Machine Learning

Tommaso Ferrario (@TommasoFerrario18)

Telemaco Terzi (@Tezze2001)

October 2023

Indice

Capitolo 1

Introduzione al Machine Learning

1.1 Introduzione

Definiamo alcuni concetti base:

- **Task** (T), il compito da apprendere. È più facile apprendere attraverso esempi che codificare conoscenza o definire alcuni compiti. Inoltre il comportamento della macchina in un ambiente può essere diverso da quello desiderato, a causa della mutabilità dell'ambiente ed è più semplice cambiare gli esempi che ridisegnare un sistema.
- **Performance** (P), la misura della bontà dell'apprendimento.
- **Experience** (E), l'esperienza sui cui basare l'apprendimento. Il tipo di esperienza scelto può variare molto il risultato e il successo dell'apprendimento.

In merito alle parti "software" distinguiamo:

- **Learner**, la parte di programma che impara dagli esempi in modo automatico.
- **Trainer**, il dataset che fornisce esperienza al learner.

Si hanno tre tipi di apprendimento:

- **Supervised learning**: dove vengono forniti a priori esempi di comportamento e si suppone che il trainer dia la risposta corretta per ogni input (mentre il learner usa gli esempi forniti per apprendere). L'esperienza è fornita da un insieme di coppie:

$$S \equiv \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \tag{1.1}$$

  e, per ogni input ipotetico x_i l'ipotetico trainer restituisce il corretto y_i.

- **Unsupervised learning**: dove si riconosce schemi nell'input senza indicazioni sui valori in uscita. Non c'è target e si ha libertà di classificazione. Si cerca una regolarità e una struttura insita nei dati. In questo caso si ha:

$$S \equiv \{x_1, x_2, \ldots, x_n\} \tag{1.2}$$

  Il clustering è un tipico problema di apprendimento non supervisionato. Non si ha spesso un metodo oggettivo per stabilire le prestazioni che vengono quindi valutate da umani.

- **Reinforcement learning**: dove bisogna apprendere, tramite il learner sulla base della risposta dell'ambiente alle proprie azioni. Si lavora con un addestramento continuo, aggiornando le ipotesi con l'arrivo dei dati. Durante la fase di test bisogna conoscere le prestazioni e valutare la correttezza di quanto appreso. Il learner viene addestrato tramite rewards e quindi apprende una strategia per massimizzare i rewards, detta strategia di comportamento e per valutare la prestazione si cerca di massimizzare "a lungo termine" la ricompensa complessivamente ottenuta.

Possiamo inoltre distinguere due tipi di sistemi di apprendimento:

- **Attivo**: dove il learner può *domandare* sui dati disponibili.

- **Passivo**: dove il learner apprende solo a partire dai dati disponibili.

L'obiettivo degli algoritmi di apprendimento automatico è quello di fornire per ogni istanza di addestramento una risposta eventualmente corrispondente al nostro target, qualora esista. Per rappresentare la soluzione esistono diverse metodologie:

- Booleano: $I \to \{0, 1\}$.
- Multi-classificazione: $I :\to \{A, B, C, \dots\}$.
- Calcolato da una funzione: $O = f(I)$.
- Dei cluster di istanze.

La correttezza della soluzione fornita deve essere valutata con dei metodi appositi per il modello selezionato.

1.1.1 Terminologia

- X, **spazio delle istanze**, ovvero la collezione di tutte le possibili istanze. In termini statistici lo spazio delle istanze non è altro che lo *spazio campione*.
- $x \in X$, **istanza**, ovvero un singolo "oggetto" preso dallo spazio delle istanze. Ogni istanza è rappresentata tramite un vettore di attributi unici.
- c, **concetto**, $c \subseteq X$, ovvero un sottoinsieme dello spazio delle istanze che descrive una classe di oggetti alla quale siamo interessati per costruire un modello di machine learning. La nozione statistica equivalente è quella di evento, ovvero un sottoinsieme dello spazio campione. Si ha quindi che, preso un concetto $A \subseteq X$:

$$f_A : X \to \{0, 1\} \implies f_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{altrimenti} \end{cases} \tag{1.3}$$

- h, **ipotesi**, $h \subseteq X$. ovvero una congiunzione $\wedge$ di vincoli sugli attributi. Tale ipotesi è **consistente**, ovvero è coerente con tutti gli esempi.

$$Consistente(h, D) \equiv \forall \langle x, c(x) \rangle \in D \text{ vale } h(x) = c(x) \tag{1.4}$$

  Un'istanza x **soddisfa** un'ipotesi h se e solo se tutti i vincoli espressi da h sono soddisfatti dai valori di x e si indica con:

$$h(x) = 1 \tag{1.5}$$

- H, **spazio delle ipotesi**.
- $(x, f(x))$, **esempio**, ovvero prendo un'istanza e la vado ad etichettare con la sua classe di appartenenza. La funzione f è detta funzione target.
- $D = \{(x_1, f(x_1)), \dots, (x_n, f(x_n))\}$, **training set**, ovvero è la raccolta degli esempi. Qualora si avesse a che fare con un training non supervisionato si avrebbe: $D = \{x_1, \dots, x_n\}$.
- $\{(x'_1, f(x'_1)), \dots, (x'_n, f(x'_n))\}$, **test**.
- Un modello di machine learning è un insieme di parametri θ di una funzione f che, dato uno spazio delle istanze in input X effettua delle previsioni Y.
- **Linguaggio delle ipotesi**, è il linguaggio che definisce lo spazio delle ipotesi/modelli.
- **Cross validation**, è una tecnica statistica utilizzabile in presenza di una buona numerosità del dataset di training. Tale tecnica consiste nella suddivisione del dataset totale in k parti (k-fold validation) e, ad ogni passo, la parte $\left(\frac{1}{k}\right)$-esima del dataset viene ad essere il validation dataset, mentre la restante parte costituisce il training dataset. Così, per ognuna delle k parti si allena il modello, evitando quindi problemi di overfitting, ma anche di campionamento asimmetrico del training dataset, tipico della suddivisione del dataset in due sole parti.
- **Bias**: è un fenomeno che si verifica a causa di presupposti errati nel processo di apprendimento automatico. Il bias è come un errore sistematico che si verifica quando un algoritmo produce risultati sistematicamente distorti a causa di alcune ipotesi errate nel processo di apprendimento automatico.

- **Version space** ($VS_{H,D}$): dove H rappresenta lo spazio delle ipotesi e D il dataset. È il sotto-insieme dello spazio delle ipotesi che è anche consistente con il dataset:

$$VS_{H,D} = \{h \in H \text{ tale che } Consistente(h, D)\} \tag{1.6}$$

Definizione 1 (**Independent and identically distributed**). *I dati sono campionati in modo indipendente e dalla stessa distribuzione. Nella pratica questa assunzione è verificata in quanto il modello considera un solo esempio alla volta, ignorando le features degli altri esempi.*

1.2 Inductive Learning

Si parla di **inductive learning** quando voglio apprendere una funzione da un esempio. Si cerca quindi un'ipotesi h, a partire da un insieme di esempi di apprendimento, tale per cui $h \approx f$.

Questo è un modello semplificato dell'apprendimento reale in quanto si ignorano a priori conoscenze e si assume di avere un insieme di dati.

L'ipotesi h è **consistente** con gli esempi di addestramento se e solo se $h(x) = f(x)$ per ogni esempi di training $\langle x, f(x)\rangle$.

Bisogna però mettere in conto anche eventuali errori, cercando di capire se esiste davvero un'ipotesi coerente e, in caso di assenza, si cerca di approssimare la soluzione. In quest'ottica bisogna prestare attenzione tra fit e complessità. Ogni sistema dovrà cercare di mediare tra questi due aspetti, un fit migliore comporta alta complessità. Si ha sempre il rischio di **overfitting**, cercando una precisione dei dati che magari non esiste. Si ha un generatore di dati ma il sistema non ha conoscenza della totalità degli stessi.

Definizione 2. *L'**overfitting**, è un problema che si verifica quando un modello statistico o di machine learning si adatta troppo ai dati di allenamento e non è in grado di generalizzare o prevedere bene i dati nuovi o di test. Questo fenomeno è causato da un eccesso di parametri o di complessità del modello rispetto al numero di osservazioni.*

Bassi tassi di errore e una varianza elevata sono buoni indicatori di overfitting. Per prevenire questo tipo di comportamento, una parte del set di dati di addestramento è tipicamente messa da parte come "set di test" per controllare l'eventuale presenza di overfitting. Dati di addestramento con un basso tasso di errore e dati di test con un alto tasso di errore sono indice di overfitting.

Viene usato un approccio che sfrutta anche il *Rasoio di Occam*, ovvero si preferisce la soluzione più semplice.

1.3 Concept Learning

Definizione 3. *Il **concept learning** è la ricerca, nello spazio delle ipotesi, di funzioni che assumano valori all'interno di* $\{0, 1\}$*. In altre parole si parla di funzioni che hanno come dominio lo spazio delle ipotesi e come codominio* $\{0, 1\}$*:*

$$f : H \to \{0, 1\} \tag{1.7}$$

Volendo si possono usare insiemi e non funzioni. Si cerca quindi con opportune procedure la miglior ipotesi che si adatta meglio al concetto implicato dal training set

In questo contesto si cerca di capire quale funzione booleana è adatta al mio addestramento. In altre parole si cerca di apprendere un'ipotesi booleana partendo da esempi di training composti da input e output della funzione.

Qualora nel concept learning si abbia a che fare con più di due possibilità si aumentano i bit usati. Nel concept learning un'ipotesi è un insieme di vincoli sugli attributi e ogni valore può essere:

- **Specificato**.
- **Non importante**, che si indica con "?", e che può assumere qualsiasi valore. Avere un'ipotesi con tutti i valori del vettore pari a "?" implica avere l'ipotesi più generale, avendo classificato tutte le istanze solo come esempi positivi.
- **Nullo** e si indica con $\emptyset$. Avere un'ipotesi con tutti i valori del vettore pari a $\emptyset$ implica avere l'ipotesi più specifica, avendo classificato tutte le istanze solo come esempi negativi.

Si ha la teoria delle ipotesi di apprendimento induttivo che dice che se la mia h approssima bene nel training set allora approssima bene su tutti gli esempi non ancora osservati. Il concept learning è quindi una ricerca del fit migliore.

Definizione 4. *Siano h_j e h_k due funzioni che restituiscono un valore booleano. Allora h_j è uguale o più generale rispetto a h_k ($h_j \geq h_k$) se e solo se:*

$$\forall x \in X : [(h_k(x) = 1) \rightarrow (h_j(x) = 1)] \tag{1.8}$$

La relazione $\geq$ impone un ordine parziale sullo spazio delle ipotesi H che viene utilizzato da diversi metodi di concept learning.

Lo spazio delle ipotesi è descritto da una congiunzione di attributi

Vediamo un esempio di studio dello spazio delle istanze X. Considero che le istanze sono specificate da due attributi: $A_1 = \{0, 1, 2\}$ e $A_2 = \{0, 1\}$. Quindi, sapendo che $X = A_1 \times A_2$, ho che:

$$|X| = |A_1| \cdot |A2|$$

e quindi in questo caso $|X| = 6$.

Vediamo un esempio di studio del numero di concetti. In X abbiamo 3 attributi, ciascuno con 3 possibili valori: $A_i = \{0, 1, 2\}$, $i = 1, 2, 3$ Abbiamo già visto come calcolare $|X|$ e quindi sappiamo che:

$$|X| = 3^3 = 27$$

Sappiamo che un concetto c è un sottoinsieme di X, quindi, chiamato $C = \{c_i \subseteq X\}$ l'insieme di tutti i concetti so che la sua cardinalità è pari alla cardinalità dell'insieme delle parti di X:

$$|C| = |P(X)| = 2^{|X|} = 2^{27}$$

Vediamo un esempio di studio del numero delle ipotesi, ovvero si studia la cardinalità dello spazio delle ipotesi, ossia il numero di differenti combinazioni di tutti i possibili valori per ogni possibile attributo. Bisogna notare che l'uso di $\emptyset$ come valore di un attributo in un'ipotesi rende tale ipotesi semanticamente equivalente a qualsiasi altra che contenga un $\emptyset$. Inoltre tutte queste sono ipotesi che *rifiutano tutto*. Tutte queste ipotesi conteranno come un unico caso nel conteggio delle ipotesi. Quindi per conteggiare tutte ipotesi semanticamente differenti dovrò fare, indicando con $|A_i|$ il numero di valori possibili per l'attributo A_i:

$$|H|_{sem} = 1 + \prod(|A_i| + 1) \tag{1.9}$$

Quindi moltiplicare tutti il numero di valori di ogni attributo più 1 (indicante "?" e che quindi va conteggiato come possibile valore per ogni attributi) e sommare 1 (indicante $\emptyset$ che va conteggiato solo una volta per il discorso fatto sopra in merito all'equivalenza semantica di tali ipotesi) a questo risultato finale.

Qualora volessi conteggiare tutte ipotesi sintatticamente differenti dovrò contare $\emptyset$ come ipotetico valore per ogni attributo e quindi:

$$|H|_{sint} = \prod(|A_i| + 2) \tag{1.10}$$

1.3.1 Algoritmo find - S

Questo algoritmo permette di partire dall'ipotesi più specifica e generalizzarla, trovando ad ogni passo un'ipotesi più specifica e consistente con il training set D. L'ipotesi in uscita sarà anche consistente con gli esempi negativi dando prova che il target è effettivamente in H. Con questo algoritmo non si può dimostrare di aver trovato l'unica ipotesi consistente con gli esempi e, ignorando gli esempi negativi non posso capire se D contiene dati inconsistenti. Inoltre non ho l'ipotesi più generale.

Osservazione 1. *Un modello che non fa ipotesi preliminari per quanto riguarda l'identità del concetto target non ha alcuna base razionale per classificare eventuali nuove istanze.*

Algorithm 1 Algoritmo Find-S

function FINDS
  $h \leftarrow$ l'ipotesi più specifica in H
  for *ogni istanza di training* ***positiva*** x **do**
    for *ogni vincolo di attributo* a_i *in* h **do**
      if il vincolo di attributo a_i in h è soddisfatto da x **then**
        non fare nulla
      else
        sostituisci a_i *in* h *con il successivo vincolo più generale che è soddisfatto da* x
      end if
    end for
  end for
  return *ipotesi* h
end function

Capitolo 2

Alberi decisionali

Vediamo come sfruttare una struttura dati discreta, l'albero di decisione, per affrontare problemi di concept learning. Per la costruzione di queste strutture utilizzeremo l'algoritmo chiamato **ID3**, il quale costruisce l'albero scegliendo gli attributi in base al valore dell'**information gain**. In questo caso, a differenza del concept learning in cui si analizza un'istanza alla volta, si prendono in considerazione gruppi di dati.

Gli attributi possono assumere diversi valori. Un albero decisionale è formato da:

- **Nodes** (nodi): rappresentano gli attributi da testare.
- **Branches** (rami): sono etichettati con i possibili valori dell'attributo specificato nel nodo sorgente del ramo.
- **Leaf nodes** (foglie): contengono i valori della classificazione.

Possiamo quindi scegliere, al posto delle classiche funzioni booleane, alberi di decisione per rappresentare un modello che applicato ad esempi non visti ci dirà se applicare in output un'etichetta vera o falsa in base a quanto appreso. Per classificare un'occorrenza si parte dal nodo radice dell'albero, si verificano i tratti indicati da questo nodo e si procede lungo il ramo dell'albero confrontandoli con il valore dell'attributo. Si ripete questo controllo fino ad arrivare alle foglie.

La flessibilità nella costruzione dell'albero sta nel scegliere gli attributi e i valori di ognuno. Con l'algoritmo ID3 si costruiscono alberi decisionali in base alle istanze che ricevo. Formule booleane possono essere rappresentate in un albero decisionale, costruendo un albero che fornisca la risposta yes solo nei casi in cui la formula booleana sia vera.

Riassumiamo alcune caratteristiche degli alberi decisionali:

- Abbiamo attributi con valori discreti.
- Abbiamo un target di uscita discreto, le foglie hanno valori precisi.
- Posso costruire ipotesi anche con disgiunzioni.
- Può esserci "rumore" nel training dei dati.
- Possono esserci attributi di cui non ho informazioni.

Un percorso dalla radice fino alla foglia mi rappresenta una congiunzione di attributi, mentre analizzando l'albero nella sua interezza posso definirlo come una disgiunzione di congiunzioni. Un albero quindi formalizza la congiunzione di vincoli su attributi, percorso da radice a foglia, ma può anche estendere il linguaggio a disgiunzioni di vincoli su attributi.

Posso avere alberi diversi a seconda della scelta del nodo radice.

Teorema 1. *Con un albero decisionale posso rappresentare tutte le funzioni booleane.*

Possiamo dire che gli alberi decisionali descrivono tutte le funzioni booleane. Avendo n attributi booleani avremo un numero distinto di tabelle di verità, ciascuna con 2^n righe, pari a:

$$\text{Numero di alberi } = 2^{2^n} \tag{2.1}$$

Usando funzioni booleane posso convertire tabelle di verità in alberi decisionali, con quindi ogni percorso dalla radice ad una foglia che rappresenta una regola e l'intero albero che rappresenta la congiunzione di tutte le regole. Le foglie quindi sono gli assegnamenti di verità della tabella.

Vediamo quindi un algoritmo generale per la costruzione dell'albero:

1. Si inizia con un albero vuoto.
2. Scelgo un attributo opportuno per fare lo split dei dati.
3. Per ogni split dell'albero:
   - Se non c'è altro da fare si fa la predizione con l'ultimo nodo foglia.
   - Altrimenti si torna allo step 2 e si procede con un altro split.

Una strategia per selezionare l'attributo per fare lo split, può essere quella "a maggioranza", ovvero prendendo l'attributo che con i suoi valori ha meno disomogeneità. Per farlo calcolo la differenza tra yes e no di ogni valore per un certo attributo, sommandone i risultati. Scelgo l'attributo con più omogeneità, che ha una distribuzione più pulita dei risultati.

Idealmente, un buon attributo divide gli esempi in due sotto-insiemi che sono composti tutti da istanze positive o da istanze negative.

Un altra metodologia per scegliere come effettuare lo split consiste nell'utilizzo del **Gini Index**.

Definizione 5. *Per un insieme di istanze posso definire il* ***Gini index*** *come:*

$$Gini(D) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.2}$$

dove:

- *D è l'insieme di istanze.*
- *c rappresenta il numero di classi.*
- *p_i è la proporzione di istanze nella classe i che sono presenti nel dataset.*

Il valore di questo indice è compreso tra 0 e 1, dove 0 indica che nell'insieme sono presenti solo elementi di una classe, mentre 1 indica che nelle classi sono presenti istanze diverse.

Questo indice può essere utilizzato per calcolare la *purezza* dell'intero dataset, oppure per calcolare la *purezza* delle partizioni che si creano utilizzando un determinato attributo per fare lo split.

Per selezionare che attributo utilizzare per lo split tramite l'indice di Gini si calcola una somma pesata per ogni attributo:

$$WeightedSum = \sum_{i} p_i \cdot Gini_i \tag{2.3}$$

dove:

- p_i: rappresenta la probabilità di osservare uno specifico valore.
- $Gini_i$: rappresenta l'indice di Gini calcolato dove l'attributo assume il valore osservato.

2.1 Algoritmo ID3

Lo scopo di questo algoritmo è quello di trovare un albero, di dimensioni ridotte, che sia consistente con gli esempi di training. L'idea alla base di questo algoritmo è quella di scegliere, in modo ricorsivo, l'attributo più significativo come radice del sotto-albero.

Si fanno quindi crescere in modo coerente gli alberi piccoli scelti in partenza. Si punta ad arrivare ad un albero valido per tutti gli esempi ricevuti e anche per quelli non visti.

Iniziamo a vedere l'algoritmo anche se saranno necessarie molte specifiche:

Per la selezione del miglior attributo, bisogna capire cosa si intende come attributo migliore. Per farlo introduciamo la seguente notazione:

$$[\text{esempi positivi}+, \text{esempi negativi}-] \tag{2.4}$$

entrambi rappresentati con un valore intero.

Per essere ancora più precisi bisogna considerare l'**entropia** degli insiemi, ovvero il contenuto informativo dell'insieme.

Algorithm 2 Algoritmo ID3

```
function ID3
    A ← il "miglior" attributo di decisione per il prossimo nodo
    Assegno A come attributo di decisione per il nodo
    for Ogni valore dell'attributo A do
        Creo un discendente
    end for
    Ordina gli esempi di training alla foglia in base al valore dell'attributo del branch
    if Ho classificato tutti gli esempi di training then
        Mi fermo
    else
        Itero sulle foglie appena create
    end if
end function
```

Definizione 6. *Dato un training set S contenete i valori v_i, $i = 1 \dots n$. Possiamo definire l'**entropia** (Contenuto informativo) di un insieme S come:*

$$H[V] = I(P(v_1), \dots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i) \tag{2.5}$$

Dove $P(y)$ rappresenta la probabilità che il valore y sia presente.

Nel caso booleano le istanze presenti in un certo insieme S sono associate ad un'etichetta, conteggiandole. Si utilizza una variabile p per rappresentare i valori di S a cui è associata un'etichetta positiva, mentre, si utilizza la variabile n per rappresentare i valori di S a cui è associata un'etichetta negativa. Posso quindi riscrivere la sommatoria nel caso booleano come:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \left(\frac{p}{p+n}\right) - \frac{n}{p+n} \log_2 \left(\frac{n}{p+n}\right) \tag{2.6}$$

Se inoltre diciamo che p_+ è la proporzione di esempi positivi e p_- di quelli negativi possiamo misurare l'impurità di S utilizzando l'entropia:

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_- \tag{2.7}$$

Avrò quindi alta entropia se positivi e negativi sono bilanciati.

A volte risulta utile calcolare l'entropia condizionata al valore di un attributo. Questo si può fare utilizzando la seguente formula:

$$H[Y|X] = \sum_{i=1}^{n} p(X = x_i) \cdot H[Y|X = x_i] \tag{2.8}$$

Con il termine **information gain** IG ci riferiamo al valore che viene calcolato su ogni attributo A presente nelle istanze contenute in S:

$$IG(S, A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A) \tag{2.9}$$

dove $remainder(A)$ rappresenta la somma pesata sul totale delle entropie calcolate sui sotto-insiemi che si vanno a creare se si sceglie A come attributo per lo split dei dati.

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \cdot I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right) \tag{2.10}$$

Si sceglie l'attributo il valore dell'information gain più alto.

Facciamo qualche osservazione finale sull'algoritmo ID3:

- Lo spazio delle ipotesi è completo e sicuramente contiene il target.
- Ho in output una singola ipotesi.

- Non si ha backtracking sugli attributi selezionati, si procede con una ricerca greedy.
- Fa scelte basate su una ricerca statistica, facendo sparire incertezze sui dati.
- Il bias non è sulla classe iniziale, essendo lo spazio delle ipotesi completo, ma sulla scelta di solo alcune funzioni, preferendo alberi corti e posizionando attributi ad alto information gain vicino alla radice. Il bias è quindi sulla preferenza di alcune ipotesi. Si usa il criterio euristico di rasoio di Occam.
- H è l'insieme potenza delle istanze X.

Viene introdotto però l'**overfitting**.

Definizione 7. *Definiamo formalmente l'**overfitting** come l'adattamento eccessivo, ovvero quando un un modello statistico molto complesso si adatta al campione perché ha un numero eccessivo di parametri rispetto al numero di osservazioni. Si ha quindi che un modello assurdo e sbagliato può adattarsi perfettamente se è abbastanza complesso rispetto alla quantità di dati disponibili.*

Nel machine learning se il learner viene addestrato troppo a lungo il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi quindi le prestazioni sui dati non visionati saranno drasticamente peggiori. L'opposto è l'underfitting.

Se misuro l'errore di una ipotesi h sul training set ($error_{train}(h)$) e poi misuro l'errore di quella ipotesi sull'intero set delle possibili istanze D ($error_D(h)$) ho che l'ipotesi h va in overfit sul quel data set se:

$$error_{train}(h) < error_{train}(h') \ \wedge \ error_D(h) > error_D(h') \tag{2.11}$$

quindi vado in overfitting se: presa un'altra ipotesi h' questa presenta un errore maggiore di h sul training set, ma l'errore commesso sull'intero dataset è minore di h. Il problema è che non posso sapere se esiste tale h'.

Per evitare il problema uso sempre il rasoio di Occam scegliendo ipotesi semplici ed evitando di far crescere l'albero quando lo "split" non è statisticamente significativo.

Un altro modo è quello di togliere pezzi, all'albero, che toccano poche istanze o pure calcolare una misura di complessità dell'albero, minimizzando la grandezza dell'albero e gli errori del training set, usando il **Minimum Description Length** (MDL).

In ID3 quindi posso scegliere sia in base all'information gain massimo o all'entropia minima tra gli attributi.

Capitolo 3

Reti neurali

3.1 Neuroni biologico

L'idea alla base delle reti neurali è quella di replicare, grazie all'utilizzo di un computer, il funzionamento del cervello umano. Per fare ciò si è utile farsi un idea su come funziona il cervello. Esso è composto da un insieme di neuroni molto grande di neuroni che comunicano tra di loro. Ogni neurone è composto nel seguente modo:

- **Corpo**: che implementa tutte le funzioni logiche del neurone.
- **Assone**: ovvero il canale di uscita verso gli altri neuroni, è quello che si occupa di trasmettere gli impulsi nervosi.
- **Dendrite**: la parte che permette al neurone di ricevere gli impulsi nervosi.
- **Sinapsi**: ovvero la regione funzionale in cui avviene lo scambio dei segnali, ovvero dove ogni singolo ramo terminale dell'assone del neurone trasmette impulsi nervosi provenienti dal neurone ai dendriti di altri neuroni.

Le connessioni tra i neuroni si modificano con l'apprendimento. Inoltre, l'informazione non è localizzata ma è distribuita globalmente nella rete dei processi.

Si hanno due stati possibili per il neurone biologico:

- **Eccitazione**: quando il neurone invia, tramite le sinapsi, segnali *pesati* ai neuroni connessi.
- **Inibizione**: quando il neurone non invia segnali.

La transizione di stato dipende dall'entità complessiva dei segnali eccitatori e inibitori ricevuti dal neurone.

3.2 Neuroni formali

Partendo dalla descrizione biologica possiamo passare a una rappresentazione matematica del neurone:

$$\langle n, C, W, \theta \rangle \tag{3.1}$$

dove:

- n specifica il nome del neurone stesso.
- C specifica l'insieme dei canali di input c_i.
- W specifica il vettore dei pesi w_i, associati ad ogni canale di input c_i. È una rappresentazione formale dei pesi delle sinapsi.
- θ specifica la soglia, utile per definire quando un neurone manda effettivamente il segnale.

Iniziamo analizzando il neurone più semplice, ovvero quello binario. In questo caso, gli stati possono assumere un valore dell'insieme $\{0, 1\}$ oppure $\{-1, 1\}$. Se definiamo w_i il peso associato all'input i, allora possiamo definire la funzione di transizione come:

$$s(t+1) = 1 \iff \sum w_i \cdot s_i(t) \geq \theta \tag{3.2}$$

L'insieme degli stati, rappresentato in modo binario, comporta che la funzione di transizione sia una funzione a scalino:

$$f(x) = \begin{cases} 1 & se\ x \geq \theta \\ 0 & altrimenti \end{cases} \qquad con\ x = \sum w_i \cdot s_i \tag{3.3}$$

Questo modello è comunque estremamente semplificato. L'insieme degli stati potrebbe non essere booleano ma potrebbe essere $\mathbb{R}$, infatti anche nella biologia il segnale in uscita dai neuroni è graduato e continuo. Si potrebbe quindi avere una funzione logistica o sigmoide:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad con\ x = \sum w_i \cdot s_i \tag{3.4}$$

3.3 Percettrone

Il **percettrone** - **Linear treshold unit** (LTU) è la tipologia di rete neurale più semplice.

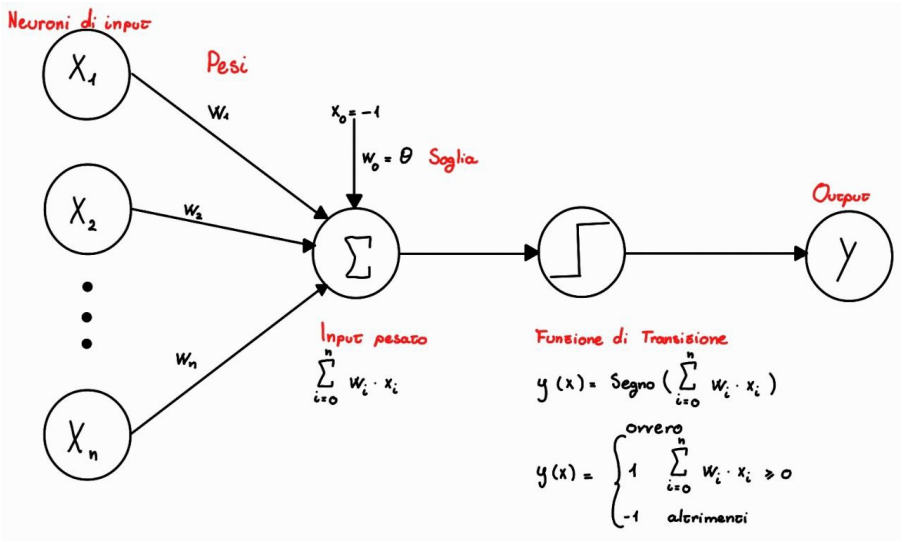

Figura 3.1: Rappresentazione grafica del percettrone

Un percettrone è composto da un vettore di input x_i, ognuno con associato un peso w_i. Oltre a questi, esso prende in input una soglia θ, la quale è trattata come il peso di una unità di input in stato fisso $x_0 = -1$. La transizione di stato è determinata dal risultato del prodotto interno tra il vettore dei pesi $\mathbf{w}$ e il vettore degli stati di input $\mathbf{x}$. Da un punto di vista geometrico, il vettore dei pesi determina un iperpiano che separa i possibili vettori di input in due classi, a seconda che formino con $\mathbf{w}$ un angolo acuto oppure ottuso.

Solitamente coi percettroni si parla di learning supervisionato.

Il processo di apprendimento di un percettrone parte assegnando in modo casuale i pesi agli input. Questi pesi verranno modificati nella fase di apprendimento.

Se assumiamo che l'obiettivo è quello di separare i vettori in input in due classi A e B, possiamo utilizzare la seguente procedura per raggiungere questo scopo. Si sottomette una sequenza infinita $\{x_k\}$ di vettori tale che ve ne siano un numero infinito sia di A che di B. Per ogni x_k la rete calcola la risposta y_k. Se la risposta è errata, si modificano i pesi nel seguente modo:

- Incremento i pesi delle unità di input attive se si è risposto 0 anziché 1.
- Decremento i pesi delle unità di input attive se si è risposto 1 anziché 0.

$$w' = w \pm x \tag{3.5}$$

Vediamo due teoremi utili per lo studio dell'apprendimento del percettrone su due classi A e B, banalmente rappresentanti, nella nostra situazione binaria e semplificata, il caso in cui si abbia il neurone che emette il segnale o altrimenti. Nel nostro caso le classi sono discriminabili.

Teorema 2 (**Teorema della convergenza**). *Comunque si scelgano i pesi iniziali, se le classi A e B sono discriminabili, la procedura di apprendimento termina dopo un numero finito di passi.*

Teorema 3 (**Teorema di Minsky e Papert**). *La classe delle forme discriminabili da un percettrone semplice è limitata alle forme linearmente separabili.*

Teorema 4. *Se l'insieme degli input estesi è partito in due classi linearmente separabili A e B allora è possibile trovare un vettore di pesi w tale che:*

$$\begin{array}{ll} \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 & se \quad x \in A \\ \boldsymbol{w} \cdot \boldsymbol{x} < 0 & se \quad x \in B \end{array} \tag{3.6}$$

Vediamo ora una nuova regola per aggiornare i pesi del percettrone:

$$w_i' = w_i + \Delta w_i = w_i + \eta \cdot (t - y) \cdot x_i \tag{3.7}$$

dove:

- t rappresenta il valore del target.
- y rappresenta l'output del percettrone.
- η rappresenta il learning rate.

Così facendo si aggiornano i pesi del percettrone quando esso sbaglia la risposta. Usando questa regola, l'algoritmo converge alla classificazione corretta se i dati del training sono linearmente separabili e se η assume un valore abbastanza piccolo.

3.3.1 Discesa del gradiente

Si consideri ora una unità lineare, con stati continui e un output calcolato come:

$$y = w_0 + w_1 x_1 + \cdots + w_n x_x \tag{3.8}$$

Una possibile strategia di apprendimento per questo modello consiste nel minimizzare una opportuna funzione dei pesi w_i. Questo risultato può essere ottenuto cercando di minimizzare l'errore ottenuto sul training set usando la discesa del gradiente:

$$w' = w - \eta \cdot \nabla E[\mathbf{w}] \tag{3.9}$$

dove posso approssimare la discesa del gradiente come:

$$\sum_d (t_d - y_d)(-x_i) \tag{3.10}$$

Per effettuare questo aggiornamento abbiamo due modalità:

- **Modalità Batch**: dove l'aggiornamento dei pesi viene calcolato rispetto all'intero insieme D.

$$w = w - \eta \cdot \nabla E_D[w] \text{ con } E_D[w] = \frac{1}{2} \sum_{d \in D} (t_d - y_d)^2 \tag{3.11}$$

  Esiste una variante chiamata **mini batch** che consiste nel dividere il dataset in n porzioni e aggiornare i pesi ogni volta che si è analizzato un batch.

- **Modalità incrementale**: dove l'aggiornamento dei pesi viene calcolato rispetto a singoli esempi d.

$$w = w - \eta \cdot \nabla E_d[w] \text{ con } E_d[w] = \frac{1}{2} (t_d - y_d)^2 \tag{3.12}$$

  Questa modalità può approssimare la discesa del gradiente con la modalità Batch arbitrariamente se η è abbastanza piccolo.

Rispetto a quanto visto per il percettrone la discesa lungo il gradiente converge all'ipotesi con il minimo errore quadratico se η è abbastanza basso, anche per dati di training molto rumorosi.

3.4 Reti neurali

Fino ad ora abbia studiato il comportamento del singolo neurone, ma come per il cervello umano, la forza di questo approccio è data dall'unione di più neuroni. Per realizzare questo risultato la soluzione consiste nel collegare i neuroni in modo da ottenere una struttura a grafo aciclico orientato. Così facendo, l'output di un neurone diventa l'input di un altro.

Le reti neurali così composte hanno delle caratteristiche strutturali come:

- hanno un gran numero di unità.
- permettono operazioni elementari.
- hanno un alto livello di interconnessione.

e delle caratteristiche dinamiche:

- Si hanno cambiamenti di stato in funzione dello stato dei neuroni collegati in input.
- Si ha una funzione di uscita per ogni unità.
- Si ha la modifica dello schema di connessione, tramite la modifica dei pesi, per l'apprendimento.

L'output di un neurone è uno stato per un altro neurone. Si hanno alcuni elementi caratterizzanti di una rete neurale:

- Il tipo di unità.
- La topologia, ovvero la direzione delle connessioni, il numero di neuroni, ...
- Le modalità di attivazione.
- Un algoritmo di apprendimento con lo studio dei pesi.

L'apprendimento di un neurone modifica i pesi sinaptici. Se due neuroni connessi sono per più volte di seguito contemporaneamente attivi, il peso della sinapsi aumenta.

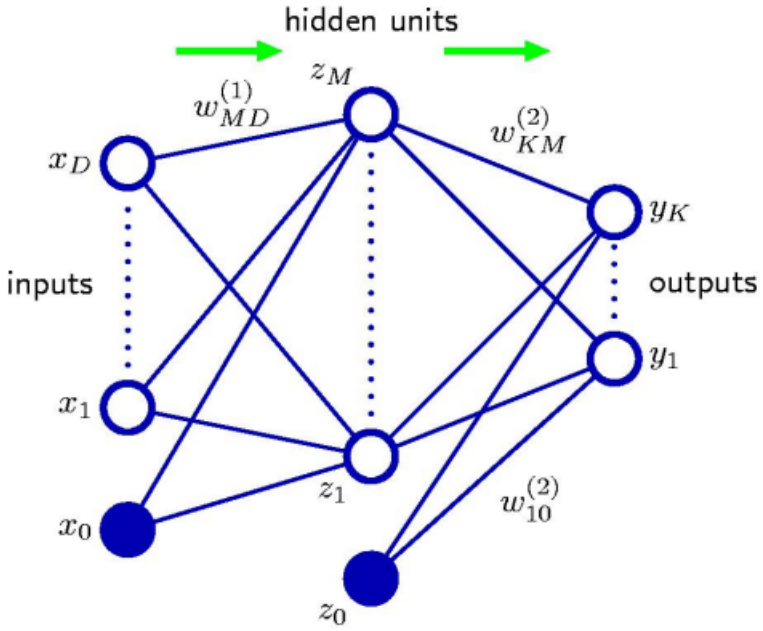

Figura 3.2: Rete Neurale multi strato

Dal punto di vista formale, dovendo espandere le definizioni fatte nel caso del singolo neurone a più neuroni, si lavora con:

- D le variabili di input $x_1, \ldots, x_D$.

- Una matrice dei pesi W, con i valori indicati tramite w_{ij}, per gli archi che collegano i neuroni. I pesi sono i pesi correnti.
- Un vettore delle soglie Θ, con i valori indicati tramite θ_i, una per ogni neurone.
- L'input netto per il neurone i al tempo t, indicato con:

$$n_i(t) = \sum_{j=1}^{n} w_{ij} \cdot x_j(t) - \theta_i \tag{3.13}$$

  quindi si ha che la soglia influisce già nell'input del neurone e quindi θ_i viene considerata come soglia di azzeramento del valore entrante.
- La funzione di transizione indicata con:

$$s_i(t+1) = g(n_i(t)) \tag{3.14}$$

- M ovvero il numero delle unità di attivazione nascoste:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3.15}$$

  dove $j = 1, \ldots, M$.
- $z_j = h(a_j)$ come le funzioni di attivazione nascoste.
- K ovvero il numero delle unità di attivazione in output:

$$a_k = \sum_{i=1}^{M} w_{ki}^{(2)} x_i + w_{k0}^{(2)} \tag{3.16}$$

  dove $k = 1, \ldots, K$.
- Indichiamo con $y_k = \sigma(a_k)$ la funzione di attivazione dell'ultimo strato.

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} \cdot h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{3.17}$$

I singoli *percettroni* ci permettono di costruire delle frontiere di separazione lineari, questo però limita l'applicabilità di questa tecnica. Nel caso in cui si ha la volontà di costruire frontiere di separazione non lineari è necessario combinare più neuroni u_i, anche posizionati in strati intermedi tra input e output, andando a formare una rete.

Per questa modifica si passa dalla funzione di attivazione a gradino a una funzione di attivazione derivabile, ovvero la sigmoide:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.18}$$

Questa scelta viene fatta per semplificare la fase di aggiornamento dei pesi.

A questo punto, per ogni configurazione in ingresso al primo strato x, la rete calcola una configurazione y dell'ultimo strato.

L'obiettivo è fissata una mappa f tra configurazioni di ingresso e di uscita, sulla base di una sequenza di stimoli x_k, la rete cambia i pesi w_{ij} in modo che, dopo un numero finito s di passi l'uscita y_k coincida con $f(x_k)$ per ogni $k > s$, almeno approssimativamente. Il criterio di apprendimento della rete consiste nel minimizzare la discrepanza tra il target atteso e la previsione della rete.

Nella fase di addestramento l'obiettivo è quello di determinare i pesi w da assegnare alle connessioni della rete partendo da un training set. La procedura di apprendimento è composta da due step:

1. Valutare il gradiente dell'errore $\nabla E(w)$ rispetto ai pesi $w_1, \ldots, w_T$.
2. Usare il vettore di derivate ottenuto per calcolare i nuovi pesi.

$$w^{(\tau+1)} = w^{(\tau)} - \eta \cdot \nabla E(w^{(\tau)}) \tag{3.19}$$

Usando un **Computational Graph** possiamo rappresentare una funzione composta come un grafo aciclico orientato dove ogni nodo rappresenta una funzione o un operazione e ogni arco rappresenta l'input per il nodo.

Possiamo calcolare la derivata di una funzione composta $f(g(h(x)))$ come:

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx} \tag{3.20}$$

3.4.1 Backpropagation

Quando si lavora con le reti l'aggiornamento dei pesi risulta più complicato rispetto a quando si lavora con il singolo neurone. Infatti, nel caso delle reti l'errore lo sappiamo solamente alla fine. Risulta quindi difficile il processo di aggiornare gli stati intermedi.

Quando si lavora con più strati, l'aggiornamento dei pesi viene ottenuto per mezzo di una derivata della funzione composta.

Abbiamo quindi l'algoritmo di **Backpropagation** che si divide in 5 passi:

1. **Input**: al neurone di input u_j viene assegnato lo stato x_j.
2. **Propagazione**: si calcola lo stato dei neuroni nascosti o di output u_j:
$$s_j = f_j(n_j) \tag{3.21}$$
dove n_j è l'input netto al neurone u_j e si calcola come:
$$n_j = \sum_{i=0}^{n} w_{ij} \cdot s_i \tag{3.22}$$
3. **Confronto**: per ogni neurone di output u_j, noto l'output atteso t_j, si calcola:
$$\delta_j = f'_j(n_j) \cdot (t_j - y_j) \tag{3.23}$$
Possiamo vedere la propagazione in avanti nella rete si ottiene come moltiplicazione tra matrici. In particolare, ogni strato è in funzione dello strato precedente.
4. **Retropropagazione dell'errore**: per ogni neurone nascosto u_j, si calcola:
$$\delta_j = f'_j(n_j) \cdot \left(\sum_h w_{jh} \cdot \delta_h \right) \tag{3.24}$$
5. **Aggiornamento dei pesi**: si ha:
$$w_{ij} = w_{ij} + \eta \cdot \delta_i \cdot s_j \tag{3.25}$$

Vediamo quindi una possibile implementazione dell'algoritmo:

Algorithm 3 Algoritmo di Backpropagation

function BACKPROPAGATION
inizializzo ogni Δw_i *ad un valore piccolo casuale*
while *non raggiungimento della condizione di terminazione* **do**
for *ogni esempio* $\langle (x_1, \ldots, x_n), t \rangle$ **do**
immetto l'input $(x_1, \ldots x_n)$ *nella rete e calcolo* y_k
for *ogni unità di output* k **do**

$$\delta_k = y_k \cdot (1 - y_k) \cdot (t_k - y_k)$$

end for
for *ogni unità nascosta* h **do**

$$\delta_h = y_h \cdot (1 - y_h) \cdot \sum_k w_{hk} \delta_k$$

end for
for *ogni peso* w_{ij} **do**

$$w_{ij} = w_{ij} + \Delta w_{ij}, \text{ con } \Delta w_{ij} = \eta \cdot \delta_j \cdot x_{ij}$$

end for
end for
end while
aggiorno i pesi:

$$w_i = w_i + \Delta w_i$$

end function

Si trova però un minimo locale e non globale e spesso include termini di *momento* per cambiare la formula dell'aggiornamento dei pesi del tipo:

$$\delta w_{ij}(n) = \eta \cdot \delta_j \cdot x_{ij} + \alpha \cdot \Delta w_{ij}(n-1) \tag{3.26}$$

in modo da avere una sorta di inerzia per la variazione dei pesi. Esistono tecniche che permettono di "uscire" dai minimi locali.

Si minimizzano gli errori sugli esempi di training ma si rischia l'overfitting.

Tutti questi fattori comportano un addestramento lento, ma dopo l'addestramento si ha una rete veloce.

3.4.2 Overfitting

Più la rete è ricca di neuroni e pesi e più può imparare semplicemente "fotografando" nella sua memoria le istanze, questo porta a una situazione di overfitting.

Un idea per limitarlo consiste nel penalizzare la rete durante il training, ovvero costringo la rete a generalizzare cogliendo regolarità nelle istanze, invece di "fotografarle". In generale queste sono dette tecniche di **regolarizzione**.

Esistono diverse tecniche per limitare l'overfitting, una di queste è chiamata **dropout**. Questa tecnica consiste nello spegnere un determinato numero di neuroni in modo da obbligare la rete a generalizzare.

3.4.3 Utilizzi

Uno degli utilizzi delle reti neurali è quello di ridurre la dimensionalità dei dati mantenendo le informazioni essenziali. Questo risultato può essere ottenuto anche con tecniche che non richiedono l'uso di reti neurali, ad esempio utilizzando una tecnica nota come **PCA** (*Principal Component Analysis*). L'idea dietro questa tecnica consiste nel trovare una trasformazione lineare delle caratteristiche che massimizza la varianza o minimizza l'errore quadratico di ricostruzione.

Gli **Autoencoder** sono delle reti neurali che mi permettono di codificare gli input utilizzando poche informazioni. Questo particolare modello di rete neurale è unsupervised e viene addestrata in modo che l'input sia uguale all'output. Per riuscire a ridurre la codifica dell'input la rete è composta da due parti che sono collegate tra di loro una dopo l'altra:

- **Encoder**: si occupa di prendere l'input e ridurlo di dimensionalità.
- **Decoder**: partendo dalla codifica in pochi bit ricostruisce l'input originale.

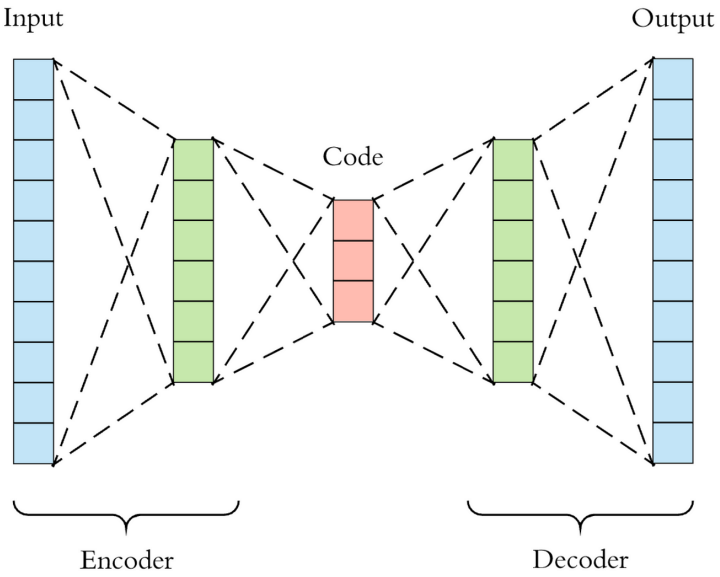

Figura 3.3: Struttura di un Autoencoder

Oltre a questo si cerca di trasformare features categoriche in features numeriche in modo da semplificare i processi di apprendimento.

3.4.4 Limiti

Si hanno quindi i seguenti limiti:

- Mancanza di teoremi generali di convergenza.
- Può portare in minimi locali di E.
- Difficoltà per la scelta dei parametri.
- Scarsa capacità di generalizzazione.

Si possono però avere varianti che permettono di migliorare il modello tramite:

- Un tasso di apprendimento adattivo: $\eta = g(\nabla E)$
- Range degli stati da -1 a 1.
- L'uso di termini di momento.
- Deviazioni dalla discesa più ripida.
- Variazioni nell'architettura (numero di strati nascosti)
- Inserimento di connessioni all'indietro.

Osservazione 2. *Rispetto alberi decisionali si ha:*

- *Le reti neurali sono più lente in fase di apprendimento ma uguali in fase di esecuzione.*
- *Le reti neurali hanno una migliore tolleranza del rumore.*
- *Le reti neurali risultano più complesse da capire.*

Si nota che ne le reti neurali ne gli alberi decisionali possono usare della conoscenza a priori.

Capitolo 4

Support Vector Machines

Precedentemente abbiamo introdotto:

- **percettrone semplice**: è un algoritmo di apprendimento efficiente solo per funzioni di separazione lineare
- **percettrone multistrato**: è un algoritmo difficile da addestrare perché deve aggiornare molti pesi e ci sono molti minimi locali, apprende solo funzioni non lineari complesse

Le **Support Vector Machines** (SVM) sono un algoritmo efficiente per apprendere funzioni di separazione non lineari complesse.

Anche nel caso delle SVM viene ripreso comunque il concetto di separazione lineare ma con una scelta dell'iperpiano migliore. Si usa la *teoria statistica dell'apprendimento*, utilizzando la **programmazione matematica**, la quale dice che tra tutti gli iperpiani che possiamo usare per separare due classi, si sceglie quello che sia in grado di etichettare meglio nel futuro. L'intuizione è quella di prendere un iperpiano ottimo rispetto alla misura della distanza minima che si ha tra gli esempi. Si guardano quindi tutti i punti del training set e cerco di piazzare in mezzo l'iperpiano, in modo che intorno ad esso ci sia massima ampiezza. Tale ampiezza è detta **margine**.

Definizione 8 (**Margine**)**.** *Il margine è la distanza tra i vettori di supporto e l'iperpiano.*

Se riuscissimo a separare i dati con un largo margine avremmo ragione di credere che il classificatore sia "più robusto" tanto da avere una migliore generalizzazione. Quando arriva quindi un nuovo punto, generato con la stessa regola degli altri, *sarà sicuramente* classificato correttamente, una volta scelto l'iperpiano.

Ci serve quindi la separabilità delle istanze, serve quindi che la funzione generatrice sia linearmente separabile.

Teorema 5 (**Dimensione di Vapnik-Cervonenkis**)**.** *Con la teoria statistica dell'apprendimento si dimostra che più allarghiamo il margine meglio l'iperpiano generalizza, raggiungendo la dimensione di Vapnik-Cervonenkis (VC). Prese tutte le funzioni che generano il training set si produce la Vapnik-Cervonenkis che esprime quanto è difficile sbagliare sulle ipotesi future in base alla scelta dell'iperpiano.*

Dobbiamo quindi scrivere un algoritmo per trovare l'iperpiano di separazione di massimo margine. In input si hanno le istanze etichettate e in output un vettore che identifica l'iperpiano. Viene usata una notazione matematica, la quale prevede che, preso un insieme di punti di training:

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \tag{4.1}$$

dove ad ogni vettore x_i associo la classe di appartenenza y_i:

$$y_i \in \{-1, +1\} \tag{4.2}$$

ho i punti linearmente separabili nel seguente modo:

$$\begin{cases} \langle w, x_i \rangle + b > 0 & \text{se } y_i = +1 \\ \langle w, x_i \rangle + b < 0 & \text{se } y_i = -1 \end{cases} \tag{4.3}$$

questo può essere riassunto in un solo vincolo nel seguente modo:

$$y_i(\langle w, x_i \rangle + b) > 0, \ i = 1, \ldots, n \tag{4.4}$$

Si ha che il vettore w mi dirà l'inclinazione del piano mentre b è la distanza tra l'origine e il piano, queste due variabili identificano i vari iperpiani possibili tra cui cercare il migliore.

L'ipotesi quindi tra w e b è una funzione che prende il segno di $\langle w, x\rangle + b$ per associare l'etichetta:

$$h_{w,b}(x) = sgn(\langle w, x\rangle + b) \tag{4.5}$$

Siano d_- e d_+ le distanze tra l'iperpiano separatore e il punto positivo e negativo più vicino, allora definiamo:

- **Margine funzionale**.
- **Margine geometrico**.

Definizione 9 (**Margine funzionale**). *Definiamo il* ***margine funzionale*** *di un punto (x_i, y_i) rispetto all'iperpiano (w, b) come:*

$$\hat{\gamma}_i = y_i(\langle w, x_i\rangle + b) \tag{4.6}$$

e quindi il margine funzionale dell'iperpiano rispetto al training set S è definito come:

$$\hat{\rho} = \min_{i=1,\dots,n} \hat{\gamma}_i \tag{4.7}$$

Si hanno quindi due casistiche:

1. Se si ha un punto x_i tale che $y_i = +1$, perché il margine funzionale sia grande è necessario che la quantità $\langle w, x_i\rangle + b$ abbia un grande valore positivo.
2. Se si ha un punto x_i tale che $y_i = -1$, perché il margine funzionale sia grande è necessario che la quantità $\langle w, x_i\rangle + b$ abbia un grande valore negativo

Il margine funzionale specifica quanta è buona la classificazione ma non non misura la distanza degli esempi dall'iperpiano.

Teorema 6. *Se $\hat{\gamma}_i > 0$, per ogni $classificazione_i$ la classificazione è approvata in quanto le classi sono linearmente separabili e l'iperpiano (w, b) separa effettivamente le classi*

Si ha quindi che un ampio margine funzionale fornisce una maggior qualità di previsione, anche, se l'uso del solo $\hat{\gamma}$ può essere problematico, in quanto il margine funzionale non è invariante rispetto ad un iperpiano ri-scalato.

Per come è stato scelto il classificatore f, se si scala l'iperpiano:

$$(w, b) \to (c \cdot w, c \cdot b) \tag{4.8}$$

si ottengono:

- Lo stesso iperpiano, ovvero lo stesso luogo di punti.
- La stessa funzione di decisione, visto che quest'ultima dipende solo dal segno di $\langle w, x\rangle + b$, con il segno che può essere invertito se $c < 0$.

ma dato che il margine funzionale viene moltiplicato per c, non possiamo usarlo come distanza di un punto dall'iperpiano, perché non è **invariante** rispetto alla scala. Bisogna studiare la distanza d del punto x dall'iperpiano. Si ha quindi:

$$d = \frac{\sum_{i=1}^{n} w_i \cdot x_i + b}{\|w\|} = \frac{\langle w, x_i\rangle + b}{\|w\|} \tag{4.9}$$

Definizione 10 (**Margine geometrico**). *Definiamo il* ***margine geometrico*** *di un punto (x_i, y_i) rispetto all'iperpiano (w, b) come:*

$$\gamma = \frac{y_i(\langle w, x_i\rangle + b)}{\|w\|} \tag{4.10}$$

e quindi il margine geometrico dell'iperpiano rispetto al training set S è definito come:

$$\rho = \min_{i=1,\dots,n} \gamma_i \tag{4.11}$$

A differenza del margine funzionale, il margine geometrico ci specifica quanto sono distanti i punti dall'iperpiano.

Teorema 7. *Se $\gamma_i > 0$, per ogni $classificazione_i$ la classificazione è approvata analogamente a quanto detto per il margine funzionale.*

Dato un punto positivo/negativo il margine geometrico rappresenta la sua distanza geometrica dall'iperpiano, il margine geometrico rende meglio l'idea della distanza di un punto da un iperpiano in $\mathbb{R}^n$.

Inoltre, si ha che il margine geometrico è invariante rispetto alla scala di w e quindi possiamo scalare l'iperpiano senza cambiamenti della distanza dei punti rispento all'iperpiano, in questo modo il margine rimane invariato. Al contrario il margine funzionale è variante perché cambiando la scala di w allora cambiava anche il valore del margine, cambiando la semantica della classificazione, ciò comporta errori.

Dalla formula notiamo che, posto $\|w\| = 1$ si scala l'iperpiano (w, b):

$$(w, b) \to \left(\frac{w}{\|w\|}, \frac{b}{\|w\|} \right) \tag{4.12}$$

Si considera quindi un iperpiano $\left(\frac{w}{\|w\|}, \frac{b}{\|w\|} \right)$ con il vettore di pesi $\frac{w}{\|w\|}$ di norma unitaria.

Definizione 11 (**Iperpiano canonico**). *Definiamo un **iperpiano canonico** se:*

$$\min_{i=1,\dots,n} |\langle w, x_i \rangle + b| = 1 \tag{4.13}$$

e quindi, per un iperpiano canonico si hanno:

- *margine funzionale pari a* 1
- *margine geometrico pari a* $\frac{1}{\|w\|}$

Si nota che se $\|w\| = 1$ allora margine funzionale e geometrico coincidono, infatti si possono mettere in correlazione:

$$\gamma = \frac{\hat{\gamma}}{\|w\|} \tag{4.14}$$

Bisogna quindi cercare di estendere il margine, cerchiamo quindi di massimizzare una certa funzione obiettivo:

$$\begin{aligned} &\max f(x) \\ \text{s.t. } &g(x) \leq 0 \\ &h(x) = 0 \end{aligned}$$

Vorremmo assicurarci che tutti i punti cadano al di fuori del margine e quindi, dato un certo γ vogliamo, $\forall i \in \{1, \dots, n\}$:

$$\frac{y_i \cdot (\langle w, x_i \rangle + b)}{\|w\|} \geq \gamma \tag{4.15}$$

e quindi vogliamo:

$$\begin{aligned} &\max \gamma \\ \text{s.t. } &y_i(\langle w, x_i \rangle + b) \leq \gamma \\ &\|w\| = 1 \end{aligned}$$

avendo, con il secondo vincolo, margine geometrico uguale a quello funzionale. Riscriviamo quindi:

$$\begin{aligned} &\max \frac{\hat{\gamma}}{\|w\|} \\ \text{s.t. } &y_i(\langle w, x_i \rangle + b) \leq \gamma \end{aligned}$$

Non avendo comunque un vincolo convesso.

Possiamo però scalare l'iperpiano senza variazioni, grazie all'invarianza. Lo scalo quindi in modo da avere l'iperpiano canonico, con margine funzionale pari a 1:

$$\begin{aligned} &\max \frac{1}{\|w\|} \\ \text{s.t. } &y_i(\langle w, x_i \rangle + b) \leq \gamma \end{aligned}$$

Si cerca quindi di rendere massimo $\frac{1}{\|w\|}$ che equivale a rendere minimo $\frac{1}{2}\|w\|^2$. Quindi si ottiene che vogliamo minimizzare $\frac{1}{2}\|w\|^2$, che per comodità chiamiamo $\tau(w)$:

$$\begin{aligned} &\max \tau(w) = \frac{1}{2}\|w\|^2 \\ \text{s.t. } &y_i(\langle w, x_i \rangle + b) \leq \gamma \end{aligned}$$

Teorema 8. *Si dimostra che esiste una sola soluzione al problema, ovvero esiste un unico iperpiano di massimo margine.*

Ricapitolando si hanno due ragioni a supporto delle SVM:

1. La generalizzazione, ovvero la capacità dell'iperpiano di separazione di massimo margine.
2. Esiste un'unica soluzione del problema di ottimizzazione appena descritto.

Si sta cercando:

$$\max \tau(w) = \frac{1}{2}\|w\|^2$$
$$\text{s.t. } y_i(\langle w, x_i\rangle + b) \leq \gamma$$

e si ha che:

$$w = \sum_{i \in Q} \alpha_i \cdot x_i \tag{4.16}$$

ovvero scritta in termini di un sottoinsieme di esempi del training set che giacciono sul margine dell'iperpiano, anche noto come **vettori di supporto**, prendendo una somma pesata dei contenuti dei vettori.

Il passaggio finale è che, se ho $\langle w, x\rangle + b$ che indicano cosa ho appreso, se ricevo un vettore x da classificare posso determinare l'etichetta:

$$sgn(\langle w, x\rangle + b) = sgn\left(\left\langle \sum_{i \in Q} \alpha_i \cdot x_i, x \right\rangle + b\right) = sgn\left(\sum_{i \in Q} \alpha_i \cdot \langle x_i, x\rangle + b\right) \tag{4.17}$$

Quindi la funzione di decisione associata alla soluzione può essere scritta in termini del prodotto interno tra i vettori di supporto (support vector) x_i e il vettore da classificare x.

4.1 Punti non sono linearmente separabili

Quando si lavora con punti non lineare separabili si usa lo stesso approccio, rivedendo la formulazione del problema tramite i **metodi kernel**. Si cerca di mappare lo spazio di input in un nuovo spazio che è a dimensione maggiore in cui i punti siano linearmente separabili, in questo modo è possibile classificare anche esempi non separabili linearmente.

Per effettuare questa trasformazione, dobbiamo trovare una funzione che prende il nome di **funzione di trasformazione**:

$$\Phi : \mathbb{R}^n \to \mathbb{R}^m, \text{ con } m > n \tag{4.18}$$

che mappi i dati iniziali non linearmente separabili in uno spazio di dimensione superiore in cui siano linearmente separabili.

In questo nuovo spazio la funzione di decisione che classifica l'input x è:

$$sgn\left(\sum_{i \in Q} \alpha_i \cdot \langle \Phi(x_i), \Phi(x)\rangle + b\right) \tag{4.19}$$

Il calcolo delle funzioni di trasformazione Φ è generalmente computazionalmente pesante ma è semplificato nel caso di funzioni kernel.

Definizione 12 (**Funzione kernel**)**.** *Data una trasformazione $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ una **funzione kernel** è una mappa:*

$$K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \text{ tale che } K(x, y) = \Phi(x) \cdot \Phi(y) \tag{4.20}$$

Definizione 13 (**Kernel trick**)**.** *Si definisce quindi il **kernel trick** che consiste nel computare il prodotto interno del trasformare di due vettori x e y, tramite Φ, senza computare le trasformate, semplificando il calcolo di:*

$$sgn\left(\sum_{i \in Q} \alpha_i \cdot \langle \Phi(x_i), \Phi(x)\rangle + b\right) \tag{4.21}$$

sostituendo $\Phi(x_i) \cdot \Phi(x)$ con $K(x_i, x)$, ottenendo:

$$sgn\left(\sum_{i \in Q} \alpha_i \cdot K(x_i, x) + b\right) \tag{4.22}$$

Non si cercano tutte le possibili trasformazioni ma solo alcune.

Esempio 1. *Sia $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ la funzione di trasformazione definita nel seguente modo:*

$$\phi(x, y) = (x^2, y^2, \sqrt{2xy}) \tag{4.23}$$

avremo che $p_1 = (x_1, x_2), p_2 = (x_2, y_2) \in \mathbb{R}^2$, allora ho $K(p_1, p_2) = \langle \Phi(p_1), \Phi(p_2) \rangle \in \mathbb{R}^3$, più precisamente:

$$K(p_1, p_2) = \langle \begin{pmatrix} x_1^2 \\ y_1^2 \\ \sqrt{2x_1y_1} \end{pmatrix}, \begin{pmatrix} x_2^2 \\ y_2^2 \\ \sqrt{2x_2y_2} \end{pmatrix} \rangle = (x_1x_2)^2 + (y_1y_2)^2 + 2\sqrt{x_1y_1x_2y_2}$$

Si hanno alcune proprietà:

- Se i dati sono mappati in uno spazio di dimensioni sufficientemente elevate, saranno quasi sempre linearmente separabili
- Quattro dimensioni sono sufficienti per separare linearmente un cerchio in qualsiasi punto del piano
- Cinque dimensioni sono sufficienti per separare linearmente qualsiasi ellisse
- Se abbiamo N esempi sono sempre separabili in spazi di dimensioni $N - 1$ o più
- Calcolare $K(x, y)$ può essere molto economico anche se $\Phi(x)$ è molto costoso, ad esempio con vettori di dimensione elevata e, in tali casi (che vanno dimostrati), si addestrano le SVM nello spazio di dimensionalità maggiore senza mai dover trovare o rappresentare esplicitamente i vettori $\Phi(x)$

Vediamo una lista di kernel standard:

- **Lineare**: $K(x, y) = x \cdot y$
- **Polinomiale**: $K(x, y) = (1 + x \cdot y)^d$
- **Radial basis function**: $K(x, y) = e^{-\gamma \|x-y\|^2}$
- **Gaussian radial basis function**: $K(x, y) = e^{\frac{-(x-y)^2}{2 \cdot \sigma^2}}$
- **Percettrone multistrato**: $K(x, y) = \tanh(b(x \cdot y) - c)$

Teorema 9. *Un kernel definisce una matrice $K_{i,j}$ che è simmetrica e definita positiva.*

Teorema 10 (**Teorema di Mercer**)**.** *Ogni matrice simmetrica e definita positiva è un kernel.*

SVM può essere applicato a problemi non binari tramite *one vs rest*, ovvero consiste nell'effettuare, per ogni classe, un'esecuzione di SVM tra la singola classe contro tutte le altre. In aggiunta, si combinano tutti gli iperpiani trovati tra di loro per effettuare la classificazione multiclasse.

Si ha quindi l'addestramento di un singolo classificatore per classe, con i campioni di quella classe considerati come positivi e tutti gli altri campioni come negativi.

Questa strategia richiede che i classificatori di base producano un punteggio di confidenza a valore reale per la sua decisione, piuttosto che solo un'etichetta di classe; infatti, le sole etichette di classi discrete possono portare a ambiguità. Abbiamo quindi:

- Come input:
  - Un learner L
  - Un set di esempi X
  - Delle label y_i associate ad ogni $x_i \in X$ con $i = 1, \ldots, K$
- Come output una lista di classificatori f_k con $l = 1, \ldots, K$

La procedura è quindi, $\forall k \in \{1 \dots K\}$ costruisco un nuovo vettore di label z tale che:

$$\begin{cases} z_i = y_i & \text{se } y_i = k \\ z_i = 0 & \text{altrimenti} \end{cases} \tag{4.24}$$

applicando poi L a (x, z) per ottenere f_k.

Quindi prendere decisioni significa applicare tutti i classificatori ad un nuovo esempio e prevedere l'etichetta k per la quale il classificatore corrispondente riporta il punteggio di confidenza più alto:

$$\hat{y} = argmax f_k(x),\ k \in \{1, \dots K\} \tag{4.25}$$

Questa euristica soffre di diversi problemi:

- La scala dei valori di confidenza può differire tra vari classificatori binari
- Anche se la distribuzione in classe è bilanciata nel training set, i learner per la classificazione binaria vedono distribuzioni sbilanciate perché tipicamente l'insieme di negativi che vedono è molto più grande dell'insieme di positivi.

Un'alternativa è *one vs one* prendendo le varie classi e produrre sistemi di SVM tra coppe di classe. La quantità di problemi derivati esplode in modo quadratico. Alla fine, si attribuisce maggior probabilità ad una singola classe. Si ha il train di:

$$\frac{K \cdot (K-1)}{2} \tag{4.26}$$

classificatori binari per un problema a K classi e ognuno riceve i campioni di un paio di classi dal training set originale e deve imparare a distinguere queste due classi.

In fase di predizione tutti i $\frac{K \cdot (K-1)}{2}$ classificatori sono applicati al nuovo sample e la classe che con il più alto numero di predizioni positive viene usata come previsione per il classificatore combinato. Anche questa tecnica soffre di ambiguità in quanto alcune regioni del suo spazio di input possono ricevere lo stesso numero di voti.

Capitolo 5

Apprendimento Bayesiano

Ci sono stati 2 modi per rappresentare la probabilità:

- **soggettivo** (grado di conoscenza o probabilità Bayesiana): la probabilità di un evento sapendo che è accaduto un altro evento.
- **oggettivo** (Long-term Frequency): la probabilità di un evento è il limite della sua frequenza relativa in molti tentativi.

La probabilità in termini di limite sulle frequenze a lungo termine dei casi, questo è un'interpretazione oggettiva perché si sfruttano delle formule e poi abbiamo dei casi immutabili. Mi baso su parametri fissati e quindi calcolo la probabilità in base ai parametri.

L'**apprendimento bayesiano** è un approccio all'apprendimento automatico che si basa sulla probabilità bayesiana. L'idea è quella di utilizzare la probabilità per rappresentare la conoscenza incerta e quindi aggiornare la probabilità in base ai nuovi dati. Quindi non cerchiamo l'ipotesi che combacia con i dati ma cerchiamo l'ipotesi più probabile.

L'apprendimento bayesiano è importante per due motivi:

1. Si ha una manipolazione esplicita della probabilità rispetto ad altri approcci pratici di alcuni tipi di problemi di apprendimento.
2. Fornisce una prospettiva utile per comprendere metodi di apprendimento che non manipolano esplicitamente la probabilità.

Le regole del calcolo mi permettono di associare un numero a una probabilità ma non un significato. La probabilità bayesiana è una probabilità che ha un significato.

Dal punto di vista delle funzionalità si ha che ogni esempio di training osservato può aumentare o diminuire la probabilità di un'ipotesi. Inoltre, la conoscenza pregressa può essere combinata con i dati di training per ottenere la probabilità finale delle varie ipotesi.

Si hanno alcune difficoltà nell'apprendimento bayesiano:

- La **probabilità bayesiana** è una probabilità **soggettiva**, quindi dipende dalla conoscenza pregressa.
- Si hanno costi **computazionali elevati**, perché si ha una complessità esponenziale rispetto al numero di variabili.

La statistica bayesiana è una probabilità soggettiva. Si ha una natura soggettiva perché si ha una conoscenza incerta a cui viene associata una probabilità. Si ha quindi una variabile casuale a cui associo una distribuzione di probabilità. Lo stato delle cose è rappresentato da una variabile soggetta a probabilità. Il parametro non è più certo ma è casuale.

Nota 1. *Nell'apprendimento bayesiano la* ***miglior ipotesi*** *è quella che* ***massimizza*** *la probabilità.*

Il punto centrale dell'apprendimento bayesiano è la **regola di Bayes**, la quale fornisce un metodo diretto per calcolare la probabilità di un'ipotesi h dato un insieme di dati D. La regola di Bayes è la seguente:

$$P(h|D) = \frac{P(D|h) \cdot P(h)}{P(D)} \tag{5.1}$$

dove:

- $P(h|D)$ è la probabilità dell'ipotesi h dato il dataset D (**posterior**), è la probabilità che l'ipotesi dopo aver osservato l'evidenza.
- $P(D|h)$ è la probabilità del dataset D dato l'ipotesi h (**likelihood**), è la probabilità di osservare l'evidenza D dato che l'ipotesi h è vera.
- $P(h)$ è la probabilità dell'ipotesi h (**prior**), ovvero la probabilità dell'ipotesi prima di osservare l'evidenza. È un informazione a priori, il grado di fiducia rispetto all'ipotesi prima di osservare l'evidenza. Prima si fissa la prior e poi si osserva l'evidenza.
- $P(D)$ è la probabilità del dataset D (**evidence**), ovvero la probabilità di osservare l'evidenza D. È la probabilità di come si comporta l'evidenza senza considerare l'ipotesi. L'evidenza è il dataset e quindi al distribuzione dei dati.

  Quando ho due eventi indipendenti la probabilità congiunta è il prodotto delle due probabilità:

$$P(A, B) = P(A) \cdot P(B) \tag{5.2}$$

  Per calcolare la probabilità di evidenza $P(D)$ si utilizza la seguente formula:

$$P(D) = \sum_{h \in H} P(D, h) \cdot P(h) \tag{5.3}$$

Con la formula di Bayes si può calcolare la probabilità a posteriori. L'ipotesi che meglio si adatta ai dati è quella che ha la probabilità a posteriori più alta. Viene anche definita come **ipotesi MAP** (*Maximum A Posteriori*).

$$h_{MAP} = \text{argmax}_{h \in H} P(h|D) = \text{argmax}_{h \in H} \frac{P(D|h) \cdot P(h)}{P(D)} = \text{argmax}_{h \in H} P(D|h) \cdot P(h) \tag{5.4}$$

dove D rappresenta il dataset, argmax è l'ipotesi h che massimizza la probabilità a posteriori, si esclude dai conti $P(D)$ perché non stiamo cercando la probabilità di $P(h|D)$, ma bensì stiamo cercardo quella che tra tutte ha probabilità massima, di conseguenzza $P(D)$ non modifica i risultati.

Spesso si assume che tutte le ipotesi siano equiprobabili (distribuzione uniforme), quindi si ha la possibilità di semplificare i conti, perché, con lo stesso procedimento che abbiamo utilizzato sopra, possiamo escludere dal conto anche $P(h)$. In tal caso, la ricerca dell'**ipotesi MAP** prenderà il nome di ricerca dell'**ipotesi ML** (*Maximum Likelihood*), perché prenderemo h:

$$h_{ML} = h_{MAP} = \text{argmax}_{h \in H} P(h|D) = \text{argmax}_{h \in H} P(D|h) \tag{5.5}$$

potendo quindi trascurare $P(h)$ in quanto equivalente per tutte le ipotesi.

Si può utilizzare la formula di Bayes per specificare un algoritmo di apprendimento molto semplice detto **algoritmo Brute-Force map learning**, che si articola nei seguenti passi:

1. Per ogni ipotesi $h \in H$ calcolo la probabilità a posteriori $P(h|D)$, utilizzato la formula di Bayes:

$$P(h|D) = \frac{P(D|h) \cdot P(h)}{P(D)} \tag{5.6}$$

2. Restituisco l'ipotesi h con probabilità a posteriori massima:

$$h_{MAP} = \text{argmax}_{h \in H} P(h|D) \tag{5.7}$$

Nota 2. *La probabilità è un conteggio. Devo capire quante combinazioni di valori posso ottenere. Nello specifico si hanno:* $\prod_{i=1}^{n} |D_i|$ *combinazioni, dove* D_i *è il dominio della variabile* X_i*.*

Il modo per semplificare l'apprendimento bayesiano è quello di usare il **naive bayes**.

Usando la formula di Bayes posso calcolare la probabilità dell'ipotesi h dato il training set D: $P(h|D)$. Tale probabilità prende il nome di **posterior**, posso calcolarla come segue:

$$P(h|D) = \frac{P(D|h) \cdot P(h)}{P(D)} \tag{5.8}$$

e non dalla tabella.

Viene indotta in maniera indiretta dalle probabilità presenti nella formula.

Supponiamo di conoscere $P(D|h)$ per conoscere *cosa* posso calcolare:

$$\sum_{i=0}^{|H|} P(D, h_i) = \sum_{i=0}^{|H|} P(D|h_i) \cdot P(h_i) \tag{5.9}$$

$$h_{MAP} = \text{argmax}_{h_i \in H} P(h_i|D) = \text{argmax}_{h_i \in H} P(D|h_i) \cdot P(h_i) \tag{5.10}$$

Posso escludere $P(D)$ dal denominatore perché è una costante, se vogliamo conoscere la posterior ci serve conoscere il denominatore.

Definizione 14 (**Indipendeza condizionata**). ***Indipendeza condizionata**:*

$$P(X, Y|Z) = P(X|Z) \cdot P(Y|Z) \tag{5.11}$$

la distribuzione è legata al fattore condizionante Z.

Questo mi permette di calcolare le singole probabilità sulle colonne del dataset e poi moltiplicarle per ottenere quella totale:

$$P(D|h) = \prod_{i=0}^{|D|} P(d_i|h) \tag{5.12}$$

dove d_i è la colonna i-esima del dataset D.

5.1 Naive Bayes

Supponiamo di avere un dataset D con n attributi $a_1, \ldots, a_n$ si cerca di semplificare il calcolo utilizzando la proprietà dell'indipendenza condizionata:

$$P(D|h) = \prod_{i=0}^{|D|} P(d_i|h) \tag{5.13}$$

Questo ci permette di definire il classificatore Bayesano naive è definito come:

$$f_{NB} = \text{argmax}_{h_i \in H} P(h_i) \cdot \prod_{i=0}^{|D|} P(d_i|h_i) \tag{5.14}$$

Algoritmo naive bayes:

1. Calcola le probabilità a priori $P(h_i)$.
2. Calcola le probabilità condizionate $P(d_i|h_i)$.
3. Calcola la probabilità a posteriori $P(h_i|D)$.
4. Restituisci la classe con probabilità a posteriori maggiore.

Esempio 2. *Dato il seguente dataset: $h_0 = (T = 0)$ allora abbiamo:*

Esempi	A	B	C	Target
x_1	1	1	1	0
x_2	0	1	1	0
x_3	1	0	1	1
x_4	0	0	0	1
x_5	0	1	0	1
x_6	1	1	0	?

$$P(T = 0|A = 1, B = 1, C = 0) = \frac{P(A = 1, B = 1, C = 0|T = 0) \cdot P(T = 0)}{P(x_6)} \tag{5.15}$$

posso togliere il denominatore perché è una costante, e quindi uso solo quelle a numeratore.

$$P(T=0|A=1,B=1,C=0) = P(A=1|T=0)\cdot P(B=1|T=0)\cdot P(C=0|T=0)\cdot P(T=0) \tag{5.16}$$

A questo punto utilizzando i valori presenti nel dataset posso effettuare i calcoli:

$$P(T=0|A=1,B=1,C=0) = \frac{1}{2}\cdot 1\cdot 0\cdot \frac{2}{5} = 0 \tag{5.17}$$

Calcolo anche il caso $h_1 = (T=1)$:

$$P(T=1|A=1,B=1,C=0) = \frac{P(A=1,B=1,C=0|T=1)\cdot P(T=1)}{P(x_6)} \tag{5.18}$$

posso togliere il denominatore perché è una costante, e quindi uso solo quelle a numeratore.

$$P(T=1|A=1,B=1,C=0) = P(A=1|T=1)\cdot P(B=1|T=1)\cdot P(C=0|T=1)\cdot P(T=1) \tag{5.19}$$

ottenendo quindi:

$$P(T=1|A=1,B=1,C=0) = \frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}\cdot\frac{3}{5} = \frac{2}{45} \tag{5.20}$$

A questo punto calcoliamo $P(x_6)$ come:

$$\begin{aligned} P(x_6) &= \sum_{i=0}^{|H|} P(x_6,h_i) = \sum_{i=0}^{|H|} P(A=1,B=1,C=0|T=i)\cdot P(T=i) \\ &= P(A=1|T=0)\cdot P(B=1|T=0)\cdot P(C=0|T=0)\cdot P(T=0) + \\ &\quad P(A=1|T=1)\cdot P(B=1|T=1)\cdot P(C=0|T=1)\cdot P(T=1) \end{aligned} \tag{5.21}$$

In generale per tutte le feature non vale l'**indipendenza condizionata**, ma è stato dimostrato che la correttezza rimane invariata.

I principali problemi del **Naive Bayes** si hanno quando non si hanno abbastanza dati per il training del modello. In questi casi risulta complesso stimare le probabilità numericamente. Inoltre, se non osserviamo nessuna feature con una particolare etichetta avremo una probabilità uguale a 0. La soluzione per quest'ultimo consiste nel aggiungere un piccolo numero di elementi in modo da non rendere nulla la probabilità ma molto vicina allo zero.

5.2 Gaussian Naive Bayes

Naive bayes stima le probabilità basandosi sulle frequenze della classe di ciascuna feature presente nel training set, quindi funziona solo nel calcolo di distribuzioni discrete. Se le feature sono continue, la probabilità di una feature può essere stimata usando la loro distribuzione continua oppure approssimando con una **distribuzione gaussiana**.

Nella realtà si assumono che tutti gli attributi derivino da distribuzioni gaussiane per semplificare i conti.

Per classificare si calcolano sempre le probabilità, più precisamente la prior si deriva dalla probabilità che una classe c venga osservata tra le etichette del dataset. La likelihood si può calcolare usando la gaussiana:

$$P(x_i|c_j) = \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}}\cdot e^{-\frac{(x_i-\mu_{i,j})^2}{2\sigma_{i,j}^2}} \tag{5.22}$$

dove $\mu_{i,j}$ rappresenta la media dell'i-esimo elemento della classe j-esima e $\sigma_{i,j}$ la deviazione standard dell'i-esimo elemento della classe j-esima, statistiche calcolate dal dataset. Dove i sono le feature e j sono le classi.

La previsione di un nuovo elemento può essere fatta come:

$$P(c_i|x) \overset{Bayes}{=} \frac{P(X|c_j)\cdot P(c_j)}{P(x)} \overset{Naive}{=} \frac{\prod_i P(x_i|c_j)\cdot P(c_j)}{P(x)} \overset{Gauss}{=} \frac{\prod_i \frac{1}{\sqrt{2\cdot\pi\cdot\sigma_{i,j}^2}}\cdot e^{-\frac{1}{2}\left(\frac{x_i-\mu_{i,j}}{\sigma_{i,j}}\right)^2}\cdot P(c_j)}{P(x)}$$

Capitolo 6

Clustering

Le tecniche di apprendimento sono:

- **supervisionato**: etichette nel dataset
- **non supervisionato**: non ho le etichette e quindi devo capire le cose comune

Spesso si può usare l'apprendimento non supervisionato al posto del supervisionato, perché potrebbe ottenere dei risultati migliori, soprattutto quando il dataset è pieno di rumore. I sistemi non supervisionati tornano comodi quando la distribuzione dei valori degli attributi è informazione sufficiente per separare le istanze in più classi, si hanno quindi domini dove questo fattore è determinante.

Il fatto di utilizzare un apprendimento non supervisionato porta a diversi **vantaggi**:

- non è richiesta alcuna conoscenza a priori quindi evito problemi legati ad eventuali etichette errate (errori umani ridotti)
- tutte le classi che hanno caratteristiche uniche vengono identificate
- efficaci con elementi di tipo numerico perché vengono rappresentate in uno spazio di punti o di ordinamento intrinseco grazie al calcolo delle distanze

Il **lato negativo** del fatto che l'apprendimento è non supervisionato:

- le classi ottenute potrebbero non avere significati
- l'utente ha poco controllo sulla procedura e sui risultati
- scarsa efficacia su elementi ordinati in modo arbitrario o parziale

Il **clustering** consiste nel separare, categorizzare, raggruppare in autonomia con un apprendimento **non supervisionato**. Per il clustering, la fase di apprendimento cambia, ma la fase predizione rimane identica perché ho trovato una regola che spiega il dataset. Il criterio per cui raggruppare è arbitrario, non si conoscono il numero di gruppi in cui partizionare il dataset e si dovrà trovare un modo per valutare il raggruppamento ottenuto. In sostanza si cerca una regola che possa spiegare come raggruppare i dati senza avere una label di riferimento e il numero di gruppi. La ricerca del raggruppamento migliore si baserà sempre sulla ricerca dell'ipotesi più semplice. Non conoscendo le label allora raggrupperemo sfruttando le **distanze**, ovvero un elemento fa parte di un raggruppamento noto se si trova vicino agli elementi di quel raggruppamento.

I dati che vengono utilizzati per effettuare il clustering sono principalmente vettori numerici, dove ogni elemento misura una specifica **caratteristica** (**feature**) dell'istanza. Per poter raggruppare questi elementi è necessario definire dei criteri di similarità tra i vettori.

Si hanno quindi due criteri da rispettare:

- **Omogeneità**: gli elementi di un cluster devono essere simili tra loro.
- **Separazione**: gli elementi di cluster diversi devono essere dissimili tra loro.

Definizione 15 (**Cluster**)**.** *Sia $N = \{e_1, \ldots, e_n\}$ l'insieme degli elementi da raggruppare e sia $C = \{c_1, \ldots, c_k\}$ una partizione di N in sottoinsiemi disgiunti C_i. Ogni sottoinsieme C_i è un* ***cluster****. Due elementi si chiamano* ***mates*** *rispetto a C se sono membri dello stesso cluster.*

Definizione 16 (**Misura di similarità**). *Possiamo definire una* ***misura di similarità*** *tra due elementi e_i e e_j utilizzando la distanza. In particolare possiamo utilizzare:*

- ***Distanza euclidea****: invariante rispetto a traslazioni e rotazioni degli assi, rappresenta la lunghezza del segmento tra i due punti.*

$$d(e_i, e_j) = \sqrt{\sum_{k=1}^{n} (e_{ik} - e_{jk})^2} \tag{6.1}$$

- ***Distanza di Manhattan****: non è invariante rispetto a traslazioni o rotazioni degli assi e pone meno enfasi sulle variabili con distanze maggiori, non elevando al quadrato le differenze.*

$$d(e_i, e_j) = \sum_{k=1}^{n} |e_{ik} - e_{jk}| \tag{6.2}$$

- ***Distanza di Minkowski****:*

$$d(e_i, e_j) = \sqrt[p]{\sum_{k=1}^{n} |e_{ik} - e_{jk}|^p} \tag{6.3}$$

  La distanza di Manhattan è un caso particolare della distanza di Minkowski con $p = 1$. La distanza euclidea è un caso particolare della distanza di Minkowski con $p = 2$. La distanza di Minkowski è un caso particolare della distanza di Minkowski con $p = \infty$.

Esistono diverse tipi di clustering, possiamo principlamente inividuare:

- **Clustering gerarchico**: si costruisce una gerarchia di cluster. Si parte da un cluster che contiene tutti gli elementi e si procede a dividere i cluster in sotto-cluster. Si può procedere in modo top-down o bottom-up. Si collocano gli elementi in input in una struttura gerarchica ad albero, in cui le distanze tra nodi riflettono le similarità degli elementi. Gli elementi sono localizzati sulle foglie dell'albero.
- **Clustering partizionale**: si costruisce una partizione di N in k cluster. Si parte da una partizione iniziale e si procede a migliorarla iterativamente. Si può procedere in modo deterministico o probabilistico. Si collocano gli elementi in input in un insieme di cluster, in cui le distanze tra i cluster riflettono le similarità degli elementi. Gli elementi sono localizzati nei cluster. I metodi non gerarchici mirano a ripartire le n unità della popolazione in k gruppi, fornendo una sola partizione anziché una successione di partizioni tipica dei metodi gerarchici

Esistono algoritmi di clustering che funzionano su grafi, in questo caso la distanza è data dall'arco. Noi ci concentriamo su algoritmi che funzionano su vettori di numeri reali.

6.1 K-Means

Definizione 17 (**Dendrogramma**). *Un* ***dendrogramma*** *è un metodo per rappresentare le informazioni come un albero o un grafo utilizzato per visualizzare la somiglianza nel processo di raggruppamento.*

L'algoritmo $k-$means ha come prerequisito quello di conoscere il numero di cluster che si vogliono andare ad individuare. L'obiettivo di questo algoritmo è quello di minimizzare le distanze tra gli elementi di un cluster e il centroide dello stesso. Spostando i centroidi si ottiene una nuova partizione.

Definizione 18 (**Centroide**). *Nel caso dell'algoritmo $k-$means il* ***centroide*** *è ottenuto come media di tutti i punti apparteneti al cluster.*

L'algoritmo del $k-$means lavora con dati numerici nel seguente modo:

1. Si fissano a caso k centroidi iniziali di altrettanti cluster.
2. Per ogni individuo si calcola la distanza da ciascun centroide e lo si assegna al più vicino.
3. Per la partizione provvisoria così ottenuta si ricalcolano i centroidi di ogni cluster (*media aritmetica* della posizione dei punti).
4. Per ogni individuo si ricalcola la distanza dai centroidi e si effettuano gli eventuali spostamenti tra cluster

5. Si ripetono le operazioni 3 e 4 finché si raggiunge il numero massimo di iterazioni impostate o non si verificano altri spostamenti.

Questo algoritmo è molto semplice da implementare e richiede un tempo di calcolo pari a: $\mathcal{O}(tKn)$ con:

- n cardinalità dell'insieme dei dati.
- K numero di cluster.
- t numero di iterazioni del ciclo (avendo quindi $kt << n$)

K-means ha diversi problemi:

- ha una sensibilità rispetto alla scelta dei centroidi iniziali, sperimentalmente si è dimostrato che brutti valori iniziali invalidano l'intero processo
- non si può predire il numero di cluster non conoscendo a priori i dati, in aggiunta, non esiste un K ottimale e non ci sono proprietà che ce lo possano suggerire. si possono però usare delle approssimazioni studiando gli iperparametri. Non sapendo a priori il numero di K posso ottenere scarsi risultati magari non avendo abbastanza centroidi. Adattarsi ad un numero "errato" di centroidi può portare a risultati "sporchi".
- l'algoritmo è sensibile rispetto alle dimensioni geometriche che hanno le istanze nello spazio, infatti, lavorando sui centroidi potrebbe classificare in modo errato questo dettaglio, non distinguendo i corretti insiemi di punti.
- l'algoritmo è sensibile anche alla densità degli esempi, infatti se non avessimo k sufficientemente grande allora si potrebbe arrivare a classificazioni errate

In ogni caso mappa sensatamente i vari elementi in base alle loro caratteristiche e quindi è un approccio parecchio usato, essendo generalmente efficacie. Per risolvere i problemi introdotti precedentemente, si può provare ad aumentare il numero di cluster, unendo poi, in un secondo passaggio, i vari cluster secondo certi criteri, magari avendo una netta separazione lineare tra cluster. Così si possono risolvere i problemi legati alla distribuzione geometrica degli esempi, però rimane sempre il problema dell'identificare le posizioni di inizializzazione dei centroidi migliori.

Per misurare le prestazioni di clustering bisogna capire come classificare i risultati. Si usa la misura di **silhouette** per capire se un clustering è migliore di un altro. Tale misura si basa sempre sul concetto di distanza.

Fissando una misura di distanza tra istanze i, j, $d(i, j)$, definiremo:

1. Distanza media **intra-cluster**, ovvero simiglianza tra elementi dello stesso cluster:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i, j) \tag{6.4}$$

2. Distanza media **inter-cluster**, ovvero simiglianza tra un elemento di un claster rispetto a tutti gli elementi tra gli altri cluster:

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j) \tag{6.5}$$

ottenendo quindi la silhouette per il punto i come:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{6.6}$$

La qualità viene quindi visualizzata da un diagramma che studia la silhouette al variare di K per ogni valore di ogni attributo (mettendo per ogni attributo in alto i valori più studiabili).

La media $s(i)$ su tutti i punti di un cluster è una misura di quanto siano strettamente raggruppati tutti i punti del cluster. Pertanto, la media $s(i)$ su tutti i dati dell'intero set di dati è una misura di quanto adeguatamente i dati sono stati raggruppati. Se ci sono troppi o troppo pochi cluster, come può accadere quando una scelta sbagliata di k viene utilizzata nell'algoritmo di clustering, alcuni dei cluster mostreranno tipicamente linee molto più strette rispetto al resto nel diagramma di silhouette. Quindi grafici e media di silhouette possono essere utilizzati per determinare il numero "naturale" di cluster all'interno di un set di dati.

Capitolo 7

Performance Evaluation

In machine learning una delle fasi più importanti è la valutazione del modello, utile per capire la sua bontà e per avere delle metodologie di confronto con gli altri.

Innanzitutto per valutare il modello non si considerano gli errori, calcolati come scarto, delle predizioni sui dati di training. Questo perché successivamente si otterranno nuovi dati completamente differenti da quelli utilizzati per la fase di training, invalidando la metrica calcolata.

La valutazione del modello permette fin da subito di rilevare eventuali comportamenti di underfitting o di overfitting. Queste dinamiche vengono illustrate nella figura 7.1, in cui si può notare che un modello all'aumentare della sua complessità si avrà tre fasi:

- **fase di underfitting**: fase in cui il modello è ancora semplice per cui gli errori sul training e sul test sono elevati. Questo significa che si è in underfitting perché il modello non fitta bene sul training e, in aggiunta, non riesce a generalizzare.
- **fase ideale**: fase in cui il modello ha una complessità adeguata per cui non è troppo semplice da andare in underfitting e non è troppo complesso da andare in overfitting. Questa fase è ideale perché si ha un buon fit sul training e una buona generalizzazione sul test set
- **fase di overfitting**: fase in cui il modello è troppo complesso quindi si fitta molto bene sul training ma non riesce a generalizzare sul test. Questo comporta errori molto piccoli sul training, con un conseguente aumento degli errori sul test.

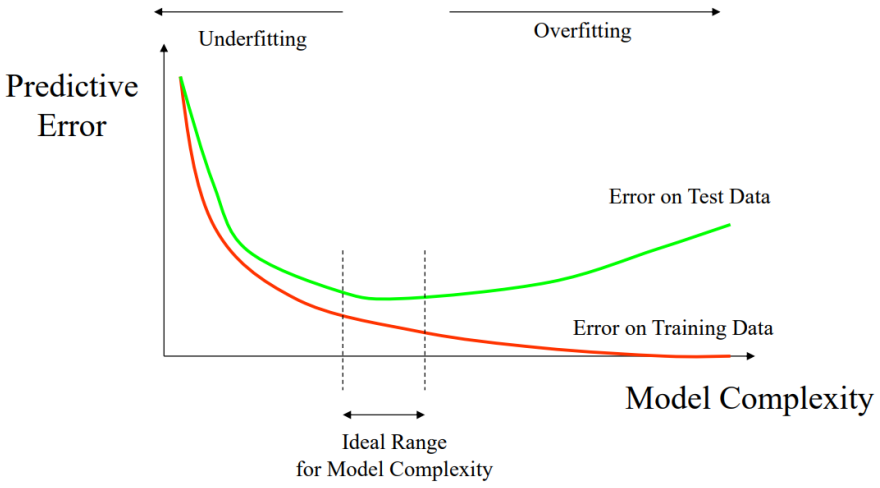

Figura 7.1: Rappresentazione dell'overfitting e underfitting

Le metriche di valutazione evidenziano tre componenti:

- bontà del modello sul train
- bontà del modello sul test

- complessità del modello (quanti parametri): spazio, tempo in termini di O

Per le metriche sarà fondamentale introdurre la **matrice di confusione**, in cui

$$T[i,j] = \#\text{ degli esempi etichettati come } i \text{ predetti come classe } j \tag{7.1}$$

Se si è nel caso di classificazione binaria allora:

- $i = 0, j = 0$, true positive, **TP**: esempi positivi predetti come positivi
- $i = 0, j = 1$, false negative, **FN**: esempi positivi predetti come negativi
- $j = 1, j = 0$, false positive, **FP**: esempi negativi predetti come positivi
- $j = 1, j = 1$, true negative, **TN**: esempi negativi predetti come negativi

L'obiettivo sarà quello di massimizzare la diagonale e minimizzare il resto.

7.1 Metriche di valutazione

Le metriche di valutazione sono differenti e si possono calcolare in modo aggregato su tutte le classi oppure si possono calcolare su una classe alla volta e in un secondo momento aggregarle secondo delle medie. Generalmente in caso di problemi di classificazione multiclasse le metriche si calcoleranno sulle singole classi e si aggregheranno utilizzando diversi metodi.

Le metriche aggregate sono:

- **Tasso di errore**: Metrica più naturale che consiste nel calcolo della proporzione degli errori su tutto l'insieme di istanze dato in input al modello. Gli errori possono essere calcolati secondo lo scarto o secondo le distanze.
- **Accuracy**: Misura la proporzione di istanze correttamente classificate su tutto l'insieme dato in input sul set.
$$Ac = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.2}$$
- **Precision**: La precision misura la proporzione di esempi positivi predetti come positivi (TP) sul numero totale di esempi predetti come positivi.
$$P = \frac{TP}{TP + FP} \tag{7.3}$$
- **Recall**: La recal misura la proporzione di esempi positivi predetti come positivi (TP) sul numero totale di esempi positivi predetti come positivi ed esempi positivi predetti come negativi.
$$R = \frac{TP}{TP + FN} \tag{7.4}$$
- **F-measure**: La F-measure calcola la media armonica della precision e della recall.
$$F = \frac{2 \cdot P \cdot R}{P + R} \tag{7.5}$$

Le ultime tre le calcoleremo sulle singole classi:

- **Precision**:
$$P(l) = \frac{\#\text{ di istanze correttamente predette come } l}{\#\text{ di istanze predette come } l} \tag{7.6}$$
- **Recall**:
$$R(l) = \frac{\#\text{ di istanze correttamente predette come } l}{\#\text{ di istanze di classe } l} \tag{7.7}$$
- **F-measure**:
$$F(l) = \frac{2 \cdot P(l) \cdot R(l)}{P(l) + R(l)} \tag{7.8}$$

Come introdotto precedentemente, si potranno aggregare secondo i seguenti metodi:

- **macro avg**: fa la media della performance tra le diverse classi delle singole metriche, utilizzata quando le classi sono di uguale importanza.

$$Perf* = \frac{1}{|L|}\sum_{l=1}^{|L|} Perf(l),\ Perf \in \{Ac, P, R, F\} \tag{7.9}$$

- **micro avg**: media pesata della performance rispetto alla cardianlità della classe, utilizzata quando le classi sono di diversa importanza.

$$Perf* = \sum_{l=1}^{|L|} \frac{|class(l)|}{\text{tot. istanze}} Perf(l),\ Perf \in \{Ac, P, R, F\} \tag{7.10}$$

(possiamo anche specificare un peso a piacere l'importante che si garantisca il range della metrica)

Il confronto dei modelli si può effettuare utilizzando le **curve ROC** che mettono a confronto $TPrate$ e $FPrate$. La curva ROC non sono altro che la valutazione del modello con diverse soglie decisionali da 0 a 1.

Definizione 19. *La **soglia decisionale** di un modello è il parametro di confidenza che specifica se classificare l'esempio secondo una classe oppure un'altra.*

Esempio 3. *Per esempio nell'algoritmo di Bayes la soglia decisionale è quella che permette di specificare sopra che probabilità classificare l'esempio come T. Di default è* 0.5.

Modificando la soglia del modello si possono avere diversi comportamenti:

- $\theta \to 1$: il modello è più conservativo quindi si migliorerà la precision, peggiorando la recall.
- $\theta \to 0$: il modello è meno conservativo, tendendo a classificare quindi si migliorerà la più facilmente come T, quinndi migliorerà la recall, ma si peggiorerà la precision.

La **curva ROC** si costruisce eseguendo i modelli con treshold diverse incrementali, al termine dell'esecuzione si ricalcola la **matrice di confusione**, per ottenere i nuovi TPR e FPR da utilizzare per disegnare il punto sulla curva associato alla soglia settata inizialmente.

Per effettuare il confronto tra modelli, disegneremo nello stesso grafico le **curve ROC** associate, permettendoci di studiare il comportamento di ciascun modello. Generalmente si aggiungono al confronto anche classificatore perfetto (curva verde) e quello **randomico** (curva rossa). Si definirà il modello migliore quello che ha la curva dominante rispetto alle altre. Quando le curve sono simili e si intersecano, allora si deve studiare l'area sottesa alla curva chiamata **area AUC**, calcolabile facilmente utilizzando le librerie.

Confrontando secondo la metrica dell'area AUC, il modello migliore è quello con la curva con AUC maggiore. L'AUC è utile per:

- **training model**: compara l'apprendimento dei modelli diversi sullo stesso training
- **training e production**: specifica una valutazione delle performance del modello facendo variare la treshold.

La metrica AUC ha delle limitazioni:

- perde di significato quando si hanno classi di diversa importanza
- quando le classi del dataset sono molto sbilanciate può essere sensibile perché una classe sarà predominante nel calcolo della metrica rispetto alle altre

Un altro modo per valutare il modello sono le **Curve di apprendimento**, sono delle curve che dicono, al variare della numerosità del campione, come si comporta il modello in termini di accuratezza. In sostanza si traina il modello su tutte le combinazioni del dataset di dimensione fissata, si calcola l'accuratezza, alla fine si computa la media (corrisponde al punto rosso) e si calcola la varianza (rappresentata dalle barre blu). Successivamente, si reitera il procedimento aumentando la dimensione del campione. Le curve specificano quanti dati sono necessari per la fase di training, per avere un modello con performance accettabili in termini di tempo di apprendimento e della performance. Il calcolo della curva è molto pesante ma spesso si costruisce parzialmente.

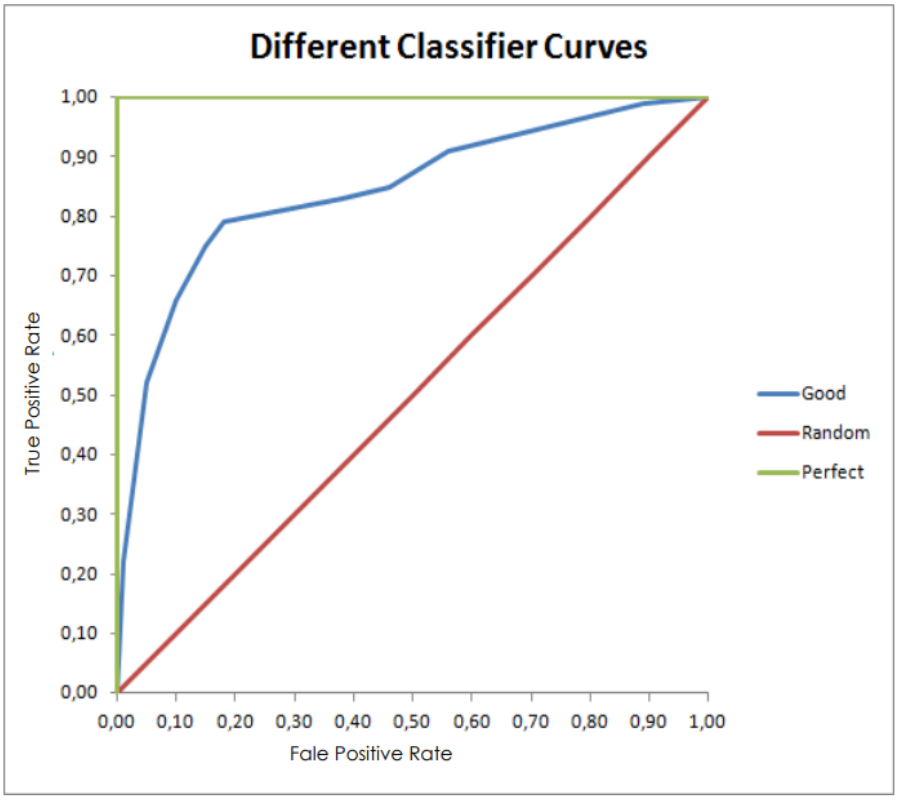

Figura 7.2: Esempio di curva di ROC

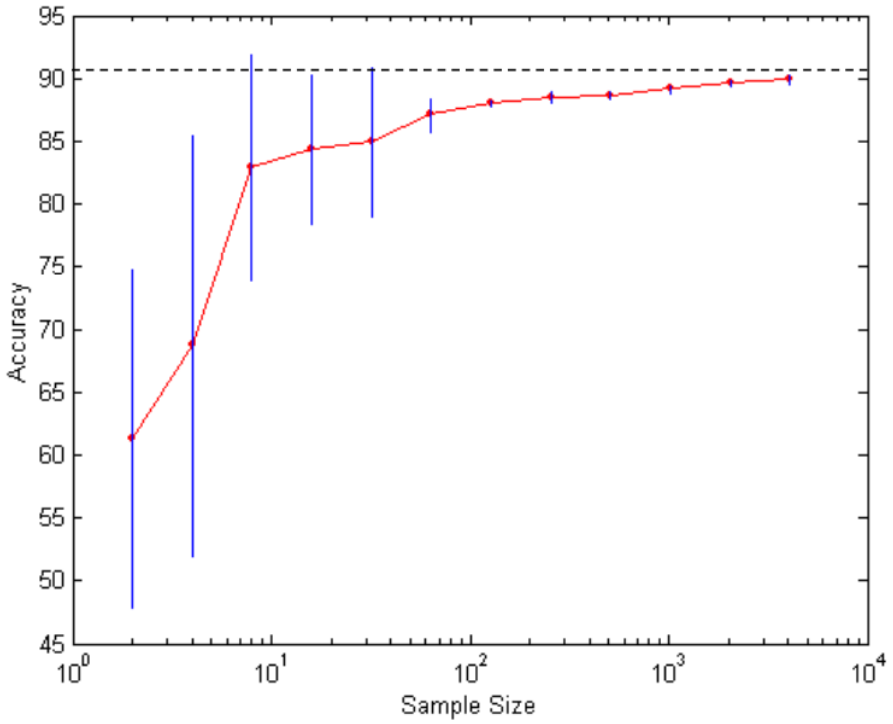

Figura 7.3: Esempio di curva di apprendimento

7.2 Valutazione del modello rispetto alla dimensionalità del dataset

La dimensionalità del dataset è importante per la valutazione del modello, innanzitutto bisogna identificare quando un dataset è di grandi dimensioni:

- **dataset di grandi dimensioni**: quando il dataset contiene almeno 10000 istanze
- **dataset di piccole dimensioni**: quando il dataset è di dimensione molto piccola, < 100 istanze

La valutazione del modello su un dataset di grandi dimensioni può essere fatta effettuando la suddivisione tra 80% train set e 20% test test randomicamente. Si allenerà il classificatore sul train set e si valuta sul test set.

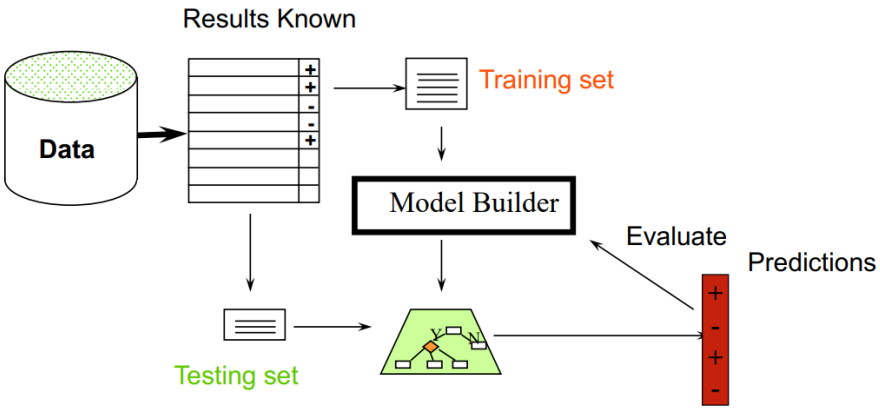

Figura 7.4: Schema di valutazione di un modello su un dataset di grosse dimensioni

Ricorda sempre che i dati di test non devono mai essere usati per il train e nemmeno per il tuning degli iperparametri. Per effettuare il tuning degli iperparametri allora si partizionerà il train set in training e validation, in modo da calocolare le metriche sul validation per tunnare gli iperparametri. Per la valutazione del modello finale si userà sempre il test set.

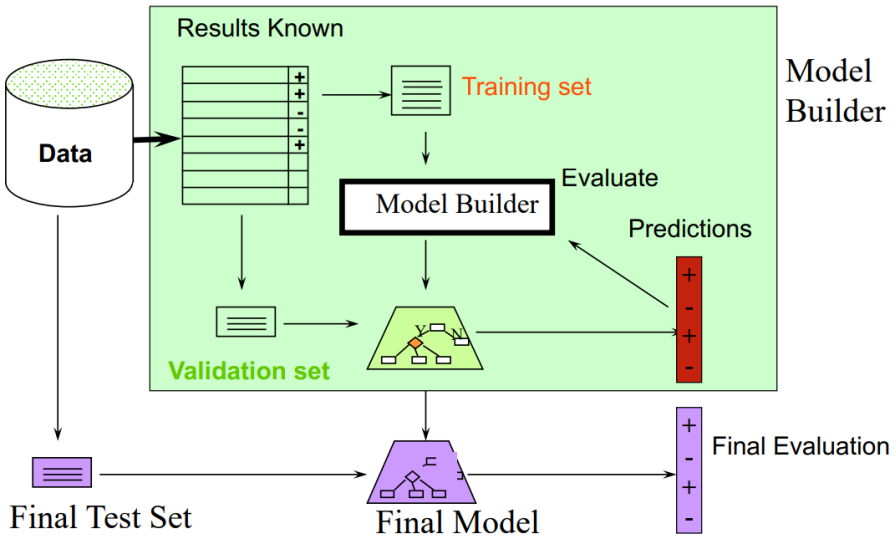

Figura 7.5: Schema di valutazione di un modello su un dataset di grosse dimensioni con suddivisione train-validation-test

Per valutare il modello su un dataset di piccole dimensioni non si effettua più la suffivisione training, validation e test set, perché i dati sono pochissimi non avremo una rappresentazione coerente della distribuzione dei dati. La soluzione è quella di utilizzare **repeted holdout**, ovvero effettuare la valutazione del modello allenandolo in più interazioni su una suddivisione dei dati che cambia ogni volta, stando attenti che in una iterazioni i sottoinsiemi di train e test siano disgiunti. Ad ogni iterazione si valuta il modello e alla fine si ha la valutazione finale aggregando le valutazioni delle singole iterazioni.

Questo ragionamento si può applicare su dataset di dimensioni medie sfruttando la **k-fold cross-validation** che consiste del suddividere i dati in k sottoinsiemi di uguale dimensione e considerare la maggior parte dei k sottoinsiemi come train mentre i rimanenti come test. Con questa suddivisione si esegue il modello iterandolo più volte su questa suddivisione cambiando ogni volta i sottoinsiemi dedicati al test set e quelli dedicati al

train, ci si fermerà una volta che sono stati testati tutte le combinazioni dei test set. La valutazione finale sarà l'aggregazione delle valutazioni ad ogni iterazione. Generalmente si utilizza stratified 10 -fold cross-validation ovvero si ha $k = 10$ sottoinsiemi in cui per ciascun sottoinsieme si cerca di mantenere la stessa distribuzione delle classi presente nel dataset, in questo modo si riesce a stimare meglio la valutazione e si diminuisce la varianza della stima, in aggiunta, le iterazioni saranno 10.

Per i dataset molto piccoli si usano **leave-one-out**, uguale al k-fold cross-validation con la differenza che il test sarà composto da un unico elemento, mentre il train sarà composto da tutti gli altri elementi del dataset. Si ripete la valutazione del modello fino a quando tutti gli elementi sono stati utilizzati come test. In questo modo si riesce ad utilizzare molto bene i dati, ma è molto costoso.

Infine, se il modello deve essere messo in produzione, allora, una volta trovato il modello migliore, effettuo il training sull'intero dataset senza effettuare la partizione.

7.3 Affidabilità delle misure di performance

Per avere maggior affidabilità sul confronto tra le misure della performance, non ci si può solo basare sulla media o sulla varianza della misura, perché sono incomplete. Il metodo migliore è quello di stimare gli intervalli di confidenza perché:

- si capisce la posizione della media
- ci da la robustezza determinata dall'ampiezza dell'intervallo di confidenza

Dati due modelli con la stessa media, ma uno ha un'ampiezza dell'intervallo minore, allora quello è il migliore. Se gli intervalli non si sovrappongono e hanno la stessa ampiezza, allora si prende quello con media migliore.

Un altro approccio di confronto tra modelli consiste nell'utilizzare il test statistico di significatività che specifica con una certa confidenza, quanto è significativa la differenza ottenuta:

- **ipotesi nulla**: non esiste una reale differenza
- **ipotesi alternativa**: esiste una reale differenza

Il test specifica con che confidenza possiamo rigettatre l'ipotesi nulla.

Il test si divide in:

- **test paired**: quando entrambi i modelli hanno imparato nello stesso modo sugli stessi dati
- **test unpaired**: altrimenti

In generale possiamo dire che:

- non guardare solo le performance
- considera sempre la complessità
- controlla anche l'output

Capitolo 8

Deeplearning

Le principali tipologie di apprendimento sono:

- **Supervisionato** (Supervised): i dati utilizzati per l'apprendimento sono etichettati (label).
- **Non supervisionato** (Unsupervised): i dati utilizzati per l'apprendimento non sono etichettati.

Inizialmente, le architetture di deeplearning sono state composte da particolari strutture simili agli enconder-decoder chiamate **autoencoder**. Queste strutture sono reti neurali, costruite con una particolare struttura a clessidera, il cui compito è quello di imparare come ricostruire l'input. La metodologia di apprendimento

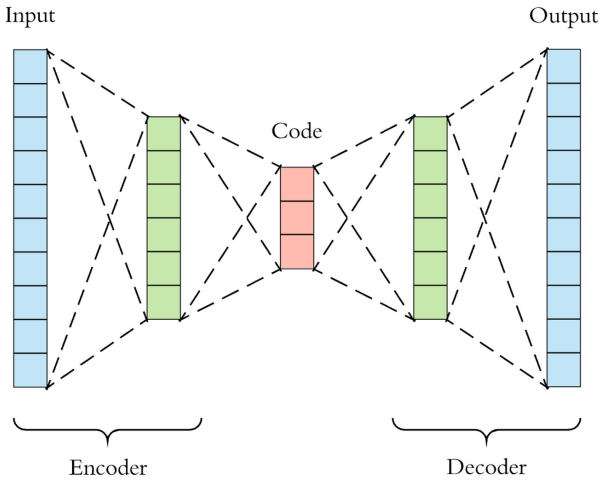

Figura 8.1: Autoencoder

utilizzata da questa rete prende il nome di **fake task**, ovvero si adotta una metodologia supervisionata in cui non si ha una label, ma quello che si vuole ottenere è l'input stesso. L'obiettivo è ridurre l'errore sulla ricostruzione dell'input. Il loro risultato è dovuto alla loro struttura a collo di bottiglia che separa la rete in due:

- **Enconder**: impara a codificare l'input utilizzando poche informazioni (funzione di codifica, spesso segnata come $Enc(X) = Z$).
- **Denconder**: impara a decodificare l'input partendo da poche informazioni (funzione di decodifica, spesso segnata come $Dec(Z) = Y$).

L'algortimo di apprendimento di queste reti spesso è una delle tante implementazioni di backpropagation, applicandolo su ciascun neurone di uscita. Più precisamente l'apprendimento si effettua definendo una **funzione di loss** $\mathcal{L}$ che calcolerà la distanza tra l'output del decoder Y e l'input X.

$$\mathcal{L} = \|X - Y\| \tag{8.1}$$

Per gli autoencoder si deve obbligatoriamente avere sempre una struttura a clessidra con la strozzatura centrale, la quale permette di generare la codifica e separare le due reti. In aggiunta, l'architettura non prevede che ci siano collegamenti tra encoder e decoder altrimenti la codifica nella strozzatura non è più consistente.

L'encoder della rete non fa altro che trovare una codifica dell'input in uno spazio di dimensioni ridotto, questo significa che equivale a calcolare la PCA sull'input, con la differenza che uno utilizza una rete neurale mentre l'altro è un processo iterativo basato sugli autovalori e autovettori. Più precisamente PCA cambia il sistema di riferimento dello spazio in modo tale che gli assi siano in direzione della massima variabilità. Vista questa dualità con la PCA, gli autoencoder possono trovare una rappresentazione dell'input a dimensioni minori per risolvere dei task specifici. La rappresentazione codificata prenderà il nome di **rappresentazione latente**.

L'autoencoder può essere utilizzato anche per effettuare task di classificazione più precisamente si allena l'autoencoder e nel mentre si allena una rete che prende la rappresentazione latente e la classifica. Quindi la rete sarà formata da:

- encoder $Z = Enc_{\theta_1}(X)$
- classificatore $\hat{y} = F(Z)$
- decoder $Y = Dec_{\theta_2}(Z)$

La loss dovrà essere l'unione delle loss del classificatore e dell'autoencoder.

$$\mathcal{L} = \|X - Y\| + \|y - \hat{y}\| \tag{8.2}$$

Le architetture odierne di deeplearning si basano su strutture neurali e utilizzano i **transformer**, componenti neurali che prendono in input sia il nuovo input, sia un vecchio output del trasformer. Sono componenti fondamentali per la generazione di informazioni. L'apprendimento di queste reti si basa su strutture **feed-forward** e su componenti **Attention**, quest'ultimi fondamentali perché permettono l'effettivo cambiamento dei valori. Prima di arrivare ai transformer si è passati dagli autoencoder, a **Word2Vec**, un modello neurale

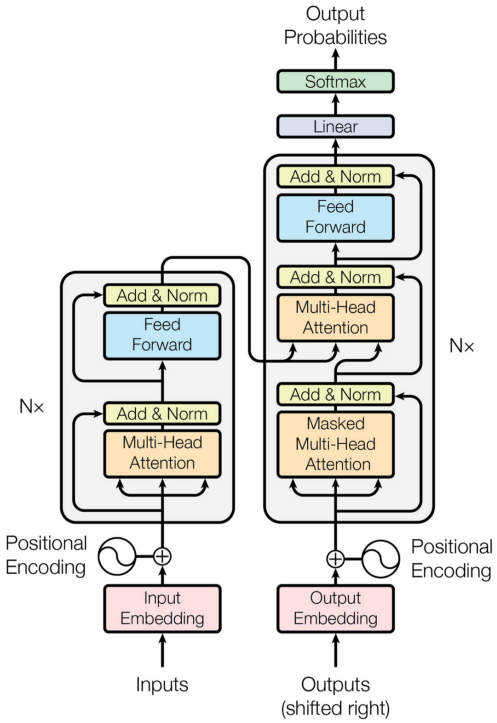

Figura 8.2: Struttura di un transformer

utilizzato per problemi di **NPL**. Più precisamente si occupa di associare per ogni parola del linguaggio un vettore di numeri reali, in questo modo è possibile rappresentare le singole parole in uno spazio n-dimensionale. Questa proprietà ha permesso di scoprire una proprietà: parole simili vengono rappresentate in una regione vicina dello spazio. Questo è dato dal fatto che le reti vengono allenate su tutti i testi raggiungibili e l'assegnamento del vettore si effettua in base a quante volte due parole vengono veste vicine in una frase. Quindi la rete riesce a codificare come vettori simili parole simili, infatti c'è una corrispodenza geometrica col significato.

Successivamente si è passati ai **denoising autoencoder**, ovvero rimozione del rumore e ricostruzione dell'input originale. Queste reti vengono allenate dando in input il dato sporcato in precedenza, successivamente l'output viene confrontato con l'input originale.

In seguito, sono stati introdotti i componenti **Attention**, composti da 3 elementi in input e il suo compito è di assegnare in automatico diversi pesi agli elementi in input. Queste reti permettono quindi di cambiare i pesi degli input portando a dei vantaggi notevoli, per esempio, si possono interpretare parole ambigue in base al contesto.

Infine, si è passati ai tranformer, introcendo la ricorrenza dei dati in modo da considerare in input i vecchi dati della rete, sfasati, con un meccanismo di Attention.

I modelli di deeplearning vengono allenati in 2 fasi:

- **pretraining generico**: allenamento non supervisionato, si allena la rete in modo generale senza specificare un particolare task. Questa fase è la più pesante e permette di definire un primo stato di partenza dei parametri.
- **finetuning**: si prende la rete preallenata e si addestra per un task specifico utilizzando la metodologia supervisionata.

In questo modo è possibile ridurre i tempi di addestramento e rendere riutilizzabile la stessa rete per task simili.

Alla fine si arriva alla creazione di reti più complesse come GPT, basata sempre su trasformer e attention, ma addestrata in 2 fasi, con la possibilità di effettuare del reinforcement learning per migliorare ancora di più i risultati.

Capitolo 9

Ripasso di algebra lineare

Per praticità ripasseremo i concetti fondamentali facendo riferimento a $\mathbb{R}^2$, formato quindi da elementi, dette coordinate, che sono coppie ordinate (x_1, x_2) (rappresentabili con un punto nel piano o con un segmento orientato con partenza nell'origine e destinazione nelle coordinate del punto nel piano).

Ricordiamo le operazioni fondamentali, dati R pari a (x_1, x_2) e Q pari a (x_3, x_4)

- addizione: $P + Q = (x_1 + x_3, x_2 + x_4)$
- prodotto per uno scalare $\lambda \in \mathbb{R}$: $\lambda \cdot R = (\lambda \cdot x_1, \lambda \cdot x_2)$
- prodotto scalare tra vettori: $\langle P, Q \rangle \equiv P \cdot Q^T = \sum_{i=1}^{n} r_i \cdot q_i$ (dove r_i e q_i sono rispettivamente gli elementi di R e Q all'indice i)

Ricordiamo la *norma* di un vettore X:

$$\|X\| = \equiv \sqrt{X \cdot X^T} = \sqrt{\sum_{i=1}^{n} x_i \cdot x_i} = \sqrt{\langle X, X \rangle} \tag{9.1}$$

Con $X = 0$ indichiamo il *vettore nullo* (che ha anche norma nulla).

Definiamo il *versore* (*vettore unitario*) come:

$$\frac{X}{\|X\|}, \quad X \neq 0 \tag{9.2}$$

In $\mathbb{R}^2$ l'angolo θ sotteso tra due vettori X e Y è:

$$\cos\theta = \frac{\langle X, Y \rangle}{\|X\| \cdot \|Y\|} \tag{9.3}$$

La proiezione di un vettore X sul vettore Y è:

$$X_Y = \|X\| \cdot \cos\theta \tag{9.4}$$

Si hanno quindi tre casi:

1. $\theta < 90 \iff \langle X, Y \rangle > 0$
2. $\theta > 90 \iff \langle X, Y \rangle < 0$
3. $\theta = 90 \iff \langle X, Y \rangle = 0$

(quindi disegnando una retta sul piano tutti i punti sopra di essa apparterranno ad una certa classe e quelli sotto ad un'altra).

Posso definire una retta r che passa per l'origine in $\mathbb{R}^2$ assegnando un vettore $W = (w_1, w_2)$ ad essa ortogonale, infatti tutti i punti, ovvero vettori, $X = (x_1, x_2)$ sulla retta sono ortogonali a W:

$$\langle W, X \rangle = w_1 \cdot x_1 + w_2 \cdot x_2 = 0 \tag{9.5}$$

Quindi la retta (ovvero l'iperpiano) mi separa due semispazi, a seconda che $\langle X, W \rangle$ sia strettamente positivo o strettamente negativo.

Generalizzando ora a n dimensioni ho che, dato l'iperpiano h (di dimensione $n - 1$):

- se h passa dall'origine allora si ha l'equazione $\langle X, Y \rangle = 0$
- se non passa per l'origine $\langle X, Y \rangle + b = 0$

I vettori in un iperpiano si proiettano tutti nello stesso modo e i punti ad un lato e all'altro dell'iperpiano sono distinti dal fatto che $\langle X, Y \rangle + b$ sia strettamente positiva o strettamente negativa.