

ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA  
SEDE DI CESENA  
FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI  
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

STUDIO DELLE PRESTAZIONI DEL TRAINING  
PARALLELO DI SUPPORT VECTOR MACHINE SU  
CLUSTER IBRIDO 32/64 BIT

**Tesi di Laurea in**

Biofisica delle Reti Neurali e loro applicazioni

**Relatore**

Chiar.mo Prof. Renato Campanini

**Presentata da**

Francesco Turrone

**Co-Relatore**

Dott. Matteo Roffilli

Sessione II

Anno Accademico 2004/2005



*Alla mia famiglia per il loro supporto in  
questi anni, a tutti i ragazzi del Laboratorio  
di Fisica Numerica per i continui stimoli.*



# Ringraziamenti

Desidero ringraziare il Prof. Renato Campanini ed il Dott. Matteo Roffilli per avermi concesso la possibilità di svolgere questo lavoro, per il supporto, la supervisione, la disponibilità e per avermi trasmesso la passione per questo affascinante campo dell'Intelligenza Artificiale chiamato Machine Learning.

Ringrazio il Dott. Luca Benini per il suo contributo tecnico e la pazienza, Rocco Zanni, Manuele Bastianelli ed il suo "64 bit", il Dott. Omar Schiaratura e tutto il Laboratorio di Fisica Numerica.

Ringrazio tutti i ragazzi che hanno contribuito a rendere "migliori" questi tre anni passati a Cesena, in particolare Filippo Cesari e Mauro Schiavone la cui amicizia risale a molti anni fa e con me hanno condiviso molte "tensioni", Massimo Montanari, Simone Pascuzzi, Michele Boccalini e tutti quelli che in qualche modo sono entrati nella mia vita accademica.

Ringrazio i Tabula Rasa, ovvero Donato Giavolucci, Milo Berardi, Mirko Canini, Luigi Pedicini, Mauro Canesi per la fiducia in me e la pazienza dimostrata ogni volta che è stato annullato qualche concerto a causa dei miei esami.

Ringrazio poi vivamente gli amici di sempre, ovvero quelli dai quali non ti potrai mai allontanare, Mattia e l'Irene, Maio, Valerio, Mirco, Francesco, Nicola, Michele, Luca, la Mire, la Raffella la Francesca per aver creduto in me e per i grandi momenti passati insieme.

Vorrei ringraziare infine i miei genitori Giovanna e Roberto e mio fratello Marco. Con questo lavoro spero di poter ripagare almeno in parte tutti i sacrifici che hanno fatto per permettermi di arrivare a questo primo traguardo da me tanto desiderato.



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Introduzione . . . . .	1
1.1.1	Obiettivi . . . . .	2
1.2	Apprendimento Automatico . . . . .	2
1.2.1	Apprendimento Supervisionato . . . . .	3
1.2.2	Apprendimento Non Supervisionato . . . . .	3
1.2.3	Apprendimento Con Rinforzo . . . . .	4
1.3	Classificazione e Regressione . . . . .	4
<b>2</b>	<b>Support Vector Machine</b>	<b>7</b>
2.1	Introduzione . . . . .	7
2.2	Fondamenti teorici . . . . .	7
2.2.1	Dimensione di Vapnik-Chervonenkis . . . . .	7
2.2.2	Minimizzazione del Rischio Strutturale . . . . .	8
2.3	Linear Support Vector Machine . . . . .	9
2.3.1	Il caso separabile . . . . .	9
2.4	I moltiplicatori di Lagrange . . . . .	10
2.4.1	Le condizioni di Karush-Kuhn-Tucker . . . . .	13
2.5	Il caso non linearmente separabile . . . . .	13
2.6	Macchine non lineari . . . . .	15
<b>3</b>	<b>Algoritmi Sequenziali e Strumenti per SVM</b>	<b>19</b>
3.1	Algoritmi . . . . .	19
3.1.1	Chunking . . . . .	19
3.1.2	Osuna's Decomposition Technique . . . . .	20
3.1.3	SMO . . . . .	20

3.2	SVM Tools . . . . .	21
3.2.1	$SVM^{light}$ . . . . .	21
3.2.2	$SVM^{struct}$ . . . . .	25
3.2.3	LIBSVM . . . . .	26
3.2.4	GPDT . . . . .	30
3.3	Confronto tra i tools . . . . .	31
3.3.1	Dataset . . . . .	31
3.3.2	Test Sequenziali . . . . .	32
3.3.3	Proprietà dei Tools per SVM . . . . .	35
<b>4</b>	<b>Sistemi Paralleli</b>	<b>37</b>
4.1	Introduzione . . . . .	37
4.2	Tassonomia dei Sistemi Paralleli . . . . .	38
4.2.1	Multiprocessori . . . . .	39
4.2.2	Multicomputer . . . . .	40
4.2.3	Modelli di comunicazione tra processori . . . . .	41
4.3	Comunicazione tra Multicomputer . . . . .	42
4.3.1	Parallel Virtual Machine . . . . .	43
4.3.2	Message Passing Interface . . . . .	43
4.3.3	Installazione di MPI . . . . .	44
4.4	Symmetric Multi-Processor . . . . .	45
<b>5</b>	<b>Parallel GPDT</b>	<b>47</b>
5.1	Introduzione . . . . .	47
5.2	Tecnica Parallela di Decomposizione . . . . .	47
5.3	Risoluzione Parallela dei QP . . . . .	49
5.4	Aggiornamento Parallelo del Gradiente . . . . .	51
5.5	Selezione del Working Set . . . . .	52
5.6	Implementazione . . . . .	52
<b>6</b>	<b>Risultati Sperimentali</b>	<b>53</b>
6.1	Test Paralleli di Addestramento . . . . .	53
6.2	PGPDT: Scaling su un cluster ibrido 32/64 bit . . . . .	54
6.3	Ottimizzazione: il caching . . . . .	62



<b>Conclusioni e Sviluppi Futuri</b>	<b>65</b>
<b>A Test Environment</b>	<b>67</b>
<b>Bibliografia</b>	<b>73</b>



# Elenco delle figure

1.1	Machine Learning Framework . . . . .	5
2.1	Sottoinsiemi concentrici . . . . .	8
2.2	Iperpiano separatore per il caso linearmente separabile . . . . .	12
2.3	Iperpiano separatore per il caso non linearmente separabile . . . . .	16
2.4	Mapping in uno spazio a dimensione superiore . . . . .	17
3.1	Esempi di pattern contenuti nel dataset Mnist . . . . .	32
6.1	PGPDT Scaling: test sul dataset MNIST . . . . .	59
6.2	Cluster Speedup: test sul dataset MNIST . . . . .	59
6.3	PGPDT Scaling: test sul dataset UCI Adult6 . . . . .	60
6.4	Cluster Speedup: test sul dataset UCI Adult6 . . . . .	60
6.5	PGPDT Scaling: test sul dataset WEB6a . . . . .	61
6.6	Cluster Speedup: test sul dataset WEB6a . . . . .	61
6.7	Prestazioni di PGPDT al variare della dimensione della cache sul dataset MNIST . . . . .	63
A.1	Test Environment . . . . .	68



# Elenco delle tabelle

2.1	Funzioni Kernel . . . . .	17
3.1	Tecniche di Decomposizione . . . . .	21
3.2	Formato dei dati . . . . .	23
3.3	Caso di classificazione . . . . .	25
3.4	GPDT Mnist-Dataset Test . . . . .	33
3.5	GPDT Uciadu6-Dataset Test . . . . .	33
3.6	GPDT Web6a-Dataset Test . . . . .	33
3.7	$SVM^{light}$ Results sul dataset MNIST . . . . .	34
3.8	$SVM^{light}$ Results sul dataset UCI Adult6 . . . . .	34
3.9	$SVM^{light}$ Results sul dataset WEB6a . . . . .	34
3.10	Risultati sequenziali con il tool LIBSVM . . . . .	35
3.11	Proprietá . . . . .	36
3.12	Train e Algoritmi . . . . .	36
3.13	Portabilitá . . . . .	36
3.14	Licenze e Formati d'Input . . . . .	36
4.1	Tassonomia dei sistemi paralleli . . . . .	38
4.2	Compilazione MPI . . . . .	44
6.1	MNIST Parallel Tests . . . . .	55
6.2	UCI Adult6 Parallel Tests . . . . .	56
6.3	WEB6a Parallel Tests . . . . .	57
6.4	Speedup ed efficienza . . . . .	58
6.5	Comportamento di PGPDt sul dataset MEDICAL al variare di q ed n . . . . .	62
6.6	Miglioramenti Prestazionali utilizzando il caching . . . . .	63

# Capitolo 1

## Introduzione

### 1.1 Introduzione

Le tematiche presentate in questa tesi, fanno parte di una disciplina di largo interesse nella ricerca scientifica moderna definita con il termine di Machine Learning. Tale disciplina é un ramo dell'Intelligenza Artificiale che si occupa dell'apprendimento automatico delle macchine. Nonostante la nostra capacità di sviluppare sistemi software complessi, per molti problemi di carattere tipicamente umano come il riconoscimento di pattern o il controllo motorio, nessun sistema artificiale é in grado di trovare una soluzione soddisfacente. Il divario tra intelligenza artificiale e intelligenza naturale é quindi ancora enorme. Nel seguente lavoro viene discusso nel particolare il problema dell'apprendimento per realizzare una classificazione di pattern, utilizzando una potente e recente tecnica di Machine Learning chiamata Support Vector Machine. Tali sistemi necessitano di un processo, l'apprendimento (learning), attraverso il quale un set di esempi chiamato training set viene presentato al sistema ed in base al quale viene generata una funzione che permette di generalizzare su nuovi dati mai visti prima. Spesso la quantità di dati con la quale il sistema si trova ad affrontare é enorme: i tempi di apprendimento utilizzando un'unica macchina fisica diventano improponibili. L'impiego di tecniche di parallelizzazione del lavoro utilizzando più macchine diventa quindi necessario.

### 1.1.1 Obiettivi

Obiettivo del seguente lavoro é un analisi del training parallelo nelle SVM. Verranno analizzati diversi tool efficienti e conosciuti nella Machine Learning Community che permettono il training seriale di SVM. L'attenzione poi si focalizzerá su un tool che ne permette l'addestramento parallelo chiamato PGPDT. Verranno testate le potenzialitá di tale tool (sviluppato da ricercatori italiani) su un cluster ibrido composto da macchine multiprocessori a 32 e 64 bit e utilizzando dei dataset rappresentanti casi reali.

## 1.2 Apprendimento Automatico

L'apprendimento automatico affronta il problema dello sviluppo di sistemi che apprendono, dove per apprendere si intende il migliorare le capacità di esecuzione di un certo compito (task) attraverso l'esperienza acquisita. Le osservazioni passate vengono accumulate formando una base di conoscenza, l'esperienza, che può essere codificata attraverso esempi di comportamenti positivi e negativi - ad esempio un'insieme di immagini digitali di pazienti malati e pazienti sani - attraverso i quali é possibile poi apprendere in modo automatico come valutare nuove informazioni. Dopo una fase di training attraverso un insieme di esempi chiamato training set, un sistema che apprende acquisisce l'abilitá di generalizzare su dati mai visti precedentemente, il test set. La teoria dell'apprendimento sfrutta strumenti matematici che derivano dalla teoria della probabilità e dalla teoria dell'informazione: ciò permette di valutare l'ottimalitá di alcuni metodi rispetto ad altri. Esistono fondamentalmente tre paradigmi di apprendimento [4] [6] che ora verranno brevemente discussi:

1. Apprendimento supervisionato
2. Apprendimento non supervisionato
3. Apprendimento con rinforzo

Si ricorda comunque che non tutti i problemi di apprendimento possono essere associati in maniera univoca a queste categorie.

### 1.2.1 Apprendimento Supervisionato

L'apprendimento supervisionato é caratterizzato dalla presenza di un insegnante che conosce l'etichetta o il costo di ogni pattern del training set. Questa conoscenza dell'ambiente, fornisce la capacità di restituire l'output desiderato ogni volta che viene ricevuto un input. In questo modo é possibile aggiornare i pesi della struttura che apprende, che può essere ad esempio una rete neurale, in base all'output restituito e all'output conosciuto. Questo processo iterativo di modifica dei pesi permette alla struttura di poter emulare l'insegnante generalizzando su pattern mai incontrati. Esistono due importanti tipi di apprendimento supervisionato, cioè l' apprendimento induttivo (Concept Learning, Decision Tree) e l'apprendimento analogico . Nell'apprendimento induttivo esistono diverse istanze positive e negative di un problema, la macchina che apprende deve formulare un concetto che permette di discriminare le due (o più) categorie. Nell'apprendimento analogico, vengono formulate delle analogie in base a singoli esempi, viene definita e applicata una funzione di mapping per nuovi esempi, viene effettuata una validation della soluzione attraverso simulazioni ed infine la nuova conoscenza viene salvata per usi futuri. Un esempio potrebbe essere la creazione automatica di forme plurali di nomi a partire dalla forma singolare. Alcuni esempi di algoritmi d'apprendimento supervisionato, sono l'algoritmo LMS (Least Mean Square) e l'algoritmo di Backpropagation.

### 1.2.2 Apprendimento Non Supervisionato

Nell'apprendimento non supervisionato, non é presente un insegnante che per ogni input conosce il rispettivo output. La struttura che apprende, non riceve alcuna informazione sulla correttezza dell'output fornito oppure sulla funzione target che deve essere approssimata: il learner costruisce concetti attraverso la sperimentazione nell'ambiente. La stessa struttura deve individuare delle regolarità dai pattern di training e codificarle in risposte. Una volta individuate queste regolarità, viene acquisita la capacità di generalizzare su pattern mai visti. Una tecnica di apprendimento non supervisionato é il Clustering.



### 1.2.3 Apprendimento Con Rinforzo

Nell'apprendimento con rinforzo, come nell'apprendimento supervisionato, la struttura adatta i propri parametri in base al feedback ricevuto dall'ambiente, che anziché essere di tipo istruttivo, è di tipo valutativo. Ad esempio, un gioco che utilizza il risultato delle mosse precedenti per migliorare le proprie prestazioni è un sistema che apprende con rinforzo. L'apprendimento con rinforzo può essere di tipo associativo e non associativo, immediato e con ritardo, diretto e indiretto. Alcuni algoritmi per l'apprendimento con rinforzo molto utilizzati sono il Temporal Difference ( $TD(\lambda)$ ) e Q-Learning.

## 1.3 Classificazione e Regressione

Il termine classificazione, può avere in generale due significati. Questo può essere visto come la ricerca di classi o raggruppamenti all'interno di un insieme di dati, oppure può essere visto come l'assegnamento di un pattern ad una determinata classe, dove le classi in questo caso sono note a priori. Solitamente ci si riferisce al primo caso come classificazione non supervisionata, mentre al secondo come classificazione supervisionata. Un'importante questione, riguarda la natura delle classi. Esistono tre definizioni di classe, ovvero:

- Le classi corrispondono alle etichette assegnate a diverse popolazioni. Ad esempio cani e gatti appartengono a due classi o popolazioni distinte. L'appartenenza alla popolazione dei cani o dei gatti (o a nessuno dei due) viene stabilita da un'autorità esterna ed indipendente, il supervisore.
- Le classi sono il risultato di un problema di predizione. Dati un'insieme di attributi, la classe è l'output che viene predetto. In termini statistici, può essere definita come una variabile aleatoria.
- Le classi possono essere estrapolate partizionando un insieme di dati, ad esempio in base a certi attributi. La classe può essere definita come funzione di questi attributi. Ad esempio un oggetto elettronico può essere difettoso se gli attributi superano certi limiti, altrimenti non è difettoso.

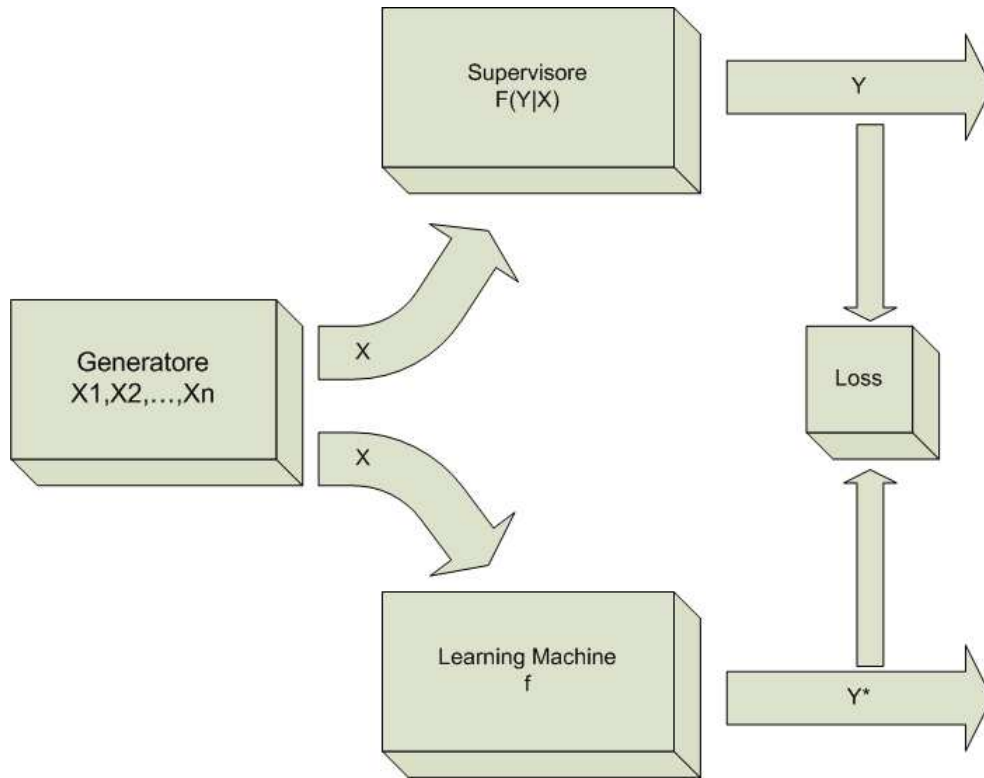


Figura 1.1: Machine Learning Framework

Dal punto di vista della Statistical Learning Theory [21] la classificazione può essere vista nel seguente modo: sia definita la funzione  $L(y, f(x, \alpha))$  come funzione loss, ovvero di discrepanza tra la risposta del supervisore  $y$  e la risposta della macchina. Considero il valore atteso per la loss chiamato Rischio Funzionale definito come:

$$R(\alpha) = \int L(y, f(x, \alpha)) dF(x, y) \quad (1.1)$$

dove  $F(x, y)$  indica la distribuzione di probabilità sconosciuta. Sia quindi  $y$  l'output del supervisore tale che possa assumere i valori  $\{0, 1\}$  e sia  $f(x, \alpha)$ ,  $\alpha \in \Lambda$  un insieme di funzioni indicatrici, ovvero che possono assumere solo due valori: 0 e 1. Considero la seguente funzione loss:

$$L(y, f(x, \alpha)) = \begin{cases} 0 & \text{se } y = f(x, \alpha), \\ 1 & \text{se } y \neq f(x, \alpha), \end{cases} \quad (1.2)$$

Per questa funzione loss, il rischio funzionale determina la probabilità di risposta sbagliata. Il caso di risposte differenti tra il supervisore e la macchina viene chiamato errore di classificazione. Si vuole quindi trovare quella funzione che minimizza

la probabilità di errore dato il training set ma con la distribuzione  $F$  sconosciuta. Oltre a questo tipo di problema, nell'ambito dell'apprendimento esistono altre situazioni dove l'output corrisponde a valori di variabili continue. Questi problemi vengono detti di regressione, dove la funzione da approssimare è proprio la funzione di regressione.

# Capitolo 2

## Support Vector Machine

### 2.1 Introduzione

Le Support Vector Machine (SVM) sono una nuova e potente tecnica di classificazione sviluppata da Vapnik [21] nell'ambito della Statistical Learning Theory. Questo recente algoritmo di apprendimento può essere visto come una tecnica di addestramento alternativa per classificatori Polinomiali, RBF, Percettrone Multistrato. Le SVM nascono per ottimizzare la capacità di generalizzazione della struttura risultante tenendo sotto controllo il fenomeno dell'overfitting e sono basate sull'idea della separazione delle classi con una superficie che massimizza la distanza tra loro.

### 2.2 Fondamenti teorici

Vengono presentati ora i punti cardine della Statistical Learning Theory.

#### 2.2.1 Dimensione di Vapnik-Chervonenkis

Sia  $\alpha$  un insieme generico di parametri, la dimensione VC [21] è una proprietà di un insieme di funzioni  $\{f(\alpha)\}$  che può essere definita per diverse classi di funzioni  $f$ . Considerando il caso di un insieme di funzioni indicatrici, cioè tali per cui  $f(\vec{z}, \alpha) \in \{-1, +1\}$ , la dimensione VC è il massimo numero  $h$  di vettori  $z_1, \dots, z_h$  che possono essere separati in due classi in tutti i  $2^h$  possibili modi usando le funzioni  $f$  dell'insieme considerato: diremo che l'insieme di pattern viene suddiviso ("shattered") dall'insieme di funzioni.

### 2.2.2 Minimizzazione del Rischio Strutturale

La formulazione dell'SVM, si basa sul principio di Minimizzazione del Rischio Strutturale (Structural Risk Minimization - SRM) che si é dimostrato superiore rispetto al classico principio di Minimizzazione del Rischio Empirico (Empirical Risk Minimization - ERM) applicato nelle reti neurali. Per implementare il principio SRM é necessario considerare una struttura che divida lo spazio delle ipotesi in sottoinsiemi concentrici

$$H_1 \subset H_2 \subset \dots \subset H_n \subset \dots$$

con la proprietá che  $h(n) \leq h(n+1)$  dove  $h(n)$  é la dimensione VC dell'insieme  $H_n$ .

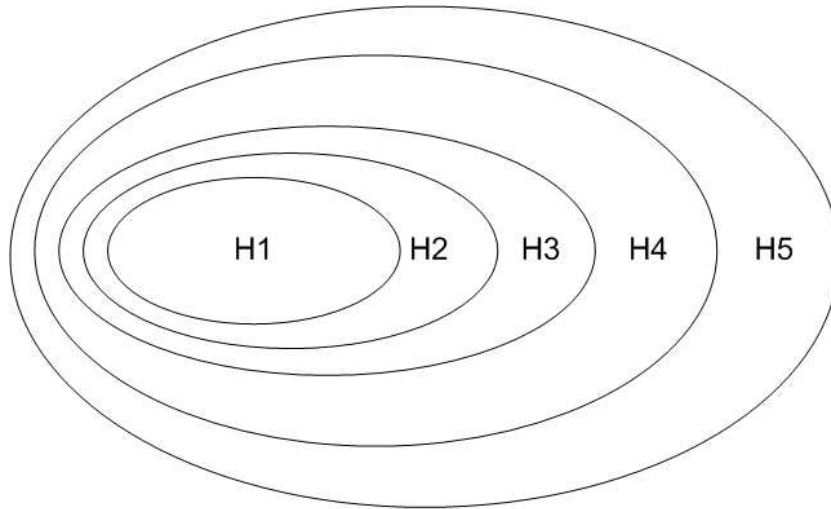


Figura 2.1: Sottoinsiemi concentrici

Nella ricerca della migliore  $f$ , l'SRM suggerisce di risolvere il seguente problema:

$$\min_{H_n} \left( R_{emp}[\lambda] + \sqrt{\frac{h(n)}{l}} \right) \quad (2.1)$$

dove  $l$  indica il numero di pattern del training set mentre  $\lambda$  consiste in un insieme di parametri. Questo principio puó essere difficile da implementare in pratica per due ragioni:

1. La dimensione VC di  $H_n$  potrebbe essere difficile da calcolare

2. Anche assumendo che la dimensione VC sia calcolata per  $H_n$ , non é semplice risolvere il problema di minimizzazione (2.1). In molti casi si minimizzerá il rischio empirico per ogni  $H_n$ , e quindi si sceglierá quello che minimizza la (2.1).

Il controllo della dimensione VC di una tecnica di apprendimento durante l'addestramento non é semplice. L'SVM realizza questo obiettivo, minimizzando un limite sulla dimensione VC e sul numero di errori di training allo stesso tempo.

## 2.3 Linear Support Vector Machine

### 2.3.1 Il caso separabile

Questo caso risulta essere il piú semplice [4]. Si consideri un training set definito come  $\{\vec{x}_i, y_i\}$ ,  $i = 1, \dots, l$ ,  $\vec{x}_i \in \mathbb{R}^d$ ,  $y_i \in \{-1, +1\}$  e si supponga che i pattern siano separabili da un iperpiano (detto iperpiano separatore) che permetta la distinzione degli esempi positivi da quelli negativi. I punti che appartengono all'iperpiano soddisfano l'equazione  $\vec{w} \cdot \vec{x} + b = 0$ , dove  $\vec{w}$  é un vettore perpendicolare all'iperpiano,  $\frac{b}{\|\vec{w}\|}$  é la distanza dell'iperpiano dall'origine e  $\|\vec{w}\|$  indica la norma euclidea di  $\vec{w}$ . Si indichi con  $d_+$  la distanza dell'iperpiano separatore dal piú vicino esempio positivo e con  $d_-$  la distanza dal piú vicino esempio negativo. Il margine é dato da  $d_+$  piú  $d_-$ .

Si supponga che i pattern del training set soddisfino i vincoli

$$\vec{w} \cdot \vec{x}_i + b \geq +1 \quad y_i = +1 \quad (2.2)$$

$$\vec{w} \cdot \vec{x}_i + b \leq -1 \quad y_i = -1 \quad (2.3)$$

che in forma piú compatta possono essere espressi con la disuguaglianza

$$y_i [(\vec{w} \cdot \vec{x}_i) + b] \geq +1, \quad i = 1, \dots, l \quad (2.4)$$

L'iperpiano ottimo é quello che separa i vettori del training set nelle due differenti classi  $y \in \{+1, -1\}$  con la piú piccola norma di coefficienti, cioé margine massimo. Il problema puó essere risolto risolvendo il seguente problema di programmazione quadratica:

$$\min \quad \frac{1}{2} \|\vec{w}\|^2$$

soggetto a

$$y_i [(\vec{w} \vec{x}_i) + b] \geq +1, \quad i = 1, \dots, l$$

Il problema può essere riformulato utilizzando i moltiplicatori di Lagrange, quindi:

- I vincoli nella (2.4) sono sostituiti da vincoli per i moltiplicatori di Lagrange, più facili da soddisfare.
- I dati di training compaiono sotto forma di prodotto scalare tra vettori, questo sarà cruciale per la generalizzazione nel caso non lineare.

## 2.4 I moltiplicatori di Lagrange

Data una funzione  $F$  da ottimizzare ed un insieme di condizioni  $(f_1, f_2, \dots, f_n)$ , un lagrangiano é una funzione  $L(F, f_1, f_2, \dots, f_n, \alpha_1, \alpha_2, \dots, \alpha_n)$  che incorpora le condizioni nel problema di ottimizzazione. Gli  $\alpha$  vengono chiamati moltiplicatori di Lagrange e ne esiste uno per ogni condizione. Vengono quindi introdotti  $l$  lagrangiani positivi, uno per ogni vincolo del tipo (2.4). Per i vincoli con forma  $c_i \geq 0$ , per ottenere il lagrangiano é necessario moltiplicare le equazioni dei vincoli per gli  $\alpha_i$  e sottrarle alla funzione obiettivo. Indico con  $\alpha_i \geq 0$  i moltiplicatori lagrangiani, quindi il problema lagrangiano primale diventa:

$$\min L_p = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^l \alpha_i y_i [(\vec{w} \vec{x}_i) + b] + \sum_{i=1}^l \alpha_i \quad (2.5)$$

Ora si deve minimizzare  $L_p$  rispetto a  $\vec{w}$  e a  $b$ , richiedendo che le derivate rispetto agli  $\alpha_i$  si annullino, con  $\alpha_i \geq 0 \quad \forall i$ .

Il problema é di tipo quadratico convesso in quanto la funzione obiettivo é convessa e i punti che soddisfano i vincoli formano un insieme convesso. É quindi possibile risolvere in maniera equivalente il suo duale  $L_d$ : si richiede che  $L_p$  sia massimizzata ed il gradiente di  $L_p$  rispetto a  $\vec{w}$  e a  $b$  si annulli con  $\alpha_i \geq 0 \quad \forall i$ .

$$\frac{\partial L_d(\vec{w}, b, \alpha)}{\partial b} = 0 \quad (2.6)$$

$$\frac{\partial L_d(\vec{w}, b, \alpha)}{\partial \vec{w}} = 0 \quad (2.7)$$

Richiedere che il gradiente rispetto a  $\vec{w}$  (2.6) e a  $b$  (2.7) si annulli, comporta:

$$\sum_{i=1}^l \alpha_i y_i = 0 \quad \forall i \quad \alpha_i \geq 0 \quad (2.8)$$

$$\sum_{i=1}^l \alpha_i \vec{x}_i y_i = \vec{w} \quad \forall i \quad \alpha_i \geq 0 \quad (2.9)$$

Questo deriva dalle condizioni di Karush-Kuhn-Tucker che verranno esaminate nel paragrafo successivo. L'iperpiano ottimo é quindi una combinazione lineare dei vettori del training set. Sostituendo le equazioni (2.8) e (2.9) nel problema primale  $L_p$ , otteniamo la seguente formulazione duale:

$$L_d = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \vec{x}_i \vec{x}_j \quad (2.10)$$

Si può notare che esiste un moltiplicatore di Lagrange per ogni pattern del training set, ma solamente quei punti per cui  $\alpha_i \geq 0$  corrispondono ai vettori che contribuiscono alla soluzione  $\vec{w}$ . Nella teoria delle SVM questi sono chiamati vettori di supporto. Essi sono i punti più vicini al confine di decisione. Tutti gli altri punti hanno  $\alpha_i = 0$ .

Il problema finale diventa:

$$\max \quad L_d = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \vec{x}_i \vec{x}_j$$

soggetto a

$$\sum_{i=1}^l \alpha_i y_i = 0 \quad \alpha_i \geq 0$$

Dato un pattern di test  $x$ , la funzione decisionale che l'assegna ad una o l'altra classe sarà:

$$f(x) = \text{sign} \left( \sum_{\text{support vectors}} \alpha_i y_i (\vec{x}_i \vec{x}) + b \right) \quad (2.11)$$

dove gli  $x_i$  sono i vettori di supporto, gli  $\alpha_i$  sono i moltiplicatori lagrangiani e  $b$  indica la threshold.

Infine la risoluzione del lagrangiano duale soggetto ai vincoli  $\sum_{i=1}^l \alpha_i y_i = 0$  e  $\alpha_i \geq 0$  é utilizzato per determinare i moltiplicatori. La (2.9) determina l'iperpiano ottimo. Il classificatore é dato dalla (2.11).



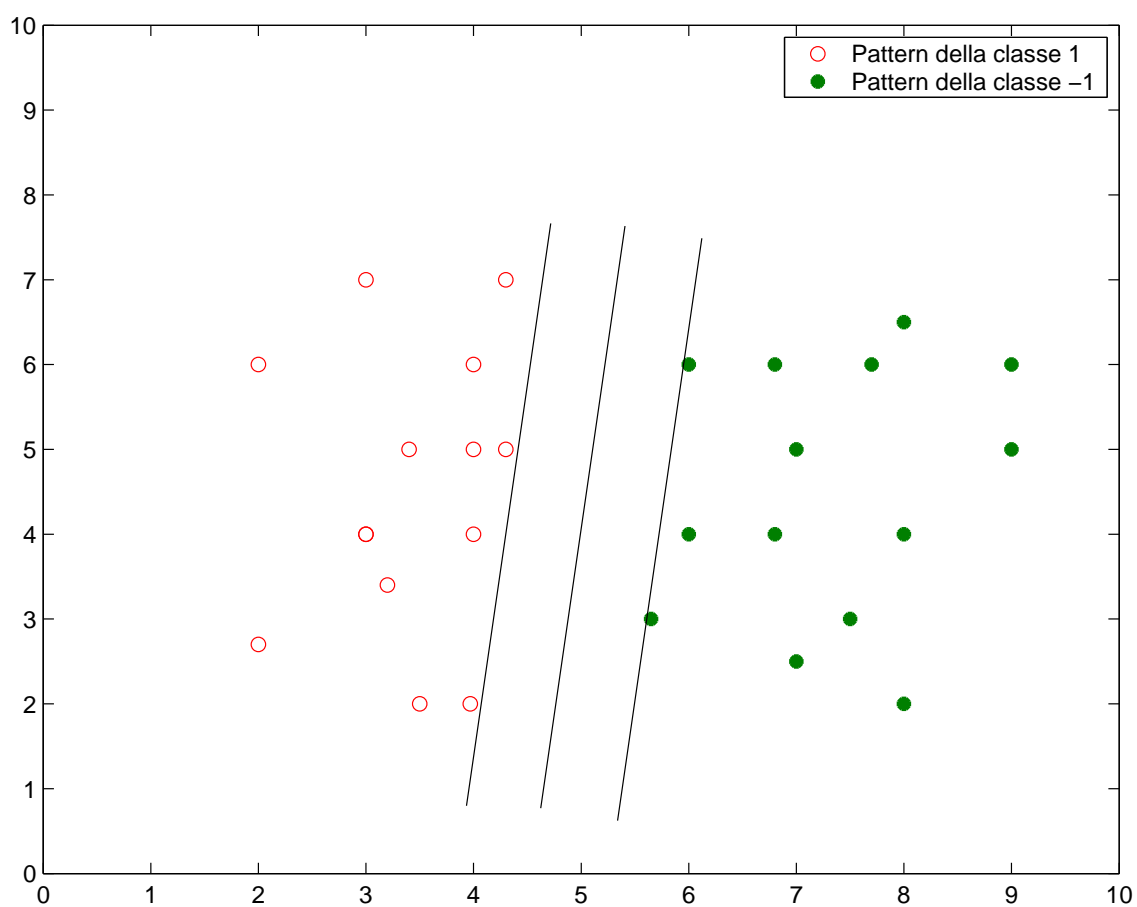


Figura 2.2: Iperpiano separatore per il caso linearmente separabile

### 2.4.1 Le condizioni di Karush-Kuhn-Tucker

Le condizioni di Karush-Kuhn-Tucker giocano un ruolo fondamentale per tutti i problemi di ottimizzazione con vincoli. Per il problema primale visto precedentemente, le condizioni KKT possono essere formulate nel seguente modo:

$$\begin{aligned}\frac{\partial L(w_v, b, \alpha)}{\partial b} &= - \sum_i \alpha_i y_i = 0 \\ \frac{\partial L(w_v, b, \alpha)}{\partial w_v} &= w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \dots, d \\ y_i [(\vec{w} \vec{x}_i) + b] - 1 &\geq 0, \quad i = 1, \dots, l \\ \alpha_i &\geq 0 \quad \forall i \\ \alpha_i [y_i (\vec{w} \vec{x}_i + b) - 1] &= 0 \quad \forall i\end{aligned}$$

Le condizioni di Karush-Kuhn-Tucker sono soddisfatte dalla soluzione di un qualunque problema di ottimizzazione con vincoli arbitrari, ammesso che l'intersezione delle direzioni ammesse con le direzioni di discesa coincida con l'intersezione delle direzioni dei vincoli linearizzati con le direzioni di discesa. Questa condizione vale sempre per le SVM in quanto i vincoli sono sempre lineari. Il problema per le SVM é di tipo convesso, e per i problemi convessi le condizioni di KKT sono necessarie e sufficienti affinché  $\vec{w}$ ,  $b$  e  $\alpha$  siano una soluzione. Il training di una macchina a supporto vettoriale corrisponde quindi a determinare una soluzione per le condizioni di KKT.

## 2.5 Il caso non linearmente separabile

Quando i pattern non sono linearmente separabili, l'algoritmo precedente non può determinare una soluzione ammissibile. Per risolvere il problema, é necessario rilassare i vincoli

$$\begin{aligned}\vec{w} \vec{x}_i + b &\geq +1 & y_i &= +1 \\ \vec{w} \vec{x}_i + b &\leq -1 & y_i &= -1\end{aligned}$$

I vincoli rilassati, si ottengono introducendo delle variabili di slack positive  $\xi_i, i = 1, \dots, l$ :

$$\vec{w} \vec{x}_i + b \geq +1 - \xi \quad y_i = +1 \quad (2.12)$$

$$\vec{w} \vec{x}_i + b \leq -1 + \xi \quad y_i = -1 \quad (2.13)$$

quindi  $F(\xi) = \sum_i \xi_i$  diventa una limite superiore sul numero di errori di training. Perché vi sia un errore, la variabile di slack corrispondente deve superare l'unità. In un formato più compatto i vincoli si possono anche esprimere come:

$$y_i [(\vec{w} \vec{x}_i) + b] - 1 + \xi_i \geq 0, \quad \xi_i \geq 0 \quad i = 1, \dots, l \quad (2.14)$$

Assegnando un costo agli errori  $\xi$ , si può cambiare la funzione da minimizzare da

$$\frac{1}{2} \|\vec{w}\|^2$$

in

$$\frac{1}{2} \|\vec{w}\|^2 + C \left( \sum_i \xi_i^k \right) \quad (2.15)$$

dove C é un parametro scelto dall'utente: ad un valore elevato corrisponde un'alta penalità assegnata agli errori. Si tratta di un problema di programmazione convessa per ogni intero positivo k. Il lagrangiano primale ora diventa:

$$L_p = \frac{1}{2} \|\vec{w}\|^2 + C \left( \sum_{i=1}^l \xi_i^k \right) - \sum_{i=1}^l \alpha_i [y_i (\vec{w} \vec{x}_i + b) - 1 + \xi_i] - \sum_i \mu_i \xi_i \quad (2.16)$$

dove i  $\mu_i$  sono i moltiplicatori introdotti per imporre che  $\xi_i \geq 0 \quad \forall i$ .

In seguito alle condizioni di KKT otteniamo per il caso non linearmente separabile la seguente formulazione duale del problema:

$$\max \quad L_d = \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j y_i y_j \vec{x}_i \vec{x}_j$$

soggetto a

$$\sum_{i=1}^l \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C$$

L'iperpiano ottimo é dato da

$$\vec{w} = \sum_{support\,vectors} \alpha_i y_i \vec{x}_i$$

La differenza rispetto al caso linearmente separabile consiste nel fatto che i moltiplicatori lagrangiani  $\alpha_i$  sono limitati superiormente da C.

Il classificatore é dato da

$$f(x) = sign \left( \sum_{support\,vectors} \alpha_i y_i (\vec{x}_i \vec{x}) + b \right)$$

## 2.6 Macchine non lineari

Per molti problemi, le superfici decisionali lineari trattate fino ad ora, non sono appropriate. In alcuni casi, alcuni problemi che non sono separabili nello spazio di input  $\mathfrak{R}_d$  possono diventarlo in un spazio  $H$  a maggiore dimensionalit . Si pu  notare che l'unico modo in cui appaiono i dati nel problema d'addestramento   quello di prodotto scalare  $\vec{x}_i \vec{x}_j$ . Si supponga di effettuare un mapping dei dati in un spazio  $H$  a pi  elevata dimensionalit , denominato spazio di Hilbert, attraverso la funzione  $\phi$ :

$$\phi : \mathfrak{R}_d \rightarrow H$$

In questo modo, l'algoritmo di apprendimento dipende dai dati solo attraverso i prodotti scalari  $\phi(\vec{x}_i) \phi(\vec{x}_j)$ . Attraverso l'introduzione di una funzione Kernel  $K$  tale che

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \phi(\vec{x}_j) \quad (2.17)$$

nell'algoritmo di training sar  sufficiente utilizzare  $K$  senza rendere esplicita la funzione  $\phi$ . Sostituendo nell'algoritmo di apprendimento tutte le occorrenze di  $\vec{x}_i \vec{x}$  con  $K(\vec{x}_i, \vec{x})$  si ottiene il nuovo classificatore

$$f(x) = sign \left( \sum_{support\,vectors} \alpha_i y_i K(\vec{x}_i, \vec{x}) + b \right)$$

Per individuare per quali Kernel esiste una coppia  $\{H, \phi\}$  con le propriet  descritte precedentemente,   possibile utilizzare il teorema di Mercer: esiste una funzione  $\phi$

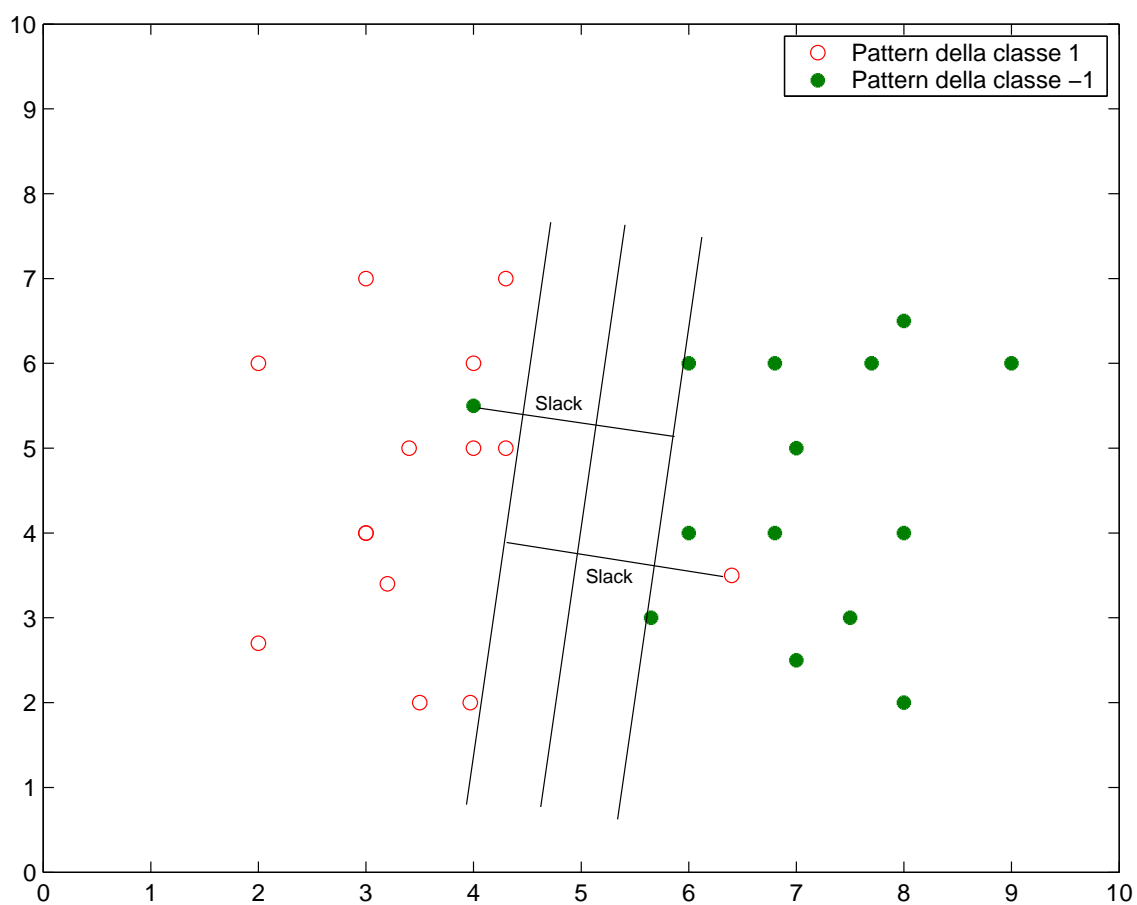


Figura 2.3: Iperpiano separatore per il caso non linearmente separabile

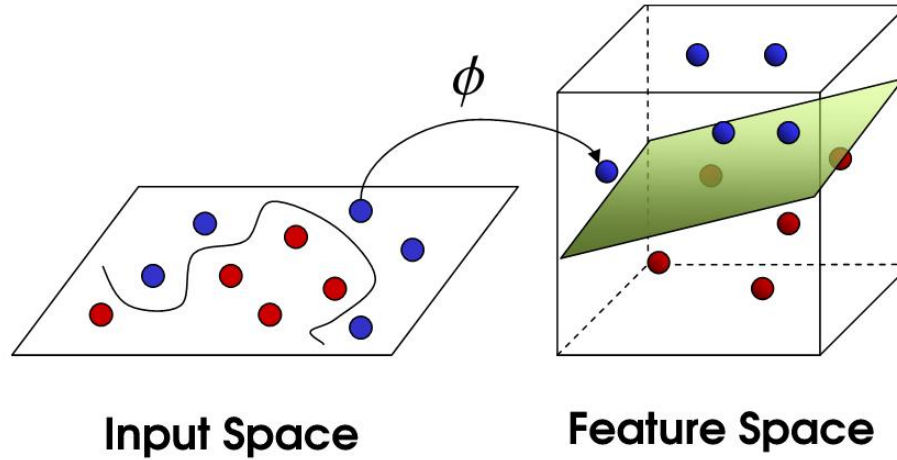


Figura 2.4: Mapping in uno spazio a dimensione superiore

e un'espansione  $K(u, v) = \sum_{k=1}^{\infty} a_k \phi_k(u) \phi_k(v)$  con  $a_k$  coefficienti positivi se e solo se, per ogni funzione  $g \neq 0$  tale che:

$$\int g^2(u) du \leq +\infty \quad (2.18)$$

si ha che

$$\int K(u, v) g(u) g(v) dudv \geq 0 \quad (2.19)$$

Questa é sempre soddisfatta per potenze positive del prodotto scalare

$K(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^P$ . Alcuni kernel comunemente usati sono mostrati in tabella 2.1 .

Classificatore	Funzione Kernel
Lineare	$K(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}$
Polinomiale di grado d	$K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + 1)^d$
Gaussiano	$K(\vec{x}, \vec{y}) = \exp(-1/2\sigma^2 \ \vec{x} - \vec{y}\ ^2)$
Percettrone Multistrato	$K(\vec{x}, \vec{y}) = \tanh(\vec{x} \cdot \vec{y} - \theta)$

Tabella 2.1: Funzioni Kernel

Nei test successivi, verrà utilizzato il Kernel Gaussiano, il quale fa parte delle funzioni kernel chiamate RBF (Radial Basis Function) ed é computazionalmente comodo da calcolare.

Il Kernel Gaussiano é definito nel seguente modo:

$$\begin{aligned} K(\vec{x}, \vec{y}) &= \exp \left[ -\frac{1}{2\sigma^2} \|\vec{x} - \vec{y}\|^2 \right] \\ &= \exp \left[ -\frac{1}{2\sigma^2} (\vec{x} \cdot \vec{x} + \vec{y} \cdot \vec{y} - 2\vec{x} \cdot \vec{y}) \right] \\ &= \exp \left[ -\frac{1}{2\sigma^2} (\|\vec{x}\|^2 + \|\vec{y}\|^2 - 2\vec{x} \cdot \vec{y}) \right] \end{aligned} \tag{2.20}$$

## Capitolo 3

# Algoritmi Sequenziali e Strumenti per SVM

Dal punto di vista pratico, l'addestramento di una SVM consiste nella risoluzione di un problema di programmazione quadratica soggetta a vincoli lineari, in un numero di variabili determinate dal numero di pattern in input. Una generica tecnica di ottimizzazione quadratica é però insoddisfacente a causa dell'elevato spazio di memorizzazione e a causa del tempo richiesto. Verranno esaminati in questo capitolo alcuni algoritmi per il Problema Quadratico delle SVM che sfruttano il paradigma della decomposizione in sottoproblemi. Questi algoritmi di tipo sequenziale sono il chunking, la tecnica di decomposizione di Osuna e il Sequential Minimal Optimization. Vengono poi presentati gli strumenti software piú noti per operare con le SVM. Nei capitoli successivi si esaminerá l'addestramento parallelo.

### 3.1 Algoritmi

#### 3.1.1 Chunking

Il problema quadratico risultante nella formulazione SVM, lavora su una matrice che ha un numero di elementi pari al quadrato degli esempi del training set. Data la limitatezza della memoria dei calcolatori, spesso non é possibile memorizzare tale struttura. Il chunking é un algoritmo utilizzato per risolvere il QP (Quadratic Problem) nelle SVM e sfrutta il fatto che non vi sono cambiamenti nel risultato del QP se si rimuovono le righe e le colonne della matrice che corrispondono ai moltiplicatori



con valore zero. Il problema può quindi essere suddiviso in una serie di sottoproblemi più piccoli allo scopo di identificare i lagrangiani diversi da zero ed eliminare quelli nulli. Questo algoritmo permette di ridurre la dimensione della matrice dal quadrato degli esempi di training al quadrato dei moltiplicatori lagrangiani diversi da zero. Tuttavia, l'algoritmo di chunking non è in grado di gestire problemi molto grandi in quanto anche dopo la riduzione non potrebbe essere ancora in grado di memorizzare la matrice. Un modo per scavalcare il problema è quello dell'utilizzare particolari strutture che evitano la memorizzazione dell'intera matrice contenente tutti i prodotti scalari (Hessiana).

### 3.1.2 Osuna's Decomposition Technique

Questa tecnica di risoluzione, è basata sulla decomposizione del QP in una serie di sottoproblemi più piccoli. Dato che viene mantenuta costante la dimensione della matrice per ogni sottoproblema, questo implica l'aggiunta e la cancellazione dello stesso numero di esempi ad ogni passo. Usando una matrice di dimensione costante è possibile l'addestramento di data set con dimensione arbitraria. Questa tecnica di decomposizione effettua lo splitting del problema in una parte inattiva e una parte attiva, il working set. Un potenziale svantaggio di questo algoritmo è dato dal fatto che l'addestramento può richiedere molto tempo. Per risolvere questo problema viene spesso utilizzato:

1. Un metodo efficiente per selezionare il working set
2. Un restringimento euristico (shrinking) del problema da ottimizzare
3. Miglioramenti computazionali come caching e aggiornamenti incrementali del gradiente e del criterio di terminazione

### 3.1.3 SMO

Sequential Minimal Optimization (SMO) è un algoritmo che risolve velocemente il QP delle SVM decomponendolo in sottoproblemi [14]. SMO sceglie ad ogni passo di risolvere il problema di ottimizzazione più piccolo, ovvero considerando due soli moltiplicatori di Lagrange, perché devono obbedire ad un vincolo lineare di uguaglianza: viene trovato il valore ottimo per questi moltiplicatori e viene aggiornata

la soluzione. Un vantaggio sostanziale di questo metodo é dato dal fatto che la risoluzione per due moltiplicatori può essere fatta analiticamente. SMO non richiede un'ulteriore matrice di memorizzazione, quindi possono essere gestiti problemi anche molto grossi. L'algoritmo é formato da tre componenti:

- Un metodo analitico per calcolare i due moltiplicatori
- Una euristica per scegliere quali moltiplicatori ottimizzare
- Un metodo per calcolare la soglia  $b$

	Modulo QP	Dimensione Iterazione
Chunking	SI	Arbitraria
Osuna's Technique	SI	Arbitraria
SMO	NO	2

Tabella 3.1: Tecniche di Decomposizione

## 3.2 SVM Tools

### 3.2.1 $SVM^{light}$

#### Caratteristiche generali

$SVM^{light}$  [9] é un'implementazione efficiente sviluppata in linguaggio C delle Support Vector Machine di Vapnik, per il problema della classificazione, della regressione e dell'apprendimento di una funzione di ranking . Il tool é rilasciato freeware alla pagina web [http : //svmlight.joachims.org/](http://svmlight.joachims.org/) ed é costituito da due moduli: `svm_learn` e `svm_classify`. Il primo modulo é utilizzato per addestrare l'SVM attraverso un training set, mentre il secondo per testare l'SVM attraverso un test set. Dopo la fase di apprendimento, nel caso della classificazione, il tool restituisce la classe corrispondente ad ogni esempio di test. La classificazione avviene per due classi distinte: la seguente versione del tool non permette la multilabel classification. Per quest'ultima classificazione e per la gestione di output strutturati, esiste un'estensione del tool chiamata  $SVM^{struct}$  della quale si parlerá successivamente. Oltre alla classificazione con approccio induttivo, il tool permette una classificazione di tipo

transduttivo. Nell'approccio induttivo (dal particolare al generale) il learner cerca di indurre la funzione decisionale (il generale) cercando di minimizzare l'error rate sulla sola distribuzione dei pattern d'esempio (il particolare). In diverse situazioni non ci si preoccupa della particolare funzione decisionale, ma ci si preoccupa della classificazione di un set di esempi (il test set) con il minimo errore possibile. Questo é l'obiettivo dell'approccio transduttivo (dal particolare al particolare). Nel caso della regressione, l'output  $Y$  appartiene ai numeri reali. Il tool permette anche l'apprendimento di una ranking function. Una ranking function può essere utilizzata ad esempio nell'information retrieval per stilare una classifica delle pagine web che soddisfano una certa query  $Q$ .

### **Tecnica di Decomposizione e Ottimizzazione**

Un approccio trattabile utilizzato per addestrare le SVM in problemi con molti esempi di training, é quello di decomporre il problema in una serie di sottoproblemi. Le  $SVM^{light}$ , utilizzano l'idea di decomposizione di Osuna. Questa decomposizione effettua lo splitting del problema di ottimizzazione in una parte chiamata parte inattiva, ed una parte chiamata parte attiva, quello che viene chiamato working set. Il principale vantaggio di questa decomposizione, consiste nel fatto che comporta algoritmi con richieste di memoria lineari nel numero di esempi di training e lineari nel numero di vettori di supporto. Un potenziale svantaggio consiste nel possibile eccesso di tempo per l'addestramento. L'algoritmo utilizzato dal tool incorpora un metodo efficiente per selezionare il working set, un successivo restringimento (shrinking) del problema di ottimizzazione e miglioramenti computazionali come caching e aggiornamenti incrementali del gradiente e del criterio di terminazione. Per quanto riguarda la scelta del working set, é desiderabile scegliere un insieme di variabili che permettono di effettuare la minimizzazione del problema. L'algoritmo utilizza una strategia basata sul metodo di Zoutendijk. L'idea consiste nel trovare la direzione di discesa più ripida  $d$ , che ha solo  $q$  elementi diversi da zero. Le variabili corrispondenti a questi elementi formeranno il working set corrente. Per molti problemi il numero di vettori di supporto é molto più piccolo del numero di esempi di addestramento. Se fosse conosciuto a priori quale esempio d'addestramento risulta essere vettore di supporto, potrebbe essere sufficiente effettuare l'addestramento con questi esempi e avere lo stesso risultato. Questo potrebbe far risultare il problema di ottimizzazione

piú piccolo e facile da risolvere, risparmiando tempo e spazio. Su questo principio é basato il restringimento (shrinking) del problema di ottimizzazione. Per quanto riguarda i miglioramenti computazionali, é importante ricordare il caching, ovvero la memorizzazione in una cache interna al programma di quegli'elementi che sono utilizzati in un gran numero di iterazioni.

### Formato dei dati

Il tool adotta una ben precisa rappresentazione dei dati utilizzata anche in altri tool (LIBSVM, GPDT). Il formato utilizzato per i campioni di addestramento e di test é visibile in tabella 3.2. Ogni linea rappresenta un campione ed i commenti sono

$\langle linea \rangle = \langle target \rangle \langle \text{feature}_i : valore_i \rangle \dots \langle \text{feature}_n : valore_n \rangle \#info$
$\langle target \rangle = +1   -1   0   \langle float \rangle$
$\langle feature \rangle = \langle intero \rangle   qid$
$\langle valore \rangle = \langle float \rangle$
$\langle info \rangle = \langle stringa \rangle$

Tabella 3.2: Formato dei dati

preceduti dal carattere cancelletto. Ogni campione é costituito dal valore della funzione target per l'esempio considerato e da una serie di coppie  $\langle \text{feature}_i : valore_i \rangle$ . Il valore della funzione target per l'esempio considerato puó assumere i valori +1, -1, 0 o un qualsiasi valore floating point . Le features sono rappresentate da valori sia interi sia di tipo qid, ovvero un valore speciale usato nella modalitá ranking, mentre i loro valori sono di tipo floating point. Il valore target ed ogni coppia feature-valore sono separati da uno spazio bianco. Le coppie feature-valore sono disposte in ordine crescente per valore di feature e le features con valore 0 possono essere saltate. Un campione con valore target pari a 0, indica che l'esempio deve essere classificato utilizzando la transduzione.

### Addestramento e Test

L'apprendimento avviene attraverso il seguente comando:

```
svm_learn [-options] train_file model_file
```

dove il file di training contiene gli esempi di addestramento, mentre `model_file` é il modello che verrà creato. Il tool supporta diverse opzioni che riguardano l'apprendimento. Ricordiamo ora le più importanti mentre per le altre si rimanda alla documentazione ufficiale. L'opzione `-z` permette di scegliere il problema sul quale si vuole lavorare: é possibile scegliere tra classificazione (c), regressione (r) e ranking (p). L'opzione `-c`, permette di determinare il trade-off tra l'errore e il margine. Ad un alto valore di  $C$ , dove  $C > 0$ , corrisponde un'alta penalità dovuta agli errori, mentre un piccolo valore di  $C$ , incrementerà gli errori di training. Attraverso l'opzione `-t` é possibile utilizzare diversi tipi di kernel, cioè:

- Lineare
- Polinomiale
- Radial Basis Function
- Percettrone Multistrato
- Kernel definito dall'utente

Il test avviene tramite il seguente comando:

```
svm_classify [-options] test_file model_file predictions_file
```

Il file d'output delle  $SVM^{light}$  é costituito da una linea per ogni campione di test contenente il valore della funzione decisionale per l'esempio considerato. Per il problema della classificazione, il segno della label predetta determina la classe. Nel problema della regressione, la label é il valore predetto, mentre per il ranking é usata per ordinare gli esempi di test.

## Performance

Per quanto riguarda la valutazione delle performance, e supponendo di riferirsi alla classificazione in tabella 3.3

(B risulta essere un falso negativo mentre C risulta essere un falso positivo), il tool mostra a video:

- l'accuratezza (AC) : indica la proporzione del numero totale di predizioni corrette

$$Accuratezza = (A + D) \div (A + B + C + D)$$

TARGET	PREDIZIONI	
	<i>Positivo</i>	<i>Negativo</i>
<i>Esempio Positivo</i>	A	B
<i>Esempio Negativo</i>	C	D

Tabella 3.3: Caso di classificazione

- la recall : indica la proporzione di casi positivi correttamente identificati

$$Recall = A \div (A + B)$$

- la precisione : indica la proporzione dei casi positivi predetti classificati correttamente

$$Precisione = A \div (A + C)$$

- il runtime : indica il tempo d'esecuzione, senza contare l'IO, in cpu-seconds

### 3.2.2 $SVM^{struct}$

Il tool  $SVM^{struct}$  può essere definito come un'estensione del tool  $SVM^{light}$  descritto precedentemente che sfrutta il Quadratic Optimizer per predire output strutturati. L'algoritmo realizza un apprendimento supervisionato attraverso l'approssimazione di un mapping  $h : X \longrightarrow Y$ , utilizzando esempi di addestramento  $(x_1, y_1), \dots, (x_n, y_n)$  e tale che l'errore di predizione sia minimo. Diversamente dalle tipiche SVM, tuttavia, che considerano solo predizioni ad una variabile in output come nella regressione ( $X \in \mathbb{R}, Y \in \mathbb{R}$ ) o nella classificazione ( $X \in \mathbb{R}, Y \in \{-1, +1\}$ ), le  $SVM^{struct}$  possono predire oggetti complessi come alberi, sequenze, insiemi. Esempi di problemi con output complessi sono:

- Multi-Label Classification
- Natural Language Parsing
- Information Retrieval
- Noun-Phrase Co-reference

### 3.2.3 LIBSVM

#### Caratteristiche generali

LIBSVM [5] consiste in un tool per la classificazione (C-SVC, nu-SVC), regressione (epsilon-SVR, nu-SVR) e stima della distribuzione (SVM ad una classe). Il tool a differenza di  $SVM^{light}$  supporta la classificazione multiclasse ed é rilasciato freeware alla pagina [http : //www.csie.ntu.edu.tw/ ~ cjlin/libsvm/](http://www.csie.ntu.edu.tw/~cjlin/libsvm/) . Un obiettivo importante di questa libreria é quello di permettere agli utenti, non solo informatici, un facile utilizzo delle SVM. Il package é costituito da tre tool a linea di comando: svm-train, svm-predict ed svm-scale. Il primo modulo é utilizzato per addestrare l'SVM attraverso un training set, il secondo per testare l'SVM attraverso un test set, il terzo modulo é utilizzato per effettuare una normalizzazione dei dati nel range definito dall'utente, che pu essere ad esempio  $[1, -1]$  oppure  $[0, 1]$ . Oltre a questi moduli, é presente nel tool un utile strumento scritto in linguaggio python chiamato grid.py, il quale permette di scegliere i parametri ottimali di apprendimento. Il tool é in grado di risolvere efficacemente cinque problemi:

- C-Support Vector Classification
- v-Support Vector Classification
- $\epsilon$ -Support Vector Regression
- v-Support Vector Regression
- Distribution Estimation (one-class SVM)

I passi fondamentali nell'utilizzo del tool sono i seguenti:

1. Convertire i dati in input nel formato del tool che si intende utilizzare.
2. Effettuare la normalizzazione dei dati nel range opportuno ( es.  $[1, -1]$  )
3. Scegliere il tipo di funzione kernel da utilizzare, ad esempio Kernel Gaussiano
4. Utilizzare la cross-validation per determinare i parametri C (soft-margin) e  $\sigma$  (ampiezza della gaussiana)
5. Con i valori di C e  $\sigma$  trovati con la cross-validation, ri-effettuare la fase di training

### Formato dei dati

La rappresentazione adottata dal tool per i campioni di addestramento e di test é la stessa di quella di  $SVM^{light}$ , questo avvantaggia notevolmente le fasi di addestramento, test e confronto. Come già descritto precedentemente, un tipico campione (sia di training sia di test) é costituito da una linea del file contenente il valore della funzione target per l'esempio considerato (es. +1 o -1), seguito da uno spazio e quindi le varie feature separate dal loro valore dal carattere ":" e disposte in ordine crescente per numero di feature.

### Normalizzazione

Supponendo di aver eseguito precedentemente la fase di preprocessing, quindi di avere il data set nel formato adottato dal tool, é possibile passare alla fase di normalizzazione dei dati tramite il comando `svm-scale`. L'obiettivo di questa fase é quello di effettuare dei piccoli aggiustamenti ai dati allo scopo di renderli piú adatti alle valutazioni del kernel. Supponendo ad esempio di prendere in considerazione un Kernel Gaussiano - un particolare tipo di kernel che fa parte di quelle funzioni kernel chiamate RBF (Radial Basis Functions). Se una certa feature di un campione, presenta un maggiore range di variazione rispetto ad un'altra, allora questa dominerá la sommatoria nella gaussiana, mentre feature con piccole variazioni di range saranno essenzialmente ignorate. Da questo se ne puó ricavare che feature con maggiore range di variazione, avranno maggiore attenzione da parte dell'algoritmo SVM. É necessaria quindi una normalizzazione dei valori delle feature in modo tale che questi cadano all'interno dello stesso range, che puó essere ad esempio  $[0,1]$  oppure  $[-1,+1]$ . Questa normalizzazione puó essere eseguita attraverso il comando `svm-scale` che accetta in input un data set e restituisce come output lo stesso data set normalizzato nel range scelto e un insieme di parametri di scalatura. Questi dati riscalatati verranno poi utilizzati dal modulo che realizza l'apprendimento. Il formato del comando di scalatura é il seguente:

$$svm - scale - s \ param\_scalatura \ training\_file > training\_file\_scalato$$

Per specificare il range di scala, vengono utilizzate due opzioni che specificano rispettivamente il lower bound e l'upper bound del range, ovvero "-l" e "-u".



### Selezione ottima dei parametri

Il processo di selezione dei parametri é la fase piú delicata nell'utilizzo di una SVM. Come già accennato precedentemente, il tool é fornito di uno strumento scritto in linguaggio python che permette di scegliere i parametri ottimali per l'apprendimento. In particolare effettua una stima delle performance di generalizzazione per un range di valori di  $C$  e dell'ampiezza  $\sigma$  del kernel gaussiano. I risultati di questa stima, possono essere visualizzati (supponendo di trovarsi in ambiente operativo Linux e supponendo di aver installato il visualizzatore di plot "gnuplot") graficamente in un plot a due dimensioni. L'obiettivo é quello di scegliere i valori ottimali per i parametri  $C$  e  $\sigma$ . Il metodo utilizzato dal tool per effettuare questa scelta, consiste nella stima delle performance di generalizzazione per un range di valori di  $C$  e  $\sigma$ , quindi esaminare i risultati graficamente e scegliere quelli appropriati. Il seguente comando permette di realizzare un plot delle performance di cross validation per il data set già scalato:

```
grid.py -log2 c - 5, 5, 1 -log2 g - 20, 0, 1 -v 10 training_file_scalato
```

Il range di ricerca dei valori di  $C$  e  $\sigma$ , é specificato dalle opzioni " $-\log_2 c$ " e " $-\log_2 g$ ". I tre valori che seguono le due opzioni corrispondono rispettivamente al valore di inizio, di fine e di step (start-end-step) e indicano che si vuole effettuare una ricerca logaritmica usando i valori:

$$2^{begin}, 2^{begin+step}, \dots, 2^{end}$$

L'opzione "-v" indica che si vuole effettuare una n-fold cross validation. La n-fold cross validation é un metodo dove il data set viene suddiviso in n sottoinsiemi ed il metodo holdout viene applicato n volte. Ad ogni iterazione, uno degli n sottoinsiemi viene usato come test set, mentre tutti i rimanenti n-1 sottoinsiemi formano il training set. Infine viene calcolato l'errore medio sulle n prove. L'output di questa fase é un plot che mostra attraverso delle linee colorate l'accuratezza del classificatore in funzione di  $C$  e  $\sigma$ , misurata in percentuale di classificazioni corrette. É possibile empiricamente verificare se si può ottenere migliori risultati continuando la ricerca su un differente range di valori.

### Addestramento e Test

Come si può notare la cross validation non utilizza tutti gli esempi di training, infatti ad ogni iterazione alcuni di questi vengono esclusi dalla valutazione. É necessario quindi effettuare l'addestramento sull'intero training set utilizzando i parametri che son stati determinati dalla precedente fase. La fase di training avviene tramite il comando `svm-train` nel seguente modo:

```
svm - train [-options] training_file_scalato model_file
```

Il tool fornisce diverse opzioni di training. Per una descrizione più accurata si rimanda alla documentazione ufficiale. Per prima cosa il tool permette di scegliere quale tipo di problema risolvere, quindi quale tipo di SVM utilizzare, attraverso l'opzione "-s". É possibile scegliere tra:

- C-SVC
- v-SVC
- one-class SVM
- $\epsilon$ -SVM
- v-SVR

Per quanto riguarda i kernel utilizzabili, quelli elencati sono quelli messi a disposizione dal tool. Di default é impostato il Kernel RBF.

- Lineare
- Polinomiale
- Radial Basis Function
- Sigmoidale

Una volta terminata la fase di training, si ha a disposizione la funzione decisionale appresa memorizzata nel model file e i dati di test a loro volta scalati. A questo punto é possibile effettuare le predizioni delle label per gli esempi di test attraverso il comando `svm-predict` che ha la seguente forma:

```
svm - predict [-options] test_file_scalato model_file predictions
```

Come é possibile notare, l'input del comando consiste nel test set scalato, nel modello ed il file in cui memorizzare l'output. L'output consiste nelle predizioni delle label,  $\text{sgn}(f(x))$ , del test-set.

### 3.2.4 GPDT

#### Caratteristiche Generali

Le tecniche di decomposizione precedentemente descritte sono particolarmente efficienti quando la dimensione dei sottoproblemi é molto piccola (di solito meno di  $10^2$ ), ma quando i problemi QP diventano di dimensione piú elevata questi tool non mostrano elevate performance. Il tool GPDT (Gradient Projection-based Decomposition Technique) é stato sviluppato da T.Serafini, G.Zanghirati e L.Zanni [19] ed é basato sull'idea di  $SVM^{light}$ . A differenza di quest'ultimo é sviluppato in particolar modo per ottenere efficienza decomponendo il problema in sottoproblemi di media scala. I sottoproblemi vengono risolti attraverso metodi di proiezione del gradiente che risultano essere efficienti per sottoproblemi di grandi dimensioni. Il tool GPDT é fornito di alcuni solver per il QP, ovvero SPGM, GVPM, AL-VPM, Dai-Fletcher-GPM. Le caratteristiche principali del tool sono le seguenti:

- sviluppato per decomporre in sottoproblemi di media-scala (  $O(10^2)/O(10^3)$ )
- presenta due metodi di proiezione del gradiente per i sottoproblemi QP interni
- presenta un meccanismo di caching per le valutazioni del kernel
- supporta le funzioni kernel standard e permette di definirne alcune personalizzate
- utilizza una rappresentazione dei dati a vettori sparsi (come  $SVM^{light}$ )
- efficienza su sia piccoli sia grandi sottoproblemi (di media scala sono preferibili)
- I/O simile ad  $SVM^{light}$

#### Formato dei dati

La rappresentazione adottata dal tool per i campioni di addestramento e di test é la stessa di quella di  $SVM^{light}$ , questo avvantaggia notevolmente le fasi di addestra-

mento, test e confronto. Come già descritto precedentemente, un tipico campione (sia di training sia di test) é costituito da una linea del file contenente il valore della funzione target per l'esempio considerato (es. +1 o -1), seguito da uno spazio e quindi le varie feature separate dal loro valore dal carattere ":" e disposte in ordine crescente per numero di feature.

## 3.3 Confronto tra i tools

### 3.3.1 Dataset

Nel seguente paragrafo verranno presentati i dataset e il formato dei dati utilizzati dalla SVM durante i test. L'SVM é stata addestrata con funzione Kernel Gaussiano dove il parametro  $\gamma$  é impostato a  $1.54e - 07$ .

Il dataset UCI Adult (<http://www.research.microsoft.com/jplatt/smo.html>) permette di addestrare una SVM per determinare se una famiglia ha delle entrate maggiori di \$50000 in base a 14 campi, otto dei quali sono in forma categorica, mentre gli altri sei sono in forma continua. I sei continui sono suddivisi in quintili. Il dataset é costituito da vettori sparsi binari con 123 features e livello di sparsità di  $\approx 89\%$ . Il dataset é inoltre stato suddiviso in 9 files ognuno con un numero differente di pattern. Per i test eseguiti é stato utilizzato il training file denominato UCI Adult6 costituito da 11221 elementi.

Il dataset MNIST consiste in un database di numeri manoscritti (reperibile al sito <http://yann.lecun.com/exdb/mnist>). Questo database é un sottoinsieme di un set piú grande reperibile al NIST (National Institute of Standards and Technology, [www.nist.gov](http://www.nist.gov)). Le immagini originali sono state normalizzate per rientrare in un box di 28x28 pixel e centrate calcolando il centro di massa dei pixel.

La dimensione del dataset considerato é di 10000 vettori sparsi, 5000 rappresentanti il numero 8 mentre 5000 rappresentanti gli altri numeri (da 0 a 9). Ogni pattern é costituito da 784 feature ognuna delle quali rappresenta un pixel dell'immagine.

Il dataset WEB (<http://www.research.microsoft.com/jplatt/smo.html>) riguarda la



Figura 3.1: Esempi di pattern contenuti nel dataset Mnist

classificazione di pagine web in base a 300 features chiave. In media il livello di sparsità degli esempi è del  $\approx 96\%$ . Il dataset è stato suddiviso in 7 files ognuno con un numero differente di esempi. Quello considerato nel test di addestramento è denominato web6a ed è composto da 17188 pattern.

Il dataset MEDICAL permette di addestrare una SVM per la rivelazione automatica di patologie tumorali in immagini mammografiche. Tale dataset è stato preso dal progetto CAD [3] [15] sviluppato dal gruppo di ricerca MIG [12]. Ogni pattern del dataset rappresenta un'immagine mammografica digitale generata da uno strumento chiamato mammografo digitale. La rappresentazione delle immagini è basata su ranklet [11]. Il dataset considerato è costituito da 4626 vettori ognuno dei quali è costituito da 1449 feature. I primi 1183 vettori sono positivi, quindi rappresentanti masse tumorali, mentre i restanti 3443 sono negativi, cioè zone sane.

### 3.3.2 Test Sequenziali

Le tabelle 3.4, 3.5 e 3.6, riportano i risultati ottenuti applicando il tool ai diversi dataset descritti precedentemente ed utilizzando le macchine M1, M2, M3, M64bit.

<b>Machines</b>	<b>It</b>	<b>Time (s)</b>	<b>SV</b>	<b>BSV</b>	<b>b</b>
M1	53	55.64	1591	39	6.44
M2	53	55.15	1591	39	6.4400561
M3	53	24.72	1591	39	6.4400561
M64Bit	53	17.30	1591	39	6.4400561

Tabella 3.4: GPDT Mnist-Dataset Test

<b>Machines</b>	<b>It</b>	<b>Time (s)</b>	<b>SV</b>	<b>BSV</b>	<b>b</b>
M1	48	28.77	5390	5378	-0.99941883
M2	48	28.92	5390	5378	-0.99941883
M3	48	17.09	5390	5378	-0.99941883
M64Bit	48	13.58	5390	5378	-0.99941883

Tabella 3.5: GPDT Uciadu6-Dataset Test

<b>Machines</b>	<b>It</b>	<b>Time (s)</b>	<b>SV</b>	<b>BSV</b>	<b>b</b>
M1	9	8.56	1073	1025	0.99856238
M2	9	8.50	1073	1025	0.99856238
M3	9	5.74	1073	1025	0.99856238
M64Bit	9	4.80	1073	1025	0.99856238

Tabella 3.6: GPDT Web6a-Dataset Test

Le macchine sono disposte in ordine di prestazioni. M1 ed M2 sono macchine simili ed infatti i tempi di addestramento sono molto vicini tra loro. Nelle tabelle 3.7, 3.9 e 3.8 é possibile vedere gli stessi dataset testati con il tool  $SVM^{light}$  utilizzando la macchina M3, equipaggiata con il risolutore PR-LOQO e applicando diversi valori per la dimensione dei sottoproblemi ( $q$ ) e per il massimo numero di variabili entranti nel working set ad ogni iterazione ( $n$ ). Nella tabella 3.3.2 sono presenti i risultati ottenuti con il tool LIBSVM utilizzando la macchina M3.

Da questi risultati é possibile notare che  $SVM^{light}$  ottiene delle performance ragionevoli solo quando il valore dei parametri  $q$  ed  $n$  é molto piccolo. Al crescere della dimensione dei sottoproblemi il tool evidenzia la sua inefficienza, infatti i tempi di addestramento aumentano notevolmente. Dato che la risoluzione del QP dei sottoproblemi é un compito relativamente piú semplice rispetto alle valutazioni del kernel richieste per i dati e per l'aggiornamento del gradiente, utilizzando dei

<b>q</b>	<b>n</b>	<b>Time(s)</b>	<b>SV</b>	<b>b</b>	<b>It</b>
4	2	40.62	1591	6.4398095	7500
8	4	40.11	1591	6.4399778	4127
20	10	40.13	1591	6.4398135	1639
40	20	41.30	1591	6.4401426	821
90	30	44.85	1591	6.4404503	369
400	132	78.98	1591	6.4402121	59

Tabella 3.7:  $SVM^{light}$  Results sul dataset MNIST

<b>q</b>	<b>n</b>	<b>Time(s)</b>	<b>SV</b>	<b>b</b>	<b>It</b>
4	2	33.30	5398	-0.99899041	5253
8	4	33.04	5384	-0.9988997	2949
20	10	28.65	5391	-0.99940819	1138
40	20	26.47	5390	-0.99920281	573
90	30	29.36	5395	-0.99949461	374
400	132	58.58	5514	-0.99931634	87

Tabella 3.8:  $SVM^{light}$  Results sul dataset UCI Adult6

<b>q</b>	<b>n</b>	<b>Time(s)</b>	<b>SV</b>	<b>b</b>	<b>It</b>
4	2	8.20	1077	0.99874871	855
8	4	7.84	1058	0.99882744	544
20	10	7.08	1057	0.99738375	189
40	20	6.83	1060	0.99900006	105
90	30	6.96	1050	0.99873638	64
400	132	12.05	1086	0.9984923	15

Tabella 3.9:  $SVM^{light}$  Results sul dataset WEB6a

risolutori migliori di PR\_LOQO i tempi di addestramento non migliorano.

Il tool GPDT invece, riporta ottimi risultati quando la dimensione dei sottoproblemi é elevata ( $q = 400$ ). Questo é dovuto sia alle elevate performance dei risolutori sia alla strategia usata per ridurre le valutazioni del kernel e per l'aggiornamento del gradiente ad ogni iterazione.

Mentre i tool precedenti utilizzavano una solver QP per la risoluzione dei sot-

Dataset	q	SV	Time (s)	b	Objective	It
MNIST	2	1591	32.27	6.439738	-2083.575544	8698
UCI Adult6	2	5384	24.26	-0.998976	-53828.642710	2711
WEB6a	2	1050	5.98	-0.998605	-10499.778074	525

Tabella 3.10: Risultati sequenziali con il tool LIBSVM

toproblemi, il tool LIBSVM non necessita di solver poiché l'algoritmo di decomposizione utilizzato é SMO il quale considera un working set di dimensione 2 e non necessita matrici di memorizzazione ulteriori. LIBSVM tende a minimizzare il costo computazionale per iterazione dove vengono aggiornate solo poche variabili. Ne consegue che  $q=n=2$ : il numero di variabili entranti nel working set é uguale alla dimensione dei sottoproblemi.

Il tool GPDT mostra risultati interessanti e competitivi a confronto dei ben piú noti  $SVM^{light}$  e LIBSVM. Come era prevedibile, utilizzando GPDT il numero di iterazioni richieste per raggiungere la convergenza sono molto minori rispetto agli altri due tool testati mentre le valutazioni del kernel sono maggiori. Anche per quanto riguarda i tempi di addestramento é necessario fare delle considerazioni. Infatti per i dataset MNIST e UCI Adult6, dove il livello di sparsitá dei vettori é molto elevato, GPDT ha prestazioni superiori rispetto a  $SVM^{light}$  e LIBSVM. I test sul dataset WEB6a, costituito da vettori con basso livello di sparsitá, mostrano tempi d'addestramento molto simili per tutti e tre i tool. Per quanto riguarda l'accuratezza, non vi sono "estreme" differenze tra i tool che si comportano allo stesso modo per quanto riguarda il numero di support vector e bias.

In conclusione si puó affermare che LIBSVM ed  $SVM^{light}$  si comportano bene con sottoproblemi QP di piccole dimensioni. GPDT presenta risultati interessanti e simili a gli altri due tool ben piú noti utilizzando una dimensione elevata dei sottoproblemi QP ed impiegando meno iterazioni. GPDT puó essere considerato efficiente come gli altri due tool ed ha la proprietá di essere facilmente parallelizzabile.

### 3.3.3 Proprietá dei Tools per SVM

Nelle seguenti tabelle son riportate le caratteristiche principali dei tool per SVM considerati in questa tesi.



Tool	Class				Reg		DE	Rank
	Binary		Multi					
	C-svm	v-svm	C-svm	v-svm	$\varepsilon$ -svr	v-svr		
LIBSVM	X	X	X	X	X	X	X	
$SVM^{light}$	X				X			X
$SVM^{struct}$	X		X		X			X
GPDT	X							

Tabella 3.11: Proprietá

Tool	Train		Algorithms
	Serial	Parallel	
LIBSVM	X		SMO-Type, Shrinking, Caching
$SVM^{light}$	X		Shrinking, Caching, Osuna's Decomp.
$SVM^{struct}$	X		Shrinking, Caching, Osuna's Decomp.
GPDT	X	X	GPM, GVPM, Dai-Fletcher-GPM

Tabella 3.12: Train e Algoritmi

Tool	OS	Sources	Interfaces
LIBSVM	Linux, Win	C++/Java	Python, Matlab
$SVM^{light}$	Linux, Win, Solaris, Sun OS	C/C++	Matlab, Java
$SVM^{struct}$	Linux, Win, Solaris, Sun OS	C/C++	Matlab, Java
GPDT	Linux, Windows, AIX, HP-UX	C/C++	/

Tabella 3.13: Portabilità

Tool	Input Type	License
LIBSVM	$SVM^{light}$ Format	BSD Modified
$SVM^{light}$	$SVM^{light}$ Format	Free Academic
$SVM^{struct}$	$SVM^{light}$ Format	Free Academic
GPDT	$SVM^{light}$ Format	GPL

Tabella 3.14: Licenze e Formati d'Input

# Capitolo 4

## Sistemi Paralleli

### 4.1 Introduzione

Sebbene i computer diventino sempre piú veloci, le aspettative riguardo alle prestazioni aumentano almeno altrettanto velocemente. Per quanta potenza computazionale si possa raggiungere oggi, per alcuni utenti, in particolare in campo scientifico, ingegneristico o industriale, questa non sarà mai abbastanza. Anche se le frequenze di clock sono continuamente in crescita, la velocità dei circuiti non potrà mai essere incrementata indefinitamente. Allo scopo di controllare problemi via via piú grandi e sempre piú complessi, l'architettura dei computer farà ricorso sempre maggiormente al calcolo in parallelo. Il parallelismo può essere introdotto a diversi livelli, a livello di istruzione - ad esempio - il pipelining e le architetture superscalari possono essere sfruttate per guadagnare all'incirca un fattore 10 nella performance, calcolata rispetto ad un'architettura puramente sequenziale. In ogni caso, per superare fattori di guadagno dell'ordine delle centinaia o migliaia, é necessario moltiplicare e duplicare intere CPU, o almeno sostanziali porzioni di esse, e farle lavorare in sincronia in modo il piú possibile efficiente. Una sfida dell'attuale ricerca, consiste nel creare architetture scalabili, ovvero performanti anche quando vengono aggiunti ulteriori processori. Verranno esaminati in questo capitolo i principi architetturali e software dei sistemi paralleli [18].

## 4.2 Tassonomia dei Sistemi Paralleli

Nel corso degli anni sono stati progettati, proposti o costruiti diversi tipi di computer paralleli, ci si chiede quindi se é possibile categorizzarli tramite una tassonomia. L'unica classificazione con una certa utilit   é quella proposta da Flynn [7] anche se fatta con una certa approssimazione. La tassonomia di Flynn si basa su due concetti fondamentali: instruction stream e data stream. L' instruction stream é in relazione al program counter (il registro che contiene la prossima istruzione da eseguire). Un sistema con  $n$  CPU, ha  $n$  program counter visto che ha  $n$  instruction stream da calcolare. Un data stream consiste in un insieme di operandi. I data stream e le instruction stream sono in qualche modo indipendenti, quindi é possibile avere le quattro combinazioni in figura.

Instruction Stream	Data Stream	Nome	Esempi
1	1	SISD	Macchina di Von Neumann
1	Multiple	SIMD	Vector Supercomputer
Multiple	1	MISD	Raramente visibili
Multiple	Multiple	MIMD	Multiprocessore, Multicomputer

Tabella 4.1: Tassonomia dei sistemi paralleli

Il sistema SISD corrisponde alla classica macchina di Von Neumann sequenziale con un'unica sequenza di istruzioni, un'unica sequenza di dati ed effettua una sola operazione per volta. I sistemi paralleli in tabella sono quindi le macchine SIMD, MISD e MIMD.

- SIMD (Single Instruction stream Multiple Data stream) si riferisce a quei computer usati per risolvere problemi sia scientifici sia ingegneristici molto rilevanti con strutture dati regolari, quali vettori o array. Questi sistemi sono caratterizzati dal fatto di avere un'unica unit   di controllo che esegue un'istruzione per volta e ciascuna di queste istruzioni opera su molti dati contemporaneamente. I due tipi principali di macchine SIMD sono chiamati Array Processor e Vector Processor. Gli Array Processor sono basati sull'idea che una singola unit   di controllo guidi gli altri elementi di calcolo utilizzando opportuni segnali. Ognuno di questi elementi di calcolo é costituito da una CPU o da un'ALU pi   complessa e da della memoria locale. Dal momento che vi é una

sola unità di controllo che governa gli elementi di calcolo, allora questi lavorano contemporaneamente. Anche se questo modello è il più utilizzato dagli array processor, esistono alcune differenze architetturali tra i vari tipi che riguardano la struttura, l'interconnessione e l'autonomia degli elementi di calcolo. I Vector Processor sono sistemi che riescono a prendere coppie di vettori di  $n$  elementi in input e operano in parallelo sui corrispondenti elementi di ciascun vettore usando un vettore di unità logiche che lavora su tutti gli elementi corrispondenti simultaneamente. Il risultato di questa operazione è un vettore che può essere salvato in memoria o in speciali registri vettoriali. Il problema fondamentale di tali macchine non è tecnico ma economico. Può essere ricordato a titolo d'esempio il Cray-1, uno dei primi supercomputer vettoriali.

- MISD (Multiple Instruction stream Single Data stream) si riferisce a quella particolare categoria di sistemi dove più processori eseguono contemporaneamente differenti operazioni sullo stesso insieme di dati. Questi tipi di sistemi sono visibili raramente e vengono anche chiamati systolic arrays.
- MIMD (Multiple Instruction stream Multiple Data stream) indica quei tipi di sistemi dotati di più CPU che operano all'interno di grandi sistemi. Questi sistemi si possono dividere sostanzialmente in due tipi che ora verranno esaminati, i multiprocessori e i multicomputer.

### 4.2.1 Multiprocessori

Un multiprocessore è un sistema dotato di più CPU ed un unico spazio di indirizzamento visibile a tutte le CPU. Un sistema di questo tipo viene spesso anche chiamato sistema a memoria condivisa (shared memory) ed è dotato di un unico sistema operativo: questo fatto già lo differenzia dai multicomputer. Un multiprocessore come ogni altro sistema è dotato di alcuni dispositivi di I/O. Mentre in alcuni sistemi solo alcune CPU hanno accesso ai device di I/O, in altri ogni CPU ha la possibilità di accedere a qualsiasi device. Quando ci si trova nelle condizioni in cui ogni CPU ha la stessa possibilità di accedere ad ogni modulo di memoria e ad ogni dispositivo di I/O ed è per il sistema operativo intercambiabile con ogni altra CPU, il sistema viene chiamato SMP (Symmetric Multi-Processor). Esistono tre tipi di multiprocessori che differiscono nel modo in cui viene realizzata la memoria condivisa, ovvero UMA

(Uniform Memory Access), NUMA (Non Uniform Memory Access), COMA (Cache Only Memory Access). Solitamente nei multiprocessori la memoria viene suddivisa in moduli. Nelle macchine UMA l'accesso ai moduli di memoria avviene con lo stesso identico tempo di accesso. Quando questo non é possibile, le componenti piú veloci vengono adeguate ai tempi di quelle piú lente. Un esempio di mutiprocessore UMA é il Sun Enterprise 10000 . Nei multiprocessori NUMA questa proprietá di accesso uniforme ai moduli non viene garantita, infatti spesso il modulo piú vicino alla CPU presenta tempi d'accesso piú veloci rispetto ai moduli piú lontani. Un esempio di multiprocessore CC-NUMA (cache coherent) é il DASH di Stanford. Un tipo alternativo di multiprocessore viene chiamato COMA, il quale utilizza la memoria principale delle CPU come una cache.

### **4.2.2 Multicomputer**

I multicomputer a differenza dei multiprocessori non sono dotati di nessuna memoria condivisa a livello hardware. In questo sistema una CPU puó accedere alla memoria di un'altra CPU attraverso un meccanismo indiretto di messaggi. Ogni nodo in un multicomputer consiste in una o piú CPU, di RAM, un disco con altri device di I/O ed un processore per la comunicazione connesso ad altri tramite una rete d'interconnessione. I multicomputer possono essere suddivisi in due grandi classi, gli MPP (Massively Parallel Processors) e i COW (Cluster Of Workstations). Gli MPP sono macchine molto costose utilizzate nelle applicazioni scientifiche, nell'ingegneria e nell'industria per effettuare grandi masse di calcoli. Si puó dire che queste macchine hanno rimpiazzato le macchine SIMD ed usano CPU standard come processore (Pentium, UltraSparc). Una caratteristica molto importante di questi sistemi consiste nell'utilizzo di una rete d'interconnessione ad elevata velocitá progettata per lo sfruttamento di un'ampia larghezza di banda e trasmissione a bassa latenza. Questi sistemi possiedono inoltre un'enorme capacitá di I/O e un'elevata resistenza ai guasti. Alcuni esempi di questi sistemi MPP sono il Cray T3E, l'Intel Sandia Option Red e l'IBM SP/2 . La seconda categoria di multicomputer, consiste in quella chiamata COW. Si tratta di un sistema composto da semplici workstation connesse tra loro tramite una scheda di rete commerciale. Le differenze tra un MPP e un COW sono ben poche. Storicamente la differenza principale era data dal fatto che gli MPP sfruttavano interconnessioni ad alta velocitá, ma con l'avvento di dispositivi ad alta

efficienza reperibili in commercio il divario tra i due sistemi si é molto ridotto. Esistono due tipologie dominanti di COW: centralizzate e decentralizzate. Un COW centralizzato é un cluster di workstation montate su un rack e poste in un'unica stanza, chiamate anche headless workstation. Un COW decentralizzato invece consiste in alcune workstation sparse in un area di media grandezza non sempre attive e collegate tramite LAN. Mentre in una LAN ogni utente dispone di una propria macchina, in un COW decentralizzato un utente dispone di piú macchine alle quali può sottoporre job che richiedono piú CPU.

### 4.2.3 Modelli di comunicazione tra processori

Nei sistemi paralleli, le CPU che lavorano su differenti parti dello stesso job devono comunicare tra loro per scambiarsi informazioni. Questo può avvenire in due modi:

1. Attraverso la memoria condivisa (Shared Memory). Tramite questa tecnica la memoria viene condivisa tra i processori. Qualsiasi processo può leggere o scrivere una parola di memoria eseguendo semplici istruzioni. Due processi possono comunicare in maniera molto semplice: uno scrive in memoria e l'altro legge dalla memoria. Purtroppo però vengono introdotti una serie di problemi. Molte CPU potrebbero cercare di leggere una word contemporaneamente, mentre altre potrebbero scrivere sulla stessa locazione di memoria. Inoltre alcuni messaggi potrebbero arrivare con ordini differenti dalle aspettative. Per risolvere questi problemi sono state introdotte alcune regole chiamate modelli di consistenza.
2. Attraverso lo scambio di messaggi (Message Passing). I processori tramite questa tecnica comunicano tramite un rete d'interconnessione tra i nodi della struttura parallela. Per realizzare il message passing sono necessarie delle librerie specializzate per controllare le comunicazioni tra i processi e la sincronizzazione. I sistemi che utilizzano lo scambio di messaggi hanno due o piú processi che lavorano indipendentemente ed hanno due primitive fondamentali chiamate send e receive.

## 4.3 Comunicazione tra Multicomputer

Per la programmazione di multicomputer, é necessario utilizzare librerie specializzate che permettano di controllare le comunicazioni tra processi e la sincronizzazione. Spesso il software eseguito su MPP può essere eseguito anche su COW. I sistemi che utilizzano il message passing presentano due primitive importanti chiamate send e receive. A queste due primitive possono corrispondere differenti semantiche, ma le principali varianti sono:

1. Scambio sincrono di messaggi (bloccante)
2. Scambio di messaggi bufferizzati
3. Scambio di messaggi non bloccanti

Nello scambio sincrono di messaggi se il mittente esegue una send ed il ricevente non ha ancora risposto attraverso una receive, il mittente si blocca in attesa. Quando il mittente ottiene nuovamente il controllo, dopo una chiamata, sa che il messaggio é stato ricevuto correttamente. Lo svantaggio principale di questa tecnica é dato dal fatto che il mittente rimane bloccato in attesa, finché il ricevente non ha ricevuto il messaggio e ha inviato una risposta della corretta ricezione. Nello scambio di messaggi bufferizzati, quando il messaggio viene spedito prima che il ricevente sia pronto, il messaggio viene bufferizzato in un area apposita, dove rimane fino a quando non viene prelevato dal ricevente. Tramite questa tecnica il mittente non é costretto a bloccarsi dopo l'esecuzione di una send nel caso in cui il ricevente sia occupato, ma continua nel suo lavoro. In questa situazione, il problema fondamentale riguarda la garanzia che il messaggio sia arrivato correttamente al ricevente, garanzia che non é assicurata anche se il canale é affidabile. Nello scambio di messaggi non bloccanti, il mittente ha la possibilità di continuare nel suo lavoro dopo aver effettuato una send se il ricevente é occupato. Le librerie avvertono solo il sistema operativo che più tardi deve rieffettuare la chiamata. Il mittente quindi non si blocca se il ricevente é occupato ma questa continuazione porta lo svantaggio dell'impossibilità dell'utilizzo del buffer dei messaggi dato che il messaggio non é ancora stato spedito. Per far fronte a questo problema si possono utilizzare diversi metodi, come il polling oppure un interrupt quando il buffer ridiventa disponibile. Due sistemi di message pas-

sing molto diffusi sui multicomputer sono PVM (Parallel Virtual Machine) e MPI (Message Passing Interface).

### 4.3.1 Parallel Virtual Machine

PVM é un sistema di scambio di messaggi progettato inizialmente per funzionare sui COW basati su Unix. Il software é composto da due parti, una libreria richiamabile da utente e da un processo demone che gira continuamente su ogni macchina del multicomputer. Quando PVM parte, determina quale macchine fanno parte del suo multicomputer virtuale leggendolo da un file di configurazione. Un demone PVM viene fatto partire su ognuna di queste macchine. PVM permette di lanciare  $n$  processi paralleli attraverso il comando `spawn`. Ogni processo PVM ad ogni istante ha un send buffer attivo ed un receive buffer attivo. Per l'invio di un messaggio, un processo chiama una procedura di libreria per impacchettare i valori in un send buffer attivo e che siano decodificabili dal ricevente. Una volta assemblato il messaggio, viene chiamata la procedura `pvm_send` dal mittente, la quale é di tipo bloccante. Il ricevente risponde a questa send tramite una `pvm_receive` che lo blocca. Se il ricevente é preoccupato di restare bloccato per sempre, può effettuare una chiamata che lo blocca per un certo intervallo di tempo. Oltre a queste primitive PVM supporta anche il broadcasting ed il multicasting. La sincronizzazione tra i processi avviene tramite la procedura `pvm_barrier`.

### 4.3.2 Message Passing Interface

Un altro package per programmare i multicomputer é MPI (Message Passing Interface). Questo package contiene molte pi procedure rispetto al package precedente ed é basato su quattro concetti fondamentali: i communicator, i messaggi di dati, le operazioni di comunicazione e le topologie virtuali. Un communicator é un gruppo di processi ed un contesto. Il contesto é una label che identifica qualcosa, come una fase di esecuzione. Quando un messaggio viene spedito o ricevuto, il contesto può essere usato per impedire che messaggi non correlati interferiscano tra loro. I messaggi sono tipizzati e sono disponibili molti tipi di dati. MPI supporta anche un insieme notevole d'operazioni di comunicazione. Attraverso le funzioni `MPI_Send` e `MPI_Receive` é possibile inviare e ricevere messaggi. MPI supporta quattro mo-



dalit  di comunicazione: sincrona, bufferizzata, standard e ready. Queste primitive sono disponibili in maniera bloccante e non bloccante. MPI inoltre supporta metodi di comunicazione collettiva. L'ultimo concetto fondamentale di MPI   la topologia virtuale: i processi possono essere organizzati secondo diverse topologie, come ad albero, ad anello, a toroide. Questo permette di fonire un modo per assegnare un nome alle interconnessioni e per facilitare la comunicazione.

### 4.3.3 Installazione di MPI

Nella seguente sezione viene descritta l'installazione di MPI su un sistema Unix [16] [17]. La versione di MPI al momento disponibile   la 2.0 .

I passi i per la compilazione sono:

```
cd /usr/local
tar xzf mpi-2.0.tar.gz
mv mpi-2.0 mpich
./configure --arch=LINUX --device=ch_p4 --listenersig=SIGUSR2 --
mpe --comm=shared --enable-c++ --opt=O --disable-dev --debug
--prefix=/usr/local/mpichi --exec-prefix=/usr/local/mpich
make serv_p4
make
make install
```

Tabella 4.2: Compilazione MPI

Le opzioni di configurazione sono:

1. installare librerie ed eseguibili sotto la root */usr/local/mpich*;
2. usare il device *ch\_p4*, utilizzato per la comunicazione in un cluster;
3. utilizzo del SIGUSR 2, per poter essere utilizzata con i thread (altrimenti genera un conflitto di segnali)
4. compilare il supporto per l'interfaccia di debug MPE;
5. compilare le librerie in versione dinamica;
6. disabilitare il debug del codice della libreria.

Nel file `/usr/local/mpich/var/machines.LINUX`, vanno inserite, una per ogni riga, i nomi delle macchine facenti parte del cluster con il relativo numero di processori  $n_p$  che si vuole utilizzare sulla macchina separati da ":" :

$$\text{hostname} : n_p$$

Nel file `/etc/skel/.bashrc` devono essere inserite le seguenti righe:

```
export P4.GLOBMEMSIZE=200000000
export PATH=$PATH:/usr/local/mpich/bin
```

Questa indica di utilizzare una dimensione massima di circa 200 MB per i pacchetti di MPI; saranno poi le stesse librerie che li convertiranno in piú pacchetti TCP validi di dimensione adeguata. Questo valore é idoneo al nostro cluster, ma va modificato secondo le relative necessità. In questo modo tutti i nuovi utenti aggiunti utilizzeranno questo valore, la seconda riga aggiunge la directory con gli eseguibili per mpi nel path. Infine, i nomi di tutti gli host, o in alternativa gli indirizzi IP, vanno scritti nel file `/etc/hosts.equiv` uno per riga.

Questo passo si rende necessario per utilizzare correttamente i comandi `rexec` e `rsh` senza dover inserire di volta in volta la password dell'utente che li ha invocati; MPI utilizza pesantemente questi comandi e saltarlo renderebbe inutilizzabile quest'ultimo. Il file delle password viene sincronizzato attraverso uno script che copia i file `passwd`, `shadow`, `group` e `sgroup` nelle directory `/etc` dei nodi. Lo script va lanciato ad ogni creazione di nuovi utenti. Ora i programmi potranno essere eseguiti attraverso il comando `mpirun`.

## 4.4 Symmetric Multi-Processor

SMP consiste in un'architettura dove ogni CPU ha la stessa possibilità di accedere ad ogni modulo di memoria e ad ogni dispositivo di I/O ed é per il sistema operativo intercambiabile con ogni altra CPU. Con il supporto adeguato del sistema operativo, i sistemi SMP possono passare i task ai processori bilanciando efficientemente il carico di lavoro. Quello che differenzia i sistemi SMP é il modo in cui si accede alla memoria condivisa. Alcuni presentano una memoria cache di secondo livello L2

indipendente. Questa tecnica, che dovrebbe portare maggiore efficienza computazionale, nel sistema operativo Linux non é ovvia. La possibilità di passare i processi ai vari processori unita alla presenza di una cache separata, comporta un overhead significativo. Al momento sono presenti una varietà di sistemi operativi e supporti hardware per gli SMP. Con il kernel Linux 2.6 si ha il supporto della libreria nativa POSIX, per quanto concerne le performance delle applicazioni multithread, che permette di scalare almeno a 32 processori, superando la limitazione degli 8 SMP del kernel 2.4 .

# Capitolo 5

## Parallel GPDT

### 5.1 Introduzione

Una caratteristica molto importante del tool GPDT é la sua facile parallelizzazione. PGPDT (Parallel Gradient Projection-based Decomposition Technique) consiste in una tecnica parallela di decomposizione basata sulla proiezione del gradiente sviluppata da L.Zanni, T.Serafini e G.Zangherati [10] [8] e reperibile al sito <http://www.dm.unife/gpdt> . In questo capitolo viene descritto come avviene la parallelizzazione e come può essere sfruttato un sistema multiprocessore per avere elevate performance rispetto ai packages di addestramento sequenziali.

### 5.2 Tecnica Parallela di Decomposizione

Per descrivere la tecnica di decomposizione utilizzata, é necessario definire una notazione di base. Si definisce con  $B$  un insieme di variabili base, solitamente chiamato working set, e con  $N$  un insieme di variabili non base. Queste variabili sono generate dallo splitting degli indici  $i = 1, \dots, n$  ad ogni iterazione di decomposizione. La matrice kernel  $G$  e i vettori  $x$  e  $y$  possono essere disposti rispetto a  $B$  e a  $N$  nel seguente modo:

$$G = \begin{bmatrix} G_{BB} & G_{BN} \\ G_{NB} & G_{NN} \end{bmatrix}, \quad x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}, \quad y = \begin{bmatrix} y_B \\ y_N \end{bmatrix}$$

Si definisce con  $n_{sp}$  la dimensione del working set, con  $x^*$  una soluzione del QP e con  $N_p$  il numero di processori utilizzati nel sistema a memoria distribuita con-

siderato. La strategia di decomposizione di PGPDT segue dallo schema generale dell'algoritmo PDT descritto in [10], che a sua volta segue dallo schema di decomposizione proposto da Joachims [9] nel tool  $SVM^{light}$ . A differenza di quest'ultimo però, PDT permette di distribuire la fase di computazione del QP e la fase di aggiornamento del gradiente tra i vari processori. La tecnica parallela di decomposizione risulta essere le seguente:

1. Inizializzazione. Viene settato  $x^{(1)} = 0$  ed  $n_{sp}$  e  $n_c$  tali che  $n \geq n_{sp} \geq n_c$ , con  $n_c$  pari. Vengono scelti  $n_{sp}$  indici per il working set  $B$  e settato  $k \leftarrow 1$
2. Soluzione distribuita del sottoproblema QP. Viene calcolata la soluzione  $x_B^{(k+1)}$  di

$$\min \frac{1}{2} x_B^T G_{BB} x_B + (G_{BN} x_N^{(k)} - 1_B)^T x_B$$

soggetto a

$$\begin{aligned} \sum_{i \in B} y_i x_i &= - \sum_{i \in N} y_i x_i^{(k)}, \\ 0 \leq x_i &\leq C, \quad \forall i \in B, \end{aligned}$$

$$\text{dove } 1_B = n_{sp} - \text{vettore e } x^{(k+1)} = (x_B^{(k+1)T}, x_N^{(k)T})^T$$

3. Aggiornamento distribuito del Gradiente. Aggiornamento del gradiente con

$$\nabla F(x^{(k+1)}) = \nabla F(x^{(k)}) + \begin{bmatrix} G_{BB} \\ G_{NB} \end{bmatrix} (x_B^{(k+1)} - x_B^{(k)}) \quad (5.1)$$

e terminazione se  $x^{(k+1)}$  soddisfa le condizioni KKT .

4. Aggiornamento del Working Set. Si aggiorna  $B$  con la seguente regola

- (a) Si cercano gli indici corrispondenti alle variabili non zero della soluzione di

$$\min F(x^{(k+1)})^T d$$

soggetto a

$$\begin{aligned}
y^T d &= 0, \\
d_i &\geq 0 \quad \text{per } i \text{ tale che } x_i^{(k+1)} = 0, \\
d_i &\leq 0 \quad \text{per } i \text{ tale che } x_i^{(k+1)} = C, \\
-1 &\leq d_i \leq 1, \\
\# \{d_i | d_i \neq 0\} &\leq n_c
\end{aligned}$$

Sia  $B^*$  l'insieme di questi indici.

- (b) Si inseriscono in  $B^*$  fino a  $n_{sp}$  elementi con indici appartenenti a  $B$ .
- (c) Si setta  $B = B^*, k \leftarrow k + 1$  e si ritorna al passo 2.

La seguente procedura iterativa usata nel PGPD, viene terminata quando le condizioni KKT sono soddisfatte secondo un certo valore di tolleranza che per default é impostato a  $10^{-3}$ . Questo tool é effettivamente efficace per valori di  $n_{sp}$  sufficientemente grandi, quindi si assume che la dimensione dei sottoproblemi sia di media scala (tra  $10^2$  e  $10^3$ ).

## 5.3 Risoluzione Parallela dei QP

I sottoproblemi QP del passo 2 dell'algoritmo precedente, possono essere visti nella seguente forma piú generale:

$$\min_{w \in \Omega} f(w) = \frac{1}{2} w^T A w + b^T w \quad (5.2)$$

dove  $A \in \mathbb{R}^{n_{sp}}$  é densa, simmetrica e definita positiva,  $w, b \in \mathbb{R}^{n_{sp}}$  e la regione  $\Omega$  é definita come:

$$\Omega = \{w \in \mathbb{R}^{n_{sp}}, l \leq w \leq u, c^T w = \gamma\}, \quad l, u, c \in \mathbb{R}^{n_{sp}}, l < u \quad (5.3)$$

Dato che 5.2 compare ad ogni iterazione di decomposizione, allora é necessario un solver QP efficiente per le performance di questa tecnica di decomposizione. I solver classici come PR\_LOQO [1] o MINOS [13], possono essere applicati solamente nel caso in cui la dimensione dei sottoproblemi é piccola (generalmente minore di  $10^2$ ) in quanto per sottoproblemi di media scala le performance degradano facilmente.

Per lo schema di decomposizione usato dal PGPDT é importante avere un solver QP efficiente che sfrutti le caratteristiche della 5.2 e sia facilmente parallelizzabile. I metodo di proiezione del gradiente sono quindi approcci molto invitanti. Questi consistono in una sequenza di proiezioni in una regione  $\Omega$  e comportano operazioni non costose per i vincoli del problema considerato. La proiezione può essere effettuata in  $O(n_{sp})$  operazioni. L'algoritmo PGPM é uno schema generale del parallel gradient projection, dove ad ogni iterazione una possibile direzione di discesa  $d^{(k)}$  viene ottenuta proiettando un punto su  $\Omega$ , che deriva dal prendere il passo di discesa più ripida dal corrente  $w^k$  con lunghezza del passo  $\alpha_k$ . Viene poi applicata una procedura chiamata linesearch sulla direzione  $d^{(k)}$  per decidere se la lunghezza dello step  $\lambda_k$  permette di garantire la convergenza globale. La parallelizzazione viene quindi ottenuta attraverso una distribuzione di blocchi della matrice  $A$  e attraverso il calcolo parallelo del più difficile task di ogni iterazione ovvero il prodotto  $Ad^{(k)}$ . La scelta dei parametri di lunghezza del passo  $\alpha_k$  e il parametro  $\lambda_k$  nella linesearch giocano un ruolo molto importante per la velocità di convergenza. Per la lunghezza dello step buoni risultati sono stati ottenuti con strategie basate sulle regole di Barzilai e Borwein [2]. L'algoritmo PGPM può essere schematizzato nel seguente modo:

1. Inizializzazione.

- (a) Distribuzione dei dati ai processori. Per ognuno degli  $N_p$  processori assegno una porzione di  $A$  tale che  $A_p = (a_{ij})_{i \in I_p}, j = 1, \dots, n_{sp}$ , dove  $I_p$  é un sottoinsieme di indici di riga assegnati al processore  $p$ . Questi sottoinsiemi sono disgiunti e la loro unione copre tutti gli indici possibili della matrice. Viene inoltre allocata una copia locale dei dati in input e vengono inizializzati i parametri di steplength e linesearch nel seguente modo:

$$w^{(0)} \in \Omega, 0 < \alpha_{min} < \alpha_{max}, \quad \alpha_0 \in [\alpha_{min}, \alpha_{max}], k = 0$$

- (b) Step Distribuito. Per ognuno degli  $N_p$  processori viene calcolata la propria porzione  $t_p^{(0)} = A_p w^{(0)}$  e viene inviata agli altri processori; si assembla una copia locale di  $t^{(0)} = Aw^{(0)}$ .

$$\text{Si setta } g^{(0)} = \nabla f(w^{(0)}) = Aw^{(0)} + b = t^{(0)} + b$$

2. Proiezione. Si termina se  $w^{(k)}$  soddisfa il criterio di terminazione altrimenti si calcola la direzione di discesa:

$$d^{(k)} = P_{\Omega} \left( w^{(k)} - \alpha_k g^{(k)} \right) - w^{(k)}$$

3. Prodotto Matrice-Vettore distribuito. Per ognuno degli  $N_p$  processori viene calcolata una porzione locale  $z_p^{(k)} = A_p d^{(k)}$  e viene inviata agli altri processori; viene assemblata poi una copia locale completa  $z^{(k)}$ .
4. Linesearch. Viene calcolato lo step  $\lambda_k$  e  $w^{(k+1)} = w^{(k)} + \lambda_k d^{(k)}$
5. Aggiornamento. Viene calcolato

$$\begin{aligned} t^{(k+1)} &= Aw^{(k+1)} = t^{(k)} + \lambda_k Ad^{(k)} = t^{(k)} + \lambda_k z^{(k)}, \\ g^{(k+1)} &= \nabla f(w^{(k+1)}) = Aw^{(k+1)} + b = t^{(k+1)} + b \end{aligned}$$

e il nuovo steplength  $\alpha_{k+1}$ . Vengono inoltre aggiornati i parametri della linesearch,  $k = k + 1$  e si ritorna al passo 2.

Il tool PGPDPT può eseguire il metodo di proiezione del gradiente parallelo appena descritto seguendo sia lo schema GVPM sia lo schema Dai-Fletcher. Risultati sperimentali hanno mostrato che il solver GVPM è molto più efficiente rispetto a pr\_LOQO [1] e MINOS [13] largamente utilizzati dalla machine learning community.

## 5.4 Aggiornamento Parallelo del Gradiente

L'aggiornamento del gradiente è il lavoro solitamente più dispendioso in un'iterazione di decomposizione. Tecniche per risparmiare valutazioni del kernel o ottimizzare la loro computazione sono importanti per minimizzare il tempo di aggiornamento del gradiente. Alcuni miglioramenti nel numero di valutazioni del kernel possono essere ottenuti introducendo strategie di caching, ovvero l'utilizzo di un'area di memoria per la memorizzazione di alcuni elementi della matrice  $G$  per evitare il successivo ricalcolo nelle iterazioni successive. Il tool PGPDPT inserisce nell'area di caching le colonne di  $G$  ottenute al passo di aggiornamento del gradiente nello schema PDT descritto precedentemente. Quando la cache è piena le colonne correnti sostituiscono quelle meno usate. I miglioramenti apportati dal caching, sono dipendenti dalla



dimensione dell'area utilizzata. La quantità di memoria disponibile nei sistemi multiprocessori permette di migliorare le performance della tecnica di decomposizione. Il tool GPDT implementa un aggiornamento parallelo del gradiente dove il prodotto matrice-vettore e la strategia di caching viene distribuita tra i processori. Viene chiesto ad ogni processore di effettuare una parte delle combinazioni tra colonne e rendere disponibile una parte della sua memoria locale per il caching di colonne di  $G$ . In questo modo l'aggiornamento del gradiente beneficia non solo della distribuzione del calcolo ma anche della riduzione delle valutazioni del kernel.

## 5.5 Selezione del Working Set

Nel tool PGPDT la scelta del working set aggiornato avviene in due fasi. Nella prima di queste al più  $n_c$  indici vengono scelti per il nuovo working set, nella seconda fase almeno  $n_{sp} - n_c$  elementi sono selezionati da  $B$  per completare il nuovo working set. La procedura di selezione è correlata alla violazione delle condizioni di ottimalità. Per la convergenza, la scelta di  $n_c$  e la fase di aggiornamento del nuovo working set hanno un ruolo importante. Infatti scegliendo  $n_c = n_{sp}$  si ottiene un fenomeno di zigzagging. Quindi  $n_c$  dovrebbe essere scelto più piccolo rispetto a  $n_{sp}$ . Per scegliere i successivi indici con i quali riempire il working set vi è una strategia proposta in [20] che introduce anche una riduzione del parametro  $n_c$  utile per working set di grandi dimensioni.

## 5.6 Implementazione

Il tool PGPDT è scritto in standard C++ e utilizza le librerie di comunicazione parallela MPI in quanto sono portabili su molti sistemi multiprocessore. La parallelizzazione è imposta sugli step dell'algoritmo di decomposizione che richiedono la maggiore computazione. Per far questo vengono distribuiti blocchi di righe della matrici  $Q_{BB}$  e  $Q_{BN}$  ai processori disponibili, ed in questo modo è possibile sfruttare l'elevata memoria disponibile per strategie di caching e parallelizzare il calcolo.

# Capitolo 6

## Risultati Sperimentali

### 6.1 Test Paralleli di Addestramento

I seguenti esperimenti, riguardano il comportamento del tool PGPD<sup>T</sup> sui dataset descritti nel paragrafo 3.3.1. Negli esperimenti, l'SVM viene addestrata con kernel gaussiano e con gli stessi parametri utilizzati nei precedenti test sequenziali, ovvero con la dimensione dei sottoproblemi pari a 400, con il numero di variabili entranti nel working set pari a 132 e con la dimensione della cache per le valutazioni del kernel pari a 300MB per ogni CPU. Le macchine utilizzate per i test sono descritte in Appendice A [22]. Il numero di processori totali utilizzati sono 8 dei quali uno dotato di Hyper-Threading (HT). Tale cluster é un ibrido a 32 e 64 bit, quindi si avrà una valutazione finale del tool su un sistema che può essere definito eterogeneo, ovvero composto da macchine disuguali dal punto di vista prestazionale.

Le tabelle 6.1, 6.2 e 6.3, sommarizzano i risultati ottenuti eseguendo PGPD<sup>T</sup> su differenti combinazioni di macchine e di conseguenza su differenti combinazioni di processori. Nell'analisi delle performance del tool parallelo, viene preso come riferimento il corrispondente tool sequenziale in grado di risolvere lo stesso problema. Queste tabelle, oltre a mostrare il tempo di esecuzione parallelo del tool, mostrano anche altre importanti informazioni che riguardano il suo comportamento. É possibile notare che l'accuratezza e il numero di decomposizioni, sono le stesse della versione sequenziale mostrata nelle tabelle 3.4, 3.5 e 3.6. L'accuratezza viene valutata in base al numero di Support Vectors (SV), in base al numero di Bound Support Vectors (BSV) ed in base al valore della soglia (b). É importante notare anche che

il numero di iterazioni di decomposizione (It) é costante.

Come si nota nelle tabelle 6.1, 6.2 e 6.3 l'impiego della macchina M6 nei test influenza notevolmente il tempo di addestramento parallelo dell'SVM. Dalle sue caratteristiche mostrate in Appendice A, é evidente che questa macchina risulta essere meno performante delle altre e come conseguenza si ha che l'algoritmo si deve "adeguare" ai suoi tempi di calcolo.

Utilizzando il tool parallelo, l'accuratezza rimane pressoché identica a quello seriale per quanto riguarda il numero di Support Vector (SV), il numero Bound Support Vector (BSV, tali per cui  $\alpha_i = C$ ), il bias (b) e le iterazioni (It).

Il tempo d'esecuzione ovviamente decresce all'aumentare dei processori fino ad un punto nel quale continuando ad inserire cpu, non si hanno sostanziali miglioramenti.

## 6.2 PGPDt: Scaling su un cluster ibrido 32/64 bit

Vengono valutate le performance del tool parallelo attraverso lo speedup  $S$ , ovvero il fattore che si guadagna nell'esecuzione rispetto ad un uniprocessore.  $S$  é definito come il rapporto tra il tempo di esecuzione sequenziale (quindi su singolo processore)  $T_s$  e il tempo di esecuzione parallelo ( $T_{N_p}$ ) con un numero di processori  $P$  che varia tra  $P = 2$  e  $P = 7$ . Il caso ideale si ha quando lo speedup é lineare nel numero di processori, ma purtroppo é raramente raggiunto.

Nelle figure riportate in questo paragrafo, oltre allo speedup misurato utilizzando il PGPDt su diversi dataset, verrà visualizzato anche lo speedup lineare per permettere di valutare la deviazione. Nelle figure dello scaling e dello speedup viene considerato il caso pessimo registrato nei test senza considerare la macchina M6, in quanto rappresenta un limite allo speedup e allo scaling del cluster utilizzando PGPDt con i dataset descritti.

Si definiscono:

$$Speedup = \frac{T_s}{T_{N_p}} \quad (6.1)$$

$$Efficienza = \frac{Speedup}{P} \quad (6.2)$$

Oltre a queste misure di prestazione, viene introdotta anche la legge di Amdahl:

$$Speedup_{amdahl} = \frac{1}{Seq + \frac{(1-Seq)}{N_p}} \quad (6.3)$$

dove  $Seq$  rappresenta la parte di programma eseguita sequenzialmente,  $1 - Seq$  é la frazione eseguita parallelamente e  $Speedup_{amdahl}$  é lo speedup che teoricamente può essere raggiunto con l'utilizzo di  $N_p$  processori. Dall'equazione 6.2, si nota che  $1/Seq$  é il valore limite di  $Speedup_{amdahl}$  nel caso di un numero infinito di processori. Il valore ideale si ha quando  $Seq = 0$ . Quindi il miglioramento delle prestazioni é

MNIST DATASET						
Proc	Macchine	Time(s)	It	SV	BSV	b
2	M1	39.58	55	1590	39	6.4401868
2	M2	39.02	55	1590	39	6.4401868
2	M3	34.21	55	1590	39	6.4401868
2	M6	97.53	55	1590	39	6.4401868
3	M1+M4	27.33	55	1589	39	6.4403787
3	M2+M4	27.04	55	1589	39	6.4403787
3	M3+M4	22.1	55	1589	39	6.4403787
3	M4+M6	67.83	55	1589	39	6.4403787
4	M1+M2	19.00	52	1589	39	6.4400569
4	M1+M3	20.88	52	1589	39	6.4400569
4	M3+M6	50.93	52	1589	39	6.4400569
4	M2+M3	20.9	52	1589	39	6.4400569
5	M1+M3+M4	18.01	54	1590	39	6.4403525
5	M2+M3+M4	16.46	54	1590	39	6.4403525
5	M1+M2+M4	17.97	54	1590	39	6.4403525
5	M3+M4+M6	42.02	54	1590	39	6.4403525
6	M1+M2+M3	15.07	53	1590	39	6.4398316
6	M2+M3+M6	35.09	53	1590	39	6.4398316
7	M1+M2+M3+M4	13.72	54	1591	39	6.4404683
7	M2+M3+M4+M6	31.74	54	1591	39	6.4404683

Tabella 6.1: MNIST Parallel Tests

UCI Adult6 DATASET						
Proc	Macchine	Time(s)	It	SV	BSV	b
2	M1	19.79	48	5390	5378	-0.99941883
2	M2	19.22	48	5390	5378	-0.99941883
2	M3	18.2	48	5390	5378	-0.99941883
2	M6	40.53	48	5390	5378	-0.99941883
3	M1+M4	13.7	48	5390	5378	-0.99941883
3	M2+M4	13.42	48	5390	5378	-0.99941883
3	M3+M4	12.08	48	5390	5378	-0.99941883
3	M4+M6	27.18	48	5390	5378	-0.99941883
4	M1+M2	10.55	48	5390	5378	-0.99941883
4	M1+M3	10.55	48	5390	5378	-0.99941883
4	M3+M6	20.97	48	5390	5378	-0.99941883
4	M2+M3	10.71	48	5390	5378	-0.99941883
5	M1+M3+M4	8.97	48	5390	5378	-0.99941883
5	M2+M3+M4	8.8	48	5390	5378	-0.99941883
5	M1+M2+M4	9	48	5390	5378	-0.99941883
5	M3+M4+M6	16.62	48	5390	5378	-0.99941883
6	M1+M2+M3	7.48	48	5390	5378	-0.99941883
6	M2+M3+M6	14.74	48	5390	5378	-0.99941883
7	M1+M2+M3+M4	6.74	48	5390	5378	-0.99941883
7	M2+M3+M4+M6	12.46	48	5390	5378	-0.99941883

Tabella 6.2: UCI Adult6 Parallel Tests

limitato alla frazione di tempo in cui tale miglioramento si applica.

É possibile notare che ogni volta che viene introdotta la macchina a singolo processore a 64 bit nel cluster, ovvero quando si passa da 2 a 3, da 4 a 5 e da 6 a 7 processori, non vi sono miglioramenti allo speedup rispetto alla situazione in cui vengono introdotte macchine biprocessore, infatti si ha una modifica della retta dello speedup del cluster che si allontana dalla retta lineare teorica, quindi la deviazione aumenta.

In tabella 6.4 viene visualizzata l'efficienza che si ottiene aumentando i processori. Nel nostro caso, con macchine disuguali dal punto di vista prestazionale, si ha che

WEB6a DATASET						
Proc	Macchine	Time(s)	It	SV	BSV	b
2	M1	5.78	9	1073	1025	0.99856238
2	M2	5.79	9	1073	1025	0.99856238
2	M3	5.79	9	1073	1025	0.99856238
2	M6	13.26	9	1073	1025	0.99856238
3	M1+M4	4.63	9	1073	1025	0.99856238
3	M2+M4	4.64	9	1073	1025	0.99856238
3	M3+M4	4.22	9	1073	1025	0.99856238
3	M4+M6	8.90	9	1073	1025	0.99856238
4	M1+M2	3.13	9	1073	1025	0.99856238
4	M1+M3	3.18	9	1073	1025	0.99856238
4	M3+M6	6.83	9	1073	1025	0.99856238
4	M2+M3	3.05	9	1073	1025	0.99856238
5	M1+M3+M4	2.66	9	1073	1025	0.99856238
5	M2+M3+M4	2.57	9	1073	1025	0.99856238
5	M1+M2+M4	2.66	9	1073	1025	0.99856238
5	M3+M4+M6	5.58	9	1073	1025	0.99856238
6	M1+M2+M3	2.28	9	1073	1025	0.99856238
6	M2+M3+M6	4.67	9	1073	1025	0.99856238
7	M1+M2+M3+M4	2.08	9	1073	1025	0.99856238
7	M2+M3+M4+M6	4.05	9	1073	1025	0.99856238

Tabella 6.3: WEB6a Parallel Tests

l'efficienza decresce all'aumentare dei processori.

Questo é dovuto al fatto che aumentano le comunicazioni, cioè aumentano le matrici che devono essere inviate ai vari processori per la soluzione dei sottoproblemi QP e per l'aggiornamento del gradiente, unici passi che vengono distribuiti nello schema di PGPDT. Tali comunicazioni intermedie sono però necessarie nel processo di addestramento parallelo di SVM secondo lo schema descritto, come in generale anche nella maggioranza delle applicazioni scientifiche reali.

In tabella 6.2 sono contenuti i risultati dei test di PGPDT applicato al dataset di immagini mediche chiamato MEDICAL e utilizzando gli stessi altri parametri dei

#CPU	MNIST		UCI Adult6		WEB6a	
	S	Eff	S	Eff	S	Eff
2	1.4058	0.702	1.4613	0.731	1.4784	0.739
3	2.0359	0.679	2.1109	0.704	1.8448	0.615
4	2.6622	0.666	2.7028	0.676	2.6918	0.673
5	3.0860	0.617	3.2133	0.643	3.2180	0.644
6	3.6921	0.615	3.8663	0.644	3.7544	0.626
7	4.0554	0.579	4.2908	0.613	4.1154	0.588

Tabella 6.4: Speedup ed efficienza

test precedenti.

Tali test riguardano le prestazioni del tool al variare di  $q$  (dimensione dei sottoproblemi QP) ed  $n$  (numero massimo di variabili entranti nel working set ad ogni iterazione). Nel primo caso, mantenendo costante il numero di variabili entranti ad ogni iterazione di decomposizione  $n = 300$ , si ha che aumentando la dimensione dei sottoproblemi QP si hanno ovviamente meno iterazioni, il tempo di training dell'SVM raggiunge valori ottimi quando il valore di  $q$  é circa tre volte il valore di  $n$ . Lo stesso comportamento lo si nota nel secondo caso quando  $q$  viene mantenuto costante. Si può affermare che buoni risultati in termini di tempo possono essere ottenuti scegliendo  $n = \lfloor q/3 \rfloor$ .

La scelta di  $q$  troppo piccolo comporta più iterazioni e di conseguenza i costi per operazioni non distribuite diventano maggiori. Tali costi riguardano la selezione del working set e la determinazione del criterio di terminazione.

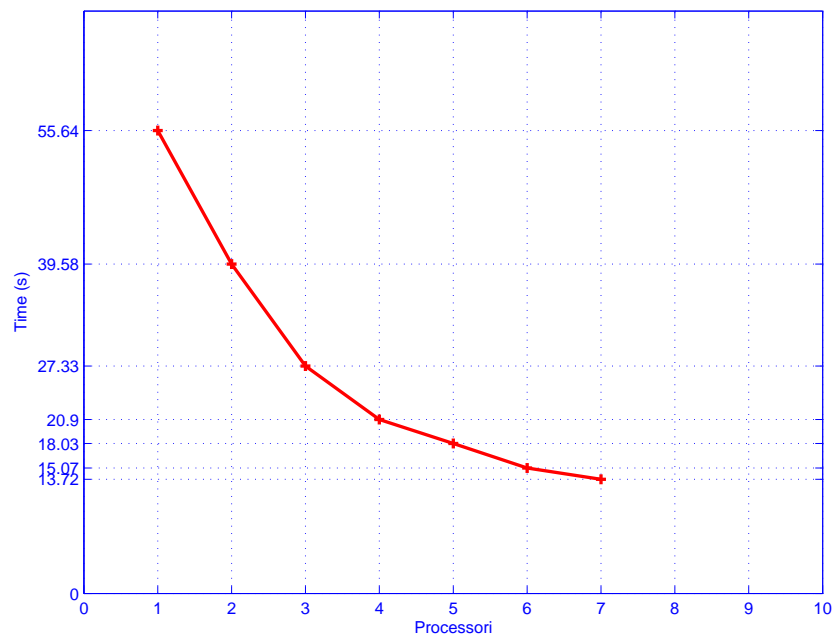


Figura 6.1: PGPDT Scaling: test sul dataset MNIST

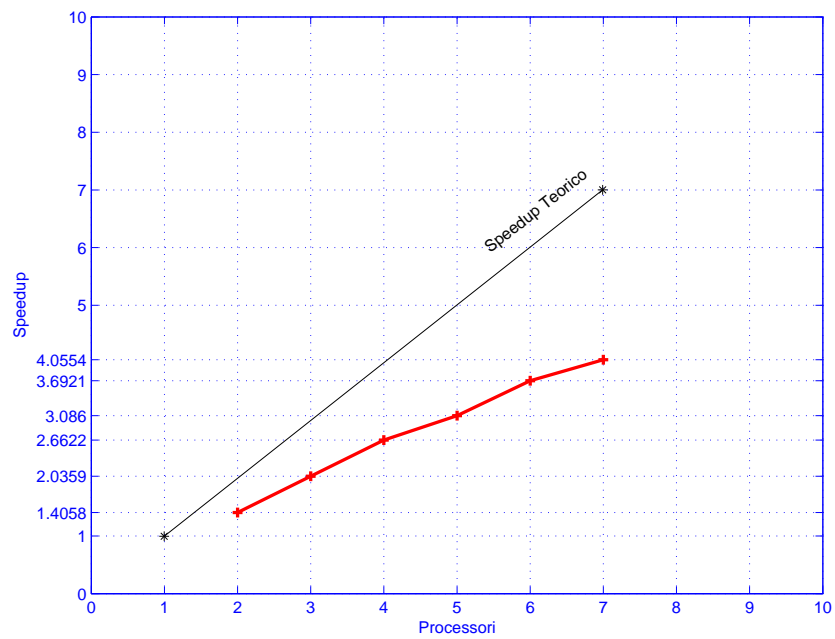


Figura 6.2: Cluster Speedup: test sul dataset MNIST



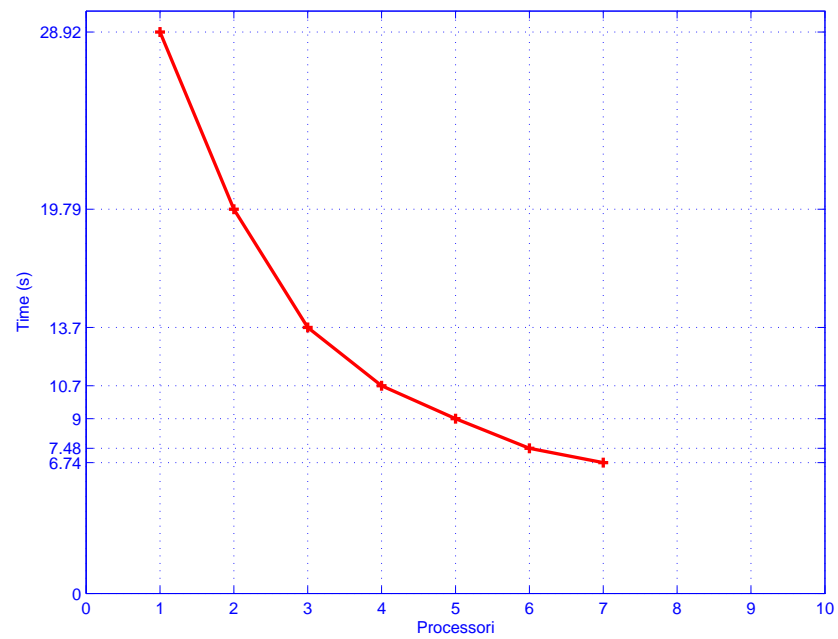


Figura 6.3: PGPDT Scaling: test sul dataset UCI Adult6

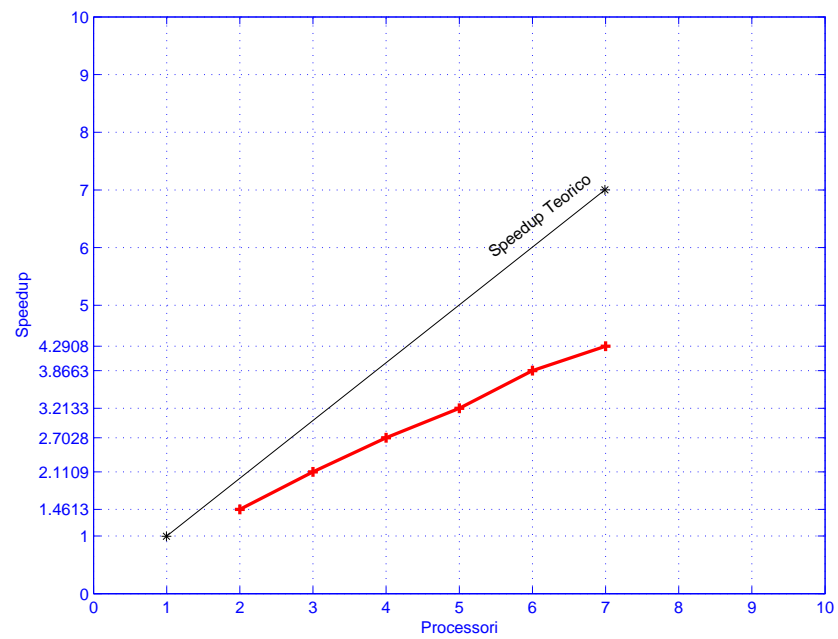


Figura 6.4: Cluster Speedup: test sul dataset UCI Adult6

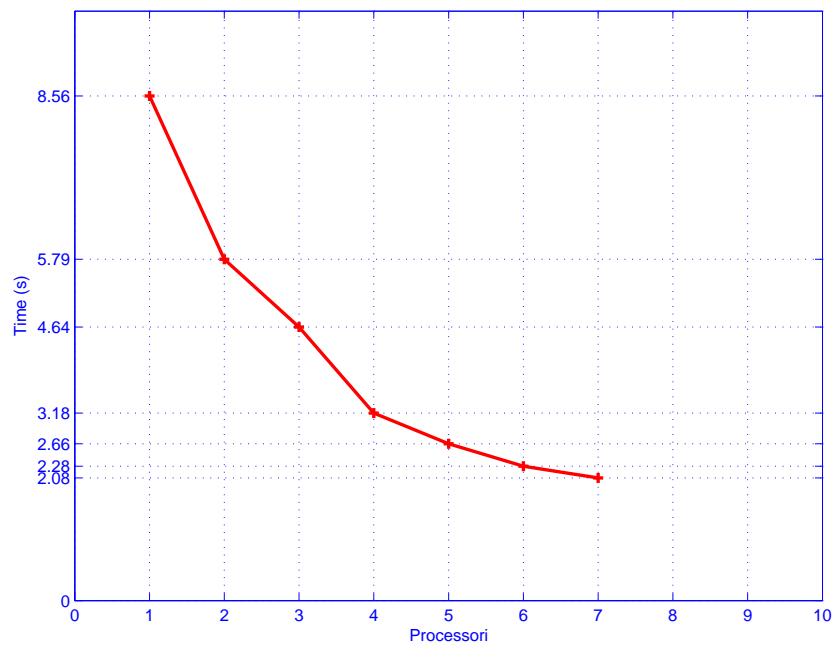


Figura 6.5: PGPDT Scaling: test sul dataset WEB6a

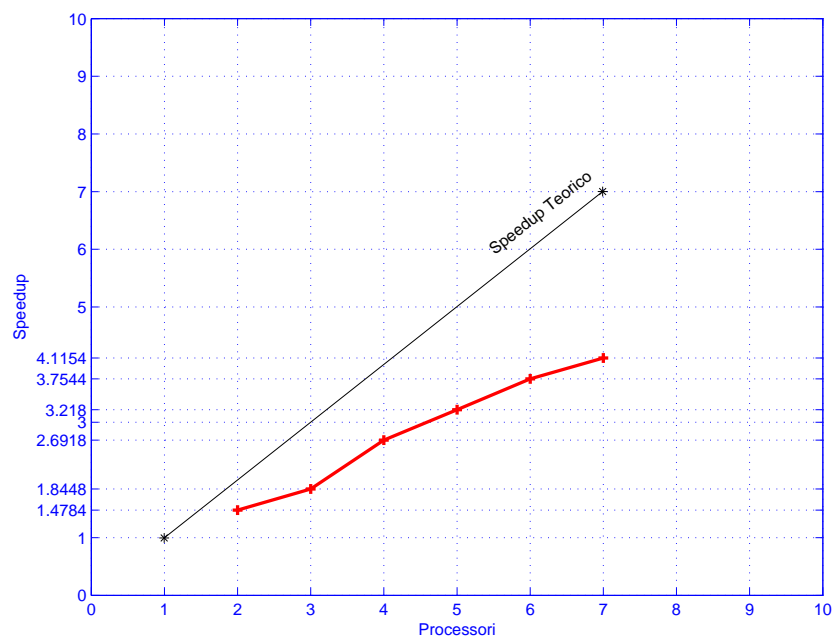


Figura 6.6: Cluster Speedup: test sul dataset WEB6a

<b>q</b>	<b>Time (s)</b>	<b>It</b>	<b>SV</b>	<b>n</b>	<b>n</b>	<b>Time (s)</b>	<b>It</b>	<b>SV</b>	<b>q</b>
300	61.67	12	2376	300	2	68.44	1094	2368	400
400	60.22	10	2376	300	20	58.43	111	2368	400
500	59.13	10	2376	300	80	57.79	32	2374	400
700	59.01	9	2375	300	130	57.66	20	2375	400
900	58.43	9	2371	300	180	57.66	15	2375	400
1200	65.17	8	2376	300	200	57.51	14	2376	400
1600	75.95	8	2375	300	280	56.78	10	2371	400
2000	83.17	7	2371	300	310	57.92	10	2376	400
2500	91.17	5	2371	300	360	59.71	10	2375	400

Tabella 6.5: Comportamento di PGPDT sul dataset MEDICAL al variare di q ed n

### 6.3 Ottimizzazione: il caching

Sostanziali miglioramenti prestazionali si possono ottenere sfruttando la quantità di memoria disponibile nei vari processori. PGPDT infatti permette di utilizzare un'area di memorizzazione locale ad ogni processore per le valutazioni del kernel, al fine di mantenere in memoria alcuni prodotti senza doverli ricalcolare: tale strategia di ottimizzazione viene chiamata *caching*.

L'utilizzo di questa strategia, riguarda in particolare l'aggiornamento del gradiente, che é il compito piú costoso in una iterazione di decomposizione. Quando la cache é piena, le colonne correnti rimpiazzano quelle meno utilizzate.

I miglioramenti di performance implicati dalla strategia di *caching* dipendono quindi dall'area di *caching* utilizzata. L'ammontare di memoria disponibile nei moderni sistemi di calcolo diventa quindi una risorsa importante per la strategia di decomposizione.

Agendo sulla dimensione della cache, é possibile ottenere ottimi risultati nel tempo di addestramento. In base al problema considerato si può determinare empiricamente la migliore dimensione per la cache.

La tabella 6.7 mostra le diverse prove effettuate con la macchina M3, con diverse dimensioni della cache ed utilizzando il dataset MNIST. Si può notare che la presenza o l'assenza della cache influisce notevolmente sul tempo di addestramento. Infatti solo il passaggio da una dimensione di default pari a 5 MB alla dimensione 10 MB

Dataset	Time (s)	SV	BSV	b	It
MNIST	9.17	1589	39	6.4402556	52
UCI Adult6	3.64	5390	5378	-0.9994183	48
WEB6a	1.28	1073	1025	0.99856238	9
MEDICAL	32.78	2375	2362	0.84925798	19

Tabella 6.6: Miglioramenti Prestazionali utilizzando il caching

comporta un risparmio di 30 secondi.

Come in tutte le strategie che comportano l'utilizzo della cache, anche in questo caso si può vedere che l'aumento della dimensione oltre una certa soglia non comporta alcun beneficio aggiuntivo. Risultati interessanti sono stati ottenuti impiegando le

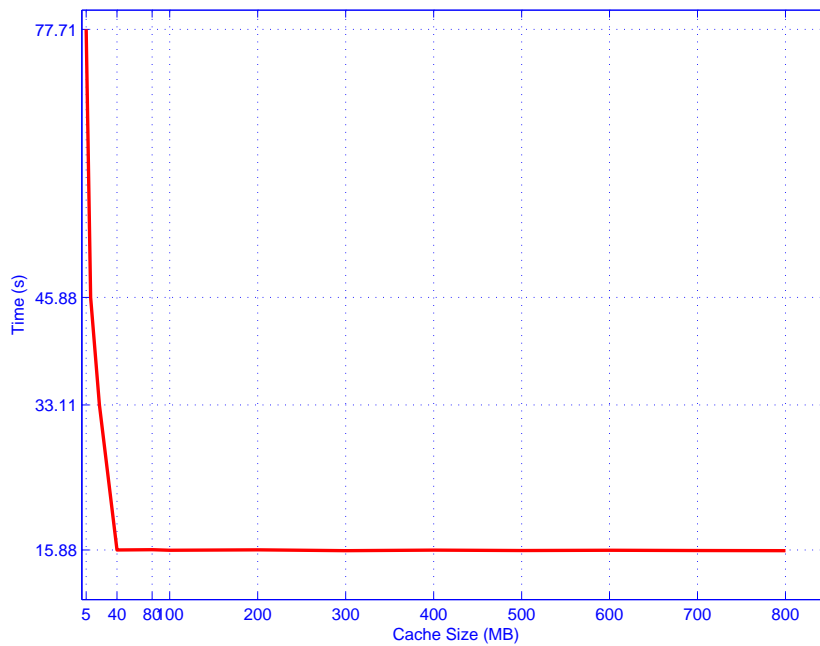


Figura 6.7: Prestazioni di PGPDT al variare della dimensione della cache sul dataset MNIST

due macchine più veloci del cluster, M3 ed M5, dove si ricorda che M3 utilizza un singolo processore con tecnologia HT mentre M5 utilizza un Dual Opteron.

Utilizzando questi multiprocessori e sfruttando la strategia di caching con dimensione della cache pari a 500 MB, PGPDT è stato testato sui quattro dataset portando i risultati mostrati in tabella 6.6.

Come si nota dalla tabella, questa strategia di ottimizzazione ha permesso di

ridurre i tempi di addestramento dell'SVM di circa la metà.

# Conclusioni e Sviluppi Futuri

*Sono state analizzate le prestazioni di un sistema di addestramento parallelo di Support Vector Machine per il problema della classificazione basato sullo schema di decomposizione di  $SVM^{light}$ . L'ambiente di test consiste in un cluster eterogeneo contenente macchine a 32 e 64 bit. Tale sistema é stato testato su vari benchmark ben noti in letteratura ed in seguito su un dataset di immagini mediche.*

*L'impiego di un cluster eterogeneo riduce le prestazioni in termini di tempo di addestramento a causa dell' "adeguamento" del tool alle macchine piú lente.*

*Dai grafici dello speedup si é potuto notare che aggiungendo processori su una stessa macchina lo speedup migliora perché la retta aumenta verso la retta teorica mentre all'aggiunta di macchine non usate già nel cluster la retta si allontana.*

*All'aumentare del numero dei processori aumenta anche la quantità di dati che deve essere trasmessa in generale e lo si vede dal fatto che la retta si allontana da quella teorica.*

*L'introduzione di macchine a 64 bit, non porta miglioramenti per quanto riguarda il tempo di addestramento dell'SVM in quanto il lavoro avviene per la maggior parte con macchine che operano con registri a 32 a bit.*

*É stata valutata una tecnica di ottimizzazione che mette a disposizione PGPDt per ridurre le valutazioni del kernel chiamata caching. I miglioramenti di performance implicati dalla strategia di caching dipendono dall'area di caching utilizzata e dal problema considerato. L'ammontare di memoria disponibile nei moderni sistemi di calcolo diventa quindi una risorsa importante per la strategia di decomposizione. Lo sfruttamento della tecnica di caching ha permesso di migliorare notevolmente le prestazioni di addestramento riducendo di oltre la metà il tempo necessario.*

*Dal punto di vista dell'accuratezza, risultati sperimentali hanno dimostrato che il tool si comporta in maniera ottimale, senza perdite, per quanto riguarda il numero*

*di Support Vector, il bias e il numero di Bound Support Vector.*

*Miglioramenti consistenti si possono ottenere attraverso:*

- *Ottimizzazione del codice*
- *Impiego del Multithreading nel solver QP*
- *Impiego di librerie specializzate come ATLAS*
- *Ottimizzazioni per specifiche architetture*

# Appendice A

## Test Environment

Vengono qui riportati i dati tecnici delle macchine utilizzate per i test in questa tesi.

### Nodo M1

Processore	Dual AMD Athlon MP 1800+
Frequenza	1546.439 Mhz
Cache Size	256 KB
RAM	1024 MB
Scheda di Rete	2x Gigabit Ethernet
Sistema Operativo	Linux Gentoo 2005.1
Kernel	2.6.11.12

### Nodo M2

Processore	Dual AMD Athlon MP 1800+
Frequenza	1546.527 Mhz
Cache Size	256 KB
RAM	1024 MB
Scheda di Rete	2x Gigabit Ethernet
Sistema Operativo	Linux Gentoo 2005.1
Kernel	2.6.11.12



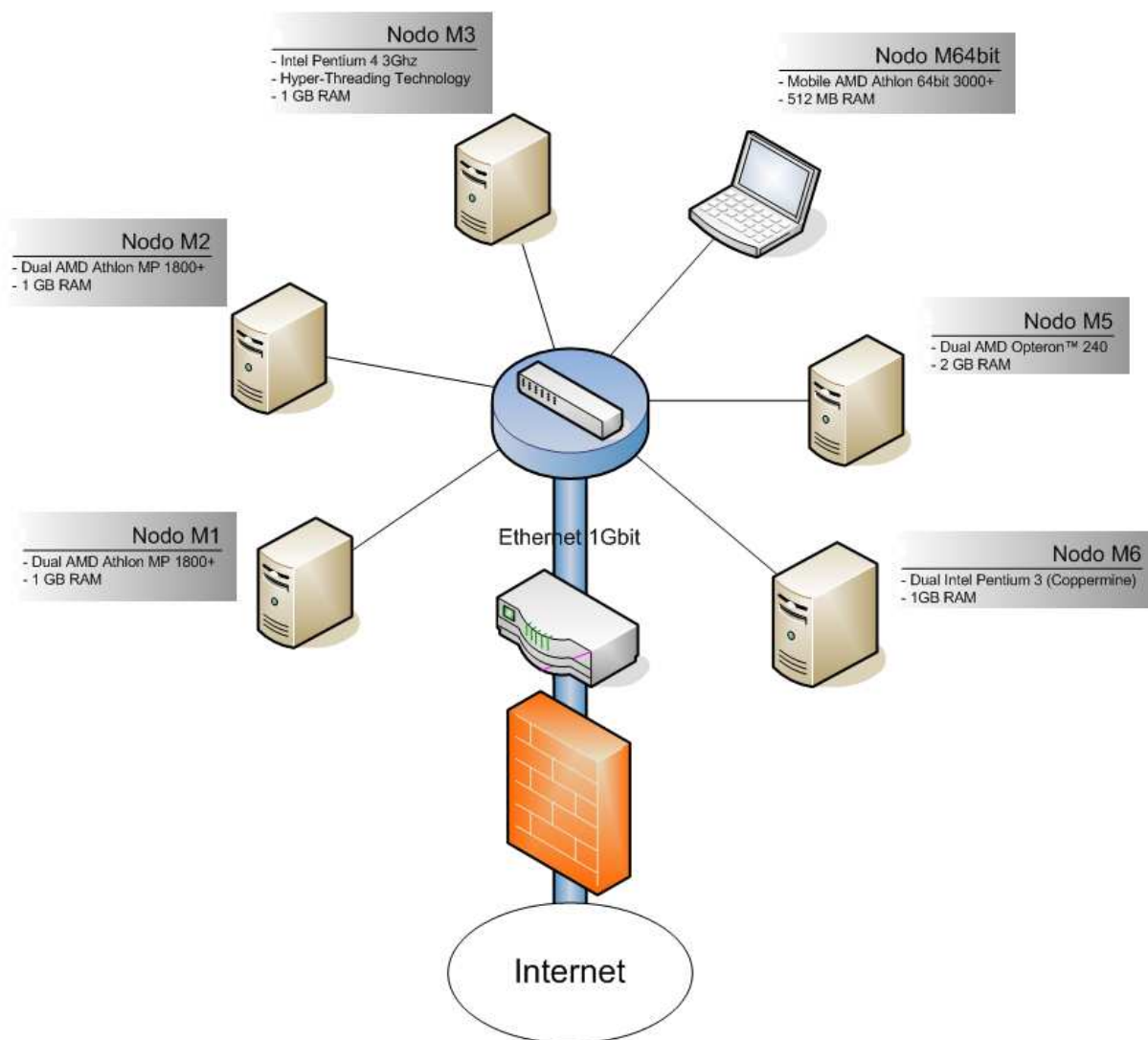


Figura A.1: Test Environment

---

### Nodo M3

Processore	Intel Pentium 4 con Hyper-Threading Technology
Frequenza	3012.97 Mhz
Cache Size	1024 KB
RAM	1024 MB
Scheda di Rete	3x Gigabit Ethernet
Sistema Operativo	Linux Gentoo 2005.1
Kernel	2.6.11.12

### Nodo M64bit

Processore	Mobile AMD Athlon 64bit 3000+
Frequenza	1795.4 Mhz
Cache Size	1024 KB
RAM	512 MB
Scheda di Rete	1x Gigabit Ethernet
Sistema Operativo	Linux Gentoo 2005.1
Kernel	2.6.13

### Nodo M5

Processore	AMD Opteron(tm) Processor 240
Frequenza	1393.646 Mhz
Cache Size	1024 KB
RAM	2048 MB
Scheda di Rete	2x Broadcom NetXtreme Gigabit Ethernet
Sistema Operativo	Linux Gentoo 2005.1
Kernel	2.6.10-ck2

**Nodo M6**

Processore	Intel Pentium 3 (Coppermine)
Frequenza	1000.285 Mhz
Cache Size	256 KB
RAM	1024 MB
Scheda di Rete	2x Gigabit Ethernet
Sistema Operativo	Linux Slackware
Kernel	2.6.11.11

# Bibliografia

- [1] A.J.Smola. *pr-LOQO Optimizer*,  
URL <http://www.kernel-machines.org/code/prloqo.tar.gz>,. 1997.
- [2] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. pages 141–148. IMA Journal of Numerical Analysis, 1988.
- [3] R. Campanini, D. Dongiovanni, E. Iampieri, N. Lanconelli, M. Masotti, G. Palermo, A. Riccardi, and M. Roffilli. A novel featureless approach to mass detection in digital mammograms based on support vector machines. *Phys. Med. Biol.*, 49:961–975, 2004.
- [4] C.Domeniconi and M.Jordan. *Discorsi sulle Reti Neurali e l'apprendimento*. Franco Angeli, AI\*IA, 2001.
- [5] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines (version 2.31).
- [6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.
- [7] M.J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computing C-21*, (9):948–960, Sept.1972.
- [8] G.Zanghirati and L.Zanni. A parallel solver for large quadratic programs in training support vector machines. In *Parallel Computing 29*, pages 535–551. 2003.
- [9] T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

- [10] L.Zanni, T.Serafini, and G.Zanghirati. A parallel software for training large-scale support vector machines on multiprocessor systems. In *Tech.Rep 71*. October 2005.
- [11] M. Masotti. *Optimal image representations for mass detection in digital mammography*. PhD thesis, University of Bologna, Department of Physics, 2005.
- [12] Medical Imaging Group. The MIG website: <http://www.bo.infn.it/mig>, 2005.
- [13] Bruce A. Murtagh and Michael A. Saunders. Minos 5.5 user’s guide. In *Technical Report*. Department of Operation Research, Stanford University, Stanford CA, 1998.
- [14] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.
- [15] M. Roffilli. *Advanced Machine Learning Techniques for Digital Mammography*. PhD thesis, University of Bologna, Department of Computer Science, to appear in 2006.
- [16] O. Schiaratura. Progettazione ed implementazione di un sistema di calcolo ibrido multithread-multiprocesso per hpc: applicazione all’imaging medico (with the supervision of R. Campanini and M. Roffilli) [in Italian]. Master’s thesis, University of Bologna, 2003.
- [17] O. Schiaratura. Progettazione ed implementazione di un sistema di calcolo ibrido multithread-multiprocesso per hpc: applicazione alla mammografia (with the supervision of R. Campanini and M. Roffilli) [in Italian]. Master’s thesis, University of Bologna, 2004.
- [18] Andrew S. Tanenbaum. *Structured Computer Organization*. Prentice Hall, 4th edition edition, 1999.
- [19] T.Serafini, G.Zanghirati, and L.Zanni. Gradient projection methods for large quadratic programs and applications in training support vector machines. In *Optim. Meth. Soft. 20*, pages 353–378. 2005.

- [20] T.Serafini and L.Zanni. On the working set selection in gradient projection-based decomposition techniques for support vector machines. In *Optim. Meth. Soft. 20*, pages 583–596. 2005.
- [21] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.
- [22] C. Zoffoli. Progettazione, realizzazione ed ottimizzazione di un cluster ibrido 32/64 bit per hpc altamente affidabile (with the supervision of R. Campanini and M. Roffilli) [in Italian]. Master’s thesis, University of Bologna, 2005.