Machine Learning

Università Roma Tre
Dipartimento di Ingegneria
Anno Accademico 2021 - 2022

Classification:
Boosting

Sommario

- Introduzione
- Ensemble Learning
- Boosting
- AdaBoost

Boosting question

Can a set of weak learners be combined to create a stronger learner?
(Kearns e Valiant, 1988, 1989)

Sì!! —> Boosting
(Schapire, 1990)

Introduzione al Boosting

- Il Boosting è una potente tecnica per combinare molti classificatori “di base” (detti anche “weak learners”) per produrre una forma di comitato le cui prestazioni sono di gran lunga migliori di ciascuno dei classificatori.

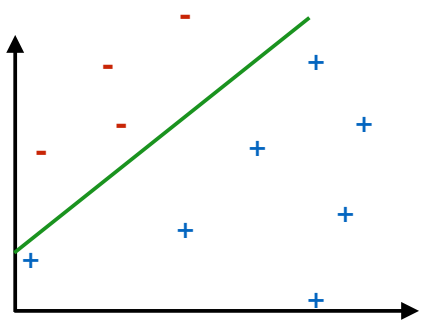
- Originariamente progettato per risolvere problemi di classificazione, può anche essere esteso alla regressione (Friedman, 2001).

Introduzione al Boosting

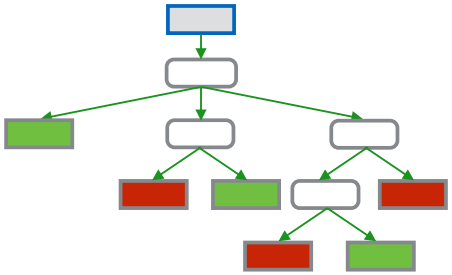
- Suo impatto per il Machine Learning:
  - approccio di default per molti task di computer vision (e.g., face detection)
  - numerose applicazioni nell'industria
  - vince molte "ML competitions" (Kaggle, KDD Cup, ecc.):
    - malware classification
    - credit fraud detection
    - sales forecasting
    - Higgs boson detection, ecc., ecc.
- Si basa sul concetto di Ensamble Learning

Weak Classifiers

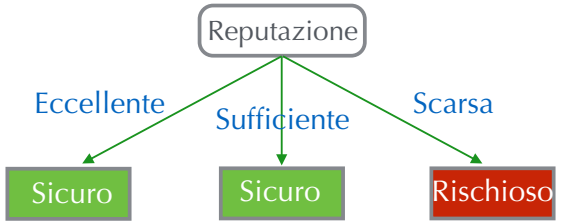
- L'idea è quella di partire da Simple (o Base o Weak) Classifiers, come ad es.:



Logistic Regression
con semplici features



Shallow
Decision Tree

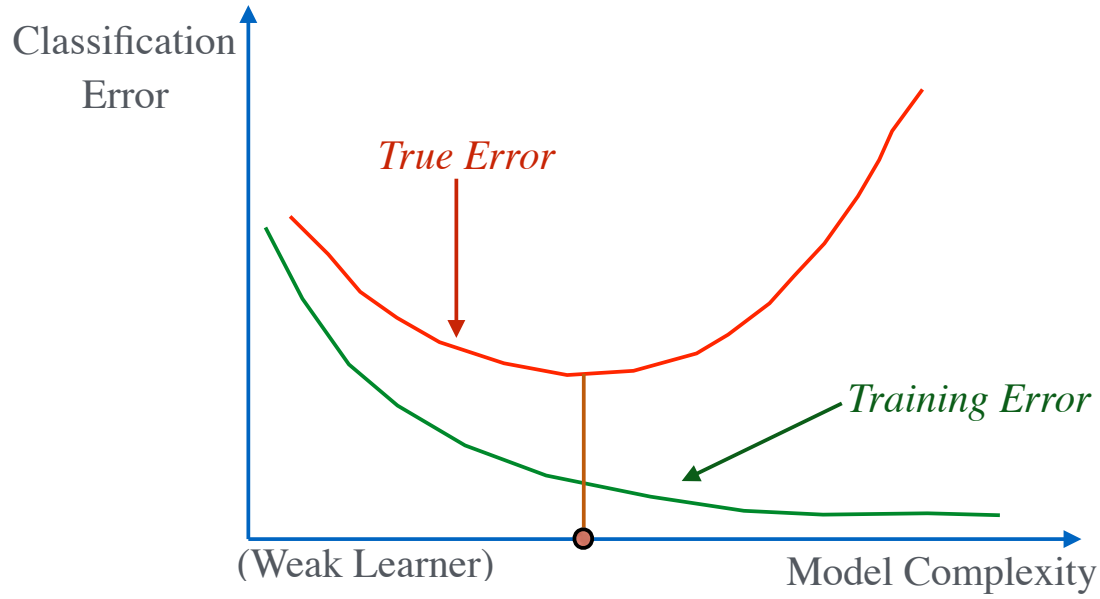


Decision Stump

- Essi in genere sono caratterizzati da bassa varianza (scarso overfitting) ma alto bias.

Andamento Errori e Bias-Variance Trade-off

- L'andamento del training error e del true error per la classification è in genere il seguente:



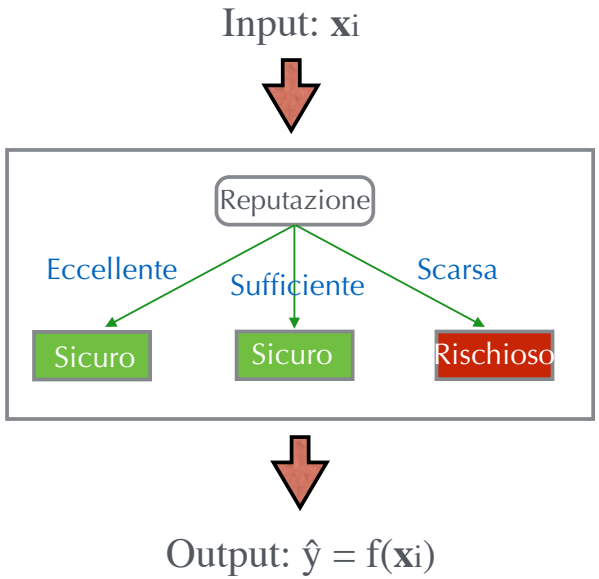
- Dobbiamo come al solito considerare il trade-off tra bias e variance.

Introduzione al Boosting

- Un approccio per migliorare un classificatore può essere quello di aggiungere più features al classificatore, ad es.:
  - logistic regression: polinomio di grado più elevato, cercando di evitare l'overfitting
  - decision trees: aumentare la profondità dell'albero
- Nel Boosting si fa qualcosa di diverso: si parte da un insieme di weak classifiers i cui risultati sono opportunamente combinati per ottenere uno strong classifier.

Ensemble Classifier

- Alla base del Boosting c'è l'idea dell'Ensemble Classifier, che ora vedremo.
- Consideriamo un weak classifier, ad esempio un Decision Stump:

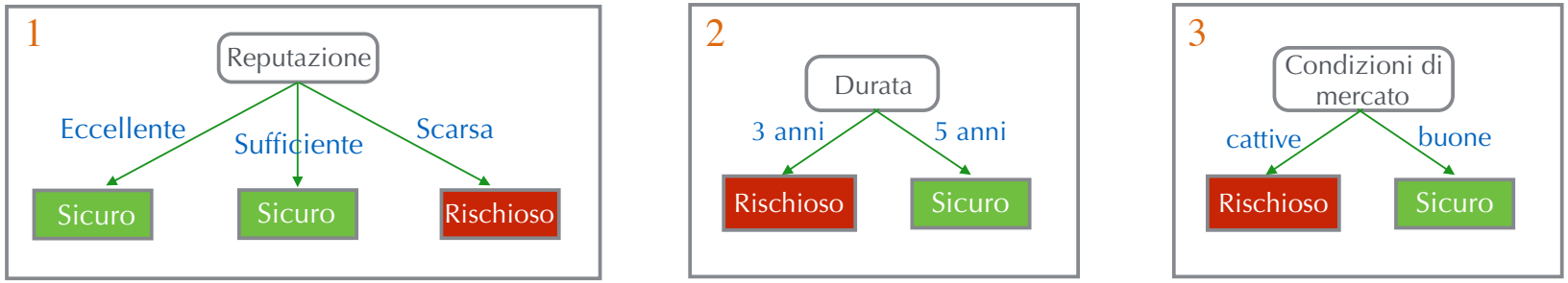


- Esso, a fronte del valore della feature d'interesse, restituisce un risultato (+1 o -1).

Ensemble Classifier

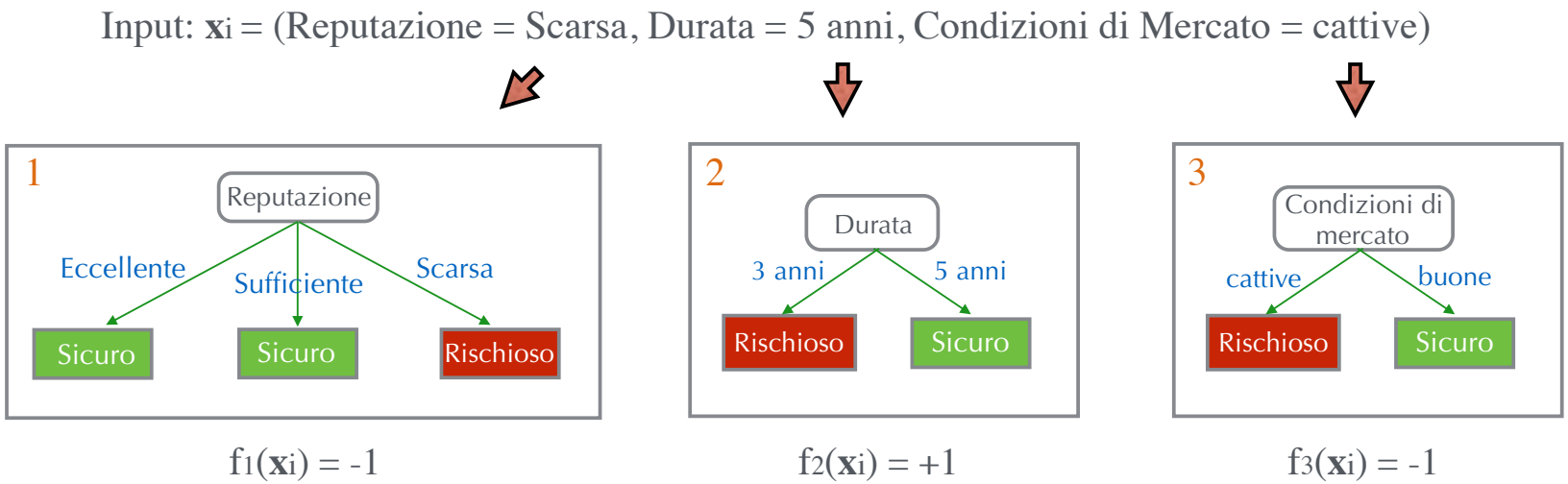
- L'idea è quella di considerare un certo numero di classificatori che, a fronte di un input, forniscono una loro previsione:

Input: $\mathbf{x}_i$ = (Reputazione = Scarsa, Durata = 5 anni, Condizioni di Mercato = cattive)



Ensemble Classifier

- L'idea è quella di considerare un certo numero di classificatori che, a fronte di un input, forniscono una loro previsione:



- Ogni classificatore esprime un voto in base al valore della feature relativa.

Ensemble Model

- I vari voti espressi dai classificatori sono combinati insieme come segue per formulare la previsione finale:

$$F(\mathbf{x}_i) = \text{sign}[w_1 * f_1(\mathbf{x}_i) + w_2 * f_2(\mathbf{x}_i) + w_3 * f_3(\mathbf{x}_i)]$$

- Se il segno è positivo la previsione vale +1, se è negativo vale -1.
- Questo è un semplice esempio di Ensemble Classifier.
- Si segnala l'importanza dei pesi w_i, che devono essere individuati mediante un processo di training.

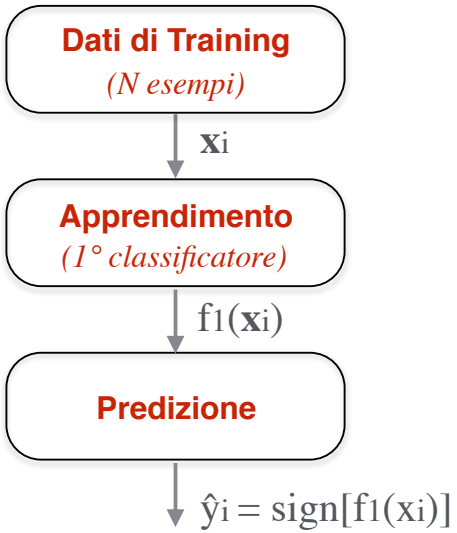
Riepilogo sugli Ensemble Classifier

- Obiettivo:
  - predire un output $\hat{\mathbf{y}}$ (+1 o -1 nell'esempio) a partire da un input $\mathbf{x}$
- Apprendimento dell'Ensemble Model:
  - Classifiers: $f_1(\mathbf{x})$, $f_2(\mathbf{x})$, …, $f_T(\mathbf{x})$
  - Coefficienti: $\hat{w}_1$, $\hat{w}_2$, …, $\hat{w}_T$
- Predizione:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

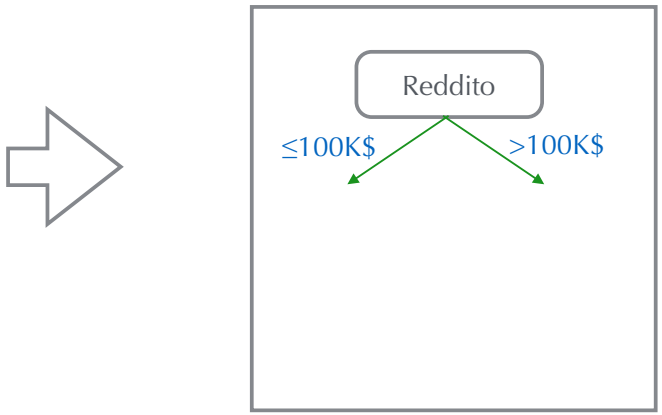
Boosting

- Consideriamo un problema di apprendimento automatico per la classificazione:



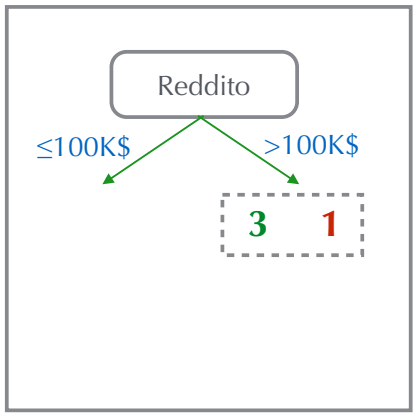
Apprendimento di un Decision Stump

Credito	Reddito	y_i
A	130K $	Sicuro
B	80K $	Rischioso
C	110K $	Rischioso
A	110K $	Sicuro
A	90K $	Sicuro
B	120K $	Sicuro
C	30K $	Rischioso
C	60K $	Rischioso
B	95K $	Sicuro
A	60K $	Sicuro
A	98K $	Sicuro



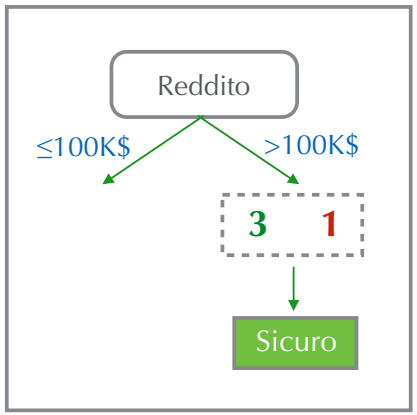
Apprendimento di un Decision Stump

Credito	Reddito	y_i
A	130K $	Sicuro
B	80K $	Rischioso
C	110K $	Rischioso
A	110K $	Sicuro
A	90K $	Sicuro
B	120K $	Sicuro
C	30K $	Rischioso
C	60K $	Rischioso
B	95K $	Sicuro
A	60K $	Sicuro
A	98K $	Sicuro



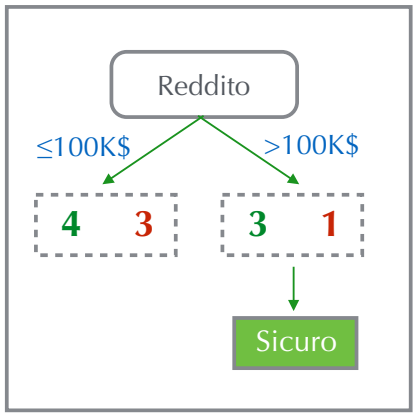
Apprendimento di un Decision Stump

Credito	Reddito	y_i
A	130K $	Sicuro
B	80K $	Rischioso
C	110K $	Rischioso
A	110K $	Sicuro
A	90K $	Sicuro
B	120K $	Sicuro
C	30K $	Rischioso
C	60K $	Rischioso
B	95K $	Sicuro
A	60K $	Sicuro
A	98K $	Sicuro



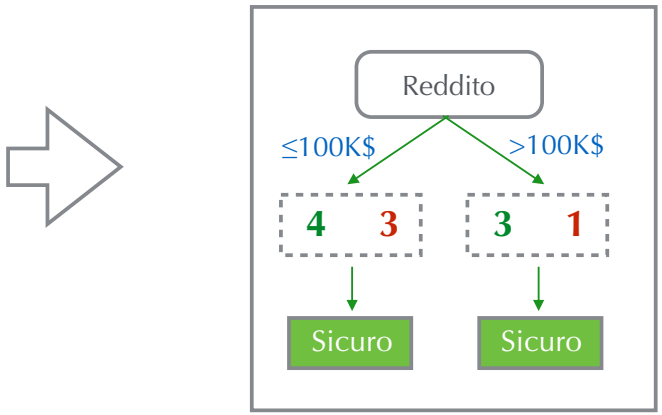
Apprendimento di un Decision Stump

Credito	Reddito	y_i
A	130K $	Sicuro
B	80K $	Rischioso
C	110K $	Rischioso
A	110K $	Sicuro
A	90K $	Sicuro
B	120K $	Sicuro
C	30K $	Rischioso
C	60K $	Rischioso
B	95K $	Sicuro
A	60K $	Sicuro
A	98K $	Sicuro



Apprendimento di un Decision Stump

Credito	Reddito	y_i
A	130K $	Sicuro
B	80K $	Rischioso
C	110K $	Rischioso
A	110K $	Sicuro
A	90K $	Sicuro
B	120K $	Sicuro
C	30K $	Rischioso
C	60K $	Rischioso
B	95K $	Sicuro
A	60K $	Sicuro
A	98K $	Sicuro

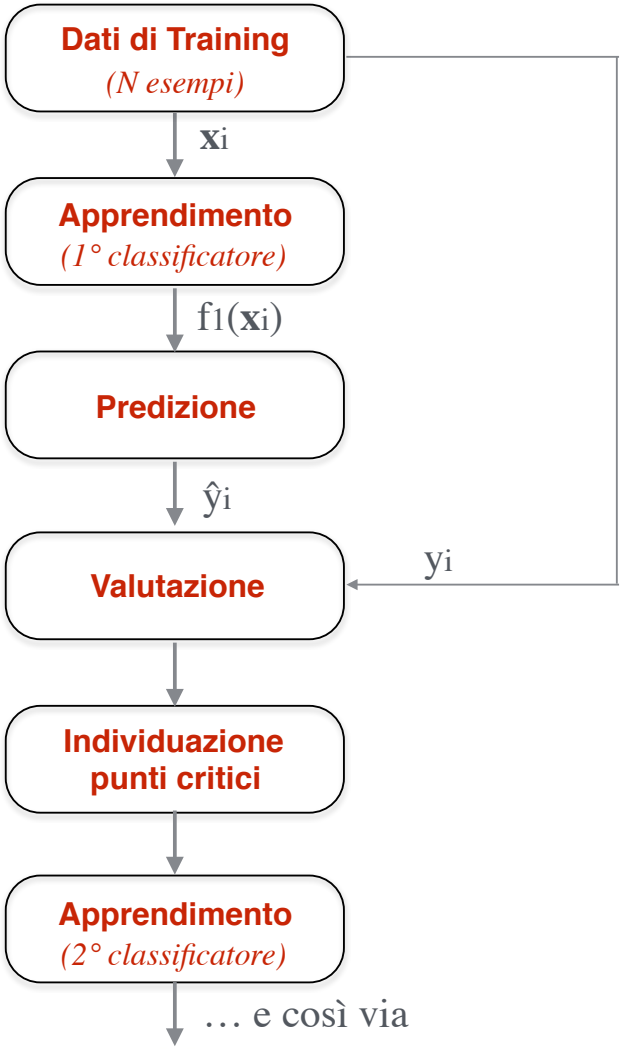


Boosting: focus sugli “hard points”

- L’esempio precedente ci mostra che il decision stump non è riuscito a catturare adeguatamente le informazioni dal numero limitato di dati disponibili.

- Quello che fa il Boosting è considerare il decision stump, lo valuta, vede come classifica i vari punti, e addestra un successivo decision stump (un successivo classificatore) con il quale si focalizza soprattutto sui punti dove il precedente classificatore era debole.

Boosting: focus sugli "hard points"



l'algoritmo di apprendimento focalizza l'attenzione sui punti "critici"

Apprendimento su Dati Pesati

[weighted data]

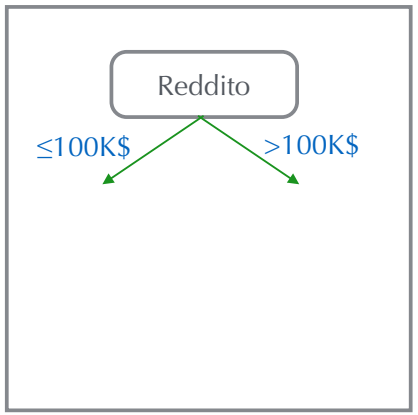
- L'idea è quella di dare maggiore attenzione ai data points ritenuti maggiormente importanti:
  - ogni data point ($\mathbf{x}_i$, y_i) è pesato mediante un α_i
  - più il punto è ritenuto importante, più è elevato il peso α_i
  - l'algoritmo di apprendimento rimane lo stesso

Apprendimento su Dati Pesati

[weighted data]

Credito	Reddito	y_i	Peso α
A	130K $	Sicuro	0.5
B	80K $	Rischioso	1.5
C	110K $	Rischioso	1.2
A	110K $	Sicuro	0.8
A	90K $	Sicuro	0.6
B	120K $	Sicuro	0.7
C	30K $	Rischioso	3
C	60K $	Rischioso	2
B	95K $	Sicuro	0.8
A	60K $	Sicuro	0.7
A	98K $	Sicuro	0.9

peso α incrementato per i punti erroneamente classificati

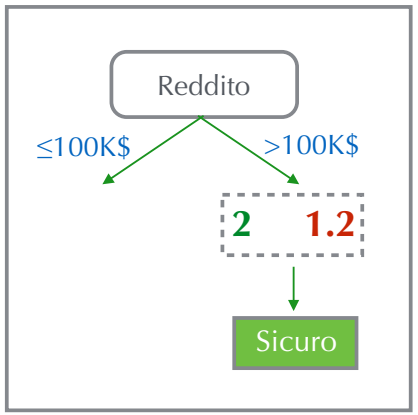


Apprendimento su Dati Pesati

[weighted data]

Credito	Reddito	y_i	Peso α
A	130K $	Sicuro	0.5
B	80K $	Rischioso	1.5
C	110K $	Rischioso	1.2
A	110K $	Sicuro	0.8
A	90K $	Sicuro	0.6
B	120K $	Sicuro	0.7
C	30K $	Rischioso	3
C	60K $	Rischioso	2
B	95K $	Sicuro	0.8
A	60K $	Sicuro	0.7
A	98K $	Sicuro	0.9

peso α incrementato per i punti erroneamente classificati

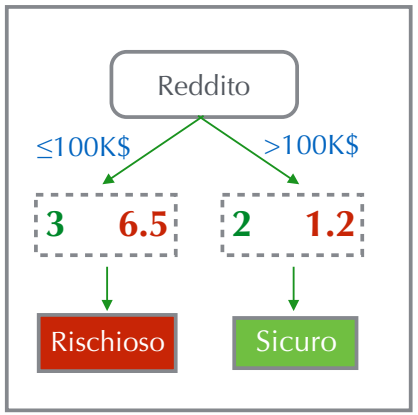


Apprendimento su Dati Pesati

[weighted data]

Credito	Reddito	y_i	Peso α
A	130K $	Sicuro	0.5
B	80K $	Rischioso	1.5
C	110K $	Rischioso	1.2
A	110K $	Sicuro	0.8
A	90K $	Sicuro	0.6
B	120K $	Sicuro	0.7
C	30K $	Rischioso	3
C	60K $	Rischioso	2
B	95K $	Sicuro	0.8
A	60K $	Sicuro	0.7
A	98K $	Sicuro	0.9

peso α incrementato per i punti erroneamente classificati



Apprendimento su Dati Pesati

[weighted data]

- Tale approccio comporta che:
  - Ogni punto i ($\mathbf{x}_i$, y_i) conta come α_i punti.
  - L'algoritmo di apprendimento rimane lo stesso.
- L'apprendimento su dati pesati non è solo relativo ai decision stumps.
- Esso si può applicare a molti algoritmi di Machine Learning
  - ad esempio, nel gradient ascent per la logistic regression:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \cdot \sum_{i=1}^{N} \phi_j(\mathbf{x}_i)\{I[y_i = +1] - P(y = +1|\mathbf{x}_i, \mathbf{w}^{(t)})\}$$

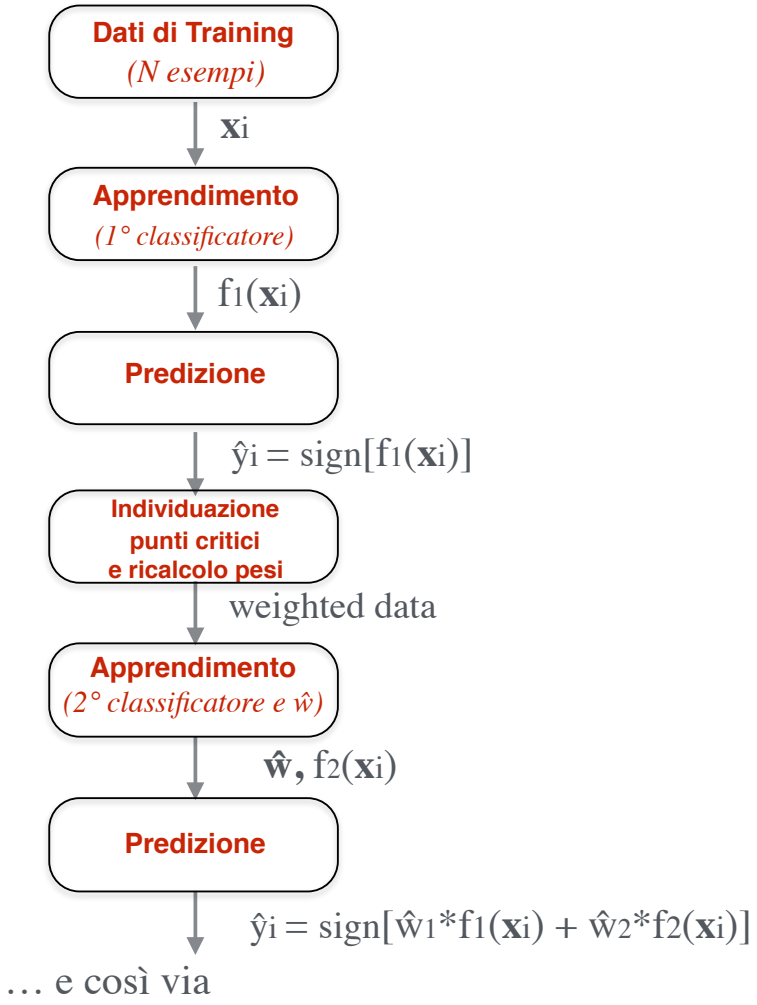
$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \cdot \sum_{i=1}^{N} \alpha_i \, \phi_j(\mathbf{x}_i)\{I[y_i = +1] - P(y = +1|\mathbf{x}_i, \mathbf{w}^{(t)})\}$$

Boosting

- Algoritmo Greedy per l'apprendimento di "ensemble" dai dati.
- Si avvale di weak learners usati come black box.
- Weak Learning Assumption: ciascun weak learner deve avere prestazioni migliori di un classificatore "random".
- Nel Boosting i base classifiers sono addestrati in sequenza.
- Per migliorare le prestazioni di un weak learner, l'algoritmo deve poter manipolare i dati in ingresso, altrimenti si ottengono sempre gli stessi risultati

Boosting framework

- Step principali:



Idea del Boosting: aggiungere via via nuovi classificatori ottimizzando i pesi per focalizzarsi sui punti critici per poi apprendere i coefficienti dei diversi classificatori.

AdaBoost

[Adaptive Boosting]

- Proposto da Yoav Freund e Robert E. Schapire nel 1996.
- I due autori hanno vinto il Gödel Prize nel 2003.
- Algoritmo estremamente utile e facile da implementare.

Riferimenti:

Freund, Y., Schapire, R. E. “Experiments with a new Boosting Algorithm”, in: *Thirteenth International Conference on Machine Learning*, 1996, pp. 148-156.

Freund, Y., Schapire, R. E. “A Short Introduction to Boosting”, in: J*ournal of Japanese Society for Artificial Intelligence*, 14(5), 1999, pp. 771-780.

Schapire, R.E., Freund, Y. *Boosting - Foundations and Algorithms*. The MIT Press, 2012.

AdaBoost

[Adaptive Boosting]

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:

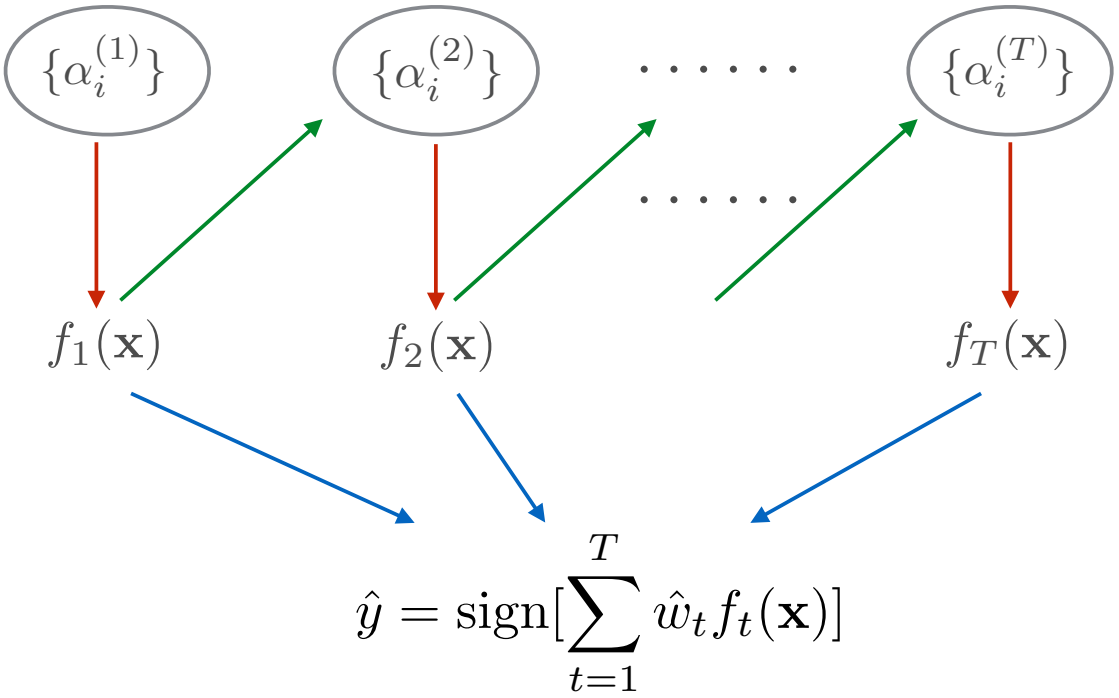
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i (minimizza funzione di costo)
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i

- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

Boosting framework

- Evidenziamo qui il processo di training dei vari classificatori, basato su una forma pesata dei punti del training set (linee rosse).
- Ogni peso dipende dalle prestazioni del precedente classificatore (linee verdi)



AdaBoost

[Adaptive Boosting]

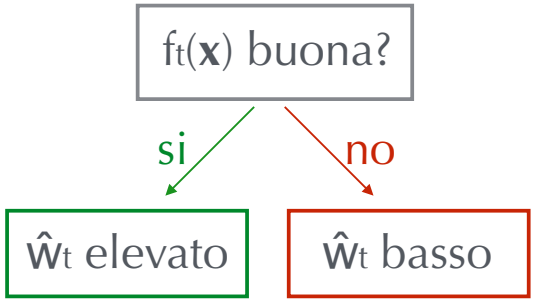
- Dobbiamo risolvere i seguenti due problemi:

  1. come calcolare il coefficiente $\hat{w}_t$ (qual è la mia "fiducia" in $f_t(\mathbf{x})$?)

  2. come ricalcolare i pesi α_i (individuare i punti "critici")

1° problema: Calcolo del coefficiente $\hat{w}_t$

- Il peso $\hat{w}_t$ rappresenta un "grado di fiducia" nella f_t. Pertanto:



- Una funzione è considerata "buona" se ha un basso training error
- Vediamo come misurare l'errore nel caso di dati "pesati" ("weighted classification error")

Weighted Classification Error

- La misura di un errore pesato è simile a quella di un errore calcolato su dati non pesati.
- Vediamo un semplice esempio:

Data point i	y_i	α_i	$\hat{y}_i$	risultato
1	+1	1.2	+1	👍
2	-1	0.5	+1	👎
3	-1	0.7	-1	👍
…	…	…	…	

peso previsioni corrette	**1.9**
peso previsioni errate	**0.5**

Weighted Classification Error

- Peso totale degli errori = $\sum_{i=1}^{N} \alpha_i I[\hat{y}_i \neq y_i]$
- Peso totale di tutti i data points = $\sum_{i=1}^{N} \alpha_i$
- L'errore "pesato" misura la frazione del peso degli errori:

$$\text{weighted\_error} = \frac{\text{peso totale degli errori}}{\text{peso totale di tutti i data points}} = \frac{\sum_{i=1}^{N} \alpha_i I[\hat{y}_i \neq y_i]}{\sum_{i=1}^{N} \alpha_i}$$

- Miglior valore: 0.0 Peggior valore: random classifier

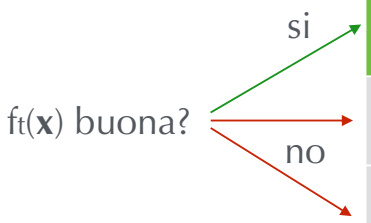
Calcolo del coefficiente $\hat{w}_t$

[per il classificatore $f_t(\mathbf{x})$]

- La formula usata in AdaBoost è la seguente:

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

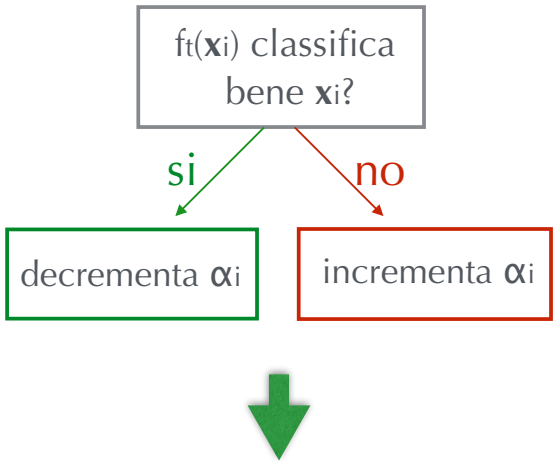
- Vediamo un esempio:



$\text{weighted\_error}(f_t)$ sui training data	$\frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)}$	$\hat{w}_t$
0.01	(1 - 0.01)/0.01 = 99	+2.3
0.5	(1 - 0.5)/0.5 = 1	0
0.99	(1 - 0.99)/0.99 = 0.01	-2.3

2° problema: Ricalcolo pesi alfa

- Come sappiamo, dobbiamo focalizzarci soprattutto sui data point dove la funzione commette errori:



$$\alpha_i \leftarrow \begin{cases} \alpha_i \cdot e^{-\hat{w}_t} & \text{se } f_t(\mathbf{x}_i) = y_i \\ \alpha_i \cdot e^{\hat{w}_t} & \text{se } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

2° problema: Ricalcolo pesi alfa

- Vediamo un esempio:

$$\alpha_i \leftarrow \begin{cases} \alpha_i \cdot e^{-\hat{w}_t} & \text{se} \quad f_t(\mathbf{x}_i) = y_i \\ \alpha_i \cdot e^{\hat{w}_t} & \text{se} \quad f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$f_t(x_i) = y_i$?	$\hat{w}_t$	moltiplicare α_i per:	implicazioni
SI (corretto)	+2.3	0.1	diminuisci l'importanza del punto
SI (corretto)	0	1	mantieni la stessa importanza
NO (errore)	+2.3	9.98	aumenta l'importanza del punto
NO (errore)	0	1	mantieni la stessa importanza

Normalizzazione pesi alfa

- La normalizzazione dei pesi α_i è suggerita dal fatto che:
  - se la funzione sbaglia spesso la classificazione di $\mathbf{x}_i$, il peso α_i tende ad assumere valori molto alti
  - se la funzione prevede spesso correttamente la classificazione di $\mathbf{x}_i$, il peso α_i tende ad assumere valori molto bassi
- Tutto ciò può causare instabilità numerica dopo varie iterazioni.
- Si normalizza come segue in modo tale che, dopo ogni iterazione, la somma dei pesi α_i risulti sempre uguale ad 1:

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i
- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i
- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i
- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i \cdot e^{-\hat{w}_t} & \text{se } f_t(\mathbf{x}_i) = y_i \\ \alpha_i \cdot e^{\hat{w}_t} & \text{se } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i

- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

AdaBoost

- **for** i = 1, 2, …, N:

  $\alpha_i = 1 \backslash N$

- **for** t = 1, 2, … T:
  - apprendi $f_t(\mathbf{x})$ con i pesi α_i
  - calcola il coefficiente $\hat{w}_t$
  - ricalcola i pesi α_i
  - normalizza i pesi α_i

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i \cdot e^{-\hat{w}_t} & \text{se } f_t(\mathbf{x}_i) = y_i \\ \alpha_i \cdot e^{\hat{w}_t} & \text{se } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

- Calcola la predizione finale:

$$\hat{y} = \text{sign}[\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})]$$

Riferimenti

Schapire, R.E. "The Strength of Weak Learnability", in: *Machine Learning*, **5**(2), 1990, pp. 197–227.

Freund, Y., Schapire, R. E. "Experiments with a new Boosting Algorithm", in: *Thirteenth International Conference on Machine Learning*, 1996, pp. 148-156.

Freund, Y., Schapire, R. E. "A Short Introduction to Boosting", in: J*ournal of Japanese Society for Artificial Intelligence*, **14**(5), 1999, pp. 771-780.

Friedman, J.H. "Greedy Function Approximation: A Gradient Boosting Machine", in: *Annals of Statistics*, **29**(5), 2001, pp. 1189-1232.

Machine Learning: Classification, University of Washington - Coursera, 2017.

Schapire, R.E., Freund, Y. *Boosting - Foundations and Algorithms*. The MIT Press, 2012.