Multi-Classificatori
(Ensemble methods)

Un multi-classificatore è un approccio dove diversi classificatori sono utilizzati (*normalmente in parallelo, ma talvolta anche in cascata o in modo gerarchico*) per eseguire la classificazione dei pattern; le decisioni dei singoli classificatori sono *fuse* ad un certo livello del processo di classificazione.

È stato dimostrato (*teoricamente ma soprattutto nella pratica*) che l'utilizzo di *combinazioni di classificatori* (in inglese multi-classifier, combination of classifiers, classifier fusion, ensemble learning) può migliorare, anche molto, le prestazioni.

- Siate pragmatici! Nella pratica investire tempo nell'ottimizzazione «spinta» di un singolo classificatore è in genere meno conveniente rispetto all'affiancamento (al classificatore iniziale) di altri classificatori.
- Attenzione però: la combinazione è efficace solo quando i singoli classificatori sono (almeno parzialmente) *indipendenti tra loro*, ovvero non commettono gli stessi errori.
- L'indipendenza (o diversità) è normalmente ottenuta:
  - Utilizzando feature diverse (non correlate o poco correlate)
  - Utilizzando algoritmi diversi per l'estrazione delle feature
  - Utilizzando diversi algoritmi di classificazione
  - Addestrando lo stesso algoritmo di classificazione su porzioni diverse del training set (*bagging*)
  - Insistendo con l'addestramento sugli errori commessi dai predecessori (*boosting*)
- La combinazione può essere eseguita a *livello di decisione* o a *livello di confidenza*.

Fusione a livello di decisione

Ogni singolo classificatore fornisce in output la propria decisione che consiste nella classe cui ha assegnato il pattern. Le decisioni possono essere tra loro combinate in diversi modi, tra cui:

- Majority vote rule: è uno dei più noti e semplici metodi di fusione; ogni classificatore vota per una classe, il pattern viene assegnato alla classe maggiormente votata.

  Più formalmente:

  Se $\{C_1, C_2, \dots C_{nc}\}$ è un insieme di nc classificatori, e

  $$\theta_{ij} = \begin{cases} 1 & se\ i\ \text{è}\ la\ classe\ votata\ da\ C_j \\ 0 & altrimenti \end{cases} \qquad (1 \le i \le s, 1 \le j \le nc)$$

  Allora il pattern è assegnato alla classe t tale che:

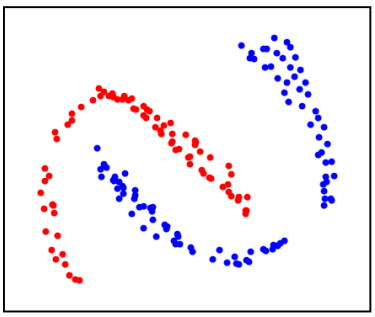
  $$t = \underset{i=1..s}{arg\,max}\left\{\sum_{j=1\dots nc} \theta_{ij}\right\}$$

- Borda count: ogni classificatore invece di una singola classe, produce una classifica o ranking delle classi (dalla prima all'ultima) a seconda della probabilità che a ciascuna di esse appartenga il pattern da classificare.

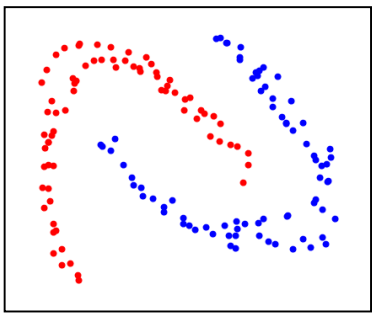
  I ranking sono convertiti in punteggi e poi sommati; la classe con il più elevato punteggio finale è quella scelta dal multi-classificatore.

  Rispetto a majority vote rule, considera anche i «non primi» di ogni classificatore.

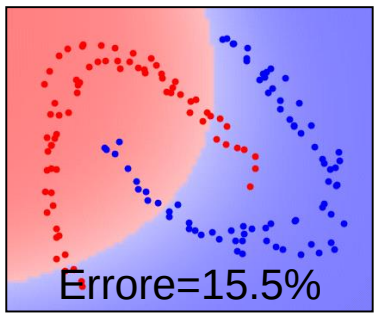
Majority vote rule (esempio)



Training set

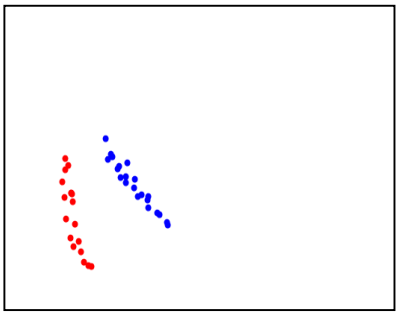


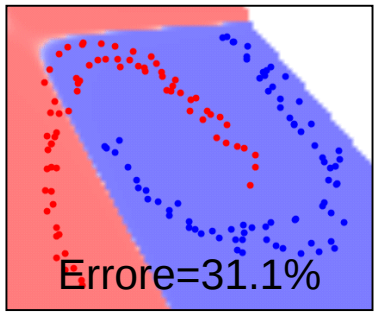
Test set



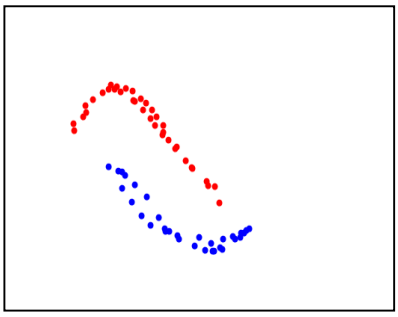
(Bayes – Distr. Normali)

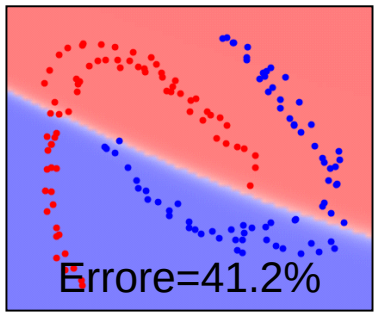
Dividendo il training set in 3 parti e addestrando 3 classificatori (stesso tipo):

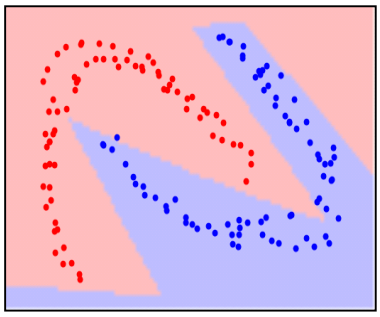




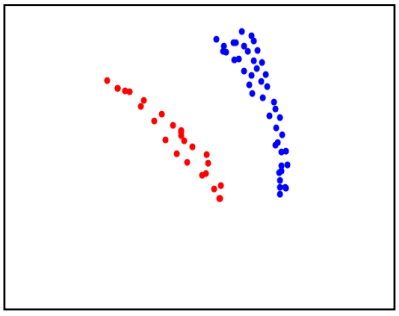
Majority vote rule:

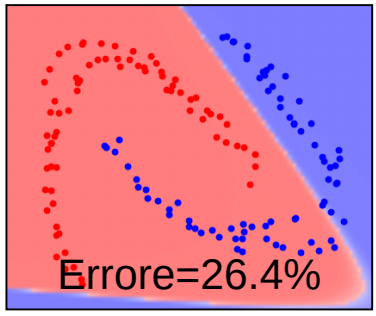






Errore=0.7%





One-Against-One

L'approccio One-Against-One, consente di risolvere un problema di classificazione multi-classe, attraverso classificatori binari.

È l'approccio adottato dalla libreria LIBSVM (usata in BioLab).

- Se s sono le classi del problema, si addestrano $s \times (s-1)/2$ classificatori binari: tutte le possibili coppie, indipendentemente dall'ordine.
- Durante la classificazione, il pattern $\mathbf{x}$ viene classificato da ogni classificatore binario, che assegna un voto alla classe (tra le due) più probabile.
- Al termine il pattern $\mathbf{x}$ è assegnato alla classe che ha ricevuto più voti (*majority vote rule*).

È in genere più accurato di One-Against-All (discusso in precedenza per SVM), anche se meno efficiente in quanto richiede l'addestramento di un numero maggiore di classificatori.

Fusione a livello di confidenza (1)

Ogni singolo classificatore $C_j,\ j = 1..nc$ fornisce in output la confidenza di classificazione del pattern rispetto a ciascuna delle classi, ovvero un vettore $\mathbf{conf}_j = [conf_{j1}, conf_{j2} \ldots conf_{js}]$ in cui l'i-esimo elemento indica il grado di appartenenza del pattern alla classe i-esima.

Diversi metodi di fusione sono possibili tra cui: somma (sum), prodotto (prod), massimo (max) e minimo (min):

- $\mathbf{sum} = \sum_{j=1\ldots nc} \mathbf{conf}_j \qquad t = \underset{i=1\ldots s}{arg\ max}\{\ sum_i\ \}$
- $\mathbf{prod} = \prod_{j=1\ldots nc} \mathbf{conf}_j \qquad t = \underset{i=1\ldots s}{arg\ max}\{\ prod_i\ \}$
- $max_i = \underset{j=1\ldots nc}{max}\{\ conf_{ji}\ \} \qquad t = \underset{i=1\ldots s}{arg\ max}\{\ max_i\ \}$
- $min_i = \underset{j=1\ldots nc}{min}\{\ conf_{ji}\ \} \qquad t = \underset{i=1\ldots s}{arg\ max}\{\ min_i\ \}$

- Il criterio del minimo può sembrare illogico. *Attenzione scegliamo il massimo dei minimi … quindi una classe che non ha ricevuto confidenza troppo bassa da nessun classificatore.*
- Il prodotto è il metodo «probabilisticamente» più corretto (la probabilità congiunta si calcola come prodotto di probabilità) ma solo nel caso di indipendenza statistica.
- Nella pratica la somma è spesso preferibile al prodotto in quanto più robusta. Infatti nel prodotto è sufficiente che un solo classificatore indichi confidenza zero per una classe per portare a zero la confidenza del multi-classificatore per quella classe.

Fusione a livello di confidenza (2)

- Una variante efficace, è quella della somma pesata, dove la somma dei vettori confidenza è eseguita pesando i diversi classificatori in base al loro grado di abilità g_j.

$$\mathbf{avgsum} = \sum_{j=1\ldots nc} g_j \cdot \mathbf{conf}_j \qquad t = \underset{i=1\ldots s}{arg\,max}\{\, avgsum_i \,\}$$

  I gradi di abilità possono essere definiti in base alle singole prestazioni dei classificatori, ad esempio in maniera inversamente proporzionale all'errore di classificazione (vedi AdaBoost).

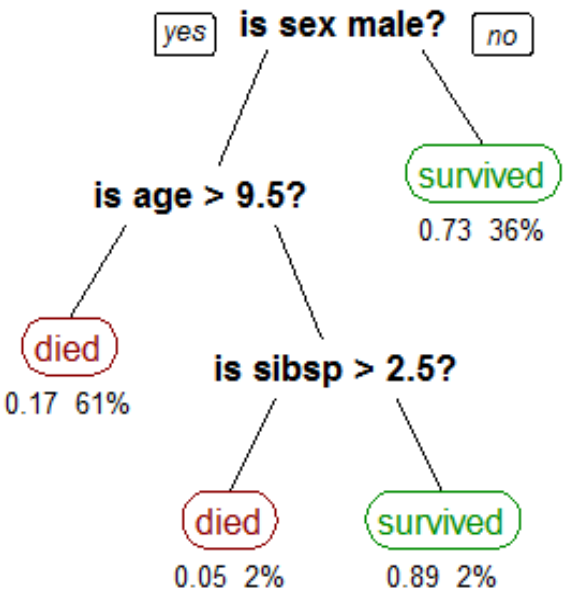
- Attenzione alla normalizzazione dei vettori di confidenza nel caso in cui essi non siano probabilità (ma ad esempio similarità). *Riferimento a quanto detto per la normalizzazione delle distanze*.

Random Forest

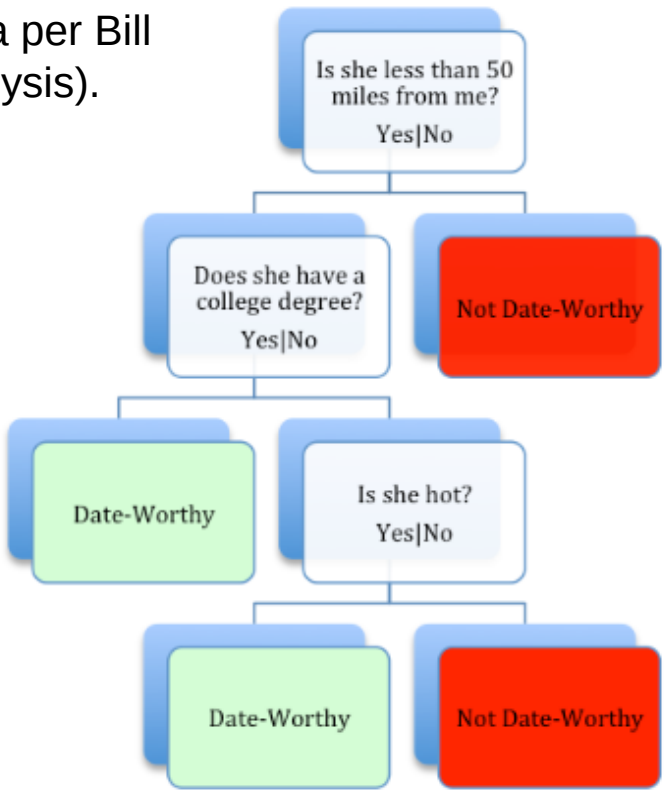
- Ideato da Leo Breiman nel 2001.
- Appartiene alla famiglia di metodi di Bagging (Boostrap Aggregating):
  - Estrazione con re-imbussolamento di un sottoinsieme S_j di pattern dal training set (tipicamente i 2/3 del totale).
  - Addestramento del classificatore C_j sul sottoinsieme S_j
  - Fusione dei classificatori (e.g., majority vote rule, somma)
- In Random Forest i singoli classificatori sono classification tree (alberi di classificazione):
  - I classification tree sono uno strumento molto utilizzato in applicazioni con pattern categorici o mixed (es. Data Mining). Per pattern numerici «puri» in genere non sono competitivi con SVM se considerati singolarmente.
  - Esistono diversi tipi di alberi, tra cui: ID3, C4.5, CART (default in Random Forest).
  - CART è un albero binario in cui ogni nodo divide i pattern del training set (assegnandoli ai due nodi figli). La suddivisione si basa sul confronto di una singola feature con una soglia.
  - Per la «crescita» dell'albero a partire da un training set, si sceglie (in modo greedy) ad ogni livello la coppia (feature, soglia di suddivisione) che meglio separa le classi. Vedi [A. Géron] per maggiori dettagli.
  - Per la classificazione di un nuovo pattern 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): majority vote rule.

Esempi di classification tree

- Sopravvivenza passeggeri del Titanic (fonte Wiki).
  - Classi: died e survived
  - **sibsp**: numero familiari a bordo.
  - I numeri sotto le foglie indicano la probabilità di sopravvivenza (sinistra) e la percentuale di campioni nella foglia (destra).



- La partner giusta per Bill (fonte GormAnalysis).



Random Forest (2)

- In Random Forest, per rendere maggiormente indipendenti i classificatori (tree):
  - 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 tree sceglierebbero con elevata probabilità le stesse variabili (quelle più discriminanti).
  - Pertanto Random Forest opera simultaneamente due tipi di bagging: uno sui pattern del training set e uno sulle features.
- Il numero di tree (iperparametro n_estimators) in una forest varia generalmente da alcune centinaia ad alcune migliaia. Aumentare n_estimators oltre al valore ottimale in genere non produce overfitting (ma rende il sistema meno efficiente).
- Gli alberi possono essere fatti crescere fino a quando lo split dei nodi termina naturalmente (in quanto non porta a una migliore separazione delle classi) oppure imponendo una massima profondità (iperparametro max_depth). Ridurre max_depth ha effetto regolarizzante e riduce overfitting.
- Grazie al bagging, le prestazioni possono essere stimate con tecnica Out-Of-Bag (OOB) che non richiede validation set separato. Infatti ciascun pattern $\mathbf{x}$ può essere utilizzato per stimare le prestazioni a partire dai soli tree nel cui training $\mathbf{x}$ non è stato coinvolto (OOB).