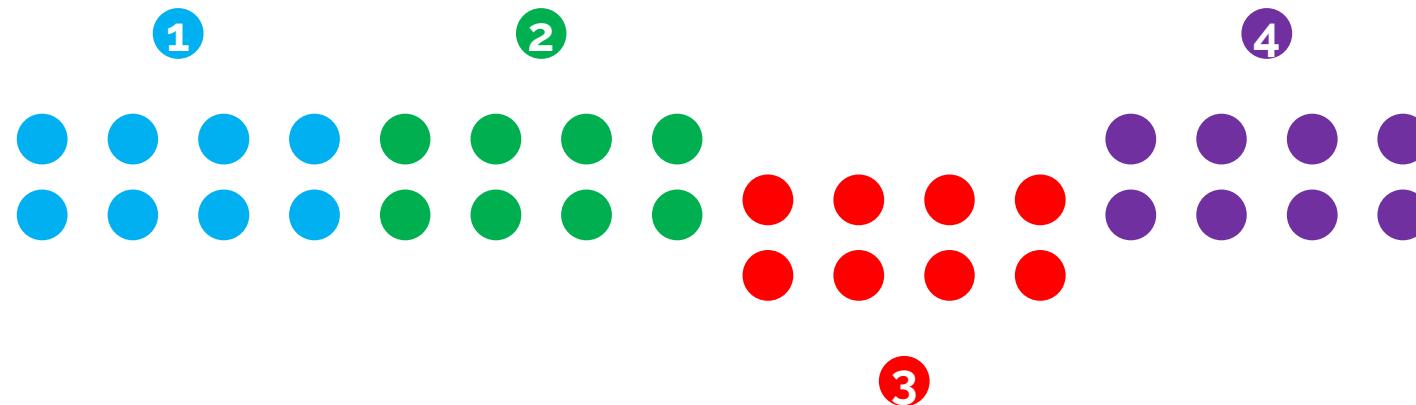
# K-fold cross validation



- ①    ③ ④    Training
- ②              Testing

The procedure is then repeated k times, until all subsets are used once as testing set.

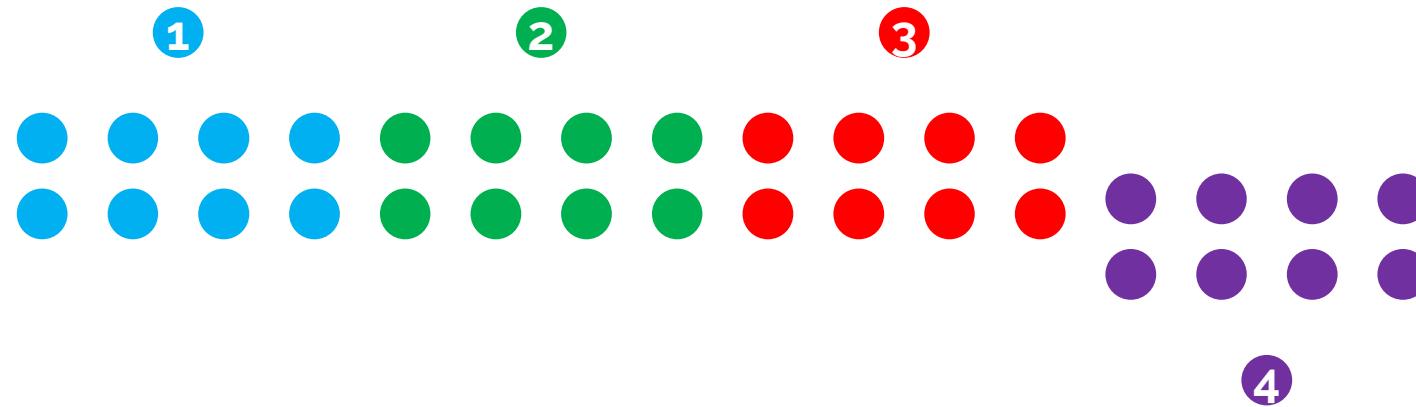
# K-fold cross validation



- 1 2 4 Training
- 3 Testing

The procedure is then repeated k times, until all subsets are used once as testing set.

# K-fold cross validation



① ② ③

Training

④

Testing

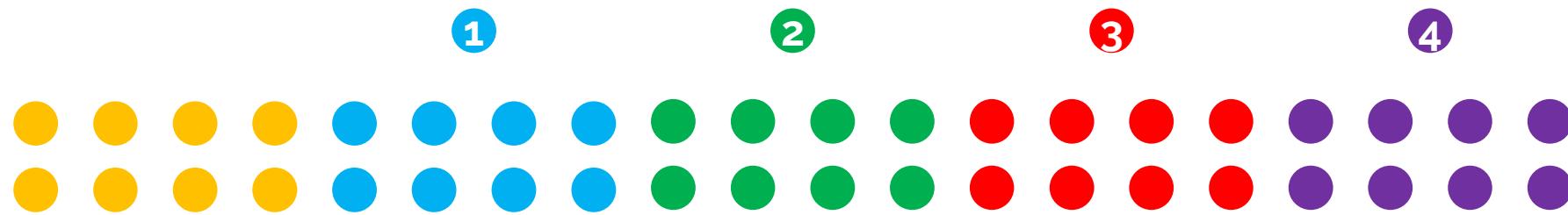
The procedure is then repeated k times, until all subsets are used once as testing set.

# K-fold cross validation

Advantages:

- each sample of the original dataset is used once for validation
- all samples being used for both training and testing phases

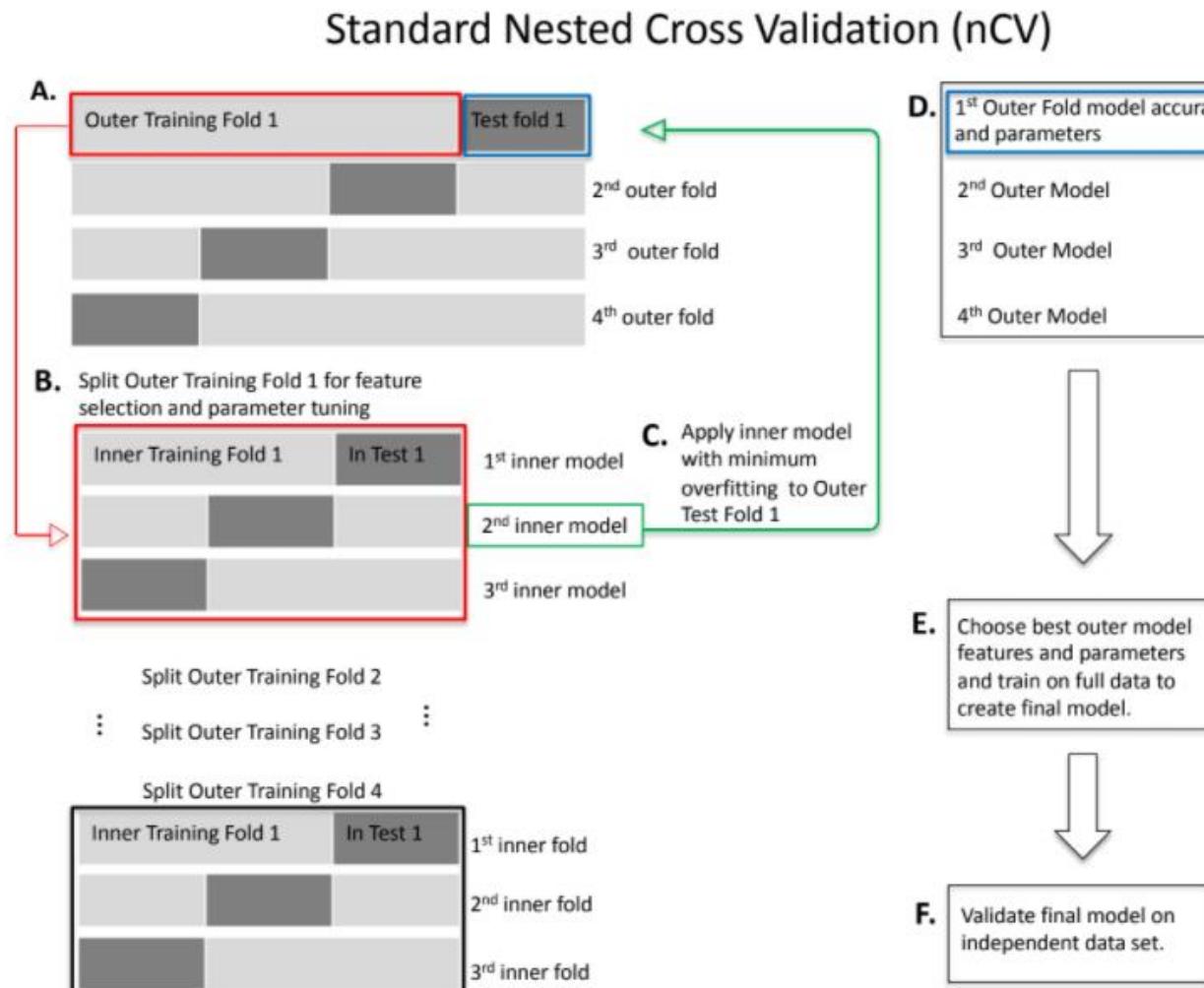
# Nested K-fold cross validation



# Nested K-fold cross validation



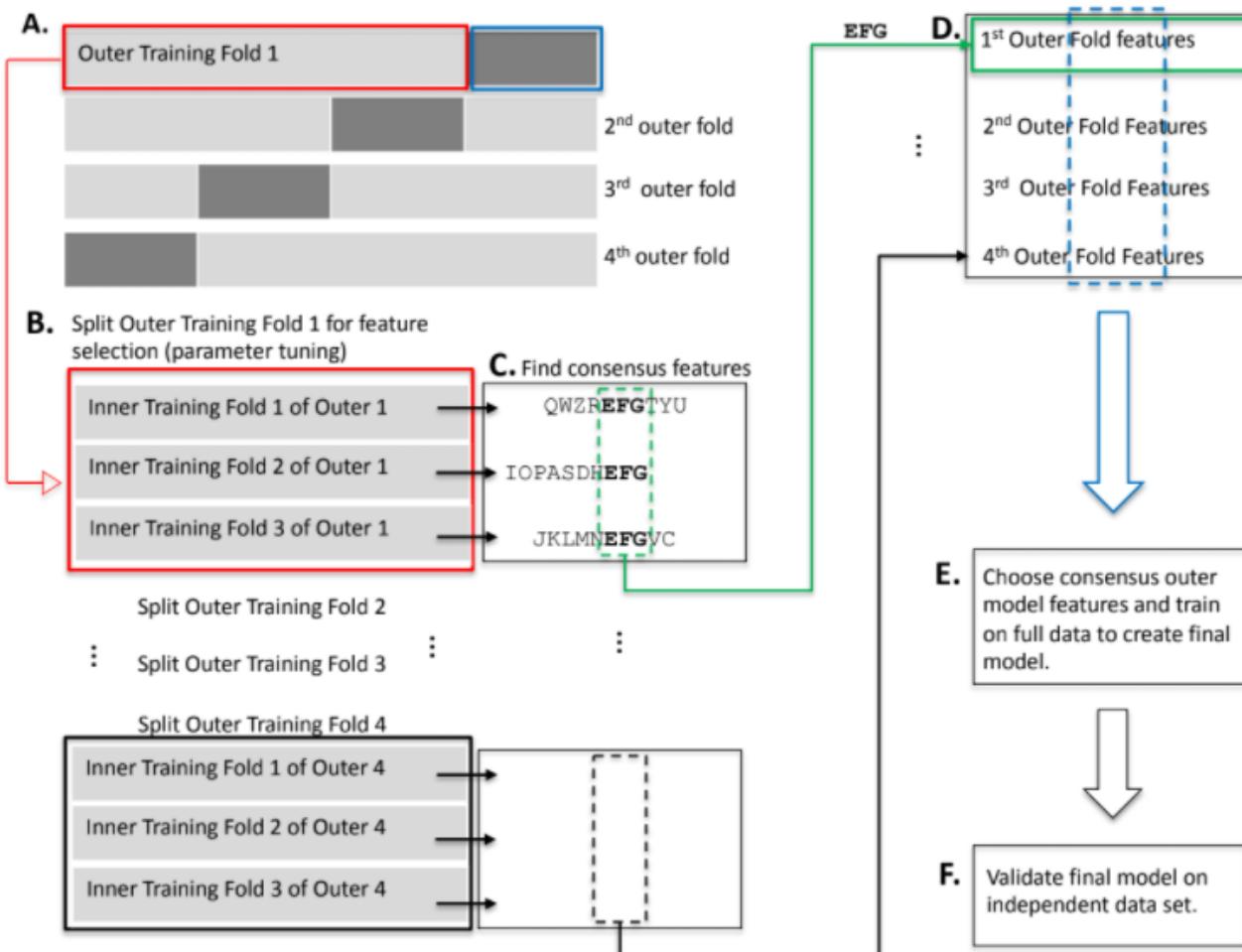
# Nested K-fold cross validation



**Fig. 1. Standard nested Cross-Validation (nCV).** A. Split the data into outer folds of training and testing data pairs (4 outer folds in this illustration). Then do the following for each outer training fold (illustration starting with Outer Training Fold 1 (red box, A)). B. Split outer training fold into inner folds for feature selection and possible hyperparameter tuning by grid search. C. Use the best inner training model including features and parameters (2nd inner model, green box, for illustration) based on minimum overfitting (difference between training and test accuracies) in the inner folds to test on the outer test fold (green arrow to blue box, Test Fold 1). D. Save the best model for this outer fold including the features and test accuracies. Repeat B-D for the remaining outer folds. E. Choose the best outer model with its features based on minimum overfitting. Train on the full data to create the final model. F. Validate the final model on independent data.

# Nested K-fold cross validation

## Consensus Nested Cross Validation (cnCV)



**Fig. 2. Consensus Nested Cross-Validation (cnCV).** A. Split the data into outer folds (4 outer folds in this illustration). Then do the following for each outer training fold (illustration starting with Outer Training Fold 1 (red box, A)). B. Split outer training fold into inner folds for feature selection and optional hyperparameter tuning by grid search. C. Find consensus features. For each fold, features with positive Relief scores are collected (e.g., “QWZREFGTYU” for fold 1). Negative Relief scores have high probability of being irrelevant to classification. The implementation allows for different feature importance methods and tuning the number of input features. Consensus features (found in all folds) are used as the best features in the corresponding outer fold. For example, features “EFG” are shared across the three inner folds. This procedure is used in the inner and outer folds of cnCV. Classification is not needed to select consensus features. D. The best outer fold features (green arrow to green box) are found for each fold (i.e., Repeat B-D for all outer folds). E. Choose the consensus features across all the outer folds to train the final model on full data. Consensus features are selected based on training data only. Classification is not performed until the outer consensus features are selected (A-D). F. Validate the final model on independent data.

# Evaluation metrics

A brief overview...  
(see next lessons)

1. Accuracy
2. Sensitivity and specificity
3. AUC

# Accuracy

Accuracy is the most used metric in classification problems

Accuracy of classification =

# correctly classified  
samples (for both  
classes)

/ # classified samples

If the error rate is defined as the number of misclassified samples (both classes) divided by the total number of classified samples, it is evident that accuracy and error rate are complementary measures.

# Sensitivity and specificity

Two metrics of great importance in medicine are sensitivity and specificity, as they measure the rate of correctly classified samples in the positive (pathological) and negative (normal) class, respectively.

Sensitivity (also known as True Positive Rate or Recall) is given by the number of correctly classified samples belonging to the positive class (true positives) divided by the total number of samples belonging to the positive class (true positives plus false negatives).

Sensitivity =

# correctly classified  
samples in the positive  
class

/ # positive samples

# Sensitivity and specificity

Specificity (also known as True Negative Rate) is given by the number of correctly classified samples belonging to the negative class (true negatives) divided by the total number of samples belonging to the negative class (true negatives plus false positives).

Specificity =

# correctly classified  
samples in the negative  
class

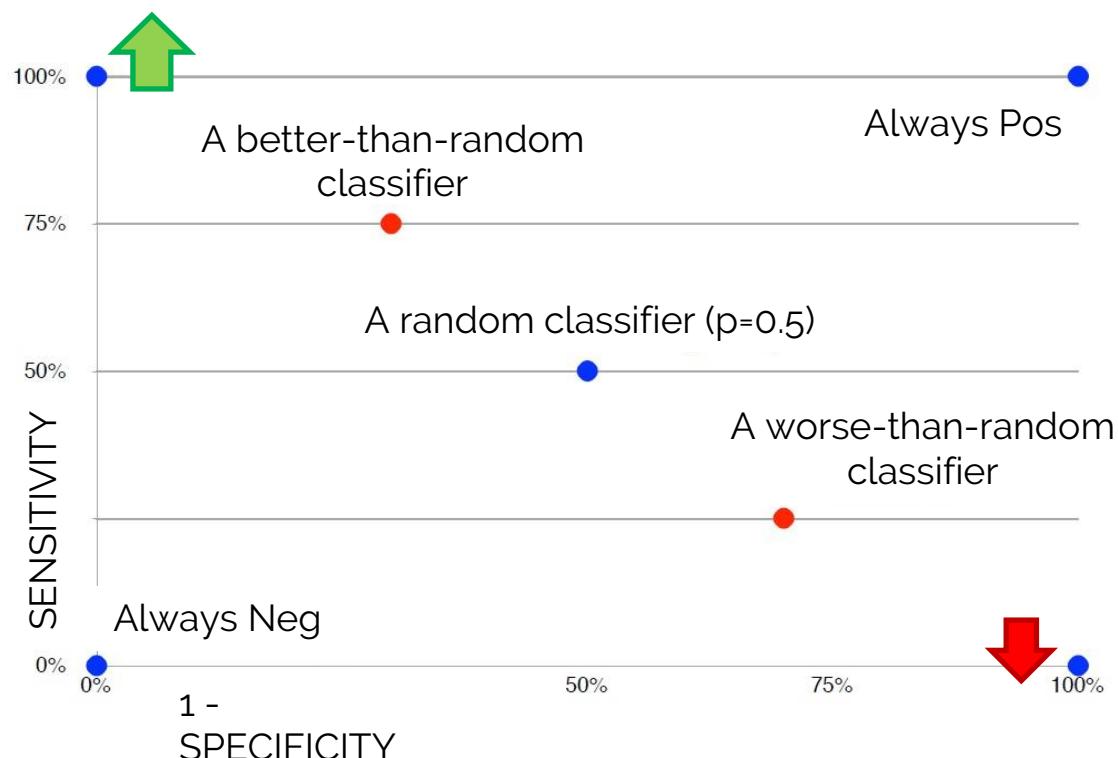
# negative samples

Here, true positive (negative) gives the number of correctly classified samples belonging to the positive (negative) class, while false positive (negative) gives the number of misclassified samples belonging to the negative (positive) class.

# ROC analysis and Area Under the (ROC) Curve

Another important metric in classification problems is given by the study of the Receiver Operating Characteristic (ROC) curve.

For a binary classifier, A ROC curve is a plot of the TPR (sensitivity) against the FPR ( $1 - \text{specificity}$ ), which can be obtained at different setting thresholds.

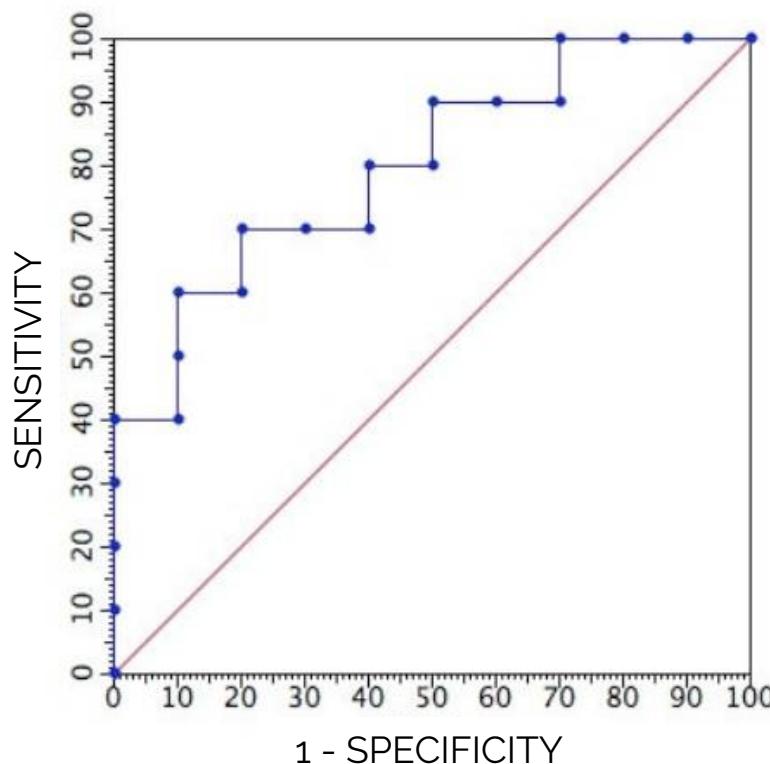


The Area Under the ROC Curve (AUC) gives a quantification of the classifier performance, with a higher statistical consistency than accuracy.

# ROC analysis and Area Under the (ROC) Curve

Another important metric in classification problems is given by the study of the Receiver Operating Characteristic (ROC) curve.

For a binary classifier, A ROC curve is a plot of the TPR (sensitivity) against the FPR ( $1 - \text{specificity}$ ), which can be obtained at different setting thresholds.

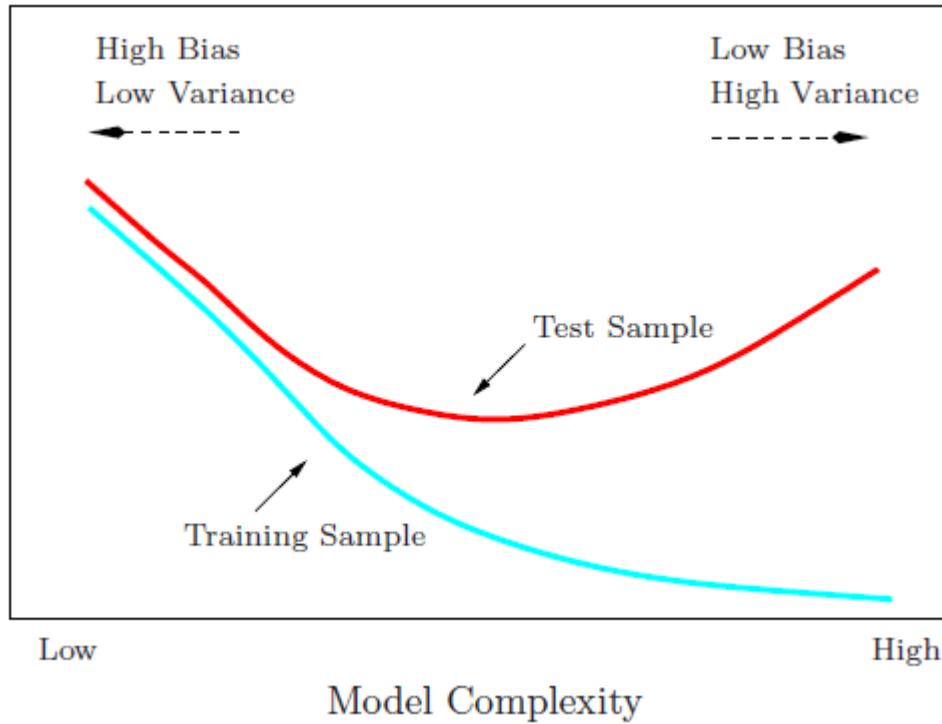


The Area Under the ROC Curve (AUC) gives a quantification of the classifier performance, with a higher statistical consistency than accuracy.

# UNDERFITTING, OVERFITTING AND...

# ... Best Fitting

Prediction Error

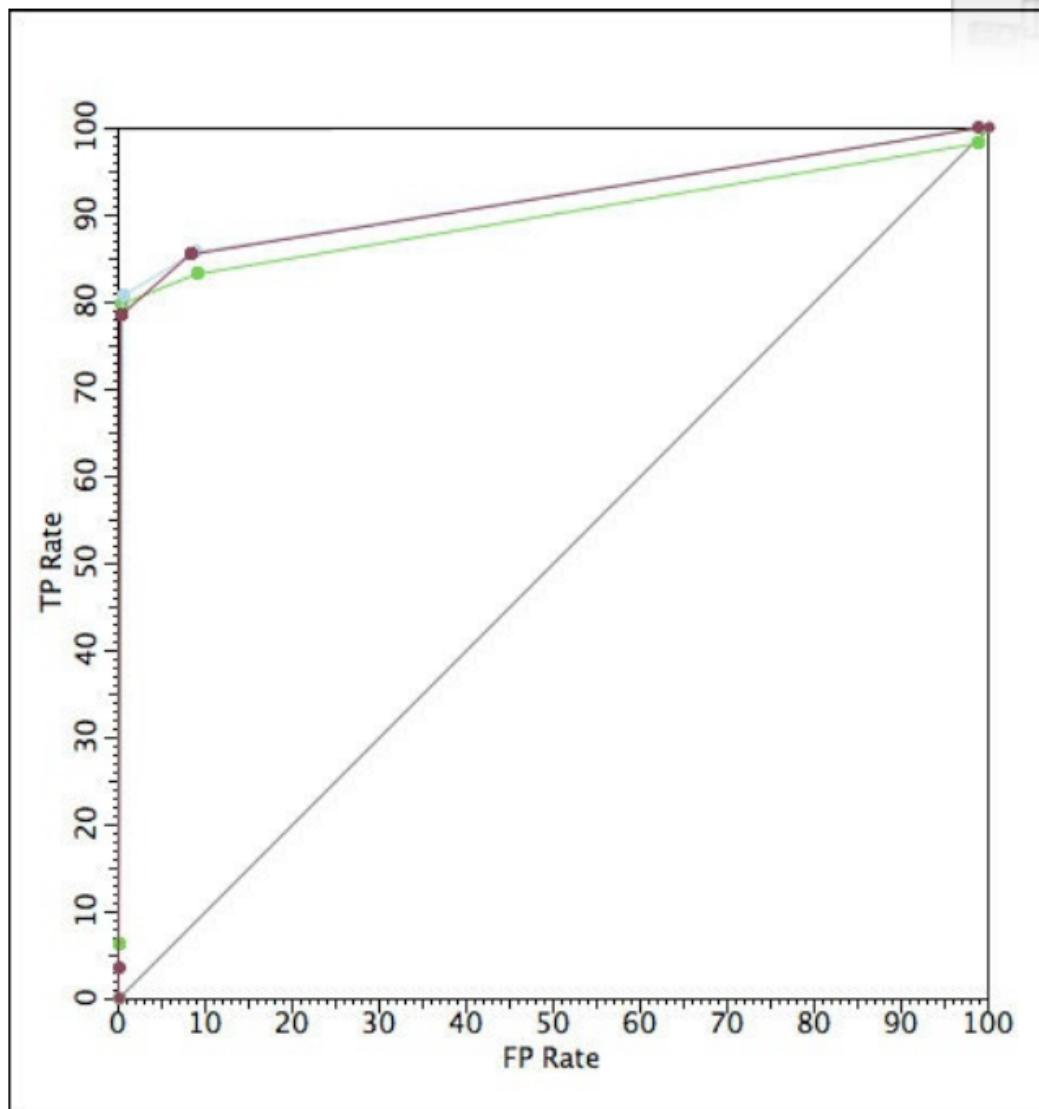


STANFORD.EDU

Symptoms	Underfitting	Just right	Overfitting
Regression illustration	<ul style="list-style-type: none"><li>• High training error</li><li>• Training error close to test error</li><li>• High bias</li></ul>	<ul style="list-style-type: none"><li>• Training error slightly lower than test error</li></ul>	<ul style="list-style-type: none"><li>• Very low training error</li><li>• Training error much lower than test error</li><li>• High variance</li></ul>
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"><li>• Complexify model</li><li>• Add more features</li><li>• Train longer</li></ul>		<ul style="list-style-type: none"><li>• Perform regularization</li><li>• Get more data</li></ul>

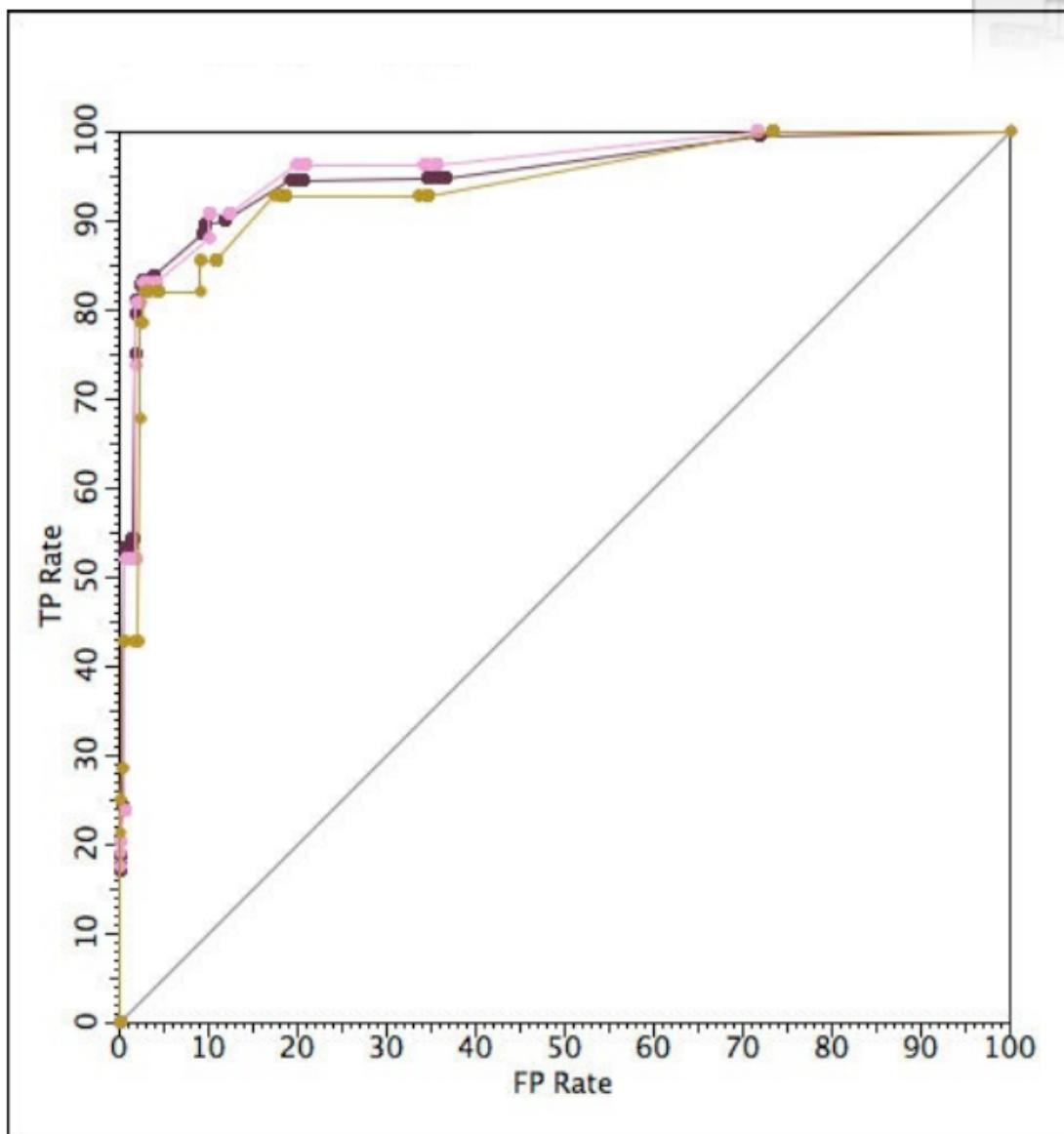
# QUIZ

# Which is which?



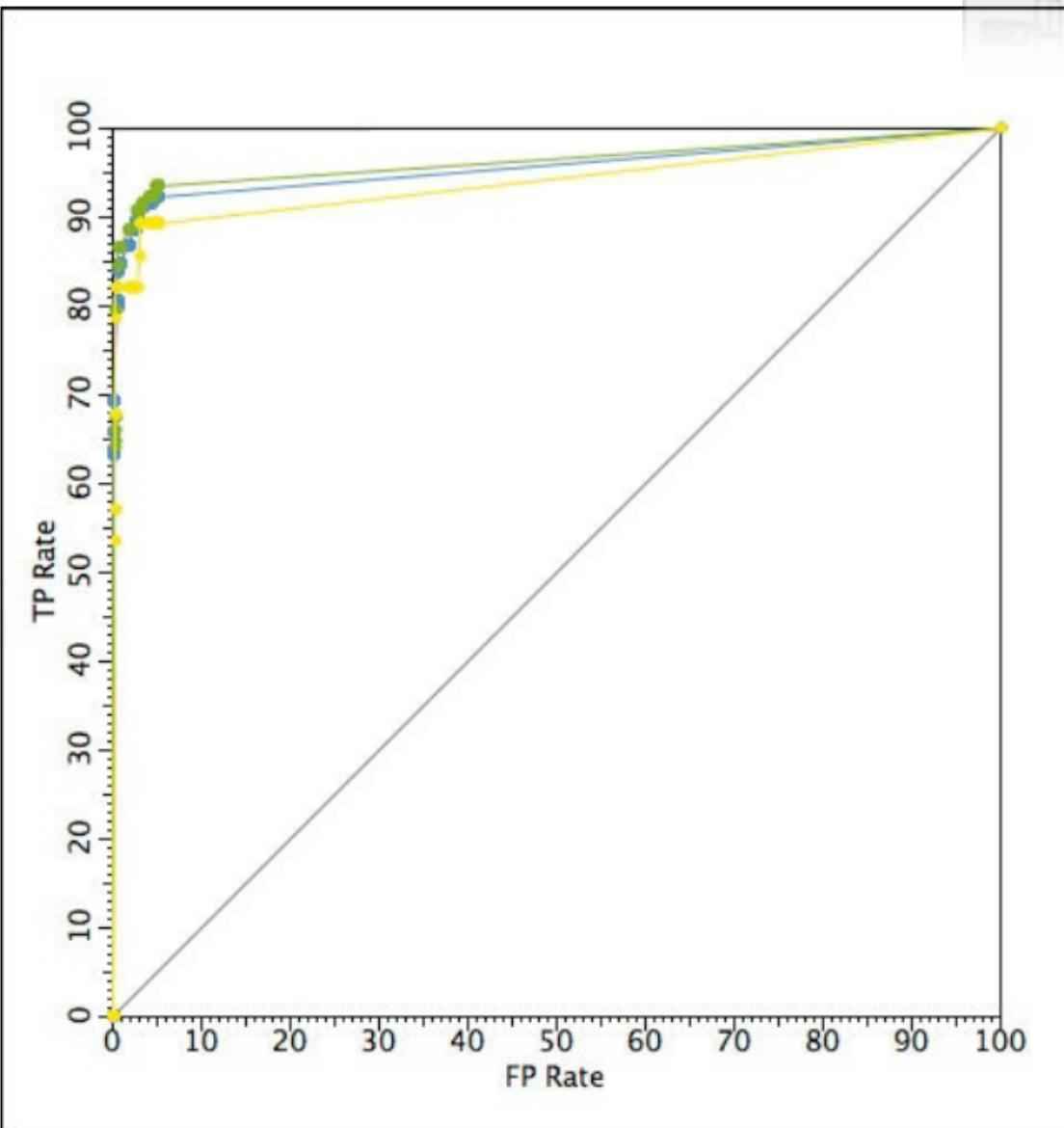
- ❖ Four models:
  - ❖ decision tree
  - ❖ k-nearest neighbour
  - ❖ linear classifier
  - ❖ naive Bayes
- ❖ trained on 2,000 examples and evaluated on
  - ❖ 18,000 test examples
  - ❖ 3,600 of those (20%)
  - ❖ 720 of those (4%)

# Which is which?



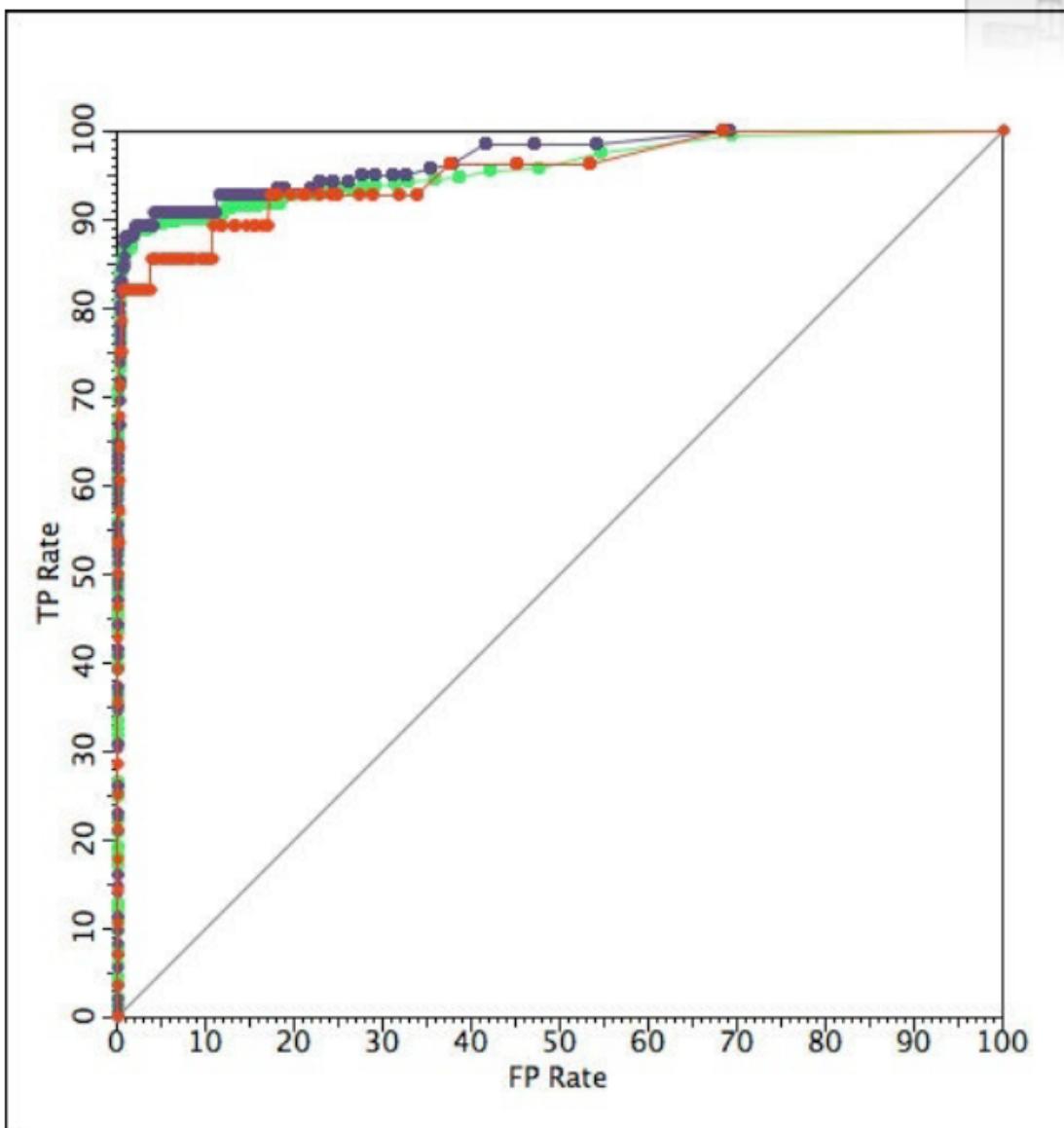
- ❖ Four models:
  - ❖ decision tree
  - ❖ k-nearest neighbour
  - ❖ linear classifier
  - ❖ naive Bayes
- ❖ trained on 2,000 examples and evaluated on
  - ❖ 18,000 test examples
  - ❖ 3,600 of those (20%)
  - ❖ 720 of those (4%)

# Which is which?



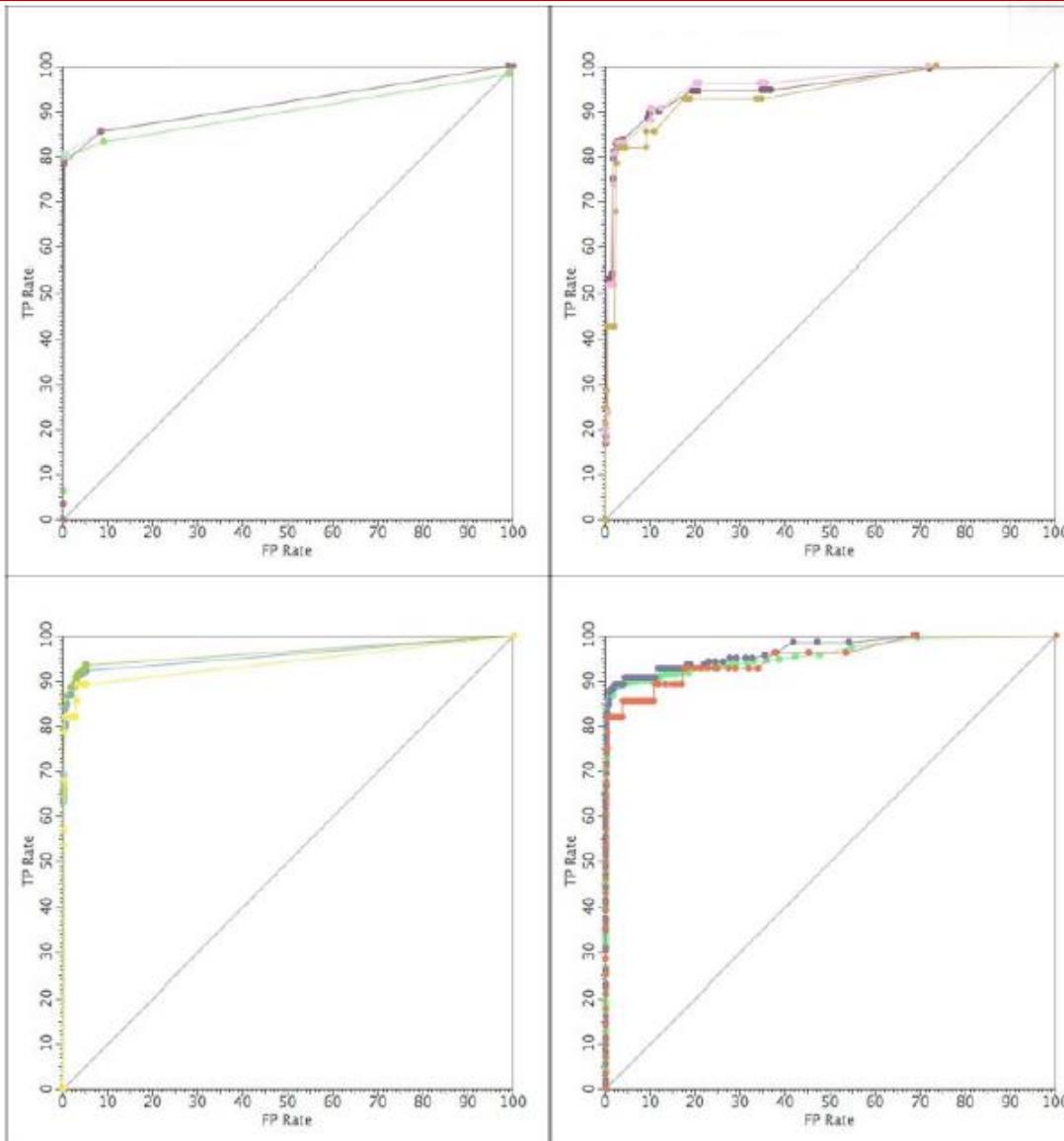
- ❖ Four models:
  - ❖ decision tree
  - ❖ k-nearest neighbour
  - ❖ linear classifier
  - ❖ naive Bayes
- ❖ trained on 2,000 examples and evaluated on
  - ❖ 18,000 test examples
  - ❖ 3,600 of those (20%)
  - ❖ 720 of those (4%)

# Which is which?



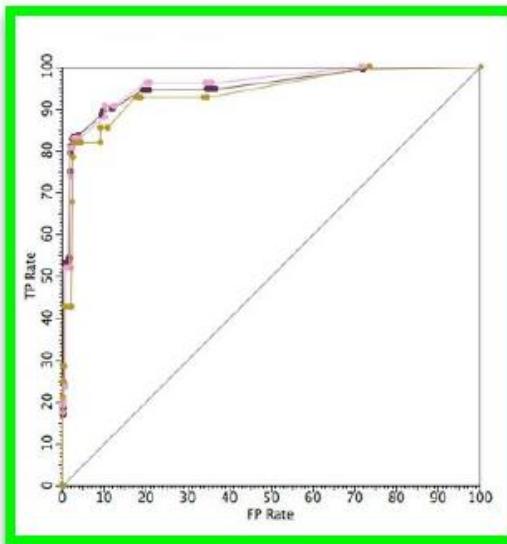
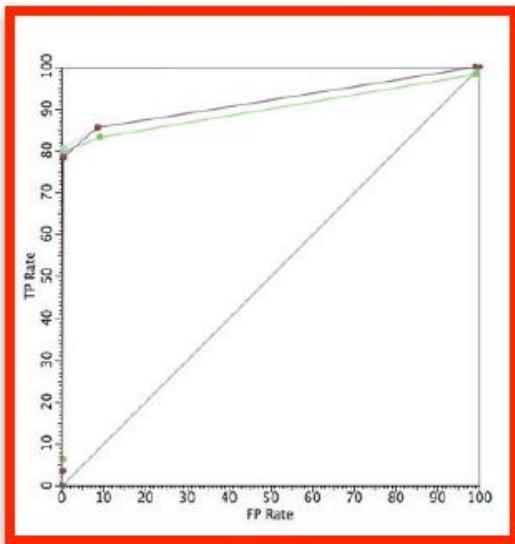
- ❖ Four models:
  - ❖ decision tree
  - ❖ k-nearest neighbour
  - ❖ linear classifier
  - ❖ naive Bayes
- ❖ trained on 2,000 examples and evaluated on
  - ❖ 18,000 test examples
  - ❖ 3,600 of those (20%)
  - ❖ 720 of those (4%)

# Which is which?



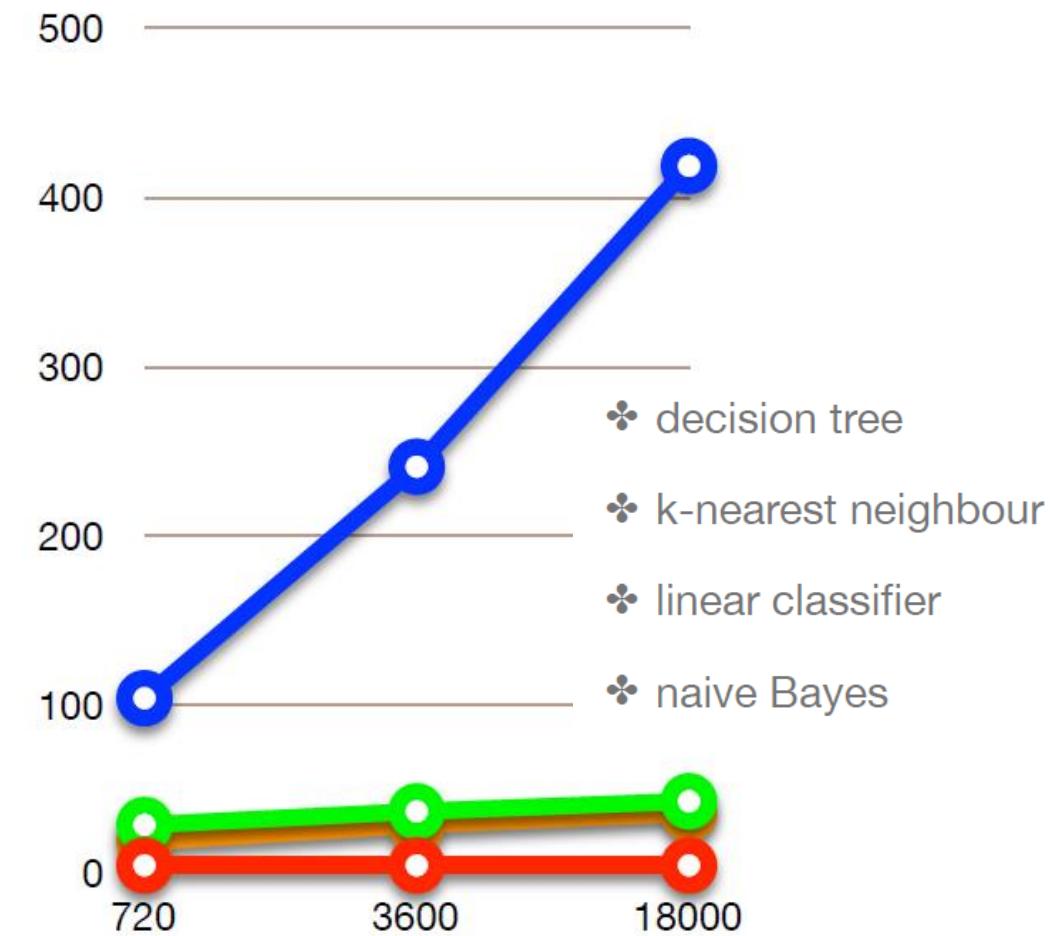
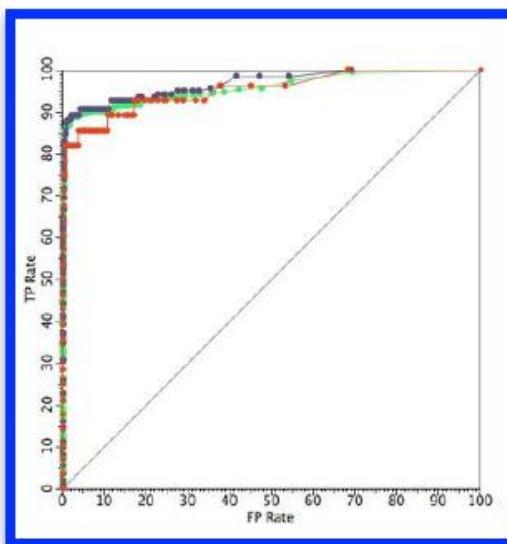
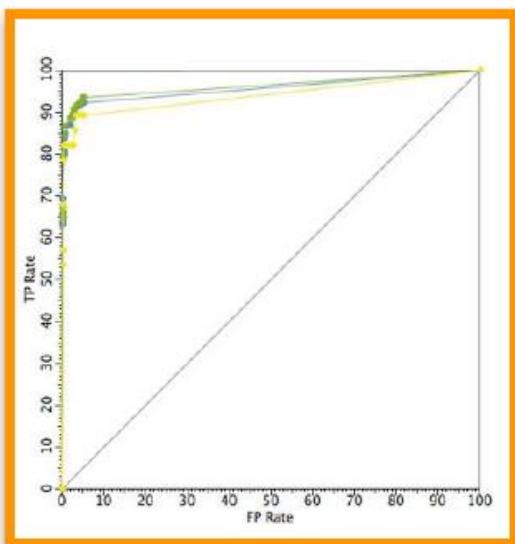
- ❖ Four models:
  - ❖ decision tree
  - ❖ k-nearest neighbour
  - ❖ linear classifier
  - ❖ naive Bayes
- ❖ trained on 2,000 examples  
and evaluated on
  - ❖ 18,000 test examples
  - ❖ 3,600 of those (20%)
  - ❖ 720 of those (4%)

# Which is which?



Top Left  
Bottom Left

Top Right  
Bottom Right



`christian.salvatore@iusspavia.it`

`https://christiansalvatore.github.io/machinelearning-iuss/`