Machine Learning

UniShare

Davide Cozzi
@dlcgold

Indice

Capitolo 1

Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: `https://github.com/dlcgold/Appunti`.
Si segnala che le immagini sono tratte dalle slide del corso.

Capitolo 2

Introduzione al ML

Il **Machine Learning (*ML*)** è sempre più diffuso nonostante sia nato diversi anni fa.
Un **sistema di apprendimento automatico** ricava da un *dataset* una conoscenza non fornita a priori, descrivendo dati non forniti in precedenza. Si estrapolano informazioni facendo assunzioni sulle informazioni sistema già conosciute, creando una **classe delle ipotesi H**. Si cercano ipotesi coerenti per guidare il sistema di apprendimento automatico. 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. In quest'ottica bisogna mediare 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.
Definiamo alcuni concetti base:

- **task (*T*)**, il compito da apprendere. È più acile 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 (e bisognerà capire come misurare la cosa)

- **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*

Durante l'**apprendimento** si estrapolano dati da **istanze di addestramento o test**. Quindi:

- si ricevono i dati di addestramento
- il sistema impara ad estrapolare partendo da quei dati
- si ricevono dati di test su cui si estrapola

L'ipotesi da apprendere viene chiamata **concetto target** (tra tutte le ipotesi possibili identifico quella giusta dai dati di addestramento).
Approfondiamo il discorso relativo all'*esperienza*. Innanzitutto nel momento della scelta bisogna valutare la rappresentatività esperienza. SI ha inoltre un controllo dell'esperienza da parte del *learner*:

- l'esperienza può essere fornita al learner senza che esso possa interagire
- il learner può porre domande su quegli esempi che non risultano chiari

L'esperienza deve essere presentata in modo causale.
Si hanno due tipi di esperienza:

1. **diretta**, dove i learner può acquisire informazione utile direttamente dagli esempi o dover inferire indirettamente da essi l'informazione necessaria (può essere chiaramente più complicato)
2. **indiretta**

Il tipo di dato che studieremo comunemente sarà il **vettore booleano** e la risposta sarà anch'essa di tipo booleano. In questo contesto l'ipotesi è una **congiunzione di variabili**.
Per ogni istanza di addestramento cerchiamo una risposta eventualmente corrispondente al nostro *target* (ovvero 1), qualora esista.
Si hanno tre tipi di apprendimento:

1. **apprendimento supervisionato**, 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:

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

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

2. **apprendimento non supervisionato**, 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:

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

   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

3. **apprendimento per rinforzo**, 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 (ad esempio per una macchina che deve giocare ad un gioco). 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 apprendimento:

1. **attivo**, dove il *learner* può "domandare" sui dati disponibili

2. **passivo**, dove il *learner* apprende solo a partire dai dati disponibili

Si parla di **inductive learning** quando voglio apprendere una funzione da un esempio (banalmente una funzione target f con esempio $(x, f(x))$, ovvero una coppia). Si cerca quindi un'ipotesi h, a partire da un insieme d'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. Viene usato un approccio che sfrutta anche il *Rasoio di Occam*.

Terminologia:

- X, **spazio delle istanze**, ovvero la collezione di tutte le possibili **istanze** utili per qualche compito di *learning*. In termini statistici lo *spazio delle istanze* non è altro che lo **spazio campione** (ovvero lo spazio degli esiti fondamentali di un esperimento concettuale)
- $x \in X$, **istanza**, ovvero un singolo "oggetto" preso dallo **spazio delle istanze**. Ogni **istanza** è rappresentata tramite un **vettore di attributi unici** (un attributo per posizione del vettore)
- c, **concetto**, $c \subseteq X$, ovvero un sottoinsieme dello *spazio delle istanze* che descrive una *classe* di oggetti (ovvero di istanze) alla quale siamo interessati per costruire un modello di *machine learning*. In pratica raccolgo quel sottoinsieme di istanze che mi garantiscono, per esempio, uno o più attributi. La nozione statistica equivalente è quella di *evento* (ovvero un sottoinsieme dello *spazio campione*). Si ha quindi che, preso un concetto $A \subseteq X$:

$$f_A : X \to \{0, 1\}$$

$$f_a(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{altrimenti} \end{cases}$$

- h, **ipotesi**, $h \subseteq X$
- 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**
- $D = \{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$, **training set**, ovvero è la raccolta degli esempi. Qualora si avesse a che fare con un *training non supervisionato* si avrebbe: $D = \{x_1, \ldots, x_n\}$
- $\{(x'_1, f(x'_1)), \ldots, (x'_n, f(x'_n))\}$, **test**

- un **modello di machine learning** (dove *machine learning* viene anche definito come lo studio di diverse strategie, più precisamente di ottimizzazione, per cercare ipotesi soddisfacenti/efficienti nello spazio delle ipotesi) è quindi l'*ipotesi migliore*. Questo **modello predittivo** viene addestrato tramite il *training set* e servirà per inferire nuove informazioni mai state osservate nel *training set*. Lo *spazio delle ipotesi* può quindi essere chiamato anche **spazio dei modelli** (come del resto *ipotesi* e **modello** intendono la stessa cosa)

- **linguaggio delle ipotesi**, è il linguaggio che definisce lo *spazio delle ipotesi/modelli*

  Esempio 1. *Prendiamo un problema semplice di regressione lineare.*
  *In questo contesto un'istanza è un punto (x, y), lo spazio delle ipotesi è quello formato dalle rette $y = a + bx$ (dove a è il nostro **bias**). Tra tutte queste rette cerco quella che sarà il **modello predittivo**, tramite la quale poter prevedere l'andamento dei vari punti (di modo che dato un x sia in grado di stabilire un buon y)*

- **cross validation**, ovvero ripeto m volte la validazione su campioni diversi di input per evitare che un certo risultato derivi dalla fortuna

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

- **soddisfazione di un'ipotesi**: un'istanza x soddisfa un'ipotesi h sse tutti i vincoli espressi da h sono soddisfatti dai valori di x e si indica con:

  $$h(x) = 1$$

  Esempio 2. *Avendo:*

  $$x = \langle S, W, N, S, W, S\rangle \text{ e } h = \langle S, ?, ?, S, ?, S\rangle$$

  posso dire che x soddisfa h.
  se invece ho che:

  $$h = \langle S, ?, ?, \emptyset, ?, S\rangle$$

  posso dire che x non soddisfa h.

Si avrà, in realtà, a che fare con dati, di target e ipotesi, booleani e questo ambito è propriamente chiamato **concept learning**.

Definizione 1. *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* $\{0, 1\}$*:*

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

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 valori di 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

Esempio 3. *Vediamo quindi la rappresentazione di una ipotesi (ipotizzando di avere a che fare con solo 4 attributi* A_i*, sempre in prospettiva booleana):*

$$h = \langle 0, 1, ?, 1\rangle = \langle A_1 = 0, A_2 = 1, A_3 = ?, A_4 = 1\rangle$$

nella realtà, grazie al "?" riferito all'istanza, l'ipotesi h è un insieme di due ipotesi:

$$h \in \{(0, 1, 0, 1),\ (0, 1, 1, 1)\}$$

passando quindi da una notazione per ipotesi ad una insiemistica.
Ricordiamo inoltre che lo spazio delle istanze X, dal punto di vista insiemistico è:

$$X = \{(x_1, x_2, x_3, x_3) :\ x_1 \in A_1, x_2 \in A_2, x_3 \in A_3, x_4 \in A_4\}$$

Se un'istanza x soddisfa i vincoli di h allora h classifica x come esempio positivo:

$$h(x) = 1$$

Quindi, dato un *training set* D, cerco di determinare un'ipotesi $h \in H$ tale che:

$$h(x) = c(x), \forall x \in D$$

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 2. *Date $h_j, h_k \in H$ booleane e definite su X. Si ha che h_j è **più generale o uguale a** h_k (e si scrive con $h_j \geq h_k$) sse:*

$$(h_k(x) = 1) \longrightarrow (h_j(x) = 1), \ \forall x \in X$$

Si impone quindi un ordine parziale.
*Si ha che h_j è **più generale di** h_k (e si scrive con $h_j > h_k$) sse:*

$$(h_j \geq h_k) \wedge (h_k \ngeq h_j)$$

*Riscrivendo dal punto di vista insiemistico si ha che h_j è **più generale o uguale a** h_k sse:*

$$h_k \supseteq h_j$$

*e che è **più generale di** h_k sse:*

$$h_k \supset h_j$$

*Dal punto di vista logico si ha che h_j è **più generale di** h_k sse impone meno vincoli di h_k*

Lo spazio delle ipotesi è descritto da una congiunzione di attributi.

Esempio 4. *Facciamo un esempio "giocattolo" di situazione di **concept learning**.*
*Il **target concept**, ovvero la domanda che ci si pone, è:*

In quali tipologie di giorni A apprezza fare sport acquatici?

Abbiamo quindi una serie di attributi per il meteo con i vari valori che possono assumere:

Attributo	***Possibili valori***
sky	*Sunny, Cloudy, Rainy*
temp	*Warm, Cold*
humid	*Normal, High*
wind	*Strong, Weak*
water	*Warm, Cold*
forecast	*Same, Change*

In ottica concept learning si cerca quindi la ***funzione target*** *enjoySport (che sarebbe la nostra funzione c):*

$$enjoySport : X \to \{0, 1\}$$

Vediamo quindi anche un esempio di training set D (per praticità i valori degli attributi sono indicati con la sola iniziale):

sky	**temp**	**humid**	**wind**	**water**	**forecast**	**enjoySport**
S	W	N	S	W	S	yes
S	W	H	S	W	S	yes
R	C	H	S	W	C	no
S	W	H	S	C	C	yes

*Dove nella colonna finale si ha la risposta booleana al problema, è infatti l'****etichetta target****.*
Bisogna quindi cercare un'ipotesi h tale che $h(x) = c(x)$, per tutte le istanze nel training set.
Un'ipotesi h, anch'essa rappresentata come vettore, sarà quindi una congiunzione $\wedge$ di vincoli di valore sugli attributi.

Esempio 5. *Vediamo un esempio anche relativo alle nozioni di* ***più generale di****.*
Si hanno due ipotesi:

$$h_J = \langle S, ?, ?, ?, ?, ? \rangle$$

$$h_k = \langle S, ?, ?, S, ?, ? \rangle$$

e quindi si ha che:

$$h_J \geq h_k \text{ ovvero } h_k(x) = 1 \implies h_j(x) = 1$$

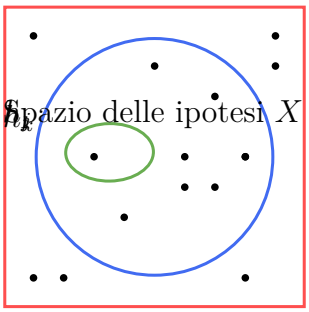


Figura 2.1: Rappresentazione di **più generale di** con i diagrammi di Eulero-Venn, dove si nota come, in X, $h_j \geq h_k$

infatti un'istanza positiva per h_k è sicuramente positiva anche per h_j, in quanto h_k ha un vincolo più restrittivo sul quarto attributo, che deve assumere il valore S, mentre il quarto attributo di h_j può assumere qualsiasi valore. Questo aspetto si potrebbe rappresentare con un ***diagramma di Eulero-Venn*** *dal punto di vista insiemistico (con il "cerchio" di h_j che conterrebbe quello di h_k (essendo un insieme più esterno e quindi più generale), nello spazio delle istanze).*

Esempio 6. *Consideriamo un'ipotesi h, con quattro attributi, dal punto di vista insiemistico:*

$$h \in \{(0,1,1,1)\,(0,1,0,0)\}$$

e ci si chiede se $h \in H$.
In primis possiamo "comprimere" questa rappresentazione insiemistica in:

$$h = \langle 0,1,?,? \rangle$$

h ora è rappresentata come tipicamente fatto nel concept learning.
Quindi $h \in H$, avendo un'ipotesi con quattro attributi.

Esempio 7. *Consideriamo un concetto c, con cinque attributi, dal punto di vista insiemistico:*

$$c \in \left\{ \begin{array}{l} (0,1,0,1,0), \\ (0,1,1,1,0), \\ (0,1,0,1,1), \\ (0,1,1,1,1) \end{array} \right\}$$

e ci si chiede se $c \in H$, quindi se la strategia è di ricerca è buona esso può essere ritrovato (altrimenti nessuna strategia lo potrebbe riconoscere).

Studiando c otteniamo che:

$$c = \langle 0, 1, ?, 1, ? \rangle$$

Usiamo la stessa rappresentazione usata per le ipotesi

Esempio 8. *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 $X = A_1 \times A_2$ (con $\times$ **prodotto cartesiano**), ho che:*

$$|X| = |A_1| \times |A_2|$$

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

Esempio 9. *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:*

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

Esempio 10. *Vediamo un esempio di studio del numero delle ipotesi, ovvero si studia la cardinalità dello spazio delle ipotesi (che altro non è che 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 $\emptyset$ (anche non per lo stesso attributo). 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 $|A|$ il numero di valori possibili per l'attributo A):*

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

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.

La seguente affermazione è un esercizio dato a casa, che io risolverei come scritto.
Qualora volessi conteggiare tutte ipotesi ***sintatticamente differenti*** *dovrò contare* $\emptyset$ *come ipotetico valore per ogni attributo e quindi:*

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

2.1 Primi algoritmi

2.1.1 Algoritmo find-S

Parliamo ora dell'algoritmo **Find-S**. Questo algoritmo permette di partire dall'ipotesi più specifica (attributi nulli, indicati con $\emptyset$) 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.

Algorithm 1 Algoritmo Find-S

```
function FINDS
    h ← 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
    return ipotesi h
```

Esercitazione sull'algoritmo find-S

Verranno usati concetti spiegati più avanti in questi appunti.
Si ha anche il **bias induttivo** che permette di studiare anche esempi non visti nel training set, anche se è una situazione rara da analizzare a causa dell'inconsistenza stessa degli esempi e dal loro rumore.

Esempio 11. *Prendiamo il seguente training set D:*

esempio	A_1	A_2	A_3	A_4	label
x_1	1	0	1	0	1
x_2	0	1	0	0	0
x_3	1	0	1	1	0
x_4	0	0	0	0	1
x_5	0	0	1	0	1

*Applico quindi **find-S**, che ad ogni iterazione cerca di generalizzare un poco l'ipotesi più specifica.*
Parto dall'ipotesi più specifica in assoluto:

$$S = \{\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$$

Si presenta il primo esempio $x_1 = (1, 0, 1, 0)$ che ha label 1, quindi $t(x_1) = 1$. Ovviamente il primo valore già non soddisfa il vincolo imposto da S e quindi in S faccio il replace del valore del primo attributo di x_1 con quello di S. Visto che S è però l'ipotesi più specifica dovrò farlo per tutti i vincoli, ottenendo che S diventa, di fatto, una copia di x_1:

$$S = \{\langle 1, 0, 1, 0 \rangle\}$$

Si presenta il secondo esempio $x_2 = (0, 1, 0, 0)$ che ha label 0, quindi $t(x_2) = 0$. Essendo un esempio negativo viene ignorato dall'algoritmo, lasciando inalterato S.
Si presenta il terzo esempio $x_3 = (1, 0, 1, 1)$ che ha label 0, quindi $t(x_3) = 0$.Essendo un esempio negativo viene ignorato dall'algoritmo, lasciando inalterato S.
Si presenta il quarto esempio $x_4 = (0, 0, 0, 0)$ che ha label 1, quindi $t(x_4) = 1$. Controllo quindi con S i vari valori. Dove si ha lo stesso valore lo tengo in S, altrimenti pongo il caso generico "?" (sempre nell'ottica di generalizzare sempre più), in quanto ho due esempi che mi dicono che avere, per quell'attributo', 1 o 0 va comunque bene. Alla fine avrò:

$$S = \{\langle ?, 0, ?, 0 \rangle\}$$

Si presenta il quinto esempio $x_5 = (0, 0, 1, 0)$ che ha label 1, quindi $t(x_5) = 1$. Procedo come sopra e ottengo, infine:

$$S = \{\langle ?, 0, ?, 0 \rangle\}$$

*Si arriva quindi a dire che $S = \{\langle ?, 0, ?, 0 \rangle\}$ è la nuova ipotesi più specifica che verrà restituita dall'algoritmo **find-S**.*

Esempio 12. *Si ha la richiesta opposta allo scorso esempio.*
Data l'ipotesi:

$$S = \{\langle ?, 0, ?, ? \rangle\}$$

Si trovino 5 esempi (3 positivi e due negativi) che portino, secondo ***find-S****, a restituire quell'ipotesi.*
Parto sempre da:

$$S = \{\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle\}$$

e mi invento gli esempi.
Un primo è $x_1 = (1, 0, 1, 0)$ *(dato che il secondo attributo ha sicuro valore 0) tale che* $t(x_1) = 1$*. Raggiungo quindi:*

$$S = \{\langle 1, 0, 1, 0 \rangle\}$$

ora me ne bastano due che rendano "?" tutti gli attributi tranne il secondo, che deve restare 0. Prendo quindi un esempio positivo che, tranne per il secondo valore che deve restare 0 e per un'altro dei tre che deve restare come per x_1 *in modo da avere un ulteriore esempio positivo, sia il complemento del primo. Ho quindi* $x_2 = (1, 0, 0, 1)$*, con* $t(x_2) = 1$*. Ho già ottenuto:*

$$S = \{\langle 1, 0, ?, ? \rangle\}$$

Ora, per il terzo esempio, prendo $x_3 = (0, 0, 0, 1)$*, con* $t(x_3) = 1$*. Ottengo esattamente:*

$$S = \{\langle ?, 0, ?, ? \rangle\}$$

Gli altri due esempi possono essere con qualsiasi valori ma devono essere negativi, non influenzando il risultato.

Nell'esercitazione il prof usa altri esempi:

- $x_1 = (1, 0, 1, 0)$ *con* $t(x_1) = 1$
- $x_2 = (0, 1, 0, 0)$ *con* $t(x_2) = 0$
- $x_3 = (1, 0, 1, 1)$ *con* $t(x_3) = 1$
- $x_4 = (0, 0, 0, 0)$ *con* $t(x_4) = 0$
- $x_5 = (0, 0, 0, 0)$ *con* $t(x_5) = 1$

Con gli ultimi due che rappresentano, in un contesto reale, un ***errore*** *e un* ***rumore****, in quanto gli stessi valori portano ad avere le due label opposte, si ha quindi inconsistenza. Sono infatti detti* ***esempi inconsistenti*** *e vengono completamente ignorati da* ***find-S****.*

Sono quindi ben chiari i problemi di **find-S**:

- potrebbero esserci altre ipotesi consistenti rispetto a quella in output
- training set inconsistenti possono ingannare l'algoritmo

D'altro canto:

- garantisce in output l'ipotesi più specifica consistente con gli esempi positivi
- l'ipotesi finale è consistente anche con gli esempi negativi, a patto che il *target concept* sia contenuto in H e che gli esempi siano corretti

2.1.2 Algoritmi di eliminazione

Definizione 3. *Definiamo* ***soddisfacibilità*** *quando un esempio x soddisfa un'ipotesi h, evento indicato con:*

$$h(x) = 1$$

a priori sul fatto che x sia un esempio positivo o negativo del target concept. Si ha quindi che i valori x soddisfano i vincoli h.

Definizione 4. *Si dice che h è* ***consistente*** *con il training set D di concetti target sse:*

$$Consistent(h, D) := h(x) = c(x),\ \forall \langle x, c(x) \rangle \in D$$

Definizione 5. *Si definisce* ***version space****, rispetto ad H e D, come il sottoinsieme delle ipotesi da H consistenti con D e si indica con:*

$$VS_{H,D} = \{h \in H | Consistent(h, D)$$

Esempio 13. *Vediamo un esempio per capire la consistenza. Riprendiamo la situazione dell'esempio 4 con un training set D leggermente diverso:*

example	sky	temp	humid	wind	water	forecast	enjoySport
1	S	W	N	S	W	S	yes
2	S	W	H	S	W	S	yes
3	R	C	H	S	W	C	no
4	S	W	H	S	C	C	yes

e prendiamo l'ipotesi:

$$h = \langle S, W, ?, S, ?, ? \rangle$$

e studiamo se h sia ***consistente*** *con D.*
Studio l'esempio 1: ($\langle S, W, N, S, W, S \rangle$,yes). Vedo che ogni valore va bene e, avendo la classificazione yes, ho la stessa etichetta assegnata dalla ***funzione target*** *enjoySport, quindi l'ipotesi è consistente con l'esempio.*
Questo vale per tutti e quattro gli esempi (il terzo non "appaia" ma questo è confermato dalla label no) e quindi la mia ipotesi è ***consistente*** *con il training set.*

Vediamo quindi algoritmo **List-Then Eliminate**:

Algorithm 2 Algoritmo List-Then Eliminate

function LTE
vs $\leftarrow$ *una lista connettente tutte le ipotesi di* H
for *ogni esempio di training* $\langle x, c(x) \rangle$ **do**
rimuovi da vs ogni ipotesi h non consistente con l'esempio di training, ovvero $h(x) \neq c(x)$
return *la lista delle ipotesi in vs*

Questo algoritmo è irrealistico in quanto richiese un numero per forza esaustivo di ipotesi.

Definizione 6. *Definiamo:*

- *G come il confine generale di* $VS_{H,D}$*, ovvero l'insieme dei membri generici al massimo. È l'insieme delle ipotesi più generali:*

$$G = \{g \in H | \, g \text{ è consistente con } D \wedge$$

$$(\nexists g' \in H \text{ t.c } g' \geq g \wedge g' \text{ è consistente con } D\})$$

Possiamo dire che $G = \langle ?, ?, ?, \ldots ? \rangle$.
G è detto anche ***insieme massimamente generico di ipotesi***

- *S come il confine specifico di* $VS_{H,D}$*, ovvero l'insieme dei membri specifici al massimo. È l'insieme delle ipotesi più specifiche:*

$$S = \{s \in H | \, s \text{ è consistente con } D \wedge$$

$$(\nexists s' \in H \text{ t.c } s' \geq s \wedge s' \text{ è consistente con } D\})$$

Possiamo dire che $S = \langle \emptyset, \emptyset, \emptyset, \ldots \emptyset \rangle$.
G è detto anche ***insieme massimamente specifico di ipotesi***

Teorema 1. *Ogni elemento di $VS_{H,D}$ si trova tra i confini definiti da S e G:*

$$VS_{H,D} = \{h \in H | \, (\exists s \in S) \; (\exists g \in G) \; (g \geq h \geq s)\}$$

con $\geq$ che specifica che è più generale o uguale

Vediamo quindi algoritmo **candidate eliminate**

Algorithm 3 Algoritmo Candidate Eliminate

```
function CE
    G ← insieme delle ipotesi più generali in H
    S ← insieme delle ipotesi più specifiche in H
    for ogni esempio di training d = ⟨x, c(x)⟩ do
        if d è un esempio positivo then
            rimuovi da G ogni ipotesi inconsistente con d
            for ogni ipotesi s in S inconsistente con d do
                rimuovi s da S

                aggiungi a S tutte le generalizzazioni minime h di s
                tali che h sia consistente con d e qualche membro di G
                sia più generale di h
            rimuovi da S ogni ipotesi più generale di un'altra in S
        else
            rimuovi da S ogni ipotesi inconsistente con d
            for ogni ipotesi g in G inconsistente con d do
                rimuovi g da G

                aggiungi a G tutte le generalizzazioni minime h di g
                tali che h sia consistente con d e qualche membro di S
                sia più generale di h
            rimuovi da G ogni ipotesi più generale di un'altra in G
    return la lista delle ipotesi in vs
```

Questo algoritmo ha alcune proprietà:

- converge all'ipotesi h corretta provando che non ci sono errori in D e che $c \in H$
- se D contiene errori allora l'ipotesi corretta sarà eliminata dal *version space*
- si possono apprendere solo le congiunzioni

- se H non contiene il concetto corretto c, verrà trovata l'ipotesi vuota

Il nostro spazio delle ipotesi non è in grado di rappresentare un semplice concetto di target disgiuntivo, si parla infatti di **Biased Hypothesis Space**. Studiamo quindi un **unbiased learner**. Si vuole scegliere un H che esprime ogni concetto insegnabile, ciò significa che H è l'insieme di tutti i possibili sottoinsiemi di X. H sicuramente contiene il concetto target. S diventa l'unione degli esempi positivi e G la negazione dell'unione di quelli negativi. Per apprendere il concetto di target bisognerebbe presentare ogni singola istanza in X come esempio di training.
Un learner che non fa assunzioni a priori in merito al concetto target non ha basi "razionali" per classificare istanze che non vede.
Introduciamo quindi il **bias induttivo** considerando:

- un algoritmo di learning del concetto L
- degli esempi di training $D_C = \{\langle x, c(x)\rangle\}$

Si ha che $L(x_i, D_c)$ denota la classificazione assegnata all'istanza x_1, da L, dopo il training con D_c.

Definizione 7. *Il **bias induttivo** (con **bias** che normalmente denota una* distorsione *o un* scostamento *dei dati) di L è un insieme minimale di asserzioni B tale che, per ogni concetto target c e D_c corrispondente si ha che:*

$$[B \wedge D_c \wedge x_i] \vdash L(x_i, D_c),\ \forall x_i \in X$$

con $\vdash$ che rappresenta l'implicazione logica

Possiamo quindi distinguere:

- **sistema induttivo**, dove si hanno in input gli esempi di training e la nuova istanza, viene usato l'algoritmo *candidate eliminate* con H e si ottiene o la classificazione della nuova istanza nulla
- **sistema deduttivo** equivalente al sistema induttivo sopra descritto dove in input si aggiunge l'asserzione "H contiene il concetto target" e si produce lo stesso output tramite un **prover di teoremi**

Abbiamo quindi visto tre tipi di *learner*:

1. il **rote learner**, dove si ha classificazione sse x corrisponde ad un esempio osservato precedentemente. Non si ha *bias induttivo*

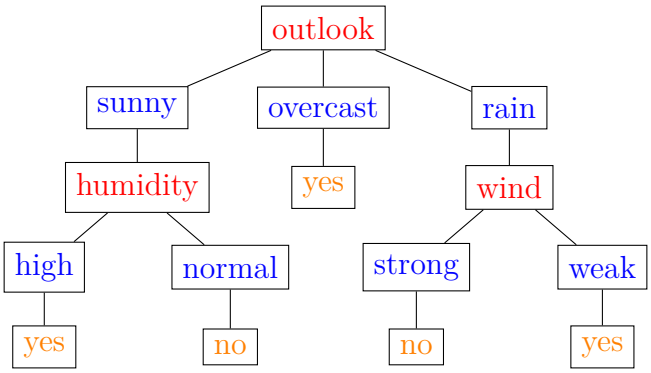


Figura 2.2: Esempio di albero decisionale

2. l'algoritmo **candidare eliminate** con **version space**, dove il *bias* corrisponde al fatto che lo spazio delle ipotesi contiene il concetto target

3. l'algoritmo **Find-S**, dove il *bias* corrisponde al fatto che lo spazio delle ipotesi contiene il concetto target e tutte le istanze sono negative a meno che il target opposto sia implicato in un altro modo

2.2 Alberi decisionali

Vediamo come sfruttare una struttura dati discreta, l'**albero di decisione**, per affrontare problemi di *concept learning*. Su questa struttura implementeremo l'algoritmo chiamato **ID3** che tra le ipotesi sceglie il risultato dell'apprendimento tramite esempi di addestramento. La lista delle ipotesi in questo caso è enorme e la scelta è guidata dal cosiddetto *information gain*.
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. Siamo in ambito di **apprendimento supervisionato**.
Si hanno attributi che hanno anche più di due valori. Per ogni ipotesi si ha un albero di decisione, come quello in figura 2.2. In rosso si hanno gli attributi, in blu i valori degli attributi e in arancione le foglie coi risultati. Le foglie sono le risposte booleane.
Avanzando nell'albero cerchiamo una risposta (che ci deve essere).
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 (per avere un albero coerente con le istanze ricevute).

Formule booleane possono essere rappresentate in un albero decisionale (figura 2.3 e figura 2.4), costruendo un albero che sia *yes* solo nei casi la formula booleana sia vera.

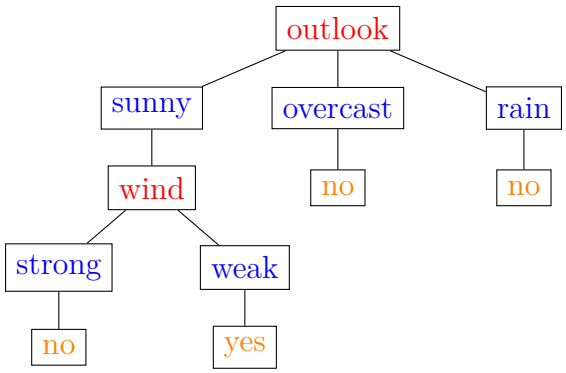


Figura 2.3: Esempio di albero decisionale per la formula $(Outlook = Sunny) \wedge (Wind = Weak)$

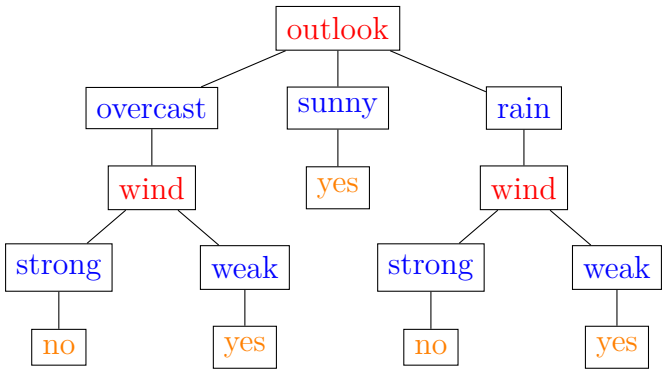


Figura 2.4: Esempio di albero decisionale per la formula $(Outlook = Sunny) \vee (Wind = Weak)$

Notiamo a questo punto come l'albero decisionale in figura 2.2 è la rappresentazione di:

$$(Outlook = Sunny \wedge Humidity = Normal)$$

$$\vee (Outlook = Overcast)$$

$$\vee (Outlook = Rain \wedge Wind = Weak)$$

Possiamo dire che gli alberi decisionali descrivono tutte le funzioni booleane. Avendo n funzioni booleane avremo un numero distinto di tabelle di verità

(e quindi di alberi decisionali), ciascuna con 2^n righe, pari a 2^{2^n}.
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

Esercitazione sugli alberi decisionali

Partiamo con un esercizio con find-S per coglierne le problematiche.

Esercizio 1. *Siano dati due attributi* $A = \{1, 2, 3\}$ *e* $B = \{1, 2\}$. *Diciamo che* H *è un congiunzione di and tra i valori degli attributi e delle istanze.*
Il concetto target è:

$$c := ((A = 1 \vee A = 2), B = 1)$$

Find-S può trovare c *in* H*?*
Prendo un training set contenente $\langle x_1 = (1, 1), 1\rangle$ *e* $x_2 = \langle(2, 1), 1\rangle$. *Seguendo find-S avremo la seguente traccia dell'algoritmo:*

A	B
$\emptyset$	$\emptyset$
1	1
?	1

Si arriva quindi a:

$$S = \langle ?, 1\rangle$$

Ma questa generalizzazione mi farebbe accettare $A = 3$*, cosa non prevista da* c*.*
Non posso quindi usare find-S in questo caso.

Ricordiamo che un albero decisionale è formato da:

- **nodes (*nodi*)** etichettati da i vari attributi
- **branches *rami*** etichettati con i possibili valori dell'attributo che etichetta il nodo sorgente del ramo

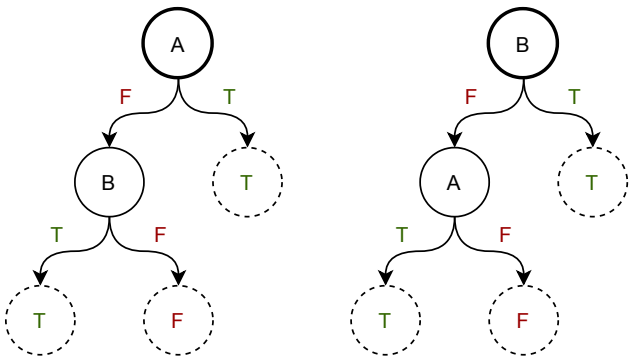
- **leef nodes (*foglie*)** etichettati con gli outcome della previsione

Un percorso dalla radice fino alla foglia mi rappresenta una congiunzione di attributi mentre l'albero in se è quindi una disgiunzione di congiunzioni (una per ogni percorso radice-foglia). Un albero quindi formalizza la congiunzione di vincoli su attributi ma può estendere il linguaggio a disgiunzioni di vincoli su attributi.
Usando funzioni booleane (quindi un attributo può avere valore $\top$, T o $\bot$, F) 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.
Posso avere alberi diversi a seconda della scelta del nodo radice.

Esempio 14. *Vediamo i due alberi per la funzione booleana* $\vee$.
Abbiamo la tabella di verità:

A	B	$B \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

rappresentata dai due alberi:



Essendo sempre nel concept learning, nel caso di funzioni booleane, anche il target è booleano.

Teorema 2. *Con un albero decisionale posso rappresentare tutte le funzioni booleane*

Dimostrazione. Prendiamo una qualsiasi funzione booleana e la traduciamo in tabella di verità.
A partire dalla tabella costruisco l'albero decisionale dove ogni percorso radice-foglia è un esempio (quindi una riga) della tabella di verità.
Dato che ogni funzione booleana può essere rappresentata con una tabella di verità posso costruire un albero decisionale per qualsiasi funzione booleana. □

Il metodo appena descritto comunque dimostra il teorema ma non è sempre efficiente, dovendo memorizzare tutto.

Teorema 3. *Avendo una funzione booleana con n attributi allora posso costruire una tabella di verità con* 2^n *righe. Inoltre posso costruire potenzialmente* 2^{2^n} *diverse ma con lo stesso significo, e quindi altrettanti alberi decisionali.*

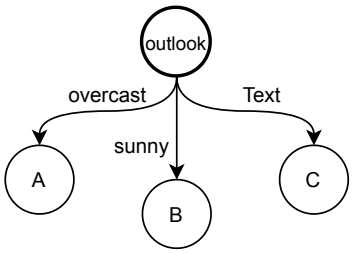
Vediamo quindi un algoritmo generale per la costruzione dell'albero:

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*

Bisognerà capire:

- come fare in modo ottimizzato lo '*split*
- quando fermare la costruzione (quindi capire quando "non c'è altro da fare")

Esempio 15. *Prendiamo il solito esempio giocattolo di quando andare a fare sport in base al clima.*
In base al valore di outlook ottengo tre tabelle, in base al valore di overcast, con target play (nell'immagine il terzo arco è etichettato con rainy e non con text):



outlook	temp	humidity	windy	play
overcast	H	H	$\bot$	yes
overcast	C	N	$\top$	yes
overcast	M	H	$\top$	yes
overcast	H	N	$\bot$	yes

Tabella 2.1: Tabella A

outlook	temp	humidity	windy	play
sunny	H	H	$\bot$	no
sunny	H	H	$\top$	no
sunny	M	H	$\bot$	no
sunny	C	N	$\bot$	yes
sunny	M	N	$\top$	yes

Tabella 2.2: Tabella B

outlook	temp	humidity	windy	play
rainy	M	H	$\bot$	yes
rainy	C	N	$\bot$	yes
rainy	C	N	$\top$	no
rainy	M	N	$\bot$	yes
rainy	M	H	$\top$	no

Tabella 2.3: Tabella C

Dividendo tramite i valori di outlook ho fatto uno split, dividendo così i dati.
Vediamo un secondo i valori del target nei vari casi:

- *nel caso di overcast ho 4 yes e 0 no*
- *nel caso di sunny ho 2 yes e 3 no*

- *nel caso di rainy ho 3 yes e 2 no*

Posso quindi usare una ***funzione di costo*** *per fissare quando una distribuzione è omogenea (come nel caso di overcast) o quando no (gli altri due casi). Lo split infatti andrebbe fatto in base ai valori del target, più sono omogenei e meglio è, in quanto nel momento in cui si presenta un nuovo test per la classificazione con outlook pari a overcast saprò già cosa fare (in quanto nello storico delle esperienze ho sempre avuto yes). Le altre due situazioni sono ambigue e quindi, in quei due casi, devo procedere con la costruzione dell'albero per ottenere informazioni cercando di rimuovere l'incertezza.*

La **funzione costo** è alla base della strategia della costruzione dell'albero. Una strategia può essere quella "a maggioranza", 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 io risultati. Scelgo l'attributo con più omogeneità (e quindi somma più bassa), che ha una distribuzione più pulita dei risultati. Se ho valori con solo *yes* o solo *no* sommo 0.

Esempio 16. *Prendendo le tre tabelle sopra, per outlook avrei:*

$$0 + 1 + 1 = 2$$

Se avessi avuto un attributo con:

- *primo valore: 1 yes e 1 no*
- *secondo valore: 2 yes e 0 no*
- *terzo valore: 0 yes e 4 no*
- *quarto valore: 2 yes e 4 no*
- *quinto valore: 2 yes e 2 no*

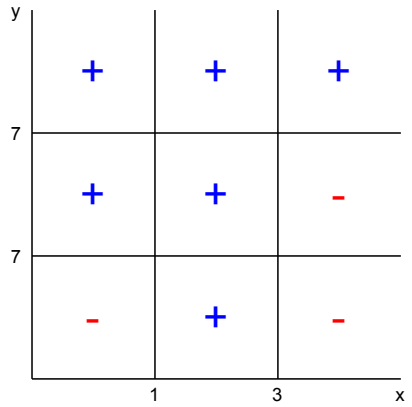
avrei avuto:

$$1 + 0 + 0 + 2 + 2 = 5$$

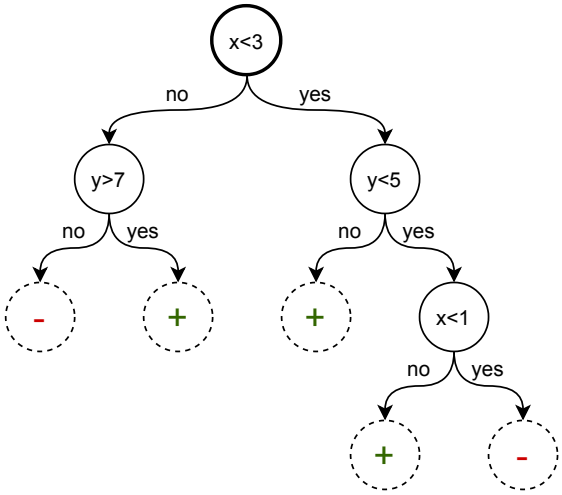
Gli alberi di decisione possono essere utilizzati anche nel continuo, per attributi numerici.

Esempio 17. *Prendiamo le istanze definite da due attributi, x e y e costruisco un albero di decisione che definisce in quali aree del piano ho $-$ e quali ho $+$.*

*Si ha il seguente piano (**sull'asse delle** y **il primo valore a partire dall'origine è un 5 non un 7**):*



E si ottiene, per esempio, il seguente albero decisionale:



Dove a ciascun nodo è associata una condizione e agli archi il fatto che sia verificata o meno.
Si nota come non si hanno tutte le condizioni, infatti, per esempio, con $x > 3$ mi basta $y > 7$ per trovare il $+$ e $y < 7$ per il $-$ (non dovendo andare specificatamente a guardare anche $y < 5$ o $y > 5$).
Quello disegnato è solo uno dei possibili alberi.

2.2.1 Algoritmo ID3

Vista la difficoltà di scegliere l'albero si ha l'idea di scegliere un piccolo albero di partenza (o più piccoli) e ricorsivamente l'attributo più significativo (sia

nei nodi rossi intermedi che nelle foglie) come radice per il 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:

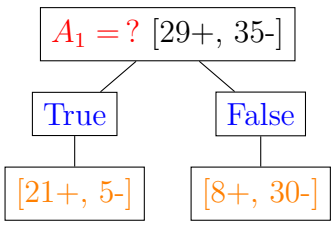
Algorithm 4 Algoritmo ID3 (Iterative Dichotomiser 3)

function ID3
    A ← *il "miglior" attributo di decisione per il prossimo nodo*
    assegno A come attributo di decisione per il nodo, che sarà rosso
    for *ogni valore dell'attributo A* **do**
        creo un discendente
    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

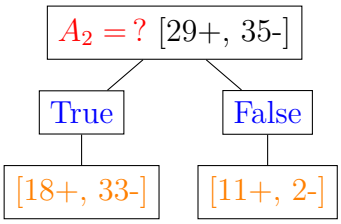
Il **bias induttivo** su ID3 è che preferisce alberi piccoli.
Bisogna in primis capire cosa si intende come **attributo migliore**. Per farlo introduciamo la seguente notazione:

$$[esempi\ positivi+,\ esempi\ negativi-]$$

entrambi rappresentati con un valore intero. Gli esempi positivi sono gli esempi che già mi hanno restituito *yes* mentre quelli negativi. In generale sono tutti esempi su cui devo ancora valutare l'attributo. E proseguo così assegnando etichette positive e negative. Vediamo quanto detto:



Se siamo nella situazione in cui dobbiamo confrontare l'attributo sopra con un altro, per esempio:



Entrambe le foglie del primo attributi ci parlano di valori sbilanciati (tra esempi positivi e negativi), a differenza delle due del secondo attributo, dove sono una sbilanciata e una no. Per ora stabiliamo ad occhio lo sbilanciamento. Il criterio di scelta ci porta a preferire lo sbilanciamento, verso un ideale "tutti positivi" o "tutti negativi". Quindi se un attributo ha divisioni sbilanciate è da ritenersi migliore.
Per essere ancora più precisi bisogna richiamare la matematica dell'**entropia**.

Definizione 8. *Dato un training set S con valori v_i, $i = 1 \dots n$. Se l'entropia di un insieme di bit misura più o meno la sua quantità di informazione (quanto è "speciale") noi possiamo richiamare una formula per l'entropia su S:*

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1} -P(v_i) \log_2 P(v_i)$$

Dove $I(x)$ indica il valore dell'entropia su x e $P(y)$ sta per la probabilità legata ad un valore y.
Nel caso booleano le istanze presenti in un certo insieme S sono associate ad un'etichetta, conteggiandole. Nella variabile p conto i valori di S con etichetta positiva e con n negativa (esempi positivi e negativi). Ottengo quindi in modo esplicito la sommatoria:

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

*Se inoltre diciamo che p_+ è la proporzione di esempi positivi e p_- di quelli negativi (saranno quindi tra 0 e 1) possiamo misurare l'**impurità** di S con l'entropia:*

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Avrò quindi alta entropia se positivi e negativi sono "metà e metà"

Parliamo quindi di **information gain** IG che viene calcolato su ogni attributo A e su S:

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

dove:

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

e quindi l'information gain è la riduzione aspettata nell'entropia per ordinare S sull'attributo A. Si sceglie l'attributo con il maggiore IG.
Possiamo riscrivere il conto come:

$$IG(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Esempio 18. *Vediamo l'esempio di calcolo di entropia di A_1 con [29+,35-]:*

$$Entropy([29+, 35-]) = -\frac{29}{64}\log_2\frac{29}{64} - \frac{35}{64}\log_2\frac{35}{64} = 0.99$$

Calcolo anche l'information gain di A_1, sapendo che $Entropy([21+, 5-]) = 0.71$ e $Entropy([8+, 30-]) = 0.74$, e quindi:

$$IG(S, A_1) = 0.99 - \frac{26}{64} \cdot 0.71 - \frac{38}{64} \cdot 0.74 = 0.27$$

ugualmente calcolo $IG(S, A_2) = 0.12$.
Quindi so che devo scegliere A_1 in quanto $0.27 > 0.12$

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 (ma trovo scelte buone localmente e non ottime)
- 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 più semplici) 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 9. *Definizione tratta da wikipedia.*
*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 ($error_{traini}(h)$) 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:

$$error_{traini}(h) < error_{traini}(h') \ \wedge \ error_D(h) > error_D(h')$$

quindi se presa un'altra ipotesi questa è migliore della prima e ha un errore sull'intera distribuzione delle ipotesi inferiore vado in *overfit*. 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.

Esercitazione su ID3

Quando parliamo di *entropia* stiamo volgendo un esperimento concettuale. Prima dell'esperimento si ha una certa incertezza su un certo eventi, che sparisce, con sorpresa, una volta eseguito l'esperimento se l'evento accade. D'altro canto qualora accade un evento atteso siamo meno sorpresi del fatto. Nel primo caso però si ritiene di ottenere molta informazione, nel secondo caso poca. Come se ponessimo in una vasca quattro palline rosse, non sarei stupito di estrarne una rossa, essendo quello che mi aspetto. D'altro canto se ne aggiungo quattro verdi l'estrazione sarà inattesa e quindi più interessante. Si cerca un modo di quantificare questa "sorpresa". Riprendendo l'esempio

della palline posso dire che nel primo caso (solo rosse) ho **bassa entropia** e nel secondo caso (palline miste) **alta entropia**. Un caso intermedio sarebbe a **media entropia**.
Quindi, detta $P(x)$ la probabilità che avvenga un evento x e con $I(p)$ l'informazione che ottengo dopo che l'evento si è verificato:

- $P(x) = 1 \rightarrow I(p) = 0$
- $P(x) = 0 \rightarrow I(p) = \infty$

L'informazione $I(p)$ e quindi una quantità non negativa:

$$I(p) \geq 0$$

Definiamo quindi l'informazione, detta anche *self-information*, per un evento E:

$$I(E) = -\log_2(P(E))$$

Inoltre ho che è additiva se gli eventi sono indipendenti, ovvero:

$$I(p_1, p_2) = I(p_1) + I(p_2)$$

Bisogna però estendere l'informazione a tutte le possibili distribuzioni di tutti i possibili esiti che ho in un esperimento concettuale.
Passiamo quindi alle definizioni matematiche per l'**entropia**:

- $X \sim P_X$, presa una certa variabile X dell'esperimento con una certa funzione di probabilità associata P_X
- $Val(X) = \{x_1, \ldots, x_n\}$, ovvero il range di valori della variabile X
- $p_i = P_X(x_i)$, probabilità per quel valore della variabile X
- $g : \mathbb{R} \rightarrow \mathbb{R}$ una funzione arbitraria tale per cui $g(X)$ è una variabile causale. Si definisce l'aspettativa di $G(X)$ su P come:

$$E_P[g(X)] = \sum_{x \in Val(X)} g(x) \cdot P_X(x)$$

Giungendo quindi alla formula dell'**entropia** di una variabile:

$$H[X] = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i = E_P[\log_2(p)]$$

Quindi:

Definizione 10. *L'**entropia**, definita da Claude Shannon (e quindi spesso chiamata **entropia di Shannon**), è l'informazione media associata ad una distribuzione di probabilità.*

Vediamo un esempio:

Esempio 19. *Suppongo di lanciare una moneta, si ha:*

$$P(testa) = P(croce) = \frac{1}{2}$$

Definiamo che testa è specificata da $X = 0$ e croce da $X = 1$.
Calcoliamo quindi:

$$H(p) = -p(0) \cdot \log_2 p(0) - p(1) \cdot \log_2 p(1) = -2 \cdot (\frac{1}{2} \cdot \log_2 \frac{1}{2}) = 1$$

Una moneta "onesta" ha quindi entropia pari a 1 (avendo due probabili esiti equiprobabili non ho certezza del risultato).
Ipotizziamo di avere una moneta magica che cade solo sulla testa (quindi $P(0) = 1$ e $P(1) = 0$:

$$H(p) = -p(0) \cdot \log_2 p(0) - p(1) \cdot \log_2 p(1) = -\log_2(1) = 0$$

Una moneta "non onesta" ha quindi entropia pari a 0 (ho infatti certezza del risultato).

Esempio 20. *Considero il seguente training set, con 4 esempi e target T:*

example	A	B	C	D	T
x_1	0	0	1	1	1
x_2	0	1	1	1	1
x_3	0	1	0	0	0
x_4	0	1	0	1	1

Si ha quindi la seguente distribuzione di probabilità relativa al target (avendo un no e tre yes):

$$P_T = \left[\frac{1}{4}, \frac{3}{4}\right]$$

e quindi posso calcolare l'entropia della tabella:

$$H(P_T) = -P_T(0) \cdot \log_2 P_T(0) - P_T(1) \cdot \log_2 P_T(1) = -\frac{1}{4} \cdot \log_2 \frac{1}{4} - \frac{3}{4} \cdot \log_2 \frac{3}{4} = 0.81$$

La tabella può essere vista come un risultato di un esperimento concettuale.

Definizione 11. *Definiamo l'**entropia di una distribuzione condizionale**. Presa X come una variabile discreta arbitraria con valori $\{x_1, \ldots, x_n\}$, che hanno ciascuno probabilità $P_X(x_i)$ ho che, per la distribuzione condizionale:*

$$PY|X = x_i(y_j) = P_{Y|X}(y_j|x_i)$$

Ovvero la distribuzione dei valori della variabile Y dato $X = x_i$.
Quindi voglio sapere la probabilità di Y condizionata da un esperimento precedente su X.
Per l'entropia ho:

$$H_{Y|X=x_i} = -\sum_{j=1}^{m} P_{Y|X}(y_j|x_i) \cdot \log_2 P_{Y|X}(y_j|x_i)$$

quindi è la solita formula ma con la probabilità condizionale.

Definizione 12. *Definiamo **entropia condizionale** come il valore medio, ovvero il valore atteso, dell'entropia di $p_{Y|X=x_i}$ per ciascun valore di X che condiziona Y, ovvero:*

$$H_{Y|X} = \sum_{i=1}^{n} P_X(x_i) H_{Y|X=x_i}$$

*ottenendo l'**entropia condizionale**:*

$$H[Y|X] = \sum P(x) \cdot H(Y|X = x)$$

Esercizio 2. *Considero il seguente training set, con 4 esempi e target T:*

example	A	B	C	D	T
x_1	0	0	1	1	1
x_2	0	1	1	1	1
x_3	0	1	0	0	0
x_4	0	1	0	1	1

Abbiamo già calcolato l'entropia associabile al valore target $H[P_T] = 0.81$.
Passiamo ora all'uso di ID3 per la costruzione dell'albero.
Dobbiamo cercare gli split corretti tramite information gain, usando l'entropia condizionale.
Partiamo con il primo attributo: A. Si ha che $P_A(0) = 1$ e $P_A(1) = 0$, quindi:

$$H[T|A = 0] = -p_{T|A}(0|0) \cdot \log_2(p_{T|A}(0|0)) - p_{T|A}(1|0) \cdot \log_2(p_{T|A}(1|0)) =$$

$$-\frac{1}{4}\cdot\log_2\frac{1}{4}-\frac{3}{4}\cdot\log_2\frac{3}{4}=0.81$$

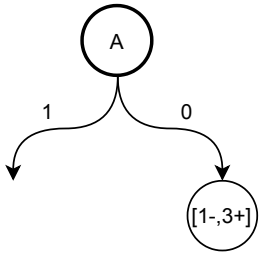
(Risultato pari a quello dell'intero training set in quanto A è sempre 0)
Non devo calcolare $H[T|A=1]$ *in quanto A non è mai pari ad 1.*
Inoltre si ha:

$$H[T|A]=P_A(0)\cdot H[T|A=0]+P_A(1)\cdot H[T|A=1]=1\cdot 0.81=0.81$$

Posso quindi calcolare l'information gain:

$$IG[T;A]=H[T]-H[T|A]=0.81-0.81=0$$

Quindi per A ho la seguente distribuzione del target (avendo nel target 3 esempi positivi e uno negativo e A sempre con valore 0):



Ricordiamo che ID3 sceglie per lo splitting l'attributo che rende massimo l'information gain.
Passo quindi all'attributo B. Si ha che $P_B(0)=\frac{1}{4}$ *e* $P_B(1)=\frac{3}{4}$*, quindi:*

$$H[T|B=0]=-p_{T|B}(0|0)\cdot\log_2(p_{T|B}(0|0))-p_{T|B}(1|0)\cdot\log_2(p_{T|B}(1|0))=$$

$$-0\cdot\log_2 0-1\cdot\log_2 1=0$$

e:

$$H[T|B=1]=-p_{T|B}(0|1)\cdot\log_2(p_{T|B}(0|1))-p_{T|B}(1|1)\cdot\log_2(p_{T|B}(1|1))=$$

$$-\frac{1}{3}\cdot\log_2\frac{1}{3}-\frac{2}{3}\cdot\log_2\frac{2}{3}=0.91$$

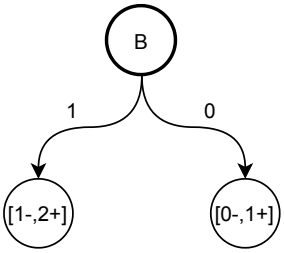
(ho quindi una partizione migliore con $B=0$*)*
Inoltre si ha:

$$H[T|B]=P_B(0)\cdot H[T|B=0]+P_B(1)\cdot H[T|B=1]=\frac{1}{4}\cdot 0+\frac{3}{4}\cdot 0.91=0.68$$

Posso quindi calcolare l'information gain:

$$IG[T;B]=H[T]-H[T|B]=0.81-0.68=0.13$$

Quindi per B ho:



Ho quindi un partizionamento più interessante.
Passo quindi all'attributo C. Si ha che $P_C(0) = \frac{1}{2}$ *e* $P_C(1) = \frac{1}{2}$*, quindi:*

$$H[T|C=0] = -p_{T|C}(0|0) \cdot \log_2(p_{T|C}(0|0)) - p_{T|C}(1|0) \cdot \log_2(p_{T|C}(1|0)) =$$

$$-\frac{1}{2} \cdot \log_2 \frac{1}{2} - \frac{1}{2} \cdot \log_2 \frac{1}{2} = 1$$

e:

$$H[T|C=1] = -p_{T|C}(0|1) \cdot \log_2(p_{T|C}(0|1)) - p_{T|C}(1|1) \cdot \log_2(p_{T|C}(1|1)) =$$

$$-0 \cdot \log_2 0 - 1 \cdot \log_2 1 = 0$$

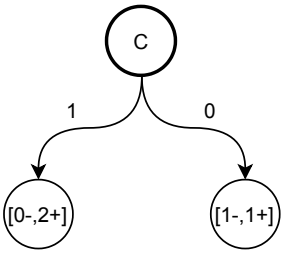
Inoltre si ha:

$$H[T|C] = P_C(0) \cdot H[T|C=0] + P_C(1) \cdot H[T|C=1] = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 0 = \frac{1}{2}$$

Posso quindi calcolare l'information gain:

$$IG[T;C] = H[T] - H[T|C] = 0.81 - \frac{1}{2} = 0.31$$

Quindi per C ho:



C migliora ancora il partizionamento.
Passo quindi all'attributo D. Si ha che $P_D(0) = \frac{1}{4}$ *e* $P_D(1) = \frac{3}{4}$*, quindi:*

$$H[T|D=0] = -p_{T|D}(0|0) \cdot \log_2(p_{T|D}(0|0)) - p_{T|D}(1|0) \cdot \log_2(p_{T|D}(1|0)) =$$

$$-1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0$$

e:

$$H[T|D=1] = -p_{T|D}(0|1) \cdot \log_2(p_{T|D}(0|1)) - p_{T|D}(1|1) \cdot \log_2(p_{T|D}(1|1)) =$$
$$-0 \cdot \log_2 0 - 1 \cdot \log_2 1 = 0$$

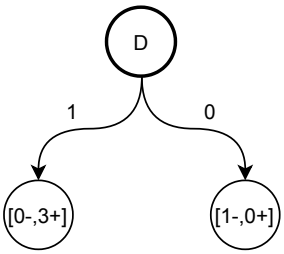
Inoltre si ha:

$$H[T|D] = P_D(0) \cdot H[T|D=0] + P_D(1) \cdot H[T|D=1] = \frac{1}{4} \cdot 0 + \frac{3}{4} \cdot 0 = 0$$

Posso quindi calcolare l'information gain:

$$IG[T;D] = H[T] - H[T|D] = 0.81 - 0 = 0.81$$

Quindi per D ho:



D rende quindi il massimo della "purezza" tra i valori di D e quelli del target. Non ho incertezza nel partizionamento.
Quindi, ricapitolando, ho i seguenti information gain:

- $IG[T;A] = 0$
- $IG[T;B] = 0.13$
- $IG[T;C] = 0.31$
- $IG[T;D] = 0.81$, ***che è il valore massimo*** *e che rende minima la "sorpresa"*

Esercizio 3. *Considero il seguente training set, con 4 esempi e target T:*

example	A	B	C	D	T
x_1	0	0	1	...	1
x_2	0	1	1	...	1
x_3	0	1	0	...	0
x_4	0	1	0	...	1

Bisogna riempire D per rendere massimo l'information gain.

Per farlo basta mettere gli stessi valori del target, in modo che sia l'attributo che meglio distribuisca i valori del target, ottenendo quindi lo stesso training set dell'esercizio precedente:

example	A	B	C	D	T
x_1	0	0	1	1	1
x_2	0	1	1	1	1
x_3	0	1	0	0	0
x_4	0	1	0	1	1

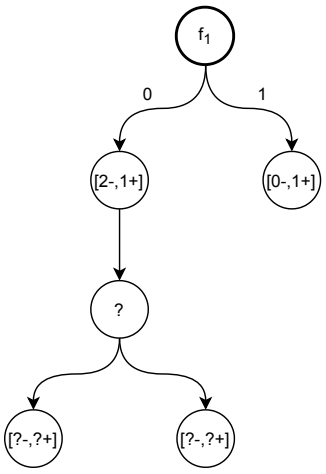
Un'alternativa è l'esatto opposto, in quanto otterrei la stessa ridistribuzione del target, rimuovendo ogni "sorpresa" ulteriore ma lasciando solo quella della tabella iniziale:

example	A	B	C	D	T
x_1	0	0	1	0	1
x_2	0	1	1	0	1
x_3	0	1	0	1	0
x_4	0	1	0	0	1

Esercizio 4. *Considero il seguente training set, con 4 esempi e target T:*

example	f_1	f_2	f_3	T
x_1	1	1	1	1
x_2	0	1	1	0
x_3	0	0	1	1
x_4	0	0	0	0

Vogliamo completare il seguente albero decisionale:



Partendo da f_1 so che se arriva un nuovo esempio non potrò proseguire da $[0-, 1+]$ in quanto so già che in quel caso avrei un'istanza positiva, valore minimo 0.
Quindi se $f_1 = 0$ vado a scegliere un nuovo attributo in quanto ancora non ho una distribuzione certa. Considero quindi le righe in cui $f_1 = 0$ e avanzo iterativamente studiando f_2 ed f_3. Studio quindi il sottoinsieme:

example	f_2	f_3	T
x_2	1	1	0
x_3	0	1	1
x_4	0	0	0

e avanzo fino a che non finisco gli attributi (o arrivo in un punto in cui, come per il ramo 1 di f_1 non posso più continuare).
Per questo nuovo sottoinsieme calcolo:

$$P_T = \left[\frac{2}{3}, \frac{1}{4}\right]$$

e quindi posso calcolare l'entropia della tabella:

$$H(P_T) = -P_T(0)\cdot\log_2 P_T(0) - P_T(1)\cdot\log_2 P_T(1) = -\frac{2}{3}\cdot\log_2\frac{2}{3} - \frac{1}{3}\cdot\log_2\frac{1}{3} = 0.91$$

Passo quindi all'attributo f_2. Si ha che $P_{f_2}(0) = \frac{2}{3}$ e $P_{f_2}(1) = \frac{1}{3}$, quindi:

$$H[T|f_2 = 0] = -p_{T|f_2}(0|0)\cdot\log_2(p_{T|f_2}(0|0)) - p_{T|f_2}(1|0)\cdot\log_2(p_{T|f_2}(1|0)) =$$

$$-\frac{1}{2}\cdot\log_2\frac{1}{2} - \frac{1}{2}\cdot\log_2\frac{1}{2} = 1$$

e:

$$H[T|f_2 = 1] = -p_{T|f_2}(0|1)\cdot\log_2(p_{T|f_2}(0|1)) - p_{T|f_2}(1|1)\cdot\log_2(p_{T|f_2}(1|1)) =$$

$$-0\cdot\log_2 0 - 1\cdot\log_2 1 = 0$$

Inoltre si ha:

$$H[T|f_2] = P_{f_2}(0)\cdot H[T|f_2 = 0] + P_{f_2}(1)\cdot H[T|f_2 = 1] = \frac{2}{3}\cdot 1 + \frac{1}{3}\cdot 0 = \frac{2}{3}$$

Posso quindi calcolare l'information gain:

$$IG[T; f_2] = H[T] - H[T|f_2] = 0.81 - \frac{2}{3} = 0.25$$

Passo quindi all'attributo f_3. Si ha che $P_{f_3}(0) = \frac{1}{3}$ e $P_{f_3}(1) = \frac{2}{3}$, quindi:

$$H[T|f_3 = 0] = -p_{T|f_3}(0|1) \cdot \log_2(p_{T|f_3}(0|1)) - p_{T|f_3}(1|1) \cdot \log_2(p_{T|f_3}(1|1)) =$$

$$-1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0$$

e:

$$H[T|f_3 = 1] = -p_{T|f_3}(0|0) \cdot \log_2(p_{T|f_3}(0|0)) - p_{T|f_3}(1|0) \cdot \log_2(p_{T|f_3}(1|0)) =$$

$$-\frac{1}{2} \cdot \log_2 \frac{1}{2} - \frac{1}{2} \cdot \log_2 \frac{1}{2} = 1$$

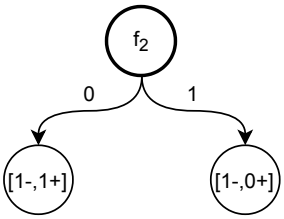
Inoltre si ha:

$$H[T|f_3] = P_{f_3}(0) \cdot H[T|f_3 = 0] + P_{f_3}(1) \cdot H[T|f_3 = 1] = \frac{1}{3} \cdot 0 + \frac{2}{3} \cdot 1 = \frac{2}{3}$$

Posso quindi calcolare l'information gain:

$$IG[T; f_3] = H[T] - H[T|f_3] = 0.81 - \frac{2}{3} = 0.25$$

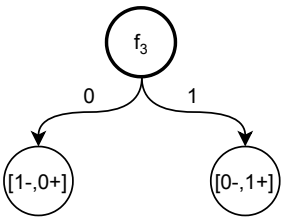
Avendo f_1 e f_2 lo stesso IG prendo il primo e quindi ho:



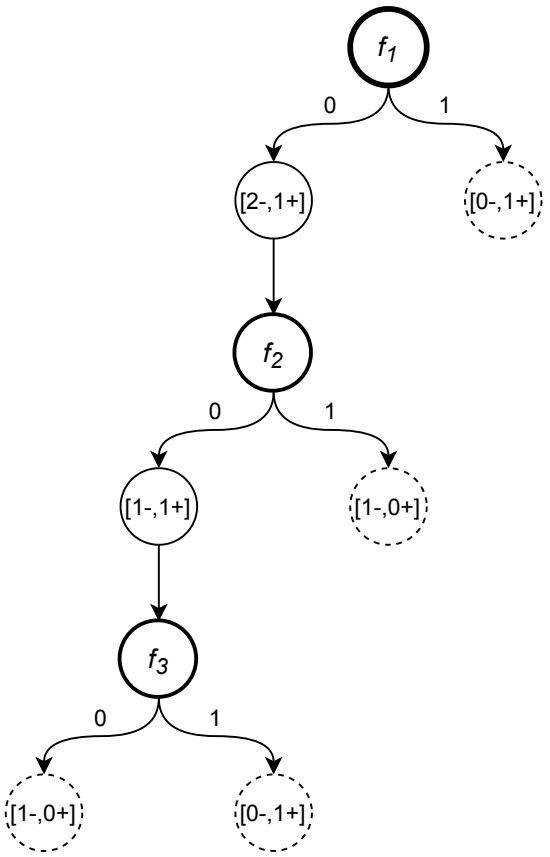
Con f_2 che verrà attaccato al nodo $[2-, 1+]$ "uscente" da f_1.
A questo punto, come sopra, abbiamo che il nodo $[1-, 0+]$ è foglia, avendo una distribuzione certa. Riduco quindi nuovamente il training set studiando solo gli esempi in cui f_2 vale 0, ovvero x_3 e x_4, per l'attributo f_3;

example	f_3	T
x_3	1	1
x_4	0	0

Non sono necessari conti in quanto f_3 è l'ultimo attributo rimasto ed è distribuito in questo modo:



Avendo per di più entrambi i risultati con distribuzione certa.
Il nodo di f_3 sarà attaccato al nodo $[1-, 1+]$ "uscente" da f_2, ottenendo così l'albero:



Capitolo 3

Reti neurali

3.1 Neurone biologico

In natura l'operazione di *learning* viene eseguita tramite il **cervello**, che tramite i **neuroni**, delle cellule nervose, è in grado di effettuare una miriade di operazioni in parallelo. Potenzialmente i neuroni hanno un tempo di risposta nell'ordine dei millisecondi mentre in circuito logico si aggira nell'ordine dei nanosecondi quindi la differenza deve essere ricercata nell'*architettura* del nostro cervello. La "potenza di calcolo" del cervello è data dal funzionamento parallelo di $\sim 10^{11}$ neuroni, collegati tra loro da $\sim 10^5$ connessioni. Vediamo quindi indicativamente gli elementi principali di un **neurone biologico**:

- **corpo**, che implementa tutte le funzioni logiche del neurone
- **assone**, 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 (**bottone sinaptico**) del neurone (detto **neurone pre-sinaptico**) trasmette impulsi nervosi provenienti dal neurone ai dendriti di altri neuroni (detti **neuroni post-sinaptici**)

Questa "architettura" è quindi basata sull'emissione di segnali da parte del neurone. Questa azione dipende da vari fattori, come ad esempio la forza del segnale ricevuto da altri neuroni e la forza delle connessioni di un neurone con le sue sinapsi. Si ha che la **funzione di risposta** di un neurone è una

funzione non lineare impulso ricevuto dai dendriti. Dopo l'invio di un impulso ogni neurone ha un tempo, detto **refractory time**, prima del quale poter inviare un altro impulso. Si hanno infatti due stati possibili per il *neurone biologico*:

1. **eccitazione**, quando il neurone invia, tramite le sinapsi, segnali (che per comodità computazionale chiamiamo già **pesati**) ai neuroni connessi
2. **inibizione**, quando il neurone non invia segnali

La **transizione di stato** dipende dall'entità complessiva dei segnali eccitatori e inibitori ricevuti dal neurone.
Una legge importante è la **regola di Hebb** che indica che i cambiamenti di forza delle connessioni delle sinapsi di due neuroni connessi è proporzionale alla correlazione tra l'emissione di segnali dei neuroni stessi (ovvero se due neuroni rispondono allo stesso input allora è bene che siano connessi).

3.2 Neuroni formali

Dopo una breve introduzione biologica possiamo alla definizione **formale e matematica** che verrà usato nello studio delle reti neurali. Questo modello matematico è stato proposto da **McCulloch** e **Pitts** e definisce formalmente un **neurone binario a soglia** come una quadrupla:

$$\langle n, C, W, \theta \rangle$$

dove:

- n specifica il **nome** del neurone stesso, una sorta di identificativo
- C specifica l'**insieme degli input** c_i
- W specifica il **vettore dei pesi** w_i, associati ad ogni input c_i. È una rappresentazione formale dei pesi delle sinapsi (e si nota che possono essere sia positivi che negativi)
- θ specifica la **soglia**, utile per definire quando un neurone manda effettivamente il segnale

Per definire i **due stati del neurone** si usa l'insieme $\{0, 1\}$ o l'insieme $\{-1, 1\}$ (si ha infatti un **neurone binario**). La **funzione di transizione** viene indicata con:

$$s(t+1) = 1 \text{ sse } \sum w_i \cdot s_i(t) \geq \theta$$

ovvero in un tempo successivo $t+1$ (con $s(t)$ che definisce uno stato al tempo t) ho che il neurone emette il segnale (stato

pari ad 1) sse al tempo precedente t ho avuto una somma pesata, tramite w_i, degli input $c_i(t)$ maggiore della soglia θ.
L'insieme degli stati, rappresentato in modo binario, comporta che la funzione di transizione sia una **funzione a scalino**:

$$f(x) = \begin{cases} 1 & \text{se } x \geq \theta \\ 0 & \text{altrimenti} \end{cases} \quad \text{con} \quad x = \sum w_i \cdot c_i$$

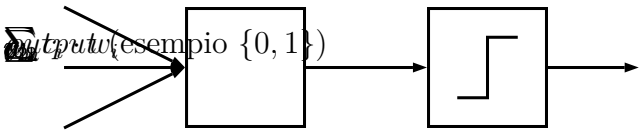


Figura 3.1: Rappresentazione matematica del modello di McColluch e Pitts, ovvero rappresentazione della **Linear Treshold Unit (*LTU*)**

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**, dove, avendo come insieme degli stati $\mathbb{R}$ si avrebbe:

$$f(x) = \frac{1}{1 + e^{-x}} \quad \text{con} \quad x = \sum w_i \cdot c_i$$

3.3 Reti neurali artificiali

Lo studio di un singolo neurone non è comunque particolarmente interessante. Si introduce quindi lo studio di **reti neurali** dove vengono posti diversi *neuroni* in modo che possano fare qualcosa di utile.
Se dal singolo neurone voglio passare alla rete collego in modo orientato i neuroni, di modo che l'output di uno sia l'input dell'altro.
Si hanno alcune **caratteristiche strutturali** delle *reti neurali artificiali*:

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

Ci sono anche alcune **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

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

- 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 di pesi per ogni neurone e quindi una matrice, che risulta a singolo colonna se ho un solo neurone)
- 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 $n_i(t) = \sum_{j=1}^{n} w_{ij} \cdot s_j(t) - \theta_i$, quindi si ha che la soglia influisce già nell'input del neurone (e quindi non si può ragionare come se θ fosse un confronto a posteriori)
- la **funzione di transizione** indicata con $s_i(t+1) = g(n_i(t))$

L'output di un neurone è, a conti fatti, 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 (*feedforward* o *feedback*), il numero di neuroni (con più layer o solo uno, *monostrato* o *multistrato*) etc...
- le **modalità di attivazione**, che può essere *seriale ciclica, seriale probabilistica, parallela* o *mista*
- un **algoritmo di apprendimento** con lo studio, in primis, dei pesi

3.3.1 Percettrone

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:

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

Con $X = 0$ indichiamo il *vettore nullo* (che ha anche norma nulla).
Definiamo il *versore* (*vettore unitario*) come:

$$\frac{X}{\|X\|}, \ X \neq 0$$

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

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

La proiezione di un vettore X sul vettore Y è:

$$X_Y = \|X\| \cdot \cos\theta$$

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:

$$\langle W, X \rangle = w_1 \cdot x_1 + w_2 \cdot x_2 = 0$$

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

Il percettrone è la tipologia di rete neurale più semplice, sono infatti una semplificazione estrema del sistema nervoso.

Definizione 13. *Un percettrone è una collezione di neuroni, di cardinalità M, a cui viene aggiunto un insieme di nodi in input, di cardinalità N (generalmente diversa da quella della collezione di neuroni). Generalmente gli input sono pesati e con la notazione w_{ij} indichiamo il peso della connessione tra il nodo i-simo in input e il neurone j-simo.*
I neuroni sono tra loro completamente indipendenti comportando anche un insieme di elementi in output. Nel caso semplice di funzioni di transizione binarie l'output sarà quindi un vettore binario contenente all'indice k-simo 1 se il neurone k ha inviato il segnale o 0 altrimenti.
Solitamente coi percettroni si parla di learning supervisionato.

Dal punto di vista geometrico il vettore dei pesi W rappresenta un **iperpiano** (che è una retta in $\mathbb{R}^2$) che separa i possibili vettori di input in due classi, a seconda che formino con W un angolo acuto o ottuso.
Studiamo quindi l'apprendimento del percettrone.

Come abbiamo visto l'output è rappresentato da un vettore booleano rappresentante 1 in posizione k in corrispondenza del neurone k-simo che ha inviato il segnale o 0 altrimenti. A questo risultato si giunge tramite un certo input pesato e, essendo un training supervisionato, esso viene verificato. Qualora un risultato non vada bene bisogna procedere cambiando i pesi. Il problema è appunto la scelta dei pesi, che non sono conoscibili a priori. Un peso troppo alto porta un neurone a mandare il segnale anche quando non dovrebbe mentre un peso troppo basso impedisce l'invio del segnale anche quando il neurone dovrebbe. Preso un neurone k che porta un risultato errato posso definire una **function error** come:

$$E = y_k - t_k$$

dove:

- y_k è l'output del neurone
- t_k è il target atteso per quel neurone

Per tale neurone bisognerà sistemare tutti i pesi w_{ik}.
Se E è negativa allora il neurone avrebbe dovuto emettere il segnale ma non lo ha fatto e quindi bisogna aumentare il peso e viceversa (nel caso binario o ho -1 o 1). Bisogna però considerare che l'input potrebbe essere negativo e quindi anche in pesi devono poter essere negativi. Perfezioniamo quindi il conto della differenza di peso necessaria con la moltiplicazione di E (messa in negativo in modo da eventualmente "sistemare" il segno per input negativi) per l'input:

$$\Delta w_{ik} = -(y_k - t_k) \times c_i$$

In questo discorso bisogna inserire anche la soglia, importante per input specifici (basti pensare ad un input pari a 0 che annullerebbe ogni cambio di peso secondo la formula precedente). Per ora trascureremo tali casi anche se una semplice soluzione per un caso limite come quello di avere solo input nulli, è quella di aggiungere un **nodo bias**, di valore -1, collegato ai neuroni con peso nullo.
Viene anche introdotto il **learning rate (*tasso di apprendimento*)** η, utile per stabilire la velocità di apprendimento della rete. Si ottiene quindi:

$$w_{ij} \leftarrow w_{ij} - \eta \cdot (y_j - t_j) \times c_i$$

In pratica η decide quanto cambiare il peso (e se si vuole trascurare il parametro basta porre $\eta = 1$). L'uso di tale parametro migliora la stabilità della

rete neurale che non avrà cambi di peso eccessivi, anche se questo comporta tempi di apprendimento più estesi. Tipicamente si ha che:

$$0.1 \leq \eta \leq 0.4$$

Si ha che ad ogni iterazione ci si aspetta un miglioramento della *rete neurale* (e questo miglioramento è dimostrabile). Viene imposto quindi un limite T di iterazioni entro le quali interrompere l'apprendimento anche se non si è arrivati al risultato corretto.
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 (valore 1 in output) o altrimenti (valore 0 in output). Nel nostro caso le classi sono discriminabili.

Teorema 4 (Teorema di 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*

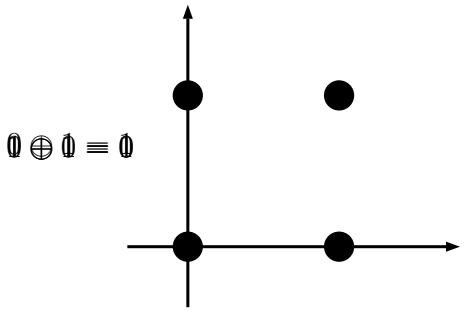
Teorema 5 (Teorema di Minsky e Papert)**.** *La classe delle forme discriminabili da un percettrone semplice è limitata alle forme linearmente separabili*

In base a questo si distinguerà in:

- **addestramento separabile**, quando si ha un iperpiano che separa le istanze positive e negative
- **addestramento non separabile**

Per capire il discorso della **separabilità** vediamo un esempio.

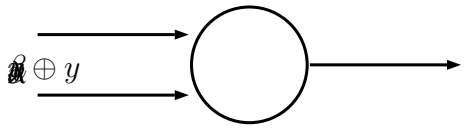
Esempio 21. *Vediamo un esempio che tratta l'addizione binaria, ovvero l'or esclusivo. Le possibili istanze sono rappresentate nel piano:*



Ho quindi che nessuna retta potrà separare tutti i punti e quindi i punti a valore 1 ***non sono linearmente separabili*** *da quelli a valore 0.*
Si vorrebbe cercare un neurone binario a soglia tale che:

$$x \oplus y = 1 \iff \alpha \cdot x + \beta \cdot y \geq \lambda$$

Tale neurone si rappresenterebbe con:



ma appunto tale neurone non può esistere in quanto non potrei mai separare i punti a valori 1 e quelli a valore 0 con una retta. Posso infatti separare o singoli punti o coppie di punti a valore opposto o triple di punti ma non potrò mai avere separati insiemi con tutti gli 0 e tutti gli 1.

Teorema 6. *Se l'insieme degli input estesi è partito in due classi linearmente separabili allora:*

$$\exists\, W \;\; t.c \begin{cases} 1 & se \;\; \sum w_i \cdot c_i \geq 0 \\ 0 & se \;\; \sum w_i \cdot c_i < 0 \end{cases}$$

con W ***vettore dei pesi****.*

Vediamo quindi l'algoritmo di learning per un percettrone, data la **funzione di transizione** per un input di cardinalità m e n neuroni:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij} \cdot c_i\right) = \begin{cases} 1 & \text{se } \sum_{i=0}^{m} w_{ij} \cdot c_i > 0 \\ 0 & \text{se } \sum_{i=0}^{m} w_{ij} \cdot c_i \leq 0 \end{cases}$$

Vediamo quindi l'algoritmo di apprendimento del percettrone:

Algorithm 5 Algoritmo di learning del percettrone

function PERCEPTRON-LEARNING

 # Inizializzazione

si imposta tutti i pesi w_{ij} a valori casuali piccoli (anche negativi)

 # Training

for *T iterazioni* ***o*** *fino a che tutti gli output non sono corretti* **do**

for *ogni vettore input* **do**

 # calcolo l'attivazione di ogni neurone j

 # tramite la funzione di transizione g:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij} \cdot c_i\right) = \begin{cases} 1 & \text{se } \sum_{i=0}^{m} w_{ij} \cdot c_i > 0 \\ 0 & \text{se } \sum_{i=0}^{m} w_{ij} \cdot c_i \leq 0 \end{cases}$$

 # aggiorno ogni peso individualmente

 # nella prossima iterazione avrò nuovi pesi

$$w_{ij} \leftarrow w_{ij} - \eta \cdot (y_j - t_j) \times c_i$$

 # Recall

 # ricalcolo l'attivazione di ogni neurone j tramite:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij} \cdot c_i\right) = \begin{cases} 1 & \text{se } w_{ij} \cdot c_i > 0 \\ 0 & \text{se } w_{ij} \cdot c_i \leq 0 \end{cases}$$

La complessità dell'algoritmo è $O(Tnm)$.
Si può dimostrare, tramite il **teorema della convergenza del percettrone** che dopo β modifiche di peso il percettrone classifica correttamente ogni input anche se tale β non è il reale numero di stadi e dipende dall'output.
L'algoritmo di apprendimento del percettrone quindi *converge* alla classificazione corretta quindi se:

- i dati di training sono linearmente separabili
- η è abbastanza piccolo

Teorema di convergenza del percettrone

Per completezza vengono aggiunte le note finali sul teorema presenti nelle slide.

Teorema 7. *Se l'insieme degli input estesi è partito in due classi linearmente separabili A e B allora é possibile trovare un vettore di pesi w tale che:*

$$\begin{cases} \langle w, x \rangle \geq 0 & \text{se } x \in A \\ \langle w, x \rangle < 0 & \text{se } x \in B \end{cases}$$

Quindi:

- *si parte con w arbitrario*
- *si classifica un punto x:*
  - *se la risposta è corretta si ha $w' \leftarrow w$*
  - *se la risposta è errata:*

$$\begin{cases} w' \leftarrow w + x & se\ x \in A \\ w' \leftarrow w - x & se\ x \in B \end{cases}$$

Dimostrazione. Vediamo la prova della correttezza del teorema.
Se si ha $x \in A$ allora si ha $\langle w, x\rangle < 0$ infatti, poiché:

$$\langle x, x\rangle \geq 0$$

si ha che:

$$\langle w', x\rangle = \langle (w + x), x\rangle = \langle w, x\rangle + \langle x, x\rangle > \langle w, x\rangle$$

Quindi w' classifica x in modo più corretto di w anche se altri input possono essere classificati "meno correttamente". □

Dimostrazione. Passiamo ora alla convergenza.
Consideriamo quindi $A' = A \cup B'$ con:

$$B' = \{-x |\, x \in B\}$$

e cerchiamo un v tale che:

$$\langle v, x\rangle \geq 0,\ \forall\, x \in A'$$

Si hanno quindi:

- la **sequenza di addestramento** $\{x_i\}_{i\in N}$ dove $x_i \in A'$ e quindi la sequenza ontiene infiniti elementi sia di A che di B'
- la **sequenza di pesi** $\{w_i\}_{i\in N}$ dove si parte arbitrariamente con $w_0 = 0$ e si procede con:

$$\begin{cases} w_{k+1} \leftarrow w_k & \text{se } \langle w_k, x_k\rangle \geq 0 \\ w_{k+1} \leftarrow w_k + x_k & \text{altrimenti} \end{cases}$$

Si hanno quindi:

- la **sequenza di pesi modificata** $\{v_i\}_{i\in N}$

- la **sottosequenza di training corrispondente** $\{v_i\}_{i \in N}$

Di modo che le coppie (v_i, t_i) corrispondano ai w_j tali per cui $w_j \neq w_{j+1}$. Ad esempio per:

$$w_0 \neq w_1 = w_2 = w_3 \neq w_4 = w_5 = w_6 \neq w_7 \neq w_8 \cdots$$

avremo:

- (v_0, t_0) associati a w_0
- (v_1, t_1) associati a w_3
- (v_2, t_2) associati a w_6
- (v_3, t_3) associati a w_7

Avendo:

$$\langle v_i, t_i \rangle < 0 \ \ \forall i$$

e avendo:

$$v_{i+1} = v_i + t_i = v_{i-1} + t_{i-1} + y_i = \sum_{k=0}^{i} t_i$$

e quindi la sequenza $\{v_i\}$ è **finita**. □

Dimostrazione. Uniamo matematicamente in una sola dimostrazione.
Sia w una qualsiasi soluzioni che sappiamo esistere per ipotesi. Si ha che:

$$\langle v, x \rangle \geq 0, \ \forall\, x \in A'$$

Pongo quindi:

$$\alpha = \min(\langle x, w \rangle, \ \forall\, x \in A')$$

sapendo che:

$$v_{i+1} = v_i + t_i = v_{i-1} + t_{i-1} + y_i = \sum_{k=0}^{i} t_i$$

ho che:

$$\langle v_{i+1}, w \rangle = \langle \left(\sum_{k=0}^{i} t_k \right), w \rangle \geq (i+1) \cdot \alpha$$

ma usando il teorema di **Cauchy-Schwarz** si ha che:

$$(\langle v_{i+1}, w \rangle)^2 \leq \langle |v_{i+1}|^2, |w|^2 \rangle$$

ottenendo che:

$$|v_{i+1}|^2 \geq \frac{(i+1)^2 \cdot \alpha^2}{|w|^2}$$

Pongo quindi:

$$M = \max\{|x|^2|,\ x \in A'\}$$

si ha che, avendo $\langle v_1, t_i \rangle < 0$:

$$|v_{i+1}|^2 = |v_i + t_i|^2 = |v_i|^2 + 2 \cdot \langle v_i, t_i \rangle + |t_i|^2 \leq |v_i|^2 + |t_i|^2$$

Si ha, di conseguenza:

$$|v_{i+1}|^2 \leq \sum_{k=1}^{i} |t_i|^2 \leq i \cdot M$$

Unendo quest'ultima con $|v_{i+1}|^2 \geq \frac{(i+1)^2 \cdot \alpha^2}{|w|^2}$ si ha che:

$$f(i) = \frac{j^2 \cdot \alpha^2}{|w|^2} \leq |v_{i+1}|^2 \leq i \cdot M = g(i)$$

avendo che $\frac{j^2 \cdot \alpha^2}{|w|^2}$ è quadratico in i e $i \cdot M$ lineare in i.
Si ottiene quindi che:

$$i \leq \frac{M \cdot |w|^2}{\alpha^2} = \beta$$

Quindi dopo β modifiche di peso il percettrone classifica correttamente ogni input. □

Esercitazione sul percettrone

Il percettrone si basa su un'unica unità di calcolo e tramite una funzione soglia valorizza l'uscita in modo da farle assumere un valore booleano in $\{-1, 1\}$.
Si hanno vari pesi w_i che scalano l'ingresso e, insieme agli input x_i (compreso l'input bias), formano il **potenziale di attivazione**:

$$\sum_{i=0}^{n} w_i x_i = \langle \overline{w}, \overline{x} \rangle$$

che verrà poi fatto passare per la funzione soglia. Se il potenziale è positivo si avrà uno in uscita, se negativo o nullo -1:

$$output = \begin{cases} 1 & \text{se } \sum w_i \cdot x_i > 0 \\ -1 & \text{se } \sum w_i \cdot x_i \leq 0 \end{cases}$$

In letteratura si hanno modelli dove il potenziale nullo comporta una terza uscita pari a 0.
Il percettrone basa il suo algoritmo di training su un ciclo che aggiorna di volta in volta i pesi tramite. Per il percettrone si ha quindi una funzione di aggregazione del tipo:

$$z = b + \sum_{i=1}^{n} w_i x_i$$

(con b valore del nodo bias)
che viene passata alla funzione soglia:

$$f(z) = \hat{y}$$

calcolando l'output:

$$\hat{y} \in \{-1, 1\}$$

Per aggiornare i pesi si calcola l'errore (la differenza tra il valore calcolato $\hat{y}$ e la label associata all'esempio in analisi y):

$$error = y - \hat{y}$$

calcola la correzione del peso, tramite la fattorizzazione dei pesi:

$$\Delta w = \eta(error)x_k$$

e aggiornando poi i pesi:

$$w_{k+1} = w_k + \Delta w_k$$

Nel caso di Adaline si ha una struttura in realtà simile ma utilizza una valutazione nel continuo degli errori, ad esempio tramite una certa funzione lineare. La funzione prende prende in input il potenziale di attivazione e la cui uscita viene valutata per il calcolo dell'errore δ tramite la δ **rule** (δ che ha valore continuo e non booleano). Si ha comunque una funzione di uscita a soglia che funziona come per il percettrone standard, se positivo il risultato restituisce 1 altrimenti -1.
Per Adaline si ha quindi una funzione di aggregazione del tipo:

$$\hat{y} = b + \sum_{i=1}^{n} w_i x_i$$

(con b valore del nodo bias)
che viene passata alla funzione soglia:

$$f(\hat{y})$$

per produrre l'uscita booleana (uguale al percettrone):

$$\hat{y}' \in \{-1, 1\}$$

per l'update dell'errore, a partire dalla funzione di aggregazione, si procede circa come per il percettrone standard, avendo però un errore quadratico:

$$error = (y - \hat{y})^2$$

e avendo poi l'update tramite:

$$w_{k+1} = w_k - \eta(2error_k)x_k$$

Riassumendo abbiamo l'algoritmo del percettrone che, per ogni esempio $\langle x, t \rangle$ con t label, esegue:

1. $y = step(f(x))$ (con step che da il booleano a seconda del segno)
2. $w = w + \eta(t - y)x$

Si ha che:

- se l'output matcha il target allora $t - y$ è nullo e non si aggiornano i pesi
- se l'output è 1 ma il target è 0 l'input, scalato con learning rate η, viene sottratto dal vettore dei pesi
- se l'output è 0 ma il target è 1 l'input, scalato con learning rate η, viene aggiunto al vettore dei pesi

Riassumendo abbiamo l'algoritmo del Adaline che, per ogni esempio $\langle x, t \rangle$ con t label, esegue:

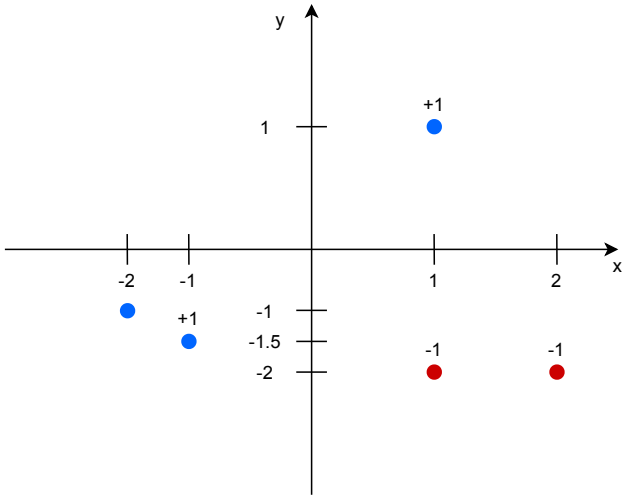
1. $y = f(x)$
2. $w = w + \eta(t - y)x$

Si ha che questo equivale la regola del *gradient descent* della regressione lineare (con errore quadratico), infatti la derivata di $(y - t)^2$ è $2(z - t)$. Il fattore costante 2 può essere omesso dato che si usa il learning rate per rimodulare quanto aggiornare i pesi.
Ovviamente entrambi possono studiare solo set di punti separabili linearmente, usando iperpiani e non rette se saliamo di grado su $\mathbb{R}$, per separare i punti.

Esercizio 5. *Prendiamo il seguente training set con $t(x)$ che ci fornisce la label:*

	x	y	$t(x)$
x_1	1	1	1
x_2	2	-2	-1
x_3	-1	-1.5	1
x_4	1	-2	-1
x_5	-2	-1	1

graficamente possiamo rappresentare tali punti (in blu con target $+1$ e rosso per target -1, notando che sono linearmente separabili):



Usiamo quindi l'algoritmo del percettrone e studiamo come si aggiornano i pesi. Si impone $\eta = 0.2$ e nodo bias di valore $b = 1$ e peso w_0, arbitrariamente inizializzato a 0.
Partiamo con il primo esempio. Si hanno:

- $x_1 = (1, 1)$
- $t(x_1) = 1$

da ciò si ricava che l'input del percettrone, il suo segnale d'ingresso, altro non è che, considerando il nodo bias:

$$\overline{x} = (b, x_1) = (1, 1, 1)$$

Arbitrariamente inizializziamo i pesi a (avendo già detto che il bias ha peso 0):

$$\overline{w} = (0, 1, 0.5)$$

Faccio quindi il prodotto scalare tra il vettore input e il vettore peso:

$$\langle \overline{w}, \overline{x} \rangle = \begin{pmatrix} 0 & 1 & 0.5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 0 + 1 + 0.5 = 1.5$$

Si ha quindi $\langle \overline{w}, \overline{x} \rangle > 0$ *e quindi:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = +1$$

e avendo che, ricordando $t(x_1) = +1$*:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = t(x_1)$$

i pesi $\overline{w}$ *non devono essere aggiornati.*
Si passa al secondo esempio.
Si ha (senza dover rispiegare gli step metto tutto nella stessa lista):

- $x_2 = (2, -2)$
- $t(x_2) = -1$
- $\overline{x} = (1, 2, -2)$
- $\overline{w} = (0, 1, 0.5)$

calcolo:

$$\langle \overline{w}, \overline{x} \rangle = \begin{pmatrix} 0 & 1 & 0.5 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ -2 \end{pmatrix} = 0 + 2 - 1 = 1$$

Si ha quindi $\langle \overline{w}, \overline{x} \rangle > 0$ *e quindi:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = +1$$

e avendo che, ricordando $t(x_2) = -1$*:*

$$sign(\langle \overline{w}, \overline{x} \rangle) \neq t(x_2)$$

Bisogna procedere all'update dei pesi, che ricordiamo essere:

$$\overline{w}_{new} = \overline{w}_{old} + \eta(y - \hat{y})\overline{x}$$

si ha quindi, avendo $\hat{y} = 1, y = -1$ *e quindi* $y - \hat{y} = -2$*:*

- $\overline{w}_{new}[0] = \overline{w}_{old}[0] + 0.2 \cdot (-2) \cdot x[0] = 0 + 0.2 \cdot (-2) \cdot 1 = -0.4$
- $\overline{w}_{new}[1] = \overline{w}_{old}[1] + 0.2 \cdot (-2) \cdot x[1] = 1 + 0.2 \cdot (-2) \cdot 2 = 0.2$
- $\overline{w}_{new}[2] = \overline{w}_{old}[2] + 0.2 \cdot (-2) \cdot x[2] = 0.5 + 0.2 \cdot (-2) \cdot (-2) = 1.3$

ottenendo quindi come nuovo vettore dei pesi:

$$\overline{w} = (-0.4, 0.2, 1.3)$$

Passo quindi al terzo esempio:

- $x_3 = (-1, -1.5)$
- $t(x_3) = -1$
- $\overline{x} = (1, -1, -1.5)$
- $\overline{w} = (-0.4, 0.2, 1.3)$

calcolo:

$$\langle \overline{w}, \overline{x} \rangle = \begin{pmatrix} -0.4 & 0.2 & 1.3 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1.5 \end{pmatrix} = -0.4 - 0.2 - 1.95 = -2.55$$

Si ha quindi $\langle \overline{w}, \overline{x} \rangle \leq 0$ *e quindi:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = -1$$

e avendo che, ricordando $t(x_3) = +1$*:*

$$sign(\langle \overline{w}, \overline{x} \rangle) \neq t(x_3)$$

Bisogna procedere all'update dei pesi, avendo $\hat{y} = -1, y = 1$ *e quindi* $y - \hat{y} = 2$*:*

- $\overline{w}_{new}[0] = \overline{w}_{old}[0] + 0.2 \cdot 2 \cdot x[0] = -0.4 + 0.2 \cdot 2 \cdot 1 = 0$
- $\overline{w}_{new}[1] = \overline{w}_{old}[1] + 0.2 \cdot 2 \cdot x[1] = 0.2 + 0.2 \cdot 2 \cdot (-1) = -0.2$
- $\overline{w}_{new}[2] = \overline{w}_{old}[2] + 0.2 \cdot 2 \cdot x[2] = 1.3 + 0.2 \cdot 2 \cdot (-1.5) = 0.7$

ottenendo quindi come nuovo vettore dei pesi:

$$\overline{w} = (0, -0.2, 0.7)$$

Passo quindi al quarto esempio:

- $x_4 = (1, -2)$
- $t(x_4) = -1$
- $\overline{x} = (1, 1, -2)$
- $\overline{w} = (0, -0.2, 0.7)$

calcolo:

$$\langle \overline{w}, \overline{x} \rangle = \begin{pmatrix} 0 & -0.2 & 0.7 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix} = 0 - 0.2 - 1.4 = -1.6$$

Si ha quindi $\langle \overline{w}, \overline{x} \rangle \leq 0$ *e quindi:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = -1$$

e avendo che, ricordando $t(x_4) = -1$*:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = t(x_4)$$

E quindi non devo aggiornare i pesi.
Passo quindi al quinto esempio:

- $x_5 = (-2, -1)$
- $t(x_5) = 1$
- $\overline{x} = (1, -2, -1)$
- $\overline{w} = (0, -0.2, 0.7)$

calcolo:

$$\langle \overline{w}, \overline{x} \rangle = \begin{pmatrix} 0 & -0.2 & 0.7 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix} = 0 + 0.4 - 0.7 = -0.3$$

Si ha quindi $\langle \overline{w}, \overline{x} \rangle \leq 0$ *e quindi:*

$$sign(\langle \overline{w}, \overline{x} \rangle) = -1$$

e avendo che, ricordando $t(x_5) = +1$*:*

$$sign(\langle \overline{w}, \overline{x} \rangle) \neq t(x_5)$$

Bisogna procedere all'update dei pesi, avendo $\hat{y} = -1, y = 1$ *e quindi* $y - \hat{y} = 2$*:*

- $\overline{w}_{new}[0] = \overline{w}_{old}[0] + 0.2 \cdot 2 \cdot x[0] = 0 + 0.2 \cdot 2 \cdot 1 = 0.4$
- $\overline{w}_{new}[1] = \overline{w}_{old}[1] + 0.2 \cdot 2 \cdot x[1] = 0.2 + 0.2 \cdot 2 \cdot (-2) = -1$
- $\overline{w}_{new}[2] = \overline{w}_{old}[2] + 0.2 \cdot 2 \cdot x[2] = 0.7 + 0.2 \cdot 2 \cdot (-1) = 0.3$

ottenendo quindi come nuovo vettore dei pesi:

$$\overline{w} = (0.4, -1, 0.3)$$

che, avendo terminato l'esercizio, è il vettore pesi finale usato dall'algoritmo.

3.3.2 Discesa lungo il gradiente

Le formule più complesse sono solo di bellezza.
Fino ad ora il singolo neurone rappresentava un singolo iperpiano che veniva "spostato" per dividere le istanze positive e quelle negative.
Ora introduciamo che ad ogni passo di aggiornamento si aggiornano i pesi e se, mano a mano che cambio i pesi, misuro l'errore sul dataset del mio sistema. Si può quindi far "scendere" questo errore e ci si augura che ci sia una costante di discesa dell'errore. L'errore complessivo sul dataset D è calcolato come:

$$E[w_0, \ldots, w_n] = \frac{1}{2} \sum_{d \in D} (t_d - y_d)^2$$

Si calcola il gradiente della curva degli errori e si punta ad essere sempre in "discesa" lungo la curva dell'errore, aggiustando in modo opportuno i pesi.
La strategia di apprendimento sarà quella di minimizzare una opportuna funzione dei pesi w_i. Mi devo spostare verso un punto di minimo, scendendo nel grafico della funzione costruita tra pesi ed errore complessivo (quindi anche in più dimensioni).
A livello di conti si ha che:

$$\nabla E[w] = \left[\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

e quindi aggiornando i pesi come:

$$w' = w + \delta w = w - \eta \nabla E[w]$$

Si ha quindi:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \frac{\partial}{\partial w_i} \cdot \frac{1}{2} \sum_d (t_d - y_d)^2$$

$$= -\eta \frac{\partial}{\partial w_i} \cdot \frac{1}{2} \sum_d (t_d - \sum_i w_i \cdot x_{di})^2 = -\eta \sum_d (t_d - y_d) \cdot (-x_i)$$

Si vuole addestrare quindi cambiando costantemente la sequenza di pesi del vettore di input $\langle (x_1, \dots x_n), t \rangle$, con t output corrispondente.
w_i è inizializzato con un valore piccolo e si ha l'algoritmo:

Algorithm 6 Algoritmo di discesa lungo il gradiente

function GRAD

inizializzo ogni Δw_i *a* 0

for *ogni input* $\langle (x_1, \dots x_n), t \rangle$ **do**

invio l'input $(x_1, \dots x_n)$ *all'unità lineare e calcola l'output* y

aggiorno la variazione dei pesi:

$$\Delta w_i = \Delta w_i + \eta \cdot (t - y) \cdot x_i$$

aggiorno i pesi:

$$w_i = w_i + \Delta w_i$$

Abbiamo una **modalità Batch**, quindi computazionalmente costosa:

$$w = w - \eta \cdot \nabla E_D[W]$$

Si può avere una **modalità incrementale**:

$$w = w - \eta \cdot \nabla E_d[W]$$

calcolata sui singoli esempi d:

$$E_d[w] = \frac{1}{2}(t_d - y_d)^2$$

La discesa lungo il gradiente incrementale può approssimare la discesa lungo il gradiente 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.

3.3.3 Reti multistrato

Le formule più complesse sono solo di bellezza.
Passiamo quindi dal percettrone ad una rete a due strati, cambiando quindi

la gestione dei pesi.
Devo avere sempre un gradiente dei pesi, che ora avranno doppio indice per indicare anche l'unità di riferimento, che scende:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} = -(t_j - y_j) \cdot \frac{\partial y_j}{\partial w_{ij}} = -\delta_j \cdot \frac{\partial \sum_i w_{ij} \cdot x_j}{\partial w_{ij}} = -\delta_j \cdot x_i$$

Si ha quindi:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot x_i$$

detta **regola delta**, che è l'evoluzione di quanto detto per il percettrone in merito alla variazione dei pesi.
Si ha che la convergenza a un minimo globale é garantita per funzioni di attivazione lineari senza unità nascoste e per dati consistenti.
Si introduce una nuova funzione di attivazione, detta **sigmoide**, che comporta l'**unità sigmoide**. La funzione sigmoide è:

$$y = \sigma(net) = \frac{1}{1 + e^{-net}}$$

Tale funzione è derivabile, infatti:

$$\mathrm{d}\sigma(x) = \sigma(x) \cdot (1 - \sigma(x))$$

riprendendo la figura 3.1 si ha quindi in primis l'aggiunta del nodo bias e poi il cambio della funzione a scalino con il **sigmoide**. L'output invece non sarà più binario ma un certo y.
Si hanno quindi in primis le **reti multistrato feedforward** con le seguenti caratteristiche:

- si hanno strati intermedi tra input e output
- si hanno connessioni da strati di livello basso a strati di livello alto, solitamente mono direzionali
- non si hanno all'interno di uno stesso strato
- il neurone ha uno stato booleano $x \in \{0, 1\}$
- si ha la seguente funzione di transizione:

  $$x_k = \sigma\left(\sum_j w_{jk} \cdot x_j\right)$$

  con x_j che può essere in alcuni casi un input istanza e in altri lo stato di altri neuroni, a seconda dell'altezza del livello in cui mi trovo

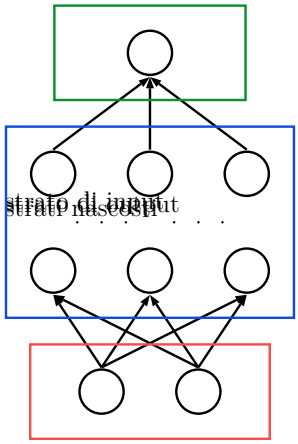


Figura 3.2: Rappresentazione stilizzata di rete multistrato

- per ogni configurazione x del primo strato (ingresso), la rete calcola una configurazione y dell'ultimo strato (uscita). Normalmente avremo uno strato nascosto più grande (poco) di quello d'uscita (anche se non si ha una regola per dimensionare lo strato nascosto). Raramente useremo più strati nascosti

Quindi fissata una mappa f tra input e output, sulla base degli stimoli x_i, la rete cambia i pesi in modo che dopo un numero di passi s si abbia l'output y_k tale che $f(x_k) = y_k, \forall\, k > s$ (almeno approssimativamente). Per la modifica bisogna minimizzare un c**riterio di discrepanza** tra risposta della rete e risposta desiderata.
In questo modo potremmo anche risolvere il problema dell'*or esclusivo*, aggiungendo uno strato nascosto.
Viene aumentata la potenza rispetto al percettrone, permettendo una classificazione **altamente non lineare**.
Si hanno quindi $u_1, \ldots, u_n$ neuroni divisi in:

- unità di input
- unità nascoste
- unità di output

Si hanno inoltre:

- pesi $w_j j$ per ogni coppia che voglio connettere

- stati di attivazione $s_j \in \mathbb{R}$
- input netto a u_j: $n_i = \sum_{i=0}^{n} w_{ij} \cdot s_i$
- funzione di transizione sigmoide:

$$s_j(t+1) = \frac{1}{1 + e^{-n_i(t)}}$$

Lo stato di uscita è determinato da una serie di strati profondi. Dato un input x, un output target t e un output effettivo y abbiamo la solita forma quadratica:

$$E = \frac{1}{2} \sum_j (t_j - y_j)^2$$

che dipende anche dagli strati nascosti. In ogni caso si ha:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

poiché:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} \cdot \frac{\partial n_j}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} \cdot s_j = (def) - \delta_j \cdot s_j$$

e si cerca:

$$\delta_j = -\frac{\partial E}{\partial n_j}$$

Abbiamo quindi l'**algoritmo di retropropagazione** 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:
$$s_j = f_j(n_j)$$
3. **confronto**: per ogni neurone di output u_j, noto l'output atteso t_j, si calcola:
$$\delta_j = f_j(n_j) \cdot (t_j - y_j))$$
4. **retropropagazione dell'errore**: per ogni neurone nascosto u_j, si calcola:
$$\delta_j = f_j(n_j) \cdot \left(\sum_h w_{ih} \cdot \delta_h \right)$$

5. **aggiornamento dei pesi**: si ha:

$$w_{ij} = w_{ij} + \eta \cdot \delta_i \cdot s_j$$

Vediamo quindi l'algoritmo:

Algorithm 7 Algoritmo di retropropagazione

function RET-PROG
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**

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

for *ogni unità nascosta* h **do**

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

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

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

aggiorno i pesi:

$$w_i = w_i + \Delta w_i$$

Questa tecnica si generalizza facilmente a grafi orientati.
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:

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

in modo da avere una sorta di *inerzia* per la variazione dei pesi.
Ci sono comunque modelli per uscire dai minimi locali (usando alcune tecniche di *ricerca operativa*).
Si minimizzano gli errori sugli esempi di training m,a si rischia l'**overfitting**.
Tutti questi fattori comportano un addestramento lento ma dopo l'addestramento si ha una rete veloce.
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 avere varianti al 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

*Le reti **feedforward** sono state usate in progetti di guida autonoma come **ALVINN**.*
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 sono meno conoscibili dopo l'esecuzione
- miglior accuratezza (?)

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

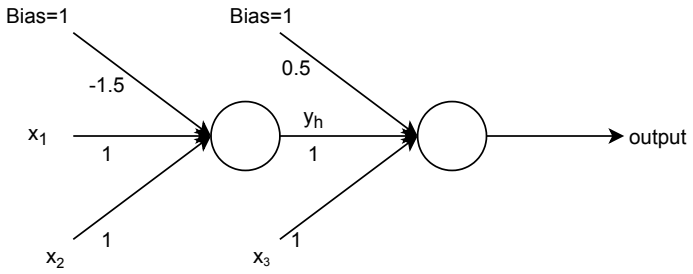
Esercitazione reti neurali multistrato

Esercizio 6. *vengono date le istanze:*

$$z_1 = (0, 2, 2)$$

$$z_2 = 1, 0, 0$$

Con la seguente architettura:



con quindi una terza istanza valutata solo nel secondo livello.
La funzione f di attivazione del neurone è definita come quella del percettrone, ovvero:

$$sgn(x) = \begin{cases} 1 & se\ x \geq 0 \\ -1 & altrimenti \end{cases}$$

L'esercizio propone di capire la label assegnata a z_1 e la label assegnata a z_2. Nel dettaglio, per entrambe le istanze, abbiamo (x_1, x_2, x_3), ovvero i primi due valori sono (insieme al bias) l'input del primo layer mentre il terzo (insieme al bias e al risultato del primo layer) è l'input del secondo layer.
Partiamo con $x_1 = (0, 2, 2)$.
Si hanno i due layer:

1. *nel layer 1 si ha:*

$$\langle \overline{x}, \overline{w} \rangle = \begin{pmatrix} 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} -1.5 \\ 1 \\ 1 \end{pmatrix} = -1.5 + 0 + 2 = 0.5$$

Si ha quindi:

$$\langle \overline{x}, \overline{w} \rangle \geq 0$$

e quindi:

$$sgn(\langle \overline{x}, \overline{w} \rangle) = sgn(0.5) = 1$$

avendo quindi:

$$y_h = 1$$

che sarà tra gli input del secondo layer

2. *valuto quindi il layer 2:*

$$\langle \overline{x}, \overline{w} \rangle = \begin{pmatrix} 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} -0.5 \\ 1 \\ 1 \end{pmatrix} = -0.5 + 1 + 2 = 2.5$$

Si ha quindi:

$$\langle \overline{x}, \overline{w} \rangle \geq 0$$

e quindi:

$$sgn(\langle \overline{x}, \overline{w} \rangle) = sgn(2.5) = 1$$

avendo quindi:

$$output = 1$$

Possiamo quindi dire che per la prima istanza:

$$f(x_1) = +1$$

Passiamo a $x_2 = (1, 0, 0)$.
Si hanno i due layer:

1. *nel layer 1 si ha:*

$$\langle \overline{x}, \overline{w} \rangle = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} -1.5 \\ 1 \\ 1 \end{pmatrix} = -1.5 + 1 + 0 = -0.5$$

Si ha quindi:

$$\langle \overline{x}, \overline{w} \rangle < 0$$

e quindi:

$$sgn(\langle \overline{x}, \overline{w} \rangle) = sgn(-0.5) = -1$$

avendo quindi:

$$y_h = -1$$

che sarà tra gli input del secondo layer

2. *valuto quindi il layer 2:*

$$\langle \overline{x}, \overline{w} \rangle = \begin{pmatrix} 1 & -1 & 0 \end{pmatrix} \begin{pmatrix} -0.5 \\ 1 \\ 1 \end{pmatrix} = -0.5 - 1 + 0 = -1.5$$

Si ha quindi:

$$\langle \overline{x}, \overline{w} \rangle < 0$$

e quindi:

$$sgn(\langle \overline{x}, \overline{w} \rangle) = sgn(-1.5) = -1$$

avendo quindi:

$$output = -1$$

Possiamo quindi dire che per la prima istanza:

$$f(x_2) = -1$$

Abbiamo quindi classificato la prima istanza come positiva e la seconda come negativa.

3.4 Support Vector Machines

Parliamo ora delle **Support Vector Machines (*SVM*)**.
Se il percettrone semplice impara solo funzioni di separazione lineari, quello multistrato anche funzioni di separazione non lineari complesse (ma con difficoltà di addestramento avendo molti minimi locali e tanti pesi) le SVM presentano un algoritmo di apprendimento efficiente e imparano funzioni di separazione non lineari complesse.
Viene quindi ripreso comunque il concetto di *separazione lineare* ma con una scelta efficiente dell'iperpiano, trovando il **miglior iperpiano separatore**, per classificare un insieme di punti linearmente separabili, trovando un vettore di pesi W per separare bene le istanze. Si usa la cosiddetto **teoria statistica dell'apprendimento** che 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 sii massima ampiezza. Tale ampiezza è detta margine (volendo che sia massima la distanza minima tra tutti i punti). SVM usa questa teoria per trovare l'iperpiano "migliore" in base a questi calcoli probabilistici.
Non si parla quindi più di neuroni.
e riuscissimo a separare i dati con un largo margine avremmo ragione di credere che il classificatore (ovvero l'iperpiano stesso) sia "più robusto" tanto da avere una migliore generalizzazione (assunto che i punti siano generati da una certa regola). 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 8 (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 VC 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 che prevede che, preso un insieme di punti di training:

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

ad ogni vettore x_i associo la classe di appartenenza y_i (con un etichetta non booleana):

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

Ho i punti linearmente separabili:

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

(quindi l'iperpiano separa positivi e negativi)
che scritto in un solo vincolo diventa:

$$y_i(\langle w, x_i \rangle + b) > 0, \;\; i = 1, \ldots, n$$

Si ha che w mi dirà quanto è inclinato il piano mentre b è la distanza tra l'origine e il piano (quindi queste due variabili identificano i vari iperpiani possibili tra cui cercare il "migliore" ovvero quello che separa meglio punti positivi e negativi).
Ci si sposta quindi dalla geometria visualizzabile all'algebra lineare.
L'ipotesi quindi tra w e b è una funzione che prende il segno di $< langlew, x\rangle + b$ per associare l'etichetta:

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

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

- margine funzionale
- margine geometrico

Definizione 14. *Definiamo il* ***margine funzionale*** *di un punto* (x_i, y_i) *rispetto all'iperpiano* (w, b) *come:*

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

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

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

Si hanno quindi due casistiche:

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

Teorema 9. *Se* $\hat{\gamma}_i > 0, \forall$ *classificazionei 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 probabilità sulla qualità della previsione anche se l'uso del solo $\hat{\gamma}$ può essere problematico in quanto il margine funzionale non è invariante rispetto ad un iperpiano "riscalato". Analizziamo meglio quest'aspetto.
Per come è stato scelto il classificatore f se si scala l'iperpiano:

$$(w, b) \to (c \cdot w, b)$$

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 ora studiare la distanza d del punto x dall'iperpiano. Si ha quindi:

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

Quindi:

Definizione 15. *Definiamo il* ***margine geometrico*** *di un punto* (x_i, y_i) *rispetto all'iperpiano* (w, b) *come:*

$$\gamma_i = \frac{y_i \cdot (\langle w, x_i \rangle + b)}{\|w\|}$$

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

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

Teorema 10. *Se* $\gamma_i > 0, \forall$ *classificazionei la classificazione è approvata analogamente a quanto detto per il margine funzionale.*

ato un punto positivo/negativo il margine geometrico rappresenta la sua distanza geometrica dall'iperpiano. Quindi il margine geometrico rende meglio l'idea della distanza di un punto sa un iperpiano in $\mathbb{R}^n$.
Inoltre si ha che **i margine geometrico è invariante rispetto alla scala di** w e quindi possiamo "riscalare" l'iperpiano senza cambiamenti, non variando il margine.
Dalla formula notiamo che, posto $\|w\| = 1$ si "riscala" l'iperpiano (w, b):

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

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 16. *Definiamo un* ***iperpiano canonico*** *se:*

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

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:

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

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

$$\max f(x)$$

$$s.t. \quad g(x) \leq 0$$
$$h(x) = 0$$

Vorremmo assicurarci che tutti i punti (sia quelli positivi sia quelli negativi) cadano al di fuori del margine e quindi, dato un certo γ vorremmo, $\forall\, i \in \{1, \ldots, n\}$:

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

e quindi vogliamo:

$$\max \gamma$$

$$s.t. \qquad y_i \cdot (\langle w, x_i \rangle + b) \leq \gamma$$
$$\|w\| = 1$$

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

$$\max \frac{\hat{\gamma}}{\|w\|}$$

$$s.t. \qquad y_i \cdot (\langle w, x_i \rangle + b) \leq \gamma$$

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:

$$\max \frac{1}{\|w\|}$$

$$s.t. \qquad y_i \cdot (\langle w, x_i \rangle + b) \leq \gamma$$

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 $\tau(w)$:

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

Teorema 11. *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:

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

e si ha che:

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

Ovvero scritta in termini di un sottoinsieme di esempi del training set (noto come vettori di supporto) che giacciono sul margine dell'iperpiano, 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:

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

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.
α viene elaborato dai vettori di supporto ma non verrà ora trattato.
Passiamo quindi oltre alla sola separabilità lineare.
Si usa lo stesso approccio rivedendo la formulazione tramite i **metodi kernel**. Si cerca di mappare lo spazio di input in un nuovo spazio, (in generale

di dimensione maggiore), in cui i punti siano linearmente separabili, per classificare mediante superfici non lineari.
Dobbiamo trovare una funzione:

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

chemappi 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:

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

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

Definizione 17. *Definiamo una* ***funzione kernel****. Data una trasformazione* $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ *una* ***funzione kernel*** *è una mappa:*

$$K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \ t.c. \ K(x, y) = \Phi(x) \cdot \Phi(y)$$

Definizione 18. *Si definisce quindi il* ***kernel trick*** *che consiste nel computare il prodotto interno delle trasformare di due vettori* x *e* y*, tramite* Φ*, senza computare le trasformate, semplificando il calcolo di:*

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

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

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

con:

$$K(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

Non si cercano tutte le possibili trasformazioni ma solo alcune.

Esempio 22. *Sia:*

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$

con rispettivamente (x, y) *e* (r, s, t)
tale che;

$$r = x^2, \ s = y^2, \ t = \sqrt{2xy}$$

presi $p_1 = (x_1, y_1), p_2 = (x_2, y_2 \in \mathbb{R}^2$ *ho il loro prodotto interno delle loro immagini in* $\mathbb{R}^3$*:*

$$(x_1^2, y_1^2, \sqrt{2x_1y_1}) \cdot [(x_2^2, y_2^2, \sqrt{2x_2y_2}) = (x_1x_2)^2 + (y_1y_2)^2 + 2x_1x_2x_2y_2$$

ma questo è il quadrato del prodotto interno tra p_1 *e* p_2*:*

$$[(x_1, y_1) + (x_2, y_2)]^2 = (x_1x_2 + y_1y_2)^2$$

non dovendo quindi calcolare immagini in $\mathbb{R}^3$

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 $\Phi(x)$

Esempio 23. *Si data la trasformazione:*

$$\Phi : \mathbb{R}^3 \to \mathbb{R}^{10}$$

tale che:

$$\Phi(x) = (1, \sqrt{2x_1}, \sqrt{2x_2}, \sqrt{2_3}, \sqrt{2x_1x_2}, \sqrt{2x_1x_3}, \sqrt{2_1x_3}, x_1^2, x_2^2, x_3^2)$$

ovvero i 10 possibili monomi fino al grado due ottenuti da tre variabili. Si ha che:

$$\Phi(x) \cdot \Phi(y) = 1 + 2 \cdot \sum_1^d x_iy_i + 2 \cdot \sum_1^d x_i^2y_i^2 + 2 \cdot \sum_{i,j} x_ix_jy_iy_j = (1 + xy)^2$$

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**:
$$K(x, y) = e^{-y\|x-y\|^2}$$
- **gaussian radial basis function**:
$$K(x, y) = e^{-\frac{(x-y)^2}{2\sigma^2}}$$
- **percettrone multistrato**:
$$K(x, y) = \tanh(b(x \cdot y) - c)$$

Teorema 12. *Un kernel definisce una matrice K_{ij} che è simmetrica e definita positiva*

Teorema 13 (teorema di Mercer)**.** *Ogni matrice simmetrica e definita positiva è un kernel*

SVM può essere applicato a problemi non binari tramite il **one vs rest**, avendo separazioni a più classi. Spesso posso ridurmi comunque a sottoproblemi binari, avendo più gruppi di punti linearmente separabili. SVM trova quindi tutte le possibili separazioni lineari. Combinando quindi i risultati ottenuti sui sottoproblemi binari derivati, risultati ottenuti con SVM, e definendo una logica di classificazione per il problema originale posso fare un'analisi multiclasse.
Si ha quindi l'addestramento di un singolo classificatore per classe, con i campioni di quella classe come campioni 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 $i = 1, \dots K$
- come output una lista di classificatori f_k con $l = 1, \dots K$

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

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

applicando poi L a (X, z) per ottenere f_k.
Quindi prendere decisioni significa applicare tutti i classificatori ad un nuovo sample e prevedere l'etichetta k er la quale il classificatore corrispondente riporta il punteggio di confidenza più alto:

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

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 il **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:

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

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.

Esercitazione su SVM e metodi kernel

Nell'esercitazione ogni $\vec{w} \cdot \vec{x}$ è da caloclarsi come $\vec{w}^T \cdot \vec{x}$.

Definizione 19. *Definiamo **iperpiano** su uno spazio euclideo a n dimensioni come un sottoinsieme piatto di $n-1$ dimensioni che divide lo spazio in due parti non connesse.*
È quindi un luogo di punti che soddisfa:

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

Ovviamente:

- *in una dimensione è un punto*
- *in due dimensioni una retta*
- *in tre dimensioni un piano*
- *in più di tre dimensioni si usa direttamente il termine iperpiano*

Nel caso a due dimensioni, per esempio, si ha che:

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

(che in due dimensioni è la forma implicita di una retta) corrisponde a:

$$w_0 x + w_1 y + b = 0$$

ovvero:

$$w_1 y = -w_0 x - b$$

Isolo quindi y:

$$y = -\frac{w_0}{w_1} x - \frac{b}{w_1}$$

e avendo:

- $m = -\frac{w_0}{w_1}$ *come coefficiente angolare*
- $q = -\frac{b}{w_1}$ *come ordinata origine della retta, il punto di intercetta*

ottengo la classica equazione della retta, in forma esplicita:

$$y = mx + q$$

Quindi $\vec{w} \cdot \vec{x} + b$ assegna una sorta di "punteggio".

Esempio 24. *Preso* $\vec{w} = (-0.4, -1)$ *e* $b = 9$ *si studino i punti* $A(1,3)$, $B(3,5)$ *e* $C(5,7)$.
Si ottiene:

- *per* A, *con* $\vec{x} = (1,3)$, *si ha* $\vec{w} \cdot \vec{x} + b = 5.6$
- *per* B, *con* $\vec{x} = (3,5)$, *si ha* $\vec{w} \cdot \vec{x} + b = 2.8$
- *per* C, *con* $\vec{x} = (5,7)$, *si ha* $\vec{w} \cdot \vec{x} + b = 0$

Avendo che C *giace sull'iperpiano (l'equazione è nulla) e gli altri punti, essendo in due dimensioni e quindi avendo a che fare con una retta, sono sotto la retta (*A *più lontano di* B *avendo "punteggio" maggiore).*

Con il percettrone si vedeva se i punti erano linearmente separabili (trovando un iperpiano che separava) con SVM si cerca l'iperpiano ottimo per separare i punti.

Esempio 25. *Sia data una funzione d'ipotesi (che definisce un'ipotesi in* H*) tale che:*

$$h(\vec{x}_i = \begin{cases} +1 & se\ \vec{w} \cdot \vec{x}_i + b \geq 0 \\ -1 & se\ \vec{w} \cdot \vec{x}_i + b < 0 \end{cases}$$

ovvero:

$$h(\vec{x}_i = sign(\vec{w} \cdot \vec{x}_i + b)$$

Quindi i valori positivi sono sopra etichettati come positivi e i negativi come negativi.
Siano dati:

- $\vec{w} = (0.4, 1)$
- $b = -9$
- $A(8,7)$ *e* $B(1,3)$

Si ha:

- *per* A, *che è sopra l'iperpiano:*

  $$\vec{w} \cdot \vec{x}_i + b = 0.4 \cdot 8 + 1 \cdot 7 - 9 = 1.2$$

  quindi è etichettato come positivo

- *per* B, *che è sotto l'iperpiano:*

  $$\vec{w} \cdot \vec{x}_i + b = 0.4 \cdot 1 + 1 \cdot 3 - 9 = -5.6$$

  quindi è etichettato come negativo

Ma non si è detto nulla sulla scelta dell'iperpiano, ne in termini di calcolo ne in termini studio della sua qualità.

Ci serve quindi una misura di confidenza per la classificazione usando i **margini**.
Abbiamo quindi il punteggio β dato da:

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

e perché tale punteggio sia utile ci serve anche la classe di appartenenza y.
Si ha quindi:

$$f = y \cdot \beta \implies f = y \cdot (\vec{w} \cdot \vec{x} + b)$$

avendo ottenendo il **margine funzionale** f tale che:

- f è positivo se la classificazione è corretta
- f è negativo se la classificazione non è corretta

Inoltre:

- se si ha $y = 1$ allora affinché il margine funzionale sia ampio (cioè, affinché la nostra previsione sia sicura e corretta), allora abbiamo bisogno che $\vec{w} \cdot \vec{x} + b$ sia un numero positivo grande
- se si ha $y = -1$ allora affinché il margine funzionale sia grande, allora abbiamo bisogno che $\vec{w} \cdot \vec{x} + b$ sia un grande numero negativo

Per una certa osservazione i si ha quindi (usiamo la dicitura *gamma* al posto di f per il margine funzionale):

$$\hat{\gamma}^{(i)} = y^{(i)}(\vec{w} \cdot \vec{x} + b)$$

e quindi per tutti i dati si cerca:

$$\hat{\gamma} = \min \hat{\gamma}^{(i)}$$

Ma si ha ancora un problema: dipende dalla scala. Se a $\vec{w}$ sostituiamo, ad esempio, $2\vec{w}$ e a b sostituiamo $2b$ si ha che la classificazione fatta con $sign(\vec{w} \cdot \vec{x}_i + b)$ resta la medesima (anche perché l'iperpiano resta lo stesso, avendo dipendenza lineare). D'altro canto il margine funzionale aumenta all'aumentare dei due valori (aumentando lo score) e quindi si ha una "falsa" confidenza (sembrando due volte più confidente quando è tutto uguale in realtà, dall'iperpiano alla classificazione).

Si introduce quindi il **margine geometrico**, dividendo il margine funzionale per la norma di $\vec{w}$:

$$\hat{\gamma}^{(i)} = y^{(i)} \left(\frac{\vec{w}}{\|\vec{w}\|} \cdot \vec{x}^{(i)} + \frac{b}{\|\vec{w}\|} \right)$$

si ha quindi un invaiante rispetto al fattore di scala, a differenza del margine funzionale (con la divisione per la norma semplifico eventuali fattori moltiplicativi costanti).
Anche in questo caso cerco il minimo:

$$\hat{\gamma} = \min \hat{\gamma}^{(i)}$$

Con SVM si cerca l'iperpiano ottimo ovvero che massimizza il margine per entrambe le classi considerate nel training set, rendendo massima la distanza tra i punti più vicini di ogni classe, minimizzando il margine geometrico di ogni esempio. Si trovano i support vector dove giacciono i più vicini punti all'iperpiano per le due classi. Il risultato dell'ottimizzazione per SVM è trovare il gap maggiore che ci separa da tutte le classi contemporaneamente

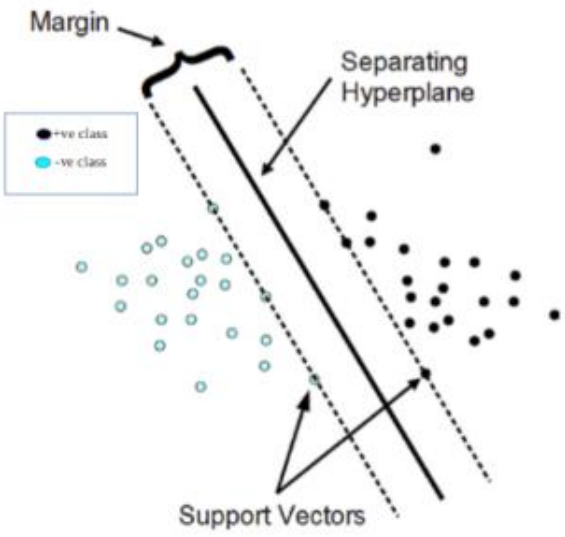


Figura 3.3: Visualizzazione di SVM

Se non si può separare positivi dai negativi (o un training set non linearmente separabile) in uno spazio di bassa dimensione utilizzando un iperpiano, bisogna mappare tutto in uno spazio dimensionale superiore dove è possibile separarli, tramite i **metodi kernel**. Tramite una funzione ϕ si passa lo spazio di input in uno nuovo, detto **feature space**, a dimensione maggiore, dove i punti sono linearmente separabili.

Esempio 26. *Dato* $[x^{(1)}, x^{(2)}]$ *si ha:*

$$\phi([x^{(1)}, x^{(2)}]) = [x^{(1)2}, x^{(2)2}, x^{(1)}x^{(2)}]$$

facendo, in questo esempio, il quadrato delle due componenti e il loro prodotto, da due dimensioni passo a 3.

Definizione 20. *Dato uno spazio delle istanze* X *e un feature space* F *si ha che una funzione* k*:*

$$k_X \to X \times X \to \mathbb{R}$$

è detta kernel valido sse esiste una feature map ϕ*:*

$$\phi : X \to F$$

tale che:

$$k(x, z) = \langle \phi(x), \phi(z) \rangle, \ \ \forall\, x, z \in X$$

avendo quindi che posso rappresentare k tramite un prodotto interno di feature risultate dal mapping definito da ϕ*.*
La funzione k ha quindi come argomenti due vettori senza restrizioni, potendo mappare un ampio spettro di entità (documenti, proteine etc...), può quindi essere penata come una ***funzione si similarità*** *(pensata come prodotto euclideo), dicendo quanto sono simili gli oggetti della funzione kernel.*
Avendo il prodotto interno scalare si ha anche la norma, dando prova della lunghezza del vettore, e avendo anche la ***distanza*** *tra due vettori nello spazio, che all'inverso è una misura di similarità.*

Definizione 21. *Definiamo* ***kernel matrix****, detta anche* ***gram matrix****, dato un kernel* k *e un insieme* $S = x_1, x_2, \ldots x_n$*, come una matrice dove ogni componente è:*

$$G_{i,j} = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}} = k(x_i, x_j)$$

Molti algoritmi interagiscono coi dati tramite prodotti scalari per questo sono importanti le funzioni kernel. Per definizione di kernel posso quindi sostituire un prodotto interno tra due vettori con la funzione kernel definita sui quei due vettori, spostandomi in una dimensione maggiore per il feature space e ottenendo un valore di similarità tra i due oggetti.

Esempio 27. *Vediamo un esempio di funzione kernel.*
Siano dati:

- $\vec{x} = (x_1, x_2)$
- $\vec{z} = (z_1, z_2)$

- $dim = 2$

Si ha quindi:

$$(\vec{x} \cdot \vec{z})^2 = (x_1 z_1 + x_2 z_2)^2 = (x_1^2 x_2^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2)$$

$$= \langle \left(x_1^2, x_2^2, \sqrt{2} x_1 x_2 \right), \left(z_1^2, z_2^2, \sqrt{2} z_1 z_2 \right) \rangle = \langle \phi(\vec{x}), \phi(\vec{z}) \rangle$$

Avendo ottenuto le feature. Quindi la funzione iniziale è rappresentabile come il prodotto interno tra le due feature, in modo implicito.
Avendo quindi:

$$\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2$$

$$\phi(x z_1, z_2) = (z_1^2, z_2^2, \sqrt{2} z_1 z_2$$

passando a $dim = 3$.

Tra i tipo di kernel si segnala quello gaussiano:

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2} \right) = \exp\left(-\frac{\mathbf{x^T x} - \ mathbf{2 x^T z} + \mathbf{z^T z}}{2\sigma^2} \right)$$

La strategia di kernel di un algoritmo è che ogni volta che si trova un prodotto interno di feature si ha che questo è uguale al kernel nello spazio originale. Non serve conoscere il mapping se il kernel è valido e posso quindi procedere direttamente con la sostituzione.

Definizione 22. *Un metodo è detto* ***kernelizzato*** *se il prodotto interno delle sue feature può essere valutato da una qualunque funzione kernel nello spazio di rappresentazione originale.*

Le distanze e le differenze sono state questioni importanti nel machine learning e nel riconoscimento di modelli per molti anni, portando a molti algoritmi noti diversi e domande importanti.
La distanza tra oggetti nello spazio delle feature è definita da:

$$d(x, x_2) = \|\phi(x) - \phi(x_2)\|_{\mathcal{H}}^2$$

denotando con $\mathcal{H}$ lo spazio del prodotto interno dove le feature sono rappresentate dal mapping:

$$\phi : X \to \mathcal{H}$$

Si ha quindi per la distanza, estendendo la norma:

$$\|\Phi(x_1) - \Phi(x_2)\|_{\mathcal{H}}^2 = < \Phi(x_1) - \Phi(x_2), \Phi(x_1) - \Phi(x_2) >$$

$$=< \Phi(x_1), \Phi(x_1) > -2 < \Phi(x_1), \Phi(x_2) > + < \Phi(x_2), \Phi(x_2) >$$

e con la notazione kernel, avendo sostituito con un kernel valido k, ottenendo una riscrittura della distanza tra due feature, kernelizzando la distanza tra due input:

$$\|\Phi(x_1) - \Phi(x_2)\|^2_{\mathcal{H}} = k(x_1, x_1) - 2k(x_1, x_2) + k(x_2, x_2)$$

Si hanno liste di funzioni kernel verificate.

Capitolo 4

Apprendimento Bayesano

Nell'ambito Bayesano si cambia l'approccio avendo la valutazione di ipotesi in base alla loro probabilità. Si studia la probabilità rispetto ai dati e rispetto alle conoscenze pregresse. Non troviamo un'ipotesi che combacia ma che è probabile.
Bisogna studiare come scegliere le ipotesi, usando risultati noti del calcolo probabilistico e quindi come funziona l'apprendimento. Useremo le nozioni di probabilità e probabilità condizionata, oltre ovviamente alla **regola di Bayes**. Il "meglio" è definito quindi tramite probabilità.
Si assume quindi che le quantità di interesse siano "governate" da distribuzioni di probabilità e che la decisione migliore può essere presa ragionando su tali distribuzioni e sull'insieme di dati di training. L'apprendimento Bayesano è importante per due ragioni principali:

1. si ha una manipolazione esplicita delle probabilità rispetto ad altri approcci pratici di alcuni tipi di problemi di apprendimento (infatti si hanno spesso paragoni con gli alberi decisionali e con le reti neurali)

2. fornisce una prospettiva utile per comprendere metodi di apprendimento che non manipolano effettivamente le probabilità

Dal punto di vista delle funzionalità si ha che ogni esempio di training osservato può aumentare o diminuire, in modo incrementale, la stima di probabilità relativa alla correttezza di un'ipotesi. Inoltre, come già anticipato, la conoscenza pregressa può essere combinata con i dati osservati per determinare la probabilità finale delle varie ipotesi. Si ha inoltre che le varie ipotesi possono effettuare predizioni probabilistiche e le istanze possono essere quindi classificate combinando le predizioni delle varie ipotesi, che sono pesate tramite il peso delle loro probabilità. Con il metodo Bayesano si ottiene quindi

uno "standard" per prendere decisioni ottimali rispetto al quale è possibile misurare ulteriori misure pratiche.
Si hanno però alcune difficoltà legate all'apprendimento Bayesano:

- si necessita avere la conoscenza di varie probabilità
- si hanno costi computazionali non indifferenti

Inquadrando nuovamente il fulcro del machine learning ricordiamo che si sta cercando la "miglior" ipotesi h, contenuta dello spazio delle ipotesi H, a partire dai dati contenuti in un training set D. Nell'apprendimento Bayesano si ha che la "miglior" ipotesi altro non è che la più probabile. Ovviamente potrei avere più ipotesi.
Il punto centrale di questo tipo è dato dal **teorema di Bayes** che fornisce un metodo diretto per calcolare la probabilità di tale ipotesi in base:

- alla sua probabilità conosciuta a priori
- alle probabilità di osservare vari dati data l'ipotesi
- ai dati stessi

Teorema 14 (Teorema di Bayes)**.** *Il teorema enuncia che:*

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

Avendo:

- $P(h)$ *che è la probabilità conosciuta a priori di* h*. Tale probabilità riflette qualsiasi conoscenza di base sulla possibilità che* h *sia corretta*
- $P(D)$ *che è la probabilità conosciuta a priori di* D*, ovvero la probabilità che* D *sia osservato*
- $P(D|h)$ *che è la probabilità di osservare* D *in presenza dell'ipotesi* h
- $P(D|h)$ *che è la probabilità a posteriori di* h*. Tale probabilità riflette la "confidenza" di avere* h *dopo che* D *è stato osservato*

In molti scenari di apprendimento, il learner considera un insieme di ipotesi candidate H ed è interessato a trovare l'ipotesi $h \in H$ più probabile in base ai dati osservati in D.

Definizione 23. *Definiamo le ipotesi maximum a posteriori (MAP) ogni ipotesi massimamente probabile:*

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

$$\underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)}$$

Ma $P(D)$ può essere cancellato in quanto costante e indipendente da h, ottenendo:

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

Spesso si assume anche che ogni ipotesi è, a priori, equiprobabile e quindi possiamo semplificare i conti.

Definizione 24. *Dato che $P(D|h)$ viene spesso chiamata **likehood (probabilità)** di D data h viene definita **ipotesi maximum likehood (ML)** ogni ipotesi che massimizza $P(D|h)$:*

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} P(D|h)$$

potendo quindi trascurare $P(h)$ in quanto equivalente $\forall\, h \in H$

Si può usare il teorema di Bayes per specificare un algoritmo di apprendimento molto semplice detto **algoritmo Brute-Force MAP LEARNING** che si articola in 2 step:

1. $\forall\, h \in H$ calcolo la probabilità a posteriori tramite il teorema di Bayes:
$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$
2. restituisco l'ipotesi h_{MAP} con la più alta probabilità a posteriori:
$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

Però, per specificare il problema d'apprendimento per l'algoritmo, bisogna obbligatoriamente specificare i valori di:

- $P(h)$
- $P(D|h)$

e quindi dobbiamo obbligatoriamente fare una serie di assunzioni:

- il training set deve essere privo di rumore, avendo:

$$d_i = c(x_i)$$

- il target concept deve essere contenuto nello spazio delle ipotesi:

$$\exists h \in H \text{ t.c. } h(x) = c(x), \ \forall x \in X$$

- devo assumere che le ipotesi siano equiprobabili e quindi:

$$P(h) = \frac{1}{|H|} \ \forall h \in H$$

  e avendo quindi:

$$P(D|h) = \begin{cases} 1 & \text{se } d_i = h(x_i), \ \forall d_i \in D \\ 0 & \text{altrimenti} \end{cases}$$

Si ha quindi che il problema di apprendimento per l'algoritmo è completamente definito. Possiamo quindi specificare che, nel primo step, ovvero quando si cerca $P(h|D)$, posso avere due casi:

1. h è inconsistente con D, avendo quindi:

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$$

2. h è consistente con D, avendo quindi:

$$P(h|D) = \frac{1 \cdot \frac{1}{|H|}}{P(D)} = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

Si ha quindi che questa analisi implica che, sotto le condizioni sopra definite, ogni ipotesi coerente è un'ipotesi MAP, perché per ogni ipotesi coerente si ha che:

$$P(h|D) = \frac{1}{|VS_{H,D}|}$$

Si ha che ogni learner consistente ha in output ipotesi MAP se si assume a priori la distribuzione uniforme delle probabilità su H dati di training deterministici e privi di rumore.
Riprendendo per esempio anche l'algoritmo find-S si ha che:

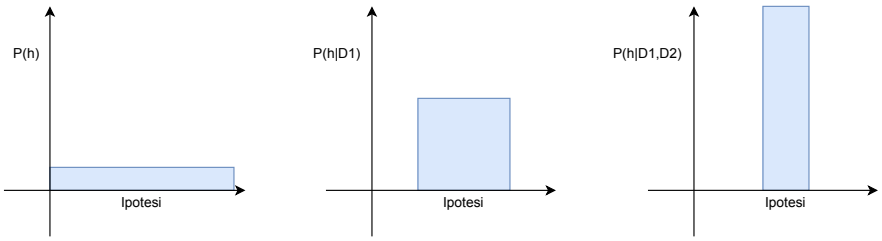


Figura 4.1: Nell'immagine le evoluzioni di probabilità delle ipotesi. Nel primo grafico tutte le ipotesi hanno la stessa probabilità, è il punto di partenza. Con gli altri due grafici si ha che man mano che i dati di addestramento si accumulano, la probabilità a posteriori delle ipotesi inconsistenti diventa zero mentre la probabilità totale che si somma a 1 è condivisa equamente tra le ipotesi consistenti rimanenti.

- ha in output ipotesi consistenti e quindi ipotesi MAP sotto la distribuzione di probabilità $P(h)$ e $P(D|h)$
- $\forall\, P(h)$ che favorisce le ipotesi più specifiche find-S trova appunto le ipotesi MAP

A riprova che il metodo Bayesano è un modo per caratterizzare il comportamento degli algoritmi di apprendimento.
inoltre, identificando $P(h)$ e $P(D|h)$ base alle quali l'output è l'ipotesi ottimale, è possibile caratterizzare le ipotesi implicite dell'algoritmo ovvero il **bias induttivo** dell'algoritmo (avendo anche che l'inferenza induttiva è modellata da un sistema di ragionamento probabilistico equivalente basato sul teorema di Bayes).
Si introduce ora il problema di apprendere funzioni target a valori continui (reti neurali regressione lineare etc...). Si ha che n base a determinate ipotesi, qualsiasi algoritmo di apprendimento che minimizzi l'errore quadratico tra l'ipotesi di output e i dati di addestramento, produrrà un'ipotesi ML. Prepariamo quindi il nostro insieme di assunzioni per il problema:

- $h : X \to \mathbb{R},\ \ \forall\, h \in H$
- li esempi sono della forma $\langle x_i, d_i \rangle$
- la funzione target è definita come $f : X \to \mathbb{R}$
- si hanno m esempi di training dove il valore target di ogni esempio è "sporcato" dal rumore casuale e_i secondo una distribuzione di probabilità normale con media nulla, avendo $d_i = f(x_i) + e_i$

Avendo quindi che $h_{ML} = \text{argmax}_{h \in H} P(D|h)$ e che gli eventi di training vengono assunti come indipendenti si ha che:

$$h_{ML} = \underset{h \in H}{\text{argmax}} \prod_{i=1}^{m} P(d_i|h)$$

Dato quindi l'errore e_i distribuito normalmente con media zero e varianza sconosciuta σ^2 si ha che anche ogni d_i segue la stessa distribuzione attorno al target $f(x_i)$. Poiché stiamo scrivendo l'espressione per $P(D|h)$, assumiamo che h sia la descrizione corretta per f, quindi:

$$\mu = f(x_i) = h(x_i)$$

e avendo quindi, per la distribuzione normale:

$$h_{ML} = \underset{h \in H}{\text{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2}$$

È comune massimizzare il logaritmo meno complicato, a causa della monotonia di questa funzione, avendo:

$$h_{ML} = \underset{h \in H}{\text{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

ma il primo termine è costante e indipendente da h e quindi può essere cancellato:

$$h_{ML} = \underset{h \in H}{\text{argmax}} \prod_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Sapendo che massimizzare questo termine negativo equivale a ridurre al minimo il termine positivo corrispondente:

$$h_{ML} = \underset{h \in H}{\text{argmin}} \prod_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

e avendo che anche tutte le costanti sono indipendenti da h e quindi possono essere rimosse:

$$h_{ML} = \underset{h \in H}{\text{argmin}} \prod_{i=1}^{m} (d_i - h(x_i))^2$$

trovando che h_{ML} è ciò che minimizza gli errori quadratici.
Si specifica la scelta della normale in quanto:

- buona approssimazione di molti tipi di rumore nei sistemi fisici
- il Teorema del Limite Centrale mostra che la somma di un numero sufficientemente grande di variabili casuali indipendenti e identicamente distribuite obbedisce a una distribuzione Normale

Si considera solo il rumore sul valore del target e non sugli attributi che descrivono le istanze stesse.
Anche in questo caso si usa il Rasoio di Occam, scegliendo di usare il principio **Minimum Description Length (*MDL*)**, scegliendo la spiegazione più breve per i dati osservati.
Tramite MDL posso "giustificare" la scelta di h_{MAP} in base alla teoria dell'informazione, infatti:

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

$$= \underset{h \in H}{\operatorname{argmax}} \log_2 P(D|h) + \log_2 P(h)$$

$$= \underset{h \in H}{\operatorname{argmin}} - \log_2 P(D|h) - \log_2 P(h)$$

e queste equazioni possono essere interpretate come un'affermazione che sono preferite ipotesi brevi, assumendo un particolare schema di rappresentazione per la codifica di ipotesi e dati.
Il principio MDL fornisce un modo per scambiare la complessità delle ipotesi per il numero di errori commessi dall'ipotesi.

Su slide altre informazioni su teoria dell'informazione.

4.1 Classificatore Bayesano ottimo

Ci si chiede quindi qual è la classificazione più probabile della nuova istanza secondo i dati di training.
Vediamo con un esempio che non basta applicare h_{MAP}:

Esempio 28. *Sia* $H = \{h_1, h_2, h_3\}$ *con* $P(h_1) = 0.4$ *e* $P(h_2) = P(h_3) = 0.3$. *Si ha quindi che:*

$$h_{MAP} = h_1$$

Consideriamo però una nuova istanza x *classificata positiva per* h_1 *e negativa per le altre due. Si ha quindi che:*

- *la probabilità che* x *sia positivo è 0.4*

- *la probabilità che x sia negativo è 0.6*

e quindi la classificazione più probabile non è quella di h_{MAP}.

Abbiamo che la classificazione più probabile si ottiene combinando le previsioni di tutte le ipotesi, ponderate in base alle loro probabilità posteriori. Data $P(v_j|D)$ come la probabilità che la classificazione corretta sia v_j:

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

ottenendo che il classificatore Bayesano ottimo è:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

dato V come insieme dei target (?).

Esempio 29. *Vediamo un esempio chiarificatore.*
Sia $V = \{+, -\}$ *e siano:*

- $P(h_1, D) = 0.4, \;\; P(-, h_1) = 0, \;\; P(+, h_1) = 1$
- $P(h_2, D) = 0.3 \;\; P(-, h_2) = 1, \;\; P(+, h2) = 0$
- $P(h_3, D) = 0.3, \;\; P(-, h_3) = 1, \;\; P(+, h3) = 0$

Si hanno:

$$\sum_{h_i \in H} P(+|h_i)P)(h_i|D) = 0.4$$

$$\sum_{h_i \in H} P(-|h_i)P)(h_i|D) = 0.6$$

$$\underset{v_j \in \{+,-\}}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

4.2 Classificatore Bayesano naive

Il Classificatore Bayesano naive si applica alle attività di learning in cui ogni istanza x è descritta da una giunzione di valori di attributi e in cui la funzione target $f(x)$ può prendere un valore qualsiasi dall'insieme finito V. Descriviamo gli esempi di training come $\langle a_1, a_2, \ldots a_n \rangle$.
Applicando il metodo Bayesano si ha:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j|a_1, a_2, \ldots a_n)$$

$$= \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2, \ldots a_n | v_j) P(v_j)}{P(a_1, a_2, \ldots a_n)}$$

e, rimuovendo le i fattori indipendenti:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2, \ldots a_n | v_j) P(v_j)$$

Avendo che:

- $P(v_j)$ può essere stimato tramite la frequenza di v_j in D
- $P(a_1, a_2, \ldots a_n | v_j)$ non può essere stimato in questo modo am il numero di questi termini è pari a $|X| \cdot |V|$

Con il classificatore Bayesano naive si hanno diverse semplificazioni. In primis i valori degli attributi sono condizionatamente indipendenti comprando:

- $P(a_1, a_2, \ldots a_n | v_J) = \prod_i P(a_i | v_j)$
- il numero dei termini $a_1, a_2, \ldots a_n$ è pari a:

  $$|DA| \cdot |DT| + |DT|$$

  ove:

  - DA sta per attributi distinti/unici
  - DT sta per valori di target distinti/unici
- non si ha la ricerca esplicita dentro H ma solo il contro delle frequenze

Si ha quindi il classificatore Bayesano naive:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

Esempio 30. *Vediamo un esempio chiarificatore.*
Sia dato il seguente dataset, con il target PlayTennis:

Day	*Sunny*	*Temp.*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Si ha la nuova istanza:

$$\langle Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong \rangle$$

Si ha quindi che:

$$\prod_i P(a_i|v_j) = (Outlook = sunny|v_j) \cdot P(Temperature = cool|v_j)$$

$$\cdot P(Humidity = high|v_j) \cdot P(Wind = strong|v_j)$$

Avendo la stima delle probabilità:

$$P(PlayTennis = yes) = \frac{9}{14} = 0.64$$

$$P(PlayTennis = no) = \frac{5}{14} = 0.36$$

In modo analogo calcolo le probabilità condizionali. Per esempio per $Wing = Strong$*:*

$$P(Wing = Strong|PlayTennis = yes) = \frac{3}{9} = 0.33$$

$$P(Wing = Strong|PlayTennis = no) = \frac{3}{5} = 0.60$$

Posso quindi calcolare v_{NB}*:*

$$P(yes) \cdot P(Sunny|yes) \cdot P(cool|yes) \cdot P(high|yes) \cdot P(cool|yes) = 0.0053$$

$$P(yes) \cdot P(Sunny|no) \cdot P(cool|no) \cdot P(high|no) \cdot P(cool|no) = 0.0206$$

Quindi si ha che:

$$v_{NB} = no$$

e normalizzando:

$$\frac{0.0206}{0.0206 + 0.0053}$$

Si è visto come, normalmente, le probabilità sono stimate dalla frazione di volte in cui si osserva che l'evento si verifica sul numero totale di opportunità N:

$$\frac{n_c}{N}$$

è spesso questo metodo fornisce una buona stima.
Si ha però un limite se n_c è molto piccolo, avendo risultati errati con:

- sottovalutazione delle probabilità a causa di un bias
- se addirittura n_c è nullo esso "dominerà" sul classificatore Bayesano

Si introduce un nuovo approccio Bayesano sfruttando il cosiddetto **m-estimate**, ovvero:

$$\frac{n_c + m \cdot p}{n + m}$$

dove p è una stima precedente della probabilità che desideriamo determinare, ed m è una costante chiamata *equivalent sample size* che determina quanto sia importante il peso di p rispetto ai dati osservati.
In assenza di informazioni aggiuntive p ha distribuzione uniforme quindi si ha, per k numero di possibili valori di attributo:

$$p = \frac{1}{k}$$

Si nota che per m nullo si ha che l'm-estimate è uguale a:

$$\frac{n_c}{n}$$

e quindi m può essere interpretato come il numero di campioni virtuali distribuiti su p a cui vengono aggiunti gli n esempi effettivi osservati.

Esercitazione su Bayes concept learning

Si ricorda che si cerca una strategia per trovare la miglior ipotesi nello spazio delle ipotesi. In questo caso si parla di "migliore" in termini di più "probabile".
Quando si fa inferenza bayesana e tutto governato da *incertezza* ma permette di sfruttare conoscenze a priori.
Ricordiamo quindi la formula di Bayes:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

con:

- D è il dataset
- θ che per noi è l'ipotesi
- $P(\theta|D)$ che è la distribuzione a posteriori
- $P(\theta)$ è la distribuzione a priori
- $P(D|\theta)$ è la verosimiglianza
- $P(D)$ è l'evidenza

Sapendo che:

$$P(D) = \sum_i (D, \theta_i) = \sum_i P(D|\theta_i)P(\theta_i)$$

$P(D)$ è invariante rispetto al valore dell'ipotesi e quindi possiamo benissimo rimuoverlo nel calcolo in quanto non influisce sulla probabilità a posteriori:

$$P(\theta_i|D) \propto P(D|\theta_i)P(\theta_i)$$

(si ricorda che $\propto$ indica **proposizionale a** quindi che due cose sono uguali al più di una costante)
Ricordiamo inoltre che la massima ipotesi a posteriori è:

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D) = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

Se si sa che anche la conoscenza a priori è ininfluente, essendo distribuita seconda una distribuzione uniforme, e si parla di ipotesi di massima verosimiglianza:

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} P(D|h)$$

L'approccio di Bayes nel machine learning è abbastanza oneroso, dovendo calcolare tutte le probabilità a posteriori di ogni ipotesi (se H è grande diventa molto costoso).

Esercizio 7. *Si assuma di avere:*

temp	h_1	h_2	h_3	h_4
hot	no	no	yes	yes
cold	no	yes	no	yes
$P(D\|h)$	0	0	1	0

Si calcoli h_{MAP}, assumendo una conoscenza a priori con distribuzione uniforme su tutto lo spazio H.
Si hanno quindi 4 ipotesi, ciascuna che esprime il valore dei due attributi hot e cold. L'ultima riga ci dice anche la verosimiglianza di ogni ipotesi.
Faccio quindi i conti, sapendo che ogni $P(h_i) = \frac{1}{4}$, avendo distribuzione uniforme per tutte le ipotesi.
Calcolo innanzitutto:

$$P(D) = \sum_i (D, h_i) = \sum_i P(D|h_i)P(h_i) = 0 \cdot \frac{1}{4} + 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{4} + 0 \cdot \frac{1}{4} = \frac{1}{4}$$

e quindi:

$$P(h_0|D) = \frac{P(D|h_0)P(h_0)}{P(D)} = \frac{0 \cdot \frac{1}{4}}{\frac{1}{4}} = 0$$

$$P(h_1|D) = \frac{P(D|h_1)P(h_1)}{P(D)} = \frac{0 \cdot \frac{1}{4}}{\frac{1}{4}} = 0$$

$$P(h_2|D) = \frac{P(D|h_2)P(h_2)}{P(D)} = \frac{1 \cdot \frac{1}{4}}{\frac{1}{4}} = 1$$

$$P(h_3|D) = \frac{P(D|h_3)P(h_3)}{P(D)} = \frac{0 \cdot \frac{1}{4}}{\frac{1}{4}} = 0$$

Si ottiene infine:

$$h_{MAP} = 1$$

Esercizio 8. *Si consideri un problema di diagnostica medica con due ipotesi alternative:*

1. *il paziente ha il cancro (indichiamo la cosa con cancer)*
2. *il paziente non ha il cancro (indichiamo la cosa con ¬cancer)*

Si hanno inoltre due possibili outcome per i dati dei laboratori:

- *positivi, indicati con "+" (che indica se si pensa che si abbia il cancro)*

- *negativi, indicati con "-" (che indica se si pensa che non si abbia il cancro)*

Abbiamo quindi le seguenti probabilità:

- $P(cancer) = 0.008$
- $P(\neg cancer) = 0.992$
- $P(test = +|cancer) = 0.98$
- $P(test = -|cancer) = 0.02$
- $P(test = +|\neg cancer) = 0.03$
- $P(test = -|\neg cancer) = 0.97$

Si supponga ora che un laboratorio studi un nuovo paziente e che ottenga un risultato positivo "+" e vediamo se possiamo dire se ha il cancro:

$$P(cancer|test = +) \propto P(test = +|cancer)P(cancer) = 0.0078$$

$$P(\neg cancer|test = +) \propto P(test = +|\neg cancer)P(\neg cancer) = 0.0298$$

Quindi:

$$h_{MAP} = \neg cancer$$

Posso inoltre determinare la corretta probabilità normalizzando a 1:

$$P(cancer|test = +) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

$$P(\neg cancer|test = +) = \frac{0.0298}{0.0078 + 0.0298} = 0.79$$

Si nota quindi come il risultato dipenda molto dalla conoscenza a priori (che deve essere disponibile).

Esercizio 9. *Si consideri la seguente tabella con un attributo e il target, entrambi booleani:*

Temperature	Play Tennis
H	yes
H	yes
H	no
C	yes
H	yes
C	no
C	no
C	no
C	yes

Si hanno quindi vari calcoli:

- $P(H|yes) = 0.6$
- $P(H|no) = 0.25$
- $P(C|yes) = 0.4$
- $P(C|no) = 0.75$
- $P(yes) = 0.56$
- $P(no) = 0.44$

Si ha anche:

$$P(H) = P(H|yes)P(yes) + P(H|no)P(no) = 0.6 \cdot 0.56 + 0.25 \cdot 0.44$$
$$= 0.336 + 0.11 = 0.447$$
$$P(C) = P(C|yes)P(yes) + P(C|no)P(no) = 0.4 \cdot 0.56 + 0.75 \cdot 0.44$$
$$= 0.224 + 0.33 = 0.554$$

Ricordiamo che per Naive Bayes si ha, avendo una sequenza $d_1, d_2 \dots d_n$ di osservazioni:

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|d_1, d_2 \dots d_n)$$
$$= \underset{h \in H}{\operatorname{argmax}} \frac{P(d_1, d_2 \dots d_n|h)P(h)}{P(D)}$$
$$\propto \underset{h \in H}{\operatorname{argmax}} P(d_1, d_2 \dots d_n|h)P(h)$$

Per semplificare, tramite naive Bayes,si può assumere:

$$P(d_1, d_2 \dots d_n|h) = \prod_i P(d_i|h)$$

ovvero indipendenza condizionata (dicendo che un valore $d : i$ è indipendente da un qualunque d_j) e quindi si ha, sostituendo:

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h) \prod_i P(d_i|h)$$

Esercizio 10. *Dati 6 esempi con tre attributi e target T (attributi e target sono booleani):*

Esempi	A	B	C	T
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	?

Si completi la tabella con la label T per x_6.
Dividiamo in:

- $h_0 = [T = 0]$
- $h_1 = [T = 1]$

Si ha quindi per la prima ipotesi (h_0), sempre pensando a x_6:

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

elimino l'evidenza:

$$\propto P(A = 1, B = 1, C = 0|T = 0)P(T = 0)$$

uso l'assunzione di Naive Bayes:

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

Avendo quindi, svolgendo il conto:

$$P(h_0|x_6) \propto \frac{1}{2} \cdot 1 \cdot 0 \cdot \frac{2}{5} = 0$$

Si passa poi alla seconda ipotesi (h_1), sempre pensando a x_6:

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

elimino l'evidenza:

$$\propto P(A = 1, B = 1, C = 0|T = 1)P(T = 1)$$

uso l'assunzione di Naive Bayes:

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

Avendo quindi, svolgendo il conto:

$$P(h_1|x_6) \propto \frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}\cdot\frac{3}{5} = \frac{2}{45}$$

So quindi che $h_{MAP} = h_1$, avendo che la probabilità con h_1 è maggiore di quella con h_0. Si ha quindi, avendo che per x_6 scelgo h_1 che corrisponde ad avere $T = 1$:

Esempi	A	B	C	T
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	1

Si può anche calcolare, per completezza:

$$P(x_6) = \sum_i P(x_6, h_i) = \sum_{i\in\{0,1\}} P(A=1,B=1,C=0|T=i)P(T=i)$$

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

Posso quindi calcolare:

$$P(h_1|x_6) = P(T=1|A=1,B=1,C=0) = \frac{P(A=1,B=1,C=0|T=1)P(T=1)}{p(x_6)}$$
$$= \frac{P(A=1|T=1)P(B=1|T=1)P(C=0|T=1)P(T=1)}{p(x_6)}$$
$$= \frac{\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}\cdot\frac{3}{5}}{\frac{1}{3}\cdot\frac{1}{3}\cdot\frac{2}{3}\cdot\frac{3}{5}} = 1$$

Ugualmente dovrei calcolare $P(h_0|x_6)$ ma sapendo che i valori sono booleani e quindi ho solo h_1 e h_0, avendo $P(h_1|x_6) = 1$ so che $P(h_0|x_6) = 0$.
Si hanno quindi:

- $h_{MAP} = h_1 := [T = 1]$
- $P(x_6) = \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{3}{5} = \frac{2}{45} = 0.04$
- $P(x_6|T = 0)P(T = 0) = \frac{1}{2} \cdot 1 \cdot 0 \cdot \frac{2}{5} = 0$
- $P(x_6|T = 0) = \frac{1}{2} \cdot 1 \cdot 0 = 0$
- $P(A = 1|T = 0) = \frac{2}{5}$
- $P(T = 0) = \frac{2}{5}$
- $P(h_0|x_6) = P(T = 0|x_6) = 0$
- $P(x_6|T = 1)P(T = 1) = \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{3}{5} = \frac{2}{45} = 0.04$
- $P(x_6|T = 1) = \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{2}{27} = 0.07$
- $P(B = 1|T = 1) = \frac{1}{3}$
- $P(T = 1) = \frac{3}{5}$
- $P(h_1|x_6) = P(T = 1|x_6) = 1$

Capitolo 5

Clustering

Il **clustering** è un approccio non supervisionato. Come se stessimo lavorando "ad occhio" si potrebbero avere vari criteri per effettuare il raggruppamento delle istanze, in mancanza di label. Il criterio di scelta è praticamente arbitrario e il termine "classificare" diventa più un "raggruppare". Il sistema è quindi flessibile nel decidere i criteri (che non può essere scelto in base alle label) sfruttando concetti come la distribuzione dei dati. I risultati ottenuti da un apprendimento non supervisionato potrebbero "sorprendere" in quanto potrebbero essere imprevedibili a priori. In ogni caso l'incertezza del clustering non è la sola, in quanto anche un sistema supervisionato potrebbe comunque lasciare spazio ad incertezza sulle ipotesi (non riuscendo magari ad avere tutte le label possibili per l'insieme dei casi).
Possiamo pensare di rappresentare nello spazio (con dimensione pari al numero di attributi delle istanze) le varie istanze per poi cercare di capire come etichettare le varie istanze la cui etichetta non è nota. Si procede quindi **raggruppando per distanza** (se è vicina a istanze con una certa etichetta nota uguale anche istanza di cui cerchiamo l'etichetta avrà quell'etichetta), usando la *distanza di Hamming* o la *distanza Euclidea*, ma prima di parlarne introduciamo meglio tutto il discorso.
La "decisione forte" è raggruppare e quindi scegliere in base alla distanza, definendo prima la distanza stessa, misurando poi il risultato secondo un certo criterio di misura (sempre basato sulla distanza).
Avendo una classificazione non supervisionata si ha una scarsa conoscenza a priori dei dati da analizzare, puntando quindi all'estrazione automatica delle classi mentre in quella supervisionata si partiva da classi etichettate, avendo magari speso costo computazionale per ottenerle, e e da una struttura classificatoria conosciuta (avendo un set di ipotesi possibili). I sistemi non supervisionati tornano comodo 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. Con l'apprendimento non supervisionato quindi la non conoscenza a priori è anche un vantaggio (magari le etichette che uso nel supervisionato potrebbero essere errate), insieme alla riduzione dell'errore umano. Un altro pro è che tutte le classi con caratteristiche uniche vengono identificati e quindi i metodi di apprendimento non supervisionato sono efficaci con elementi di tipo numerico, che diventano coordinate in uno spazio, e con elementi dotati di ordinamento intrinseco, per il calcolo delle distanze. Di contro le classi ottenute non per forza hanno un significato. Inoltre l'utente ha poco controllo sulla procedura e sui risultati e tali metodi sono meno efficaci con elementi ordinati in modo arbitrario o parziale.
Si cerca quindi di separare un insieme di elementi in sottoinsiemi con elementi accomunati da caratteristiche comuni, in base a uno o più attributi con la scelta dell'attributo viene fatta dall'algoritmo di apprendimento. Tra i campi di intersse si hanno:

- data mining
- pattern recognition
- image analysis
- bioinformatica
- ricerche di mercato
- pianificazione urbana
- sismologia
- astronomia

Ogni elemento da classificare viene specificato da un vettore caratteristico e si ha una misura di similarità tra elementi. Si hanno quindi due criteri da rispettare:

- **omogeneità** quando elementi dello stesso cluster hanno alto livello di similarità, quindi dentro il cluster
- **separazione** quando elementi di cluster diversi hanno basso livello di similarità, quindi tra cluster

Definizione 25. *Dato $N = \{e_1, \ldots, e_n\}$ un insieme di n elementi e sia $C = \{C_1, \ldots, C_k\}$ una partizione di N in sottoinsiemi. Ogni C_i è detto* ***cluster*** *e C è detto* ***clustering*** *di N.*

Definizione 26. *Dati due elementi di N essi sono **mates** rispetto a C se appartenenti allo stesso cluster C_i*

Un elemento può essere rappresentato da un vettore di numeri reali, ciascuno dei quali misura una specifica caratteristica (feature).

Definizione 27. *Definiamo la **misura di similarità** tramite la **distanza tra vettori**. Si hanno tre distanze:*

- *la **distanza euclidea** (che è la solita distanza ottenuta dal teorema di Pitagora, è semplicemente la lunghezza del segmento che collega due punti nello spazio):*

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)^2}$$

  è invariante rispetto alle traslazioni e rotazioni degli assi

- *la **distanza di Manhattan** (misuro gli spostamenti, che hanno peso 1, tra due punti in termini di somma ti tali spostamenti che formano una linea spezzata):*

$$d(\vec{x}, \vec{y}) = \sum_i (x_i - y_i)$$

  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

- *la **distanza di Minkowski** (dando un peso diverso rispetto alla distanza euclidea):*

$$d(\vec{x}, \vec{y}) = \sqrt[k]{\sum_i (x_i - y_i)^k}$$

  In altri termini questa è la forma generica, infatti al variare di k si hanno le varie distanze:

  - *con $k = 1$ ottengo la distanza di Manhattan*
  - *con $k = 2$ ottengo la distanza euclidea*
  - *con $k = \infty$ ottengo la distanza di Lagrange-Čebyšëv*

Useremo soprattutto il caso euclideo.

In generale si hanno varie tipologie di clustering, come da figura 5.1 in base all'approccio di studio delle istanze o di studio dei confini tra cluster. Questo discorso non lo approfondiremo ma si ha:

- **gerarchico** quando 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. Come pro si ha intuitività grazie ad una struttura singola, coerente e globale. Si ha quindi:
  - **Agglomerativi** quando bisogna scegliere la coppia di cluster da fondere
  - **divisivi**, quando si deve determinare il cluster da dividere

  Come algoritmi si hanno:
  - Neighbor joining
  - Metodo del centroid
- **non gerarchico/partitivo** quando mirano a ripartire le n unità della popolazione in K gruppi, fornendo una sola partizione anziché una successione di partizioni tipica dei metodi gerarchici.
  Come algoritmo si ha, ad esempio, **K-means** definendo K cluster tali per cui si abbiano punti vicini in cluster separati. È un algoritmo iterativo

5.1 K-Means

Definizione 28. *definiamo* ***dendrogramma*** *come un albero (grafo) utilizzato per visualizzare la somiglianza nel processo di "raggruppamento".*

Questi algoritmo è quello che vedremo per il metodo non approssimato. Si ha che la soluzione non è visualizzabile attraverso dendrogrammi e assume che il numeor K di cluster sia noto, puntando a minimizzare le distanze tra elementi e i centroidi (un punto geometrico di riferimento per i cluster) dei clusters loro assegnati.
L'algoritmo lavora con dati numerici in cinque step:

1. si fissano a caso K centroidi iniziali di altrettanti cluster

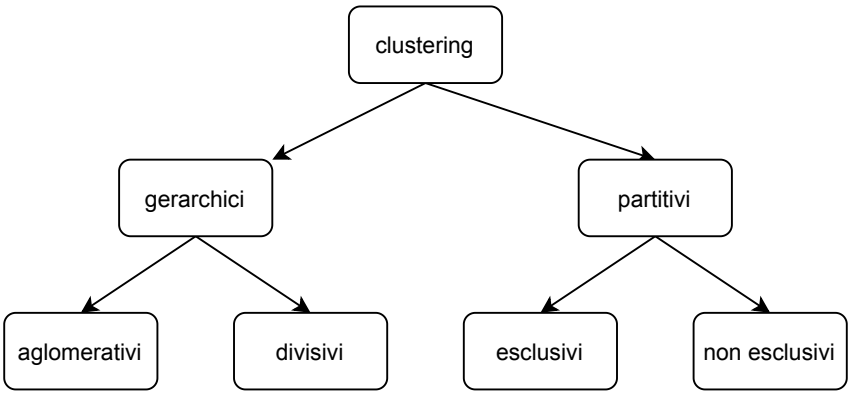


Figura 5.1: Albero delle tipologie di clustering

2. per ogni elemento si calcola la distanza da ciascun centroide e lo si assegna al più vicino. Questo è uno step di classificazione, scegliendo il cluster per un elemento dato il centroide del cluster

3. per la partizione provvisoria così ottenuta si ricalcolano i centroidi di ogni cluster, usando media aritmetica dei punti del cluster parziale, ottenendo il nuovo centroide. Si aggiustano quindi tutti i centroidi

4. per ogni individuo si ricalcola la distanza dai centroidi e si effettuano gli eventuali spostamenti tra cluster dei vari elementi in base al fatto che si hanno nuovi centroidi

5. si ripetono le operazioni 3 e 4 finché si raggiunge il numero massimo di iterazioni impostate, avendo un euristica, o non si verificano altri spostamenti, avendo una stabilità di risultato, avendo il risultato "migliore"

Si nota quindi la semplicità di implementazione e un tempo di calcolo in :

$$O(tKn)$$

con:

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

Di contro si ha una sensibilità rispetto alla scelta dei centroidi iniziali, sperimentalmente si è dimostrato che brutti valori iniziali corrompano l'intero processo. Inoltre, come detto all'inizio, non si può predire il numero di cluster non conoscendo a priori i dati e non esiste un K ottimale e non ci sono proprietà che ce lo possano suggerire.
In ogni caso mappa sensatamente i vari elementi in base alle loro caratteristiche e quindi è un approccio parecchio usato, essendo generalmente efficacie.
Non sapendo a priori il numero di K posso ottenere scarsi risultati magari non avendo abbastanza centroidi. Adattarsi ad un numero "eerato" di centroidi può portare a risultati "sporchi".
Un altro problema è dato da dati distribuiti secondo diverse "dimensioni" nello spazio ma l'algoritmo lavorando sui centroidi potrebbe misclassificare questo dettaglio, non distinguendo i corretti insiemi di punti. Analogamente succede se si hanno insiemi di punti più o meno densi, portando a misclassicazioni. Sempre in modo analogo le proprietà di distribuzione geometrica degli elementi comporta problemi (anche con un K adeguato).

Per superare queste difficoltà 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.

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. Si ha una tecnica di misura, supponendo che ogni cluster abbia almeno due elementi, avendo tecniche diverse qualora si abbia cardinalità 1, avendo la distanza tra due elementi $d(i, j)$ e due concetti:

1. **distanza media "intra-cluster"**, ovvero dentro i cluster

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

   è buono se il valore è basso

2. **distanza media "inter-cluster"**, ovvero tra cluster:

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

   è buono se il valore è alto

ottenendo quindi la **silhouette per il punto** i:

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

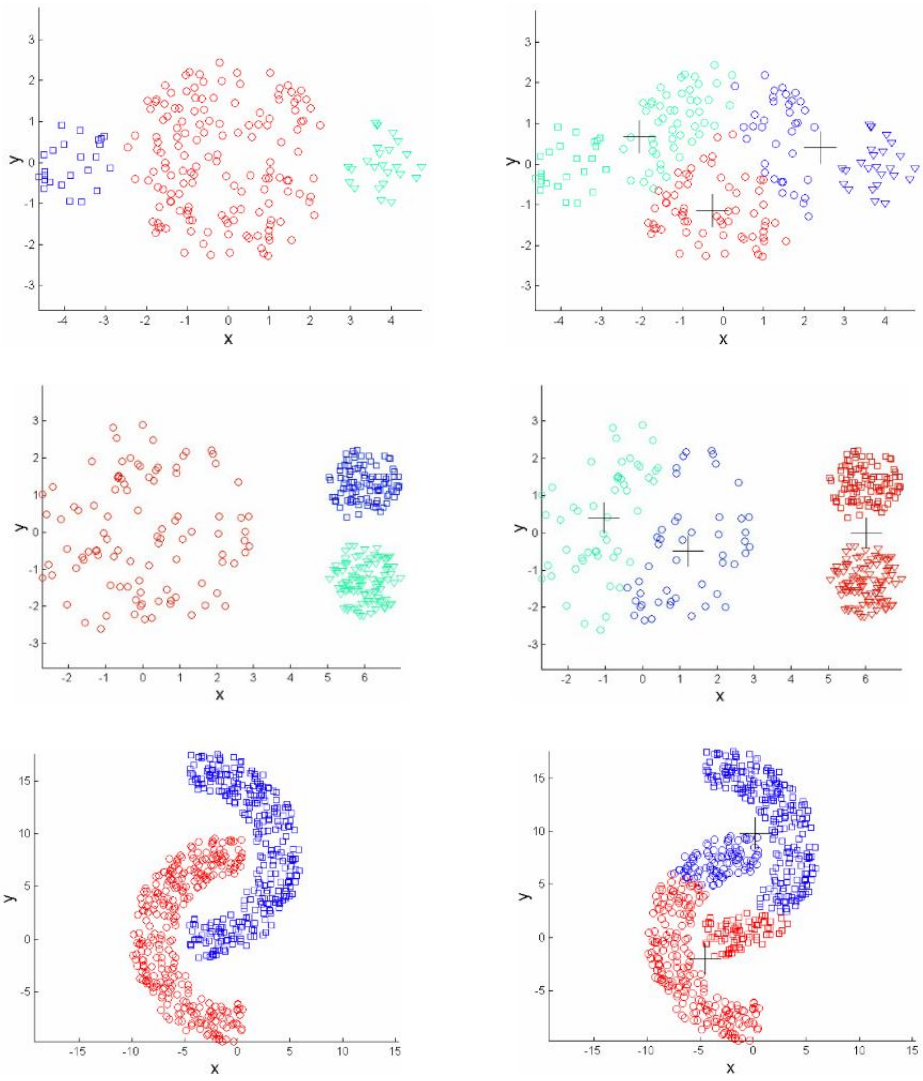


Figura 5.2: Esempi di problematiche per il clustering. A sinistra il valore atteso e a destra quello ottenuto. Prima si ha il problema di cluster con diverse dimensioni, poi con diverse densità e infine con problemi di proprietà geometriche.

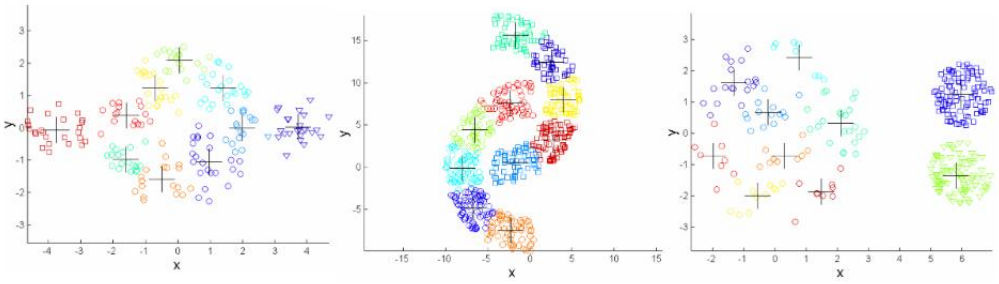


Figura 5.3: Esempi di risoluzione delle tre problematiche aggiungendo cluster.

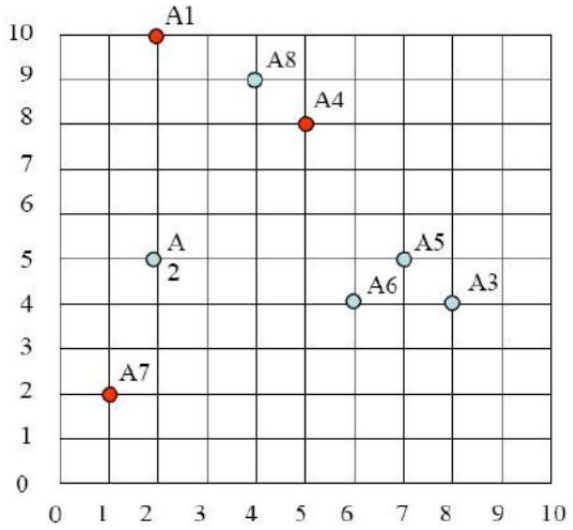
La qualità viene quindi visualizzata da un diagramma che studia la silhouette al variare di K per ogni valore di ogni attributo (mettendo per ogni attributo in alto i valori più studiabili).
La media $s(i)$ su tutti i punti di un cluster è una misura di quanto siano strettamente raggruppati tutti i punti 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.

Esercitazione clustering e K-Means

Esercizio 11. *Siano dati i seguenti 8 esempi:*

1. $A1 = (1, 10)$
2. $A2 = (2, 5)$
3. $A3 = (8, 4)$
4. $A4 = (5, 8)$
5. $A5 = (7, 5)$
6. $A6 = (6, 4)$
7. $A7 = (1, 2)$
8. $A8 = (4, 9)$

$K = 3$ e la distanza euclidea.
Si supponga che all'inizio i tre centroidi (rispettivamente seed1, seed2 e seed3) siano A1, A4 e A7 avendo:



e si esegua un'iterazione per ricalcolare i centroidi e i cluster associati.
Usando la distanza euclidea procedo coi calcoli per generare i cluster, associando il cluster scegliendo il centroide che dista meno:

- *per A1:*
  - $d(A1, seed1) = 0$ *essendo A1 il seed1 attuale*
  - $d(A1, seed2) = 3.6$
  - $d(A1, seed3) = 8.06$

  quindi $A1 \in cluster1$
- *per A2:*
  - $d(A2, seed1) = 5$
  - $d(A2, seed2) = 4.24$
  - $d(A2, seed3) = 3.16$

  quindi $A2 \in cluster3$
- *per A3:*
  - $d(A3, seed1) = 6$

  - $d(A3, seed2) = 5$
  - $d(A3, seed3) = 7.28$

  quindi $A3 \in cluster2$

- *per* $A4$*:*

  - $d(A4, seed1) = 2.6$
  - $d(A4, seed2) = 0$ *essendo* $A4$ *il* $seed2$ *attuale*
  - $d(A4, seed3) = 7.21$

  quindi $A4 \in cluster2$

- *per* $A5$*:*

  - $d(A5, seed1) = 7.07$
  - $d(A5, seed2) = 3.6$
  - $d(A5, seed3) = 6.7$

  quindi $A5 \in cluster2$

- *per* $A6$*:*

  - $d(A6, seed1) = 7.21$
  - $d(A6, seed2) = 4.12$
  - $d(A6, seed3) = 5.38$

  quindi $A6 \in cluster2$

- *per* $A7$*:*

  - $d(A7, seed1) = 8.06$
  - $d(A7, seed2) = 7.21$
  - $d(A7, seed3) = 0$ *essendo* $A7$ *il* $seed3$ *attuale*

  quindi $A7 \in cluster3$

- *per* $A8$*:*

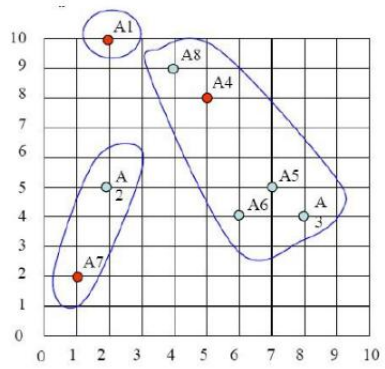
  - $d(A8, seed1) = 5$
  - $d(A8, seed2) = 2$
  - $d(A8, seed3) = 7.6$

quindi $A8 \in cluster2$

Quindi nei nuovi cluster si ha:

- $cluster1 = \{A1\}$
- $cluster2 = \{A3, A4, A5, A6, A8\}$
- $cluster3 = \{A2, A7\}$

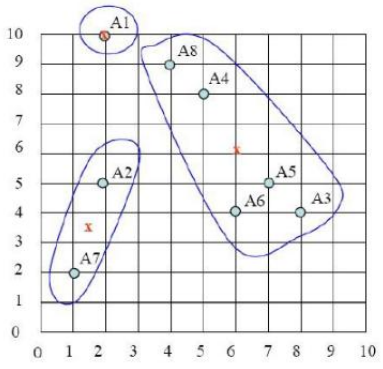
Visualmente:



Posso fare le medie dei punti di ogni cluster per calcolare i nuovi centroidi:

- $seed1New = (2, 10)$
- $seed2New = \left(\frac{8+5+7+6+4}{5}, \frac{4+8+5+4+9}{5}\right) = (6, 6)$
- $seed3New = \left(\frac{2+1}{2}, \frac{5+2}{2}\right) = (1.5, 3.5)$

Ovvero, segnando con una x rossa i nuovi centroidi:



Avendo una sola epoca da calcolare mi fermo.

Capitolo 6

Reinforcement learning

Brevissima introduzione a caso.
Questi metodi sono usati spesso in ambienti in cui serve trainare una macchina magari per giocare ad un gioco.
Si inserisce una maggiore "distanza" tra l'errore di prestazione del sistema e la modalità di affinamento del comportamento. In questo tipo di apprendimento l'errore è rappresentato come premio/punizione per il comportamento, non vedendolo più come "distanza" per come era intesa in concept learning, SVM, reti etc. . ..
Quindi, nell'esempio di un gioco, si ha un ambiente con delle regole e dei punteggi e questi vengono usati per il training, modificando il comportamento in base ai punti, modificando le policy del comportamento. L'algoritmo che deve cambiare le azioni dell'**agente** in base alle indicazioni dell'ambiente è il fulcro del reinforcement learning. La definizione di tale algoritmo è quindi ad hoc sul sistema in analisi e avere più informazioni implica una migliore evoluzione. La variante più grossa è capire cosa si ha sbagliato in base ai punti ottenuti.
Non si vede altro in merito.

Capitolo 7

Deep learning

Brevissima introduzione, nelle slide c'è un approfondimento. L'argomento comunque pare non essere nel programma d'esame (il prof l'ha detto due volte).
Ad oggi la fama dei sistemi di apprendimento è associata al **deep learning**. Dalle reti neurali si passa a reti multi-layer "complete", con algoritmi di discesa del gradiente, funzioni di attivazioni, strumenti di regolarizzazione, dati (tanti) e calcoli (tanti). Si hanno quindi sia aspetti qualitativi (come l'uso di strumenti di regolarizzazione) che quantitativi (dati e calcoli).
Bisogna quindi studiare algoritmi di learning anche in queste "nuove reti", per esempio appunto le **deep neural net**. Potrei avere anche reti con cicli tra le sinapsi, ricorrenze, salvataggio in memoria, studi temporali etc..., **convultional neural net** e **recurrent neural net** etc... (per uno schema vedere figura 7.1). Spesso tali reti sono usate nel riconoscimento di immagini, soprattutto le *convultional neural net* (anche il passaggio da immagine a feature, prima manuale, è stato integrato nelle reti). Si ha quindi un arricchimento di quanto studiato fin'ora.
Una rete tradizionale, magari con due livelli (uno di input e uno nascosto), e lo applichiamo ad un esempio base di immagine: riconoscere una matrice di numeri scritti a mano, detto **MNIST**. Si hanno 60000 immagini (di 784 pixel l'una, 28×28) di train e 10000 di test. Per lo studio si appiattisce la matrice di pixel in un array. Usando una rete come appena descritta, con tutto quanto, ha prestazioni di gran lunga superiore a quelle classiche. Si ha uno studio approfondito delle varie funzioni di attivazione. Tradizionalmente serviva una funzione derivabile mentre si introducono funzioni non derivabili in qualche punto, modificando la discesa del gradiente per funzionare anche in questo caso.
Lo schema base delle reti di questo tipo prevede l'arricchimento con molti layer, completamente connessi tra loro (avendo tutte le possibili connessioni

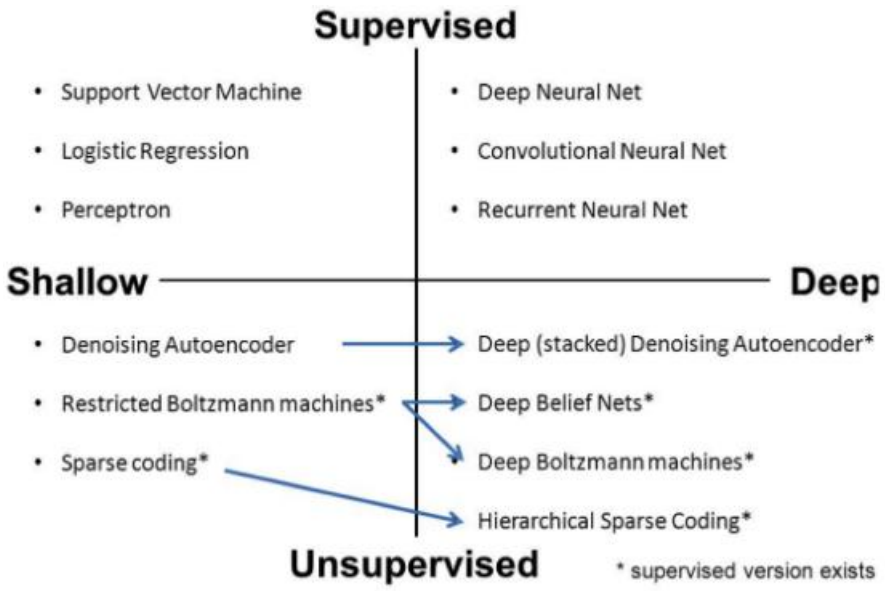


Figura 7.1: Tassonometria delle reti neurali

tra loro), aumentando la complessità computazionale. Si ha inoltre un parametro α che riduce i pesi verso lo zero, controllando la regolarizzazione.
Un'altra struttura usata è l'**autoencoder** che non è "deep" ma moderna come gli strumenti "deep" ed è usata per l'apprendimento non supervisionato. In questo caso si hanno uno o più layer nascosti con una certa funzione di attivazione ma con meno neuroni delle feature in input. Non avendo un'etichetta, non essendo supervisionato, addestro il sistema sul confronto tra gli output (che quindi è di cardinalità pari al numero di feature in input) e gli input, in modo ciclico. Come per le reti altero i pesi in base all'errore calcolato (in questo caso direttamente tra input e output).
I pesi a fine addestramento possono anche dirmi quali connessioni sono diventate "più forti". Ogni layer nascosto è quindi una codifica del dato. Questo approccio può essere usato per MNIST.
Il **deep learning** è quindi appunto l'apprendimento nei layer (molti). I layer sono associati via via a concetti sempre più complicati. Mettendo insieme le astrazioni via via più complesse si ottengono i risultati. Questo "schema" si è "scoperto" sperimentalmente, le varie feature vengono infatti definite automaticamente, per aggiornamento progressivo dei pesi.
Negli anni si è passati anche a dataset più complessi di MNIST, magari come CIFAR-100, per testare algoritmi di deep learning sempre più complessi.
In merito allo studio delle immagini si hanno anche le **convoluzioni** e le **max pooling**. Le max pooling seleziona i segnali più dominanti/"forti" di

un gruppo di neuroni in una certa regione, riducendo la cardinalità dell'insieme dei segnali (è una sorta di estrazione delle informazioni). Le convoluzioni, che nel cervello umano sono connessioni tra neuroni che evidenziano il campo visuale, invece, rappresentano il fatto che pixel vicini in ingresso alimentano singole zone di neuroni nel layer successivo (se ho vari input per un pixel nel layer successivo saranno tutti reindirizzati ad un singolo neurone, perlomeno a livello parziale). Non si ha più un connessione completa tra i neuroni ma una più specifica (non tutti con tutti ma singole "zone" con alcuni neuroni). Con regolarizzazione, il **dropout**, si indica invece l'evitare l'overfitting "sfumando" lo spazio/ipotesi appreso, migliorando l'astrazione uccidendo neuroni in una certa quantità (si ha una probabilità di eliminazione associata ad ogni neurone). L'eliminazione è fatta progressivamente nell'apprendimento della rete.
Tutte queste nuove tecniche permettono prestazioni migliori di classificazione.
Su slide esempio reale di struttura di deep learning per CIFAR-100.

Capitolo 8

Misura delle performance

Si studia quanto ci si può fidare di un modello e come classificare i risultati di un modello, per capire se è migliore di un altro.

8.1 Modelli supervisionati

Partiamo dagli approcci supervisionati.
Uno egli indicatori è legato alla misurazione dell'errore sui dati di training e testing (facendo training sul training e testing sempre sul training) ma questa non è una buona misura ma si fa per capire se il modello ha buone performance su dati "ovvi" ma questo non è garanzia di qualità, non studiando dati non osservati in fase di training. Si rischiano overfitting/underfitting (nel primo caso si impara molto bene sui dati di training ma quel modello appreso ha performance basse e complessità computazionale alta su dati nuovi mentre nel secondo caso il modello ha già performance basse e quindi risultati pessimi su nuovi dati, anche se si ha minor complessità computazionale, avendo magari meno parametri). Normalmente quindi si fa il testing su un testing set. Spesso è meglio avere un modello meno preciso ma con complessità computazionale adeguata al dominio in cui sto lavorando. Si hanno quindi varie misure di performance:

- error rate e accuratezza
- true/false positive/negative
- precisione, recall, F-Measure
- curve ROC
- complessità temporale

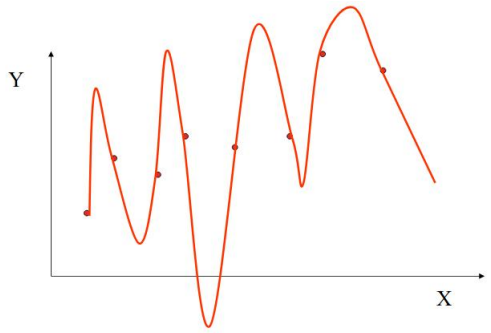


Figura 8.1: Esempio di modello in overfitting

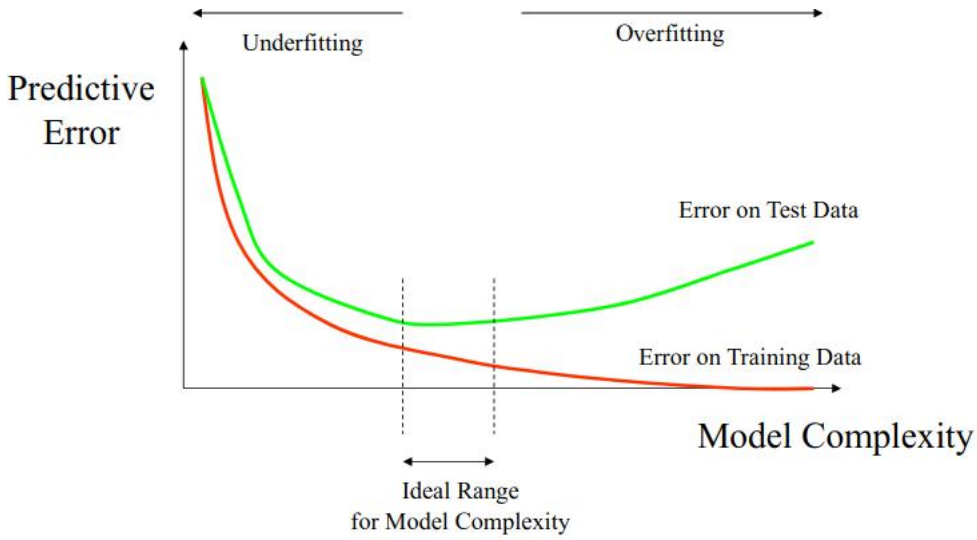


Figura 8.2: Andamento di overfitting/underfitting sulla qualità del modello, con indicato il range ideale. Spesso sull'asse delle x si ha il numero di iterazioni necessarie per capire quando terminarle per non andare in overfitting, come nel caso delle reti neurali

Si hanno quindi:

- successo, se le istanze sono predette correttamente
- errore, se le istanze non sono predette correttamente

Calcolando, per il primo punto:

- l'error rate come percentuale di errori commessi sull'intera serie di istanze
- accuratezza come proporzione di istanze classificate correttamente sull'intero set di istanze

Passiamo al secondo punto.
Normalmente si munta a minimizzare falsi positivi/negativi anche se tutte e 4 le entry della matrice di confusione andrebbero ottimizzate (con le entry che possono avere costi diversi a seconda del dominio, si pensi alle diagnosi mediche dove i falsi negativi sono più gravi dei falsi positivi).
In un problema multiclasse ho su righe e colonne le classi reali e le classi predette, rispettivamente, non avendo più chiari i veri/falsi positivi/negativi, che comunque possono essere ricavati per ogni classe.

Definizione 29. *Definiamo (in pratica per il caso binario):*
***accuratezza**:*

$$\frac{TP + TN}{TP + TN + FP + FN}$$

(che nel caso multiclasse è la somma dei valori sulla diagonale principale diviso la somma del resto)
***precisione**:*

$$\frac{TP}{TP + FP}$$

***recall**:*

$$\frac{TP}{TP + FN}$$

***F-measure**:*

$$\frac{2 \cdot precisione \cdot recall}{precisione + recall}$$

Queste sono le tradizionali misure globali.

Si costruiscono quindi le matrici di confusione a seconda del caso binario o multiclasse. Ma per il caso multiclasse posso calcolare solo la precisione di una singola classe, come anche per recall e F-measure.
Spesso quindi si lavora a livello di classe.

Definizione 30. *Si hanno, data una label l nell'insieme delle etichette L:*
***precisione**:*

$$P(l) = \frac{\#\ of\ instances\ correctly\ predicted\ as\ 1}{\#\ of\ instances\ predicted\ as\ 1}$$

***recall**:*

$$R(l) = \frac{\#\ of\ instances\ correctly\ predicted\ as\ 1}{\#\ of\ instances\ of\ class\ 1}$$

***F-measure** (ovvero tramite media armonica):*

$$F(l) = \frac{2 \cdot P(l) \cdot R(l)}{P(l) + R(l)}$$

Potrei avere un parametro β per dare più importanza ad una certa classe.

Si hanno modi per aggregare le misure di performance per scoprire comportamenti particolari.

Definizione 31. *Data un'etichetta l ho due tecniche di aggregazione:*
***Macro-average** di una misura di performance $Perf$:*

$$Perf^* = \frac{1}{|L|} \sum_{i=1}^{|L|} Perf(l)$$

dicendo che tutte le classi sono ugualmente importanti e hanno tutte lo stesso peso.
***Micro-average** dove ho una media pesata:*

$$Perf^* = \sum_{l=1}^{|L|} \frac{|class(l)|}{\#\ of\ istances} Perf(l)$$

Dando quindi più peso alle classi più "corpose" e predominanti.

Parliamo ora di curve **Receiver Operating Characteristic (*ROC*)**, che rappresentano delle curve che dimostrano la capacità discriminativa di un modello al variare di una certa soglia. Si confrontano veri/falsi positivi al variare di una certa soglia (che varia la variabile sperimentale). Variare la soglia permette di avere decisioni diverse e permette di costruire la curva ROC. In problemi con tante classi, soprattutto se alcune sotto dimensionate, si rischia di non riconoscerle queste classi ma abbassando la soglia si può aiutare il modello a riconoscerle. **Si ha una curva ROC per ogni classe** (con una soglia unica però). L'uso delle curve ROC è comodo per fare tuning

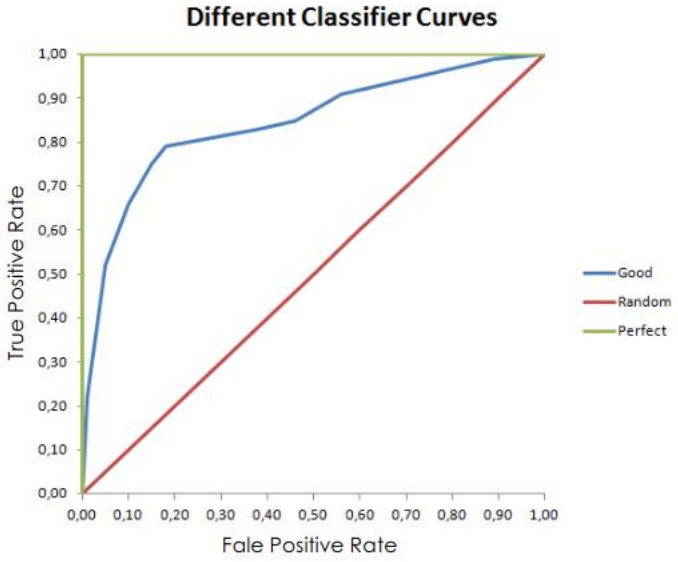


Figura 8.3: Esempio di curva ROC per una certa classe. Se una curva, associata ad un certo metodo, domina tutte le altre allora quello è il metodo migliore. Se vale per ogni classe non si hanno più dubbi.

su modelli indotti da dati molto sbilanciati. Si confrontano i vari modelli a seconda delle singole classi studiando l'impatto della soglia. Passiamo ora alla **learning curve**. Sono utili per studiare come evitare l'overfitting, confrontando la cardinalità degli esempi con una miusra di performance, come l'accuratezza. Ad un certo punto si arriva ad un plateau e quindi è inutile aumentare il numero di esempi (e la complessità computazionale) se tanto non si hanno miglioramenti. Non solo, si aumenta anche la probabilità di overfitting. Normalmente abbiamo lavorato con train e test (divisi circa per $\frac{2}{3}$ e $\frac{1}{3}$, rispettivamente, per poi trainare il modello sul training e testare i risultati sul test) ma questa è una visione parziale.
In generale più è grande il test set e più accurata la stima dell'errore. D'altro canto più è grande il training set e meglio sarà fatta la classificazione, al più di overfitting. Il test set non deve essere usato per il training e per il tuning ma si usa il validation set per il tuning, che quindi è un set aggiuntivo usato per ottimizzare i parametri (soglia, parametro di complessità della SVM etc...) quando il dataset è abbastanza grande da permetterlo. Il validation è solitamente il 10% del training set.
Se il dataset è piccolo magari la divisione appena detta non è rappresentativa ma si usano tecniche di **repeated holdout**, in cui si divide più volte usando di volta i dati nei vari set. Si usa la **cross-validation** e la

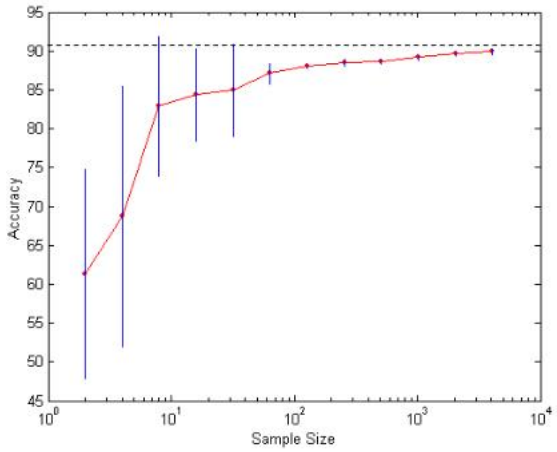


Figura 8.4: Esempio di learning curve.

k-fold-cross-validation dove si hanno due step:

1. i dati vengono suddivisi in k sottoinsieme di uguale dimensione
2. ogni sottoinsieme a sua volta viene utilizzato per il test e il resto per l'addestramento

pesso i sottoinsiemi vengono stratificati prima che venga eseguita la cross validation e viene calcolata la media delle stime di errore per fornire una stima complessiva dell'errore.
Per costruire una matrice di confusione su una time-fold-convalidation costruisco per ogni iterazione di folding la matrice di confusione dello specifico test set e alla fine le si somma per ottenere la matrice finale.
Una variante è la **stratified ten-fold cross-validation** in quanto esperimenti approfonditi hanno dimostrato che questa è la scelta migliore per ottenere una stima accurata (usare 10 fold) e avendo che la distribuzione delle istanze selezionate, rispetto alla classe da prevedere in ogni volta, è simile alla distribuzione originale del set di dati (per lo stratified). La stratificazione riduce la stima della varianza ma è ancora meglio ripetere tale stratificazione dieci volte.
Si hanno altre tecniche definite di **bootstrap** che utilizzano il campionamento con rimpiazzo dal training set, avendo che la stessa istanza può finire puiù volte nello stesso "esperimento" di time fold cross validation. Le istanze del dataset originale che non compaiono nel nuoo training set vengono usate come test. **Su slide 0.632 bootstrap, una tecnica empiricamente efficiente ma poco usata per la distorsione dei risultati a casua del suo pessimismo intrinseco**.

Una tecnica alternativa è **leave-one-out** che perfeziona il bootstrap. In questa tecnica si ha una k fold validation con k pari al numero delle istanze. Un'istanza sola viene lasciata per il test e tuutte le altre per il training, cambiando ad ogni iterazione quella di testing fino ad esaurimento. È computazionalmente pesante (n fold per n istanze)ma fa il miglior uso dei dati.