Apprendimento e Predizione On-Line

Claudio Biancalana

Piano del discorso

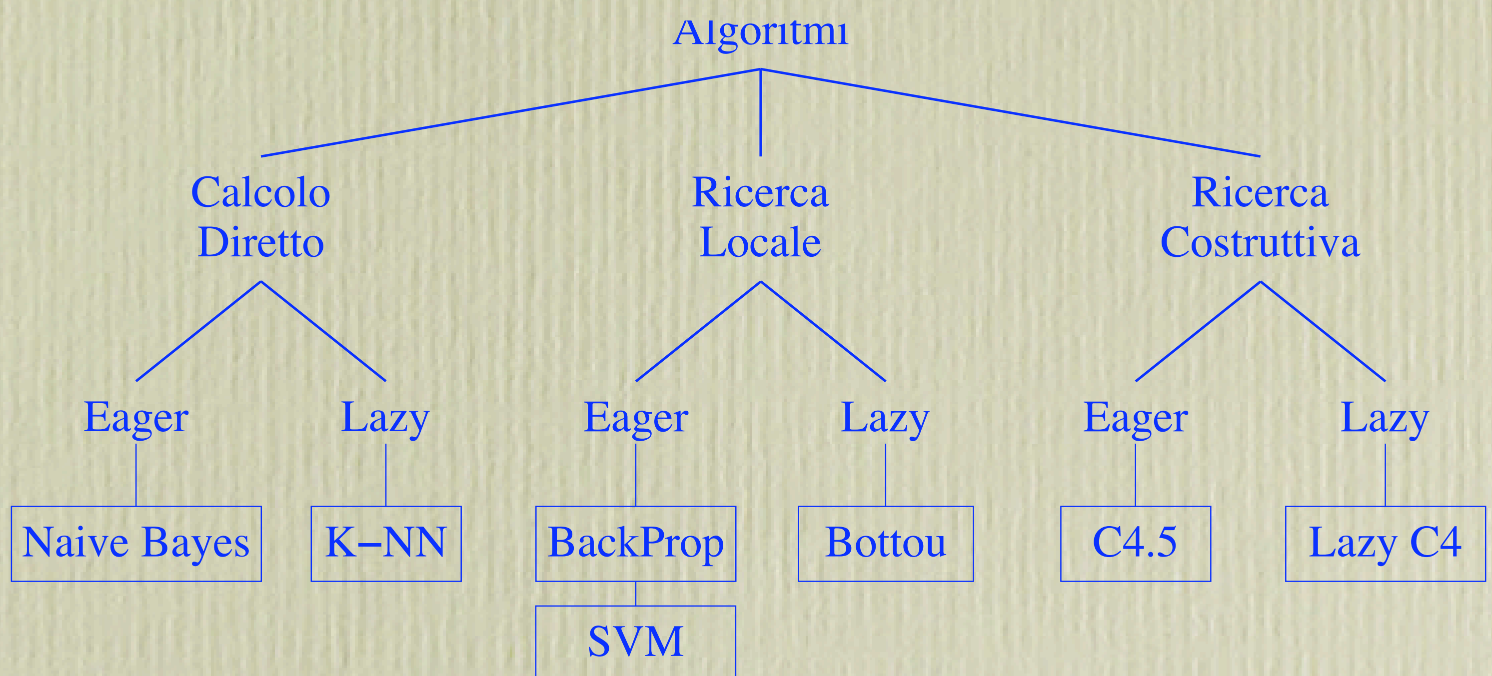
- Algoritmi On-Line per il Machine Learning
- Expert Advice
  - Halving Algorithm
  - Weighted Majority
  - Weighted Majority Randomized
  - Winnow Algorithm
  - Perceptron Algorithm

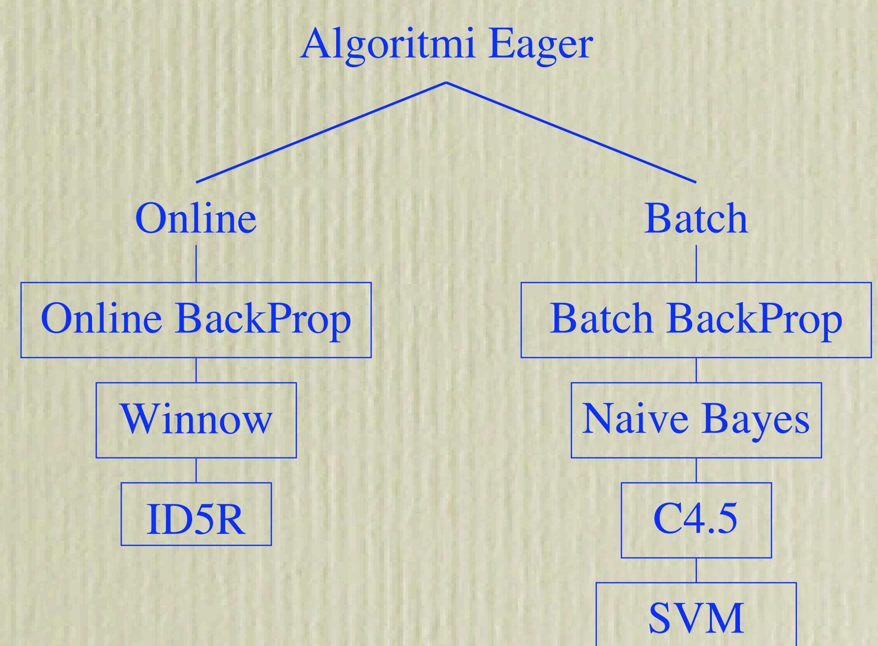
Machine Learning

Apprendimento delle Macchine

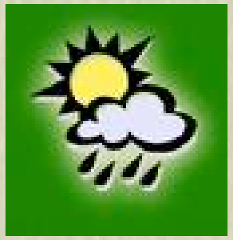
- Learning is constructing or modifying representations of what is being experienced [Michlaski, 1986]
- Apprendere = migliorare la capacità di esecuzione di un certo compito, attraverso l'esperienza
  - Migliorare nel task T
  - Rispetto ad una misura di prestazione P
  - Basandosi sull'esperienza E
    - E = esempi di comportamenti "positivi" o "negativi" forniti da un istruttore, oppure un sistema di "ricompense".

Tassonomia Algoritmi ML





Expert Advice

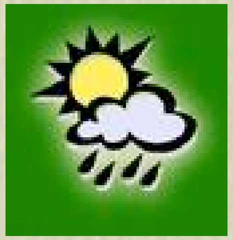


- Chiediamo ad *n* “esperti” di fornirci la previsione meterologica di domani...
- ogni esperto può esprimersi attraverso un valore nominale {rainy, sunny}
- Vogliamo utilizzare il ‘consiglio’ degli esperti per ottenere la ‘nostra’ predizione...

Expert 1	**Expert 2**	**...**	**Expert n**	**Truth**
rainy	sunny	...	sunny	sunny
rainy	sunny	...	rainy	rainy
...	...	...	...	...

Expert Advice

Halving Algorithm



- Qual è una buona strategia per combinare le opinioni degli esperti?
- Ipotizziamo di avere un esperto 'perfetto'
- Strategia: prendiamo il voto di maggioranza come predizione. Ad ogni errore verificato, tagliamo gli esperti non affidabili.
- Al più *log2(n)* errori.
  - Ogni Errore taglia lo spazio degli esperti di un fattore 2.

Expert Advice

- Cosa succede se non esiste l'esperto perfetto?
- Strategia #1: iterare l'algoritmo di Halving. Tutto come prima... ma una volta che abbiamo tagliato via tutti gli esperti ripartiamo dall'inizio.
  - Al massimo *log2(n)*OPT* errori, dove *OPT* è il numero di errori del miglior esperto.
- Algoritmo poco furbo: perde periodicamente l'addestramento! ... si può fare di meglio!

Weighted Majority Algorithm

- Intuizione: se l'esperto commette un errore lo penalizziamo (ma non lo escludiamo!).
- Anzichè tagliar fuori l'esperto, attribuiremo una penalità.
    - Start-up con esperti con peso pari a 1.
    - Predizione basata sulla scelta del voto di maggioranza (somma dei pesi).
    - Penalità per gli esperti che commettono un errore, moltiplicando per 0,5 il relativo peso.

Esempio

					PREDICTION	CORRECT
weights	1	1	1	1		
predictions	Y	Y	Y	N	Y	Y
weights	1	1	1	0.5		
predictions	Y	N	N	Y	N	Y
weights	1	.5	.5	.5		
predictions	Y	N	N	N	N	N
weights	.5	.5	.5	.5		
predictions	N	Y	N	Y	entrambi	N
weights	.5	.25	.5	.25		

Weighted Majority Algorithm [Mitchell 1997]

a_i denota la i-esima predizione dell'esperto nel pool A. w_i denota il peso associato ad a_i

- Per ogni i inizializza $w_i \leftarrow 1$
- Per ogni esempio di training $< x, c(x) >$
  - Inizializza q_0 e q_1 a 0
  - Per ogni predizione dell'esperto e_i
    - Se $a_i = 0$ allora $q_0 \leftarrow q_0 + w_i$
    - Se $a_i = 0$ allora $q_1 \leftarrow q_1 + w_i$
  - Se $q_1 > q_0$ allora predizione $c(x) = 1$
  - Se $q_0 > q_1$ allora predizione $c(x) = 0$
  - Se $q_1 = q_0$ allora predizione 0 o 1 a caso per $c(x)$
  - Per ogni predizione dell'esperto a_i di A
    - Se $a_i(x) \neq c(x)$ allora $w_i \leftarrow \beta w_i$

Weighted Majority Theorem

- Il numero di errori commessi dall'algoritmo WM è al più $2.41(m+lg(n))$.
  - n numero degli esperti
  - M = numero di errori commessi da WM
  - m = numero di errori commessi dal miglior esperto
  - W = peso totale (parte da n)

Weighted Majority Theorem

- Il numero di errori commessi dall'algoritmo WM è al più *2.41(m+lg(n))*.
  - dopo M errori, *W* decresce al più del 25%

					W	PREDICTION	CORRECT
weights	1	1	1	1	4		
predictions	Y	Y	N	N		Y	N
weights	1	1	0.5	0.5	3		
predictions	Y	Y	N	N		Y	Y

Weighted Majority Theorem

- Dopo *M* errori: $W \leq n \times (\frac{3}{4})^M$
- Il peso del miglior esperto è $(\frac{1}{2})^m$

$$(\frac{1}{2})^m \leq n(\frac{3}{4})^M$$

$$(\frac{4}{3})^M \leq n2^m$$

$$M \leq \frac{1}{\log(4/3)}(m + \log n))$$

$$M \leq 2.4(m + \log n)$$

Weighted Majority Algorithm randomized version

- WM non è così performante se il miglior esperto commette un errore con una probabilità del 20%.
- Si può fare di meglio: diluire il caso peggiore!
- Dato un insieme di predizioni X1,...,Xn
- Predizione $\hat{y} = \begin{cases} 1 & \text{with probability } \frac{q_1}{W} \\ 0 & \text{otherwise} \end{cases}$ $W = \sum_i w_i = q_0 + q_1.$
- penalità: generalizziamo con $1 - \epsilon$

$$M \leq \frac{-m \ln(1-\epsilon) + \ln(n)}{\epsilon}$$

Weighted Majority Algorithm

randomized version

- Il numero di errori commessi dall'algoritmo WM è al più $2.41(m+lg(n))$.
  - dopo M errori, W decresce al più del 25%

					W	q0/W	PREDICTION	CORRECT
weights	1	1	1	1	4	2/4		
predictions	Y	Y	N	N			Y	N
weights	1	1	0.5	0.5	3	1/3		
predictions	N	N	Y	Y			N	Y

Weighted Majority Algorithm randomized version

$$M \le \frac{-m\ln(1-\epsilon)+\ln(n)}{\epsilon}$$

$$M \le 1.39m + 2\ln n \qquad \epsilon = \tfrac{1}{2}$$

$$M \le 1.15m + 4\ln n \qquad \epsilon = \tfrac{1}{4}$$

$$M \le 1.07m + 8\ln n \qquad \epsilon = \tfrac{1}{8}$$

$$M \le (1+\tfrac{\epsilon}{2})m + \tfrac{1}{\epsilon}\ln n \qquad \epsilon \to 0$$

Weighted Majority Algorithm randomized version

Caso semplice: Esperto Perfetto.

Definiamo con Fi la frazione del peso totale sulle risposte sbagliate all'i-esimo passo.

Al tempo t, la frazione degli esperti 'vivi' che commette un errore è Ft.

Il numero di errori commessi è $M = \sum_{i=1}^{t} F_i$

Sapendo che $\prod(1 - F_t) \geq 1/n$

Weighted Majority Algorithm randomized version

$$\prod(1 - F_t) \geq 1/n$$

$$\sum \ln(1 - F_t) \geq -\ln n$$

$$-\sum F_t \geq -\ln n$$

$$\boxed{M} \leq ln(n)$$

$$\boxed{-\ln(1 - x) > x}$$

quindi meglio dell'algoritmo
di Halving!!! (log2(n))

Weighted Majority Algorithm randomized version

Definiamo con Fi la frazione del peso totale sulle risposte sbagliate all'i-esimo passo.

Ipotizziamo di aver visto t esempi.

$$M = \sum_{i=1}^{t} F_i$$

All'i-esimo esempio, il peso totale cambia secondo la seguente regola:

$$W \leftarrow W(1 - (1 - \beta)F_i)$$

Weighted Majority Algorithm randomized version

Estendendo su t esempi:

$$W = n\prod_{i=1}^{t}(1-(1-\beta)F_i)$$

Sia m il numero di errori che il miglio esperto commette:

$$W = n\prod_{i=1}^{t}(1-(1-\beta)F_i) \geq \beta^m$$

Weighted Majority Algorithm randomized version

$$W = n\prod_{i=1}^{t}(1-(1-\beta)F_i) \geq \beta^m$$

$$\ln n + \sum_{i=1}^{t}\ln(1-(1-\beta)F_i) \geq m\ln\beta$$

$$-\ln n - \sum_{i=1}^{t}\ln(1-(1-\beta)F_i) \leq m\ln(1/\beta)$$

$$-\ln n + (1-\beta)\boxed{\sum_{i=1}^{t}F_i} \leq m\ln(1/\beta)$$

$$M \leq \frac{m\ln(1/\beta)+\ln n}{1-\beta} \quad \blacksquare$$

$$\boxed{-\ln(1-x) > x}$$

$$M \leq m + \ln n + O(\sqrt{m\ln n})$$

Apprendimento On-Line da esempi

- Si ha a disposizione un insieme di esempi classificati x:<v1,v2,...,vn,o>
- dove vi sono valori delle variabili di ingresso, ed o è l'uscita. Prendono anche il nome di attributi o features.
- Implica l'esistenza di un istruttore che conosce la risposta corretta
- Si apprende una funzione obiettivo $f : V_1 \times V_2 \times V_3 \times \cdots \to O$
- a misura della prestazione consiste nel minimizzare l'errore di apporssimazione della funzionae obiettivo sugli esempi a disposizione

$$min(e) = min|f_{obb} - f|$$

Esempio: apprendimento della funzione OR

- Supponiamo di avere un vettore le cui componenti siano booleane: $x \in \{0,1\}^n$

$$h(x) = x_1 \vee x_2 \vee \ldots x_n$$

- Algoritmo semplice:
  - Invariante: {vars in h} contenute {vars in target}
  - in caso di errore sull'esempio negativo x, per ogni xi=1, rimuovi xi da h.
    - Mantiene l'invariante; |h| decresce almeno di 1
    - Al più *n* errori!

Modellazione ML del problema della funzione OR

- Supponiamo di avere un vettore le cui componenti siano booleane: $x \in \{0,1\}^n$

$$h(x) = x_1 \vee x_2 \vee \dots x_n$$

  - Il target è un vettore (a1,...,an) e una soglia t.
  - L'esempio x è positivo se $a_1 x_1 + \cdots + a_n x_n \geq t$
  - La funzione OR è composta dai valori di ai binari e t=1.

Winnow Algorithm

- Predizione positiva quando:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \geq n$$

- Inizializzazione:

$$\forall i \quad w_i = 1$$

- Regola di aggiornamento
  - In caso di errore su predizione positiva:

$$\text{Se } x_i = 1 \text{ allora } w_i \leftarrow 2w_i$$

  - In caso di errore su predizione negativa:

$$\text{Se } x_i = 1 \text{ allora } w_i \leftarrow w_i/2$$

Esempio

						PREDICTION	CORRECT
weights	1	1	1	1	1		
example	0	1	0	1	1	–	+
weights	1	2	1	2	2		
example	1	1	1	0	0	–	+
weights	2	4	2	2	2		
example	0	1	1	0	0	+	–
weights	2	2	1	2	2		
example	1	0	1	0	0		

Analisi

- I ‘pesi rilevanti’ non decrescono mai
- Ogni errore sugli esempi positivi raddoppiano almeno un ‘peso rilevante’
- Ogni peso rilevante può raddoppiare al massimo $log_2(n)+1$ volte
- Quindi al massimo ci saranno $r(log_2(n)+1)$ errori sui positivi.

Analisi

- Il peso totale parte da n.
- Ogni errore sui positivi aggiunge al più n al peso totale.
- Ogni errore sui negativi rimuove al più $n/2$ dal totale.
- Il peso totale è sempre > 0.
- # errori sui negativi < 2+2(# errori sui postivi)
- Il numero totale degli errori è al più: $2+3r(\log_2(n)+1)$

Winnow specialists

- Ogni specialista parte con peso 1.
- Predizione con WM voting.
- Se l'algoritmo globale commette un errore,
  - Moltiplica lo specialista non corretto con 0,5
  - Moltiplica lo specialista corretto con 2

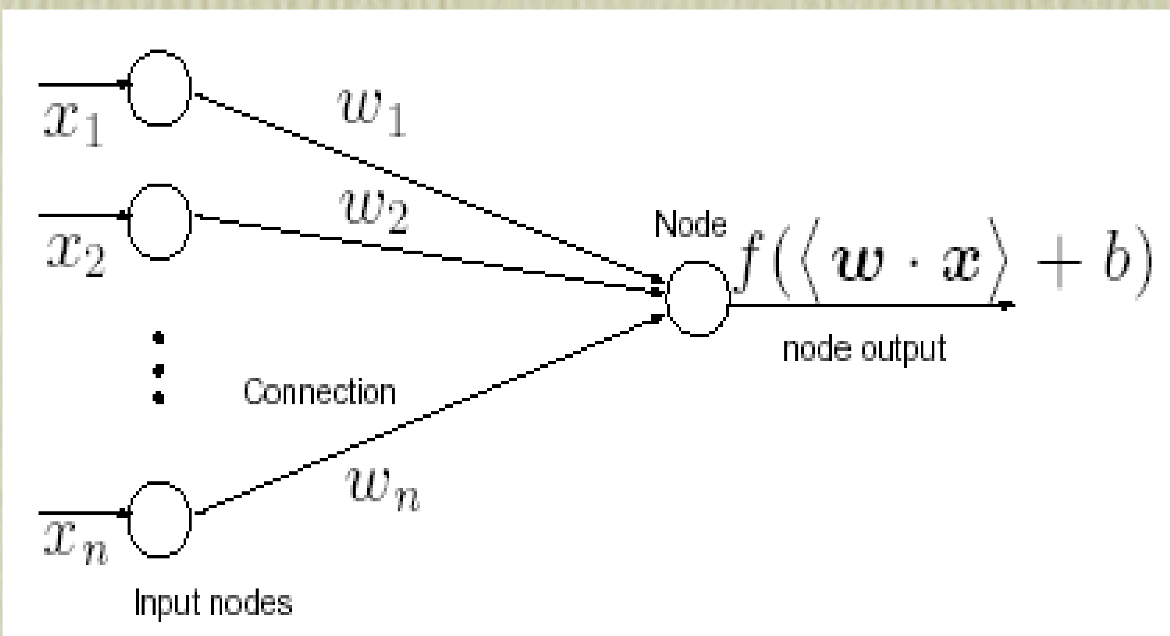
Esempio Text Categorization

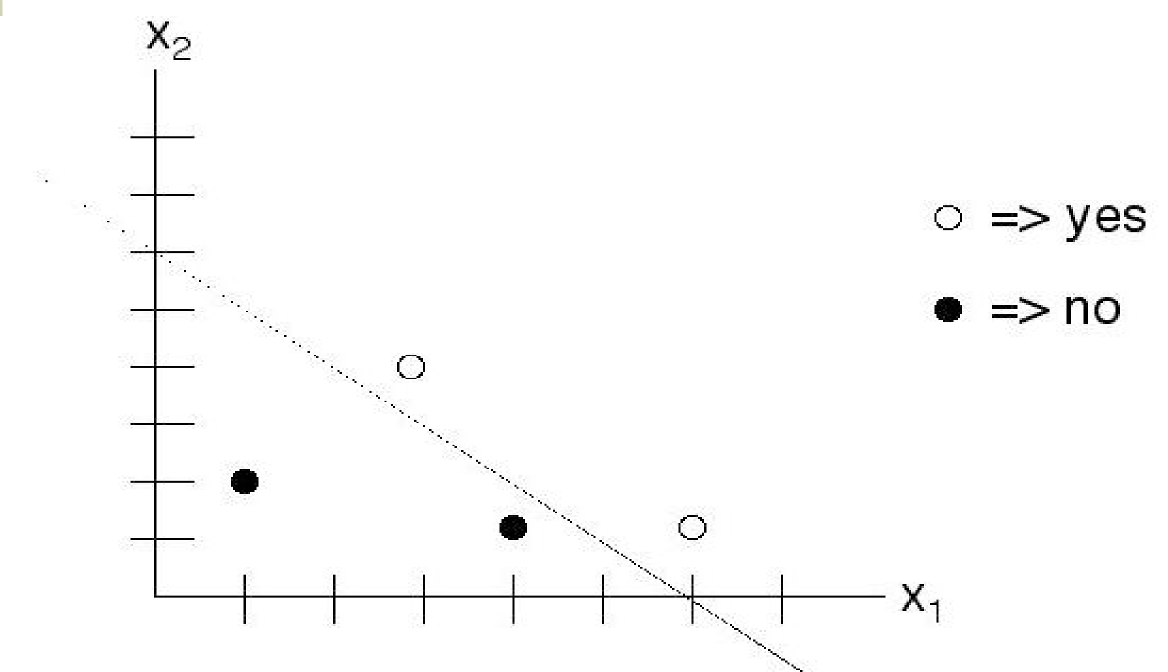
- Classificare documenti all'interno di categorie predefinite
- Uno specialista per ogni parola o piccola frase. Lo specialista si 'sveglia' quando la parola/frase compare nel testo da classificare.

Esempio [Cohen, Singer 1999]
Media del numero di errori:

	Rocchio	Ripper	1-word exp	4-word exp
TREC	91.11	84.56	92.33	80.33
Reuters	498.10	486.60	476.80	439.65

Perceptron Algorithm





Given training set S

$\mathbf{w}_0 \leftarrow \mathbf{0}; b_0 \leftarrow 0; k \leftarrow 0$

$R \leftarrow \max_{1 \le i \le l} \|\mathbf{x}_i\|$

repeat

  for $i = 1$ to l

    if $y_i(\langle \mathbf{w}_k \cdot \mathbf{x}_i \rangle + b_k) \le 0$ then

      $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$

      $\mathrm{b}_{\mathrm{k}+1} \leftarrow \mathrm{b}_{\mathrm{k}} + \eta y_i R^2$

      $\mathrm{k} \leftarrow k + 1$

    end if

  end for

until no mistakes made within the *for* loop

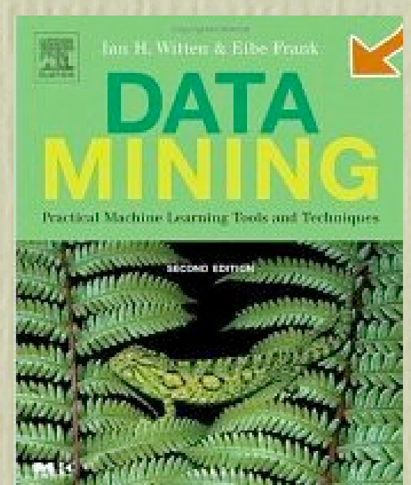
return k, $(\mathbf{w}_k, b_k)$ where k is the number of mistakes

Teorema di Novikoff

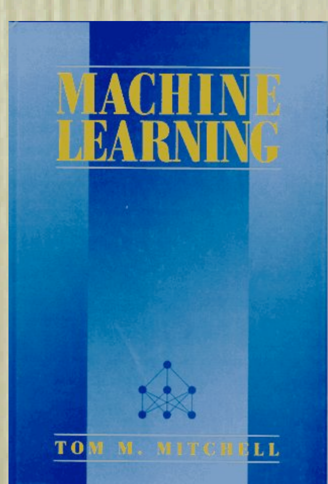
- Novikoff (1956)
  - Prova la convergenza del perceptron algorithm
  - $R = \max |x_i|$; $|w_{opt}| = 1$
  - $y_i ((w_{opt} \cdot x_i) + b_{opt}) >= \text{gamma}$
  - $k <= (2R/\text{gamma})$^2
  - (k è il numero degli errori)

Bibliografia

On-Line Algorithm in Machine Learning
Avrim Blum
in “Online Algorithms the state of the art”, Fiat and Woeginger eds.,
LNCS #1442, 1998.



Data Mining: Practical Machine Learning Tools and Techniques,
Second Edition
Ian H. Witten and Eibe Frank
Morgan Kaufmann 2006



Machine Learning
Tom M. Mitchell
McGraw-Hill 1997