Metodi di Interpolazione Polinomiale: Implementazione MATLAB e Applicazioni

Lai Stefano

Docente: Prof. Giuseppe Rodriguez

Tesina per il Corso di Calcolo Numerico 2, A.A. 2006/2007

2 marzo 2007

Introduzione

LE TECNICHE per l'approssimazione di funzioni ricoprono estrema importanza in numerose discipline: non di rado si ha a che fare con problemi in cui le funzioni esatte risultano particolarmente complesse da trattare (ad esempio, nel calcolo di integrali o di zeri), o addirittura in cui di tali funzioni siano noti solo dei valori discreti, come ad esempio accade nell'ambito delle misure sperimentali; in ambito ingegneristico, ancora, individuare l'andamento di una particolare funzione diviene fondamentale in problemi di sintesi, nei quali, a partire dalla conoscenza dei valori delle grandezze in ingresso si richiede di determinare una funzione di mappatura che associ agli ingressi le uscite desiderate.

In quest'ambito si inseriscono le tecniche di interpolazione, che permettono, dato un insieme di punti noto appartenenti ad una funzione incognita, di individuare una curva che in corrispondenza di tali punti assuma il valore della funzione stessa.

In questo lavoro si sono sviluppati in MATLAB alcuni dei più noti metodi di interpolazione polinomiale, dei quali si sono poi confrontate le prestazioni in termini di errore di interpolazione. In seguito sono descritte due possibili applicazioni di tali algoritmi: l'interpolazione di curve parametriche e la verifica di un approccio matematico al problema dell'apprendimento da esempi di alberi di decisione, problema che rientra nell'ambito del machine learning di agenti intelligenti.

L.S.

Indice

Capitolo 1

Metodi d'Interpolazione Polinomiale

1.1 Approccio Generale al Problema dell'Interpolazione

Siano assegnate $(n+1)$ coppie di numeri reali $((x_i, y_i), i = 1, \ldots, n)$: si definisce *interpolante* una funzione che in corrispondenza dei valori x_i assuma i valori y_i,

$$\phi(x_i) = y_i, \qquad i = 1, \ldots, n \tag{1.1}$$

Scelto quindi uno determinato spazio di funzioni $\mathcal{F}$ di dimensione n, con $\{\varphi_i\}$ base per tale spazio, si vuole approssimare una funzione $f(x)$ mediante un'interpolante $\phi \in \mathcal{F}$ e quindi esprimibile mediante una combinazione lineare delle $\{\varphi_i\}$,

$$\phi(x) = \sum_{j=0}^{n} \alpha_j \varphi_j(x) \tag{1.2}$$

Sostituendo le condizioni di interpolazione (1.1) in (1.2) si ottiene il sistema lineare

$$\phi(x_i) = y_i = \sum_{j=0}^{n} \alpha_j \varphi_j(x_i), \qquad i = 1, \ldots, n \qquad \Rightarrow \qquad \mathbf{\Phi} \cdot \alpha = \mathbf{y} \tag{1.3}$$

la cui soluzione $\alpha = (\alpha_1, \ldots, \alpha_n)^T$ è l'insieme dei coefficienti della combinazione lineare; si dimostra che la funzione di interpolazione esiste ed è unica se e solo se il determinante di $\mathbf{\Phi}$, detto *determinante di Haar*, è diverso da zero (**condizione di unisolvenza**), nel qual caso le funzioni $\{\varphi_i\}$ definiscono un *sistema di Chebychev* .

1.2 L'Interpolazione Polinomiale

Dall'Analisi Matematica sono noti due fondamentali teoremi, il *Teorema di Taylor* ed il *Teorema di Weierstrass*, che assicurano che una qualsiasi funzione $f \in \mathcal{C}^{(n+1)}_{[a,b]}$ (e dunque dotata di un certo grado di regolarità) possa essere approssimata mediante un polinomio, con un errore che può essere reso arbitrariamente piccolo al crescere del grado del polinomio scelto. Questo è chiaramente valido anche per un polinomio interpolante.

1.2.1 Il Polinomio Interpolante in Forma Canonica

Una prima possibile rappresentazione del polinomio interpolante è quella che sfrutta la base canonica di Π_n in una combinazione lineare del tipo (1.2),

$$p_n(x) = \sum_{j=0}^{n} \alpha_j x^j \tag{1.4}$$

e quindi, date le condizioni di interpolazione, potrà ricavarsi il sistema lineare

$$p_n(x_i) = y_i = \sum_{j=0}^{n} \alpha_j x_i^j, \qquad i = 1, \ldots, n \qquad \Rightarrow \qquad \mathbf{X} \cdot \alpha = \mathbf{y} \tag{1.5}$$

dove la matrice $\mathbf{X}$ è la *matrice di Vandermonde*,

$$\mathbf{X} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix}$$

il cui determinante può essere calcolato mediante la relazione

$$det(\mathbf{X}) = \prod_{\substack{i,j=0 \\ i>j}}^{n} (x_i - x_j) \tag{1.6}$$

Da quest'ultima equazione è semplice valutare che polinomio interpolante esiste unico se e solo se i punti di interpolazione sono tutti distinti (condizione necessaria e sufficiente affinchè sia verificata la condizione di unisolvenza).

Implementazione MATLAB

La valutazione dei coefficienti per la combinazione lineare nel sistema (1.4) può essere ricavata in MATLAB mediante poche e semplici righe di comando: la *function* `polycan.m`, il cui listato è riportato a pagina seguente, implementa tale algoritmo, sfruttando i punti di interpolazione contenuti nei vettori `xi` e `yi` e i nodi di valutazione del polinomio contenuti in `x`.

```
%calcolo della matrice di Vandermode
V = vander(xi);

%calcolo dei coefficienti del polinomio
c = V \ yi(:);

%polinomio interpolante in forma canonica
p = polyval(c, x);
```

Lo *script* `prova_polycan.m` permette di valutare tre diverse funzioni, $\sin(\pi x)$, $\cos(\pi x)$ e la funzione di Runge,

$$y = \frac{1}{1 + 25x^2}$$

nell'intervallo $[-1, 1]$, dato un certo numero di nodi di interpolazione e sfruttando 100 nodi di valutazione equispaziati: i grafici 1.1 e 1.2 riportano l'interpolazione della funzione seno per 4 e 8 nodi di interpolazione.

L'approccio così descritto, benchè funzionante, non risulta conveniente per alcuni motivi:

1. la matrice di Vandermonde risulta essere malcondizionata al crescere del numero dei nodi di interpolazione;

2. l'algoritmo risulta instabile (per piccole variazioni dei coefficienti della combinazione lineare si hanno grandi variazioni sul polinomio interpolante);

3. la complessità dell'algoritmo è $O(n^3)$.

1.2.2 Il Polinomio Interpolante di Lagrange

La causa dell'instabilità della forma canonica è da ricercarsi nell'instabilità della base scelta: un diverso approccio prevede quindi la valutazione del polinomio interpolante a partire da una base diversa da quella canonica.
Si definiscono **polinomi caratteristici di Lagrange** $\{L_j(x)\}_{j=0}^{n}$

$$L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \tag{1.7}$$

per i quali è semplice osservare che

$$L_j(x_i) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

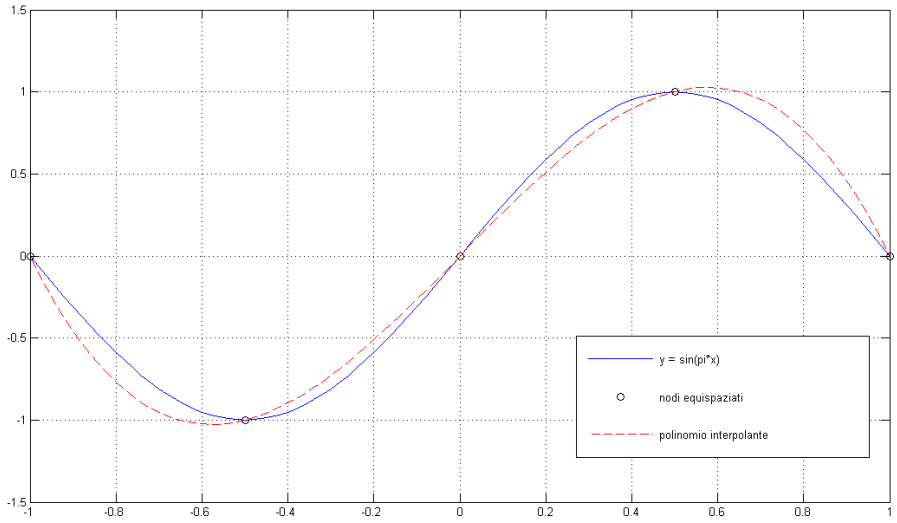

Figura 1.1: Interpolazione con Polinomio in forma canonica di $\sin(\pi x)$, 4 nodi di interpolazione

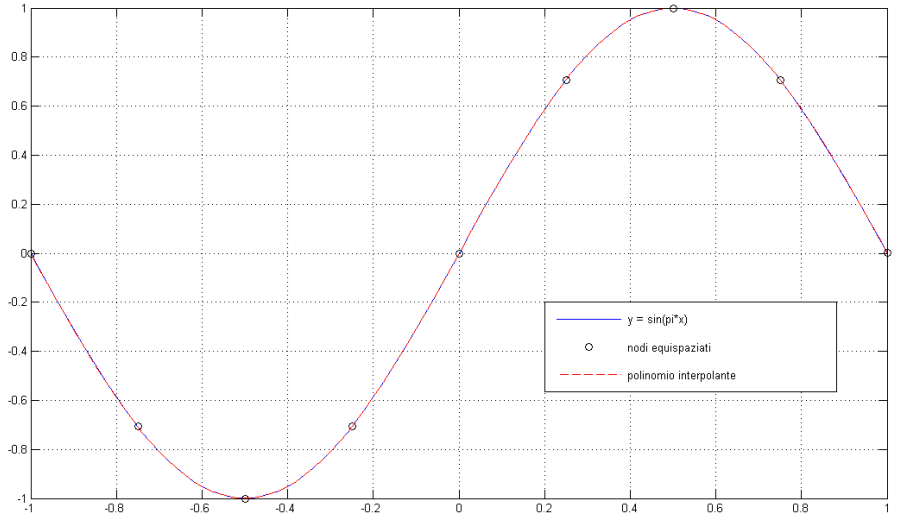

Figura 1.2: Interpolazione con Polinomio in forma canonica di $\sin(\pi x)$, 8 nodi di interpolazione

La matrice corrispondente a tale base è quindi una matrice identità, e il sistema di equazioni composto dai polinomi caratteristici di Lagrange è un sistema di Chebychev.

Il polinomio interpolante nella forma di Lagrange assume quindi la forma

$$p_n(x) = \sum_{j=0}^{n} y_j L_j(x) \tag{1.8}$$

dalla quale notiamo che, a diversamente dalla forma canonica, per la quale veniva scelta una base semplice e tutta la potenza di calcolo veniva impegata per il calcolo dei coefficienti, la base utilizzata è più complessa da calcolare e i coefficienti sono già noti.

Implementazione MATLAB

La *function* `polylag.m` implementa un possibile algoritmo per il calcolo del polinomio interpolante di Lagrange:

```
n = length(xi);
m = length(x);
sum = zeros(m,1);
for j = 1:m
    sum(j) = 0;
    for k = 1:n
        %Valutazione del j-esimo polinomio di Lagrange
        space = [1:k-1, k+1:n];
        pr = prod(x(j) - xi(space))/prod(xi(k) - xi(space));

        %j-esima componente del polinomio interpolante
        sum(j) = sum(j) + yi(k) * pr;
    end
end
```

Lo *script* `prova_polylag.m` presenta la medesima struttura dello *script* `prova_polycan.m` precedentemente descritto: le figure 1.3 e 1.4 mostrano l'interpolazione della funzione seno mediante polinomio di Lagrange per 4 e 8 nodi di interpolazione rispettivamente, con 100 punti di valutazione equispaziati. I risultati sono del tutto simili a quelli già ottenuti con il polinomio in forma canonica, ma in tal caso vengono eliminati i problemi legati al condizionamento e alla stabilità; l'algoritmo implementato presenta una complessità $O(n^2)$.

1.2.3 La Formula di Neville

Metodi come quelli descritti in precedenza permettono di ricavare l'intero polinomio interpolante: esistono dei problemi, come quelli di *estrapolazione*, nei quali si è interessati a conoscere tale polinomio solo in pochi punti, e nel qual caso è quindi preferibile adottare una tecnica diversa. La *formula*

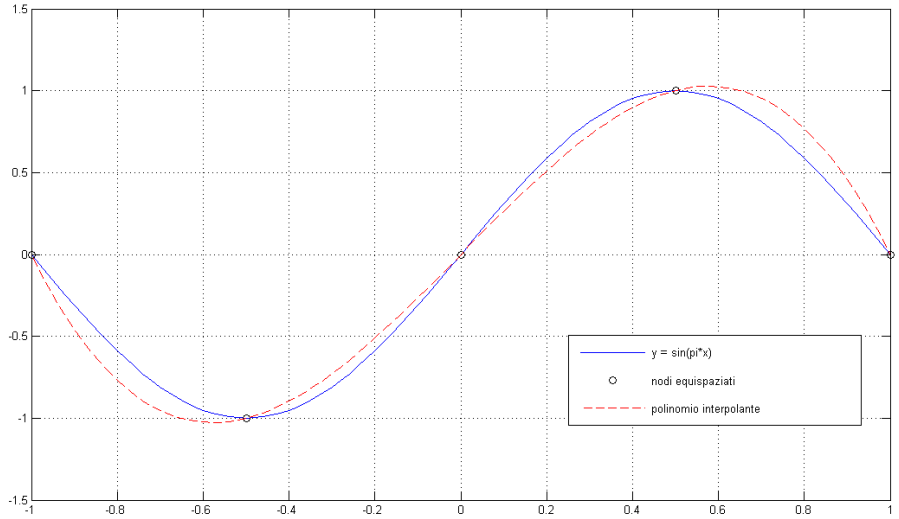

Figura 1.3: Interpolazione con Polinomio di Lagrange di $\sin(\pi x)$, 4 nodi di interpolazione

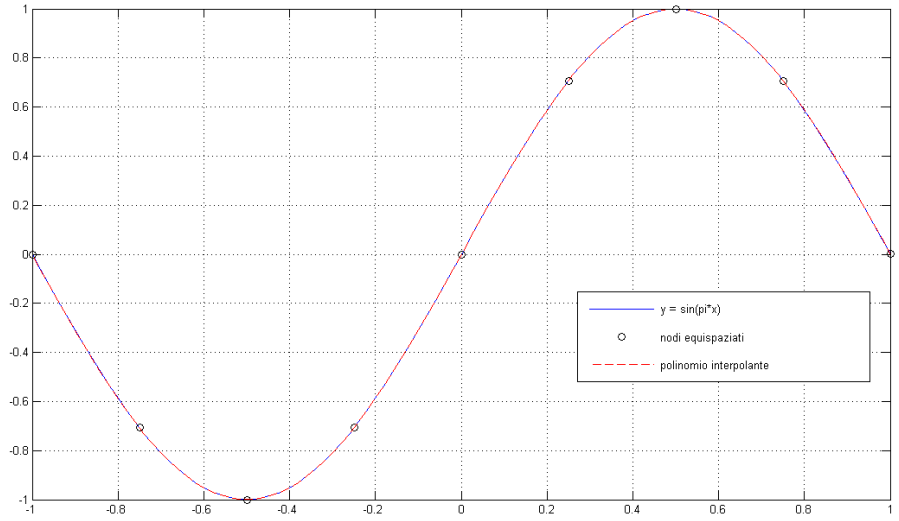

Figura 1.4: Interpolazione con Polinomio di Lagrange di $\sin(\pi x)$, 8 nodi di interpolazione

di Neville permette di valutare il polinomio interpolante in un punto (senza costruirlo esplicitamente) a partire da polinomi interpolanti su un numero inferiore di punti.

Si può dimostrare che un polinomio $Q_{r,r+1,\ldots,r+k}(x)$ interpolante $k+1$ punti $\{x_r, x_{r+1}, \ldots, x_{r+k}\}$ può essere definito a partire da due polinomi

interpolanti k punti mediante la relazione

$$Q_{r,r+1,\ldots,r+k}(x) = \frac{(x - x_r) \cdot Q_{r+1,r+2,\ldots,r+k}(x) - (x - x_{r+k}) \cdot Q_{r,r+1,\ldots,r+k-1}(x)}{x_{r+k} - x_r} \tag{1.9}$$

Dati i punti di interpolazione $(x_i, y_i), i = 1, \ldots, n$, è possibile utilizzare la relazione (1.9) per costruire iterativamente una **schema di Neville** (vedi figura 1.5), mediante il quale, posti al primo passo $Q_i = y_i$, si può valutare il polinomio interpolante $Q_{0,1,\ldots,n}(x)$.

Implemetazione Matlab

La valutazione del polinomio interpolante in un punto *alpha* può avvenire mediante una semplice implementazione della (1.9), della quale è dato un esempio nella *function* `neville.m`:

```
q = yi;
n = length(xi);
for k = 2:n
    for j = 1:n-k+1
        a(j) = ((alpha - xi(j))*q(j+1);
        b(j) = (alpha - xi(j+k-1))*q(j));
        q(j) = (a(j) - b(j))/(xi(j+k-1) - xi(j));
    end
end
pi = q(1)
```

Si noti che per l'implementazione dello schema viene utilizzato un solo vettore, le cui componenti vengono progressivamente riscritte dall'alto; la complessità di tale algoritmo è $O(\alpha n^2)$.
Nella *function* `polynev.m` viene costruito il polinomio interpolante su tutti i punti di valutazione, richiamando su ogni singolo x la funzione precedentemente descritta. Lo *script* `prova_polynev.m` permette di valutare l'andamento del polinomio interpolante così costruito: vista la complessità della valutazione dello schema, tale algoritmo non è comunque il più indicato per la valutazione dell'intero polinomio.

1.2.4 Il Polinomio Interpolante di Newton

Lo schema descritto dalla formula di Neville può comunque essere utilizzato per valutare il polinomio interpolante su n punti una volta costruito quello su $n - 1$ punti $p_{n-1}(x)$ mediante una relazione del tipo

$$p_n(x) = p_{n-1}(x) + g_n(x) \tag{1.10}$$

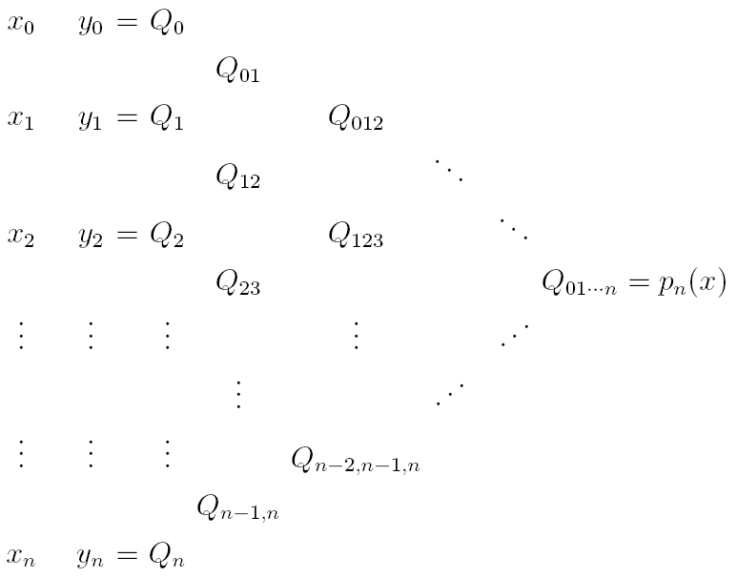

Figura 1.5: Schema di Neville

dove $g_n(x)$ deve essere anch'esso interpolante gli $n-1$ punti, e quindi nella forma

$$g_n(x) = a_n \cdot \prod_{i=0}^{n-1}(x - x_i) = a_n \cdot \omega_{n-1}(x) \tag{1.11}$$

con

$$a_n = \frac{y_n - p_{n-1}(x_i)}{\omega_{n-1}(x)} \tag{1.12}$$

come è possibile ricavare dall'applicazione delle condizioni d'interpolazione (1.1) sull'n-esimo punto. Sfruttando la relazione (1.9) è semplice ricavare che il coefficiente a_n può essere scritto mediante le **differenze divise**

$$a_n = f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0} \tag{1.13}$$

Lo schema delle differenze divise è del tutto analogo a quello di Neville, per cui, ponendo $y_i = f[x_i]$, si può sfruttare la (1.13) per valutare tutte le differenze divise secondo uno schema del tipo in figura 1.5.
Il polinomio interpolante di Newton si ottiene a partire da coefficienti calcolati con lo schema delle differenze divise come

$$\begin{aligned} p_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + \ldots + \\ &\quad + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1}) \end{aligned} \tag{1.14}$$

Implementazione MATLAB

Il calcolo del polinomio di Newton è implementato mediante le *functions* MATLAB `new_coeff.m`, `new_val.m` e `polynew.m`: la prima valuta i coefficienti del polinomio mediante lo schema delle differenze divise (genera un vettore dei coefficienti c coincidente con la prima riga di una matrice triangolare superiore); la seconda sfrutta il vettore dei coefficienti così creato calcolare il polinomio in un punto di valutazione *alpha* (mediante la *subroutine* `prdtc.m`[1]); la terza richiama iterativamente la `new_val.m` sull'intero vettore dei punti di valutazione.

```
% NEW_COEFF.M

n = length(xi);
q = zeros(n);
q(:, 1) = yi';
for j = 2:n
    for i = 1:n-j+1
        tmp(i,j) = (q(i, j-1) - q(i+1, j-1));
        q(i,j) = tmp(i, j)/(xi(i) - xi(j+i-1));
    end
end
c = q(1, :);

%NEW_VAL.M

sum = 0;
for i=1:length(xi)
    sum = sum + c(i)*prdct(alpha, xi, i);
end
pi = sum;

%POLYNEW.M

c = new_coeff(xi, yi);
for i = 1:length(x)
    p(i) = new_val(x(i), xi, c);
end
p1 = p';
```

[1] tale calcolo può essere eseguito anche mediante l'*algoritmo di Horner*, implementato nella *function* `horner.m`

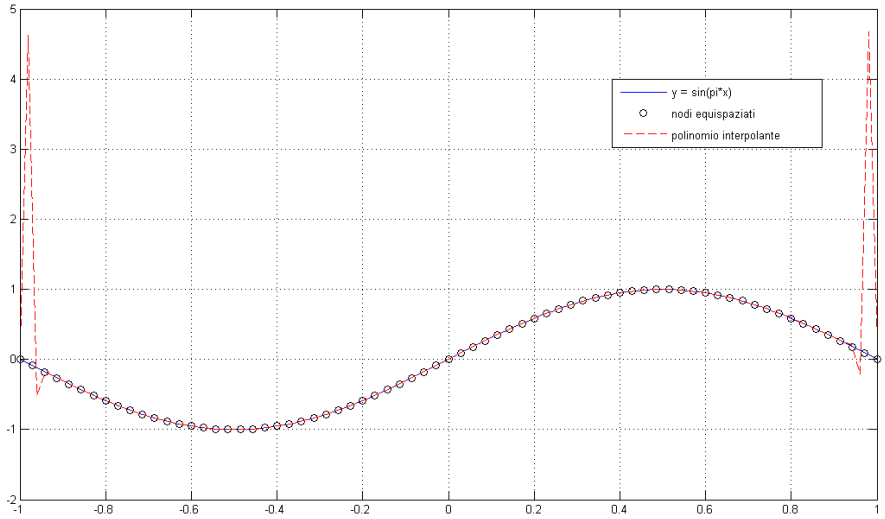

Figura 1.6: Interpolazione con Polinomio di Newton di $\sin(\pi x)$, 70 nodi di interpolazione

Il grafico in figura 1.6 mostra il limite fondamentale dell'interpolazione con nodi equispaziati: per un elevato numero di nodi di interpolazione (i.e., per nodi di interpolazione troppo vicini tra loro) si nota una certa instabilità dell'algoritmo, il che comporta un errore non strascurabile sull'andamento del polinomio interpolante.

1.3 L'Errore di Interpolazione

Si definisce **errore di interpolazione** per l'interpolazione polinomiale la differenza tra la funzione da interpolare $f(x)$ e il polinomio interpolante $p_n(x)$: si dimostra che se $f \in \mathcal{C}^{(n+1)}_{[a,b]}$ allora $\exists \xi_x \in \mathcal{I}_{(x_0,x_1,\ldots,x_n)}$, con $\{x_1, x_2, \ldots, x_n\}$ punti di interpolazione, tale che l'errore di interpolazione possa essere espresso come

$$E_n(f) = f(x) - p_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\omega_n(x) \tag{1.15}$$

Tale relazione porta con se due importanti informazioni: se la quantità $f^{(n+1)}(\xi_x)$ è certamente maggiorabile mediante una costante, essendo la funzione di classe $\mathcal{C}^{(n+1)}$, la presenza del polinomio $\omega_n(x)$ implica che la distribuzione scelta per i nodi di interpolazione influisce sull'entità dell'errore (si ricordi che $\omega_n(x)$ è un polinomio nella forma espressa in (1.11)). Dunque, se il Teorema di Weierstrass ci assicura che per una qualsiasi funzione continua esiste almeno una distribuzone di nodi che consenta di avere

convergenza a zero dell'errore di interpolazione per $n \to \infty$, il risultato testè ottenuto suggerisce che per ogni distribuzione di nodi esiste almeno una funzione per la quale l'errore d'interpolazione in norma-∞ non tenda a zero per $n \to \infty$ (*Teorema di Faber*).
È quindi lecito chiedersi quale sia la distribuzione di nodi ottimale per il problema dell'interpolazione polinomiale: dallo studio del problema di ottimo

$$\min_{\{x_i\}} \max_{x \in [-1,1]} |\omega_n(x)| \tag{1.16}$$

si è ottenuto (*Bernstein*) che per una funzione $f \in \mathcal{C}^1_{[a,b]}$ l'errore di interpolazione tende a zero per $n \to \infty$ se i nodi di interpolazione sono gli zeri del *polinomio di Chebychev*,

$$\begin{cases} T_{n+1}(x) = \cos[(n+1)\theta] \\ x = \cos\theta, \qquad\qquad \theta \in [0, \pi] \end{cases} \tag{1.17}$$

Gli zeri di tali polinomi, valutabili ponendo pari a zero la seconda delle (1.17),

$$x_k = \cos\left(\frac{2k+1}{n+1}\frac{\pi}{2}\right), \qquad k = 0, \ldots, n \tag{1.18}$$

si dimostrano essere la distribuzione di nodi ottimale per il problema dell'interpolazione polinomiale, mentre i nodi equispaziati sfruttati fino ad ora rappresentano la peggiore delle distribuzioni possibili.

1.3.1 Studio dell'Errore di Interpolazione con MATLAB

I nodi di Chebychev sono calcolati tramite la *function* `cheby.m`, che implementa semplicemente la (1.18) per un n fornito in input.

```
c = [0:n-1]';
c = cos((2*c + 1)/(n + 1)*(pi/2));
```

Lo *script* `Pol_Interp.m` permette di sfruttare i metodi di interpolazione fin qui descritti sulle tre funzioni già introdotte, mostrando oltre al risultato dell'interpolazione l'andamento dell'errore $E_n(f)$ e di $\|E_n(f)\|_\infty$ al crescere del grado n del polinomio interpolante; nei grafici che seguono sono mostrate tali quantità nel caso dell'interpolazione con polinomio di Lagrange della funzione di Runge su 30 nodi equispaziati e 30 nodi di Chebychev. É immediato notare dalle figure 1.7 e 1.8 che i nodi di Chebychev forniscono all'interpolazione maggiore stabilità; dal grafico 1.9 si osserva come l'errore in norma-∞ tenda a divergere nel caso di nodi equispaziati anche per gradi di polinomio non troppo elevati, mentre, benchè inizialmente più elevato, questo continui a decrescere nel caso di nodi di Chebychev.

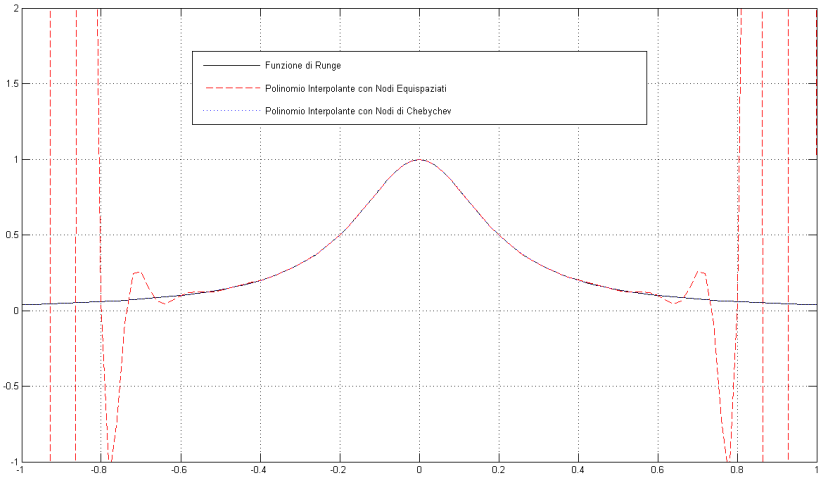

Figura 1.7: Interpolazione della funzione di Runge, polinomio di Lagrange, 30 nodi

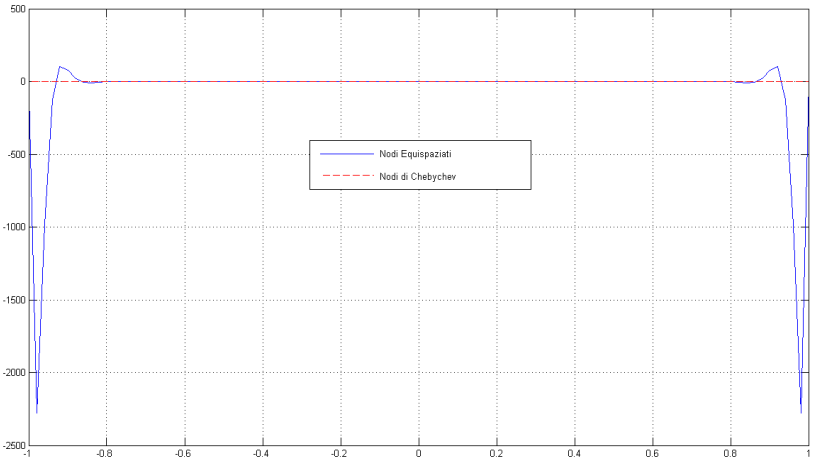

Figura 1.8: Andamento dell'errore di interpolazione nel caso di figura 1.7

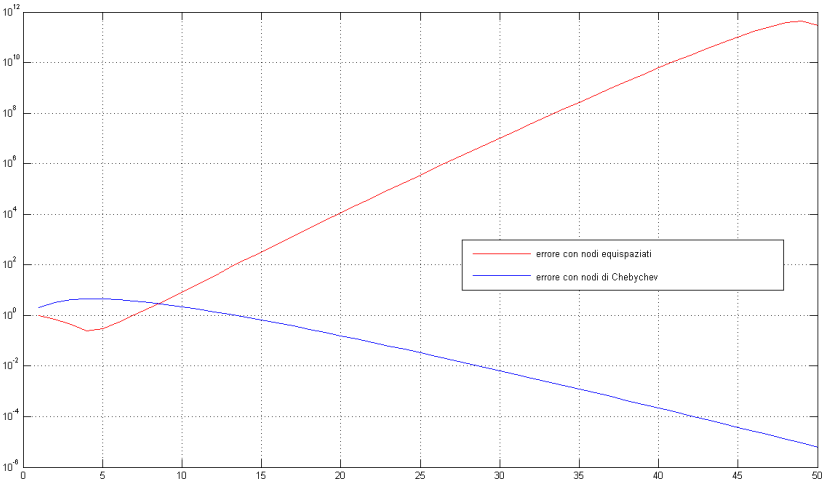

Figura 1.9: Andamento di $\|E_n(f)\|_\infty$ nel caso di figura 1.7 al crescere di n

Capitolo 2

Interpolazione di Curve Parametriche

Fino a questo momento abbiamo visto come le tecniche di interpolazione polinomiale permettano di valutare l'andamento di funzioni delle quali i punti noti fossero posti in serie: se la funzione da interpolare è parametrica e l'insieme dei suoi punti disegna quindi nello spazio delle curve chiuse, non è possibile sfruttare direttamente i metodi fin qui analizzati; provando, ad esempio, a sfruttare il polinomio interpolante di Lagrange implementato in `polylag.m` su 100 nodi di valutazione, e su un insieme di punti come quello in figura 2.1, vengono generati un numero consistente di errori dovuti a divisioni per zero.

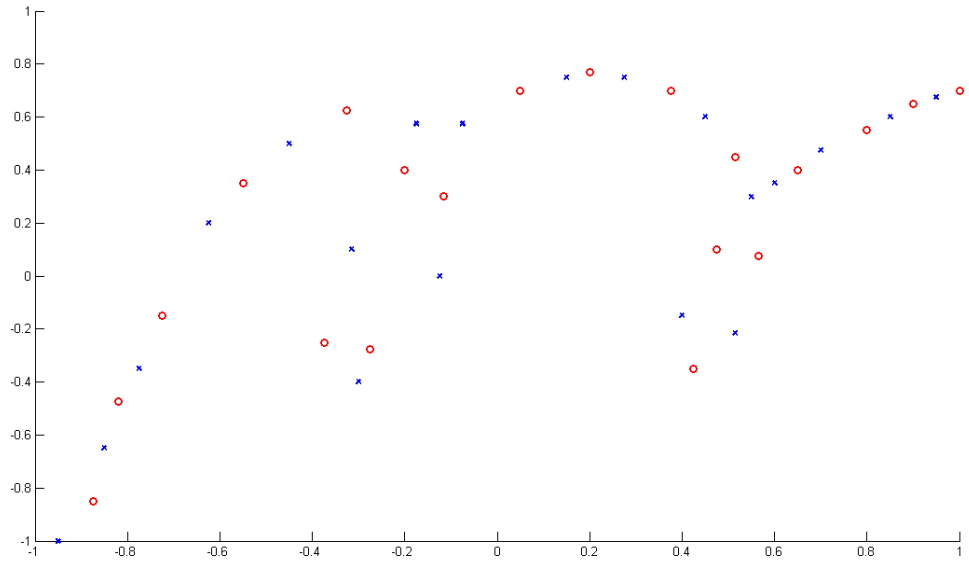

Figura 2.1: Esempio di distribuzione di nodi di interpolazione

2.1 Una Possibile Soluzione

Una possibile soluzione consiste nello sfruttare una determinata parametrizzazione per compiere l'interpolazione separatamente sui punti x_i e y_i.

Siano dati i punti di interpolazione $P_i = (x_i, y_i), i = 0, \ldots, n$ che seguano un andamento come quello in figura 2.1, e sia t una parametrizzazione composta da punti equispaziati o dagli zeri del polinomio di Chebychev: mediante una delle tecniche di interpolazione viste in precedenza, si costruiscano due polinomi, $p_n(x)$ e $q_n(y)$, che interpolino rispettivamente i punti (t_i, x_i) e (t_i, y_i), $i = 0, \ldots, n$, ovvero tali che

$$\begin{cases} p_n(t_i) = x_i \\ q_n(t_i) = y_i \end{cases} \qquad i = 0, \ldots, n \tag{2.1}$$

È evidente che tale scelta non modifica il problema d'interpolazione, poichè l'insieme definito dall'interpolazione (2.2) è composto da punti $R_i = (p_n(t_i), q_n(t_i)) = (x_i, y_i) = P_i, i = 0, \ldots n$; compiendo l'interpolazione su tutti i punti, si ricava la definizione parametrica del polinomio interpolante,

$$\begin{cases} x = p_n(t) \\ y = q_n(t) \end{cases} \tag{2.2}$$

2.1.1 Implementazione MATLAB

Lo *script* `Par_Interp.m` consente di implementare la strategia descritta in precedenza sfruttando uno a scelta dei metodi di interpolazione descritti nel Capitolo 1 su vari insiemi di punti, tra i quali quelli relativi alla distribuzione di figura 2.1 contenenti diversi numeri di punti (15, 20, 30 e 40); i punti di valutazione sono 100 equispaziati, mentre i vettori `t_eq` e `t_ch` contengono i punti della parametrizzazione, rispettivamente equispaziati e di Chebychev.

```
%Punti di Valutazione e Numero di Nodi di Interpolazione

N = 100;
x = [0:N]'/N*2-1;
n = length(xi)-1;

%Parametrizzazione (Punti Equispaziati e Punti di Chebychev)
t_eq = [0:n]'/n*2-1;
t_ch = cheby(1+n);

%Polinomi Interpolanti su X e Y (Punti Equispaziati)
p_x_eq = g(x, t_eq, xi);
```

```
p_y_eq = g(x, t_eq, yi);

%Polinomi Interpolanti su X e Y (Punti di Chebychev)
p_x_ch = g(x, t_ch, xi);
p_y_ch = g(x, t_ch, yi);


%Grafici
plot(xi, yi, 'or', p_x_eq, p_y_eq);
xlabel('x'), ylabel('y');
title('Interpolazione con Tempi Equispaziati');
grid
axis([-1 1 -1.5 1.5]);

figure, plot(xi, yi, 'or', p_x_ch, p_y_ch);
xlabel('x'), ylabel('y');
title('Interpolazione con Tempi di Chebychev');
grid
axis([-1 1 -1.5 1.5]);
```

I grafici 2.2 e 2.3 mostrano il risultato dell'interpolazione sfruttando polinomi di Lagrange per 30 punti di parametrizzazione: la scelta di punti equispaziati è palesemente infelice, e risulta tanto peggiore quanto più è alto il numero dei punti; molto più fedele è l'andamento sfruttando una parametrizzazione composta dagli zeri di Chebychev.
Questi risultati, che risultaano essere per gli insiemi di punti scelti del tutto simili per tutti e quattro i metodi d'interpolazione implementati, risultano ancora più chiari sfruttando una diversa distribuzione dei nodi riportata nei grafici 2.4 e 2.5: la scelta dei nodi equispaziati risulta essere poco performante rispetto a quella dei nodi di Chebychev anche per un numero relativamente basso di nodi e per un andamento più semplice della curva da interpolare.

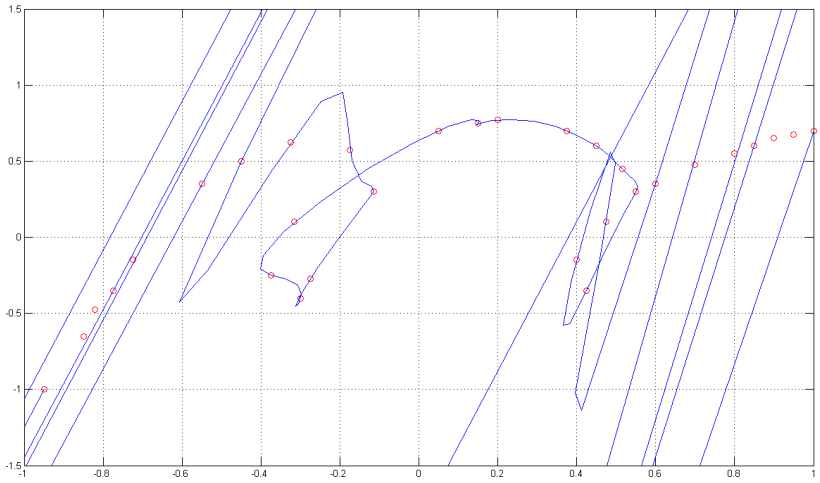

Figura 2.2: Interpolazione di una Curva Parametrica (30 punti di parametrizzazione equispaziati)

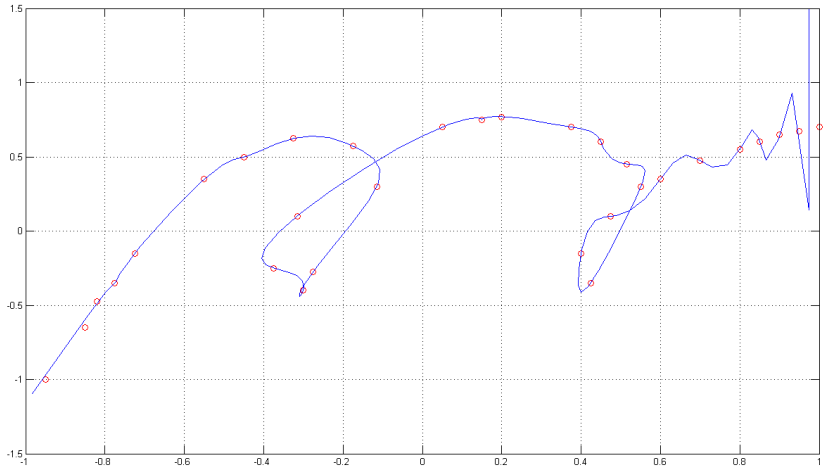

Figura 2.3: Interpolazione di una Curva Parametrica (30 punti di parametrizzazione di Chebychev)

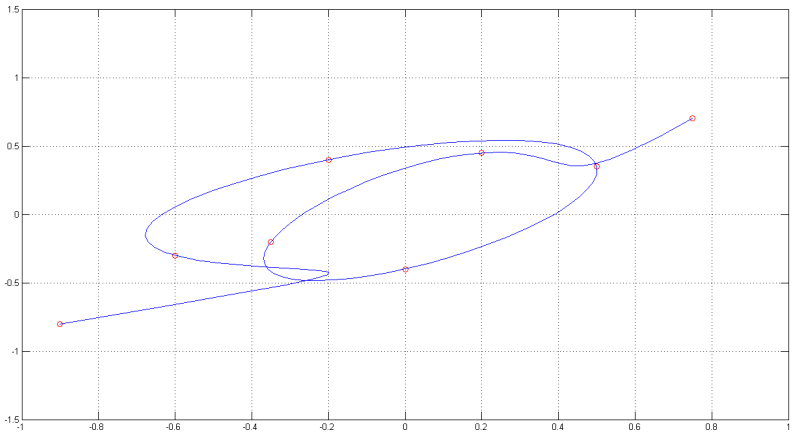

Figura 2.4: Interpolazione di una Curva Parametrica (8 punti di parametrizzazione equispaziati)

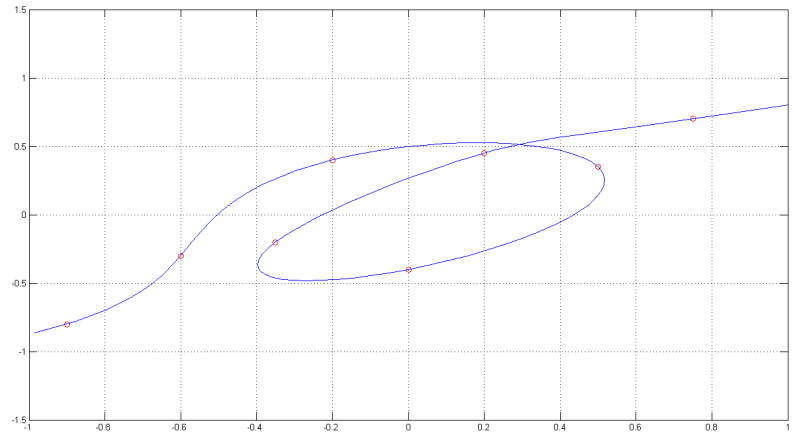

Figura 2.5: Interpolazione di una Curva Parametrica (8 punti di parametrizzazione di Chebychev)

Capitolo 3

Verifica di Metodi di *Machine Learning*

3.1 Una Breve Introduzione All'Intelligenza Artificiale

La tecnologia degli ultimi venti anni è permeata dai risultati di ricerche che affondano le proprie radici nei primi anni Cinquanta, e divenute note sotto il segno di un nome ingombrante: Intelligenza Artificiale.
Lo scopo di tale disciplina, nata dall'incontro di automatica, informatica e logica, era quello di creare delle entità, dette *agenti*, in grado di agire in modo intelligente, ovvero di scegliere la miglior azione possibile allo scopo di risolvere un determinato problema in relazione alle percezioni sull'ambiente esterno ed alla conoscenza del problema stesso; altra caratteristica fondamentale di un agente intelligente deve essere la flessibilità, che gli permetta di adeguare le proprie azioni in funzione di cambiamenti dell'ambiente esterno e migliorare le proprie prestazioni mediante una valutazione diretta dei risultati del proprio agire. Quest'ultimo problema rientra nello studio di algoritmi che permettano agli agenti intelligenti di apprendere dall'esperienza: tale campo è noto con il nome di *machine learning*.

3.2 Problemi di Inferenza Induttiva e Apprendimento da Esempi

Un agente intelligente è, nella concezione più generale possibile, una funzione che, presi determinati ingressi, produce delle uscite: tale funzione di mappatura che in generale non è completamente nota al momento della progettazione, ma che può essere mogliorata mediante l'apprendimento dall'agente stesso.
In IA sono di grande interesse problemi di *apprendimento da esempi*, che

consistono nel determinare o approssimare una funzione f ignota a partire da un insieme di *esempi* del suo comportamento I/O (un esempio è definito come una coppia $(x, f(x))$, dove x è un determinato valore dell'ingresso ed $f(x)$ è il corrispondente valore della funzione); questo è un problema di *inferenza induttiva* (dato un insieme di esempi di una funzione ignota f, trovare una funzione h appartenente ad un insieme predefinito H, detto *spazio delle ipotesi*, in grado di approssimare f); la maggiore difficoltà nei problemi di inferenza induttiva è chiaramente determinare se una data ipotesi h si una buona approssimazione di f (in tal senso è valido il *rasoio di Ockham*: l'ipotesi più semplice è anche la più verosimile) .
L'approssimazione di funzioni matematiche mediante interpolazione rappresenta un tipico caso di apprendimento induttivo.

3.3 Apprendimento da Esempi per Alberi di Decisione

Un *albero di decisione* (*decision tree, DT*) rappresenta una funzione in cui l'ingresso consiste nella descrizione di un singolo oggetto in termini di un insieme predefinito di *attributi* (i cui valori possono essere discreti o continui e anche non numerici) e l'uscita è un singolo valore discreto o continuo, anche non numerico; ogni nodo dell'albero rappresenta un test sul valore di un attributo (ed ogni arco da esso uscente rappresenta una possibile uscita del test), e ad ogni foglia è associato un valore della funzione. Per un dato valore degli ingresso, il valore della funzione si ottiene seguendo il percorso dal nodo radice a una delle foglie, per il quale tutti i test sono soddisfatti. Nel caso più comune di uscita discreta la funzione rappresentata dal DT, detta *ipotesi*, può essere vista come la descrizione di più classi di oggetti (corrispondenti ai diversi valori dell'uscita) in funzione dei valori dell'insieme di attributi; se l'uscita può poi assumere solo due valori, il DT risulta equivalente ad una funzione booleana, e in tal caso la funzione da apprendere è detta *goal predicate*.
Nei problemi di apprendimento automatico di DT l'insieme degli esempi è detto *training set* un esempio è costituito dai valori degli attributi e dal corrispondente valore della funzione; nel caso di funzioni booleane,se il valore del *goal predicate* è *Yes* (*No*), l'esempio si dice *positivo* (*negativo*). L'obiettivo principale di un algoritmo di apprendimento è costruire un'ipotesi consistente con gli esempi del *training set*, tale cioè da classificarli tutti correttamente: nel caso dei DT non è sufficiente tuttavia memorizzare un nodo per ogni esempio, ma deve essere raggiunto un compromesso tra consistenza dell'albero con l'insieme degli esempi e *capacità di generalizzazione* (i.e., capacità di classificare correttamente anche altri oggetti).

L'algoritmo che si utilizza per costruire un albero di decisione a partire da un dato *training set* E e da un insieme di attributi X si basa sulla

scelta, ad ogni passo, dell'*attributo più discriminante* rispetto all'insieme di esempi corrente, ovvero di quell'attributo i cui valori assumibili facciano essenzialmente riferimento ad una ben determinata classe di esempi; se X_r è l'attributo che rispetta tale regola per il *training set* iniziale, allora diviene nodo radice del DT, e lo stesso *training set* viene suddiviso in m sottoinsiemi E_i, se tale attributo assume m diversi valori.
A questo punto tale procedura viene iterata su ogni singolo insieme E_i così prodotto, seguendo una semplice regola ricorsiva:

1. **se in E_i ci sono sia esempi positivi che negativi** si crea un nodo scegliendo il miglior attributo X_k tra quelli non usati nel percorso tra la radice e il nodo padre e si riapplica ricorsivamente l'algoritmo su ogni sottoinsieme di esempi E_k corrispondente alle diverse uscite del test su X_k;

2. **se tutti gli esempi in E_i sono positivi (negativi)** si crea una foglia e la si etichetta con *Yes* (*No*);

3. **se in E_i ci sono sia esempi positivi che negativi ma non sono rimasti attributi**, significa che nel *training set* ci sono esempi di classe diversa ma con gli stessi valori degli attributi (causato da rumorosità dei dati, attributi non sufficienti a distinguere tra le due classi, dominio non deterministico), e in tal caso si può creare una foglia e assegnarle l'etichetta della classe più rappresentata.

Per determinare la capacità di discriminazione di un attributo rispetto ad un dato *training set* viene valutato il suo cosiddetto *guadagno d'informazione*: se un attributo X ad m valori assumibili presenta un numero p di esempi positivi ed un numero n di esempi negativi con riferimento al *training set* E, e quindi $p/(p+n)$ ed $n/(p+n)$ sono le corrispondenti frequenze relative, e per ogni singolo valore sia valutabile un sottoinsieme E_i del *training set*, tale per cui siano p_i ed n_i il numero di esempi positivi e negativi in corrispondenza di ogni uscita, allora viene definita guadagno d'informazione per l'attributo X la quantità

$$\begin{aligned} G(X) &= I\left(\tfrac{p}{p+n}, \tfrac{n}{p+n}\right) - I_A = \\ &= I\left(\tfrac{p}{p+n}, \tfrac{n}{p+n}\right) - \tfrac{p_i+n_i}{p+n} \textstyle\sum_{i=1}^{m} I\left(\tfrac{p_i}{p_i+n_i}, \tfrac{n_i}{p_i+n_i}\right) \end{aligned} \tag{3.1}$$

dove

$$I\left(\frac{\alpha}{\alpha+\beta}, \frac{\beta}{\alpha+\beta}\right) = -\frac{\alpha}{\alpha+\beta}\log_2\frac{\alpha}{\alpha+\beta} - \frac{\beta}{\alpha+\beta}\log_2\frac{\beta}{\alpha+\beta} \tag{3.2}$$

determina il contenuto informativo di un determinato insieme di simboli; si pone convenzionalmente $I(1,0) = I(0,1) = 0$ (una sorgente in cui un simbolo è certo non ha contenuto informativo).
In definitiva il guadagno d'informazione mostra quale sia il contenuto informativo di un attributo dopo la valutazione di ogni sua possibile uscita: l'attributo che per un dato *training set* presenta il guadagno d'informazione più alto è anche il più discriminante (perchè la quantità I_A è certamente la più piccola, se molti dei valori assumibili dall'attributo presenta esempi di una sola classe).

3.4 Un Approccio matematico al Problema?

Un DT a due sole uscite è equivalente ad una funzione booleana, presentandone la medesima espressività: per questa ragione gli alberi di decisione sono utilizzati per l'apprendimento e la semplificazione di funzione logiche complesse, il cui comportamento può essere ricostruito a partire dai rami dell'albero mediante disgiunzione logica.
Ogni percorso che dal nodo radice del DT porta verso una foglia corrispondente ad una uscita positiva può essere infatti descritto mediante la congiunzione logica degli attributi che compongono tale percorso,

$$A_i = X_1 \wedge X_2 \wedge \ldots \wedge X_k \tag{3.3}$$

e l'intera funzione è pari alla disgiunzione logica di tali percorsi,

$$F = A_1 \vee A_2 \vee \ldots \vee A_j \tag{3.4}$$

Se il problema di inferenza induttiva può intendersi come un problema di ricerca di una curva interpolante un certo numero di punti che rappresentano gli esempi, è lecito chiedersi se è possibile sostituire al formalismo logico espresso dalle relazioni 3.3 e 3.4 un formalismo matematico, per generare un insieme di punti interpolabili con uno dei metodi sviluppati nel Capitolo 1.

Definizione della Funzione

Per provare l'effettiva fattibilità dell'approccio matematico si è considerata una semplice funzione d'agente per l'attraversamento di un incrocio regolato da un semaforo: questa funzione presenta nove diversi attributi, due dei quali sono ternari, mentre gli altri sono binari. Diamo ad ogni attributo un valore intero i, $i = 1, \ldots, n$, e se l'i-esimo attributo presenta k valori assumibili, ognunno di questi valori verrà indicato con il numero razionale $i.j$, $j = 0, \ldots, k$; tale codifica è riportata nella tabella 3.1.

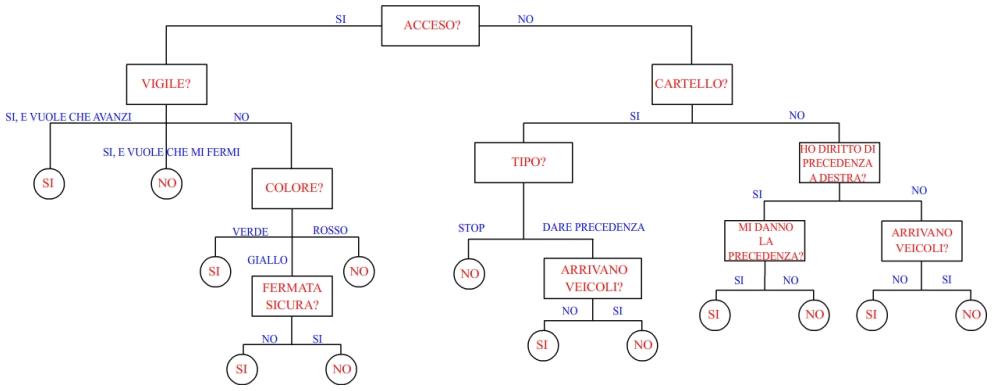

Figura 3.1: DT relativo alla funzione esempio

Attributo	No.	Valori dell'Attributo	No.
Acceso?	1	SI	1.1
		NO	1.2
Vigile?	2	SI, e vuole che avanzi	2.1
		SI, e vuole che mi fermi	2.1
		NO	2.3
Colore?	3	Verde	3.1
		Giallo	3.2
		Rosso	3.3
Fermata Sicura?	4	NO	4.1
		SI	4.2
Cartello?	5	SI	5.1
		NO	5.2
Tipo?	6	Stop	6.1
		Dare Precedenza	6.2
Arrivano Veicoli?	7	NO	7.1
		SI	7.2
Ho Diritto di Precedenza a Destra?	8	SI	8.1
		NO	8.2
Mi Danno Precedenza?	9	SI	9.1
		NO	9.2

Tabella 3.1: Codifica degli Attributi e dei loro Valori

Dalla figura 3.1 è stato poi ricavato un insieme di esempi per l'addestramento del DT: la tabella 3.2 riporta su ogni riga i valori assunti da ogni singolo attributo, e in corrispondenza l'uscita della funzione F.

EX	A1	A2	A3	A4	A5	A6	A7	A8	A9	F
1	1.1	2.1	3.1	4.2	5.1	6.1	7.2	8.1	9.1	1
2	1.1	2.2	3.3	4.1	5.1	6.2	7.1	8.1	9.2	0
3	1.1	2.3	3.1	4.2	5.2	6.1	7.1	8.2	9.1	1
4	1.1	2.3	3.2	4.1	5.1	6.1	7.2	8.1	9.2	1
5	1.1	2.3	3.2	4.2	5.2	6.2	7.2	8.2	9.1	0
6	1.1	2.3	3.3	4.1	5.2	6.2	7.1	8.2	9.2	0
7	1.2	2.1	3.2	4.2	5.1	6.1	7.2	8.2	9.1	0
8	1.2	2.2	3.3	4.2	5.1	6.2	7.1	8.1	9.1	1
9	1.2	2.1	3.1	4.1	5.1	6.2	7.2	8.1	9.2	0
10	1.2	2.1	3.3	4.1	5.2	6.1	7.1	8.1	9.1	1
11	1.2	2.3	3.1	4.2	5.2	6.2	7.1	8.1	9.1	0
12	1.2	2.1	3.2	4.1	5.2	6.1	7.1	8.2	9.1	1
13	1.2	2.2	3.3	4.2	5.2	6.1	7.2	8.2	9.2	0
14	1.1	2.1	3.2	4.2	5.2	6.2	7.1	8.1	9.2	1
15	1.1	2.2	3.1	4.2	5.1	6.2	7.2	8.2	9.2	0
16	1.2	2.3	3.3	4.2	5.1	6.2	7.2	8.2	9.1	0

Tabella 3.2: *Training Set* per la funzione esempio

3.4.1 Implementazione MATLAB

La costruzione dell'albero a partire dall'insieme di attributi e dal *training set* è implementata nelle funzioni `TrainDT.m` e `RecursiveDT.m`: la prima, i cui ingressi sono la matrice del *training set* M (la cui struttura è equivalente alla tabella 3.2, ma senza la prima colonna, che individua solamente il numero dell'esempio), una struttura dati A contenente un vettore per ogni attributo, le cui componenti sono corrisponenti ai valori del singolo attributo, e una struttura dati s che contiene i nomi degli attributi (il suo scopo è quello di produrre un'uscita sul prompt immediatamente leggibile), richiama la seconda finchè è verificata una determinata condizione sulla matrice M.
Il listato seguente rappresenta il c̈uoreẗfunzionale della `RecursiveDT.m`: per ogni colonna di M, corrispondente ciascuna ad un attributo della funzione, viene registrato nei vettori `pos_ex` e `neg_ex` il numero di esempi positivi e negativi per ogni valore assunto dall'attributo corrente (la posizione dei contatori in tali vettori rispecchia quella dei valori in A mediante l'utilizzo della *function* `pos_value.m`) , e infine il numero di esempi positivi e negativi per l'intero attributo (variabili `p_tot` e `n_tot`, ottenute richiamando una *function* che sommi le componenti dei vettori `pos_ex` e `neg_ex`, `sum_comp.m`). Questi parametri rappresentano gli ingressi della *function* `InfGain.m`, che implementa la 3.1 richiamando iterativamente la *function* `information.m`, che invece implementa la 3.2.

```
[m, n] = size(M);
G = zeros(1, n-1);

for j = 1:n-1
    pos_ex = zeros(1, length(A(j).x));
    neg_ex = zeros(1, length(A(j).x));
    for i = 1:m
        if M(i, n) == 1
            pos_ex(pos_value(M(i,j),A(j).x)) =
            pos_ex(pos_value(M(i,j),A(j).x)) + 1;
        else if M(i, n) == 0
                neg_ex(pos_value(M(i,j),A(j).x)) =
                neg_ex(pos_value(M(i,j),A(j).x)) + 1;
            end
        end
    end

    p_tot = sum_comp(pos_ex);
    n_tot = sum_comp(neg_ex);
    G(j) = InfGain(p_tot, n_tot, pos_ex, neg_ex);
end
[gain, index] = max(G);
```

Dopo l'ultima iterazione, il vettore `G` contiene i guadagni d'informazione relativi ad ogni singolo attributo: il massimo del vettore rappresenta il guadagno dell'attributo più selettivo, che resterà individuato in A dall'indice `index`.
Nella seconda parte dello funzione viene valutato quale valore dell'attributo scelto deve essere ulteriormente espanso: per semplicità, e per evitare che la chiamata ricorsiva della `RecursiveDT.m` generi un albero per profondità (senza espandere ulteriormente dei nodi, che rimarrebbero quindi senza un valore stabilito), ad ogni passo viene espanso soltanto un valore, quello dotato del maggior grado di incertezza (i.e., quello dotato del minimo scarto tra numero di esempi positivi e numero di esempi negativi).
Una volta individuato tale valore devono essere approntate le tabelle per la prossima iterazione: l'attributo già espanso viene eliminato dalla lista degli attributi espandibili, e mediante poche righe di comando, la matrice del *training set* viene ridotta, eliminando da essa la colonna relativa all'attributo già espanso, e le righe in cui non compaia il valore da espandere ulteriormente. La condizione di stop della `TrainDT.m` consiste quindi nel valutare che l'intera matrice degli esempi non sia stata annullata man mano che gli esempi venivano eliminati.

Lo *script* `Attraversare.m` applica la `TrainDT.m` ai dati relativi alla funzione prova descritta in precedenza: il risultato è riportato di seguito, e corrisponde a quanto visibile dalla figure 3.2.

```
Attributo più Selettivo per il Training Set Fornito:
Tipo?

Valore del Guadagno: 0.1812

Attributo Tipo?, Uscita 6.2:
Uscita della Funzione Pari a 0
Attributo Tipo?, Uscita 6.1:
Elemento Ulteriormente Espandibile

Attributo più Selettivo per il Training Set Fornito:
Vigile?

Valore del Guadagno: 0.3995

Attributo Vigile?, Uscita 2.1:
Uscita della Funzione Pari a 1
Attributo Vigile?, Uscita 2.3:
Uscita della Funzione Pari a 1
Attributo Vigile?, Uscita 2.2:
Elemento Ulteriormente Espandibile

Attributo più Selettivo per il Training Set Fornito:
Acceso?

Valore del Guadagno: 0.0000

Attributo Acceso?, Uscita 1.2:
Uscita della Funzione Pari a 0
Attributo Acceso?, Uscita 1.1:
Elemento Ulteriormente Espandibile
```

Già da una prima osservazione dei risultati si nota che l'albero ottenuto non risulta essere soddisfacente (l'agente intelligente ha una concezione alquanto bizzarra delle regole di precedenza ...), il che è da imputarsi ad un *training set* non sufficientemente accurato ed alla semplificazione introdotta nell'espansione dei valori; una valutazione delle funzioni MATLAB imple-

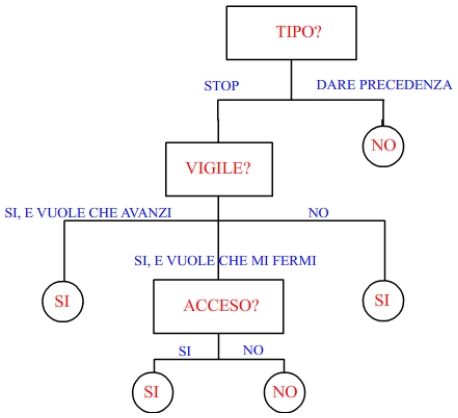

Figura 3.2: DT ottenuto per apprendimento dal training set in tabella 3.2

mentate su un problema a soluzione nota[1] ha infatti mostrato un corretto funzionamento nella valutazione dei guadagni d'informazione, dai quali si è quindi potuto evincere che il problema non è legato alla programmazione.

Resta dunque da verificare se, sfruttando una codifica matematica come quella in tabella 3.1, sia possibile sfruttare un metodo di interpolazione per la verifica dei risultati: per realizzare l'equivalente matematico della 3.3 si è scelto di sommare i valori relativi alle componenti presenti in ogni percorso dell'albero originale verso un nodo foglia, ottenendo l'insieme dei punti riportati in rosso in figura 3.3 (i valori assumibili dalla funzione sono '1', corrispondente a 'SI' e '0' corrispondente a 'NO'); in blu è riportato invece la parte visibile della funzione interpolante costruita sui valori ricavati dall'albero appreso (figura 3.2). Come atteso, il risultato è largamente insoddisfacente.

3.4.2 Conclusioni

La scelta di un'impostazione logica per il problema dell'apprendimento da esempi è largamente giustificata da una scarsa capacità espressiva di metodi puramente matematici: oltre ad una maggior difficoltà implementativa, infatti, lo svantaggio nel passaggio da funzioni logiche a funzioni matematiche è palese sia da un punto di vista della descrizione del problema che da quello della rappresentazione dei risultati. Dietro al fallimento della strategia di valutazione proposta non si trovano soltanto scelte errate sul *training set* o approssimazioni troppo spinte nell'implementazione dei metodi, ma anche una vera e propria impossibilità di ottenere una potenza di rappresentazione sufficiente sostituendo moltiplicazioni o sommme alle rispettive operazioni

[1] Russel, Norvig; *Intelligenza Artificiale: Un Approccio Moderno* (seconda edizione): esempio paragrafo 18.3, pagg. 570-71

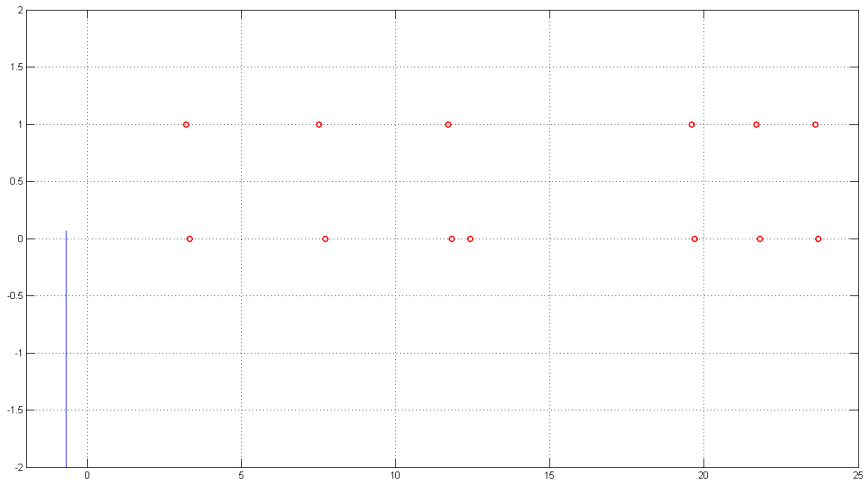

Figura 3.3: Il Risultato dell'Interpolazione

logiche: a riprova di ciò, è sufficiente provare ad interpolare direttamente i punti del grafico 3.3 ottenuti secondo la strategia descritta in precedenza con uno qualsiasi dei metodi di interpolazione polinomiale implementati.
Il risultato, alquanto esplicativo, è una lunga serie di errori e divisioni per zero.

Appendice A

Funzioni MATLAB

A.1 Interpolazione Polinomiale

`cheby.m` Valutazione dei Nodi di Chebychev;

`horner.m` Algoritmo di Horner per la valutazione del Polinomio Interpolante di Newton;

`neville.m` Costruzione iterativa su un elemento noto dello Schema di Neville;

`new_coeff.m` Valutazione dei Coefficienti del Polinomio Interpolante di Newton mediante lo Schema delle Differenze Divise;

`new_val.m` Valutazione in un punto del Polinomio Interpolante di Newton;

`Pol_Interp.m` *Script* per la valutazione delle tecniche di interpolazione implementate e dell'errore di interpolazione;

`polycan.m` Polinomio Interpolante in Forma Canonica;

`polylag.m` Polinomio Interpolante di Lagrange;

`polynev.m` Polinomio Interpolante di Neville;

`polynew.m` Polinomio Interpolante di Newton;

`prdct.m` *Subroutine* sfruttata per la valutazione del Polinomio Interpolante di Newton (alternativa all'algoritmo di Horner);

`prova_polycan.m` *Script* di prova per il Polinomio Interpolante in Forma Canonica;

`prova_polylag.n` *Script* di prova per il Polinomio Interpolante di Lagrange;

`prova_polynev.m` *Script* di prova per il Polinomio Interpolante di Neville;

`prova_polynew.m` *Script* di prova per il Polinomio Interpolante di Newton.

A.2 Interpolazione di Curve Parametriche

`cheby.m` Valutazione dei Nodi di Chebychev;

`horner.m` Algoritmo di Horner per la valutazione del Polinomio Interpolante di Newton;

`neville.m` Costruzione iterativa su un elemento noto dello Schema di Neville;

`new_coeff.m` Valutazione dei Coefficienti del Polinomio Interpolante di Newton mediante lo Schema delle Differenze Divise;

`new_val.m` Valutazione in un punto del Polinomio Interpolante di Newton;

`Par_Interp.m` *Script* per l'interpolazione di diverse curve parametriche $\times$ secondo le tecniche di interpolazione polinomiale sviluppate;

`plot_points.m` *Script* per la stampa dei soli punti di interpolazione delle diverse curve parametriche da interpolare.

`polycan.m` Polinomio Interpolante in Forma Canonica;

`polylag.m` Polinomio Interpolante di Lagrange;

`polynev.m` Polinomio Interpolante di Neville;

`polynew.m` Polinomio Interpolante di Newton;

`prdct.m` *Subroutine* sfruttata per la valutazione del Polinomio Interpolante di Newton (alternativa all'algoritmo di Horner);

A.3 Apprendimento di DT

`Attraversare.m` `Script` per l'applicazione al problema esempio della *function* `TrainDT.m`

`Infgain.m` Funzione di valutazione del Guadagno d'Informazione

`information.m` Funzione per il calcolo del contenuto informativo di una sorgente binaria di simboli;

`plot_attraversa` *Script* che genera il grafico dei punti del DT iniziale e la corrispondente funzione ottenuta per apprendimento da esempi;

`pos_value.m` *Subroutine* sfruttata in `RecursiveDT.m` per la valutazione della posizione di un elemento dato in un vettore;

`raw_not_null.m` *Subroutine* sfruttata in `RecursiveDT.m` per valutare il numero di righe non nulle in una matrice data;

`RecursiveDT.m` *Function* che implementa il problema dell'apprendimento da esempi su una serie di parametri forniti in ingresso;

`sum_comp.m` *Subroutine* sfruttata in `RecursiveDT.m` per sommare le componenti di un vettore dato;

`TrainDT.m` *Function* per la chiamata ricorsiva di `RecursiveDT.m` con condizione di terminazione.

Appendice B

Le Cartelle

B.1 File MATLAB

Tesina - Parte I - Metodi di Interpolazione Polinomiale Funzioni descritte nell'Appendice A.1;

Tesina - Parte II - Interpolazione di Curve Parametriche Funzioni descritte nell'Appendice A.2;

Tesina - Parte III - Apprendimento di DT Funzioni descritte nell'Appendice A.3.

B.2 Immagini

Elenco numerato delle immagini che compaiono nella tesina.

FLASH File FLASH relativi agli schemi dei DT.