

Lecture 8

Combinare Classificatori

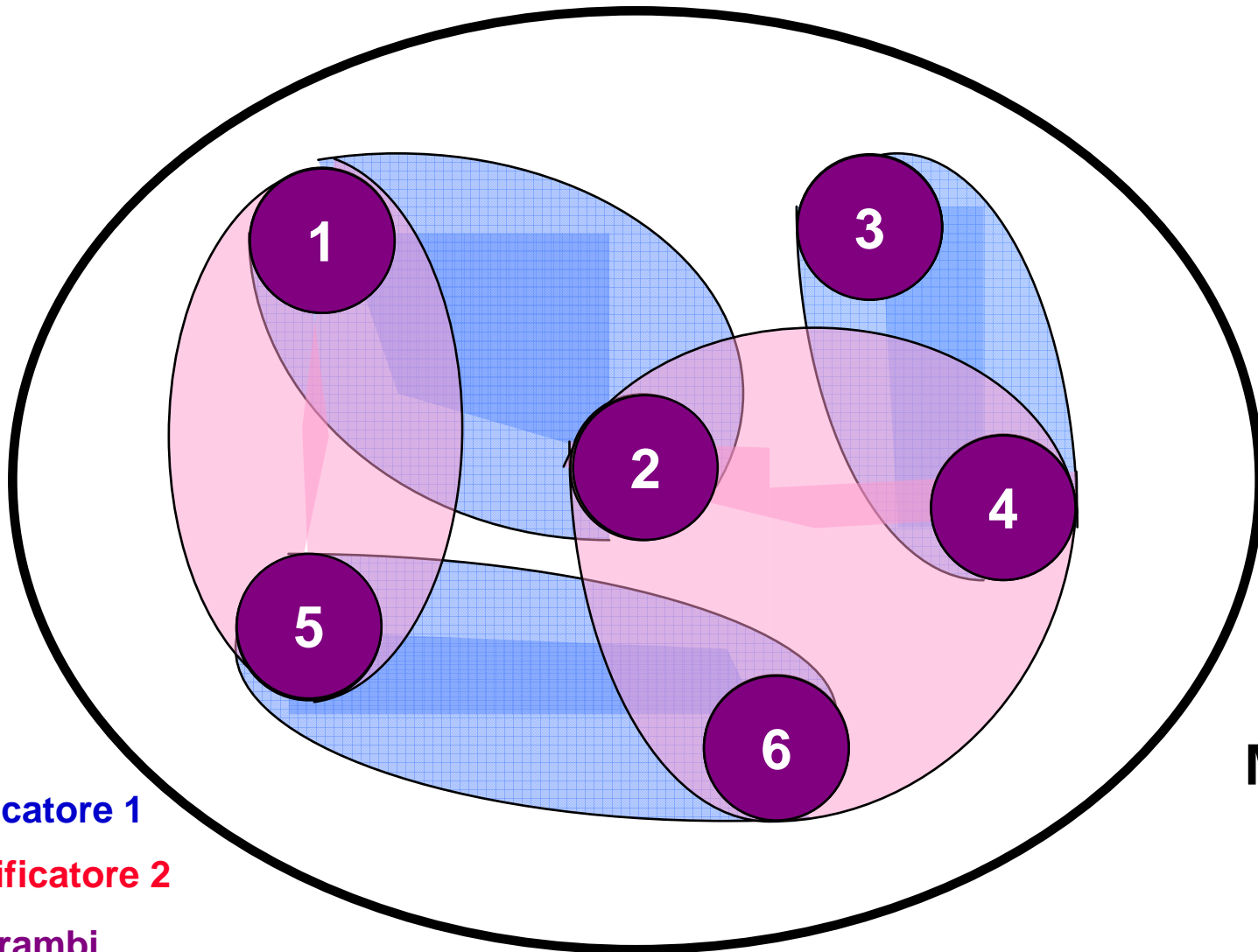
Giovedì, 18 novembre 2004

Francesco Folino

Combinare classificatori

- **Problema**
 - Dato
 - Training set D di dati in X
 - Un insieme di algoritmi di learning
 - *Una trasformazione $s: X \rightarrow X'$ (sampling, trasformazione, partizionamento)*
 - Restituire
 - Un nuovo algoritmo per $x \in X$ che combina gli outputs degli algoritmi ottenuti
- **Proprietà**
 - Limitare l'errore
- **Due approcci**
 - Apprendimento e applicazione di ogni classificatore; apprendiamo la funzione che combina dai risultati
 - Apprendimento dei classificatori e del combinatore concorrentemente

Principio: Migliorare i classificatori deboli



Classificatore 1

Classificatore 2

Entrambi

**Mixture
Model**

Metaclassificazione

Classificatori deboli

- **Classificatori deboli**
 - Non c'è garanzia che facciano meglio di una scelta random ($1 / \text{numero di classi}$)
 - Più generalmente, con errore alto
 - Obiettivo: combinando più classificatori deboli, vogliamo ottenere un'altro che sia accurato almeno quanto il migliore dei deboli

Maggioranza pesata

- **Combinazione basata sul peso**
 - Voti pesati: ogni classificatore h_i è una funzione da $x \in X$ a $h_i(x)$
 - **NB**: non sempre è necessario che il classificatore sia in H
- **Idea**
 - Collezionamo i voti da un insieme di classificatori
 - Associamo un peso ad ogni algoritmo
 - Il peso degli algoritmi con errore alto sul training set (test set) hanno peso minore
 - Il combinatore predice l'etichetta (il concetto) con la maggioranza pesata
- **Obiettivo**
 - Migliorare l'accuratezza sul training set
 - Combinare classificatori deboli
 - Limitare il numero di errori in termini del minimo fatto da ciascun classificatore

Procedura

- **Algorithm Combiner-Weighted-Majority (D, L)**
 - $n \leftarrow L.size$ // numero di classificatori
 - $m \leftarrow D.size$ // numero di istanze $\langle x \equiv D[j], c(x) \rangle$
 - FOR $i \leftarrow 1$ TO n DO
 - $P[i] \leftarrow L[i].Train-Classifier(D)$ // $P[i]$: i -esimo classificatore
 - $w_i \leftarrow 1$ // peso iniziale
 - FOR $j \leftarrow 1$ TO m DO // calcolo dell'etichetta
 - $q_0 \leftarrow 0, q_1 \leftarrow 0$
 - FOR $i \leftarrow 1$ TO n DO
 - IF $P[i](D[j]) = 0$ THEN $q_0 \leftarrow q_0 + w_i$ // voto per 0 (-)
 - IF $P[i](D[j]) = 1$ THEN $q_1 \leftarrow q_1 + w_i$ // voto per 1 (+)
 - $Prediction[i][j] \leftarrow (q_0 > q_1) ? 0 : ((q_0 = q_1) ? Random(0, 1) : 1)$
 - IF $Prediction[i][j] \neq D[j].target$ THEN // $c(x) \equiv D[j].target$
 - $w_i \leftarrow \beta w_i$ // $\beta < 1$ (penalizzazione)
 - RETURN *Make-Predictor* (w, P)

Proprietà

- **Vantaggi**
 - Limitazione dell'errore
 - D training set, L insieme di classificatori
 - k numero minimo di errori su D da parte di qualche $L[i]$, $1 \leq i \leq n$
 - Proprietà: gli errori sono al più $2.4 (k + \log n)$

Bagging

- **Bootstrap Aggregating**
 - **Campionamento mirato**
 - **Dato: D contenente m istanze**
 - **Creiamo $S[i]$ pescando a caso (con rimpiazzamento) m istanze da D**
 - **$S[i]$ di dimensione m : il rimpiazzamento può far replicare le scelte**
 - Tipicamente, si lascia fuori lo 0.37 di D
 - **Bagging**
 - **Creiamo k bootstrap training sets $S[1], S[2], \dots, S[k]$**
 - **Costruiamo un classificatore distinto su ogni $S[i]$ (k classificatori in totale)**
 - **Classifichiamo la nuova istanza con il voto (non pesato)**
- **Idea**
 - “two is better than one”
 - **Lo stesso dataset crea più classificatori**
 - ***NB*: stesso algoritmo (istanziamenti diverse) o algoritmi differenti**

Procedura

- **Algorithm *Combiner-Bootstrap-Aggregation* (D, L, k)**
 - FOR $i \leftarrow 1$ TO k DO
 - $S[i] \leftarrow \text{Sample-With-Replacement}(D, m)$
 - $\text{Train-Set}[i] \leftarrow S[i]$
 - $P[i] \leftarrow L[i].\text{Train-Classifier}(\text{Train-Set}[i])$
 - RETURN ($\text{Make-Predictor}(P, k)$)
- **Function *Make-Predictor* (P, k)**
 - RETURN (fn $x \Rightarrow \text{Predict}(P, k, x)$)
- **Function *Predict* (P, k, x)**
 - FOR $i \leftarrow 1$ TO k DO
 - $\text{Vote}[i] \leftarrow P[i](x)$
 - RETURN ($\text{argmax}(\text{Vote}[i])$)
- **Function *Sample-With-Replacement* (D, m)**
 - RETURN (m istanze campionate utilizzando la distribuzione uniforme da D)

Proprietà

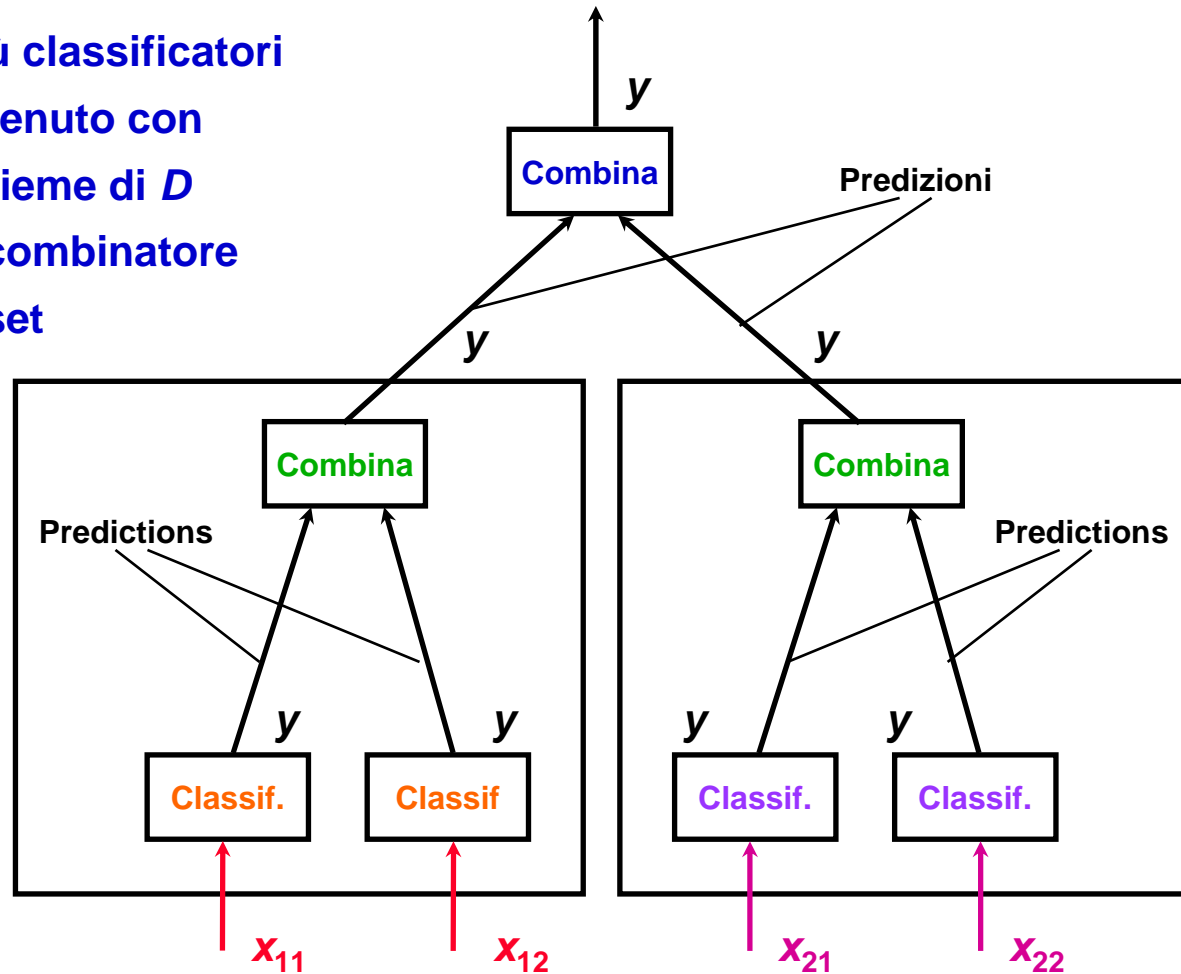
- **Quando migliora l'accuratezza?**
 - **Classificatori instabili**
 - Piccoli cambiamenti nel training set causano modifiche sostanziali nel risultato
 - decision trees, neural networks;
 - *Non vale per k-nearest neighbor*
 - **Sperimentalmente**
 - Significativo con classificatori instabili
 - Degrada le performances nei classificatori stabili

Stacked Generalization

- Stacking

- **Idea**

- Costruiamo più classificatori
 - Ognuno ottenuto con un sottoinsieme di D
- Costruiamo il combinatore sul validation set



Procedura

- **Algorithm *Combiner-Stacked-Gen* ($D, L, k, n, m', Levels$)**
 - Partiziona D in k segmenti, $S[1], S[2], \dots, S[k]$
 - FOR $i \leftarrow 1$ TO k DO
 - *Validation-Set* $\leftarrow S[i]$ // m/k istanze
 - FOR $j \leftarrow 1$ TO n DO
 - Train-Set* $[j] \leftarrow$ *Sample-With-Replacement* ($D \sim S[i], m'$) // $m - m/k$ istanze
 - IF $Levels > 1$ THEN
 - $P[j] \leftarrow$ *Combiner-Stacked-Gen* (*Train-Set* $[j], L, k, n, m', Levels - 1$)
 - ELSE
 - $P[j] \leftarrow L[j].Train-Classifer$ (*Train-Set* $[j]$)
 - *Combiner* $\leftarrow L[0].Train-Classifer$ (*Validation-Set.targets*,
Apply-Each ($P, Validation-Set.inputs$))
- *Predictor* \leftarrow *Make-Predictor* (*Combiner, P*)
- RETURN *Predictor*

Proprietà

- **Simile alla cross-validation**
 - *k-fold*: il validation set è ruotato
 - *Tipicamente, migliora la capacità di generalizzare*
 - Rimedio contro l'overfitting

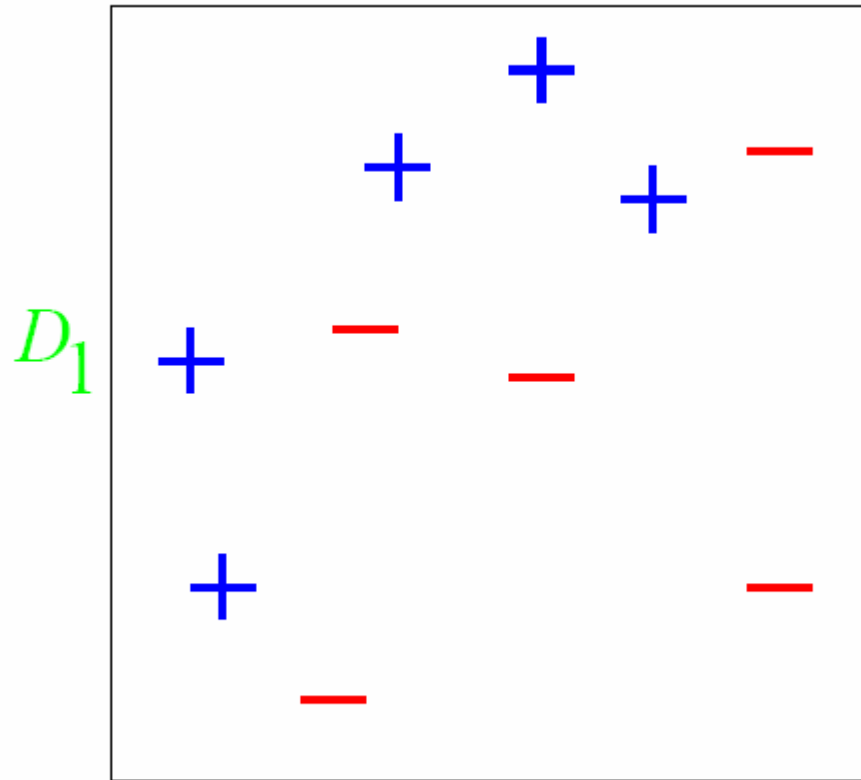
Boosting

- **Idea**
 - Apprende un insieme di classificatori da D ,
 - Ri-pesa le istanze per enfatizzare la misclassificazione
 - Classificatore finale ← combinazione pesata di classificatori
- **Differente dagli altri metodi**
 - **WM**: tutti i classificatori allenati sullo stesso D
 - **Bagging, stacking**: partizionamento su training/validation, campioni indipendenti $S[j]$ di D
 - **Boosting**: dati campionati con *distribuzioni differenti*

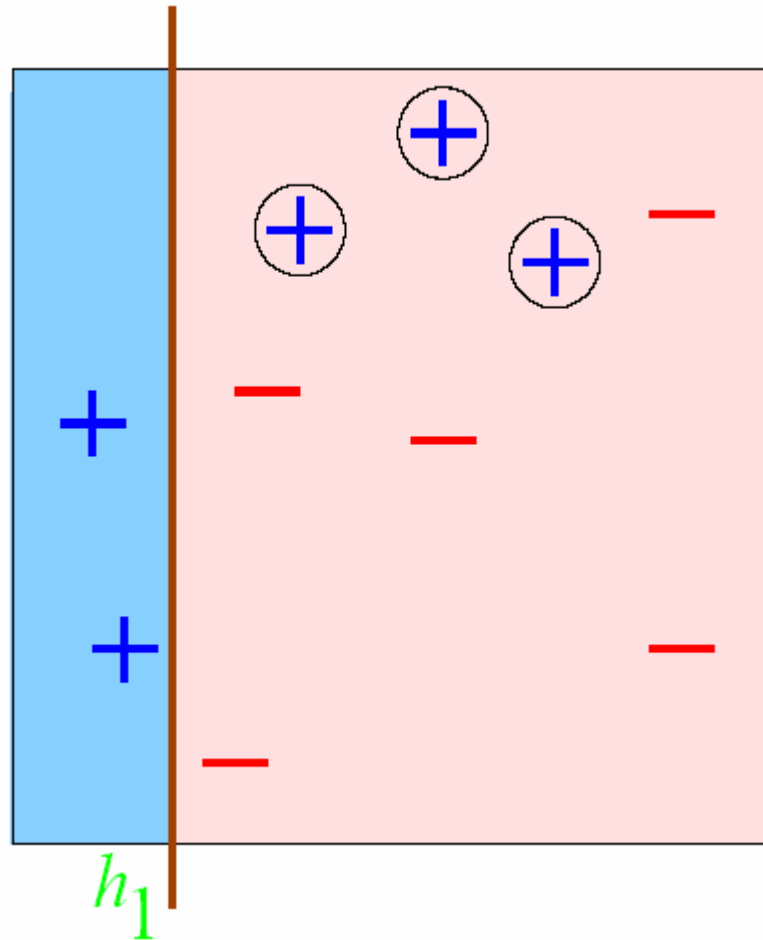
Boosting: Procedure

- **Algorithm *Combiner-AdaBoost* (D, L, k)**
 - $m \leftarrow D.size$
 - FOR $i \leftarrow 1$ TO m DO // inizializzazione
 - $Distribution[i] \leftarrow 1 / m$ // campionamento
 - FOR $j \leftarrow 1$ TO k DO
 - $P[j] \leftarrow L[j].Train-Classifer(Distribution, D)$
 - $Error[j] \leftarrow Count-Errors(P[j], Sample-According-To (Distribution, D))$
 - $\beta[j] \leftarrow Error[j] / (1 - Error[j])$
 - FOR $i \leftarrow 1$ TO m DO // aggiorniamo D
 - $Distribution[i] \leftarrow Distribution[i] * ((P[j](D[i]) = D[i].target) ? \beta[j] : 1)$
 - $Distribution.Renormalize ()$
 - RETURN ($Make-Predictor (P, D, \beta)$)
- **Function *Make-Predictor* (P, D, β)**
 - // $Combiner(x) = \operatorname{argmax}_{v \in V} \sum_{j: P[j](x) = v} \log (1/\beta[j])$
 - RETURN (fn $x \Rightarrow Combiner(x)$)

Esempio [1]

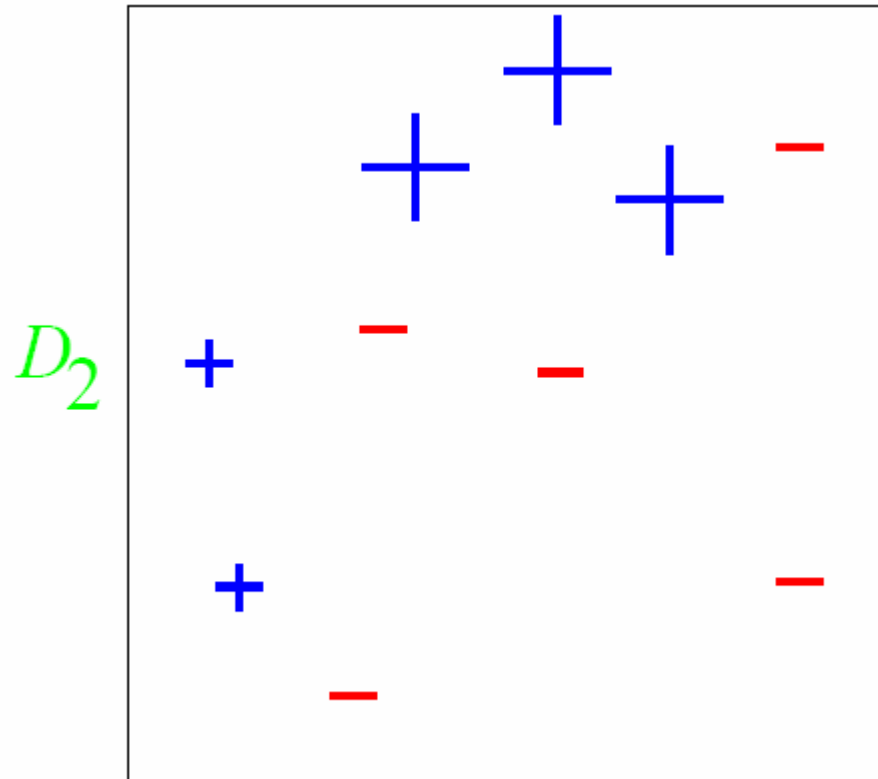


Esempio [2]

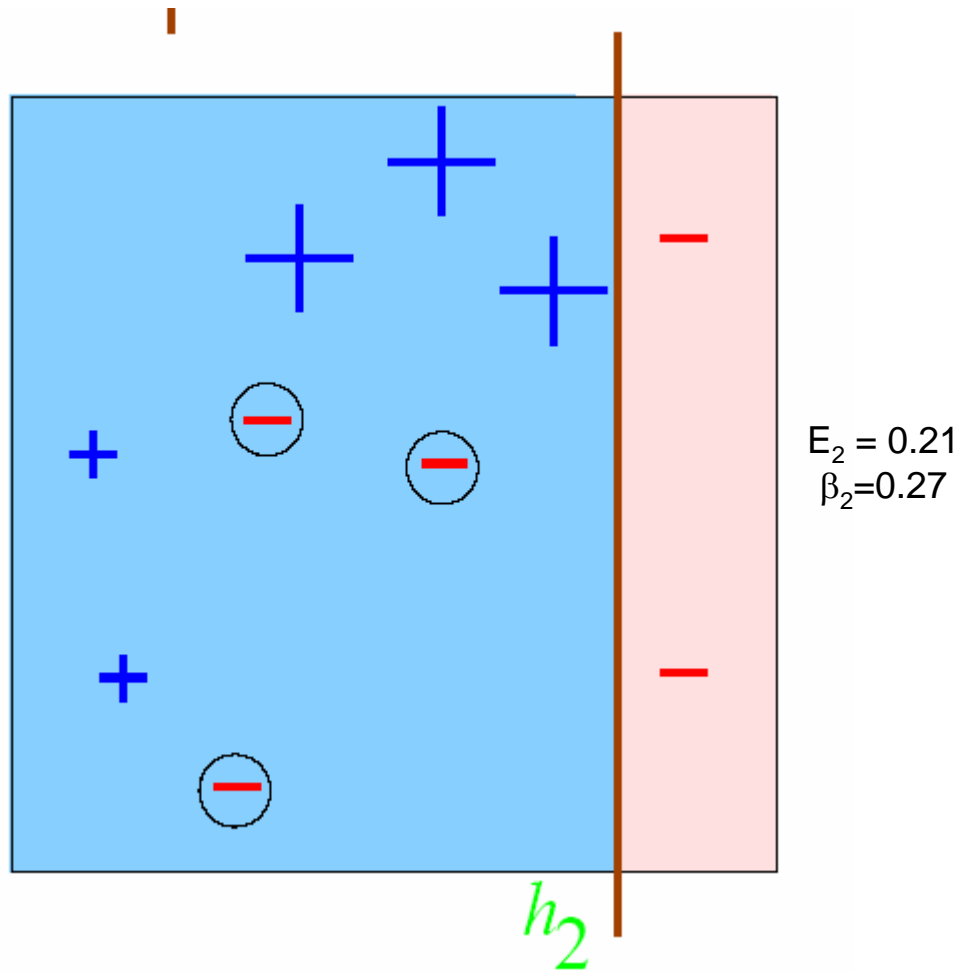


$$E_1 = 0.3$$
$$\beta_1 = 0.42$$

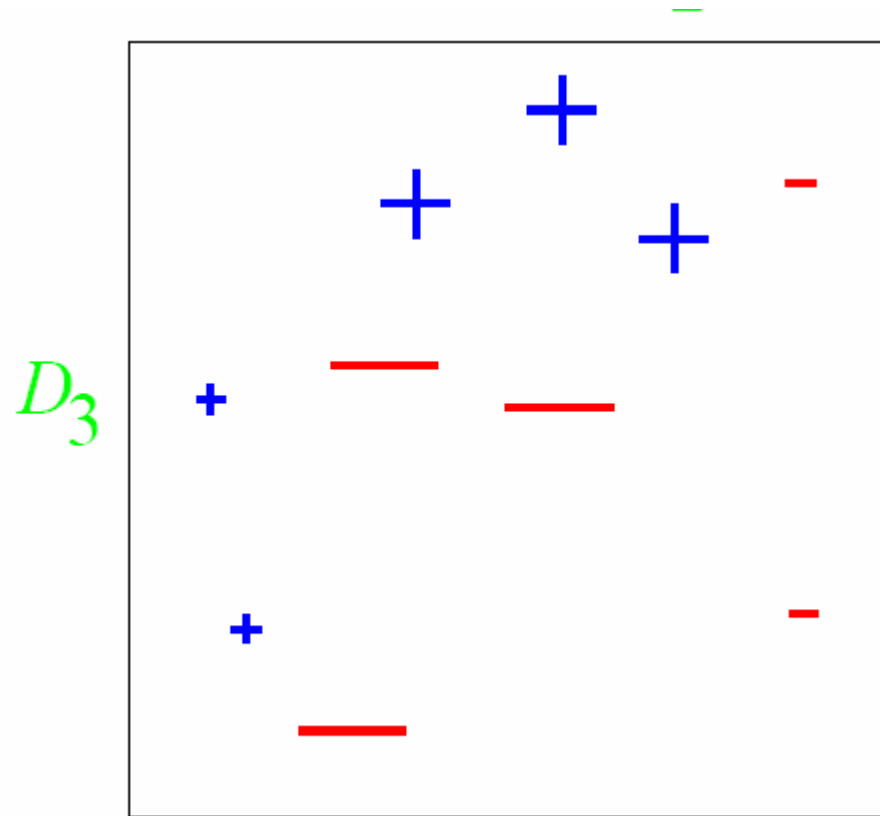
Esempio [3]



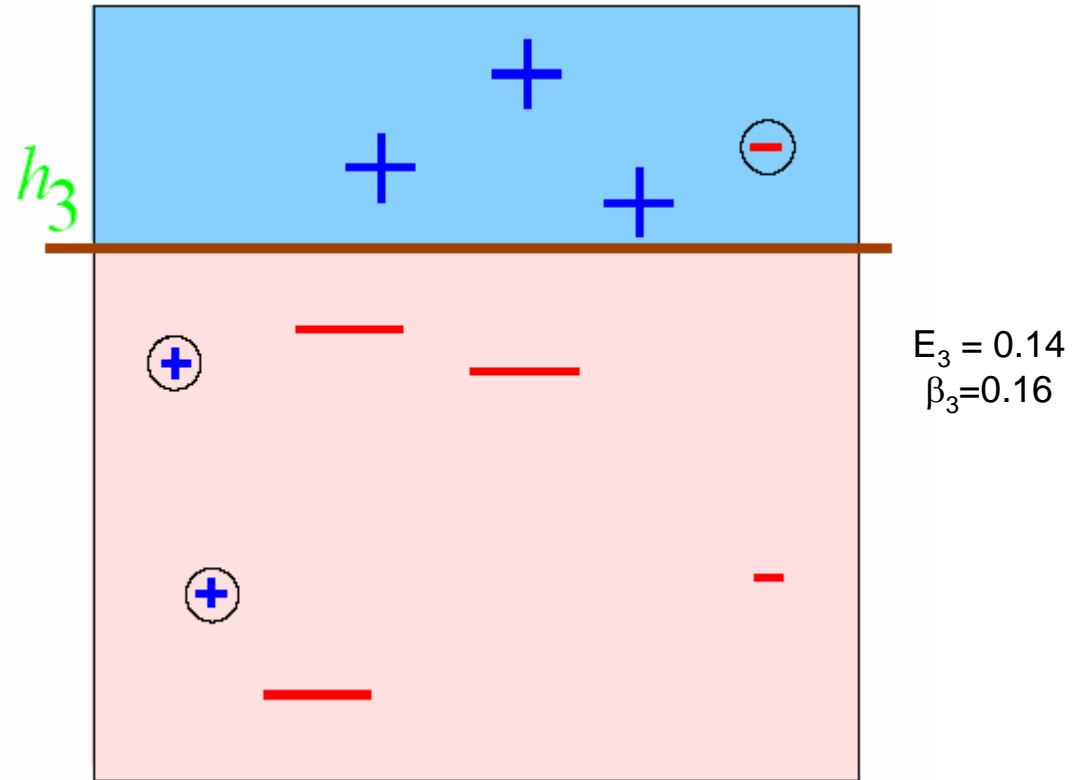
Esempio [4]



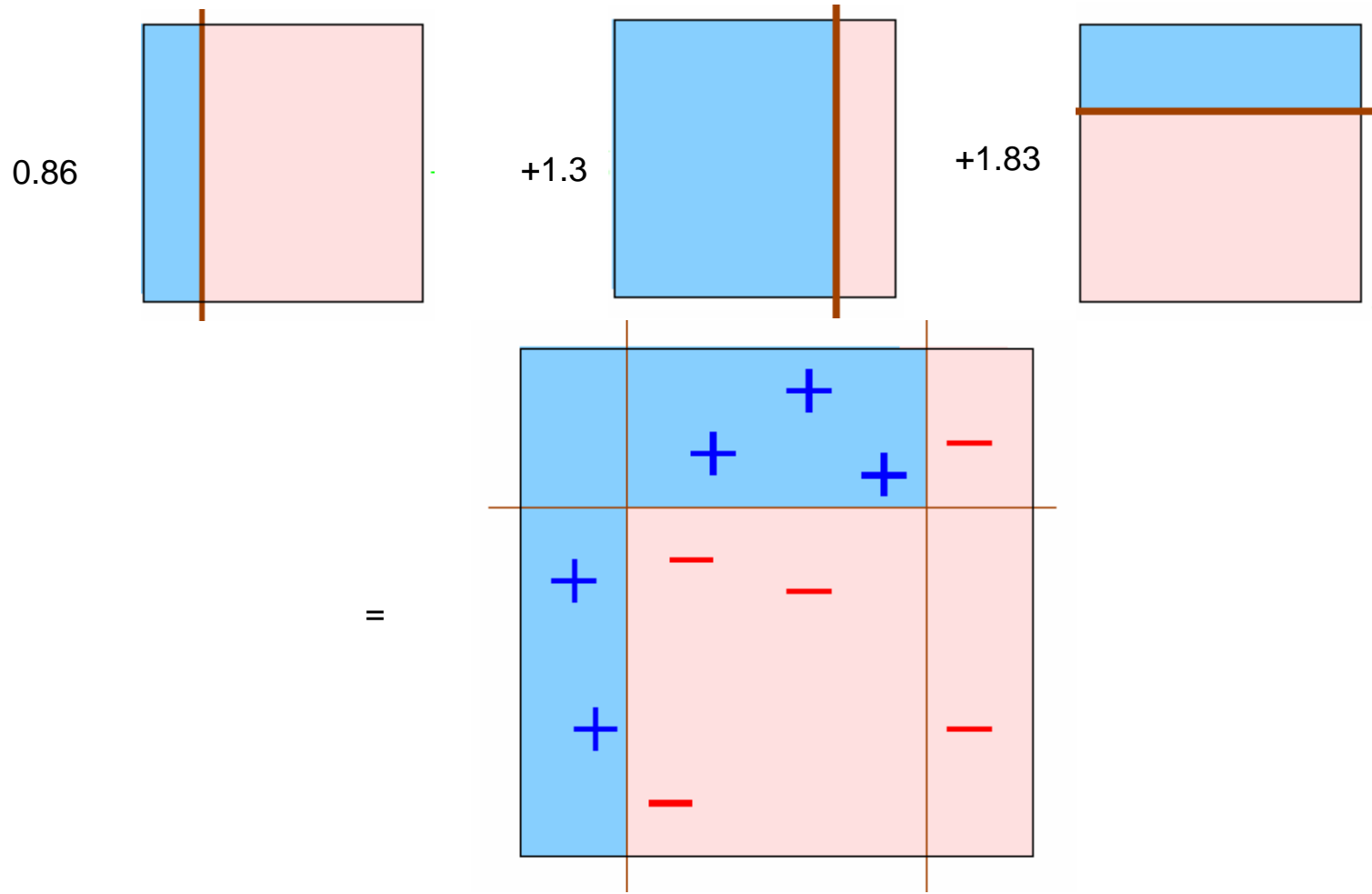
Esempio [5]



Esempio [6]



Esempio [7]



Proprietà

- **In generale**
 - **Sperimentalmente molto efficace**
 - **Molte varianti dell'algoritmo precedente**
 - **Area di ricerca attiva**