Machine Learning

UniShare

Davide Cozzi
@dlcgold

Indice

1 Introduzione **2**

2 Introduzione al ML **3**
2.1 Primi algoritmi 13
2.1.1 Algoritmo find-S 13
2.1.2 Algoritmi di eliminazione 16
2.2 Alberi decisionali 20
2.2.1 Algoritmo ID3 27

3 Reti neurali **31**
3.1 Neurone biologico 31
3.2 Neuroni formali 32
3.3 Reti neurali artificiali 33
3.3.1 Percettrone 35
3.3.2 Discesa lungo il gradiente 40
3.3.3 Reti multistrato 42

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

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

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:

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

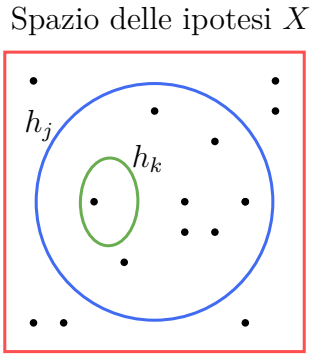


Figura 2.1: Rappresentazione di **più generale di** con i diagrammi di Eulero-Venn, dove si nota come, in X, $h_j \geq h_k$

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:*

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

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):*

$$|H|_{sem} = 1 + \prod_{A \in A_i} (|A| + 1)$$

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:*

$$|H|_{sint} = \prod_{A \in A_i} (|A| + 2)$$

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:*

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

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

  Possiamo dire che $G = \langle ?, ?, ?, \ldots ? \rangle$

- 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 $S = \langle \emptyset, \emptyset, \emptyset, \ldots \emptyset \rangle$

Ogni elemento di $VS_{H,D}$ si trova tra questi confini:

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

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*

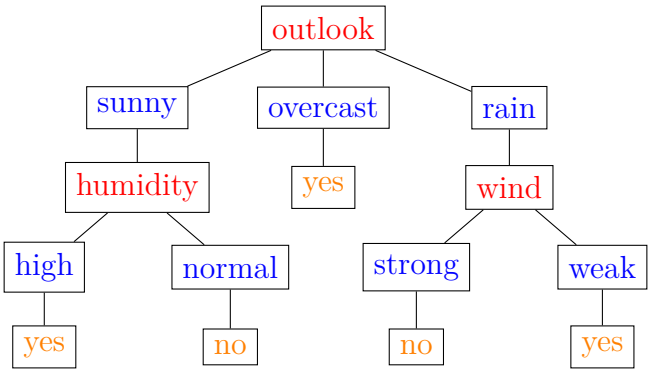


Figura 2.2: Esempio di albero decisionale

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.

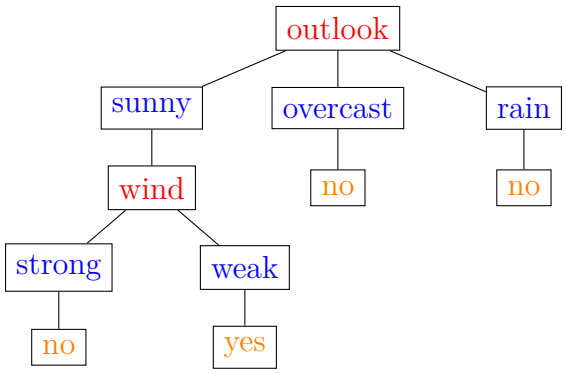


Figura 2.3: Esempio di albero decisionale per la formula $(Outlook = Sunny) \wedge (Wind = Weak)$

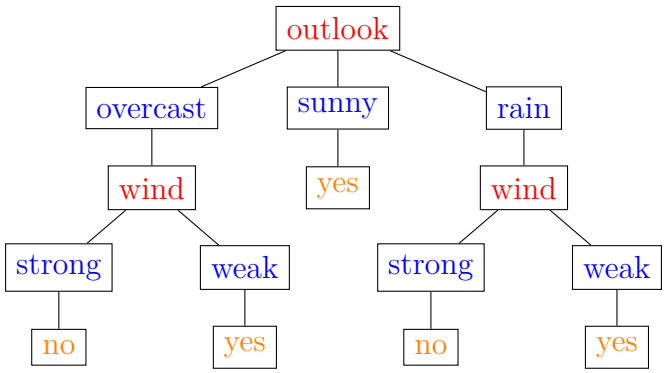


Figura 2.4: Esempio di albero decisionale per la formula $(Outlook = Sunny) \vee (Wind = Weak)$

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

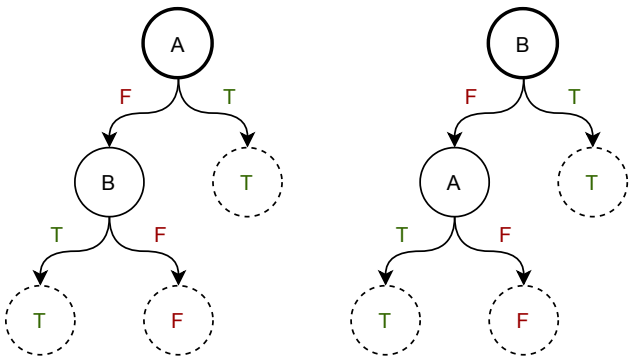
- **leef nodes (*foglie*)** etichettati con gli outcome della previsione

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 1. *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 2. *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.*

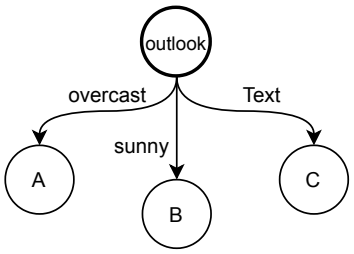
Vediamo quindi un algoritmo generale per la costruzione dell'albero:

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:



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 (o banalmente prendo il minimo tra il numero di *yes* e quello di *no*), 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:

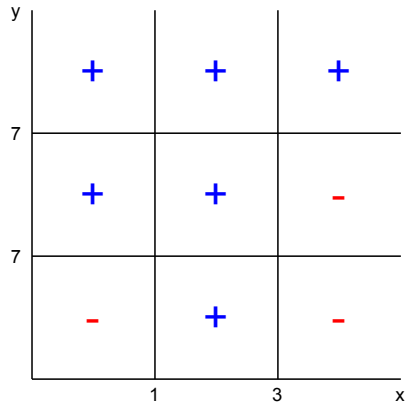
- *primo valore: 1 yes e 1 no*
- *secondo valore: 2 yes e 0 no*
- *terzo valore: 0 yes e 4 no*
- *quarto valore: 2 yes e 4 no*
- *quinto valore: 2 yes e 2 no*

avrei avuto:

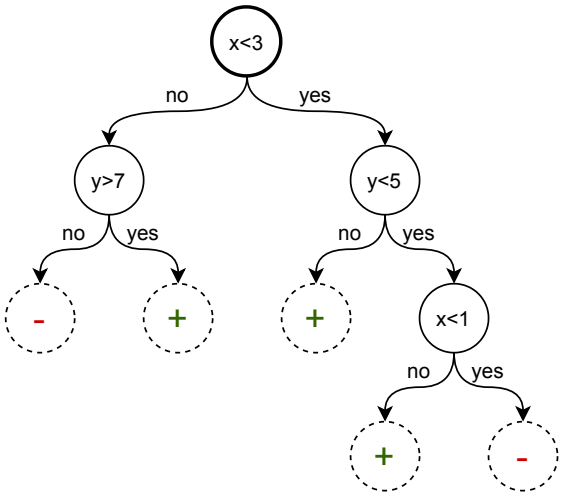
$$1 + 0 + 0 + 2 + 2 = 5$$

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:*



E si ottiene 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* $y > 5$*).*

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:

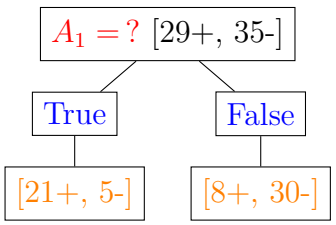
Algorithm 4 Algoritmo ID3

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

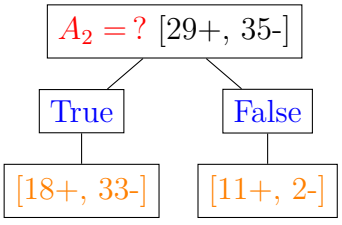
Bisogna in primis capire cosa si intende come **attributo migliore**. Per farlo introduciamo la seguente notazione:

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

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**. 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*)**

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**:

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

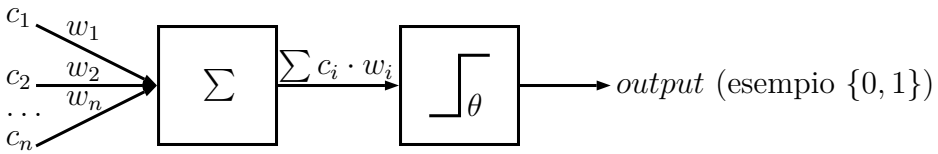


Figura 3.1: Rappresentazione matematica del modello di McColluch e Pitts, ovvero rappresentazione della **Linear Treshold Unit (*LTU*)**

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 9. *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 3 (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*

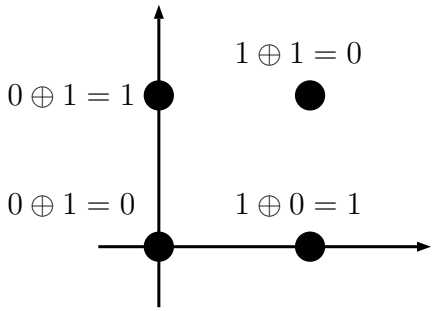
Teorema 4 (Teorema di Minsky e Papert)**.** *La classe delle forme discriminabili da un percettrone semplice è limitata alle forme linearmente separabili*

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.

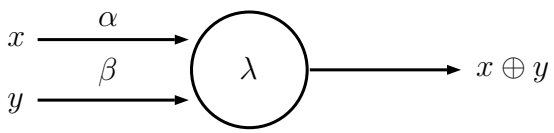
Esempio 19. *Vediamo un esempio che tratta l'addizione binaria, ovvero l'or esclusivo. Le possibili istanze sono rappresentate nel piano:*



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 5. *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

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:

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

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:

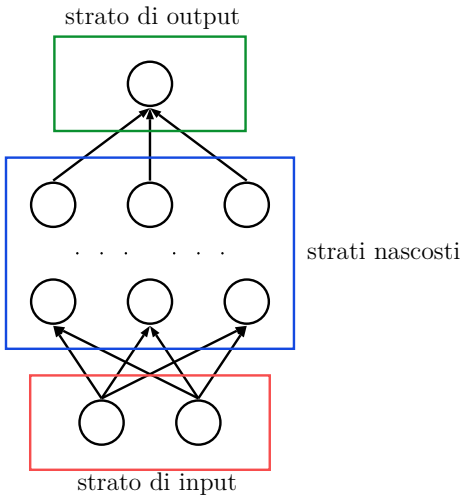


Figura 3.2: Rappresentazione stilizzata di rete multistrato

- 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

- 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.