Machine Learning Algorithm in grado di apprendere come risolvere uno specifico problema a partire da un set di dati

Computare area di un poligono

Rombo $D_p \cdot D_s / 2$ Dati → Algoritmo → output

Contare il n. di occorrenze di una parola in un testo

∃ Problemi con grado di incertezza → Prob. molto difficile da modellare → Classificare un'imm. per stabilire quale oggetto è presente

Distinguere EMAIL → SPAM / NON SPAM → ⚠ incertezza!

Alta incertezza, non riusciamo a rappr. esaustivi per risolvere problema

Guardare Contenuto EMAIL → SPAM / → NON SPAM → Non vi è sempre nesso logico

DATI INPUT → Algoritmo ⇒ OUTPUT

DATI INPUT, OUTPUT → Algoritmo di Learning → Programma

⚠ Problema del learning

relativo ad obiettivo

Un computer apprende da un'esperienza E rispetto ad un task T ed una misura di performance D, se le performance P misurate su E migliorano con l'esperienza rispetto ad un task T

TASK: Problema da risolvere → Modello

↳ Funzione Parametrizzata

↳ h

valutato su

ESPERIENZA: DATI

→ misura che valuta modello sfruttando dati in input

PERFORMANCE: $P\left(\{y^{(i)}\}_{i=1}^{m}, \{\hat{y}^{(i)}\}_{i=1}^{m}\right)$ → Val. alto → ⚠ Buona performance → Generalizzate

→ Basso → ⚠ Cattive performance

SPAM $y^{(i)}$

$h(x^{(i)}) = \hat{y}^{(i)}$

$p \in [0,1]$

$1-p$ ⇒ Misure di Errore

Prob. Machine Learning → Modelli statistici che DIPENDONO dal task

Feature → Espressione di attributi — Categoriche → Polinomiali / Ordinali; Continue → numeri

Un esempio $x = (x_1, \dots, x_d)$

$x^{(i)}$ → i-esimo el. set di dati

$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)})^T$ → vettore riga in colonna → operiamo su questo

$[\quad]$

PONGO IN LUCE CARATT. SALIENTI DEI DATI TRASCURANDO ALTRI

Estrazione Feature

$f(x) = \bar{x}$ — Creata a mano / Appresa ⇒ $\bar{x}^{(i)}$

RAPPRESENTAZIONE

NON ESISTE RAPPR. UNIVERSALE

A partire dai dati → Estrarre caratt. importanti rispetto al task da risolvere

Tipi di Task → regressione $x \in \mathbb{R}^d$, $y \in \mathbb{R}$

classificazione

$x \in \mathbb{R}^d$ $y \in \{1, \dots, k\}$ k n. classi predefinito

$h_\theta(x) = y$ h funzione: $\mathbb{R}^d \to \{1, \dots k\}$

TIPI di Learning

Supervised Learning
↳ $x \in \mathbb{R}$, $y \in A$

Unsupervised Learning
$x^{(i)} \in \mathbb{R}$

Reinforcement Learning

Provare alternative

Stimare se ALGO buono per ottenere il mio scopo

ACCURACY → CLASSIFICAZIONE

→ % di risposte corrette date in output dal mio algoritmo di ML

Good - Threat

REGRESSIONE → ERRORE

ESPERIENZA = DATI — colonne attributi = Feature

$$X = \begin{bmatrix} x^{(m)} = x_1^{(m)} & \dots & x_d^{(m)} \\ x^{(1)} = x_1^{(1)} & & x_d^{(1)} \end{bmatrix} \quad x^{(1)}$$

$x_j^{(i)}$

Set di dati → DATASET → Design Matrix

$x \in \mathbb{R}^{m \times d}$

Y = MATRICE $\in A^{m \times k}$

$A = \{1, \dots, k\}$ $k = 1$

Learning / training del modello

Prende in input BANCA DATI

MODELLO (Parametri)

Misure di Performance

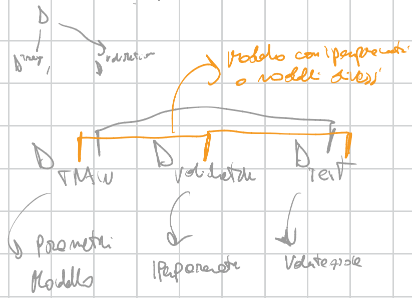
Output

Set di Parametri

che producono le migliori performance su D

Valutazione

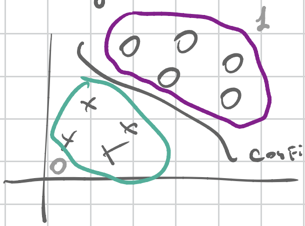
Generalizzazione



CLASSIFICAZIONE

DISCRIMINATIVA | GENERATIVA

Vista geometrica



confine di decisione → Definire stimando queste probabilità

$x \in \mathbb{R}^d$ $\quad y = \{0,1\}$

$P(y|x)$

Regola di Bayes → $P(x|y)P(y) = P(x,y) = P(x)P(y|x)$ $\qquad P(x,y)$

Ingredienti
- Modello
- Loss function (Misura errori)
- | Algoritmo di Learning
- Evaluation

Semplice macchina computazionale → Neurone

1 Modello

Modello McCulloch-Pitt 1943

Mimare neurone

Unità computazionale che riesce a prendere in input binari

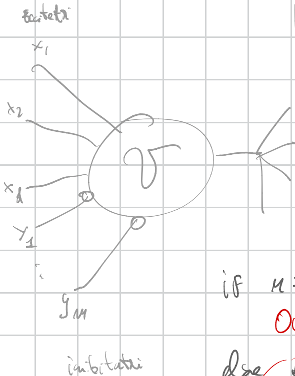
Modello ritorna output binario

Ogni neurone è associato ad un parametro ϑ (theshold)

Output può essere connesso ad altri neuroni (anche più di uno)

Input può provenire da altri neuroni o può essere vettore binario o una computazione

Due tipi di input: Eccitatorio / Inibitorio



se inibitore acceso => output = 0 ACUinetti

if $u \ge 1$ AND $\exists y_j \mid y_j = 1$ then

Output 0

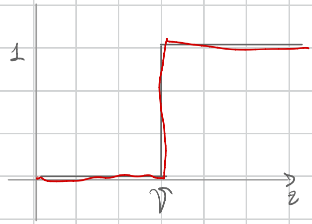
else $\sum_{i=1}^{d} x_i \ge \vartheta$ then

Output 1

else

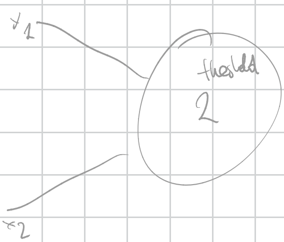
Output 0

⇒



Valori non binari in binari

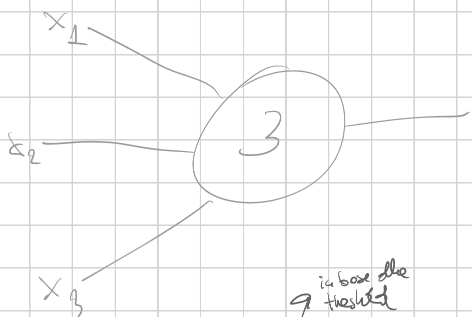
Dobb. cambiare per output



AND

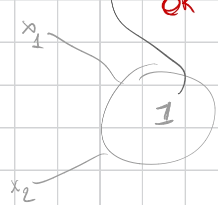
$x_1 + x_2 \ge \vartheta$

x_1	x_2	output
0	0	0
0	1	0
1	0	0
1	1	1



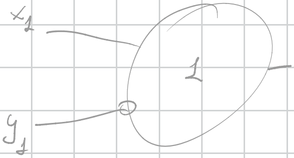
in base alla theshold

OR



Modello Parametrizzato

Funzioni lineari

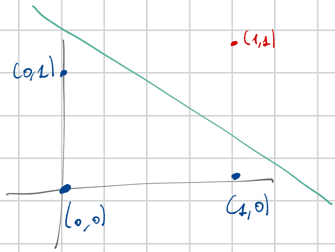


Valori soglia + inibitori = parametri

Limiti Modello

- Pensato per ricevere input binari
- Pensato solo per funzioni logiche/input binari
- Parametri fissati a mano → No procedure di learning
- Non esistono pesi
- Tutte le funzioni sono linearmente separabili → funzione AND separiamo con classi dove possiamo mettere su asse binario

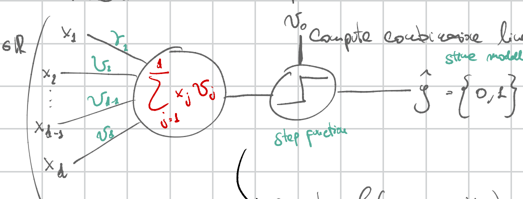
0	0	0
0	1	0
1	0	0
1	1	1



Percettrone

q abbiamo tolto il peso
↳ lo portiamo come input

v_0 Compute combinazione lineare del resto input



Compute $f(x_1, \dots, x_d) = \begin{cases} 1 & \text{se } \sum v_j x_j > v_0 \\ 0 & \text{ALTRIMENTI} \end{cases}$

$x_0 = 1,\ x_1^{(1)}, \dots,]$

$[v^T x > 0] = [\ v\ ]\ [\ X\ ]$

decision boundary

iperpiani

di una dimensione del resto input

$$v_0 + x_1 v_1 + x_2 v_2 = 0$$

→ eq. retta

$$x_2 = \boxed{\frac{v_1}{v_2}}\, x_1 - \frac{v_0}{v_2}$$

$-0,55$

$y = mx + q$

Come faccio a fissare i parametri?

$\sum x_j v_j > 0$ $\quad \bar{y} = 0 \quad \hat{y} = 1$ Come posso sistemare i pesi?

peso

$x \in \mathbb{R} \to x \in \{0,1\}$ Decremento i parametri

$y \in \{0,1\}$

$y = 1 \quad \hat{y} = 0$

Incremento i parametri

Errore

Segno di [illegible]

Cambio!

Agg pesi

incrementare

$$v_j \leftarrow v_j + (y - \hat{y}) x_j$$

quantità più arbitraria in $\{0,1\}$

Formalizzazione

Conosciamo valore x aggiustare pesi

Inizializzare i pesi random in $[0,1]$

Considera $x^{(i)}$ e compute $\hat{y}^{(i)}$ $\forall i$

Per le predizioni errate aggiusta i pesi

$y = 0 \quad \hat{y} = 1$ Decrementa

$y = 1 \quad \hat{y} = 0$ incrementa

$$v_j \leftarrow v_j + (y - \hat{y}) x_j$$

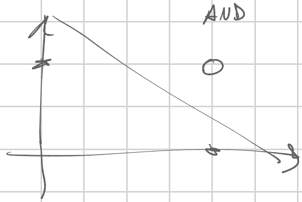
Ritorna al passo 2 fino a convergenza

$[f_v(x) > 0]$

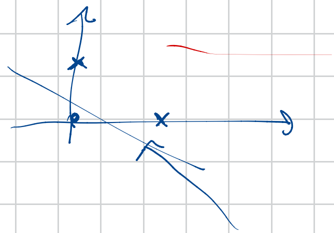
XOR

x_1	x_2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

AND



XOR



$v_0 + x_1 v_1 + x_2 v_2 > 0$

$x_1 v_1 + x_2 v_2 > -v_0$ → δ

$0 v_1 + 0 v_2 < t \Rightarrow t > 0$

$0 v_1 + 1 v_2 > t \quad v_2 > t > 0$

$1 v_1 + 0 v_2 > t \quad v_1 > t > 0$

$1 v_1 + 1 v_2 < t \quad v_1 + v_2 < t$

$v_1 > 0 \quad t > v_1 + v_2 > 2t$

$v_2 > 0$

input

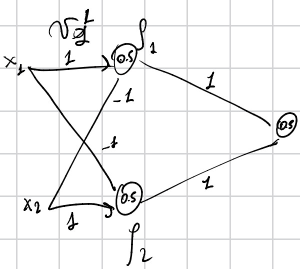
x_1	x_2	f_1	f_2	f_3
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

Riesce a risolvere il problema XOR

$v_0^1 = 0.5$

$v_1^1 = 1$

$v_2^1 = -1$



$f_1(x_1 x_2) = [x_1 - x_2 > 0.5]$ 0 / 1

$f_2(x_1,x_2) = [x_2 - x_1 > 0.5]$ 0 / 1

$f_3(f_1(x_1 x_2), f_2(x_1,x_2)) = [1 f_1(x_1,x_2) + 1 f_2(x_1,x_2)]$

[illegible]

Riusciamo a computare la XOR

$v_0^2 = 0.5$

$v_1^2 = -1$

v

Modello con 3 neuroni riesce a risolvere problema XOR

Problema XOR rimane (backpropagation) → non sappiamo come allenare

Regressione

- Modello
- Loss Function → Aggiustare pesi
- Algoritmo di Learning
- Evalutation

$$\{y^{(i)}\}_{i=1}^{m} = Y_{TEST}$$

m → punti campioni del dataset

→ insieme del dataset → campioni

$$y^{(i)} = \begin{bmatrix} y_1^{(i)} \\ \vdots \\ y_{n'}^{(i)} \end{bmatrix}$$

TASK Regression

$$f_v : \mathbb{R}^n \to \mathbb{R}^{n'}$$ ⇒ vettore in input, vettore in output

v → parametri

DATI → TRAINING, VALIDATION, TEST

$\mathbb{R}^n$ $\mathbb{R}^{n'}$
x y

Mean Squared Error → Distanza Euclidea

$$\hat{Y}_{TEST} = \{f_R(x^{(i)})\}_{i=1}^{m}$$

$$\hat{y}^{(i)} = \begin{bmatrix} \hat{y}_1^{(i)} \\ \vdots \\ y_{n'}^{(i)} \end{bmatrix}$$

$$\|\hat{y}^{(i)} - y^{(i)}\|_2 = \sqrt{\sum_{j=1}^{n'} (\hat{y}_j^{(i)} - y_j^{(i)})^2}$$

$$ERROR(Y_{TEST}, \hat{Y}_{TEST}) = \frac{1}{m}\sum_{i=1}^{m}\|\hat{y}^{(i)} - y^{(i)}\|_2$$

↓ media errore che commetto

MSE
DISTANZA RADICE QUADRATA
Voglio far pesare di + ERRORI GRAVI

$$f_v : \mathbb{R}^n \to \mathbb{R}^{n'}$$

- $n = n' = 1$ → Simple Regression → linee
- più input: $n > 1$ AND $n' = 1$ Multiple regression → piano
- $n \geq 1$ AND $n' > 1$ Multivariate Regression → insieme più predizioni linear regression

Vale per tutti

$f_v(x) = y$
$f_v(x) = y$

Modello Linear Regression

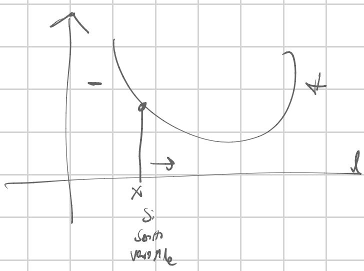
Loss Function

$$f_v(x) = v_0 + x_1 v_1 + \dots + x_n v_n$$

$$J(v) = \frac{1}{2}\sum_{i=1}^{m}\left(f_v(x^{(i)}) - y^{(i)}\right)^2$$ DIFFERENZIABILE

training

$$\hat{v} = \arg\min_{v} J(v)$$ → apprendo che minimizzo perdita



$v_1^{(0)} =$

USIAMO SLOPE e regressione

$$\frac{\partial J(v)}{\partial v_j} = \sum_{i=1}^{m} (v_0 + x_1 v_1 + \dots + x_n v_n - y) x_j = \sum_{i=1}^{m} (f_v(x^{(i)}) - y^{(i)}) x_j$$

$$v_j \leftarrow v_j - \gamma \sum_{i=1}^{m} (f_v(x^{(i)}) - y^{(i)}) x_j^{(i)}$$