Machine Learning for Human Learners

di Stefano Dal Pra

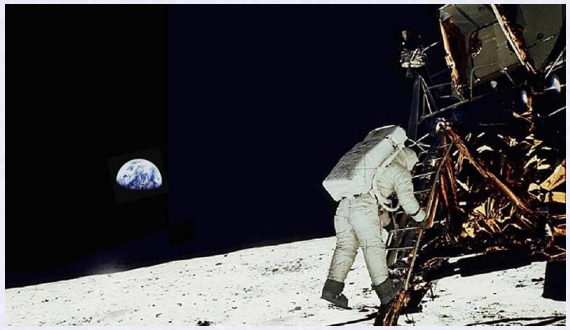
17/01/2019

Email: stefano.dalpra@cnaf.infn.it

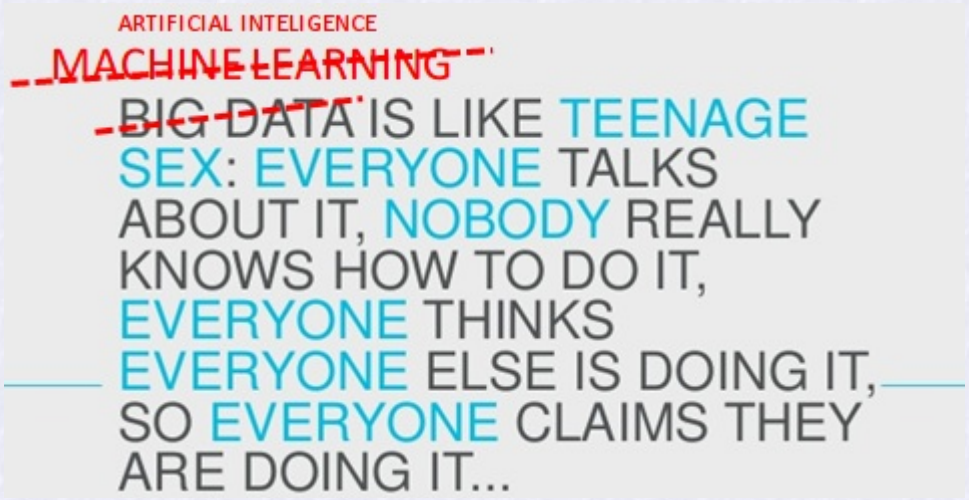
1 Premesse e Motivazioni

- Applicazioni di ML sono da tempo "in produzione" e tutti noi ne siamo utenti. Basata spesso su tecniche "classiche" o note da decenni, ma cresciute di recente grazie a enorme sviluppo di CPU power e disponibilità di data.
- Il CNAF ha Iniziato un progetto di analisi log e failure prediction (cfr. https://agenda.infn.it/event/17430/) con la collaborazione di tre summer students, con supervisione interna di B. Martelli ed esterna di D. Bonacorsi, che tiene un corso di Applied ML per PhD.
- Questa presentazione cerca di offrire qualche conoscenza iniziale di ML per chi desideri interessarsene o considerare i risultati che può offrire.

1.1 Cosa si può ottenere con ML?

Risultati fantastici	Risultati fantasy
	

1.2 La legge dell'hype



1.3 Deep Learning, the easy way

```
root@mypc:~# apt-get install python-lasagne lasagne-doc
root@mypc:~# cd /usr/share/doc/python-lasagne/examples
root@mypc: # python mnist.py
Loading data...  Downloading train-images-idx3-ubyte.gz
[SNIP]
Building model and compiling functions...
Starting training...
```

2 Artificial Intelligence e Machine Learning

1950: *Computing machinery and intelligence* (Alan Turing). Considerato il "Manifesto dell' Intelligenza Artificiale".

Inizia con: I propose to consider the question, "Can machines think?"

Invece di discutere cosa sia il pensiero, definisce un test:

Test di Turing. Se una macchina riesce a farsi passare per un essere umano, "supera il test".

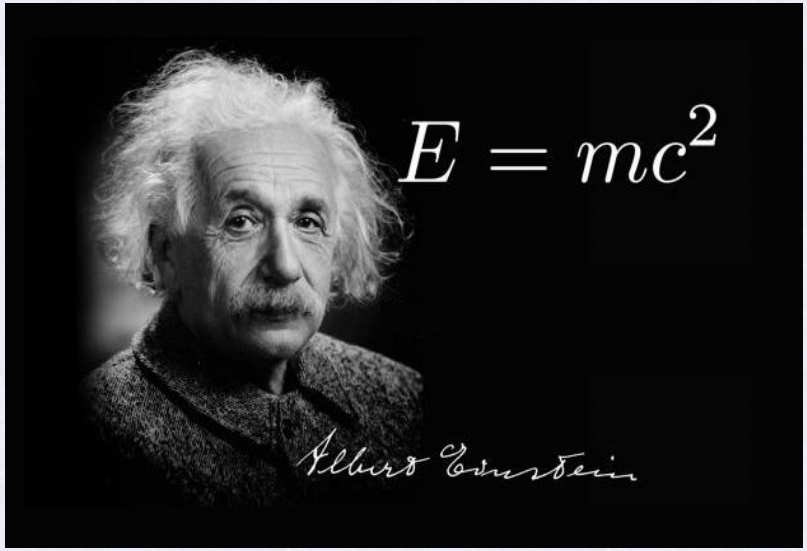
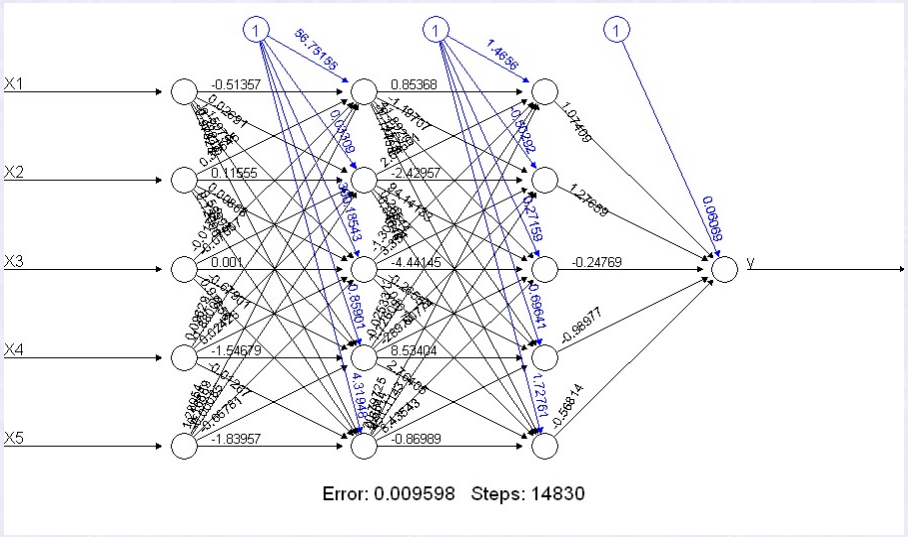
Esistono due posizioni, sulla AI:

Tesi forte dell'AI. È possibile creare delle macchine in grado di pensare, apprendere, diventare coscienti. (Impostazione "top-down")

Tesi debole dell'AI. È possibile creare macchine abili quanto o piú di un essere umano su singoli compiti specifici. (Impostazione "bottom–up")

Il Machine Learning deriva dalla Tesi debole.

2.1 La conoscenza per la AI

tesi forte	tesi debole
	
$e^{i\pi} + 1 = 0$	$\lvert(2.71828, 0)^{(0,1)(3.1416,0)} + (1, 0)\rvert < 10^{-12}$

3 Definizioni, terminologie, algoritmi

Definizioni di Machine Learning

- **Artur Samuel, 1959.** Macchine con abilità di apprendere senza apposita programmazione
- **Tom Mitchell, 1998.** Si dice che un software apprende dall'esperienza E rispetto ad un task T e una misura di performance P se le sue performaces migliorano con l'esperienza (esperienza $\equiv$ training).

Esempio: filtro antispam

- T: classificare mail come spam/non spam
- E: un set di email già classificate (**esempi**, in gergo ML)
- P: frazione di mail classificate correttamente

Nota: è un esempio di **Supervised Learning** (l'esperienza E viene fornita esternamente). Si parla di **Unsupervised Learning** quando l'algoritmo procede indipendentemente (es. **clustering:** dai quotidiani del giorno, raggruppare quelli che parlano dello stesso argomento)

3.0.1 Algoritmi

- Due gruppi principali:

  Regression: il task T produce un valore y reale (es.stimatori, predittori)

  Classification: il task T produce due o piú valori, che indicano la classe.

Note

- Se le classi sono due ($y \in \{0, 1\}$) si parla di **classificatore binario**, altrimenti di **classificatore multiclasse**
- L'algoritmo che implementa il classificatore binario, si chiama **logistic regression** (sic!)

Esempi

- OCR $\rightarrow$ classificatore. In: bitmap con una cifra; Out: $y \in \{0, 1, 2, ..., 10\}$
- data l'età di un bambino, stimare l'altezza. $\rightarrow$ Regression

4 Regression Learning

Illustriamo un esempio semplice, stile "Hello world", che però permette di introdurre piú o meno tutti i temi e le problematiche importanti per il ML.

Esempio

- vogliamo stimare il consumo medio dei chiller di una sala calcolo in base alla temperatura media esterna.
- Abbiamo ricavato N coppie di valori: $\{(x_1, y_1), ..., (x_N, y_N)\}$ (**training set**)

$x_i \to \boldsymbol{x}_N = [x_1, ..., x_N]'$:Temperatura media esterna (x_i: input o **features**)

$y_i \to \boldsymbol{y}_N = [y_1, ..., y_N]'$: Consumo Chiller (y_i: output o **target**)

Dobbiamo stimare $\hat{y}$ per un generico valore di x. Scegliamo un modello che lega y a x:

$$\hat{y} = \boxed{h_\theta(x) = \theta_0 + \theta_1 x} = [1, x] \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \boldsymbol{\varphi}'(x) \cdot \boldsymbol{\theta}$$

e "impariamo" dalle misure (x_i, y_i) (**training examples**) quali valori di $\boldsymbol{\theta} = [\theta_0, \theta_1]'$ vanno *meglio*. La funzione $h_\theta(\cdot)$ in ML è detta **ipotesi** o **stimatore** o **predittore** a seconda dei contesti.

Nota Un problema importante in ML è scegliere l'ordine del modello (la dimensione di $\boldsymbol{\theta}$).

In generale il modello (con **un** solo ingresso) si può esprimere così:

$$h_\theta(x) = [1, x, \ldots, x^m] \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix} = \boldsymbol{\varphi}'_m(x) \cdot \boldsymbol{\theta}$$

Per trovare $\boldsymbol{\theta}$ si ricorre a una funzione *costo* (gli errori "si pagano", e si vuole "pagare poco"):

$$J(\theta) = \frac{1}{2\,N} \sum_{i=1}^{N} (y_i - h_\theta(x_i))^2$$

$$= \frac{1}{2\,N} \left\| \boldsymbol{y}_N - \underbrace{\begin{bmatrix} \boldsymbol{\varphi}'_m(x_1) \\ \vdots \\ \boldsymbol{\varphi}'_m(x_N) \end{bmatrix}}_{N \times m} \boldsymbol{\theta} \right\|^2$$

$$J(\theta) = \frac{1}{2\,N} \| \boldsymbol{y}_N - A\,\boldsymbol{\theta} \|^2$$

e si cerca il $\hat{\boldsymbol{\theta}}$ che la rende minima. Tale minimo esiste ed è unico, ed è dato da:

$$\hat{\boldsymbol{\theta}} = \operatorname{Arg}\min_{\boldsymbol{\theta}} \|\boldsymbol{y}_N - A\,\boldsymbol{\theta}\|^2 = \boxed{(A'\,A)^{-1}\,A'\,\boldsymbol{y}_N}$$

Osservazioni 1 siamo in pieno "Machine Learning" perché:

- L'*esperienza* E sono i dati: $\boldsymbol{x}_N, \boldsymbol{y}_N$ (training dataset)
- Il *task* T è: "dato un x stimare y"
- Le *performances* P sono date da $\frac{1}{2\,N}\|\boldsymbol{y}_N - A\,\hat{\boldsymbol{\theta}}\|^2$ (funzione costo)
- I parametri $\hat{\boldsymbol{\vartheta}}$ sono la *conoscenza*
- È un esempio di Supervised Learning: le y_i sono le "risposte" agli x_i e sono date.

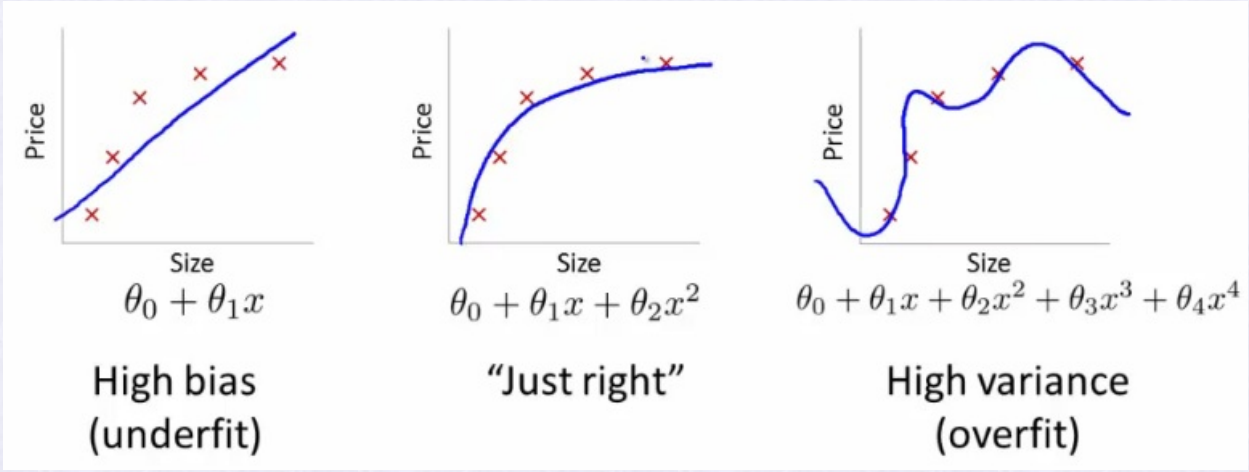
Osservazioni 2

- È comunque il metodo dei minimi quadrati, impiegato dai tempi di C.F.Gauss (1777–1855). CPU e BigData permettono di rivederlo e svilupparlo in chiave ML.

- In molti casi per minimizzare $J(\boldsymbol{\theta})$ si usano metodi iterativi (es. *steepest descent method*)
- Il modello può essere piú o meno ricco, a seconda dell'*ordine* del modello, m (è un **iperparametro**, lo dobbiamo scegliere noi, non viene "imparato").

Problema: scegliere il valore migliore per m:

- Troppo piccolo $\rightarrow$ underfitting (aumentare ordine o aggiungere features)
- Troppo grande $\rightarrow$ overfitting (ridurre ordine o features o **regolarizzare**)



Nota1 Queste considerazioni valgono assumendo che gli esempi usati reppresentino bene tutto quel che c'è da sapere. Per es. potrebbero arrivare molti esempi nuovi e confermare il primo dei tre modelli.

Nota2 I dati sono tutti già noti (**Batch Learning**). E se invece ne arrivano continuamente di nuovi?

Minimi Quadrati Ricorsivi

$$\hat{\boldsymbol{\theta}}_{n+1} = f\left(\hat{\boldsymbol{\theta}}_n, x_{n+1}, y_{n+1}\right)$$

È possibile aggiornare lo stimatore precedente ad ogni nuovo esempio (**Online Learning**).

Nota Ci sono dei casi in cui i dati piú vecchi non aiutano a fare stime migliori, perché diventano "obsoleti". Si gestiscono questi casi introducendo dei "coefficienti d'oblio", per cui gli esempi piú vecchi influiscono sempre meno.

Osservazioni 3 Per stimare il consumo dei chiller considerare *solo* la temperatura esterna può essere insufficiente (es: se la farm è spenta, i chiller consumano meno, se è a pieno regime, consumano di piú).

Esempio

In generale il modello $h(\boldsymbol{\theta})$ può considerare piú features:

$x_{1,i} \rightarrow \boldsymbol{x}_{1,N} = [x_{1,1}, \ldots, x_{1,N}]'$:Temperatura esterna

$x_{2,i} \rightarrow \boldsymbol{x}_{2,N} = [x_{2,1}, \ldots, x_{2,N}]'$:Consumo farm

$y_i \rightarrow \boldsymbol{y}_N = [y_1, \ldots, y_N]'$: Consumo Chiller

e cosí il modello può essere, ad es:

$$
\begin{aligned}
h_{\boldsymbol{\theta}}(x_1, x_2) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 \\
&= [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2] \cdot [\theta_0, \ldots, \theta_5]' \\
&= \boxed{\varphi(\boldsymbol{x})' \boldsymbol{\theta}} \\
J(\theta) &= \frac{1}{2N} \sum_{i=1}^{N} (y_i - \varphi(\boldsymbol{x}_i)' \boldsymbol{\theta})^2 \\
&= \boxed{\frac{1}{2N} \| \boldsymbol{y}_N - A \boldsymbol{\theta} \|^2}
\end{aligned}
$$

4.1 Feature Scaling

Normalmente non si applicano i dati all'algoritmo direttamente, quasi sempre vanno prima "adattati". Nell'esempio, le temperature sono numeri $-15 \leqslant x_{1,i} \leqslant 45\,C^o$, mentre i consumi dei chiller e della farm sono $x_{2,i},\ y_i \sim O(10^5)W$. Si cerca sempre di lavorare con valori numericamente vicini tra loro, per ridurre problemi di stabilità numerica. Una trasformazione tipica è:

$$\mu_i = \frac{1}{N}\sum_{k=1}^{N} x_i^{(k)} \quad ; \quad \boxed{x_i \leftarrow \frac{x_i - \mu_i}{s_i}}$$

dove s_i è scelto in modo da avere valori x_i non "troppo grandi". Di solito si usa $s_i = \mathrm{sqm}(x_i)$ o $\max(x_i) - \min(x_i)$. Questa operazione è detta **Feature scaling**.

4.2 Steepest Descent Method

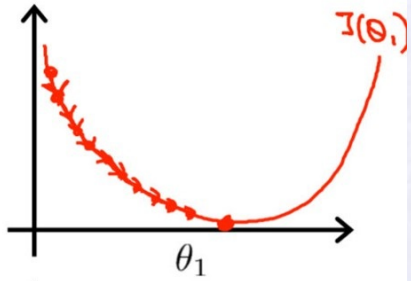
In molti casi per trovare $\hat{\boldsymbol{\theta}} = \mathrm{Arg\,min}_{\boldsymbol{\theta}}\, J(\boldsymbol{\theta})$ si ricorre a metodi iterativi: si parte con un valore iniziale $\hat{\boldsymbol{\theta}}_0$ e lo si cambia ripetutamente cercando di fare in modo che $J(\hat{\boldsymbol{\theta}}_{k+1}) < J(\hat{\boldsymbol{\theta}}_k)$

$$\boxed{\hat{\boldsymbol{\theta}}_{k+1} = \hat{\boldsymbol{\theta}}_k - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta})|_{\hat{\boldsymbol{\theta}}_k}}$$

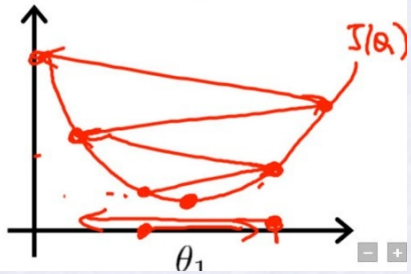
Il parametro α è detto *learning rate* ed è molto importante per la convergenza.

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



5 Logistic Regression

Usata per i classificatori: le "risposte" y_i assumono valori interi. Es. $y_i \in \{0, 1\}$ (classificazione binaria o binomiale) oppure $y_i \in \{0, 1, ..., n\}$ (classificazione multiclasse o multinomiale). Funziona bene quando tutte le classi sono abbastanza rappresentate (non ci sono "eventi rari").

5.1 Esempi

- $x_i \equiv$ ore di studio per un esame,

  $y_i \equiv$ esame superato $(y_i = 1)$ o meno $(y_i = 0)$. (Class. binaria)

- $x_i \equiv$ bitmap 20×20 con una cifra scritta a mano

  $y_i \equiv$ la cifra+1, 0 altrimenti. $y_i \in \{0, 1, ..., 10\}$. (Class. multiclasse)

- $x_i \equiv$ num. job pending in carico ad un CE,

  $y_i \equiv$ una sottomissione va in timeout $(y_i = 1)$ o ha successo $(y_i = 0)$.

In generale si tratta di casi in cui x piccolo $\Rightarrow y = 0$, x grande $\Rightarrow y = 1$ per cui si usa come modello una funzione di tipo *sigmoide*:

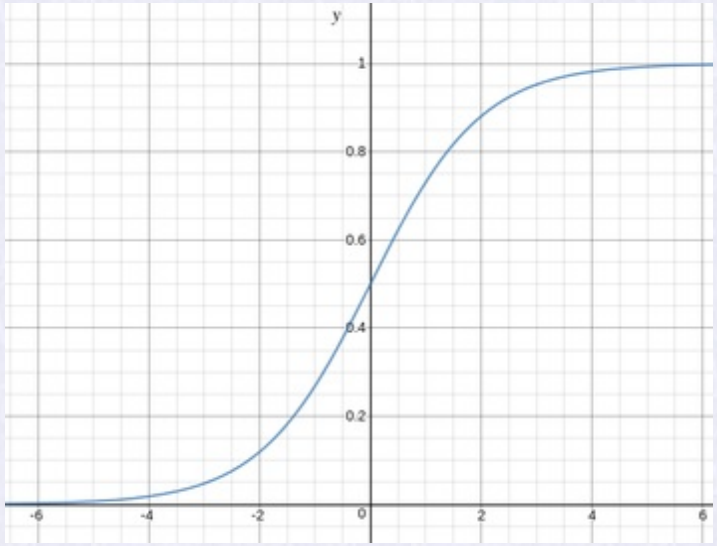


Figura 1. $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \dfrac{1}{1 + e^{-\boldsymbol{\theta}'\boldsymbol{x}}}$

con $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_m]'$, $\boldsymbol{x} = [x_1, \ldots, x_m]'$.

5.2 Funzione costo per logistic regression binaria

In questo caso h non è lineare rispetto a θ, e una conseguenza è che la $J_\theta(x)$ vista prima non è convessa, per cui non è garantito che abbia un minimo unico.

Si ricorre invece a questa:

$$J(\boldsymbol{\theta}) \;=\; -\frac{1}{N}\sum_{k=1}^{N} y_k \ln(h_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) + (1-y_k)\ln(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_k))$$

Nota siccome $y_k \in \{0,1\}$ solo uno dei due addendi nella sommatoria è $\neq 0$.

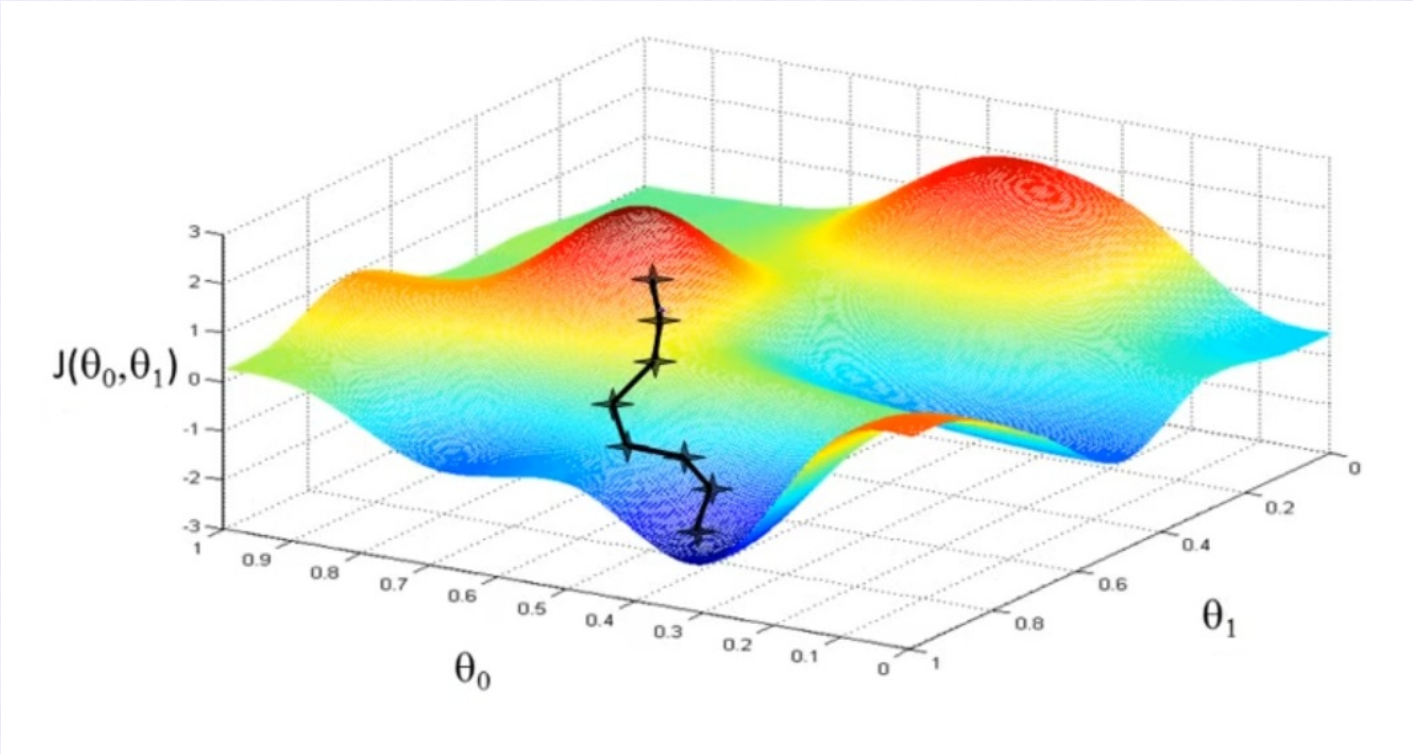
Per trovare il minimo si ricorre ad algoritmi iterativi:

- Steepest descent (metodo del gradiente)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno algorithm)
- L-BFGS (Come BFGS ma usa memoria limitata)
- Conjugate Gradient

Per il metodo del gradiente si ha:

$$\theta_{k+1} \;=\; \theta_k - \alpha \frac{\partial}{\partial\theta} J(\theta)\Big|_{\theta_k}$$

$$\theta_{k+1} \;=\; \theta_k - \alpha \frac{1}{N}\sum_{i=1}^{N} (h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)\, x_k$$

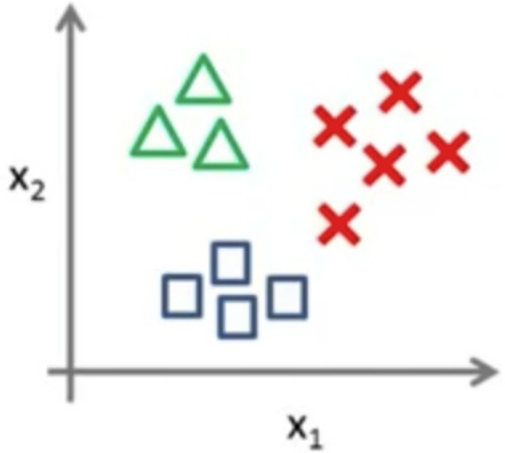


Nota Con large dataset queste operazioni diventano molto onerose!

5.3 Logistic Regression Multiclasse

Si possono implementare tante LR binarie: ciascuna addestrata per riconoscere la propria classe ($y = 1$), e dire $y = 0$ per tutte le altre.

One-vs-all (one-vs-rest):

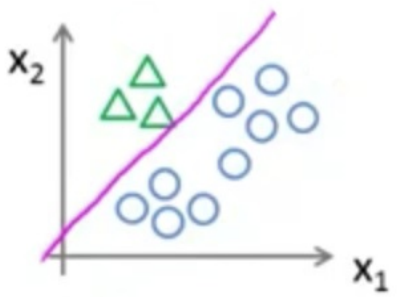


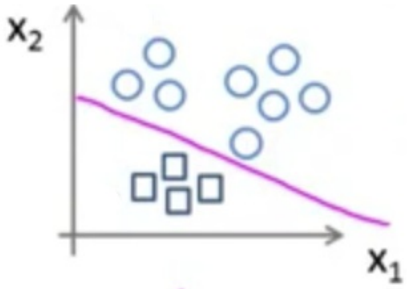
Class 1: △ ←

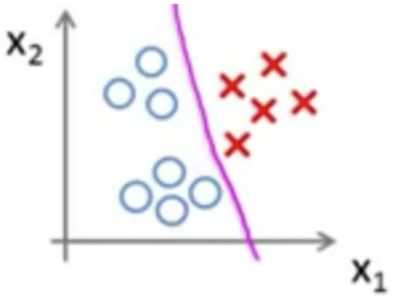
Class 2: □ ←

Class 3: ✕ ←

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$







6 Regolarizzazione

Aiuta a ridurre problemi di overfitting, abbastanza frequenti quando ci sono molte features. Un rimedio è quello di eliminare qualche feature (scelte da noi o da algoritmi di *model selection*) però si perde un po' di informazione. Si rimedia modificando la funzione costo in modo che penalizzi valori θ_i troppo grandi, che quindi diventa, per regressione lineare e logistica:

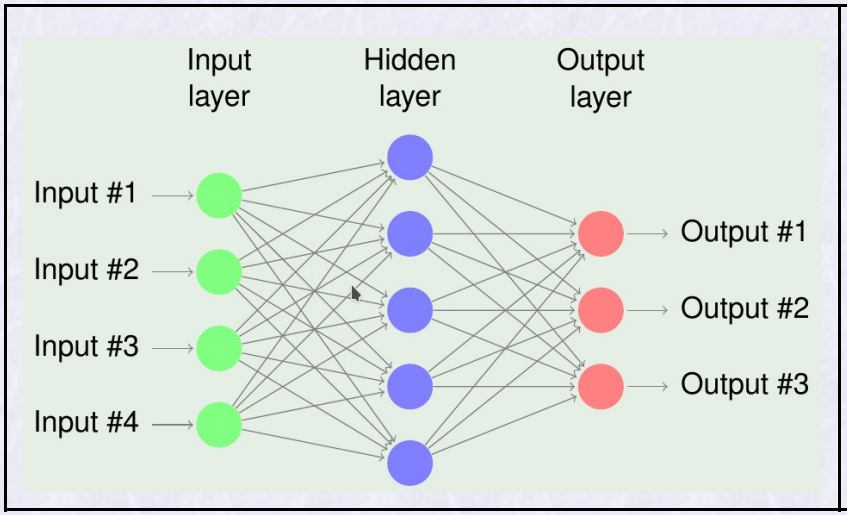
$$J(\theta) = \frac{1}{2\,N}\left[\sum_{i=1}^{N}(y_i - h_\theta(x_i))^2 + \lambda\sum_{i=1}^{m}\theta_i^2\right]$$

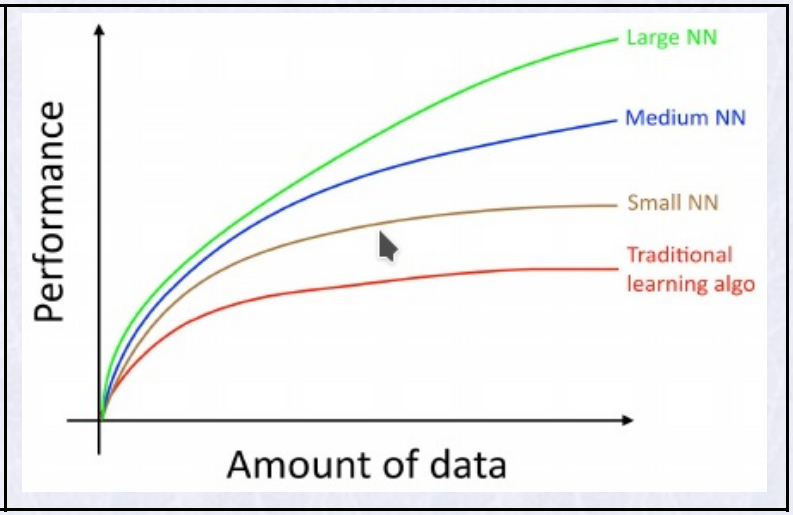
$$J(\theta) = -\frac{1}{N}\sum_{k=1}^{N} y_k \ln(h_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) + (1-y_k)\ln(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}_k)) + \frac{\lambda}{2\,N}\sum_{i=1}^{m}\theta_i^2$$

Naturalmente cambiano di conseguenza le formule per il metodo del gradiente.

7 ANN (Artificial Neural Network)

Sono attualmente lo strumento (classificatore) che può "imparare di piú" e "piú a fondo" (deep learning). Usando "large NN" si arriva a trattare $\sim O(10^6)$ features.





Alcune ANN (es. computer vision) arrivano ad avere $O(10^6)$ features «*complete chromosomes are too large to fit in state-of-the-art artificial neural networks, but they could fit in an 80 billion neuron network*»(Dr. Decebal Mocanu, 06/2018).

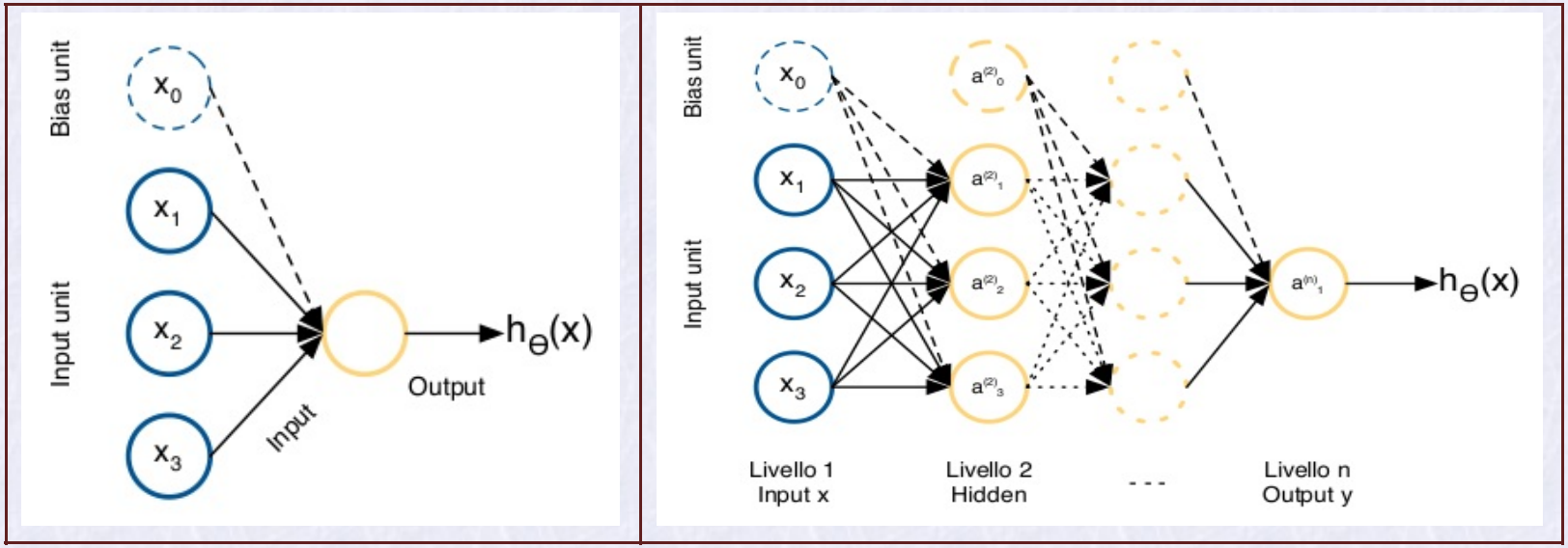


Figura 2. $x_0 = 1$ è la bias unit; $h_\theta(x) = \frac{1}{1+e^{-\theta' x}}$; $g(z) = \frac{1}{1+e^{-z}}$; $h_\theta(x) = g(\theta' x)$

7.1 Forward propagation

$$\begin{bmatrix} a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{bmatrix} = g \circ \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} & \theta_{1,4} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} & \theta_{2,4} \\ \theta_{3,1} & \theta_{3,2} & \theta_{3,3} & \theta_{3,4} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} g(\boldsymbol{\theta}_1' \boldsymbol{x}) \\ g(\boldsymbol{\theta}_2' \boldsymbol{x}) \\ g(\boldsymbol{\theta}_3' \boldsymbol{x}) \end{bmatrix}$$

$$\boldsymbol{a}_2 = g \circ \Theta_2 \boldsymbol{x}$$

Esprime il passaggio dagli input al primo layer, e analogamente per passare da un layer al successivo.

7.2 Training

Serve per trovare i pesi $\theta_{i,j}$ usando gli N esempi $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ che minimizzano la funzione costo: per una ANN con n ingressi (features), m uscite (classi), L layers,

s_l neuroni per ciascun livello (**nota**: $s_1 = n,\ s_L = m$)

$$J(\theta) = \frac{-1}{N+1}\left[\sum_{i=1}^{N+1}\sum_{k=1}^{K} y_{k,i}\,\ln(h_{\theta_k}(x_i)) + (1-y_{k,i})\,\ln(1-h_{\theta_k}(x_i))\right] + \frac{\lambda}{2\,(N+1)}\sum_{l=1}^{L-1}\sum_{i=0}^{s_l}\sum_{j=0}^{s_{l+1}}(\theta_{j,i,l})^2$$

Per trovare un minimo si usano metodi iterativi, per i quali occorre calcolare il gradiente. Per farlo si ricorre ad un algoritmo detto **Back propagation**.

Note1

- Il training può richiedere moltissimi esempi ($\rightarrow$ big data)
- Importante implementare in modo efficiente gli alg. Es. esprimere i calcoli in forma matriciale ($\sum_i \sum_j a_{i,j}\, b_i = A\,\boldsymbol{b}$). Evitare di implementarli con cicli `for` nidificati.

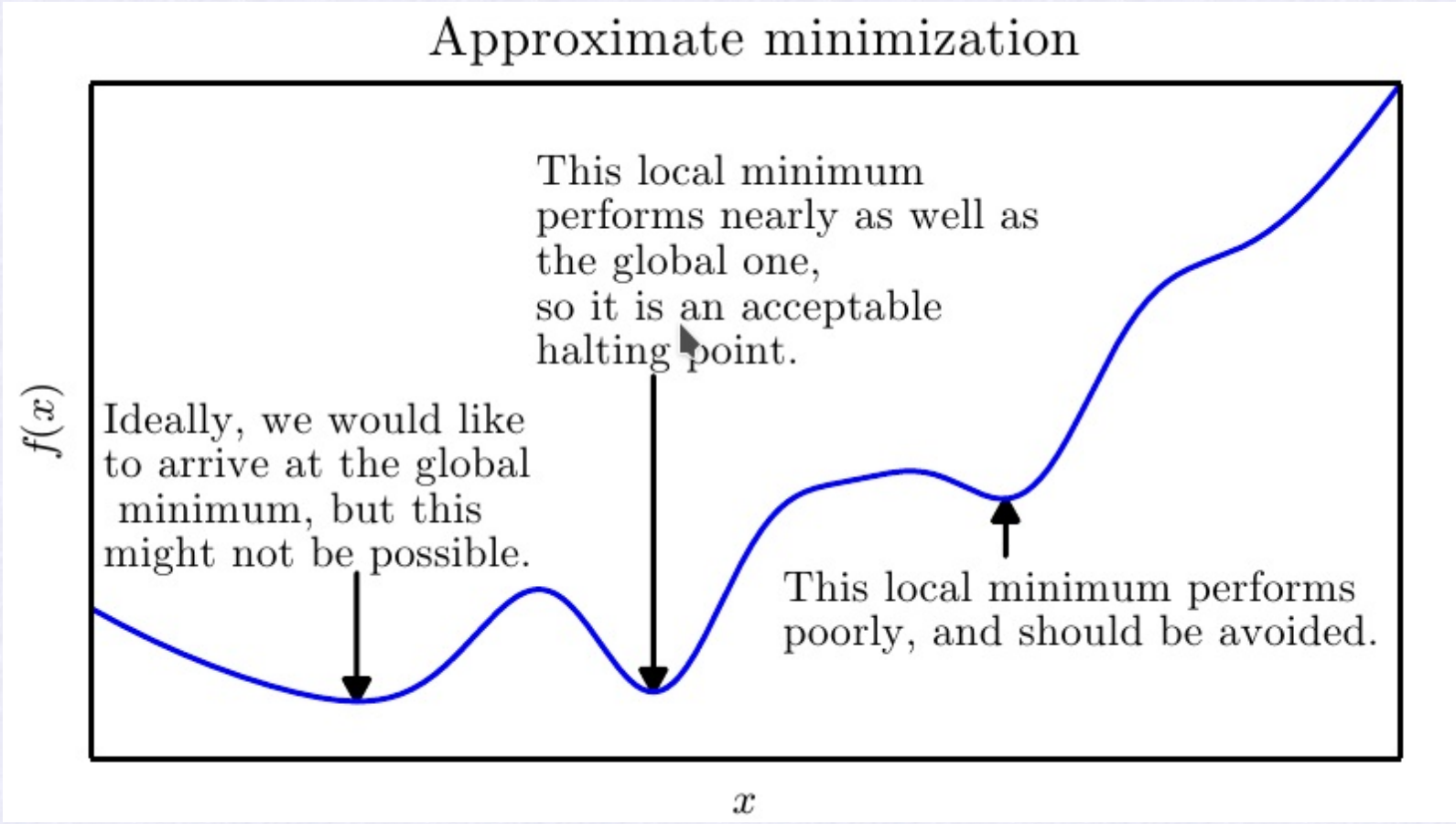


Figura 3. Le ANN

Note2 Alcuni accorgimenti empirici:

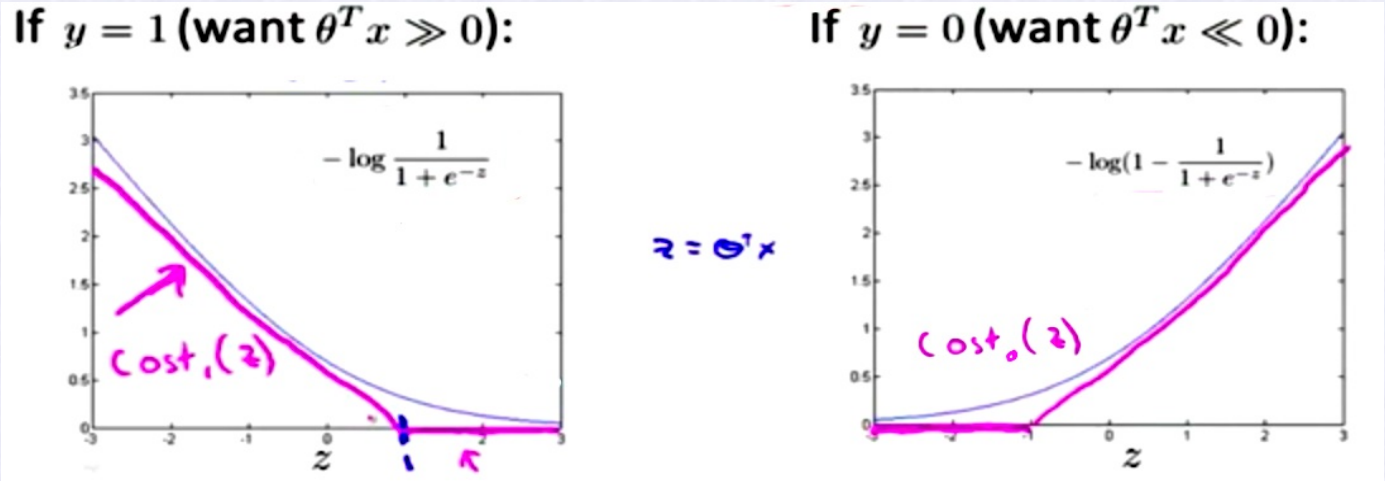
- Non conviene inizializzare i $\theta_{i,j}$ a 0 (rende nulli molti termini del gradiente), meglio assegnare dei valori random $\theta_{i,j} \in [-\varepsilon, \varepsilon]$.

- Tutti gli hidden layers dovrebbero avere lo stesso numero di neuroni
- Num. di Hidden Layers da 1 a 4, non di piú (training lento, guadagno trascurabile).

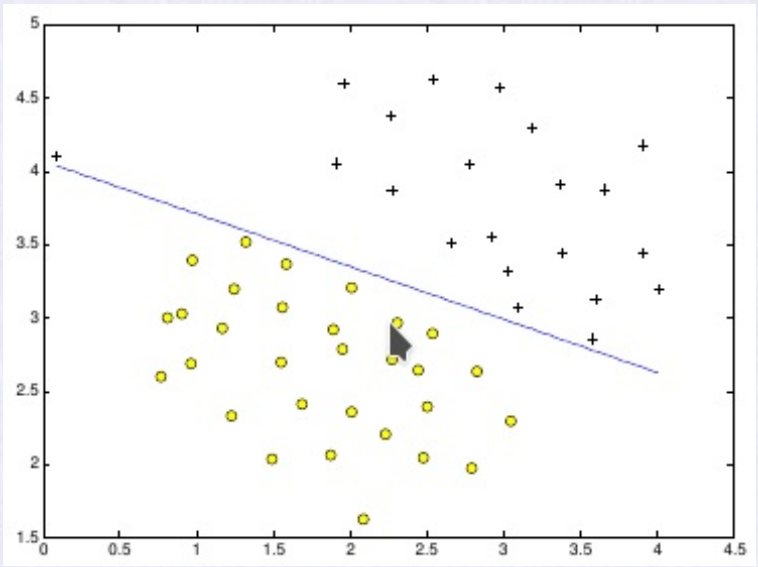
8 Support Vector Machine

È tra i piú potenti algoritmi classificatori "classici".

$$\begin{aligned} \mathrm{Cost}(h_\theta(x), y) &= \begin{cases} -\ln(h_\theta(x)) & , \quad y = 1 \\ -\ln(1 - h_\theta(x)) & , \quad y = 0 \end{cases} \\ h_\theta(x) &= \frac{1}{1 + e^{-\boldsymbol{x}'\boldsymbol{\theta}}} \end{aligned}$$



$$J(\theta) = \frac{-1}{N+1}\left[\sum_{k=1}^{N+1} y_k \,\mathrm{Cost}_1(\theta' \boldsymbol{x}_k) + (1 - y_k)\,\mathrm{Cost}_0(\theta' \boldsymbol{x}_k)\right]$$



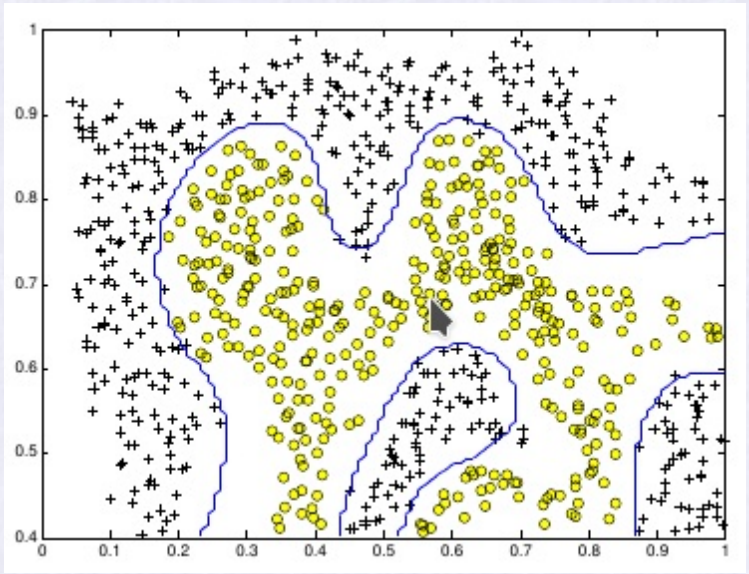


Figura 4. Il decision boundary separa le due classi col piú ampio margine possibile. Usando i **kernel** si possono ottenere decision boundary non lineary.

Si dimostra che $J(\theta)$ ha **minimo unico**, la cui ricerca si può ridurre ad un problema di programmazione quadratica di dimensione pari a quella di $\boldsymbol{\theta}$, per cui la complessità computazionale non cresce con N come avviene con le ANN.

8.1 Esempio SVM: spam detector

È necessario pretrattare i messaggi del training set:

1. Convertire ogni mail in un array di features:
   - tutto a lowercase, solo parole separate da un solo spazio.
   - eliminare tag HTML e segni d'interpunzione.
   - Gli url sono sostituiti con "httpaddr", le email con "emailaddr",$ con "dollar".
   - word stemming: via suffissi e prefissi
2. Si usa un dizionario con le parole piú comuni
3. ad ogni mail si associa un array $\boldsymbol{x}$ e si pongono a 1 gli indici corrispondenti a parole trovate nel dizionario. Alla fine si ottengono per ogni messaggio msg_i:

$$\boldsymbol{x}_i = [0, ..., 1, 0, ..., 1, ...]' \in \{0, 1\}^F, \; F \simeq 2000$$

9 Valutazione e Debugging di un algoritmo ML

Quando il "learning" non va bene (errori di stima inaccettabili) è importante capire come intervenire. Si deve fare una "diagnosi" del problema.

Es. (vale analogo per ANN) si implementa una *regressione lineare regolarizzata*:

$$J(\theta) \;=\; \frac{1}{2\,N}\left[\sum_{i=1}^{N}\,(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{N}\,\theta_i^2\right]$$

ma si vede che fornisce stime inadeguate su dati nuovi. Cosa si può fare?

1. Aggiungere training examples (N: piú dati!)
2. cambiare il grado del polinomio: $x_1^2, \ldots, x_1^2\,x_2, \ldots$ (ordine del modello)

3. Aggiungere o ridurre features ($h_\theta(\cdot)$: modificare il modello)

4. variare λ

Per capire come intervenire si usano **test diagnostici**.

9.1 Training Set, Test Set, Cross Validation Set

Per prima cosa si individua l'ordine del modello migliore.

- Si dividono gli esempi in tre sottogruppi: **Training Set** (70%), **Cross Validation Set** (10%) e **Test Set** (20%).

- Il CV Set si usa per scegliere l'ordine del modello e per il tuning di λ. In pratica si fa il training di diversi modelli usando il Tr. Set, e si confrontano usando il CV Set.

- Il Test Set si usa per verificare che il modello scelto si comporti bene anche con dati "nuovi". Idealmente, $J_{\text{test}}(\theta) \simeq J_{\text{train}}(\theta)$.

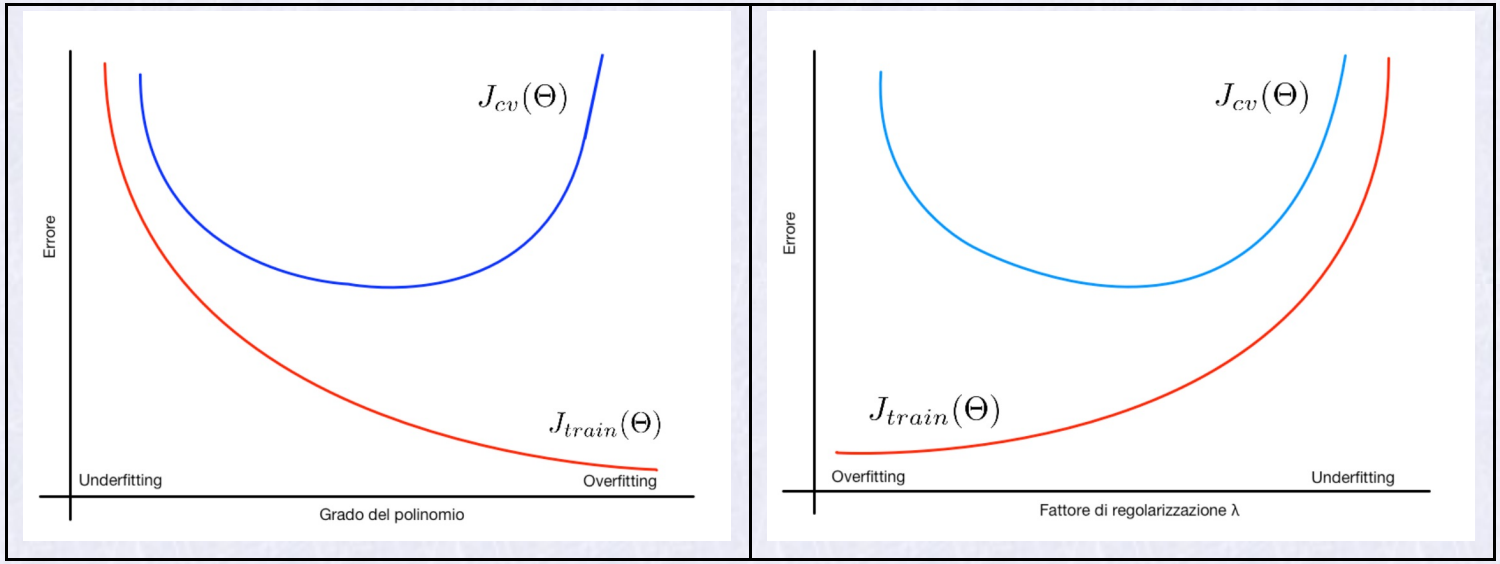

$J_{cv}(\Theta)$
$J_{train}(\Theta)$
Errore
Underfitting
Overfitting
Grado del polinomio
$J_{cv}(\Theta)$
$J_{train}(\Theta)$
Errore
Overfitting
Underfitting
Fattore di regolarizzazione λ

Tabella 1.

9.2 Learning Curves

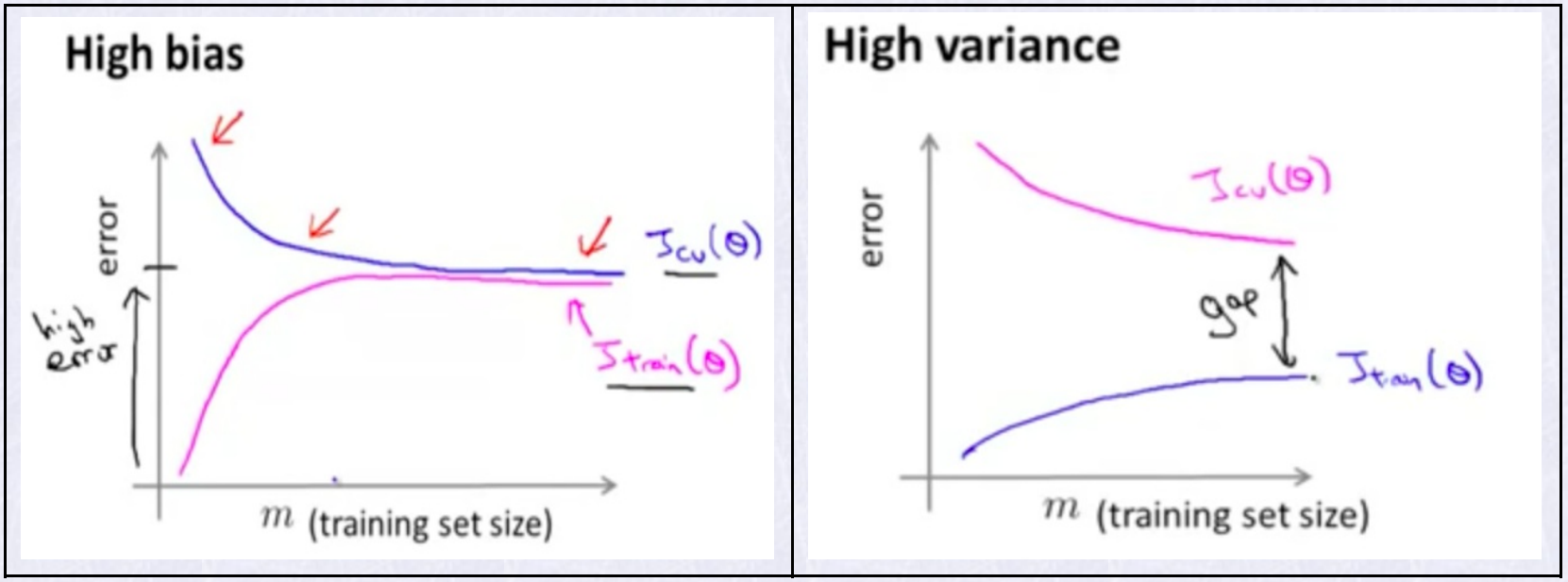


Tabella 2.

10 Clustering

Il Clustering è un esempio di **Unsupervised Learning**. Da un insieme di unlabeled data si vogliono individuare i raggruppamenti (cluster) a cui appartengono.

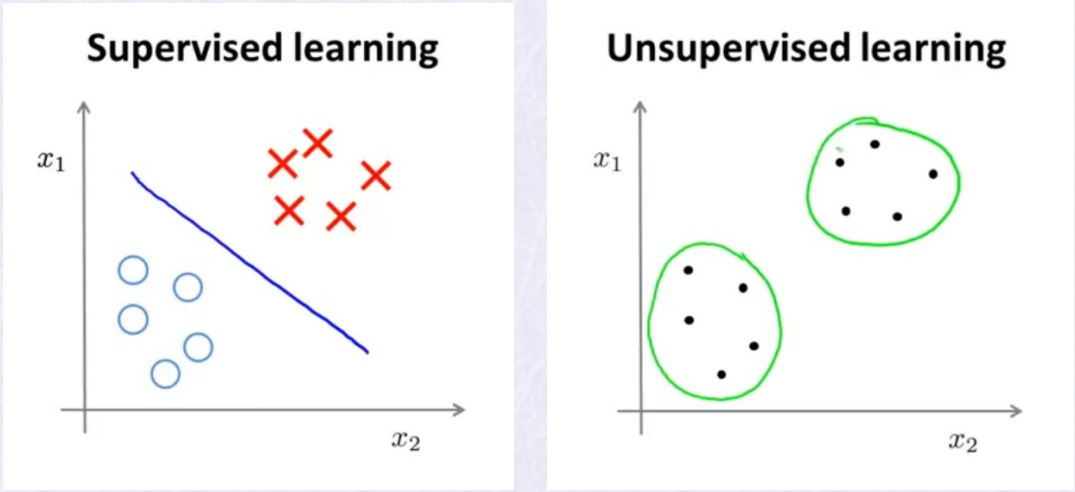


Tabella 3.

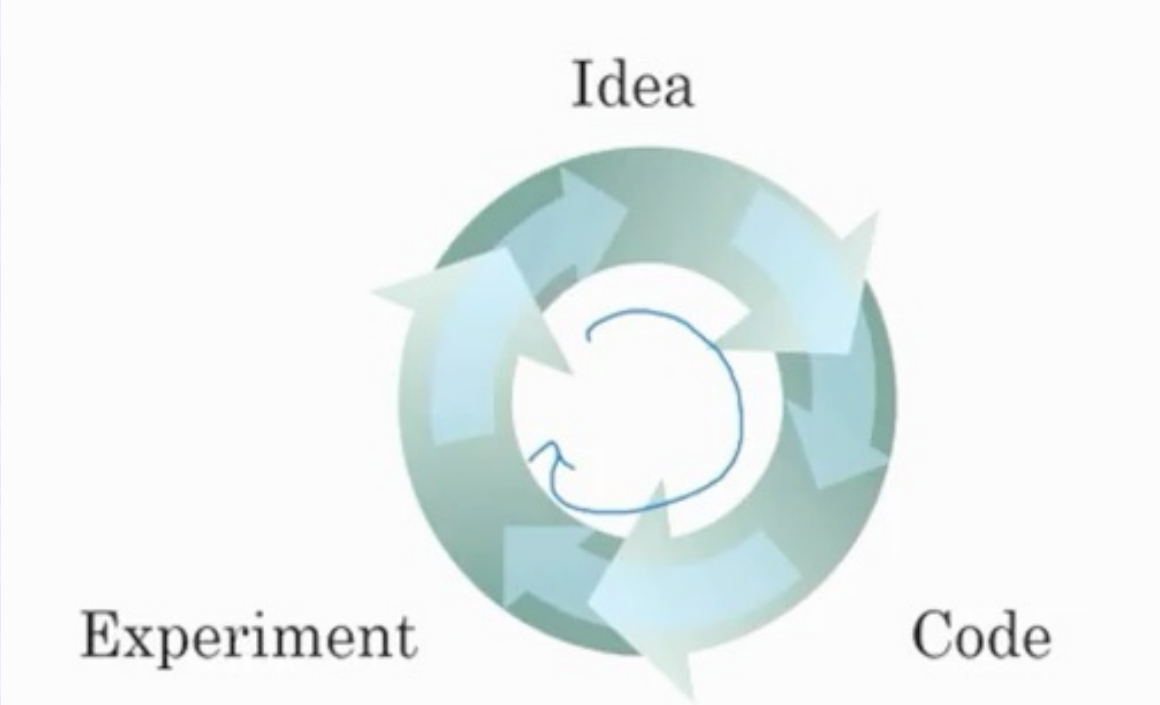
K-Means è l'algoritmo piú comunemente usato: fissato $k =$ numero di classi, si definiscono k "centroidi" inizializzati random e si assegna ogni punto (esempio) al controide piú vicino. Si calcolano i baricentri dei gruppi cosí ottenuti, che diventano i nuovi centroidi, e si ripete il processo finché non cambiano piú.

11 Linee guida per progettare un sistema ML

Scelta delle features. fare una lista piú possibile completa delle possibilità, e decidere quali sono le piú pratiche da usare.

Implementazione "Quick & dirty". Si implementa un algoritmo semplice, lo si allena e lo si valuta sul CV Set.

Tuning del modello. Si analizzano le learning curve per capire se vanno aggiunte features, se occorrono piú training examples etc.



11.1 Metriche di valutazione

Per una binary logistic regression, si considera la **confusion matrix** CM:

		classe reale: 0	classe reale: 1
classe stimata	0	#Vero Negativo	#Falso Negativo
	1	#Falso Positivo	#Vero Positivo

Accuracy. $\mathrm{acc} = \frac{\#\mathrm{ClassificatiOK}}{\#\mathrm{Classificati}} = \frac{\mathrm{CM}_{0,0} + \mathrm{CM}_{1,1}}{\sum \mathrm{CM}_{i,j}}$

Nota: non va bene per eventi rari. Es: "dire se un neo è maligno in base alla dimensione"; succede in un caso su 1000. Faccio uno stimatore che dice sempre "benigno" $\rightarrow \mathrm{acc} = 99.9\%$.

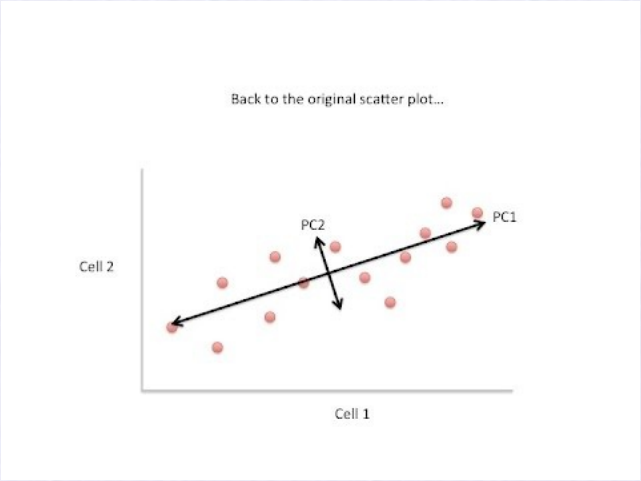
Precision. $\mathrm{Precision} = \frac{\mathrm{CM}_{1,1}}{M_{0,1} + M_{1,1}}$;

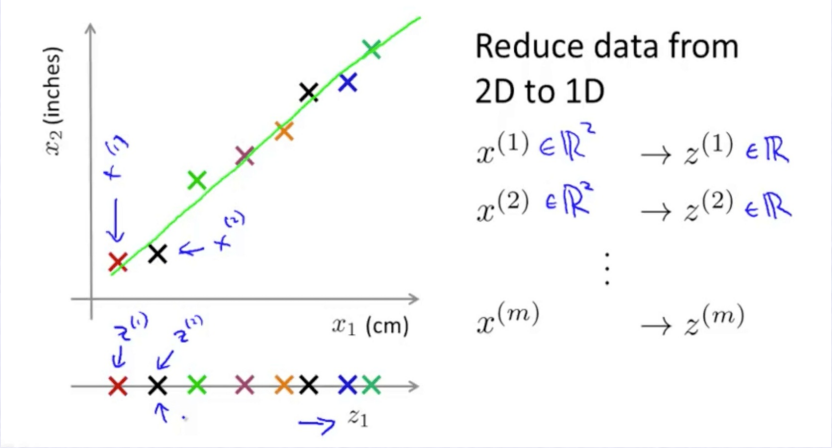
Recall o True Positive rate. $\mathrm{Recall} = \frac{\mathrm{CM}_{1,1}}{M_{1,0} + M_{1,1}}$;

Per un classificatore multiclasse (es. OCR) la $\mathrm{CM} \in \mathbb{N}^{11\times 11}$, sulla diagonale i numeri di classificazioni corrette, $\mathrm{CM}_{n,k}$ numero di "riconosciuto n ma era k".

12 Principal Component Analisys (1901, Karl Pearson)

Tecnica utile per "data compression" e per velocizzare il training.





Idea di fondo: **Se** i dati del nostro dataset sono punti che stanno $\sim$ tutti su una retta, possiamo conservare solo i valori x_i e poi ricostruire gli y_i come $y_i = a\, x_i + b$. Passiamo da $x_i \in \mathbb{R}^2$ a $z_i \in \mathbb{R}$. Abbiamo due possibili vantaggi:

1. Riduciamo dimensione dataset (in ambito bigdata può essere molto comodo)

2. Possiamo fare il training a un modello piú piccolo usando il dataset ridotto.

In generale, può essere possibile passare da $x_i \in \mathbb{R}^m$ a $z_i \in \mathbb{R}^k$ con $k < m$. Spesso si può avere $k \sim \frac{m}{10}$. Servono alcuni passaggi:

i. mean normalization e feature scaling. Per ogni feature:

$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}$; $x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$; s_j scelto in modo da avere range confrontabili tra le features.

ii. Si usa la **SVD** (Singular Value Decomposition) sulla *Matrice di covarianza* $\Sigma \in \mathbb{R}^{m \times m}$

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_k \boldsymbol{x}_k'$$
$$[U, S, V] = \text{svd}(\Sigma)$$

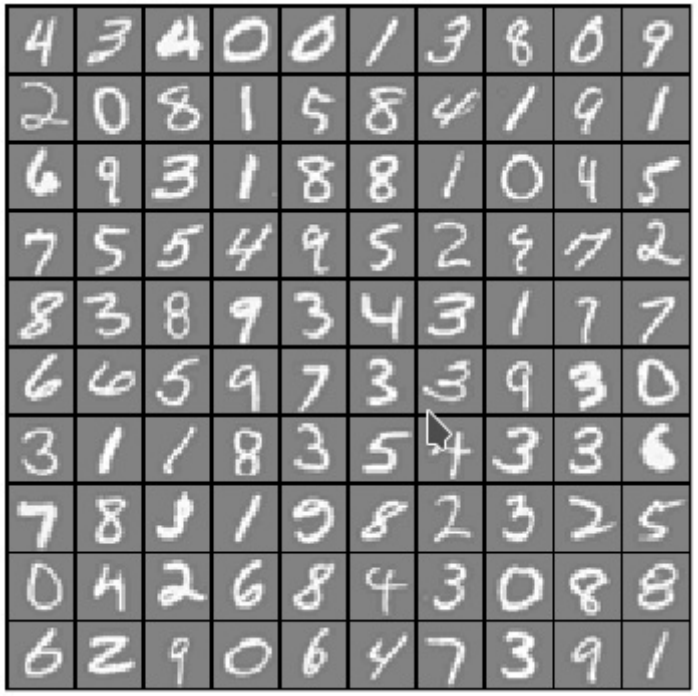
iii. Si sceglie il piú piccolo valore k per cui p.es.

$$\frac{\sum_{i=1}^{k} S_{i,i}}{\sum_{i=1}^{m} S_{i,i}} \geqslant 0.99$$

iv. $\boldsymbol{z} \leftarrow U_r' \boldsymbol{x}$, dove U_r è fatta prendendo le prime k colonne di U.

Esempio

Con Octave, caricando un dataset del mnist (5000 digit rappresentate su bitmap 20×20px)



```
>> load('mnistdata.mat');
```

```
>> size(X) %contiene 5000 bmp 20x20

ans =

   5000    400

>> MC=X'*X;  %Covariance Matrix

>> [U,S,V] = svd(MC);

>> sv = diag(S);

>> sum(sv(1:128))/sum(sv)

ans =  0.99018

>> Z = X*U(:,1:128); %spazio ridotto: 400 to 128

>> size(Z)

ans =
```

```
5000     128

>> x = X(1,:); z = Z(1,:);

>> x1= z*(V'(1:128,:)); %ricostruzione da z
```

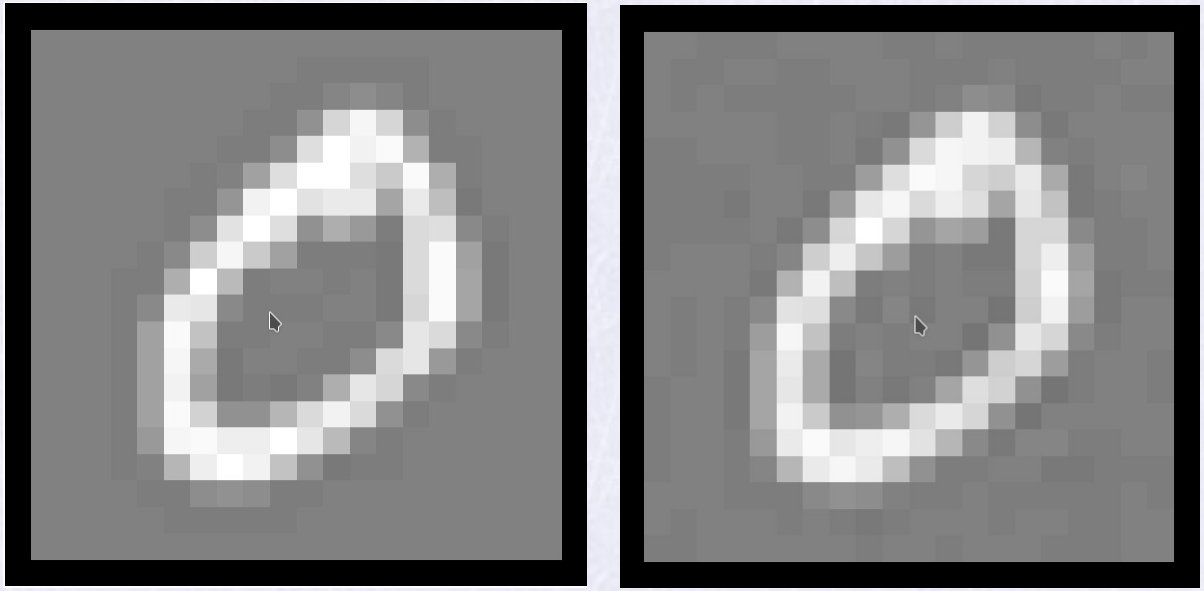


Figura 5. Dato originale (`x=X(1,1:400)`) e `x1`, ricostruito da PCA (`z=Z(1,1:128)`).

Possiamo fare il training del classificatore usando esempi (z_i, y_i) invece di (x_i, y_i). Riducendo cosí la dimensione dei dati e del modello.

13 Anomaly detection

Riscontrare quando un sistema assume un comportamento inatteso (improbabile).

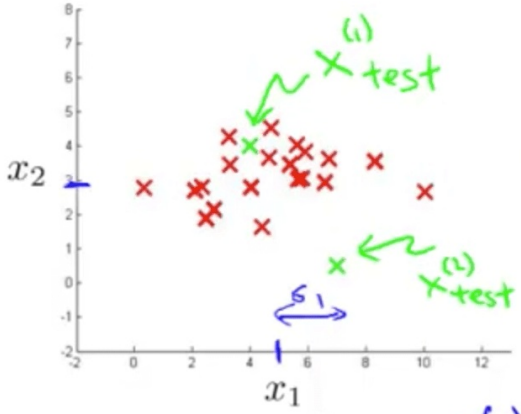
Esempio in una farm di WN individuare quelli in condizioni di lavoro "anormali".

- **Scegliere** n features $x_i \in \mathcal{N}(\mu_i, \sigma_i)$ che possono essere indicative di anomalie (es. HDD e cpu temp, fan rpm, ...)
- Stimare statistiche: $\mu_i = \frac{1}{N}\sum_{k=1}^{N} x_{i,k}$; $\sigma_i^2 = \frac{1}{N}\sum_{k=1}^{N} (x_{i,k} - \mu_{i,k})^2$

$$p(x) = \prod_{k=1}^{N} p(x_k; \mu_k, \sigma_k^2) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi}\,\sigma_k} e^{-\frac{(x_k-\mu_k)^2}{2\sigma_k^2}}$$

- Anomalia se $p(x) < \varepsilon$.

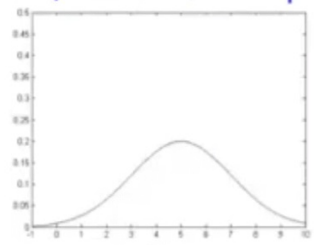
Anomaly detection example

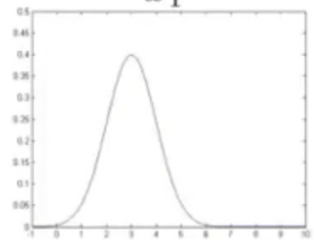


$\mu_1 = 5, \sigma_1 = 2$

$\mu_2 = 3, \sigma_2 = 1$

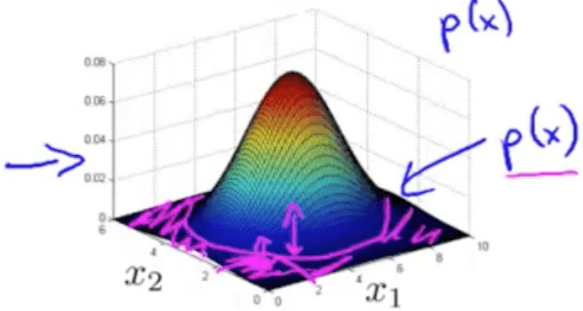
$\rightarrow p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2)$





$p(x_1; \mu_1, \sigma_1^2)$

$p(x_2; \mu_2, \sigma_2^2)$



$\varepsilon = 0.02$

$p(x_{test}^{(1)}) = 0.0426 \geq \varepsilon$

$p(x_{test}^{(2)}) = 0.0021 < \varepsilon$

14 Conclusioni: Bigdata + ML Tools + Know How...

15 Riferimenti

- https://work.caltech.edu/telecourse.html

  Corso completo con video, slides, homework; molto chiaro, serve un po' di background di matematica.

- http://www.deeplearningbook.org/
- https://intelligent-optimization.org/LIONbook/
- Machine Learning — Andrew Ng, Stanford University [coursera o youtube]
- https://phys.org/news/2018-06-ai-method-power-artificial-neural.html

16 Ringraziamenti

D. Bonacorsi, B. Martelli

e tutti i colleghi, per sostegno e partecipazione.