Algoritmi di classificazione

Algoritmi per l'intelligenza artificiale

Vincenzo Bonnici
Corso di Laurea Magistrale in Scienze Informatiche
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Universitá degli Studi di Parma

2025-2026

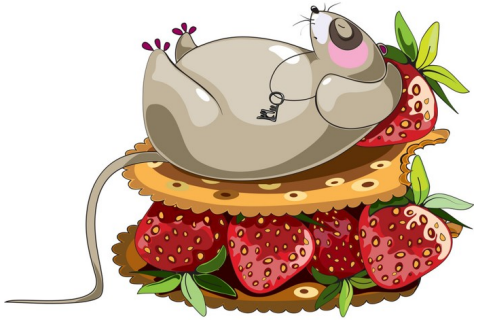
Classificazione lazy

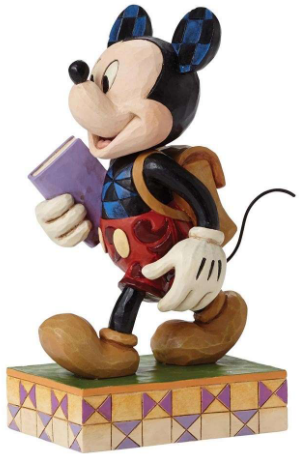
Possiamo dividere i modelli di classificazione in due tipologie

- **eager** (volenterosi) che costruisco un modello generale ed indipendente dall'input a partire dai dati di addestramento. Tale modello viene poi usato per classificare nuovi dati.

- **lazy** (pigri) che semplicemente memorizzano i dati di addestramento in forma grezza, o con piccoli processamenti, e rimandano tutto il processo di classificazione all'arrivo della nuova istanza da classificare.

Classificazione lazy





In generale i metodi lazy richiedono meno **tempo** in fase di addestramento ma molto più tempo in fase di classificazione.

Per quanto riguarda l'**accuratezza**, i metodi lazy usano uno spazio delle ipotesi molto più ricco. Essi infatti possono usare in modo più diretto funzioni locali di classificazione che forno poi delle approssimazioni a livello globale.
Al contrario, i metodi eager fanno riferimento ad una sola ipotesi per coprire tutto l'universo delle osservazioni.

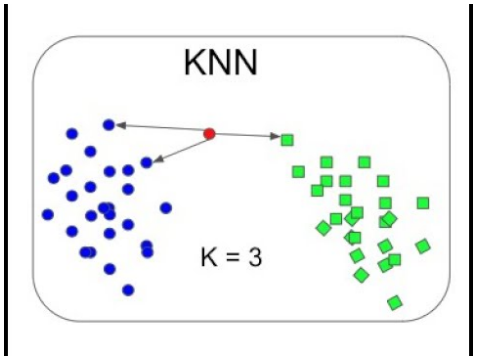
Classificazione lazy

Approcci tipicamente lazy sono

- **k-nearest neighbors** o **kNN** (letteralmente i k vicini più prossimi) in cui i data point sono rappresentati in uno spazio euclideo, ma sono possibili rappresentazioni alternative.
- **locally weighted regression** che costruisce approssimazioni locali.
- **case-based reasoning** che usa la rappresentazione simbolica e l'inferenza basata sulla conoscenze (knowledge-based inference)

Di questi, noi vedremo solamente il kNN! per poi passare ad altro.

k-nearest neighbors



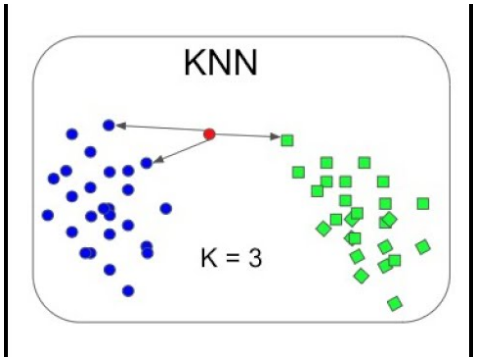
L'insieme delle osservazioni è rappresentato come insieme di data point in uno spazio n dimensionale.

Ogni dimensione può avere valori reali o discreti.

Lo spazio è uno spazio euclideo, su cui si calcolano distanze euclidee.

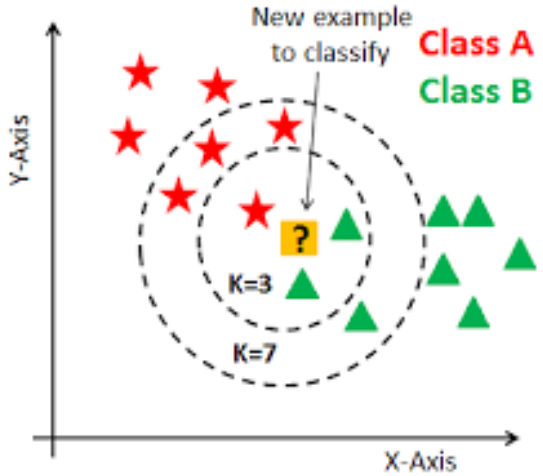
Tuttavia, è anche possibile utilizzare feature categoriali, anche se una diversa metrica deve essere ingaggiata (Jaccard, Tanimoto, Dice, Hamming o la distanza al coseno).

k-nearest neighbors



Dato un TS ed una istanza da classificare, l'algoritmo assegna a tale istanza la classe che più è rappresentata tra i suon k vicini più prossimi.

k è quindi un punto (parametero) chiave dell'algoritmo.



k-nearest neighbors

Il kNN è uno strumento **non parametrico**, ossia non fa alcuna ipotesi sulla distribuzione dei dati che analizza.

In altre parole, la struttura del modello è determinata dai dati ed è piuttosto utile, perché nel "mondo reale", la maggior parte dei dati non obbedisce ai tipici assunti teorici fatti (come nei modelli di regressione lineare, per esempio).

Di conseguenza, kNN potrebbe e probabilmente dovrebbe essere una delle prime scelte per uno studio di classificazione quando c'è poca o nessuna conoscenza precedente sulla distribuzione dei dati.

k-nearest neighbors: potere predittivo

Il potere predittivo dipende dal parametro k scelto inizialmente.

Quando k è piccolo, stiamo limitando la regione di una determinata previsione e costringendo il nostro classificatore ad essere "più cieco" rispetto alla distribuzione generale.

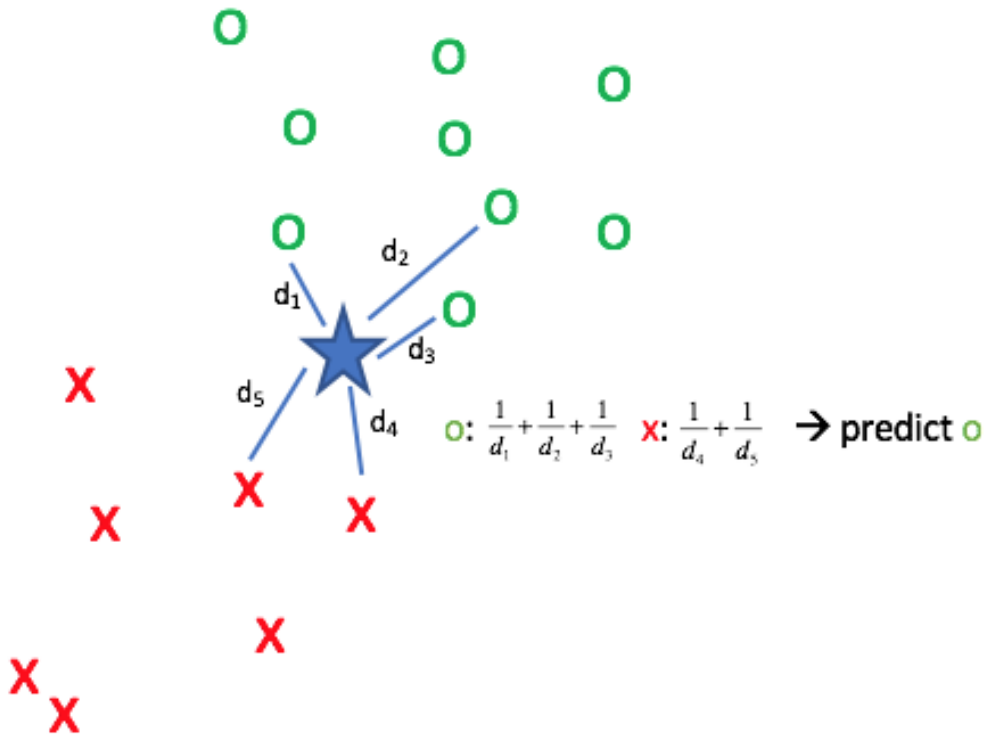
Al contrario, un k grande riduce l'impatto della varianza causato da un errore casuale, ma corre il rischio di ignorare piccoli dettagli che potrebbero essere rilevanti.

Alcuni autori suggeriscono di impostare k uguale alla radice quadrata del numero di osservazioni nel set di dati di addestramento.

Una alternativa è utilizzare la cross-validazione per identificare il valore di k con le performance migliori.

Weighted k-nearest neighbors

Per migliorare le performance di kNN si può utilizzare una versione in cui le classi dei vicini sono **pesate in base alla distanza** dei vicini stessi.



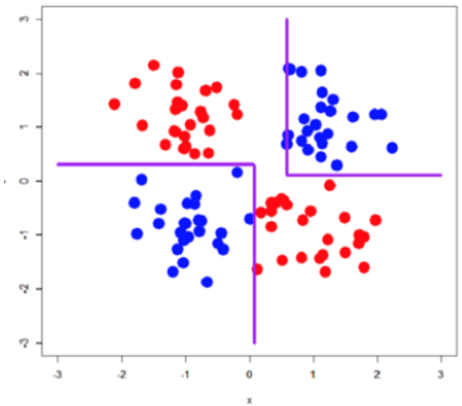
Per data set **sbilanciati** si può anche pensare di pesare l'apporto di ogni vicino con l'inverso del numero totale di punti presenti nel data set ed aventi la stessa classe del vicino in questione.

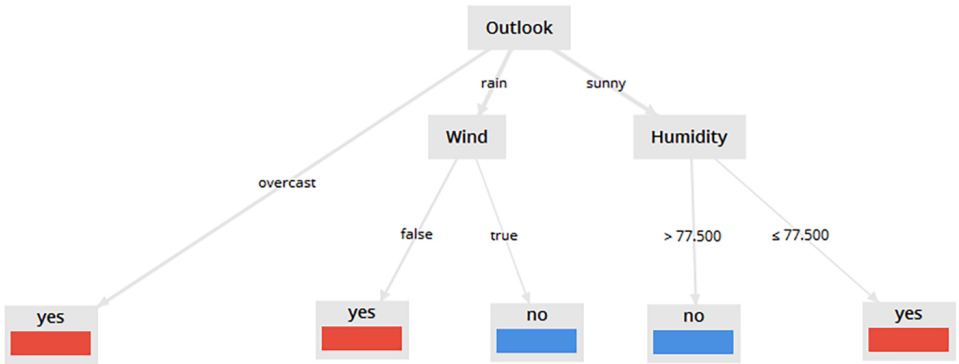
Alberi di decisione

Un albero di decisione descrive un albero di ricerca (n-ario) dove i nodi foglia rappresentano le classificazioni e le ramificazioni l'insieme delle proprietà che portano a quelle classificazioni.

Ogni nodo interno, quindi, contiene un test che stabilisce quale sotto-albero deve essere visitato.

Un test tipicamente valuta una feature (booleana, numerica, categoriale, ecc.) o combinazioni di feature.





Alberi di decisione: costruzione

Si inizia assumendo un **nodo radice** che racchiude la totalità dei dati di addestramento. Quindi si segue il seguente algoritmo:

1. se tutti i dati di training che sono all'interno di un nodo t assumono lo stesso valore sull'attributo classificatore creare un **nodo foglia** contenente tutti i dati
2. altrimenti mediante una **misura di goodness** effettuare una **partizione** S degli elementi rappresentati dal nodo
3. si sceglie la partizione S^* che (ad esempio) **massimizza** la misura di goodness
4. si creano tanti **figli** del nodo t quante sono le classi presenti in S^*
5. un nodo figlio è detto **puro** se tutti i suoi elementi assumono lo stesso valore sull'attributo classificatore.
6. si applicano **ricorsivamente** i passi precedenti sui nodi impuri o che non rappresentano una singola classe secondo una soglia di maggioranza.

Alberi di decisione: costruzione

Obiettivi:

- tra tutti i possibili alberi, scegliere l'albero più **semplice** a parità di performance
- costruire un albero che sia più **compatto** possibile

Ostacoli:

- trovare l'ipotesi consistente minimale è **NP-hard**

Soluzione:

- l'algoritmo proposto implementa una strategia ricorsiva divide-et-impera ma è di tipo **greedy** e non garantisce l'ottimalità
  - **decisione principale**: scegliere l'attributo su cui biforcare in modo che
    - divida in insiemi che sono il più possibile puri
    - porti più rapidamente a dei nodi foglia

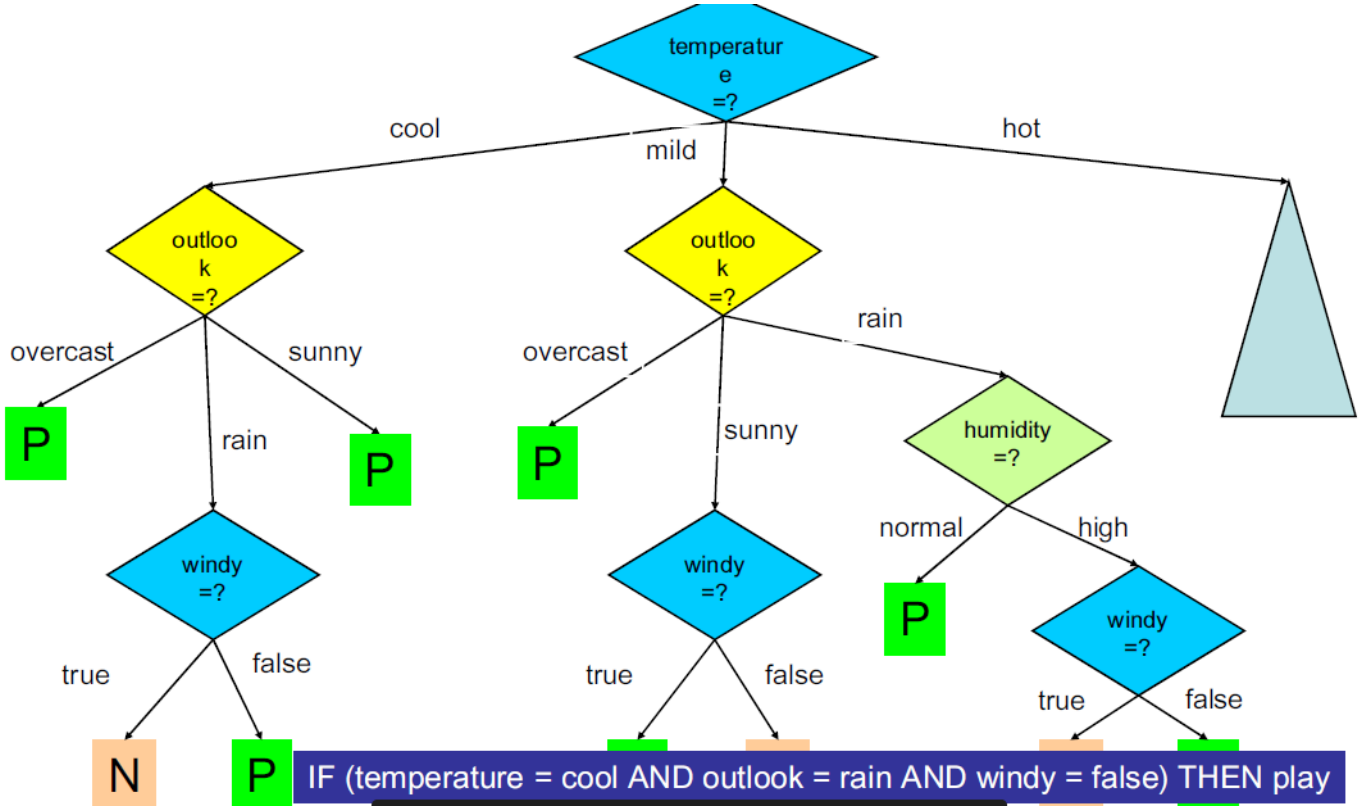
Alberi di decisione: costruzione

Esempio: si vuole predire la possibilità di giocare date alcune informazioni sul meteo.

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

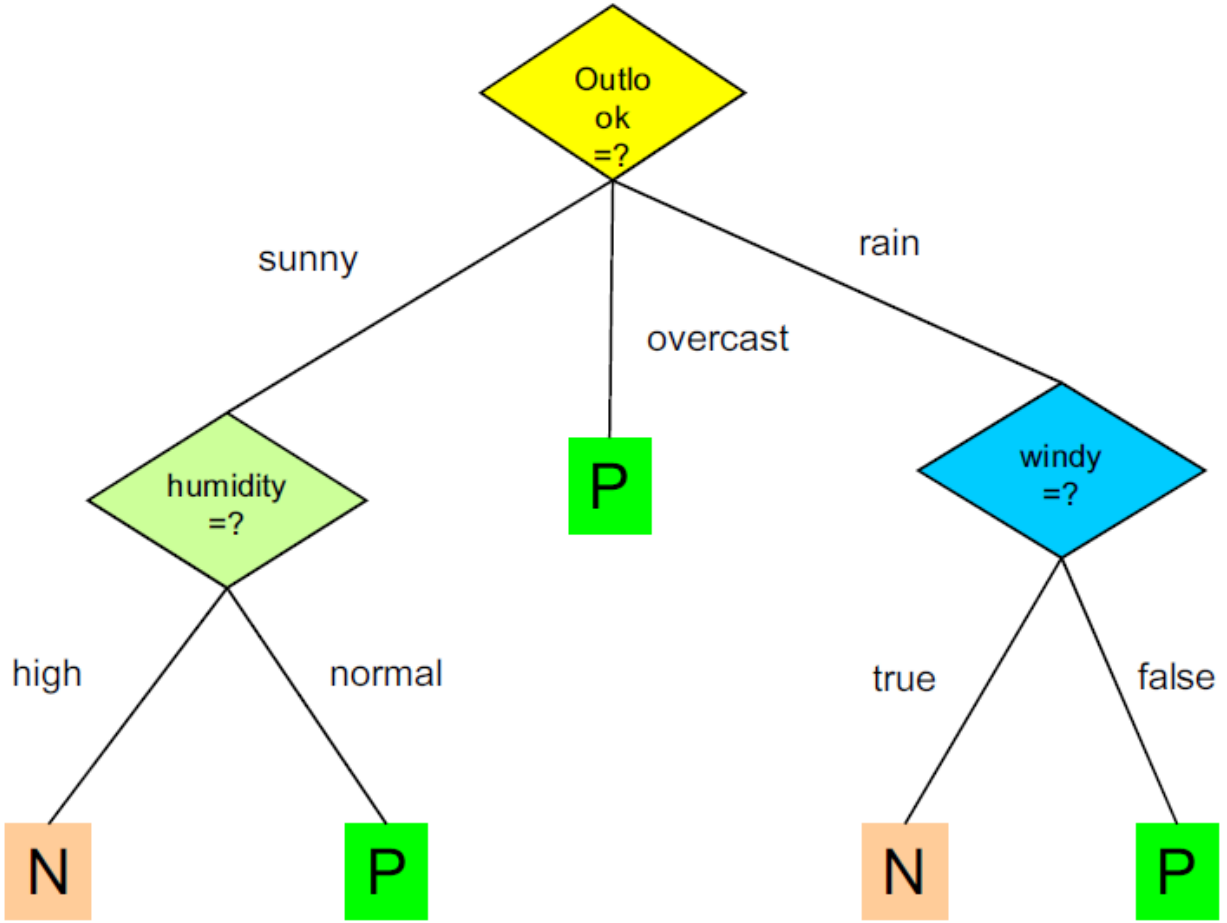
Alberi di decisione: costruzione

Un albero poco minimale



Alberi di decisione: costruzione

Un albero decisamente migliore



Alberi di decisione: goodness function

Tra le misure di goodness function più utilizzate abbiamo:

- **informaiton gain** (ID3, C4.5)
  - tutti gli attributi definiscono una categoria
  - possono essere modificati per attributi con valori continui
- **Indice di Gini** (CART)
  - tutti gli attributi con valori continui
  - assume che esistono diversi valori di splitting per ogni attributo
  - possono essere accompagnati da altri metodi quali clustering
  - possono essere modificati per attributi che definiscono categorie

Alberi di decisione: information gain

La scelta dell'algoritmo basato sull'**information gain** mira alla **riduzione progressiva dell'entropia**.

La **purezza** è definita in base a quanto un insieme di istanze è prossimo alla situazione ideale, ovvero ad una sola etichetta.

ripasso

Una variabile aleatoria X può assumere k valori x_i, per $1 \leq i \leq k$, ognuno con probabilità p_i.
$H(p_1, \ldots, p_k)$ è definita come l'incertezza media che viene rimossa all'accadere si un x_i. H misura il disordine di una variabile.
H si chiama entropia perché tale funzione risulta massima quando tutti gli eventi sono equiprobabili, non c'è struttura nell'insieme, c'è massimo disordine:

$$H = -\sum_i p_i log_b p_i$$

Alberi di decisione: information gain

Assumiamo che ci siano **due classi**, P e N.

Sia S un insieme contenente p elementi della classe P ed n elementi della classe N.

La **quantità di informazione** necessaria per decidere se un esempio arbitrario in S sta in P o N è definita come:

$$Info(p, n) = -\left(\frac{p}{p+n} log_2 \frac{p}{p+n}\right) - \left(\frac{n}{p+n} log_2 \frac{n}{p+n}\right)$$

Alberi di decisione: information gain

Assumiamo che usando l'**attributo** A , avente come valori $\{a_1, a_2, \ldots, a_v\}$, come radice dell'albero, allora S sarà partizionato in $\{S_1, S_2, \ldots, S_v\}$ insiemi.

Ora P ed N si dividono in $\{P_1, P_2, \ldots, P_v\}$ e $\{N_1, N_2, \ldots, N_v\}$.

Se S_i contiene p_i elementi di P ed n_i elementi di N, allora le informazioni necessarie per classificare gli oggetti nei vari S_i saranno del tipo $Info(p_i, n_i)$.

L'**informazione** richiesta per l'albero con radice A è ottenuta da un media pesata degli $Info$ secondo le partizioni che A impone su S:

$$Info_A(S) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} Info(p_i, n_i)$$

Alberi di decisione: information gain

Ora guardiamo il problema dal punto di vista delle classi reali e non delle partizioni imposte dagli attributi.

Sia S avente k classi differenti $C_1, \ldots, C_k$.

L'**informazione** per classificare un dato campione è

$$Info(S) = -\sum_{i=1}^{k} \frac{freq(C_i, S)}{|S|} log_2 \frac{freq(C_i, S)}{|S|}$$

detta anche entropia dell'insieme S.

Supponiamo di avere un insieme di record S, allora $Info(S)$ misura la **quantità di informazione media necessaria** per identificare la classe di un record di S.

Alberi di decisione: information gain

Sia A, un attributo, avente i seguenti valori $\{a_1, \ldots, a_n\}$.

Partizionando S nei vari S_i a seguito del test A, l'entropia di S sarà la **media ponderata delle entropie** dei singoli sottoinsiemi, ovvero

$$Info_A(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} Info(S_i)$$

La **quantità di informazione guadagnata dal partizionamento** di S secondo l'attributo A sarà:

$$Gain(A) = Info(S) - Info_A(S)$$

L'attributo A è selezionato se l'**information gain** è massimo.

Questo significa selezionare l'attributo A tale che $Info_A(S)$ sia minimo in quanto $Info(S)$ è lo stesso per tutti gli attributi di quel nodo.

Alberi di decisione: information gain

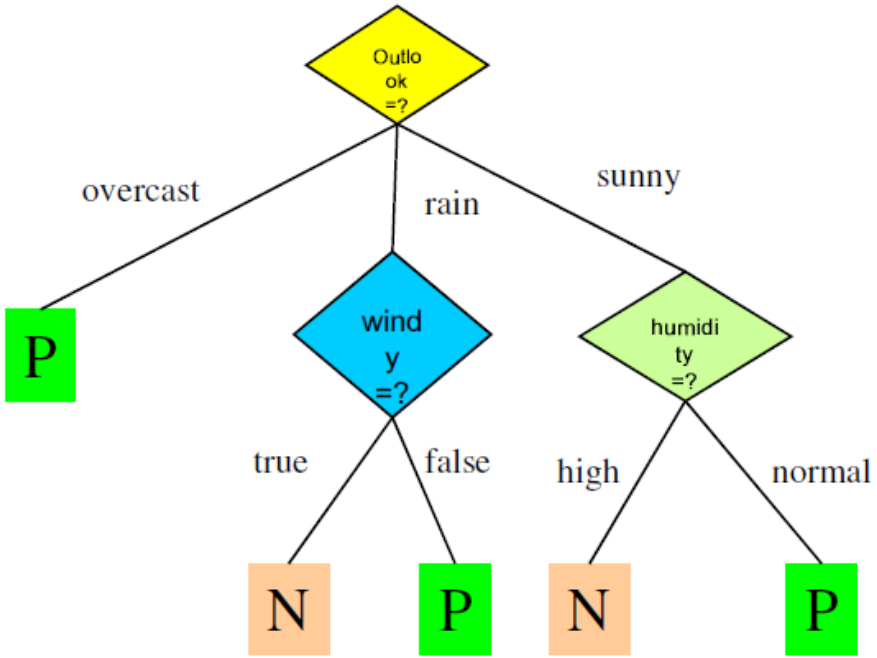
Esempio. L'attributo *outlook* è scelto come radice, infatti:

$gain(outlook) = 0.246$

$gain(humidity) = 0.151$

$gain(windy) = 0.0.48$

$gain(temperature) = 0.029$



Outlook	Tempreature	Humidity	Windy	Class
overcast	hot	high	false	P
overcast	cool	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
rain	mild	normal	false	P
rain	mild	high	true	N
sunny	hot	high	false	N
sunny	hot	high	true	N
sunny	mild	high	false	N
sunny	cool	normal	false	P
sunny	mild	normal	true	P

Alberi di decisione: information gain

Problema

L'information gain è fortemente **sbilanciato** in favore dei test che hanno molti esiti.

In determinati casi si verifica che un test, che può essere molto discriminante ai fini della divisione dell'albero e quindi con un forte potere predittivo, non venga effettuato perché si basa su un attributo con pochi valori possibili, in favore di un altro test con molti esiti possibilmente poco significativi in termini di predizione.

Soluzione

Normalizziamo il gain in base alla quantità di informazione di S stesso.

Alberi di decisione: information gain

Consideriamo l'**informazione** contenuta da un messaggio sulla probabile divisione dell'insieme S negli insiemi S_i.

Tale informazione sarà data da:

$$-log_2 \frac{|S_i|}{|S|}$$

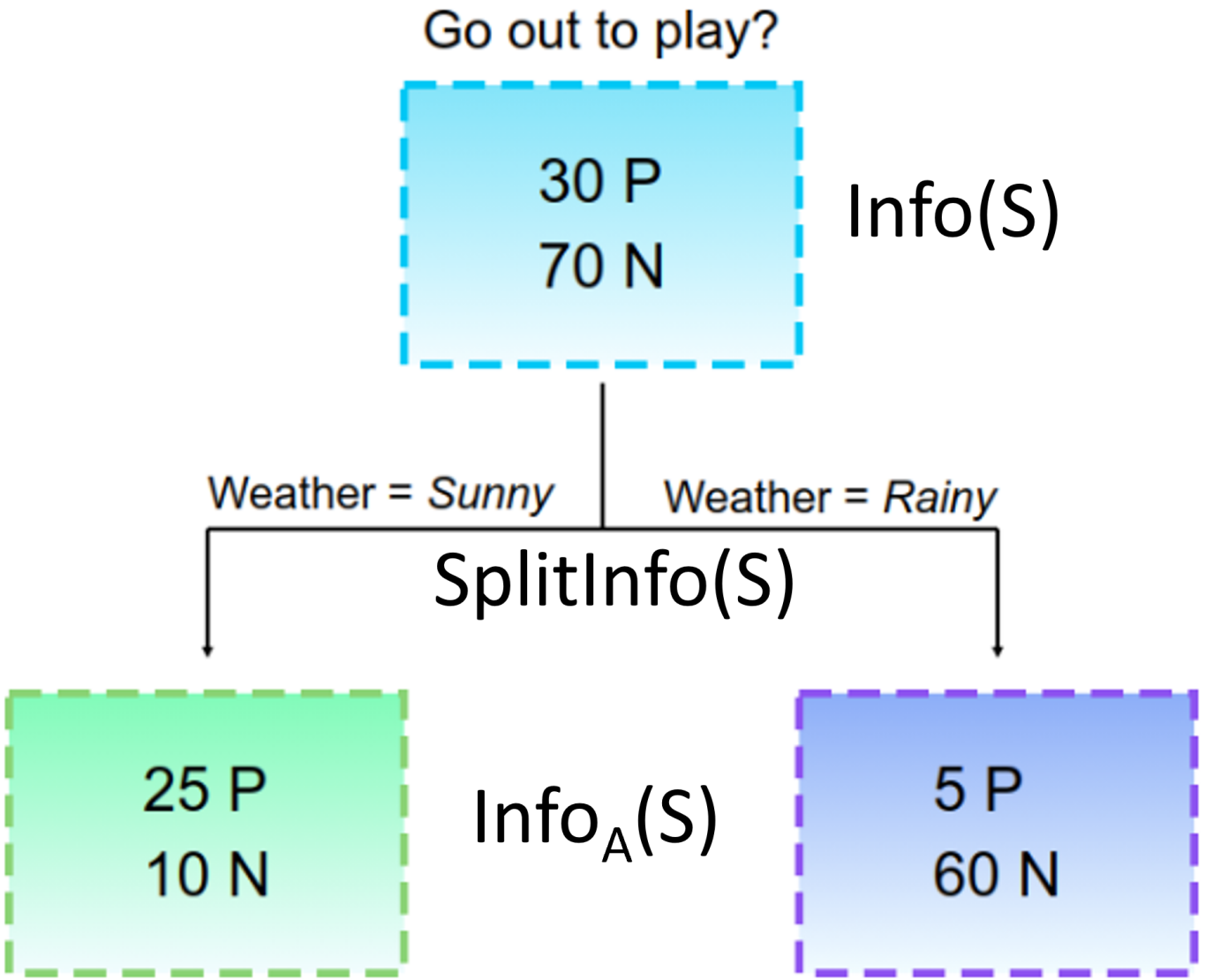
E per analogia con la definizione di $Info(S)$ si ha

$$SplitInfo(A) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}$$

Che rappresenta la potenziale informazione generata dalla divisione di S. Da quì, il *gain* **normalizzato** secondo $SplitInfo(A)$:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

Alberi di decisione: information gain



Alberi di decisione: information gain

GainRatio non risolve tutti i **problemi**. Infatti potrebbe succedere che attributi significativi, ma che assumono qualche valore in più rispetto agli altri, potrebbero essere sfavoriti.

Di solito la strategia utilizzata e la seguente:

1. si calcola il guadagno per ogni attributo
2. si calcola la media dei guadagni calcolati
3. si selezionano SOLO gli attributi che hanno guadagno al di sopra della media
4. si sceglie, fra gli attributi selezionati, quello che ha *GainRatio* maggiore

Alberi di decisione: information gain

Attributi continui

Nel caso di test su **attributi di tipo continuo**, l'esito è quello di una comparazione con un **valore soglia** Z, per cui risultati saranno due: o il valore A dell'attributo è maggiore di Z o non lo è ($A > Z$ o $A <= Z$).

Il criterio introduce anche un'altra restrizione, ovvero che per ogni divisione i due più piccoli insiemi risultanti devo avere un numero minimo stabilito di elementi.

La ricerca di questa soglia per effettuare il test viene fatta valutando tutti i valori assunti dall'attributo in questione in tutti i casi presenti nel training set.

Questi valori vengono raggruppati in un insieme ordinato $\{v_1, v_2, \ldots v_m\}$ di conseguenza, scelto v_i come valore soglia, l'insieme verrebbe partizionato in $\{v_1, v_2, \ldots, v_i\}$ e $\{v_{i+1}, vi + 2, \ldots, v_m\}$.

Attributi con valori mancanti

Come procedere?

- Assegnare il valore più comune all'attributo nullo
- Assegnare la probabilità ad ognuno dei possibili valori

In pratica il $Gain(A)$ può essere riscritto come:

$$Gain(A) = F \cdot (Info(S) - Info_A(S))$$

dove F è la percentuale di casi nel training set per cui A è noto.

Va ricalcolata $SplitInfo(S)$, e di conseguenza anche il $GainRatio$. Questo avrà effetto sul partizionamento.

Alberi di decisione: information gain

Attributi con valori mancanti

Possiamo anche pesare i singoli record.

Se il record ha un valore sconosciuto, il peso è proprio la probabilità che il valore sia v_i.

Ogni sottoinsieme S_i è ora una collezione di casi con pesi possibilmente frazionali, così che $|S_i|$ potrebbe essere reinterpretata come la somma di pesi frazionali dei casi nel sottoinsieme.

In generale, un caso di S con peso w il cui esito non è conosciuto è assegnato a ogni sottoinsieme S_i con peso w * probabilità che l'esito sia v_i.

Alberi di decisione: indice di Gini

Dato un insieme di oggetti S contenente n classi, l'**indice di Gini** è definito come:

$$gini(S) = 1 - \sum_{j=1}^{n} p_j^2$$

dove p_j è la frequenza della classe j-esima di S.

Esempio: date due classi P e N, tale che $S = P \cup N$ contenenti rispettivamente p ed n elementi, calcoliamo

$p_P = \frac{p}{p+n}$

$p_N = \frac{n}{n+p}$

$gini(S) = 1 - p_P^2 - p_N^2$

Alberi di decisione: indice di Gini

Se S è diviso in due sottoinsiemi S_1 ed S_2, rispettivamente con dimensione N_1 ed N_2, l'**indice di Gini dello split** è definito come

$$gini_{split}(S, a) = \frac{N_1}{N} gini(S_1) + \frac{N_2}{N} gini(S_2)$$

dove c è l'attributo che divide S in due insieme.

Generalizzando, se a divide S in m sottoinsiemi, ognuno di dimensione N_i, l'**indice di impurità** dello split è calcolato come

$$gini_{split}(S, a) = \sum_{i=1}^{m} \frac{N_i}{N} gini(S_i)$$

L'attributo con il **minore** valore è selezionato come attributo di split!

Alberi di decisione: indice di Gini

(Algoritmo CART) Nel caso di attributi a valori numerici (interi) si può proseguire calcolando tutti i possibili split binari definiti dai valori stessi.

Esempio:

RID	Age	Car Type	Risk
0	23	family	high
1	17	sport	high
2	43	sport	high
3	68	family	low
4	32	truck	low
5	20	family	high

Attribute list for 'Age'

Age	RID	Risk
17	1	high
20	5	high
23	0	high
32	4	low
43	2	high
68	3	low

Attribute list for 'Car Type'

Car Type	RID	Risk
family	0	high
sport	1	high
sport	2	high
family	3	low
truck	4	low
family	5	high

Alberi di decisione: indice di Gini

Possibili valori di split per *Age*:
$Age \leq 17, Age \leq 20, Age \leq 23, Age \leq 32, Age \leq 43, Age \leq 68$

Iniziamo con 17

Tuple count	High	Low
Age<=17	1	0
Age>17	3	2

$$G(Age \leq 17) = 1 - (1^2 + 0^2) = 0$$
$$G(Age > 17) = 1 - ((3/5)^2 + (2/5)^2) = 1 - (13/25)^2 = 12/25$$
$$G_{SPLIT} = (1/6) \cdot 0 + (5/6) \cdot (12/25) = 2/5$$

Alberi di decisione: indice di Gini

Tuple count	High	Low
Age<=20	2	0
Age>20	2	2

$$G(Ag \leq 20) = 1 - (1^2 + 0^2) = 0$$
$$G(Age > 20) = 1 - ((1/2)^2 + (1/2)^2) = 1/2$$
$$G_{SPLIT} = (2/6) \cdot 0 + (4/6) \cdot (1/8) = 1/3$$

Tuple count	High	Low
Age<=23	3	0
Age>23	1	2

$$G(Age \leq 23) = 1 - (1^2 + 0^2) = 0$$
$$G(Age > 23) = 1 - ((1/3)^2 + (2/3)^2) = 1 - (1/9) - (4/9) = 4/9$$
$$G_{SPLIT} = (3/6) \cdot 0 + (3/6) \cdot (4/9) = 2/9$$

Alberi di decisione: indice di Gini

Tuple count	High	Low
Age<=32	3	1
Age>32	1	1

$$G(Age \leq 32) = 1 - ((3/4)^2 + (1/4)^2) = 1 - (10/16) = 6/16 = 3/8$$
$$G(Age > 32) = 1 - ((1/2)^2 + (1/2)^2) = 1/2$$
$$G = (4/6) \cdot (3/8) + (2/6) \cdot (1/2) = (1/8) + (1/6) = 14/48 = 7/24$$

Il piu' piccolo valore di G_{SPLIT} è per $Age \leq 23$.

Alberi di decisione: pruning

Problema

L'elevato numero di attributi in un training set o la particolare distribuzione dei valori degli attributi, può fare crescere notevolmente la dimensione dell'albero.

Questo fenomeno di overfitting. Rende più difficile classificazione (dovuto ad outliers) ed è associato ad un aumento non giustificato di errori.

Soluzione

Pruning (o **potatura**): rimuovere una parte di albero i rami che non contribuiscono ad una corretta classificazione, producendo qualcosa di meno complesso e così più comprensibile.

Alberi di decisione: pruning

Le strategie possibili per effettuare il pruning di un albero sono due:

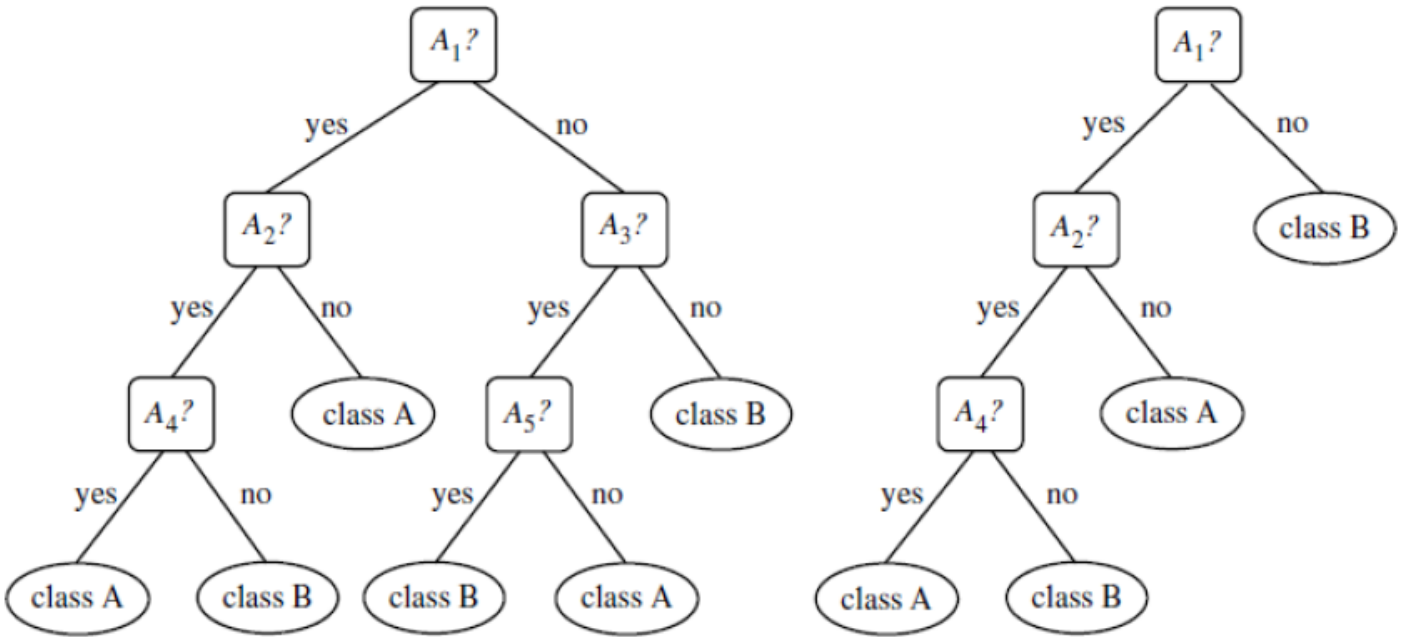
- **pre-pruning**: viene attuata in fase di costruzione dell'albero, prendendo la decisione di dividere ulteriormente o meno un determinato sottoinsieme;
  fisso una soglia t tale che i rami per cui si ottiene un gain inferiore di t vengono troncati o utilizzando metodi statistici.

- **post-pruning**: si effettua la recisione di alcuni rami a costruzione dell'albero già avvenuta. statistici.

Alberi di decisione: pruning

Il post-pruning sebbene più dispendioso, consente un'analisi più approfondita delle partizioni producendo un albero più realistico.

L'algoritmo C4.5 effettua il post-pruning.



Alberi di decisione: pruning

Problema

Potando un albero decisionale, causeremo quasi certamente la errata classificazione di alcuni casi. Di conseguenza le foglie dell'albero potato non conterranno casi appartenenti ad un singola classe.

Soluzione

Invece di una classe associata con una foglia, verrà specificata una classe di distribuzione, per ogni classe, e la **probabilità** che un caso appartenga a quella classe.

Questa modifica può lentamente migliorare la determinazione della classe più probabile per un caso non visto.

Alberi di decisione: pruning

Il C4.5 nelle operazioni di potatura oltre a poter **sostituire un sottoalbero con una singola foglia**, come la maggior parte degli altri sistemi, permette di **sostituire un intero sottoalbero con un suo ramo**.

Supponendo che si può predire il tasso di errore di un albero e di ogni suo sottoalbero, una scelta razionale sarebbe quella di effettuare la potatura di un sottoalbero in funzione della variazione di errore predetto che porterebbe all'intero albero.

Il C4.5 tipicamente usa il set di training ed adotta un metodo chiamato di potatura "pessimistica". Il suo scopo consiste nel ridurre l'errore totale dell'albero decisionale. Quando l'**errore medio** di un sottoalbero dopo aver annullato uno specifico test (nodo interno) è minore di quello che riserva il test stesso, il nodo corrispondente al test viene potato insieme a tutto il suo sottoalbero.

Alberi di decisione: minimal cost-complexity pruning

Il **minimal cost-complexity pruning** è un algoritmo di potatura parametrizzato tramite un parametro $\alpha \geq 0$ detto anche **parametro di complessità**.
Tale parametro è usate per definire una misura di **costo** di complessità, $R_\alpha(T)$ per un dato albero T:

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}|$$

dove $|\widetilde{T}|$ è il numero di nodi terminali in T e $R(T)$ è il tasso di errore dei nodi terminali di T.

L'algoritmo ha lo scopo di trovare il sottoalbero di T che minimizza $R_\alpha(T)$.

Alberi di decisione: minimal cost-complexity pruning

Il costo di un singolo **nodo** è dato da $R_\alpha(t) = R(t)$.

Il **ramo** T_t è un albero avente t come radice.

In generale, l'impurità di un nodo radice è maggiore della somma delle impurità dei nodi terminali, ovvero $R(T_t) < R(t)$.
Tuttavia, il costo di un nodo t e del relativo albero T_t può essere uguale, in base al valore di α utilizzato.

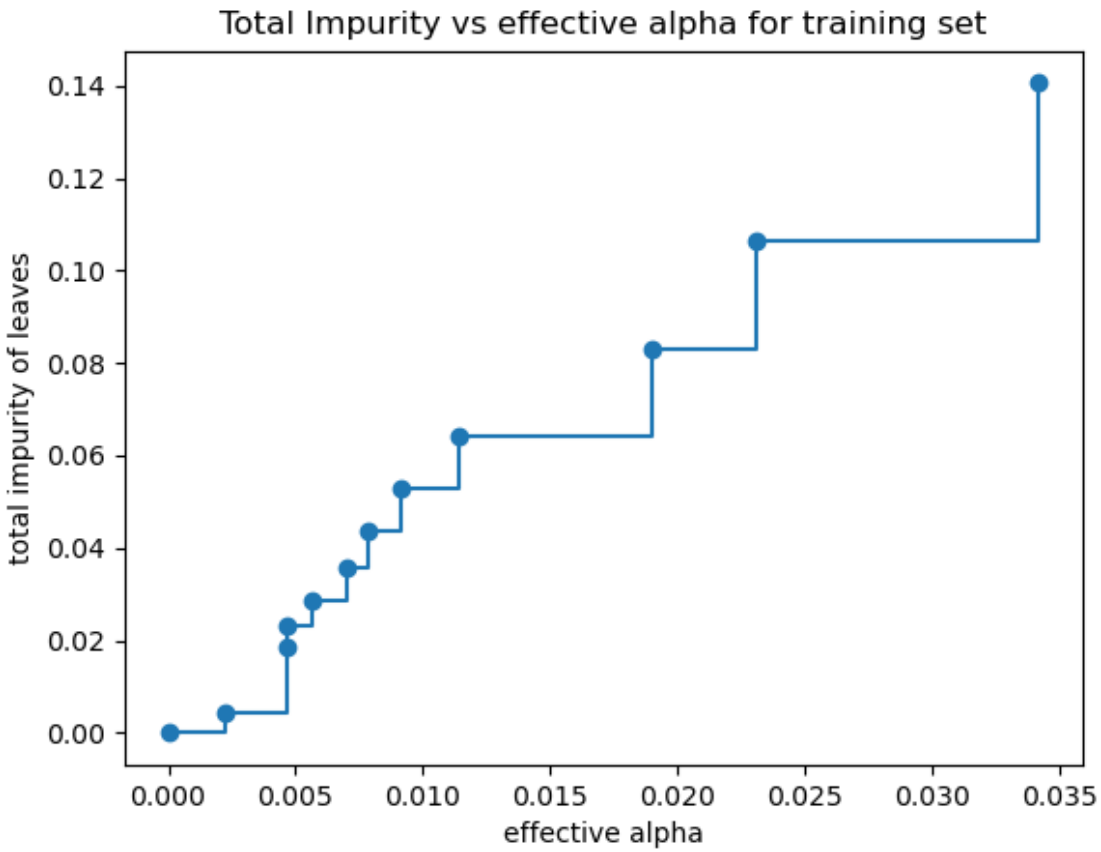
Definiamo α **effettivo** per un nodo t l'α per cui $R_\alpha(T_t) = R_\alpha(t)$, oppure $\alpha_{eff} = \frac{R(t) - R(T_t)}{|T| - 1}$.

Il nodo non terminale con il più piccolo valore di α_{eff} è il nodo che dovrà essere potato.

L'algoritmo termina quando l'α_{eff} minimale del sottoalbero potato è maggiore di un valore soglia.

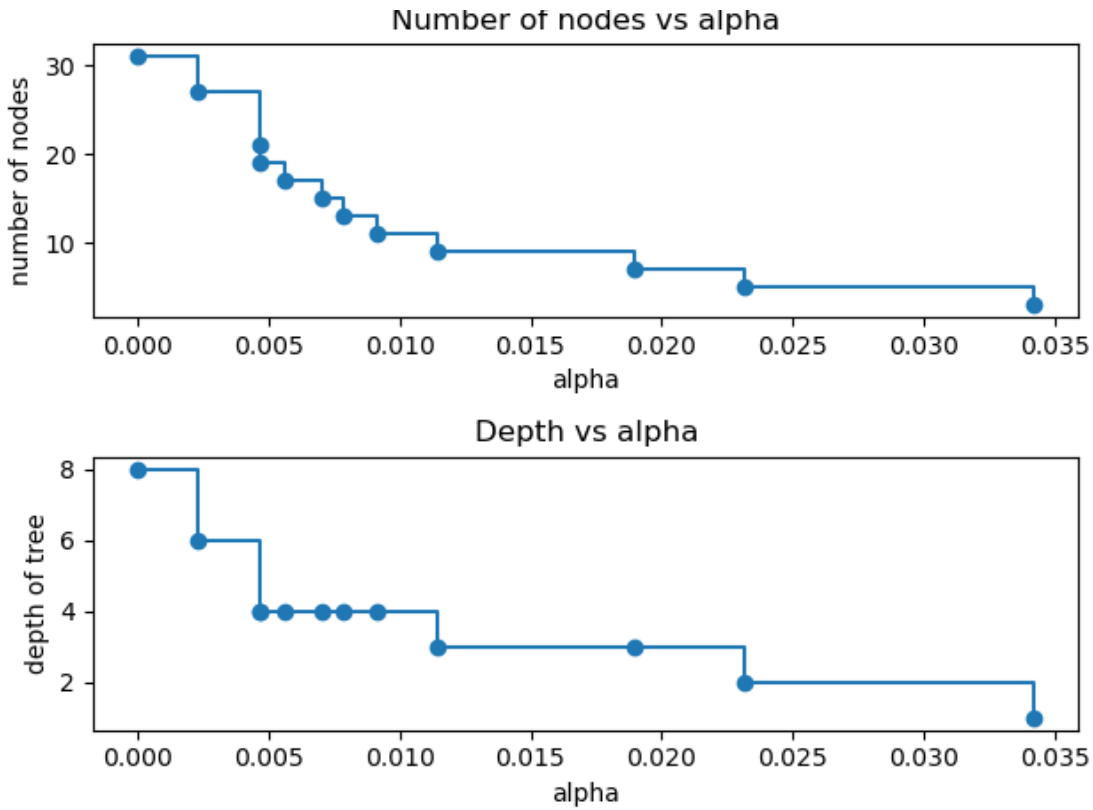
Alberi di decisione: minimal cost-complexity pruning

All'aumentare di α, viene potata una parte maggiore dell'albero, il che aumenta l'impurità totale delle sue foglie.



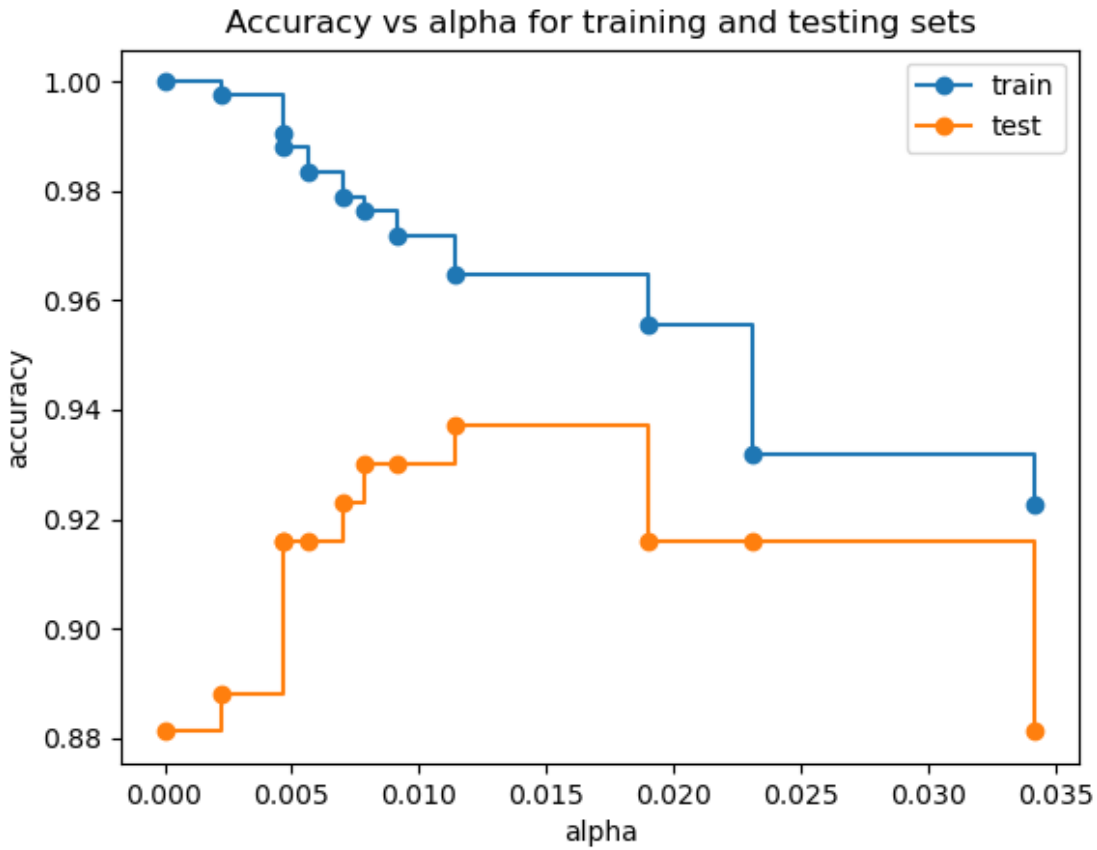
Alberi di decisione: minimal cost-complexity pruning

Il numero di nodi e la profondità dell'albero diminuiscono all'aumentare di α.



Alberi di decisione: minimal cost-complexity pruning

Quando il valore di soglia è settato a 0, l'albero va in overfitting, portando a una precisione di allenamento del 100% e a una precisione di test dell'88%.
All'aumentare di α, viene tagliata una parte maggiore dell'albero, creando così un albero decisionale che generalizza meglio.



Metodi consenso

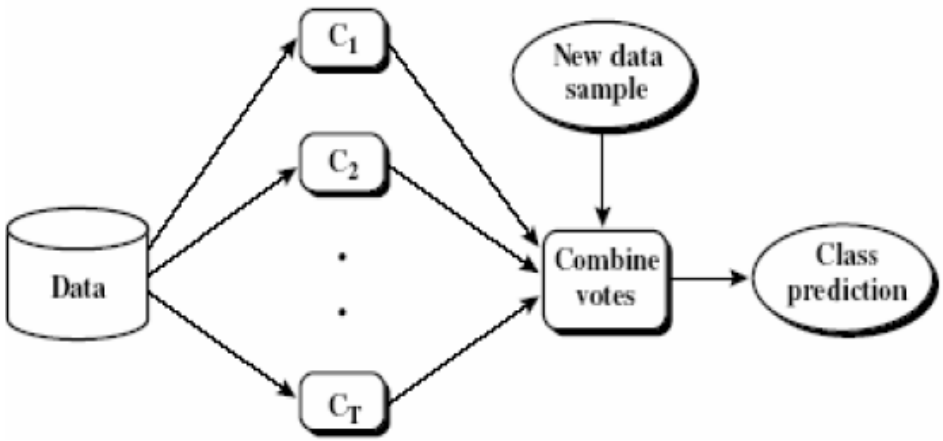
Classificatori basati su alberi di decisione hanno il vantaggio di essere semplici, di **facile interpretazione** ed efficienti a livello computazionale.

Tuttavia, non sono in genere competitivi in termini di qualità delle predizioni con i migliori metodi di apprendimento supervisionati. Inoltre, se cambiamo di poco i dati di training, l'albero appreso potrebbe cambiare di molto. Cioè manca stabilità. I risultati hanno un'elevata varianza, in genere.

Per ovviare a questi svantaggi sono state proposte diverse strategie, tra cui **bagging**, **random forests**, e **boosting**. Questi metodi costruiscono molteplici alberi che poi vengono combinati in una singola predizione **consenso**.

Questo può portare notevoli miglioramenti in termini di **accuratezza** delle predizioni, a svantaggio di una **perdita parziale di interpretabilità** del modello.

Metodi consenso



In linea generale, si vuole impiegare una combinazione di modelli M^*, ottenuta come combinazione di k modelli addestrati separatamente, per incrementare l'accuratezza.

I metodi più popolari sono:

- **bagging**: si **media** la predizione su una collezione di classificatori
- **bosting**: si impiega una collezione di classificatori tramite un **voto pesato**
- **ensemble**: si combina un insieme **eterogeneo** di classificatori

Metodi consenso: bagging

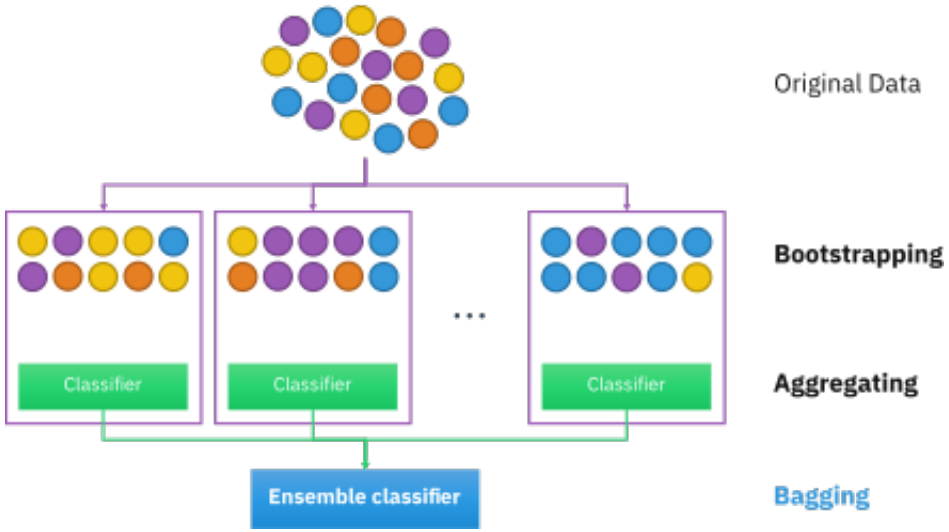
Il **bootstrap aggregation** (o **bagging**) è un tecnica generale utilizzata prevalentemente per ridurre la varianza di un metodo di apprendimento statistico o per stimare l'accuratezza di uno stimatore.

Particolarmente utile nel caso degli alberi di decisione.

Idealmente, se si avessero a disposizione più insiemi di osservazioni, si potrebbe apprendere un albero di decisione per ogni insieme, quindi combinare i risultati (media o voto di maggioranza).

Tuttavia di solito si dispone di un unico insieme di osservazioni.

Metodi consenso: bagging



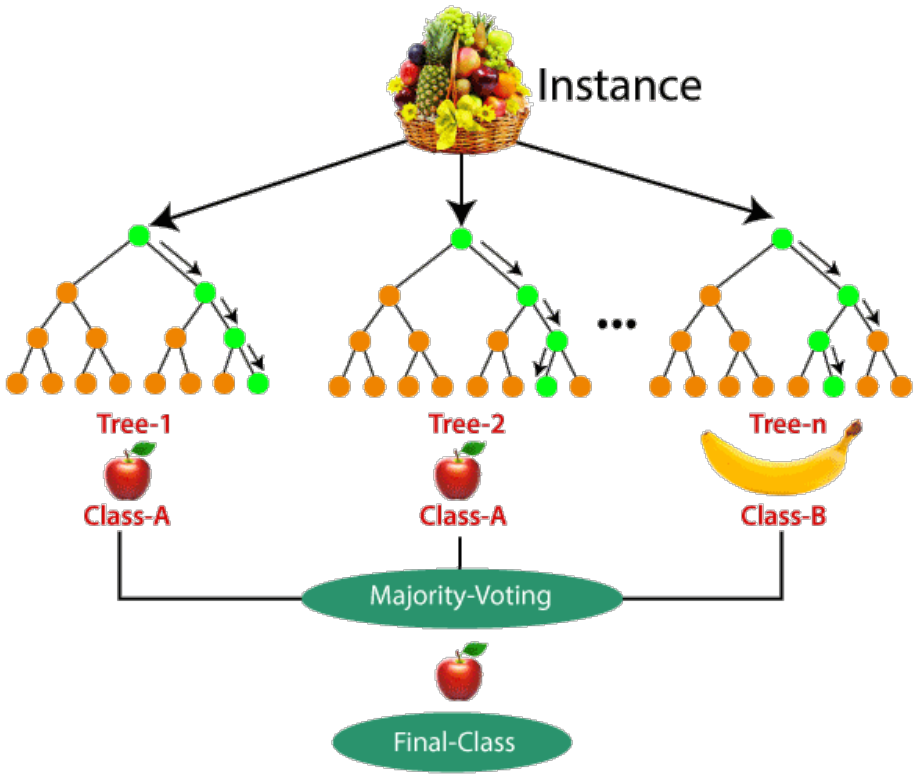
L'idea del bootstrap è di **estrarre casualmente** B diversi sottoinsiemi di osservazioni dall'unico training set, per simulare la variabilità dei dati.

Apprendere una classificatore f_i per ciascun sottoinsieme i seperatamente.

Quindi, aggregare (ensemble) le predizioni sulle nuove istanze x di test:

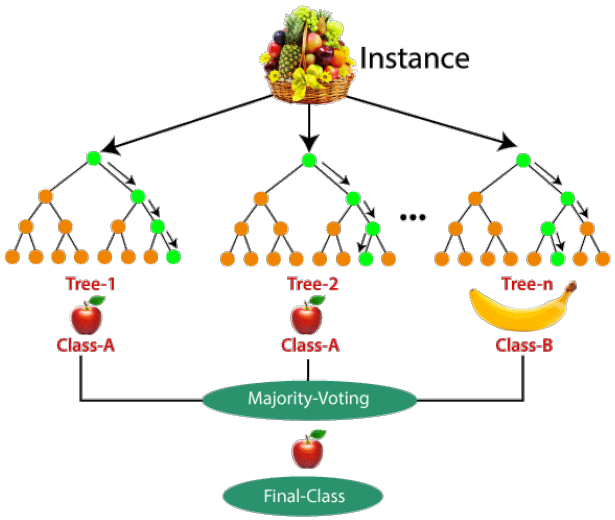
$$f_{bag}(x) = \frac{1}{B}\sum_{i=1}^{B} f_i(x)$$

Metodi consenso: random forest



È un concenso costituito da un bagging di **alberi di decisione non potati** (non-pruned) e complessi, con una **scelta casuale** di un sottoinsieme delle feature (predittori) ad ogni split.

Metodi consenso: random forest



Metodo non lineare (come gli alberi di decisione) e robusto, bassa varianza e **stabilità** delle predizioni rispetto al variare dei dati di input.

Migliora le **prestazioni** di un singolo albero di decisione appreso su tutti i dati.

Perde la facile **intepretabilità** degli alberi e scala meno bene.

Metodi consenso: random forest

Addestramento della foresta

Fissato un numero a piacere n_{tree} e il numero di predittori (feature, variabili) m_{try} da scegliere casualmente tra le disponibili, per n_{tree} volte ripete la procedura che segue :

1. Scegli un sottoinsieme di bootstrap S dai dati di training
2. Apprendi un albero di decisione su S accurato ($minsplit = 1$) in cui ad ogni split solo m_{try} predittori sono scelti (casualmente) come candidati per lo split tra tutti
3. Non eseguire pruning dell'albero
4. Salva l'albero ottenuto quelli disponibili

Predizione nuove istanze

Aggrega le predizioni sui nuovi dati come fatto nel bagging, mediante voto di maggioranza delle predizioni fatte dagli n_{tree} alberi (classificazione) o mediante media delle stesse (regressione)

Metodi consenso: random forest

Di solito m_{try} è posto a $m/3$ per problemi di regressione e $\sqrt{m}$ per problemi di classificazione, dove m è il numero totale di predittori.

Metodi consenso: random forest

Scalabilità: RF aumenta in maniera contenuta la complessità temporale rispetto agli alberi, dato che ogni singolo albero è appreso solo su un sottoinsieme dei dati e a ogni split solo un sottoinsieme dei predittori viene preso in considerazione. Può gestire dati anche di grande taglia prima di offrire rallentamenti evidenti.

RF non presenta problemi di **overfitting** all'aumentare del numero di alberi. Questo è dovuto al fatto che solo una piccola porzione dei predittori è usata a ogni split, oltre al fatto che RF predice mediante aggregazione.

RF è più **stabile** alle variazioni dei dati di input, grazie al bagging.

Gli alberi appresi sono **indipendenti** (a differenza del boosting). Questo rende il metodo facilmente **parallelizzabile** con hardware appropriato (e.g. più core o processori)

Metodi consenso: random forest

Una costruzione più raffinata

L'idea fondamentale alla base del bagging è quella di calcolare la media di molti modelli contenenti errore ma approssimativamente non distorti, in modo da ridurre la varianza.

Gli alberi sono candidati ideali per il bagging, poiché possono catturare strutture di interazione complesse presenti nei dati e, se fatti crescere con sufficiente profondità, hanno una distorsione relativamente bassa.

Poiché gli alberi sono notoriamente soggetti ad errore,possono beneficiare in maniera importante dal calcolo della loro media.

Una costruzione più raffinata

Inoltre, ogni albero generato nel bagging proviene da una distribuzione identica (i.d.: identicamente distribuito), il valore atteso di una media di B tali alberi sarà lo stesso valore atteso di uno qualunque di loro.

Ciò significa che la distorsione di alberi "bagged" è la stessa dei singoli alberi, e la sola opzione di miglioramento sarà tramite la riduzione della varianza.

Una media di variabili B casuali i.i.d., ciascuna con varianza σ^2, ha varianza $\frac{1}{B}\sigma^2$.

Una costruzione più raffinata

Se le variabili sono semplicemente i.d. (identicamente distribuite, ma non necessariamente independenti) con correlazione a coppie ρ positiva, la varianza della media è

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Mano a mano che B cresce, il secondo termine della somma si riduce, mentre il primo rimane,e quindi la valore della correlazione di coppie di alberi bagged limita i benefici del calcolo della media. L'idea nelle foreste casuali è quella di migliorare la riduzione della varianza del bagging riducendo la correlazione tra gli alberi, senza accrescere troppo la varianza. Questo è raggiunto nel processo di crescita degli alberi attraverso la selezione casuale delle variabili di input.

Metodi consenso: random forest

Una costruzione più raffinata

Nello specifico, quando si fa crescere un albero su un dataset "bootstrapped":

prima di ogni split, seleziona a caso $m \leq p$ delle variabili di input come candidate per lo split.

Valori tipici di m possono andare da p a valori molto piccoli, quale 1, con valore usualmente preferibile pari a $p/3$.

Una costruzione più raffinata

Dopo aver fatto crescere un insieme B di alberi $\{T_b(x;\theta_b)\}_1^B$ in questo modo, il predittore basato su foresta casuale sarà

$$\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x;\theta_b)$$

dove θ_b caratterizza il b-esimo albero della RF in termini di variabili di split, punto di split in ogni nodo, valori del nodo terminale; $T_b(x;\theta_b)$ è la previsione del b-esimo albero della RF.

Intuitivamente, la riduzione di m produce una riduzione della correlazione tra coppie di alberi nell'insieme, e quindi ridurrà la varianza media.

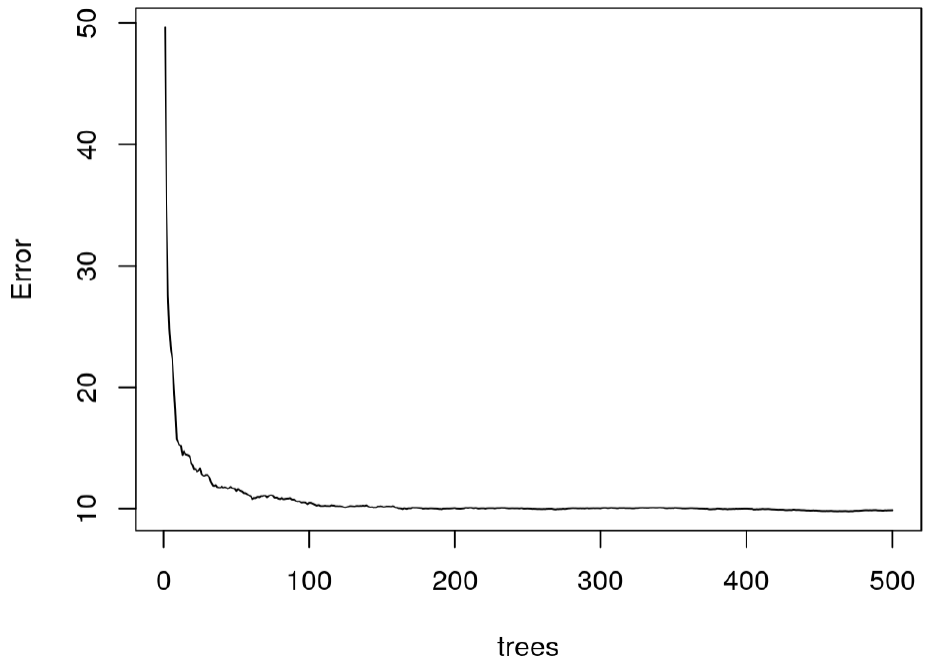
Una costruzione più raffinata

L'algoritmo di costruzione può essere riassunto come:

1. Per $b = 1 \dots B$
   a. estrai un campione boostrap Z^* di dimensione N dai dati di training
   b. fai crescere un albero T_b delle foresta sui dati bootstrapped, ripetendo ricorsivamente i seguenti passaggi per ciascun nodo terminale dell'albero, fino al raggiungimento delle dimensione minima di nodi n_{min}
      i. seleziona m variabili a caso tra le p variabili
      ii. selezione la migliore variabile e punto di split tra le m
      iii. dividi (split) il nodo in due nodi figli
2. ritorna l'insieme degli alberi $\{T_b\}_1^B$

Metodi consenso: random forest

Generalmente un numero di alberi pari a 100 (o superiore) è un buon compromesso tra tempo di addestramento e potere predittivo.



Tuttavia, questo dipende da molti fattori (es numero di attributi, livelli di rumorosità nei dati, e dipendenza tra le feature).

Metodi consenso: random forest

Le random forest (RF) forniscono anche una **stima dell'errore** di test chiamato Out Of Bag (OOB) error, che è la media degli errori di predizione su ogni istanza di training x_i mediante l'aggregazione delle predizioni dei soli alberi che non contenevano x_i tra le istanze di training.

Per calcolare l'**importanza** di ogni variabile/attributo/feature si possono apllicare due strategie:

- **Mean decrease in accuracy**. Viene calcolata l'accuratezza della RF sui dati out-of-bag, quindi, per ogni albero per ogni variabile i, si permuta la colonna *i*-ma dei dati OOB, in modo da perdere la correlazione (eventuale) con la variabile di outcome. Si predicono le istanze così ottenute e si ricalcola l'accuratezza, valutando la differenza con il precedente valore. Quindi si fa la media sugli alberi
- **Mean decrease in the Gini index**. Misura analoga a quella usata negli alberi di decisione: per ogni albero e per ogni variabile i, si valuta ogni nodo in cui lo split coinvolge tale variabile, calcolando la differenza dell'indice di Gini (impurità) prima e dopo la suddivisione, pesandola per il numero di istanze nel nodo. Quindi si fa la media sugli alberi

Metodi consenso: random forest

Quando evitare di utilizzare i random forest:

- **Estrapolazione**: La regressione casuale della foresta non è l'ideale nell'estrapolazione dei dati. A differenza della regressione lineare, che utilizza le osservazioni esistenti per stimare i valori oltre l'intervallo di osservazione. Questo spiega perché la maggior parte delle applicazioni della foresta casuale si riferiscono alla classificazione.
- **scaristà di dati iniziali**: La foresta casuale non produce buoni risultati quando i dati sono molto scarsi. In questo caso, il sottoinsieme di funzionalità e il campione bootstrap produrranno uno spazio invariante. Ciò porterà a divisioni improduttive, che influenzeranno il risultato.

Metodi consenso: random forest

Vantaggi:

- Può eseguire sia attività di regressione che di classificazione.
- Una foresta casuale produce buone predizioni che possono essere comprese facilmente.
- Può gestire in modo efficiente set di dati di grandi dimensioni.
- L'algoritmo della foresta casuale fornisce un livello di precisione più elevato nella previsione dei risultati rispetto all'algoritmo dell'albero decisionale

Svantaggi:

- Quando si usa una foresta casuale, sono necessarie più risorse per il calcolo.
- Consuma più tempo rispetto a un algoritmo di albero decisionale, all'aumentare della dimensione della foresta.

Boosting: analogia

Consultare diversi medici, sulla base di una combinazione di diagnosi pesate: il peso viene assegnato in base all'accuratezza della diagnosi precedente.

Come si procede?

1. vengono assegnati dei pesi ad ogni tupla di training
2. viene addestrata una serie di k classificatori
   - ogni volta che l'i-esimo classificatore M_i viene addestrato, i pesi sono aggiornati in modo da consentire al successivo classificatore, M_{i+1}, di porre più attenzione alle tuple che sono state classificate erroneamente dal classificatore M_i
3. il classificatore finale M^* combina i voti di ogni classificatore intermedio, tale che il peso assegnato ad ogni voto è proporzionale alla accuratezza dello specifico classificatore

Metodi consenso: Adaboost

Sia dato un insieme D di d tuple etichettate , $(X_1, y_1), \ldots, (X_{,} y_d)$, dove X_i è l'i-esima tupla e y_i è l'etichetta ad essa assegnata.

Settiamo inizialmente tutti i pesi allo stesso valore di $1/d$.

Generiamo i k classificatori in k passi. Ad ogni passo i:

1. le tuple in D vengono campionate (con "replacement" ovvero rimettendone una copia in D dopo l'estrazione) per formare un training set D_i della stessa dimensione di D
2. la probabilità di ogni tupla di essere selezionato è proporzionale al suo peso
3. viene addestrato un classificatore M_i da D_i
4. viene calcolato il tasso di errore di M_i utilizzando D_i come test set
5. se una tupla viene classificata erroneamente, il suo peso viene incrementato, altrimenti viene decrementato

Metodi consenso: Adaboost

Sia $err(X_j)$ l'errore nel classificare la tupla X_j ($\{0, 1\}$ se si parla di classificatori e non di regressori).

L'errore del classificatore M_i è dato da

$$error(M_i) = \sum_{j=1}^{d} w_j \times err(X_j)$$

dove w_j è il peso corrente per la tupla X_j.

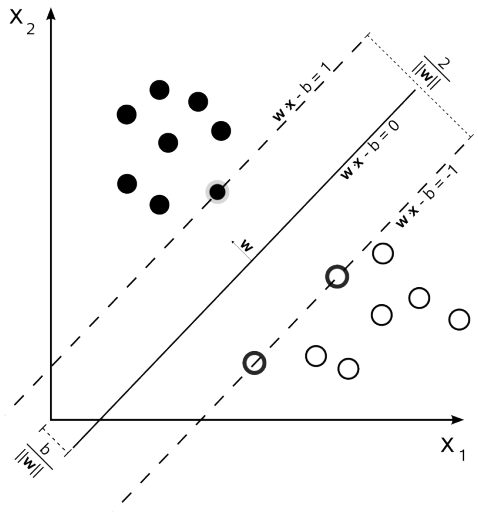
Il peso del classificatore M_i nella votazione consenso sarà:

$$log \frac{1 - error(M_i)}{error(M_i)}$$

SVM: Support Vector Machines

La teoria che governa i meccanismi di funzionamento di SVM è stata introdotta da Vapnik a partire dal 1965 statistical learning theory e perfezionata più recentemente 1995 dallo stesso Vapnik e altri SVM è uno degli strumenti più utilizzati per la classificazione di pattern.

Invece di stimare le densità di probabilità delle classi per un classificatore Bayesiano, Vapnik suggerisce di risolvere direttamente il problema di interesse (che considera più semplice), ovvero determinare le superfici decisionali tra le classi (**classification boundaries**).



SVM: Support Vector Machines

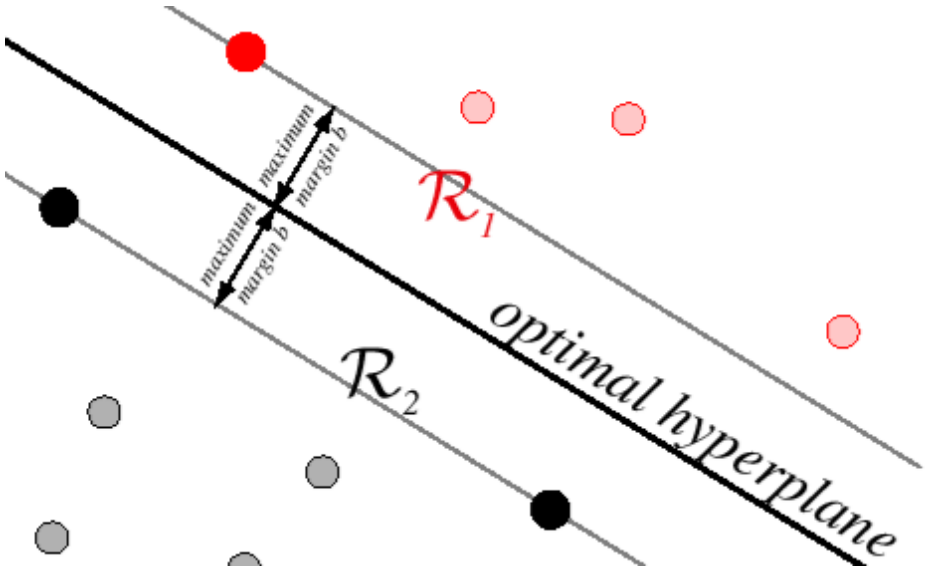
SVM nasce come classificatore binario, estendibile a più classi:

- SVM **lineare**: la superficie di separazione è un **iperpiano** tra i pattern del training set **linearmente separabili** (es. esiste per ipotesi almeno un iperpiano in grado di separarli)
- SVM **lineare** e pattern **non linearmente separabili**. Ci saranno inevitabilmente errori di classificazione nel training set non esistendo alcun iperpiano in grado di separare i pattern
- SVM **non lineare** (superficie di separazione complessa senza ipotesi sulla separabilità dei pattern)
- Estensione **multiclasse**

SVM: Support Vector Machines

Date due classi linearmente separabili, tra tutti i possibili iperpiani di separazione, SVM determina quello in grado di **separare le classi** con il maggior **margine** possibile.

Il margine è la distanza minima dei punti delle due classi nel training set dall'iperpiano individuato,



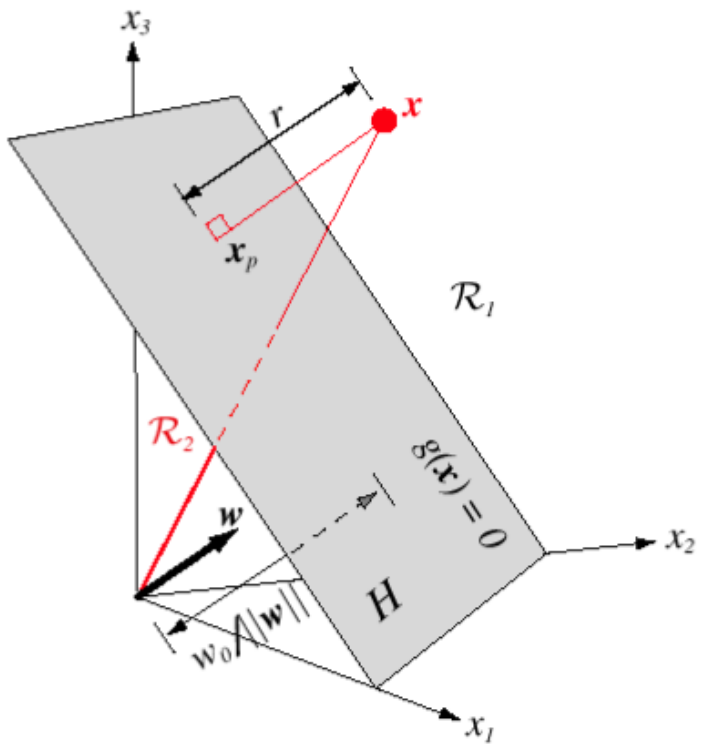
SVM: Support Vector Machines

La massimizzazione del margine è legata alla **generalizzazione**. Se i record del training set sono classificati con ampio margine si può sperare che anche i record del test set vicini al confine tra le classi siano gestiti correttamente.

Come vedremo il problema può essere impostato come una ottimizzazione convessa (convex optimization). Come tale ammette **un solo minimo globale** (vantaggio non si rischia di essere intrappolati in minimi locali).

SVM lineari

Date due **classi linearmente separabili** e un training set TS contenente n campioni di coordinate $(x_1, y_1), \ldots, (x_n, y_n)$ dove $x_i \in \mathbb{R}^d$ sono i **record multidimensionali** e $y_j \in \{+1, -1\}$ le **etichette** delle due classi, esistono diversi **iperpiani** in grado di eseguire la separazione voluta.

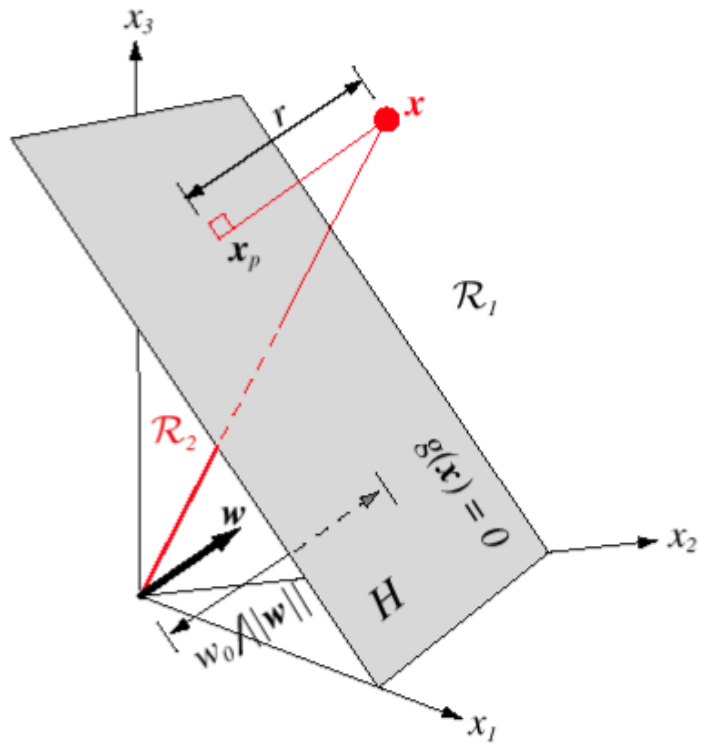


SVM lineari

Un generico **iperpiano** è definito dai parametri (w, b) (per semplicità omettiamo il trasposto del prodotto scalare):

$$D(x) = w \cdot x + b$$

dove w è il vettore normale all'iperpiano, $b/||w||$ è la distanza dall'origine, $D(x) = 0$ il luogo dei vettori sul piano, abbiamo che, se $x = x_p + r\frac{w}{||w||}$ e $D(x_p) = 0$, allora $D(x) = w \cdot x + b = r \cdot ||w||$, pertanto la distanza di un vettore x dall'iperpiano vale $r = \frac{D(x)}{||w||}$.



SVM lineari

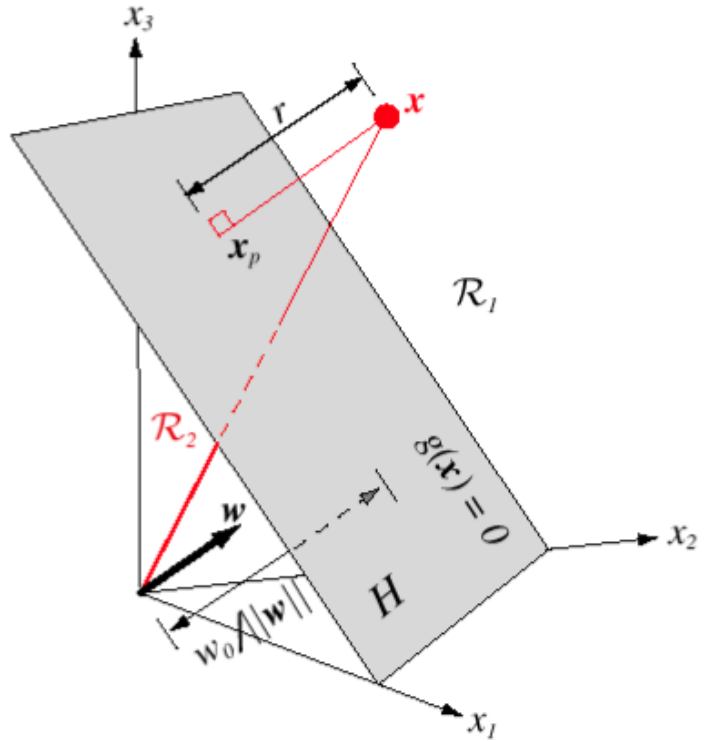
Gli **iperpiani** (w, b) che separano i dati del TS, con distanza minima $1/||w||$ su ogni lato, soddisfano, per $i = 1, \ldots, n$, le equazioni:

$w \cdot x_i + b \geq +1$ se $y_i = +1$

$w \cdot x_i + b \leq -1$ se $y_i = -1$

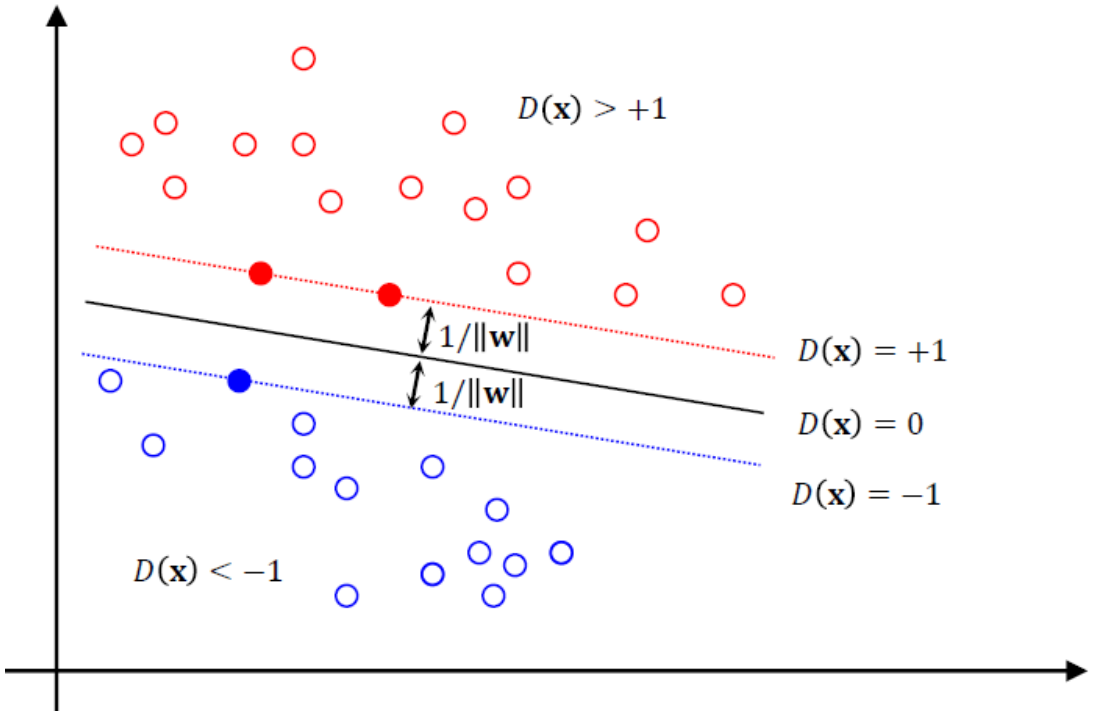
o in modo più compatto

$y_i[w \cdot x_i + b] \geq 1$ per $i = 1, \ldots, n$.



SVM lineari

La **minima distanza** tra l'iperpiano di separazione e un dato del training set è detta **margine** (τ)



La **distanza** dei punti che giacciono sull'iperpiano $D(x) = +1$ dall'iperpiano di separazione ($D(x) = 0$) è $1/||w||$; e lo stesso vale per i punti sull'iperpiano $D(x) = -1$. Pertanto il **margine** è $\tau = 2/||w||$.

SVM lineari

L'iperpiano **ottimo** secondo SVM è quello soddisfa i **vincoli di separazione** dei dati e massimizza il margine τ (o alternativamente minimizza il suo inverso):

minimizza: $\frac{||w||^2}{2}$
vincoli: $y_i[w \cdot x_i + b] - 1 \geq 0$ per $i = 1, \ldots, n$.

I dati del training set che giacciono sul margine (cerchi pieni in figura) sono detti **support vector**. Tali dati, che costituiscono i casi più complessi, **definiscono completamente la soluzione** del problema, che può essere espressa come funzione di solo tali dati **indipendentemente dalla dimensionalità** dello spazio d e **dal numero** n di elementi in TS.

E' un problema di ottimizzazione risolvibile tramite formulazione Lagranciana e successivamente come formulazione duale.

SVM lineari

Vantaggi:

- Definizione della soluzione sulla base di un numero ridotto di support vector (solitamente pochi)
- Il numero di support vector n_{sv} indica la complessità del problema e può essere dimostrato che l'errore medio (sui possibili training set) è limitato da n_{sv}/n
- SVM scala molto bene rispetto alla dimensionalità d dello spazio delle feature (grazie ai prodotti scalari). La complessità computazionale nel training è quadratica rispetto al numero n di dati nel TS. In pratica il problema può essere risolto per $d = 10^7$ e per n fino a 10^4.

SVM per dati non separabili

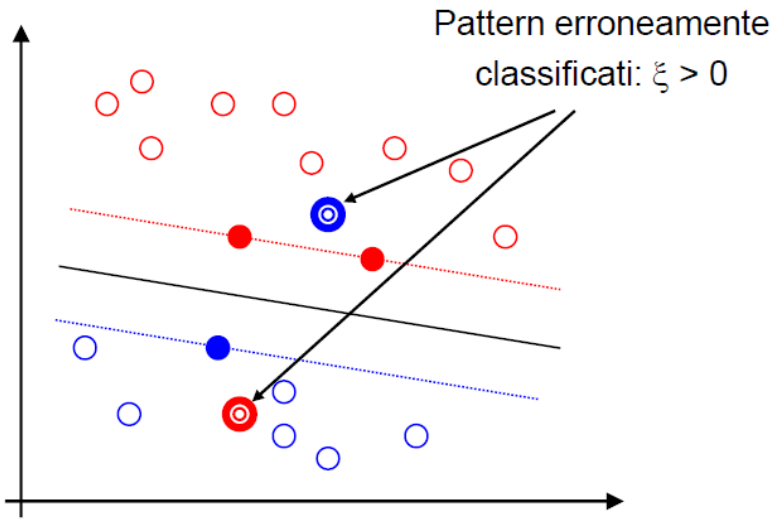
In questo caso **non tutti** i dati possono essere separati da un iperpiano, ed è necessario **rilassare i vincoli di separazione** per far sì che alcuni pattern (il minor numero possibile) possano valicare il confine della classe.

SVM per dati non separabili

A tal fine si introducono n variabili di **slack** positive ξ_i, per $i = 1, \ldots, n$, e si modificano i vincoli di separazione:
$y_i[w \cdot x_i + b] \geq 1 - \xi_i$ per $i = 1, \ldots, n$

Per ogni dato x_i del TS, la variabile ξ_i codifica la deviazione dal margine.

Per i dati separabili del TS le corrispondenti variabili di slack assumono valore 0.



SVM per dati non separabili

L'iperpiano ottimo deve in questo caso ancora **massimizzare il margine** ma allo stesso tempo **minimizzare il numero di elementi non correttamente classificati**. La funzione obiettivo, e di conseguenza il problema di ottimizzazione vengono così modificati:

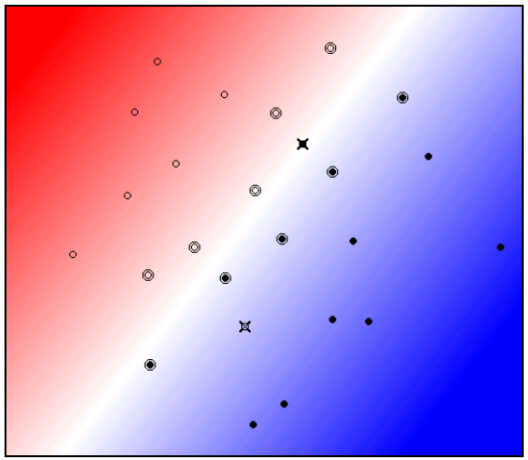
minimizza: $\frac{||w||^2}{2} + C \sum_{i=1,\ldots,n} \xi_i$

vincoli: $y_i[w \cdot x_i + b] \geq 1 - \xi_i$ per $i = 1, \ldots, n$.

Il coefficiente C del problema di ottimizzazione precedente, indica l'importanza relativa degli errori di classificazione rispetto all'ampiezza del margine.

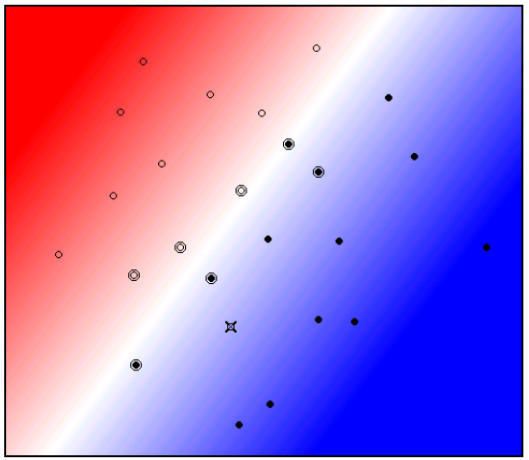
$C = 10$

2 errori, margine maggiore



$C = 200$

1 solo errore, margine minore



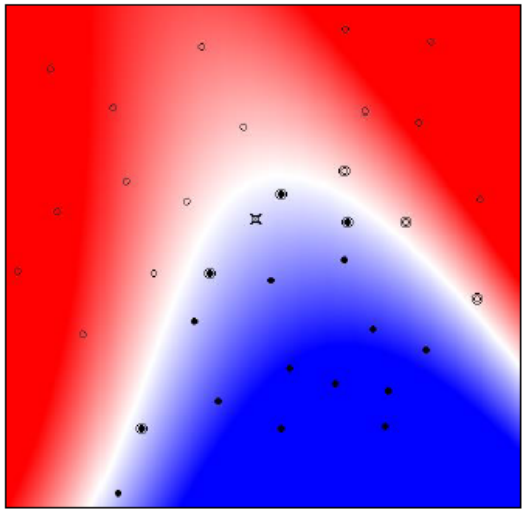
SVM non lineari

Viene definito un **mapping Φ non lineare** dei dati dallo spazio di partenza $\mathbb{R}^d$ verso uno spazio $\mathbb{R}^m$ **a più alta dimensionalità** $(m > d)$:
$\Phi : \mathbb{R}^n \mapsto \mathbb{R}^m$, $\Phi(x) = [g_1(x), g_2(x), \ldots, g_m(x)]$.

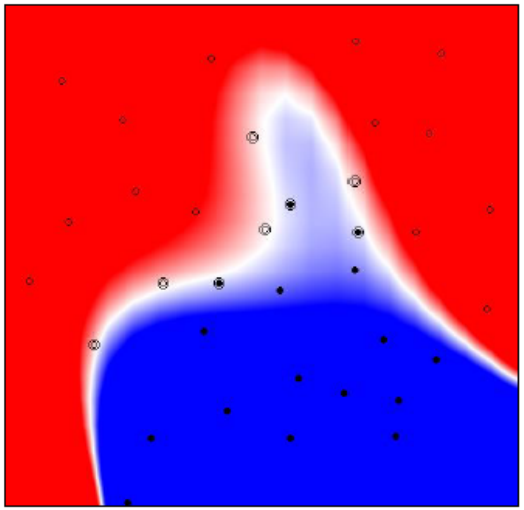
Nello spazio $\mathbb{R}^m$, dove **maggiori sono i gradi di libertà**, i dati $\Phi(x_1), \Phi(x_2), \ldots, \Phi(x_n)$ possono essere più facilmente separati da un iperpiano utilizzando la teoria nota.
Ciò equivale a separare i dati $x_1, x_2, \ldots, x_n$ in $\mathbb{R}^d$ con superfici arbitrariamente complesse.

SVM non lineari

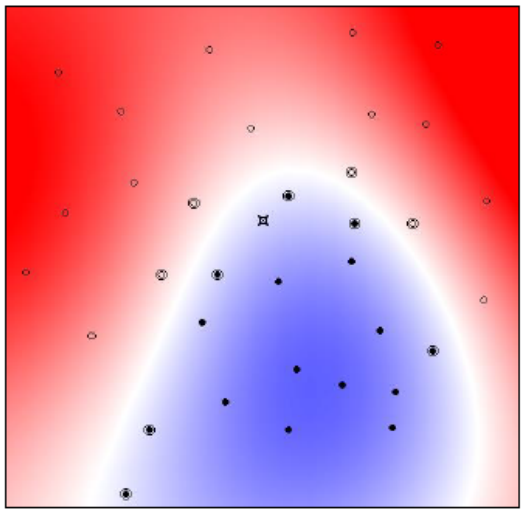
Polinomio q = 2



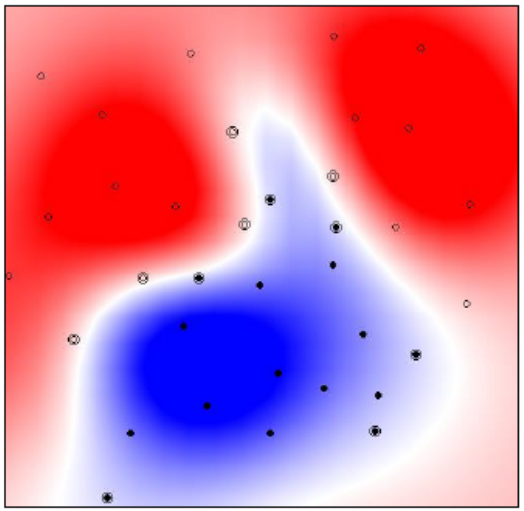
Polinomio q = 10



RBF σ = 1



RBF σ = 0.2



SVM: estensione multiclasse

SVM è in grado di determinare la superficie di separazione tra 2 classi di pattern come gestire allora i problemi con più di 2 classi?

Si tratta di un problema ancora aperto anche se esistono diverse soluzioni le più utilizzate sono:

One-Against-One trovando gli iperpiani di separazione a coppie di classi ed utilizzando poi un metodo ensemble.

SVM: estensione multiclasse

One-Against-All:

- date s classi $w_1, w_2, \ldots, w_s$
- per ogni classe w_k si determina con SVM la superficie di separazione tra i dati di w_k (etichetta $+1$) da una parte, e i dati di tutte le rimanenti classi $w_{h \neq k}$ (etichettati -1) dall'altra, ottenendo la funzione $D_k(x)$ che indica quanto x è distante dalla superficie decisionale in direzione di w_k. Maggiore è $D_k(x)$ più confidenti siamo dell'appartenenza di x a w_k.
- al termine del training, si assegna il dato X alla classe k^* per cui è massima la distanza dalla superficie decisionale.

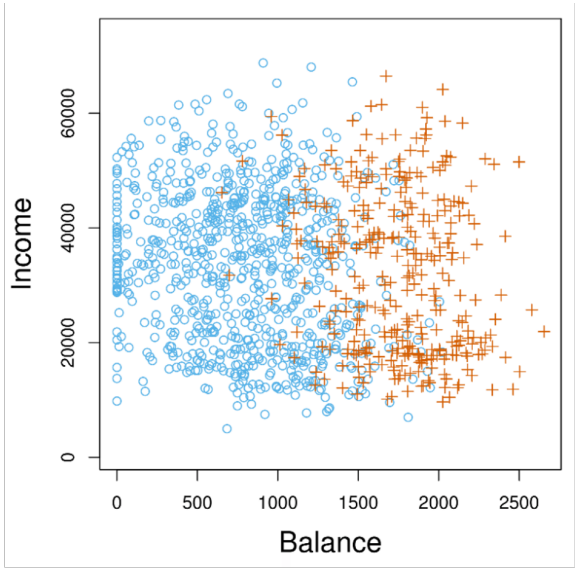
Nota bene: non è necessario eseguire s training SVM diversi.

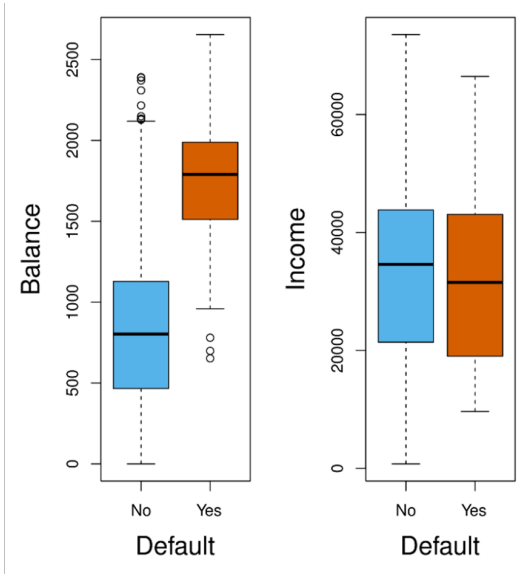
Analisi discriminante

Una società creditizia vuole stimare la **probabilità di default** (incapacità di fare fronte ai pagamenti) in base ad alcune caratteristiche del debitore:

- Y default: variabile binaria (default o no)
- X_1 student: variabile binaria (studente o no)
- X_2 balance: l'importo medio di debito residuo sulla carta di credito dopo i versamenti mensili
- X_3 income: il reddito dell'unità

default	student	balance	income
No	No	729.5265	44361.625
No	Yes	817.1804	12106.135
No	No	1073.5492	31767.139
No	No	529.2506	35704.494
No	No	785.6559	38463.496
No	Yes	919.5885	7491.559





Analisi discriminante

Consideriamo *Default* come **classe** e supponiamo di usare un solo **predittore**, *student*, per determinare

$$P(default = Yes | student = Yes)$$

Incrociando i dati otteniamo:

	student		
default	No	Yes	Sum
No	6850	2817	9667
Yes	206	127	333
Sum	7056	2944	10000

Si noti che in questo caso, possiamo stimare direttamente

$$\hat{P} = (default = Yes | student = Yes) = \frac{127}{2944} = 0.043$$

Tuttavia nella pratica, volendo prevedere la probabilità di default sulla base di variabili qualitative e quantitative, risulta più semplice passare attraverso l'utilizzo dl teorema di Bayes.

Calcolo delle probabilitá

La **probabilitá condizionata** di un evento A dato un evento B, $P(A|B)$, esprima la probabilitá implicazionale che A occorra dato che B sia giá occorso.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Se A e B sono **indipendenti** allora $P(A|B) = P(A)$, e lo indichiamo come $A||B$.

Se A e B sono **disgiunti** allora $P(A \wedge B) = 0$.

La proposizione di **negazione** $\neg A$ deve quindi essere considerata in termini di insieme complementare, ovvero se $A = (X \in S)$ allora $\neg A = (X \in range(X) - S)$.

Calcolo delle probabilitá

Regole:

1. $P(A) \geq 0$
2. $P(A \vee \neg A) = 1$
3. $P(A \wedge \neg A) = 0$
4. $P(\neg A) = 1 - P(A)$
5. $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$ (principio di Inclusione-Esclusione)
6. $P(A|B) = \frac{P(A \wedge B)}{P(B)}$
7. $A||B \Leftrightarrow P(A \wedge B) = P(A) \cdot P(B)$

Calcolo delle probabilitá

Un pilota ha la probabilitá di morire in volo del 2%. Quale probabilitá ha il pilota di morire in 50 voli?

Intuitivamente viene da sommare ovvero $2\% \times 50 = 100\%$.
In realtá i vari voli non sono disgiunti perché dobbiamo considerare che dopo ogni volo il pilota é sopravvissuto. Si ha quindi

$$p + (1-p)p + (1-p)^2 p + \ldots (1-p)^{49} p$$

dove $(1-p)$ riflette il fatto che é sopravvissuto al primo volo,
$(1-((1-p)p)) = 1-(p-p^2) = a - p + p^2 = (1-p)^2$ al secondo, etc.
Di conseguenza ha circa uno $0.98^{50} = 0.36$ di sopravvivenza ed uno 0.64 di morte.

Teorema di Bayes

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

Dim.
Dalla definizione di probabilitá condizionata si ha che

$$P(A \wedge B) = P(A|B)P(B)$$

analogamente

$$P(A \wedge B) = P(B \wedge A) = P(B|A)P(A)$$

da cui

$$P(A|B)P(B) = P(B|A)P(A)$$

da cui la tesi $\square$

Approccio Bayesiano

Indichiamo con $\pi = P(default = Yes)$ e corrispondentemente $(1 - \pi) = P(default = No)$. Inoltre,

$$P(S|D) = P(student = Yes|default = Yes)$$

e

$$P(S|\bar{D}) = P(student = Yes|default = No)$$

Utilizzando il teorema di Bayes otteniamo

$$P(default = Yes|student = Yes) = \frac{\pi \cdot P(S|D)}{\pi \cdot P(S|D) + (1 - \pi) \cdot P(S|\bar{D})}$$

Approccio Bayesiano

In assenza di qualsiasi informazione ulteriore, selezionata un'unità a caso, utilizzando la tabella, possiamo stimare:

$$\hat{\pi} = \frac{333}{1000} = 0.333$$
$$\hat{P}(S|D) = \hat{P}(student = Yes|default = Yes) = \frac{127}{333} = 0.38$$
$$\hat{P}(S|\bar{D}) = \hat{P}(student = Yes|default = No) = \frac{2817}{9667} = 0.29$$

Inserendo i valori ottenuti nella formula di Bayes otteniamo:

$$\hat{P}(default = Yes|student = Yes) = \frac{0.0333 \cdot 0.38}{0.0333 \cdot 0.38 + 0.9667 \cdot 0.29} = 0.043$$

L'analisi discriminante utilizza il teorema di Bayes per produrre delle stime che una variabile di risposta qualitativa Y appartenga ad una certa categoria $k = 1, 2, \ldots, K$ sulla base di informazioni fornite da predittori qualitativi e quantitativi.

Perchè un altro metodo di classificazione?

Quando le classi di Y sono ben separate dai predittori, le stime per il modello di regressione logistica sono sorprendentemente instabili. L'analisi discriminante lineare non soffre di questo problema.

Se n è piccolo e la distribuzione del predittori X è approssimativamente normale in ciascuna delle classi, il modello discriminante lineare è più stabile rispetto al modello di regressione logistica.

L'analisi discriminante lineare è più semplice da usare quando abbiamo più di due classi per Y.

Approccio Bayesiano

Supponiamo che la variabile risposta qualitativa Y possa assumere $K \geq 2$ possibili valori distinti (non ordinati).

Indichiamo con $\pi_k, k = 1, 2, \ldots, K$, la probabilità (a priori) che una osservazione scelta a caso appartenga alla classe k.

Indichiamo con

$$f_k(x) = Pr(X = x|Y = k)$$

la funzione di densità (di probabilità) di X condizionata a $Y = k$.

Il ricorso alle densità è necessario poiché X può essere un carattere continuo.

Approccio Bayesiano

Il teorema di Bayes ottiene

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{j=1} K \pi_j f_j(x)}$$

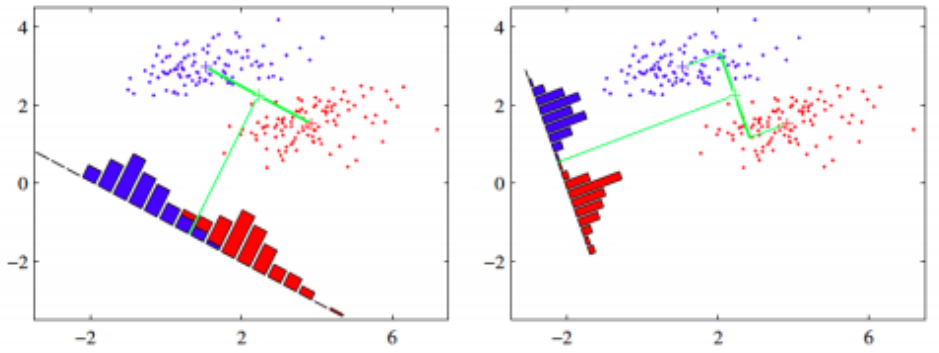
Definiamo

$$p_k(x) = P(Y = k|X = x)$$

la probabilità a posteriori che l'unità appartenga alla classe k dato il valore dei predittori $X = x$.

Poiché le probabilità a priori π_k possono essere facilmente stimate ricorrendo alle frequenze relative di ciascuna classe di Y, il problema si riduce alla determinazione di una buona stima di $f_k(x)$.

Analisi discriminante lineare (LDA): $p = 1$



L'assunzione di base della LDA, che permette di stimare agevolmente $f_k(x)$, è quella per cui f_k è normale o gaussiana:

$$f_k(x) = \frac{1}{\sigma_k \sqrt{2\pi}} e^{\left(-\frac{1}{2\sigma_k^2 (x-\mu_k)^2}\right)}$$

dove μ_k e σ^2 sono rispettivamente la media e la varianza della classe k.

Nella LDA assumiamo che $\sigma_1^2 = \cdots = \sigma_k^2 = \sigma^2$.

Analisi discriminante lineare (LDA): $p = 1$

Le probabilità a posteriori diventeranno:

$$p_k(x) = \frac{\pi_k \frac{1}{\sigma_k\sqrt{2\pi}} e^{\left(-\frac{1}{2\sigma_k^2(x-\mu_k)^2}\right)}}{\sum_{j=1}^{K} \pi_j \frac{1}{\sigma_k\sqrt{2\pi}} e^{\left(-\frac{1}{2\sigma_k^2(x-\mu_k)^2}\right)}}$$

Si ricordi che la regola del classificatore di Bayes assegna l'unità alla classe k per la quale $p_k(x)$ è massima.

Rielaborando l'espressione per $p_k(x)$ si ottiene che $p_k(x)$ è massima quando

$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\mu_k)$$

è massimo.

Analisi discriminante lineare (LDA)

Esempio.

Se $K = 2$ e $\pi_1 = \pi_2 = 0.5$ allora il classificatore di Bayes assegna l'unità alla classe 1 se

$$2x \cdot (\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$$

ed alla classe 2 altrimenti.

Possiamo determinare la frontiera di decisione come

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}$$

LDA: stima dei parametri

Per implementare il metodo in pratica è necessario procedere alla stima dei parametri $\mu_1, \ldots, \mu_k, \sigma^2, \pi_1, \ldots, \pi_K$.

Se n_k indica la numerosità di osservazioni training nel gruppo k, gli stimatori usati sono molto semplici:

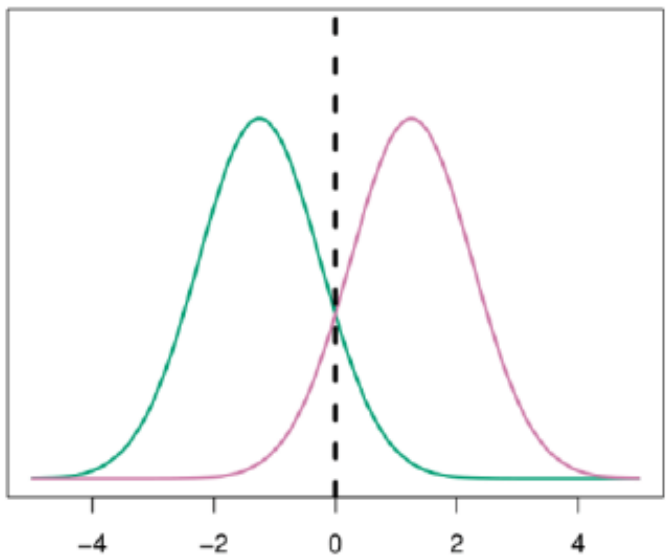
$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

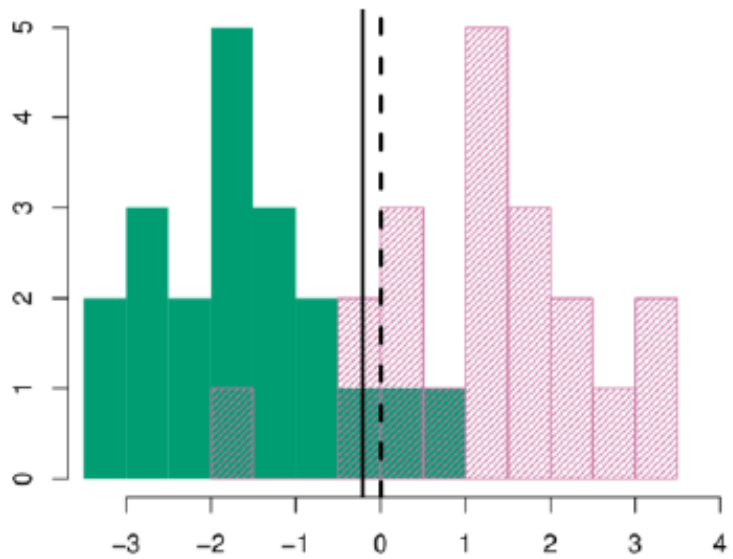
$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \mu_k)^2$$

$$\hat{\pi}_k = \frac{n_k}{n}$$

I parametri stimati sono sostituiti nelle formule viste sopra per ottenere stime della probabilità di appartenenza ad una certa classe e per la classificazione.

LDA: esempio grafico





Sinistra: popolazione. Destra: campione
Tratteggiato: frontiera di decisione di Bayes
Continuo: frontiera di decisione LDA

LDA: $p > 1$

Nel caso in cui vi siano più predittori $X = (X_1, X_2, \ldots, X_p)$ l'assunzione tipica nella LDA è che le ossevarzioni del gruppo k, $(k = 1, 2, \ldots, K)$ provengano da una distribuzione normale multivariata $X \sim N(\mu_k, \Sigma)$.

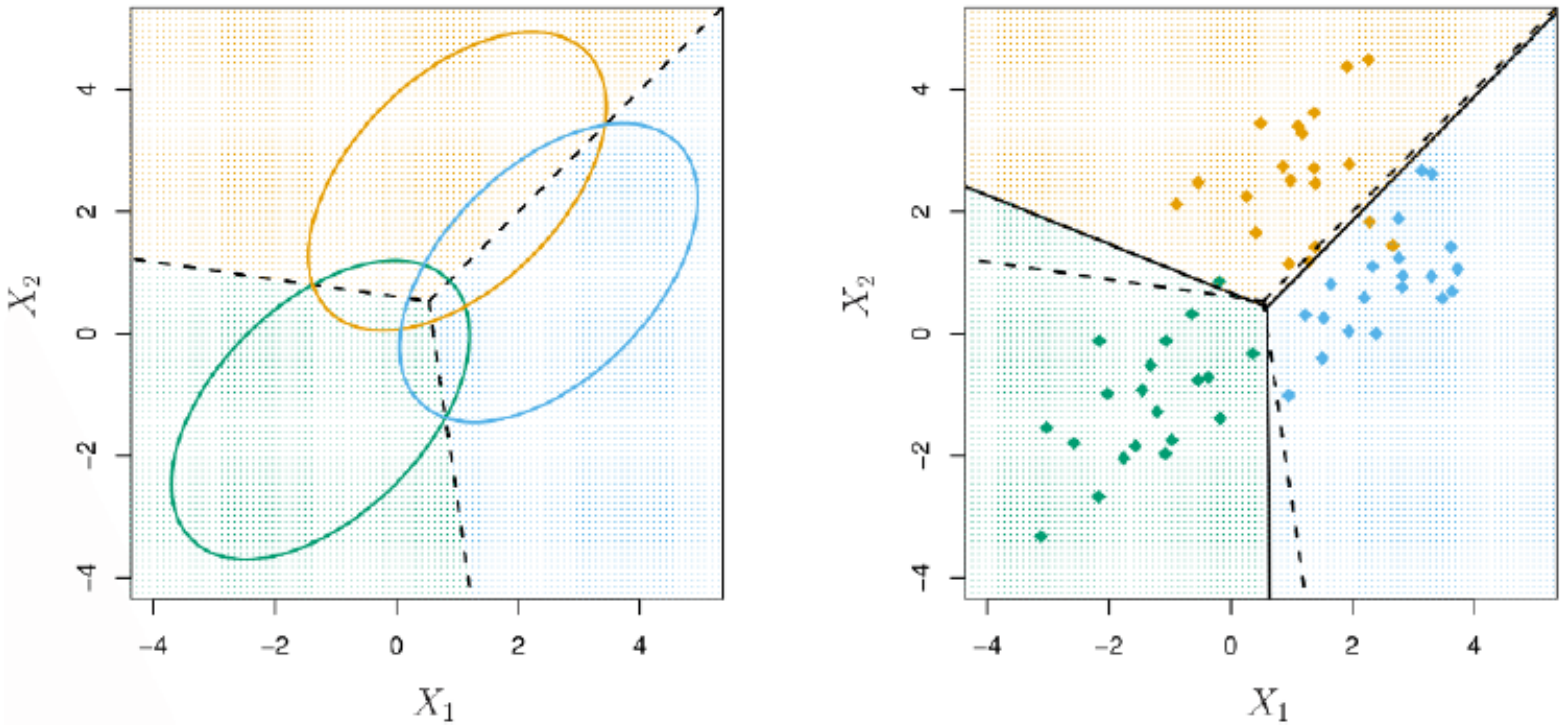
Nota

Nella LDA, il vettore μ_k è specifico per ciascuna classe mentre Σ è comune a tutte le classi K.

Effettuata la stima dei parametri (μ_k, Σ), $k = 1, 2, \ldots, K$, le probabilità $P(Y = k|X = x)$ sono stimate ricorrendo al teorema di Bayes. Attraverso queste è poi possibile procedere alla classificazione delle unità.

LDA: $p > 1$, esempio

Nel grafico seguente vi sono $K = 3$ classi e tre frontiere di decisione.
Il classificatore di Bayes assegna l'unità in base alla regione in cui trova.



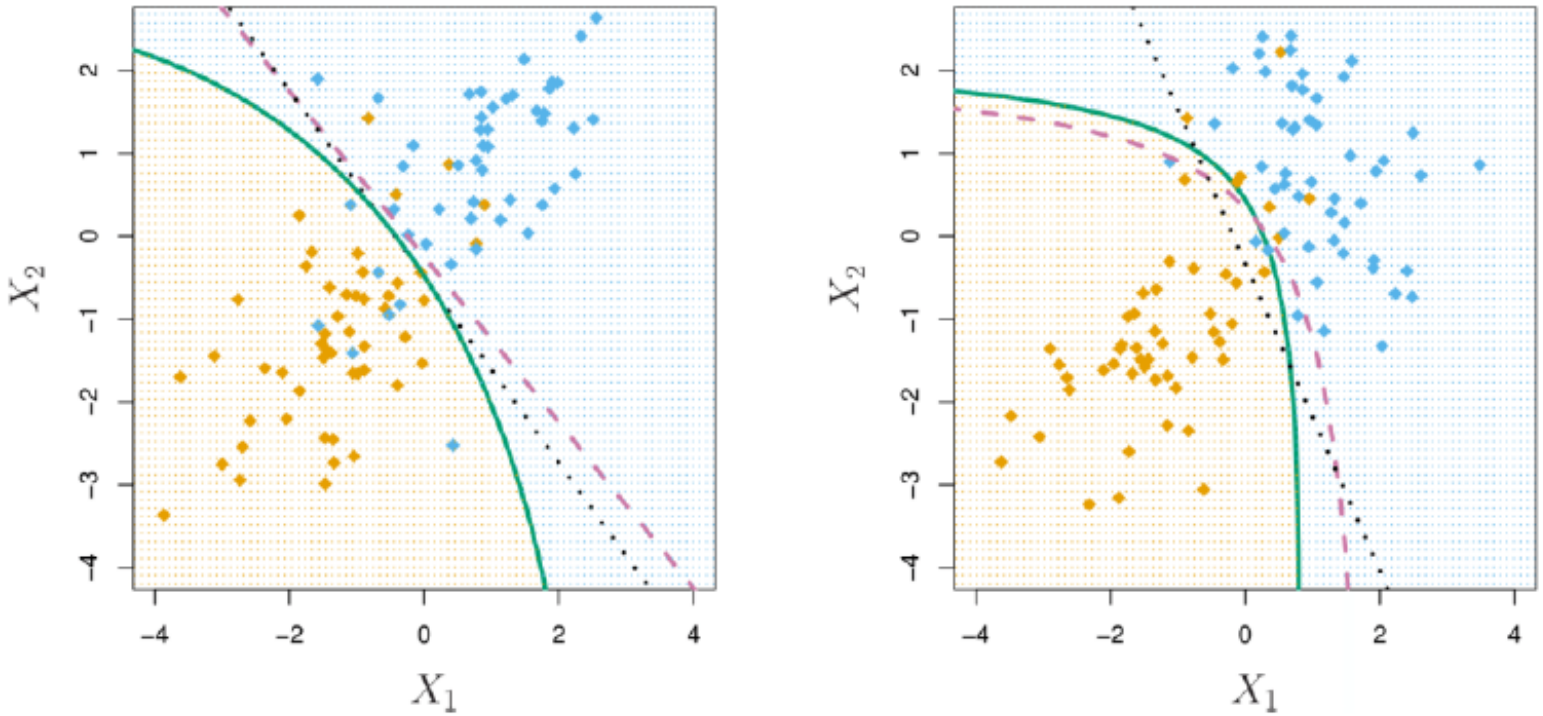
Sinistra: popolazione. Destra: campione
Tratteggiato: frontiera di decisione di Bayes; Continuo: frontiera di decisione LDA

Analisi discriminante quadratica (QDA)

Nella LDA l'assunzione di base è che per ogni classe k, $X \sim N(\mu_k, \Sigma_k)$, ottenendo frontiere di decisione lineari.

Nella QDA l'assunzione di base è che per ogni classe k, $X \sim N(\mu_k, \Sigma_k)$, ossia si prevede la possibilità che ogni gruppo abbia **variabilità diversa**.

In quest'ultimo caso si ottengono frontiere di decisione non-lineari.



LDA-QDA: confronto

LDA è meno flessibile di QDA ed ha quindi minore varianza.

Ma c'è un tradeoff: se l'assunto di LDA che le K classi condividano la stessa matrice di covarianza è totalmente errato, ci può essere bias elevato.

LDA tende ad essere la scelta migliore se ci sono relativamente poche osservazioni training e ridurre la variabilità è cruciale.

Al contrario, QDA è consigliata se il training set è molto grande, o se l'assunzione di una matrice di covarianza comune per le K classi è chiaramente insostenibile.

Mixture discriminant analysis (MDA)

Per MDA si presume che ogni classe sia una miscela gaussiana di sottoclassi, in cui ogni punto ha una probabilità di appartenere a ciascuna classe.

Si assume ancora l'uguaglianza della matrice di covarianza, tra le classi.