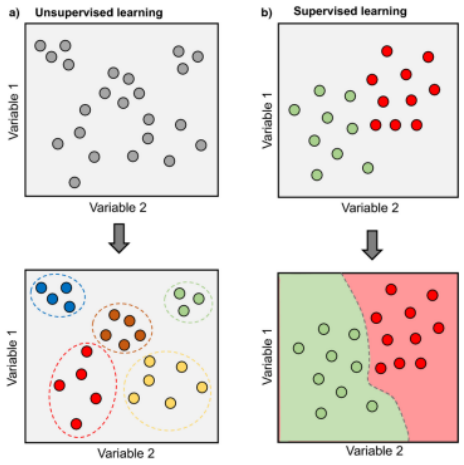
Lezione 5

Status	In progress
Attach file	5_ClassificazioneParte1 (2).pdf

1. Introduzione

La classificazione divide in gruppi un insieme di dati. A differenza del clustering, le classi sono note a priori. Pertanto, è utilizzata per assegnare ad un nuovo dato una delle classi note al modello.
Per questo motivo, la classificazione è un processo di apprendimento supervisionato (supervised learning) mentre il clustering è un processo di apprendimento non supervisionato (unsupervised learning).



Come si costruisce un classificatore?

1. Costruzione del modello: partendo da un insieme di dati chiamato training set si costruisce un modello in grado di classificare correttamente i record (o tuple) del training set.
2. Validazione del modello: si testa la bontà del modello applicandolo su un secondo set di dati, chiamato test set, di cui si conoscono le classi, confrontando per ogni tupla la classe nota e quella predetta dal modello.
3. Uso del modello: si applica il modello alla classificazione di nuove tuple.

Quali sono le principali caratteristiche?

- Accuratezza
  - del classificatore
  - del predittore
- Velocità
  - per costruire il modello
  - per usare il modello
- Robustezza: capacità di manipolare dati con rumore e/o dati mancanti
- Scalabilità: efficenza su dati presenti im memoria secondaria

Predizione

La predizione consiste nel predire un valore sconosciuto di un attributo numerico continuo (non categoriale) dei dati.
Come nella classificazione, ciò è ottenuto mediante la costruzione di un modello a partire da un training set in cui i valori dell'attributo da predire sono conosciuti.
A differenza della classificazione, il modello non è costruito per cercare di "separare" al meglio dati di classi diverse, ma per individuare una funzione matematica continua in grado di predire al meglio i valori dell'attributo continuo a partire dagli altri attributi dei dati.

2. Alberi decisionali

Sono strutture ad albero che effettuano test sui valori degli attributi per predire la classe di una tupla. Ogni nodo contiene un test su un attributo. Ogni foglia contiene un'etichetta di classe.

La classe ti una tupla si ottiene percorrendo un path dalla riadice ad una delle foglie.

Ogni path applica una regola del tipo IF-THEN per stabilire la classe: i nodi intermedi sono le condizioni legate in AND. Le regole sono esaustive e mutuamente esclusive.

Costruzione di un albero

Costruzione top-down:

1. se tutte le tuple S_x hanno la stessa classe C, si crea un nodo foglia associato a quella classe

2. altrimenti seleziona un attributo A tra quelli rimanenti ed effettua uno splitting sulla base dei valori di A, creando tanti nodi quanti sono i possibili valori di A (o raggruppandoli in base ad una soglia o altre classificazioni)
3. Per ciascuno nodo X_i figlio di X:
    a. se X_i è puro, ovvero tutte le tuple di quel nodo appartengono alla stessa classe, allora quel nodo è foglia e l'algoritmo si ferma
    b. altrimenti si applica ricorsivamente l'algoritmo sul nodo X_i

In alcuni casi si può decidere che al nodo X sia una foglia, anche se impuro: associo ad X

- l'etichetta di classe più frequente tra le tuple di X
- oppure la distribuzione di probabilità data dalla frequenza relaztiva delle classi in X

Lo splitting può avvenire:

- ATTRIBUTI CONTINUI= se gli atributi sono booleani o numerici, si sceglie una soglia: maggiore vado a sinistra, minore a destra
    - devo scegliere l'attributo continuo che massimizza la goodness
- se gli attributi sono categoriali, posso generare una diramazione binaria definendo un insieme non vuoto di valori: vado a sinistra se il valore della tupla appartiene al sottoinsieme, atrimenti a destra
    - devo scegliere il sottoinsieme che massimizza la goodness.

Albero ottimale

Come costruisco l'albero ottimale? Il problema è NP-hard, allora uso una strategia greedy:

> Scelgo l'attributo che divide le tuple in sotto-partizioni il più pure possibili

1. Se tutte le tuple hanno classe C, creo nodo foglia e gli associo C
2. Altrimenti seleziono A che massimizza la goodness
    a. Splitto A in base a che tipo di attributo è
    b. creo figli di A sulla base dello splitting
3. Applico ricorsivamente.

Se ho solo 2 classi, ordino in senso decrescente i valori degli atributi sulla base della percentuale di tuple che hanno quel valore.

Come scelgo il miglior attrbuto in base alla goodness?

Misure di goodness

Information Gain (ID3)

Seleziona l'attributo in base alla riduzione dell'entropia:

- entropia massima: le classi del nodo hanno frequenza simile (poco rilevante)
- entropia minima: i dati di un nodo dell'albero hanno tutti la stessa classe (o quasi) (nodo puro)

Dato X un nodo, e S_x l'insieme delle tuple associate. Suppongo 2 classi N e P: p sono le tuple di P, n sono le tuple N.

L'entropia è pari a:

$$H(S_x) = -\frac{p}{p+n}log(\frac{p}{p+n}) - \frac{n}{p+n}log(\frac{n}{p+n})$$

Dopo aver creato una partizione di S_x allo stesso modo calcolo l'entropia dei vari S_i (la formula è uguale). Calcolo poi l'entropia media delle partizioni, che è pari a:

$$\overline{H}_A(S_X) = \sum_{j=1}^{k} \frac{|S_j|}{|S_X} \times H(S_j)$$

L'information gain è definito come:

$$gain(A) = H(S_X) - \overline{H}_A(S_X)$$

Si sceglie come attributo di split colui che **massimizza il gain**75.

Limitazione: Risulta fortemente sbilanciato in favore di attributi con molti valori.

Gain Ratio (C4.5)

Introduce il concetto di splitInfo per ridurre il problema dell'information gain, misurando quanto l'attributo A divide i dati:

$$splitInfo(A) = -\sum_{i=1}^{k} \frac{|S_i|}{|S_X|} \log_2 \frac{|S_i|}{|S_X|}$$

Il gain ratio è dato da:

$$gainRatio(A) = \frac{gain(A)}{splitInfo(A)}$$

Gini Index (CART)

Misura la probabilità che una tupla scelta a caso da S_X sia di classe i e una diversa j.

$$gini(S_X) = \sum_{i=1}^{n} p_i(1 - p_i) = 1 - \sum_{i=1}^{n} p_i^2$$

Il Gini index dello splitting (in k sottinsiemi) è definito come:

$gini_{split}(S_X) = \sum_{l=1}^{k} \frac{|S_i|}{|S_X|} gini(S_l)$

L'attributo A che **minimizza il Gini split** è selezionato come attributo di splitting.

Pruning dell'albero

Il pruning consiste nel sostituire un intero sottoalbero con un nodo radice etichettato con la classe più frequente per evitare l'**overfitting**.

- **Pre-pruning:** in fase di costruzione dell'albero effettua lo splitting solo se la *goodness* ottenuta è al di sopra di una soglia o varia significativamente.
- **Post-pruning:** rimuove rami e nodi a costruzione dell'albero già avvenuta, sostituendo un intero sottoalbero con una foglia.
  - Pruning pessimistico (Algoritmo C4.5).
  - Cost-complexity pruning (Algoritmo CART).

Pruning Pessimistico (C4.5)

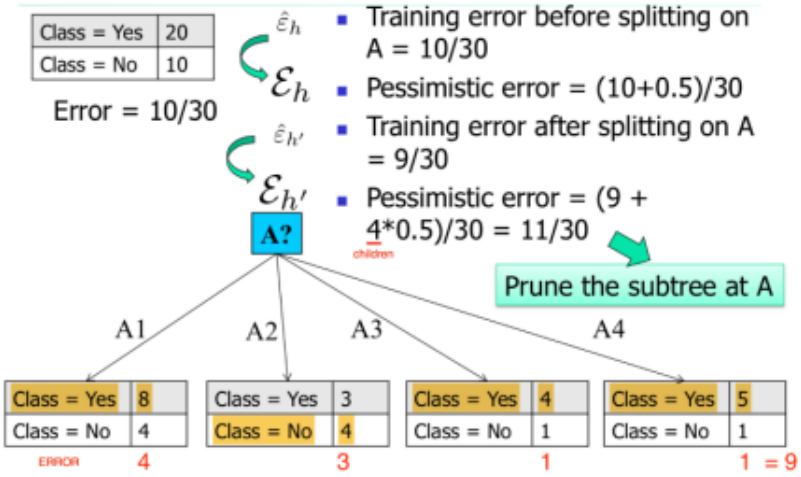
- Error rate pessimistico prima dello split:

$$\text{Error rate pessimistico } E_p(T) = \frac{|\{t \in S_X : \text{class}(t) \neq C\}| + \epsilon}{|S_X|}$$

- Error rate pessimistico dopo lo split

$$\text{Error rate pessimistico } E'_p(T) = \frac{\sum_{l=1}^{k} |\{t \in S_{X_l} : \text{class}(t) \neq c_l\}| + k\epsilon}{|S_X|}$$

Se $E'_p(T) > E_p(T)$, il sottoalbero T radicato in X viene sostituito con X che diventa la foglia.



Cost-complexity pruning (CART)

Tiene conto sia della variazione di error rate atteso che della complessità introdotta nel modello dall'aggiunta di nuovi nodi a seguito dello splitting.

Definisco il cost-complexity error:

$$\alpha = \frac{E(T) - E'(T)}{k - 1}$$

Se α è sufficientemente piccolo posso evitare lo splitting.

Vantaggi e svantaggi

- **Vantaggi:**
  - Permettono una facile interpretazione dei risultati.
  - Modelli veloci da utilizzare per classificare.
- **Svantaggi:**
  - Generalmente poco accurati, specialmente in presenza di dati ad elevata dimensionalità.
  - Soffrono di overfitting.

3. Classificatori generativi

I classificatori generativi producono un **modello probabilistico** a partire dai dati e predicono la classe di appartenenza più probabile per un nuovo dato a partire dal modello sviluppato. Si basano sul **teorema di Bayes**. Esempi: Naive Bayes, reti bayesiane.

Classificatore di Bayes

Sia X una tupla la cui classe è sconosciuta, e H_c l'ipotesi che X appartenga alla classe C.

L'obiettivo è trovare la probabilità a posteriori $P(H_c|X)$. Il teorema di Bayes afferma che:

$$P(H_C|X) = \frac{P(X|H_C) \times P(H_C)}{P(X)}$$

La classe C da assegnare a X è quella che **massimizza** $P(H_c|X)$..
Le probabilità condizionali sono molto piccole e il loro prodotto può portare a problemi di **underflow**113. Si considera il **log-likelihood**, che si traduce in una somma:

$$logP(X|C_i) = log \prod_{k=1}^{n} P(x_k|C_i) = logP(x_1|C_i) + \cdots + logP(x_n|C_i)$$

- **Vantaggi:**
  - Facile da implementare
  - Veloce.
- **Svantaggi:** L'assunzione di indipendenza condizionale può portare ad una perdita di accuratezza e può non essere vera nella pratica.

Reti Bayesane

Le reti bayesiane sono dei modelli probabilistici che permettono di modellare le dipendenze tra variabili tramite DAG.
Il likelihood è calcolato come prodotto di probabilità condizionate sulla base delle dipendenze espresse dal DAG.

4. Classificatori discriminativi

I classificatori discriminativi cercano di predire la classe direttamente a partire dai dati osservati, facendo poche assunzioni sulla loro distribuzione.
Esempi di classificatori discriminativi: Perceptron, SVM.
Vantaggi:

- Rispetto ai metodi bayesiani, una maggiore accuratezza in generale
- Robusti in presenza di errori nel training set;
- Facile calcolo del valore della funzione di decisione su un nuovo dato;

Svantaggi:

- Tempo di addestramento lungo;
- Difficile interpretare i pesi della funzione di decisione calcolata;

GENERATIVI	DISCRIMINATIVI
Sviluppa un modello probabilistico sui dati a partire da un insieme di assunzioni;	Costruisce una funzione di decisione F sui dati osservati, con pesi calcolati per ogni attributo sulla base dai valori dell'attributo stesso;
Utilizza il modello per predire la classe più probabile per un nuovo dato	Se X è un nuovo dato, si calcola F(X) e il valore ottenuto è la classe di X.

I classificatori discriminativi possono essere:

CLASSIFICAZIONE	LINEARE	NON LINEARE
NORMALE	di X è basata sul valore di una funzione ottenuta dalla combinazione lineare degli attributi di X:	la classificazione di X è effettuata usando una funzione non lineare degli attributi di X.
BINARIA	funzione lineare che separare correttamente elementi di due classi diverse (iperpiano separatore). Si parla di dati linearmente separabili;	funzione non lineare in grado di separare i dati.

Perceptron

Perceptron è un algoritmo di classificazione lineare binaria.
Cerca di separare due classi tramite un **iperpiano** che combina linearmente gli attributi del dataset.

Adotta una funzione predittiva lineare ottenuta combinando un insieme di pesi con il vettore degli attributi dell'oggetto.
Data una tupla la funzione predittiva è:

$$f(\vec{x}) = \begin{cases} 1 & \text{se } \vec{w} \cdot \vec{x} + b > \theta \\ 0 & \text{altrimenti} \end{cases}$$

dove:

- $\vec{w}$ → vettore dei pesi (uno per attributo)
- b → costante di bias
- $f(\vec{x})$→ valore della funzione di decisione
- θ → soglia decisionale (spesso 0)

Se la f risulta ≥ θ assegno classe 1, altrimenti classe 0.

Con vettore dei pesi, b costante di bias e valore soglia, sono parametri appresi dal Θ modello.
Il perceptron effettua un **apprendimento online**: processa una tupla alla volta e aggiorna i pesi ogni volta che commette un errore di classificazione.
L'algoritmo di base è cos composto:

- **Inizializzazione**Imposta tutti i pesi $w_i(0)$ a 0 o a valori casuali piccoli.
- **Per ogni tupla** (x_j, y_j) del dataset:
  - Calcola la classe predetta:

    $$y'_j = \begin{cases} 1 & \text{se } (\vec{w} \cdot \vec{x_j} + b) \geq \theta \\ 0 & \text{altrimenti} \end{cases}$$

  - Se $y'_j \neq y_j$, **aggiorna i pesi**:

    $$w_i(t+1) = w_i(t) + r \cdot (y_j - y'_j) \cdot x_{j,i}$$

- **Itera** finché:
  - l'errore medio è inferiore a una soglia γ, oppure

- è stato raggiunto un numero massimo di iterazioni.

Perceptron con soglia variabile (Θ variabile)

Se la soglia θ **non è fissata**, può essere incorporata nella funzione predittiva aggiungendo:

- un nuovo peso $w_{k+1}(t)$
- una nuova variabile dummy $x_{j,k+1} = -1$

La funzione diventa:

$$f(\vec{x}) = \sum_{i=0}^{k+1} w_i \cdot x_i$$

La **regola di aggiornamento** resta identica per tutti i pesi, inclusa la soglia.

Perceptron Multi-Classe

Il perceptron originario è binario, ma può essere **esteso a più classi** con due approcci:

1. One-Vs-One (OVO)

- Si considerano tutte le **coppie di classi** (Ci, Cj).
- Per ogni coppia si costruisce un **perceptron binario**.
- Ogni perceptron fornisce un output Oi,j, con Oi,j = −Oj,i.
- La classe assegnata a una tupla X è quella con la somma massima degli output Oi,j.

In pratica: ogni coppia di classi "vota" e vince la classe con più voti.

2. One-Vs-All (OVA)

- Si costruisce **un perceptron per ogni classe Ci**.
- Ogni perceptron distingue tra:
  - classe positiva → Ci
  - classe negativa → tutte le altre
- Per una tupla X, si calcola il valore della funzione fi(X) per ogni classe.
- X viene assegnata alla classe con fi(X) massimo.

Perceptron con dati non linearmente separabili

Il perceptron classico **funziona solo con dati linearmente separabili**.

Quando i dati non lo sono:

- si può **trasformare** lo spazio dei dati in uno spazio a **maggiore dimensionalità**, dove la separazione lineare diventa possibile;
- tuttavia, questa trasformazione:
  - aumenta i tempi di calcolo;
  - può portare a **overfitting**.

Limiti del Perceptron

1. **Non converge** se i dati non sono linearmente separabili.
2. Ogni errore fa **ruotare l'iperpiano**, rischiando di avvicinarlo troppo ai punti di una classe → bassa generalizzazione.
3. Non fornisce una misura di probabilità della classificazione.

Support Vector Machines (SVM)

Le **Support Vector Machines (SVM)** sono **classificatori discriminativi binari** che cercano l'**iperpiano ottimale** per **separare due classi** con il **margine massimo** possibile.

L'obiettivo è **massimizzare il margine** tra le due classi, cioè la distanza tra i punti più vicini delle due classi rispetto al piano di separazione.

Sono tra i metodi di classificazione supervisionata più potenti e accurati.

Date due classi di punti in uno spazio, l'SVM costruisce un **iperpiano** che:

- separa i punti delle due classi;
- **massimizza la distanza** (margine) tra l'iperpiano e i punti più vicini di ciascuna classe.

Questi punti più vicini sono detti **vettori di supporto (support vectors)** e sono quelli che **determinano effettivamente** la posizione del piano.

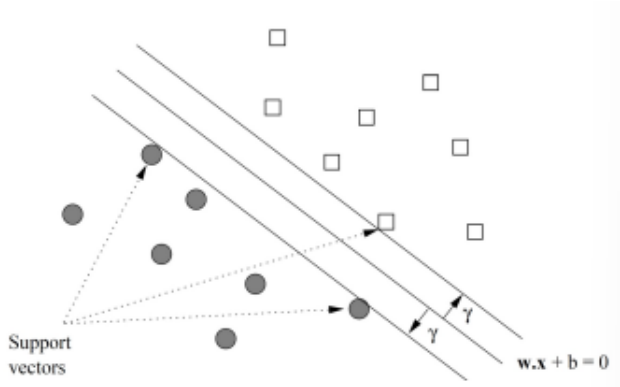
Di seguito la formulazione matematica. Sia:

- $D = (x1, y1), ..., (xn, yn) D = \{(x_1, y_1), ..., (x_n, y_n)\} D = (x1, y1), ..., (xn, yn)$ il dataset di addestramento;
- $xi \in Rn x_i \in \mathbb{R}^n xi \in Rn$ → vettori di attributi;
- $yi \in -1, +1 y_i \in \{-1, +1\} yi \in -1, +1$ → classi;

- $\vec{w} = (w1, w2, ..., wn) \vec{w} = (w_1, w_2, ..., w_n) w = (w1, w2, ..., wn)$ → vettore dei pesi;
- b → costante di bias.

L'**iperpiano separatore** ha equazione:

$$\vec{w} \cdot \vec{x} + b = 0$$



Condizioni di separabilità

Per i punti di classe:

- -1 → devono soddisfare $w \cdot x + b \leq -\gamma$, ovvero devono stare al di sotto di H_1
- -1 → devono soddisfare $w \cdot x + b \geq \gamma$, ovvero devono stare al di sopra di H_2

dove $\gamma = \frac{2}{||\omega||}$ è il margine.

👉 Massimizzare il margine equivale a **minimizzare la norma del vettore dei pesi ($||\omega||$)**

Formulazione del problema di ottimizzazione - Hard Margin

Quello descritto in seguito è una classificazione “hard margin”, poiché nessun punto può trovarsi nella regione compresa tra H1 e H2. Questo rende sensibile il modello al rumore.

Dati n esempi (x_i, y_i) con $y_i \in \{-1, 1\}$, la **formulazione normalizzata** del problema è:

$$\min_{\overrightarrow{\omega}, b} = \frac{1}{2} ||\overrightarrow{\omega}||^2$$

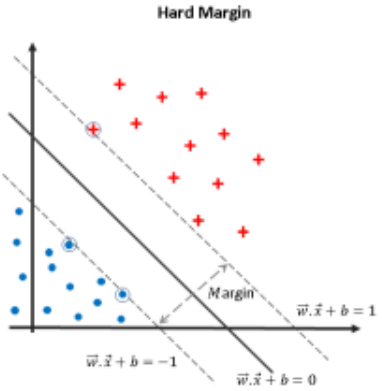
$$\overrightarrow{y}_i (w \cdot \overrightarrow{x}_i + b) \geq 1 \quad \forall i$$

- Il termine $\frac{1}{2} ||\overrightarrow{\omega}||^2$ minimizza la norma dei pesi → **massimizza il margine**.

- Il secondo vincolo assicura che tutti i punti siano **correttamente classificati** e al di fuori della banda del margine.
- Problema primale (hard margin): poiché nessun punto può trovarsi nella regione compresa tra H1 e H2. Questo rende sensibile il modello al rumore.

soggetto a:

👉 Questo è il caso **ideale**, valido **solo se i dati sono perfettamente separabili**.



Formulazione del problema di ottimizzazione - Soft Margin

Quando i dati **non sono linearmente separabili**, si ammettono alcune violazioni introducendo **variabili di slack** $\epsilon_i \geq 0$:

$$\overrightarrow{y}_i(w \cdot \overrightarrow{x}_i + b) \geq 1 - \epsilon_i$$

Le variabili εi\varepsilon_iεi rappresentano **quanto un punto viola il margine**.

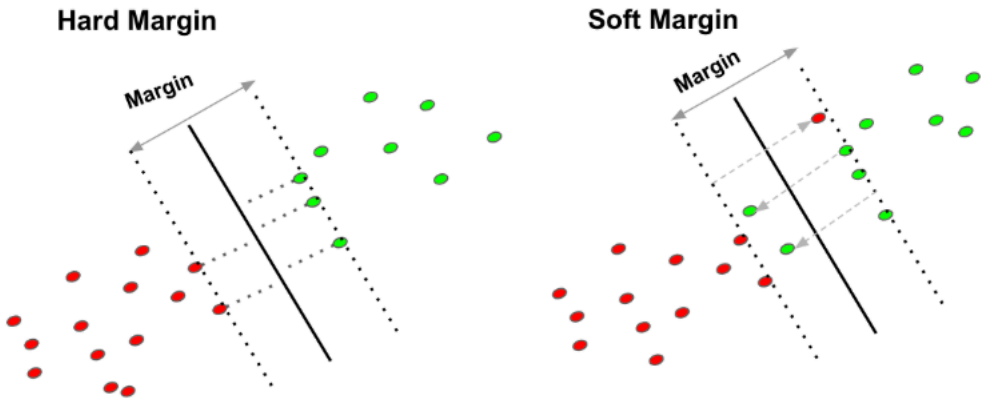
Si introduce un **parametro λ (lambda)** per bilanciare:

- **ampiezza del margine**
- **numero di violazioni tollerate**

La funzionre obiettivo diventa:

$$\begin{cases} min||\vec{w}|| \\ y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1, \ \forall \ 1 \leq i \leq n \end{cases}$$

- λ grande → penalizza molto gli errori → margine rigido
- λ piccolo → margine più morbido → modello più flessibile ma rischio overfitting



In pratica, a livello geometrico:

- I **support vectors** sono i punti che si trovano **sul bordo del margine** (o all'interno nei soft margin).
- Solo questi punti influenzano la posizione dell'iperpiano: rimuovendo altri punti, il piano non cambia.

SVM non lineari e Kernel Trick

Quando le classi **non sono linearmente separabili nello spazio originale**, si effettua un **mapping in uno spazio a dimensione superiore** tramite una funzione di trasformazione.

In questo nuovo spazio, i dati diventano **linearmente separabili**.

Tuttavia, fare questo calcolo può essere computazionalmente costoso.

Per questo si usa una **funzione kernel K**, che calcola il prodotto scalare tra le immagini mappate senza trasformarle esplicitamente:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

Grazie a questa funzione, si definisce il mapping del nuovo spazio permettendo di sostituire il prodotto scalare con la funzione kernel, risparmiando tempo di calcolo (esempi di kernel sono il kernel polinomiale, gaussiano e sigmoide)

SVM, come il perceptron, può essere esteso a problemi di classificazione multi-classe usando strategie come One–Vs–One (OVO) o One–Vs–All (OVA), addestrando più modelli binari e combinando le loro predizioni per ottenere la classificazione finale

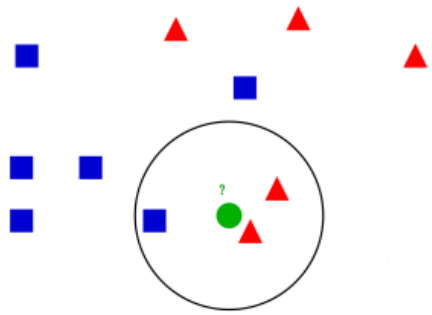
5. Apprendimento lazy

I modelli di apprendimento fino ad ora descritti sono detti eager, in cui il modello viene costruito **prima** di ricevere nuovi dati da classificare.

Nell'apprendimento lazy, il modello memorizza un training set di dati e calcola la funzione di classificazione per ogni nuovo dato.
La funzione di predizione è quindi approssimata *localmente*.

Questi metodi funzionano su grandi dataset con pochi attributi e che si aggiornano continuamente.

K-nearest neighbors (K-NN)



1. Fissato K, prendi le tuple più vicine alla tupla X da classificare
2. Assega a X la classe più ricorrente (etichetta di maggioranza) tra le classi delle k tuple scelte.
    a. Ad esempio se fisso k=3, i 3 elementi più vicini al punto da classificare (cerchio) hanno classe rosso, rosso e blu.
    b. la classe da assegnare al cerchio è rosso

Varianti ed estensioni di K-NN

K-NN pesato sulla base della distanza:

- Assegna un peso ad ogni oggetto sulla base della distanza dal punto da classificare.
- I punti più vicini hanno un peso maggiore nella classificazione.

K-NN per predire valori continui:

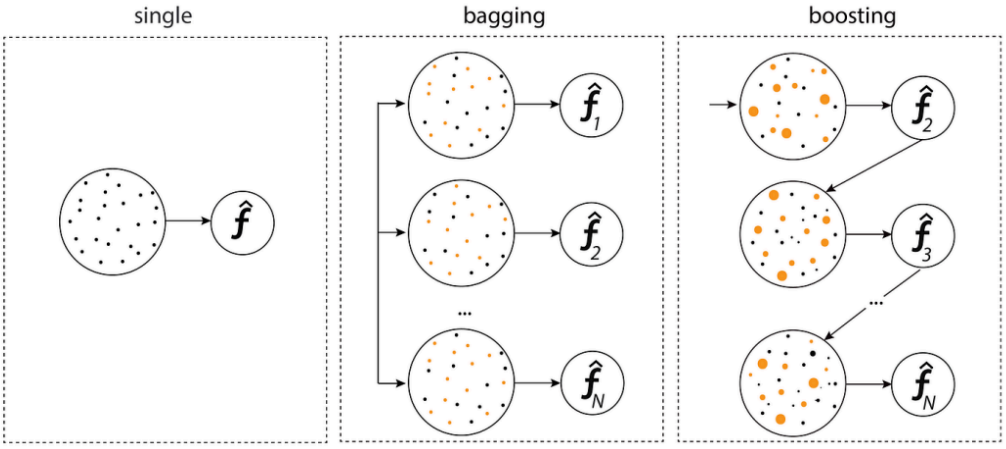
- Assegna agli attributi ignoti della tupla da classificare la media dei valori degli attributi delle tuple più vicine (rispetto agli attributi noti)

6. Apprendimento ensemble

L'ensemble learning combina due o più modelli, solitamente dello stesso tipo, per ottenere migliori performance di predizione.
Richiede molta computazione, quindi è usato per combinare algoritmi di apprendimento veloci che non hanno elevata accuratezza se presi singolarmente.
Tramite il **boostering,** in maniera iterativa addestra un classificatore e passa i risultati

della classificazione al successivo per migliorare ad ogni passo le performance della classificazione e ridurre gli errori; alla fine combina in modo pesato i risultati dei classificatori (tenendo conto della loro accuratezza) per ottenere la predizione finale.

Con il paradigma **bagging** addestra in parallelo i classificatori e combina le predizioni dei classificatori per ottenere la predizione finale.
In entrambe le tecniche, l'addestramento può essere fatto sul training set o su un campione di dati con ripetizione (bootstrap sampling o bootstrapping).



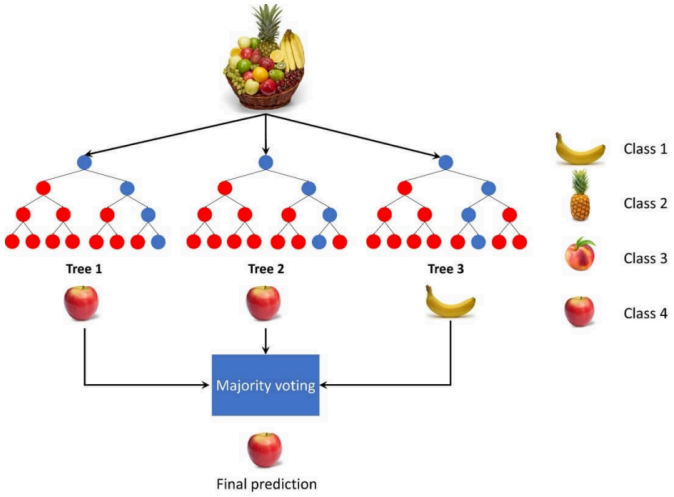
Algoritmo di Bagging

Siano $M_1, M_2, ..., M_k$ i k modelli da combinare:

1. Addestra ciascun modello su un sottoinsieme casuale del training set o su un campione di dati estratto tramite bootstrapping
2. Combina opportunamente le predizioni dei modelli:
   - Classificazione: voto di maggioranza tra le classi predette.
   - Regressione: media delle predizioni.

Random forest

Il Random Forest combina risultati di diversi alberi decisionali mediante la tecnica bagging: ogni albero è addestrato su un campione random di tuple e attributi (mediante bootstrapping).
Riduce la correlazione tra i vari alberi nel caso di attributi molto predittivi.



Algoritmo di Boosting

Siano $M_1, M_2, ..., M_k$ i k modelli da combinare:

1. Inizializza i pesi delle istanze del training set in modo uniforme.
2. Per ogni modello
    a. Addestra M sul training set ponderato.
    b. Calcola l'errore di M e aggiorna i pesi delle istanze: aumenta i pesi delle istanze mal classificate e diminuisce quelli delle istanze correttamente classificate.
3. Combina le predizioni dei modelli, pesando ciascuna predizione in base alla sua accuratezza:
    a. classificazione: si usa il voto ponderato
    b. regressione: si usa una media ponderata.

Adaboost

In questo algoritmo si usano solitamente alberi decisionali con sole due foglie chiamati stump o tronconi. Combinando opprtunamente tali classificatori semplici (e veloci) si ottiene un classificatore accurato.

1. Ogni tupla X ha un peso $w(X) = \frac{1}{N}$ con $N = \#tuple$.
2. Per $i = 1, .., k$:
    a. Crea lo stump M che minimizza l'errore ponderato sul training set.
    b. Per lo stump M, calcola un peso P basato sulla sua accuratezza:

c. Aumenta i pesi delle tuple classificate male e diminuisce quelli delle tuple classificate correttamente, dopodiché normalizza tra 0 e 1 i nuovi pesi.

3. Combina linearmente le predizioni dei modelli usando i pesi.

Gini Index Pesato:

Sia D un dataset con N tuple e k classi $C_1, C_2, ..., C_k$ e sia $w(x)$ il peso associato alla tupla X.
Data T_{C_i} l'insieme delle classi D aventi classe D_i, la probabilità di osservare una tupla di classe Ci è:

$$p_i = \frac{\sum_{X \in T_{C_i}} w(X)}{\sum_{Y \in T_D} w(Y)}$$

Gini Index dello split:

$$gini_{split}(S_X) = \sum_{i=1}^{k} \frac{|S_i|}{|S_X|} gini(S_i) \qquad gini(S_i) = 1 - \sum_{i=1}^{n} p_i^2$$

Peso dello stump:

Data $\mathrm{TotalError}$ la somma dei pesi delle tuple classificate male dllo stump D_i, allora il relativo peso è:

$$P_i = \frac{1}{2} \log \frac{1 - \mathrm{TotalError}}{\mathrm{TotalError}}$$

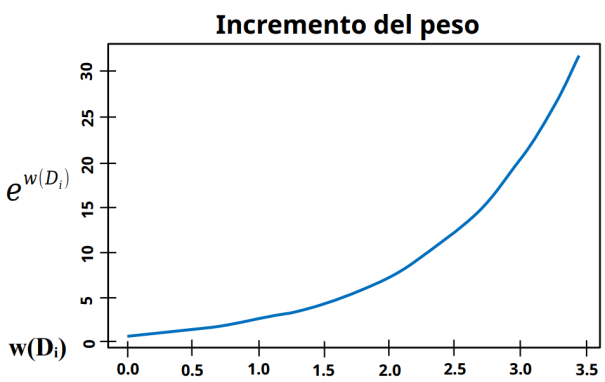
In particolare, dato il peso della tupla X all'i-esimo passo:

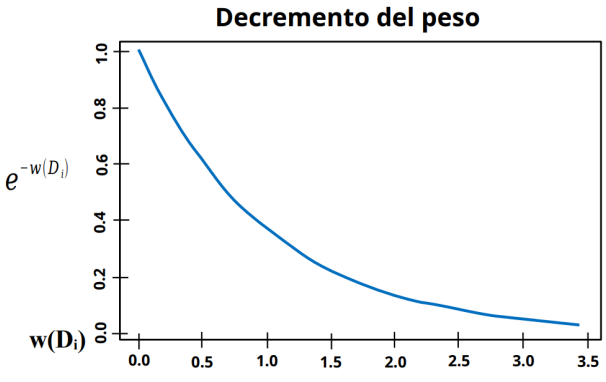
Se X NON è classificata correttamente, il peso aumenta:

$$w_{i+1}(X) = w_i(X) x \cdot e^{w(D_i)}$$

Se X è classifficata correttamente, il peso diminuisce:

$$w_{i+1}(X) = w_i(X) x \cdot e^{-w(D_i)}$$





Bagging vs Boosting

- Nel bagging il training dei classificatori può avvenire in parallelo, nel boosting è sequenziale;
- Nel bagging i classificatori sono indipendenti tra loro, nel boosting il risultato di un classificatore
  influenza il comportamento del successivo;
- Nel bagging i classificatori hanno lo stesso peso nel calcolo della classe finale, nel boosting i
  classificatori più accurati incidono maggiormente sul risultato finale;

Il bagging è più adatto per ridurre l'overfitting, il boosting è più adatto per combinare classificatori molto semplici che presi singolarmente hanno performance scadenti.

7. Validazione di un classificatore

Misure di accuratezza

La matrice di confusione è usata per rappresentare l'accuratezza della classificazione. Ogni riga contiene valori reali, ogni colonna i valori predetti.

L'elemento alla riga i-esima e colonna j-esima: contiene il numero di casi in cui il classificatore ha classificato la classe vera Ci come classe Cj.
Alta accuratezza: valori nella diagonale diversi da 0 e valori al di fuori nulli.
Sia M un classificatore, l'accuratezza acc(M) misura la percentuale di tuple classificate correttamente da M.

		Predetti			*Somma*
		Gatto	**Cane**	**Coniglio**	
Reali	**Gatto**	5	2	0	7
	Cane	3	3	2	8
	Coniglio	0	1	11	12
Somma		8	6	13	27

Consideriamo un dataset con due classi: P (positiva) e N (negativa). Indichiamo con Pos e Neg l'insieme delle tuple di classe P ed N, rispettivamente.

Definisco:

- True positive (Tpos): tuple di classe P che sono classificate come P;
- True negative (Tneg): tuple di classe N che sono classificate come N;
- False positive (Fpos): tuple di classe N che sono classificate come P (errore di tipo I);
- False negative (Fneg): tuple di classe P che sono classificate come N (errore di tipo II);

$$\text{Recall (Rec) / Sensitivity / True Positive Rate (TPR)} : \text{Rec} = \frac{|\text{Tpos}|}{|\text{Pos}|}$$

$$\text{Specificity (Spec) / True Negative Rate (TNR)} : \text{Spec} = \frac{|\text{Tneg}|}{|\text{Neg}|}$$

$$\text{False Positive Rate (FPR)} : \text{FPR} = \frac{|\text{Fpos}|}{|\text{Neg}|}$$

$$\text{False Discovery Rate (FDR)} : \text{FDR} = \frac{|\text{Fpos}|}{|\text{Tpos}| + |\text{Fpos}|}$$

$$\text{Precision (Prec)} : \text{Prec} = \frac{|\text{Tpos}|}{|\text{Tpos}| + |\text{Fpos}|}$$

$$\text{Accuracy (Acc)} : \text{Acc} = \frac{|\text{Tpos}| + |\text{Tneg}|}{|\text{Pos}| + |\text{Neg}|}$$

$$\text{F1 (armonica di Precision e Recall)} : F1 = 2 \cdot \frac{\text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$$

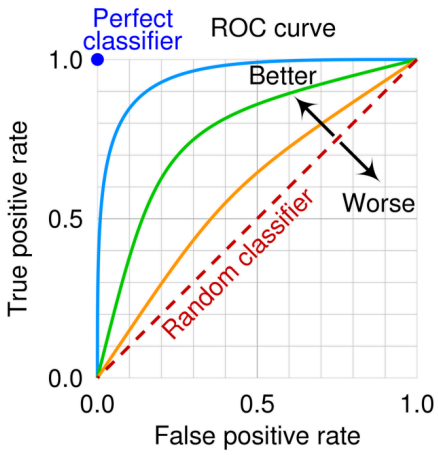
Soglia discriminativa in un classificatore binario

In alcuni casi, la distinzione tra due classi (caso binario) si basa su un valore soglia σ.
Le performance del classificatore sono valutate al variare di σ tramite curve ROC.
Esse rappresenta il True Positive Rate (TPR) in funzione del False Positive Rate (FPR), al variare di σ.
Se soglia molto alta, tutte le tuple sono classificate come negative, allora nessuna tupla negativa è classificata come positiva (FPR=0) e nessuna tupla positiva è classificata come positiva (TPR=0);

Curve ROC e di Precision-recall

Se la soglia è molto bassa, tutte le tuple sono classificate come positive: tutte le tuple negative sono classificate come positive (FPR = 1) e tutte le tuple positive sono classificate come positive (TPR = 1).

Per valori intermedi di soglia si ottengono valori di TPR e FPR compresi tra 0 e 1.
Situazione ideale: TPR aumenta fino a 1 mentre FPR si mantiene pari a 0 (miglior classificatore). Un classificatore random ha sempre valori di TPR e FPR uguali al variare di σ.



A differenza della curva ROC, la curva di Precision-Recall rappresenta la Precision in funzione della Recall al variare di σ.

Le curve ROC si utilizzano nel caso di dataset bilanciati (frequenza simile delle due classi), mentre la curva di Precision-Recall è preferibile nel caso di dataset sbilanciati.
Come indicatore dell'accuratezza del classificatore si usa l'area al di sotto delle curve ROC e Precision-Recall, detta Area Under the Curve (AUC), con $\mathrm{AUC} \in [0, 1]$.

AUC = 1 indica un classificatore perfetto.

Holdout e cross-validation

Per validare un classificatore si considerano solitamente diversi partizionamenti del dataset in training e test set e si addestra il classificatore sul training set così ottenuto. Due metodi di validazione comunemente usati sono:

- Holdout: fissata una percentuale x, partiziona il dataset in due set indipendenti:
  - training set: x%
  - test set: 100-x%

  Si addestra il modello sul training set. Poi si applica il classificatore al test set e si misura l'accuracy. Posso anche ripetere l'holdout k volte e calcolare la media delle accuracy ottenute.
- k-fold cross-validation: si partiziona il dataset in k partizioni, di dimensioni simili. Per ogni partizione addestro il classificatore formato dall'unione di tutte le partizione, esclusa l'i-esima. Testo il classificatore sull'i-esima.

8. Regressione

La regressione stima la relazione tra una o più variabili indipendenti (chiamate variabili predittive o predittori) e una variabile dipendente (chiamata variabile risposta).

Regressione lineare semplice

La regressione lineare semplice è il caso più elementare di regressione, in cui si cerca di modellare la relazione tra due variabili: una variabile indipendente x e una variabile dipendente y, mediante l'equzione di una retta:

$$y = w_0 + w_1 x$$

dove w_0, w_1 sono dette rispettivamente intercetta e pedenza, e sono chiamati coefficenti di regressione, e si ottengono tramite il metodo dei minimi quadrati.

Regressione lineare multipla

Coinvolge più predittori: ad esempio, in 3D, la funzione di regressione è:

$$y = w_0 + w_1 x_1 + w_2 x_2$$

Regressione non lineare

La funzione di regressione non è lineare. In alcuni casi è possibile trasformare la funzione di regressione in una funzione lineare.

Regressione logistica

La regressione logistica usa una funzione logistica per predire il valore a partire da una combinazione lineare delle variabili predittive.
Con una sola variabile predittiva, la funzione logistica è espressa come:

$$p(x) = \frac{1}{1 + e^{\frac{-(x-\mu)}{s}}}$$

Ponendo $\beta_0 = -\frac{\mu}{s}$ e $\beta_1 = \frac{1}{s}$ la funzione logistica diventa:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

La regressione consiste nel trovare i parametri β_0, β_1 che descrivano al meglio le classi osservate, ovvero i parametri che minimizzanola somma delle log-loss delle singole tuple.

Data una tupla T avente variabile predittiva x e risposta y, la log-loss di T è:

$$\text{log-loss} = \begin{cases} -\ln p(x), & y = 1 \\ -\ln\left(1 - p(x)\right), & y = 0 \end{cases}$$

Da notare che $\text{log-loss} \geq 0 \ \ \forall x$ e assume valore minimo quando la predizione è perfetta (classe predetta correttamente).