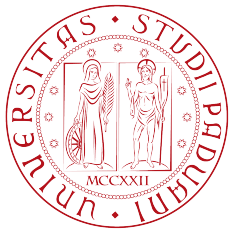

Ottimizzazione di algoritmi per la regressione logistica nell'ambito di Genome-Wide Association Studies

Laureando: Roberto Compostella

Relatore: Prof. Andrea Alberto Pietracaprina

Correlatori: Prof. Barbara Di Camillo,

Ing. Francesco Sambo, Ing. Francesco Silvestri

Corso di Laurea Magistrale in

Ingegneria Informatica

23 ottobre 2012

Anno Accademico 2011/2012

Indice

1 Introduzione **1**

2 Classificazione **3**

2.1 Data mining 3

2.1.1 Task del data mining 4

2.1.1.1 Task predittivi 4

2.1.1.2 Task descrittivi 5

2.2 Problema della classificazione 5

2.2.1 Problema computazionale 6

2.2.1.1 Input 6

2.2.1.2 Output 6

2.3 Approccio generale alla classificazione 6

2.4 Modelli di classificazione 7

2.4.1 Regressione logistica 8

2.4.1.1 Funzione di log-likelihood 9

2.4.1.2 Regressione logistica penalizzata 10

2.5 Metriche 11

2.5.1 K-fold cross-validation 12

2.6 Imputation 13

3 Genome-Wide Association Studies 15

3.1 Genome-Wide Association Study 16

3.2 Single-Nucleotide Polymorphisms 16

3.3 Legame tra analisi genetica multivariata e regressione logistica 18

3.4 Il problema in esame 19

3.5 GLMNET 21

3.5.1 Input 22

3.5.2 Output 23

3.5.3 Procedura *lognet* 23

3.5.3.1 chkvars 24

3.5.3.2 lstandard1 24

3.5.3.3 lognet2n 24

4 Ottimizzazione di GLMNET 29

4.1 Piattaforma di calcolo 29

4.2 Traduzione del pacchetto nel linguaggio C++ 30

4.3 Descrizione dei dati usati 30

4.4 Misura delle prestazioni: il profiling 31

4.4.1 Misurare le prestazioni 31

4.4.2 Profiling di un programma 31

4.4.3 Verifica delle prestazioni 32

4.4.4 Tools in C/C++ 32

4.4.4.1 clock() 32
4.4.4.2 Valgrind 33
4.4.5 Pianificazione del profiling 34
4.4.5.1 Definizione set di esperimenti 34
4.4.5.2 Risultati set di esperimenti 37
4.4.5.3 Conclusioni 47
4.5 Scelta del parametro α 48
4.5.1 Diabete di tipo 1 49
4.5.2 Diabete di tipo 2 51
4.6 Parallelizzare il codice 53
4.6.1 MPI 53
4.6.1.1 Cos'è 53
4.6.1.2 Istruzioni principali 54
4.6.2 Parallelizzazione della k-fold cross-validation 55
4.6.2.1 Tempi di esecuzione 57
4.6.3 Parallelizzazione della costruzione del modello finale . 58
4.6.3.1 La normalizzazione della matrice X 58
4.6.3.2 Il collo di bottiglia 59
4.6.3.3 Tempi di esecuzione 60
4.7 Risultati finali 61
4.7.1 Tempi complessivi 62
4.7.2 Tempi di comunicazione e attesa 63
4.8 Accuracy con iterazioni su λ ridotte 65

5 Conclusioni **67**

Capitolo 1

Introduzione

Presso l'Università degli Studi di Padova si sta volgendo un progetto di ricerca volto a migliorare computazionalmente un algoritmo di data mining usato nei Genome-Wide Association Studies. I GWAS sono studi della variabilità genetica attraverso l'intero genoma umano progettati per identificare un'associazione genetica con tratti visibili o la presenza o l'assenza di una malattia.

È stato scelto questa tipologia di studio perché si vuole effettuare un'analisi multivariata del problema dato che alcuni SNPs sono correlati fra di loro e, grazie ai recenti sviluppi in termini di efficienza computazionale, si può analizzare simultaneamente p SNPs di n soggetti, dove $p \gg n$, in tempi considerevoli. Uno SNP (Single Nucleotide Polymorphism) è una variazione di una sequenza di DNA che si verifica quando un nucleotide nel genoma si differenzia tra i membri di una stessa specie.

L'approccio base nei GWAS è il metodo caso-controllo, in cui si dividono gli individui in soggetti malati e sani e l'obiettivo è quello di trovare la funzione $Y = f(X, \beta)$ che meglio descrivere una condizione di malattia, dove X è una sequenza di SNPs, β è un vettore di pesi e Y indica se il soggetto è malato o sano.

L'algoritmo di data mining in questione è la regressione logistica penalizzata che fa parte dei problemi di classificazione. La regressione logistica penalizzata viene quindi qui utilizzata per calcolare il vettore β che determina il miglior insieme di SNPs che più si correlano a una certa malattia. Il motivo per cui ci siamo occupati di questo progetto è per scoprire se alcune malattie possono essere spiegate con un insieme di SNPs. Data però la grande quantità di SNPs in nostro possesso è necessario avere un algoritmo più efficiente.

Lo scopo del progetto è quello di ottimizzare in termini di efficienza il pacchetto GLMNET che presenta al suo interno algoritmi di regressione logistica penalizzata. Si vuole quindi identificare il collo di bottiglia dell'algoritmo per ridurlo o, se possibile, eliminarlo in modo da rendere il programma più efficiente.

Più precisamente è stato analizzato il codice e testato sul diabete di tipo 1 e di tipo 2. Una volta trovato il collo di bottiglia dell'algoritmo è stato studiato un modo per ridurlo e quindi per diminuire sensibilmente il tempo di calcolo complessivo del programma. Sono state poi aggiunte ulteriori ottimizzazioni al codice, come la parallelizzazione della k-fold cross-validation e di altre parti del codice. I risultati finali ottenuti sono uno speed up di 12.27 per il diabete di tipo 1 e di 8.65 per il diabete di tipo 2.

La struttura dei capitoli successivi è la seguente:

- nel capitolo 2 si descrive il problema della classificazione in generale e nello specifico la regressione logistica penalizzata;
- nel capitolo 3 si descrivono i GWAS e il pacchetto GLMNET;
- nel capitolo 4 si descrive lavoro effettuato per identificare il collo di bottiglia e le varie ottimizzazione apportate all'algoritmo;
- nel capitolo 5 sono presenti le conclusioni e le considerazioni finali.

Capitolo 2

Classificazione

2.1 Data mining

Il data mining è definito come un processo di estrazione automatica di informazione interessante a partire da grandi quantità di dati. Anche se il concetto di informazione interessante va adattato da caso a caso, l'informazione che volgiamo estrarre dai dati è ritenuta interessante perché:

- non è conosciuta precedentemente;
- è potenzialmente utile;
- non è esplicitamente rappresentata nei dati.

Il data mining è uno step all'interno di un processo più ampio ovvero di knowledge discovery. Il processo di knowledge discovery parte da dati in forma grezza memorizzati in basi di dati o in vari formati all'interno di file, questi dati passano poi attraverso una prima fase di pre-processing in cui i dati vengono preparati per la successiva analisi. Dopo la fase di pre-processing i dati risultanti vengono dati in pasto al data mining e quindi al tipo di applicazione che deve elaborarli. Il risultato sarà l'informazione interessante però fornita nell'output di un programma. Questa informazione

va a sua volta ripulita e trasformata per essere poi effettivamente utilizzata attraverso una fase di post-processing. Al termine di tutto questo abbiamo effettivamente l'informazione che stavamo cercando.

Il data mining utilizza tecniche e strategie da diverse discipline: la statistica, per decidere se l'informazione estratta è interessante oppure no, le basi di dati, poiché le applicazioni di data mining devono interagire con grandi basi di dati, il machine learning, che è in realtà una disciplina progenitrice del data mining, e l'high performance computing, dato che si tratta di elaborare grandi quantità di dati sono necessari sistemi distribuiti o paralleli.

2.1.1 Task del data mining

Le tipologie di problemi, o task, del data mining si dividono in predittivi e descrittivi.

2.1.1.1 Task predittivi

I task predittivi si dividono in classificazione, che si occupa della scoperta di un modello che mappa oggetti in classi predeterminate e il modello è ottenuto a partire da un training set di oggetti già classificati, e regressione, che è un caso particolare di classificazione in cui cerchiamo una funzione che mappa oggetti rappresentati da variabili di natura numerica e deve mappare questi valori in valori numerici, è sempre un task di classificazione ma per questo motivo è chiamata regression.

Le applicazione per problemi predittivi sono per esempio per classificare i clienti di aziende, per fare previsioni finanziarie, per la prevenzione, diagnosi e trattamento di malattie in ambito medico e biologico, per la classificazione di documenti.

2.1.1.2 Task descrittivi

I task descrittivi invece si dividono in association analysis, in cui si fanno ricerche di correlazioni all'interno dei dati e in particolare si possono cercare pattern che ricorrono frequentemente all'interno dei dati e regole associative, cluster analysis, dove si tratta di cercare di raggruppare oggetti in cluster, cioè come delle classi massimizzando la somiglianza tra oggetti nello stesso cluster e minimizzando la somiglianza tra oggetti in cluster diversi, e anomaly detection, dove c'è la ricerca di dati che differiscono in modo significativo dai dati tipici.

Le applicazioni per problemi descrittivi sono per esempio la market basket analysis, che va ad analizzare il comportamento di clienti che comprano certi prodotti, la ricerca di motif in sequenze di DNA o in reti di proteine, per decidere la disposizione degli oggetti in un negozio, per identificare di gruppi di clienti, per scoprire di frodi o crimini o intrusioni in reti di computer.

2.2 Problema della classificazione

La classificazione è il problema di individuare a quale classe appartiene un nuovo record sulla base di un training set di dati contenente dei record la cui classe è nota a priori. A differenza di altre tecniche la classificazione richiede la conoscenza a priori dei dati ed è una tecnica di apprendimento supervisionato:

- apprendimento perché il problema richiede di imparare il modello guardando il training set;
- supervisionato perché per ogni record presente nel training set si ha già la relativa classe.

Nella classificazione disponiamo quindi di un insieme di record chiamato *training set* contenente un attributo speciale che è l'etichetta di classe. L'e-

tichetta di classe rappresenta la categoria a cui quel record appartiene. A partire da questo insieme costruiamo un *modello* che ci permetta di assegnare queste classi a nuovi record sprovvisti dell'etichetta di classe con la massima accuracy. Vogliamo quindi imparare una funzione che, dati i valori degli attributi di un nuovo record, mi dica qual è la classe più adatta; cioè che mappa il prodotto cartesiano dei domini attributi nel dominio della classe.

2.2.1 Problema computazionale

2.2.1.1 Input

In input si ha un training set X formato da n record, chiamati $x_1, x_2, \cdots, x_n$, dove ogni record x_i è composto da p attributi, che sono chiamati features e si indicano con $x_i(A_1), x_i(A_2), \cdots, x_i(A_p)$, al quale viene associata una classe, y_i. Per ciascun attributo e per la classe ci sono domini di riferimento: l'attributo A_j appartiene ad un dominio D_j e la classe y_i ha valori su un certo dominio Γ, che è un dominio discreto e solitamente finito.

2.2.1.2 Output

In output si avrà un modello che mappa accuratamente gli oggetti nelle classi.

2.3 Approccio generale alla classificazione

Le prime cose da fare sono scegliere un particolare algoritmo di learning, scegliere il training set e determinare il modello. Una volta che abbiamo il modello vogliamo verificare se il modello è buono o no e quindi si va in una fase di validazione in cui si applica il modello su validation set di record già classificati.

Quando abbiamo un problema di classificazione si cerca di avere una famiglia di record già classificati abbastanza ampia e abbastanza distintiva della realtà, quindi si devono raggruppare i record per descrivere e testare l'efficacia e dividere la famiglia in due gruppi:

- un gruppo lo uso per trovare il modello;
- l'altro gruppo lo uso per vedere se il modello è efficace per classificare, cioè lo applico a tutti i record del validation set e vedo se la classe restituita dalla funzione corrisponde alla classe effettiva del record.

Normalmente si suddivide il dataset in $\frac{2}{3}$ per trovare il modello (training set) e il restare $\frac{1}{3}$ per la validazione (test set).

2.4 Modelli di classificazione

Per trovare il modello di classificazione si hanno a disposizione diverse tecniche. Le più importanti sono le seguenti.

Alberi decisionali Per un training set T, un albero di decisione è un albero tale che ogni nodo interno è associato ad un test su un attributo e gli archi verso i suoi figli sono etichettati con i risultati distinti del test e ogni foglia è associata ad un sottoinsieme $T_u \subseteq T$ tale che $u_1 \neq u_2,\ T_{u_1} \cap T_{u_2} = \varnothing,\ \cup_{foglie} T_u = T$ ed è etichettata con la classe di maggioranza dei record di T_u.

I valori degli attributi di ciascun record $r \in T_u$ sono coerenti con i risultati dei test incontrati nel cammino dalla radice alla foglia u

Regole di decisione Un modello R è insieme di regole ordinate $r_1, r_2, \ldots, r_k$, del tipo "condizione -> classe", tali per ogni nuovo record vengono sottoposte in ordine le k regole e la prima che restituisce valore di verità vero alla

condizione presente in essa assegna la propria classe al record. Le regole di decisioni si possono vedere come generalizzazioni degli alberi decisionali.

Regressione logistica Questo modello verrà trattato nel dettaglio nella sezione 2.4.1. Brevemente si tratta di un modello applicato nei casi in cui la variabile dipendente y_i sia di tipo dicotomico mutualmente esclusive ed esaustive, come lo sono tutte le variabili che possono assumere esclusivamente due valori: vero o falso, maschio o femmina, sano o malato, ecc. La regressione logistica non assume un relazione lineare tra la variabile dipendente e quella indipendente.

Altre tecniche sono le reti neurali, le reti bayesiane e le support vector machines. Per approfondire vedere [20].

2.4.1 Regressione logistica

Dato che nella regressione logistica le variabili y_i possono assumere solo due valori, il modello da utilizzare è:

$$P(y_i = 0|x_i) = \frac{1}{1 + e^{-\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)}}$$

$$P(y_i = 1|x_i) = \frac{1}{1 + e^{+\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)}}$$

Vengono usate queste due funzioni in quanto a sinistra e a destra dell'asse x assumono velocemente i valori 0 e 1 rispettivamente e, a differenza del gradino, sono facilmente derivabili.

Per imparare il modello e quindi per calcolare il vettore β, formato da $p+1$ numeri reali, si utilizza la funzione di log-likelihood.

Una volta imparato il modello si potrà utilizzarlo su un nuovo record x^* del quale non si conosce la classe, in altre parole non si ha il suo valore di y^*. Per classificare x^* si calcoleranno le due probabilità $P(y^* = 0|x^*)$ e

$P(y^* = 1|x^*)$ utilizzando il vettore β trovato in precedenza. La probabilità maggiore assegnerà la classe y^* al record x^*.

2.4.1.1 Funzione di log-likelihood

Come spiegato in [15], assumendo che $p(x_i) = P(y_i = 1|x_i)$ sia la probabilità che $y_i = 1$ data l'osservazione i-esima, per stimare i valori del vettore β nella regressione logistica l'approccio migliore è quello di scegliere i valori che massimizzano la funzione di log-likelihood $\ell(\beta)$ che rappresenta la probabilità dei valori osservati y_i del training set condizionati ai corrispondenti valori x_i.

$$\max_{\beta \in \Re^{p+1}} \ell(\beta) = \max_{\beta \in \Re^{p+1}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot \ln\left(p(x_i)\right) + (1 - y_i) \cdot \ln\left(1 - p(x_i)\right) \right] \right\}$$

Svolgendo alcuni semplici passaggi si ottiene:

$$\ell(\beta) = \frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot \ln\left(p(x_i)\right) + (1 - y_i) \cdot \ln\left(1 - p(x_i)\right) \right] =$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left\{ y_i \cdot \left[\ln\left(p(x_i)\right) - \ln\left(1 - p(x_i)\right) \right] + \ln\left(1 - p(x_i)\right) \right\} =$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot \ln\left(\frac{p(x_i)}{1 - p(x_i)} \right) + \ln\left(1 - p(x_i)\right) \right] =$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot \ln\left(\frac{1 + e^{+\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)}}{1 + e^{-\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)}} \right) + \ln\left(\frac{1}{1 + e^{+\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)}} \right) \right] =$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot \ln\left(e^{+\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)} \right) - \ln\left(1 + e^{+\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)} \right) \right] =$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[y_i \left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \right) - \ln\left(1 + e^{\left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j\right)} \right) \right]$$

2.4.1.2 Regressione logistica penalizzata

Quando il problema è sovradimensionato, cioè quando $p \gg n$, alla funzione di log-likelihood viene sottratta una funzione di penalità per permettere che la soluzione sia fattibile, cioè per evitare che ci sia un numero esponenziale di soluzioni. La funzione di regressione logistica penalizzata diventa quindi:

$$\frac{1}{n}\sum_{i=1}^{n}\left[y_i\left(\beta_0+\sum_{j=1}^{p}x_{ij}\beta_j\right)-\ln\left(1+e^{\left(\beta_0+\sum_{j=1}^{p}x_{ij}\beta_j\right)}\right)\right]-\lambda P_\alpha\left(\beta\right).$$

Il parametro λ serve per attribuire più o meno peso alla funzione di penalità. Nel massimizzare la funzione di log-likelihood, oltre a modificare i valori del vettore β, si dovrà anche fare il tuning di questo parametro.

Penalità elastic-net $P_\alpha\left(\beta\right)$ rappresenta la funzione di penalità elastic-net ed è definita come

$$P_\alpha\left(\beta\right)=\left(1-\alpha\right)\frac{1}{2}\left\|\beta\right\|_{\ell_2}^2+\alpha\left\|\beta\right\|_{\ell_1}=\sum_{j=1}^{p}\left[\left(1-\alpha\right)\frac{1}{2}\beta_j^2+\alpha|\beta_j|\right]$$

dove $\left\|x\right\|_{\ell_1}=\sum_i|x_i|$ e $\left\|x\right\|_{\ell_2}=\sqrt{\sum_i x_i^2}$ sono rispettivamente la norma ℓ_1 e ℓ_2.

Questa funzione è un compromesso tra la funzione di penalizzazione lasso (mettendo $\alpha=1$), dove c'è il problema che se un gruppo di SNPs sono correlati tutti i β_j associati a tale gruppo tranne 1 vengono posti uguali a 0, e la funzione ridge-regression (mettendo $\alpha=0$), dove invece nessun β_j viene posto pari a 0.

Come si può notare β_0 non è presente nella funzione di penalità in quanto non è associato direttamente ad alcun vettore predittore x_i.

2.5 Metriche

Dopo avere trovato un modello bisogna avere valutare se è buono oppure no. Per farlo ci sono a disposizione diverse metriche, di tipo qualitativo e quantitativo.

Una prima metrica è sicuramente la complessità della costruzione del modello e della sua applicazione, questa è una metrica che può essere quantitativa o qualitativa. Vogliamo un modello che possa essere imparato, determinato, utilizzato efficientemente dal punto di vista computazionale ed efficacemente dal punto di vista della qualità. Un'altra metrica qualitativa è l'interpretabilità del modello, cioè un modello che mi fa capire come è arrivato a trovare la classe.

Tra le metriche quantitative abbiamo l'accuracy e l'error rate. L'accuracy stimata sul validation set è il rapporto tra i record classificati correttamente e tutti i record, cioè la frazione di record per cui il modello predice la classe giusta. L'error rate invece, che è il rapporto tra il numero di record del validation set classificati non correttamente e il numero di record del validation set ovvero error rate = 1 - accuracy.

Un'altra metrica quantitativa è l'MCC (Matthews Correlation Coefficient) che viene usato nella classificazione binaria. È un coefficiente di correlazione tra le classi osservate predette dal classificatore. L'MCC ha un valore compreso tra -1 e +1 inclusi: +1 rappresenta una predizione perfetta, 0 è simile ad una predizione random, -1 indica che le predizioni sono tutte sbagliate. La formula per calcolarlo è:

$$MCC = \frac{TP \times TN - TP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}},$$

dove TP, TN, FP e FN sono rispettivamente veri positivi, veri negativi, falsi positivi e falsi negativi. Questi valori si possono estrarre dalla confusion matrix che mette in relazione le classi predette con quelle osservate (tabella 2.1).

		Classi predette	
		$y = 1$	$y = 0$
Classi osservate	$y = 1$	TP	FN
	$y = 0$	FP	TN

Tabella 2.1: Confusion matrix

2.5.1 K-fold cross-validation

Per valutare se un modello è buono, cioè se il modello da una buona accuracy sul test set, viene spesso usata la tecnica della k-fold cross-validation.

Il metodo della k-fold cross-validation è una tecnica statistica utilizzabile in presenza di una buona numerosità del training set. Consistere nel partizione i record in k insiemi $E_1, E_2 \dots E_k$, con $k > 1$ disgiunti e costruire k modelli tali che per il modello M_i E_i sia il validation set e l'unione dei restanti $k-1$ insiemi sia il training set. I k modelli saranno poi combinati per produrre il modello finale. Normalmente si usa $k = 10$, ma in generale k è un parametro non fissato a priori.

Nella suddivisione dei k sottoinsiemi è importante che tutti i sottoinsiemi abbiano la stessa frazione di esempi delle varie classi in proporzione all'insieme iniziale.

Il vantaggio di questo metodo è che tutti i record vengono usati sia come nel training set che nel validation set. In questo modo si può evitare il fenomeno dell'overfitting (figura 2.5.1), dove si ha il modello che rappresenta molto bene il training set ma generalizza male e se viene applicato a record al di fuori del training set presenta una bassa accuracy. Questo si ha soprattutto nei casi in cui l'apprendimento del modello dura troppo a lungo o quando c'è uno scarso numero di record nel training set. Il modello quindi si adatta troppo al training set, ma non funziona bene se applicato a nuovi record. In presenza di overfitting si ha perciò un abbassamento dell'accuracy sul validation set anche se aumenta sul training set.

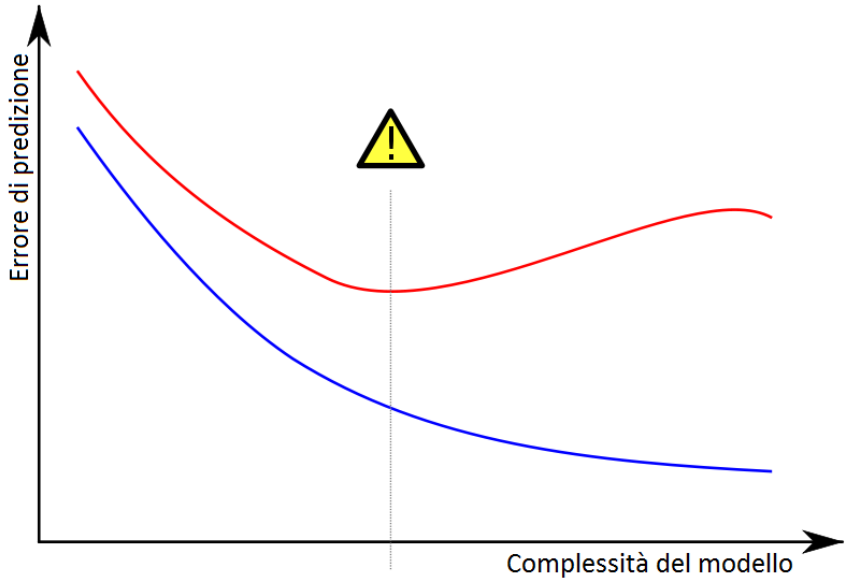

Figura 2.5.1: La curva blu mostra l'andamento dell'errore nel classificare i dati del training set, mentre la curva rossa mostra l'errore nel classificare i dati del validation set. Se l'errore di validazione aumenta mentre l'errore sui dati del training set diminuisce siamo in presenza di un possibile caso di overfitting.

Questo metodo viene anche utilizzato per effettuare il tuning di parametri se il modello ne presenta. Infatti combinando i k modelli si possono determinare i parametri ottimi.

2.6 Imputation

Se la matrice X presenta dei missing values è necessario effettuare l'imputation per attribuire un valore a quelli che non sono presenti. Ci sono diverse tecniche per effettuare l'imputation fra cui:

Mediana Si attribuisce come valore la media calcolata su tutti gli altri.

Moda Si attribuisce come valore quello che occorre più frequentemente in tutti gli altri.

K-nearest neighbor Si attribuisce come valore quello che occorre più frequentemente nei k record più somiglianti rispetto a quello in cui è presenti il missing value. Se $k = n$ si ha la moda.

Maximun likelihood estimation Si attribuisce come valore quello che massimizza la funzione di verosimiglianza definita come una funzione di probabilità condizionata $b \mapsto P(A|B = b)$.

Capitolo 3

Genome-Wide Association Studies

In genetica gli studi di associazione sono studi per l'analisi di marcatori genetici che identificano la predisposizione di individui di una popolazione in studio ad avere una determinata malattia. Un marcatore genetico è una differenza a livello genetico tra due o più individui. [17] descrive le principali tipologie di studi di associazione che sono:

- Candidate Marker, dove lo studio è incentrato su un singolo marcatore genetico che si ritiene essere implicato nell'insorgenza di una malattia.
- Candidate Gene, dove lo studio è incentrato sui marcatori presenti nella regione di un gene candidato.
- Fine Mapping, dove lo studio si concentra su un'intera regione candidata.
- Genome-Wide Association Study (GWAS), dove lo studio di concentra su tutto il genoma umano ed è quello che prenderemo in esame.

Viene scelto uno studio rispetto agli altri tenendo conto delle conoscenze cliniche-genetiche e del budget che si ha a disposizione, ma soprattutto in base a quello che si vuole ottenere dallo studio.

3.1 Genome-Wide Association Study

I GWAS sono studi della variabilità genetica attraverso l'intero genoma umano progettati per identificare un'associazione genetica con tratti visibili o la presenza o l'assenza di una malattia.

A differenza degli altri, è stato scelto questo studio perché si vuole effettuare un'analisi multivariata del problema dato che alcuni SNPs sono correlati fra di loro e, grazie ai recenti sviluppi in termini di efficienza computazionale, si può analizzare simultaneamente p SNPs di n soggetti, dove $p \gg n$, in tempi considerevoli.

L'approccio base nei GWAS è quello di analizzare il genoma umano attraverso il metodo caso-controllo. Questo metodo consiste nell'effettuare una classificazione tra un gruppo di individui sani (controllo) e un altro gruppo di individui affetti da una malattia specifica (caso) dove ciascun individuo di entrambi i gruppi può presentare una variazione genetica, evidenziata attraverso un marcatore genetico (Figura 3.1.1).

L'obiettivo dello studio di associazione caso-controllo è di identificare i marcatori genetici e le posizioni di tali marcatori che variano sistematicamente tra i due gruppi di individui in quanto le differenze di questi marcatori possono essere associate al rischio di una malattia.

3.2 Single-Nucleotide Polymorphisms

Il GWAS utilizza tecnologie automatizzate di genotipizzazione ad elevata resa per analizzare centinaia di migliaia di variazioni genetiche. Quando

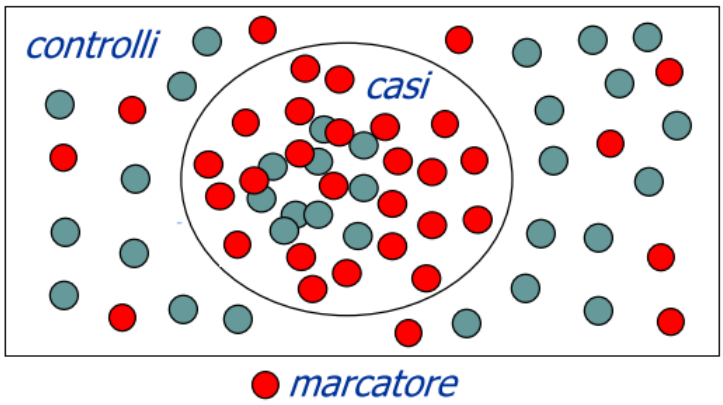

Figura 3.1.1: Esempio di una suddivisione caso-controllo evidenziando gli individui che presentano il marcatore genetico. I casi sono i soggetti all'interno del cerchio, mentre i controlli sono i restanti.

queste differenze o variazioni genetiche avvengono con una frequenza maggiore dell'1% della popolazione si parla di polimorfismi, in caso contrario di mutazione. Il più semplice polimorfismo è il Single-Nucleotide Polymorphism (SNP), che costituisce circa il 90% delle variazioni genetiche. Uno SNP è una variazione di una sequenza di DNA che si verifica quando un nucleotide nel genoma si differenzia tra i membri di una stessa specie, cioè quando un nucleotide è sostituito da un altro, come nell'esempio di figura 3.2.1. Quando un nucleotide assume forme diverse viene chiamato allele.

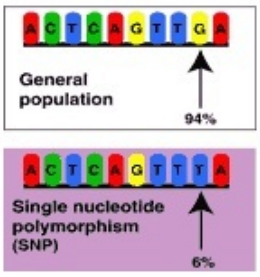

Figura 3.2.1: Esempio di SNP: la maggior parte della popolazione ha il nucleotide G mentre il restante 6% ha una sua variazione (polimorfismo) rappresentata nella figura dal nucleotide T.

Nello studio di associazione genetica si utilizzeranno quindi gli SNPs come variabili predittori del rischio di essere affetti da una malattia.

3.3 Legame tra analisi genetica multivariata e regressione logistica

Come si può vedere in [2, 21, 12] i metodi di regressione sono comunemente usati in analisi statistiche e il recente trasferimento alla sperimentazione a singolo marcatore in studi di associazione genetica è stato a causa del gran numero di variabili predittive da esaminare. Analizzando tutti marcatori insieme, cioè effettuando un'analisi multivariata, in un modello di regressione permette di:

- valutare l'impatto dei marcatori sugli altri marcatori, in quanto un debole effetto potrebbe diventare più visibile sommato ad altri effetti già trovati;
- rimuovere un falso segnale con l'inclusione di un segnale più forte a causa di un'associazione.

Tuttavia, quando il numero di marcatori è maggiore del numero di soggetti di prova o quando le variabili sono altamente correlate i metodi di regressione standard non posso essere usati a causa del sovradimensionamento del problema che genera un numero esponenziale di soluzioni. Per ovviare a tale inconveniente si utilizzano metodi di regressione penalizzata. Questi metodi operano riducendo il valore dei coefficienti, spingendo i coefficienti di marcatori con poco o nessun effetto apparente verso lo zero e riducendo i gradi di libertà del problema.

Nelle analisi di associazione genetica ci aspettiamo che solo alcuni marcatori abbiano un effetto reale. Quindi, attraverso l'uso della penalizzazione, troviamo il sottoinsieme di marcatori più associati con la malattia.

Un problema potenziale con i metodi di regressione penalizzata è che, in genere, una variabile entra nel modello solo se migliora in maniera significativa la previsione. Così, una variabile con un forte effetto marginale può

essere trascurata se altre variabili spiegano l'effetto. Tuttavia, si spera che la procedura di selezione dovrebbe selezionare le variabili che effettivamente spiegano meglio i dati.

Sfortunatamente, tutti i metodi di penalizzazione richiedono di specificare un parametro di penalizzazione. La scelta del parametro controlla il numero di variabili selezionate: maggiore è la penalizzazione, minore è il sottoinsieme selezionato. Il valore del parametro di penalizzazione deve essere scelto, ad esempio attraverso la cross-validation, per evitare la selezione di un valore sub-ottimale.

Quindi i problemi di classificazione dove il dominio della classe è binario, come il metodo caso-controllo, sono generalmente analizzati utilizzando la regressione logistica penalizzata.

3.4 Il problema in esame

Dato un insieme di SNPs espressi in forma di coppie di alleli AA, AB e BB, dove l'allele B è il più frequente nella popolazione, e codificati come variabili ternarie nel dominio $\{1, 2, 3\}$, dove 1=AA, 2=AB e 3=BB come si può vedere nell'esempio di figura 3.4.1, l'obiettivo è di identificare il miglior sottoinsieme di SNPs in grado di spiegare la condizione di malattia. Gli SNPs sono espressi come coppie di alleli perché i due alleli rappresentano rispettivamente il nucleotide nel cromosoma ereditato dalla madre e quello ereditato dal padre. Ovviamente i due alleli di ogni coppia devono assumere la stessa posizione nel cromosoma.

Si vuole un loro sottoinsieme perché, per determinare se un soggetto può essere affetto da una malattia o meno, non si può analizzare uno SNP alla volta dato che alcuni SNPs possono avere dei piccoli effetti marginali ma un grande effetto insieme ad altri in quanto non solo correlati tra di loro.

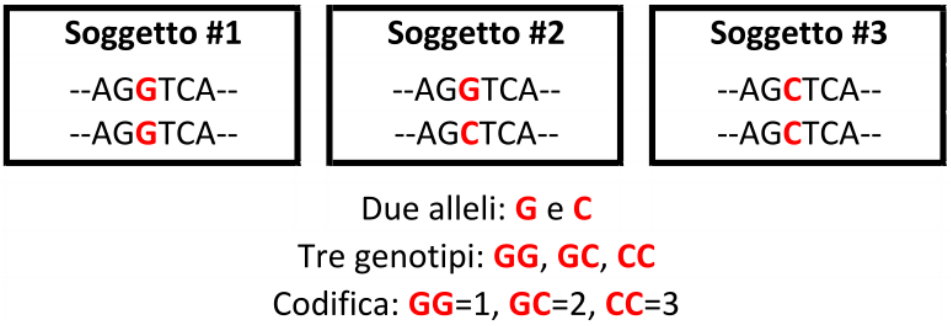

Figura 3.4.1: Esempio di codifica di uno SNP.

Quello di cui abbiamo bisogno è quindi una funzione di classificazione $Y = f(X, \beta)$ che, dati in input un vettore X contenente p SNPs di un soggetto in esame e un vettore colonna β di pesi specifici per una particolare malattia, dia in output un valore $Y \in \{1, 2\}$ (la classe), dove il valore 1 sta ad indicare che il soggetto è a rischio di avere quella malattia. Diciamo "a rischio" invece di "affetto sicuramente" perché si può incorrere in falsi positivi e in falsi negativi.

Nel nostro caso i dati di input sono quindi un insieme di n soggetti, dei quali si conosce già se sono malati o meno, che formano la matrice X, di dimensione $n \times p$ contenente n osservazioni di p SNPs, e il relativo vettore colonna Y , che nella i-esima riga avrà un 1, se la i-esima osservazione è di un soggetto malato, o un 2, se il soggetto è sano. Le righe x_i della matrice X sono chiamate vettori predittori, mentre gli elementi del vettore Y sono le variabili di risposta.

Per ogni colonna j gli elementi x_{ij} della matrice X fanno riferimento ad uno SNP, che assume la stessa posizione del DNA in ciascun soggetto, e possono perciò assumere solo i valori $1, 2, 3$, in base alla coppia di alleli che lo SNP presenta. Perciò $D_j = \{1, 2, 3\}$ $\forall j = 1, 2, \cdots, p$ e $\Gamma = \{1, 2\}$.

Come output invece si avrà il vettore colonna β dei pesi relativi alla malattia in oggetto che danno il minor numero di falsi positivi e di falsi negativi e che verrà poi utilizzato nella funzione di classificazione per predire se un soggetto

è a rischio di avere una determinata malattia sulla base dei suoi SNPs.

Tipicamente in questa tipologia di analisi il numero dei soggetti è dell'ordine delle migliaia, mentre il numero di SNPs presenti in ogni soggetto è dell'ordine dei milioni. Dato quindi che $p \gg n$ è molto difficile trovare il miglior sottoinsieme di SNPs che riesce a spiegare la malattia in quanto le soluzioni sono esponenziali. Questi problemi vengono definiti "small-n-large-p".

Per ottenere il vettore colonna β e quindi per imparare la funzione di classificazione $Y = f(X, \beta)$ a partire dal training set formato dalla matrice X e dal vettore colonna Y usiamo come algoritmo di learning la regressione logistica penalizzata. Dato che nella regressione logistica le variabili y_i assumono solo valori 0 o 1 ad un soggetto è sanno verrò attribuito il valore 0 al posto del valore 2.

3.5 GLMNET

L'algoritmo utilizzato per risolvere questo problema è presente nel pacchetto GLMNET[1] [8]. Il pacchetto ha licenza GPL-2 ed è quindi open-source. È un pacchetto scritto in linguaggio R con alla base il Fortran90 e contiene procedure efficienti ed efficaci per i modelli per regressione logistica penalizzata. I dati in input sono caricati a partire da file di testo.

Fra le diverse procedure presenti nel pacchetto GLMNET quella da utilizzare nel nostro caso è la "Symmetric binomial/multinomial logistic elastic net" (procedura *lognet*)

Listing 3.1: Chiamata alla procedura lognet con parametri di input/output

```
call lognet (parm, no, ni, nc, x, y, o, jd, vp, ne, nx,
    nlam, flmin, ulam, thr, isd, maxit, kopt, lmu, a0,
   ca, ia, nin, dev0, fdev, alm, nlp, jerr)
```

[1] http://cran.r-project.org/web/packages/glmnet/index.html

3.5.1 Input

- *parm*: il parametro α, con $0 < \alpha < 1$, che verrà utilizzato per la funzione di penalità elastic-net.
- *no*: il numero di osservazioni n.
- *ni*: il numero di variabili predittori p.
- *nc*: il numero di classi.
- *x*: la matrice $X \in \{1, 2, 3\}^{p \times n}$, contenente le n osservazioni di p SNPs.
- *y*: la variabile di risposta Y.
- *o*: gli off-set per ogni classe (non utilizzati nel nostro problema).
- *jd*: flag per l'eliminazione di alcune variabili predittori (non utilizzati nel nostro problema).
- *vp*: la penalità relativa per ogni variabile predittore (non utilizzati nel nostro problema).
- *ne*: il numero massimo di variabili permesse a entrare nel modello più grande (non utilizzato nel nostro problema).
- *nx*: il numero massimo di variabili permesse a entrare in tutti i modelli (non utilizzato nel nostro problema).
- *nlam*: il massimo numero di valori che λ può assumere.
- $flmin$: flag per controllare i valori di λ.
- *ulam*: vettore contenente i valori di λ (ignorato se $flmin < 1$).
- *thr*: soglia di convergenza per ogni valore di λ.
- *isd*: flag di standardizzazione.

- *maxit*: il numero massimo di iterazioni permesse sui β per tutti i valori di λ.
- *kopt*: flag per utilizzare l'algoritmo di Newton-Raphson o quello modificato.

3.5.2 Output

- *lmu*: numero effettivo dei valori λ (numero di soluzioni).
- $a0$: il valore β_0 per ogni soluzione.
- *ca*: il vettore β compresso escluso β_0 per ogni soluzione.
- *ia*: i puntatori per il vettore β compresso.
- *nin*: numero di coefficienti compressi per ogni soluzione.
- $dev0$: deviazione zero.
- $fdev$: frazione di deviazione spiegata da ciascuna soluzione.
- *alm*: il valore λ corrispondente alla soluzione.
- *nlp*: il numero di iterazioni sui β per tutti i valori di λ.
- *jerr*: flag di errore.

3.5.3 Procedura *lognet*

Questa procedura è la parte del pacchetto GLMNET che serve per risolvere il nostro problema. Consiste semplicemente nella chiamata di altre tre procedure dopo avere istanziato delle variabili.

Listing 3.2: Pseudocodice lognet()

```
1 subroutine lognet
2     istanzia e alloca nuove variabili
```

```
3       call chkvars
4       call lstandard1
5       call lognet2n
6       dealloca le variabili istanziate precedentemente
```

3.5.3.1 chkvars

Questa procedure controlla che per ogni SNP ci siano almeno 2 valori distinti tra tutte le osservazioni. In caso contrario si farà sempre a meno di analizzare quel SNP perché non potrà mai comportare una distinzione tra il gruppo di individui sani e quelli malati.

3.5.3.2 lstandard1

Questa procedura si occupa di normalizzare le variabili x_{ij} sottraendo la media e dividendo per la varianza.

3.5.3.3 lognet2n

Questa procedura ha il compito vero e proprio di calcolare le coppie β per ogni valore λ. Presenta un ciclo esterno su λ e dei cicli interni per il calcolo dei β utilizzando come algoritmo il Cyclic Coordinate Descent.

Listing 3.3: Pseudocodice lognet2n()

```
1  procedure lognet2n
2       istanzia e alloca nuove variabli
3       calcola il valore iniziale di beta(0) e di altri
            parametri
4       tutti gli altri valori di beta vengono posti uguale a 0
5       for ilm=1 to nlam do
6           determina il nuovo valore di lambda
7           salva i valori dei beta del ciclo predecente
8           for k=1 to ni do
```

```
9           /*questo ciclo rappresenta il Cyclic Coordinate
                Descent Algorithm che verrà spiegato nel
                dettaglio più avanti*/
10              aggiorna il valore di beta(k)
11          //endfor
12          aggiorna il valore di beta(0)
13          while (soglia superata)
14              for l=1 to nin do
15                  aggiorna il valore di beta(m(l))
16                  /*dentro il vettore m ci sono i beta che son
                        già stati modificati*/
17              //endfor
18              aggiorna beta(0)
19          //endwhile
20          calcola P(Y=0|X)
21          if (beta0 o almeno un elemento del vettore beta è
                stato significativamente modificato) o (nuovi
                beta possono entrare in soluzione)
22              then torna alla riga numero 7
23          salva tutti i valori trovati per questo valore di
                lambda
24          if (raggiunto il numero ne di beta entrati nel
                modello)
25              then esco dal ciclo su lambda
26      ///endfor
27      dealloca le variabili istanziate precedentemente
```

Calcolo valore di lambda λ viene inizialmente posto pari ad una costante λ_{max} e ad ogni iterazione decrescerà in scala logaritmica fino a $\lambda_{min} = \epsilon\lambda_{max}$.

Nella prima iterazione su λ si calcola β_0 in base al vettore y e si inizializza gli altri valori del vettore β a 0, mentre dalla seconda iterazione in avanti i valori di partenza del vettore β sono quelli trovati all'iterazione precedente.

Algoritmo Cyclic Coordinate Descent Per massimizzare la funzione di log-likelihood (vedi sezione 2.4.1.1) e cioè per trovare il vettore colonna β si utilizzerà l'algoritmo di Cyclic Coordinate Descent (CCD), che è l'algoritmo di Newton-Raphson ottimizzato per questo problema. L'ottimizzazione consiste nell'inserire una funzione soglia sotto la quale il valore di β_j viene posto pari a 0.

L'algoritmo CCD è l'approccio più naturale per risolvere problemi convessi con vincoli ℓ_1 e/o ℓ_2. Ogni iterazione è veloce, con una formula esplicita per ogni minimizzazione. Il metodo sfrutta anche la scarsità del modello, spendendo gran parte del suo tempo a valutare solo i prodotti interni per le variabili con coefficienti diversi da zero.

Con questo algoritmo si aggiorna il vettore β, ad ogni iterazione k verranno utilizzati i valori di b_{k-1} già modificati nelle iterazioni precedenti e si calcoleranno i nuovi valori nel seguente modo:

1. Si calcola $\beta_j^* = \sum_{i=1}^{n} x_{ij} r_i + xv_j \beta_j$
2. Si calcola il nuovo $\beta_j = \dfrac{S\left(\beta_j^*, \lambda\alpha\right)}{xv_j + \lambda(1-\alpha)}$,
   dove $S\left(\beta_j^*, \lambda\alpha\right) = \text{sign}\left(\beta_j^*\right)\left(|\beta_j^*| - \lambda\alpha\right)_+$ è la funzione di soglia che definisce quando i β_j sono pari a 0.
3. Se β_j è entrato in soluzione per la prima volta di inserisce j nel vettore m
4. Si calcola il nuovo $r_i = r_i - (\beta_j - b_{k-1})\, v_i x_{ij} \qquad \forall i = 1, 2, \cdots, n$
5. Si ripetono i punti dall'1 al 4 $\forall j = 1, 2, \cdots, p$

xv è un vettore contenente dei pesi da usare nella funzione soglia e v è un vettore che contiene le probabilità $P(y = 1|x)$ e $P(y = 0|x)$; entrambi si aggiornano in ogni iterazione di lambda.

Come si vede un β_j entrato in soluzione può in un iterazione successiva uscirne ritornando ad avere il valore 0, anche se resterà sempre nel vettore m. Inoltre la matrice X viene sempre passata per colonna. Per ottimizzare le letture sarebbe quindi necessario memorizzare la matrice X in column-major.

Al diminuire del valore di λ il numero di $\beta_j \neq 0$ aumenta dato che nell'utilizzo la funzione soglia viene prima sottratto un valore pari a $\lambda * \alpha$. Per ogni iterazione sul valore di λ l'algoritmo restituisce un vettore β. Solo in un secondo momento, ad esempio attraverso la k-fold cross-validation, si sceglierà quale sarà il vettore β da utilizzare per il problema in esame.

Capitolo 4

Ottimizzazione di GLMNET

4.1 Piattaforma di calcolo

Per identificare il collo di bottiglia e per apportare le successive ottimizzazioni è stato utilizzato come piattaforma di calcolo l'IBM Power 7 situato presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova.

La configurazione hardware di tale computer è composta da:

- 6 processori aventi ciascuno 8 core 4-way SMT @ 3.1 GHz;
- 640 GB di RAM;
- 10 TB di capacità totale dei dischi fissi (SAN esterno).

I sistemi operativi presenti sono SuSE Linux Enterprise 11 SP1, quello utilizzato, e AIX 6.1. I software disponibili sono i compilatori GNU e IBM, compilatori e librerie per lo sviluppo di codice parallelo, librerie scientifiche ottimizzate per i processori per Power 7, LoadLeveler Scheduler e altri software open-source come R e Python.

4.2 Traduzione del pacchetto nel linguaggio C++

A causa di problemi dovuti alla lettura della matrice di input X troppo grande le procedure originariamente scritte in R e fortran sono state tradotte e unificate tutte in un file C++.

Dato che il linguaggio C++, a differenza del fortran, è row-major è stato necessario invertire gli indici della matrice X per una questione di efficienza nella lettura dei valori.

4.3 Descrizione dei dati usati

GLMNET è stato testato sullo studio caso-controllo del WTCCC sul diabete di tipo 1 e di tipo 2 [5]: lo studio ha esaminato circa 2000 casi per ogni tipo di diabete e 3000 controlli sani. Ogni soggetto è stato genotipizzato sul Affymetrix GeneChip 500K Mapping Array Set.

Un piccolo numero di soggetti è stato escluso in accordo con la lista di campioni da escludere fornita dal WTCCC. Inoltre, è stato escluso un SNP se è sulla lista di esclusione SNP fornite dal WTCCC o ha un cluster plot povero come definito dal WTCCC.

Il set di dati risultante è costituito da 458376 SNPs, con 1963 casi e 2938 controlli per il diabete di tipo 1 e 1924 casi e 2938 controlli per il diabete di tipo 2.

Dato che entrambi i dataset presentavano al proprio interno dei missing values, per ogni SNP i valori mancanti sono stati sostituiti effettuando l'imputation utilizzando la moda dei valori degli altri soggetti.

4.4 Misura delle prestazioni: il profiling

Il profiling è un'analisi dei programmi che, fra l'altro, misura lo spazio di memoria utilizzato, la complessità temporale del programma, l'utilizzo di istruzioni particolari e la frequenza e durata delle funzioni. L'uso principale delle informazioni restituite dal profiling è per ottimizzare il programma.

4.4.1 Misurare le prestazioni

La misura e la stima delle prestazioni di un programma permette di verificare lo sfruttamento delle risorse hardware e software che si hanno a disposizione ed evidenziare eventuali problemi di prestazione e/o di velocità.

La misura delle prestazioni però ci fornisce indicazioni:

- di buono o cattivo funzionamento, ma non le cause;
- a livello di implementazione, ma non a livello di algoritmo.

4.4.2 Profiling di un programma

Il profiling di un programma ci permette fra l'altro di conoscere il tempo (assoluto e relativo) impiegato in ogni subroutine o blocco computazionale, individuare i blocchi computazionali più pesanti e i relativi colli di bottiglia, risalire a tutta la gerarchia delle chiamate, individuare dove si possono migliorare le prestazioni e individuare routines poco efficienti e modificarle.

Il profiling a livello di subroutine/funzioni in maniera limitatamente intrusiva descrive il flusso di esecuzione del codice restituendo prestazioni misurate di solito realistiche. Il profiling a livello di singolo statement eseguito invece è molto intrusivo e restituisce prestazioni misurate non indicative, ma permette di identificare costrutti eseguiti più del necessario.

4.4.3 Verifica delle prestazioni

Per effettuare la verifica delle prestazioni si deve:

- usare più casi test scegliendo diversi casi di riferimento;
- variare le dimensioni del problema per avere una stima di come cambia il peso relativo delle varie subroutines;
- assicurarsi che i test coprano correttamente l'uso di tutte le subroutine;
- fare attenzione al peso della parte di input/output nel test;
- possibilmente lavorare a processore dedicato.

4.4.4 Tools in C/C++

I tools per analizzare i programmi sono estremamente importanti per capirne il comportamento e per analizzare le sezioni critiche del codice. Spesso si usano questi strumenti per scoprire anche se l'algoritmo utilizza un buon branch prediction. Fra i tools utilizzabili in C/C++ ci sono clock() e valgrind.

4.4.4.1 clock()

Clock() è una funzione C/C++ che fornisce il tempo in clock del processore. Utile per il profiling di singoli blocchi come i cicli for. Per utilizzare questa funzione è però necessario includere la libreria *time.h*.

Listing 4.1: Esempio utilizzo clock()

```
1 #include <time.h>
2
3 clock_t time1, time2;
4 double elapsed_time;
5
6 int main() {
```

```
7       ...
8       time1 = clock();
9       for (j = 0; j < (nn); j++) {
10          ...
11      }
12      time2 = clock();
13      elapsed_time = (time2 - time1)/(double) CLOCKS_PER_SEC;
14      ...
15  }
```

4.4.4.2 Valgrind

Valgrind è un framework per la creazione di strumenti di analisi dinamici. Ci sono strumenti valgrind che possono automaticamente rilevare la gestione della memoria e fare il profiling di programmi in dettaglio. È inoltre possibile utilizzare valgrind per costruire nuovi strumenti. Uno degli strumenti messi a disposizione è il cachegrind. La documentazione ufficiale è scaricabile presso [1].

Cachegrind Cachegrind simula come il programma interagisce con le memorie cache di una macchina. Anche se alcune macchine moderne hanno tre livelli di cache, cachegrind simula esclusivamente le cache di primo e di terzo livello. La ragione di questa scelta è che la cache L3 ha più influenza sulla fase di esecuzione, in quanto maschera accessi alla memoria principale, e la cache L1 ha spesso bassa associatività. Simulandole può rilevare i casi in cui il codice interagisce male con la cache L1, per esempio attraversando una matrice per colonna con la lunghezza di riga pari ad una potenza di 2. Pertanto, cachegrind si riferisce sempre alla I1 (cache di istruzioni di primo livello), D1 (cache di dati di primo livello) e LL (cache di ultimo livello).

Cachegrind raccoglie le seguenti statistiche (l'abbreviazione utilizzata per ogni statistica è indicata tra parentesi):

- letture dalla cache I (**Ir**, che è pari al numero di istruzioni eseguite), cachemiss in lettura dalla cache I1 (**I1mr**) e cachemiss in lettura dalla cache LL (**ILmr**).
- letture dalla cache D (**Dr**, che corrisponde al numero di letture dalla memoria), cachemiss in lettura dalla cache D1 (**D1mr**) e cachemiss in lettura dalla cache LL (**DLmr**).
- scritture nella cache D (**Dw**, che equivale al numero di scritture nella memoria), cachemiss in scrittura nella cache D1 (**D1mw**) e cachemiss in scrittura nella cache LL (**DLmw**).

Per eseguire cachegrind su un programma chiamato *prog* si deve usare il comando*:*

```
valgrind --tool=cachegrind prog
```

4.4.5 Pianificazione del profiling

I set di esperimenti si possono effettuare sui cicli presenti nel programma per valutare quali parti dell'algoritmo sono più time-consuming, e in generale, resource-consuming e quindi per identificare i colli di bottiglia computazioni esistenti per poi cercar di eliminarli o ridurli il più possibile.

4.4.5.1 Definizione set di esperimenti

Un primo set di esperimenti può essere quello di calcolare i tempi di esecuzione per tutti i blocchi di codice del programma. Un secondo test consiste nel calcolare quante volte i singoli blocchi vengono eseguiti in media ogni volta che il programma viene lanciato. Infine un terzo test consiste nel misurare il numero di cachemiss accorsi durante l'esecuzione del programma. Partendo quindi dallo pseudocodice della procedura *lognet2n()* inseriamo le istruzione per il calcolo del tempo usando la funzione *clock()*, inseriamo dei contatori

per il calcolo del numero di esecuzioni dei blocchi principali e usiamo il tool valgrind per misurare il numero di cachemiss.

Listing 4.2: Pseudocodice lognet2n() con istruzioni per il calcolo del tempo

```
1  procedure lognet2n
2      double time
3      clock_t start1, start2, end1, end2
4      istanzia nuove variabli
5      calcola il valore iniziale di beta0 e di altri parametri
6      for ilm=1 to nlam do
7          start1 = clock()
8
9          start2 = clock()
10         determina il nuovo valore di lambda
11         salva i valori dei beta del ciclo predecente
12         end2 = clock()
13         time = (start2 - end2)/(double) CLOCKS_PER_SEC
14         scrivi ("Tempo per calcolo di lambda e di
               salvataggio dei precedenti valori di beta = "
               tempo)
15
16         start2 = clock()
17         cont1 = cont1 + 1
18         for k=1 to ni do
19             Cyclic Coordinate Descent Algorithm su tutto il
                   vettore beta
20         //endfor
21         aggiorna il valore di beta0
22         end2 = clock()
23         time = (start2 - end2)/(double) CLOCKS_PER_SEC
24         scrivi ("Tempo per calcolo dei beta = " tempo)
25
26         while (soglia superata)
27             start2 = clock()
```

```
28              cont2 = cont2 + 1
29              for l=1 to nin do
30                  Cyclic Coordinate Descent Algorithm solo sui
                        beta che sono già stati modificati
31              //endfor
32              aggiorna beta0
33              end2 = clock()
34              time = (start2 - end2)/(double) CLOCKS_PER_SEC
35              scrivi ("Tempo per calcolo dei beta gia'
                    modificati = " tempo)
36          //endwhile
37
38          start2 = clock()
39          cont3 = cont3 + 1
40          calcola P(Y=0|X)
41          end2 = clock()
42          time = (start2 - end2)/(double) CLOCKS_PER_SEC
43          scrivi ("Tempo per calcolo delle probabilita' = "
                tempo)
44
45          start2 = clock()
46          if (beta0 o almeno un elemento del vettore beta è
                stato significativamente modificato) o (nuovi
                beta possono entrare in soluzione)
47              then torna alla riga numero 11
48          end2 = clock()
49          time = (start2 - end2)/(double) CLOCKS_PER_SEC
50          scrivi ("Tempo per determinare la fine dell'
                iterazione su un valore di lambda = " tempo)
51
52          start2 = clock()
53          salva i valori trovati per questo valore di lambda
54          end2 = clock()
55  File: lstlang1.2 - end2)/(double) CLOCKS_PER_SEC
```

```
56          scrivi ("Tempo per salvare i valori di beta = "
                tempo)
57
58          if (raggiunto il numero ne di beta entrati nel
                modello)
59              then esco dal ciclo su lambda
60          end1 = clock()
61          time = (start1 - end1)/(double) CLOCKS_PER_SEC
62          scrivi ("Tempo su lambda = " tempo)
63          scrivi (cont1 + " " + cont2 + " " + cont3)
64      ///endfor
```

4.4.5.2 Risultati set di esperimenti

I test sono stati effettuati su due dataset che si riferiscono al diabete di tipo 1 e a quello di tipo 2, sia utilizzando la matrice X normalizzata che non normalizzata. Nel diabete di tipo uno i valori di p e n sono rispettivamente 458376 e 4411, mentre in quello di tipo due sono 458376 e 4376.

Tempi medi di esecuzione Il primo set di esperimenti si riferisce ai tempi di esecuzione delle varie parti del programma per identificare il collo di bottiglia. Vengono effettuati i test modificando la taglia di input, il tipo di diabete dato in input e normalizzando o meno la matrice X.

Blocco di codice	**p=200000**	**p=300000**	**p=458376**
Lettura file	100.85	151.29	272.89
Calcolo vettore ju e normalizzazione matrice X	98.43	147.99	212.19
Calcolo di altre variabili di supporto	21.71	32.65	31.18
Calcolo del valore di λ e del vettore ixx	0.34	0.45	0.74
Salvataggio valori iniziali del vettore β e calcolo del vettore xv	65.22	73.67	63.41
Ciclo su tutti i β	41.59	46.39	53.92
Ciclo solo sui β già modificati	52.55	51.53	35.04
Calcolo delle probabilità	18.6	20.17	22.23
Determinazione del fine ciclo sui β	1322.07	1991.32	2883.00
Determinazione del fine ciclo su λ	0.09	0.15	0.09
Tempo totale di esecuzione	1721.45	2515.61	3574.69

Tabella 4.1: Tempi medi espressi in secondi cambiando il numero di SNP osservati (p) con $n = 4411$ utilizzando come file di input il diabete di tipo 1 e normalizzando la matrice X.

Blocco di codice	**Tempo**	**Percentuale**
Lettura file	272.89	7.63%
Calcolo vettore ju e normalizzazione matrice X	212.19	5.94%
Calcolo di altre variabili di supporto	31.18	0.87%
Calcolo del valore di λ e del vettore ixx	0.74	0.02%
Salvataggio valori iniziali del vettore β e calcolo del vettore xv	63.41	1.77%
Ciclo su tutti i β	53.92	1.51%
Ciclo solo sui β già modificati	35.04	0.98%
Calcolo delle probabilità	22.23	0.62%
Determinazione del fine ciclo sui β	2883.00	80.65%
Determinazione del fine ciclo su λ	0.09	0.00%
Tempo totale di esecuzione	3574.69	100.00%

Tabella 4.2: Tempi medi espressi in secondi con relative percentuali rispetto al tempo totale di esecuzione con $p = 458376$ e $n = 4411$ utilizzando come file di input il diabete di tipo 1 e normalizzando la matrice X.

Blocco di codice	**Tempo**	**Percentuale**
Lettura file	285.79	7.27%
Calcolo vettore ju	36.34	0.92%
Calcolo di altre variabili di supporto	30.00	0.76%
Calcolo del valore di λ e del vettore ixx	1.21	0.03%
Salvataggio valori iniziali del vettore β e calcolo del vettore xv	55.27	1.41%
Ciclo su tutti i β	124.00	3.15%
Ciclo solo sui β già modificati	251.67	6.40%
Calcolo delle probabilità	28.12	0.72%
Determinazione del fine ciclo sui β	3119.74	79.34%
Determinazione del fine ciclo su λ	0.10	0.00%
Tempo totale di esecuzione	3932.24	100.00%

Tabella 4.3: Tempi medi espressi in secondi con relative percentuali rispetto al tempo totale di esecuzione con $p = 458376$ e $n = 4411$ utilizzando come file di input il diabete di tipo 1 senza normalizzare la matrice X.

Blocco di codice	**p=200000**	**p=300000**	**p=458376**
Lettura file	99.54	149.32	269.34
Calcolo vettore ju e normalizzazione matrice X	95.28	143.26	205.41
Calcolo di altre variabili di supporto	21.36	32.13	30.68
Calcolo del valore di λ e del vettore ixx	0.33	0.43	0.71
Salvataggio valori iniziali del vettore β e calcolo del vettore xv	190.16	214.79	184.88
Ciclo su tutti i β	119.78	133.60	155.29
Ciclo solo sui β già modificati	92.97	91.16	61.99
Calcolo delle probabilità	63.05	68.38	75.36
Determinazione del fine ciclo sui β	1295.64	1951.51	2825.36
Determinazione del fine ciclo su λ	0.10	0.17	0.10
Tempo totale di esecuzione	1978.21	2784.75	3809.12

Tabella 4.4: Tempi medi espressi in secondi cambiando il numero di SNP osservati (p) con $n = 4376$ utilizzando come file di input il diabete di tipo due e normalizzando la matrice X.

Blocco di codice	**Tempo**	**Percentuale**
Lettura file	269.34	7.07%
Calcolo vettore ju e normalizzazione matrice X	205.41	5.39%
Calcolo di altre variabili di supporto	30.68	0.81%
Calcolo del valore di λ e del vettore ixx	0.71	0.02%
Salvataggio valori iniziali del vettore β e calcolo del vettore xv	184.88	4.85%
Ciclo su tutti i β	155.29	4.08%
Ciclo solo sui β già modificati	61.99	1.63%
Calcolo delle probabilità	75.36	1.98%
Determinazione del fine ciclo sui β	2825.36	74.17%
Determinazione del fine ciclo su λ	0.10	0.00%
Tempo totale di esecuzione	3809.12	100.00%

Tabella 4.5: Tempi medi espressi in secondi con relative percentuali rispetto al tempo totale di esecuzione con $p = 458376$ e $n = 4376$ utilizzando come file di input il diabete di tipo 2 e normalizzando la matrice X.

Blocco di codice	**Tempo**	**Percentuale**
Lettura file	289.99	6.99%
Calcolo vettore ju	35.99	0.87%
Calcolo di altre variabili di supporto	29.71	0.72%
Calcolo del valore di λ e del vettore ixx	1.20	0.03%
Salvataggio valori iniziali del vettore β e calcolo del vettore xv	186.85	4.50%
Ciclo su tutti i β	222.16	5.35%
Ciclo solo sui β già modificati	207.99	5.01%
Calcolo delle probabilità	114.19	2.75%
Determinazione del fine ciclo sui β	3060.78	73.77%
Determinazione del fine ciclo su λ	0.11	0.00%
Tempo totale di esecuzione	4148.97	100.00%

Tabella 4.6: Tempi medi espressi in secondi con relative percentuali rispetto al tempo totale di esecuzione con $p = 458376$ e $n = 4376$ utilizzando come file di input il diabete di tipo 2 senza normalizzare la matrice X.

Numero di esecuzione dei blocchi principali Il secondo set di esperimenti invece serve per capire quante volte vengono eseguiti i blocchi principali del programma ad eccezione di quelli noti a priori. I blocchi su cui verrà effettuato questo test saranno quindi il ciclo per il calcolo di tutti i β, il ciclo per il calcolo dei β già modificati in precedenza e il ciclo per il calcolo della probabilità $P(x|y=0)$. Vengono inoltre evidenziati il numero di β che alla fine sono entrati in soluzione. Come nel test precedente, vengono effettuati i test modificando la taglia di input, il tipo di diabete dato in input e normalizzando o meno la matrice X.

Blocco di codice	**p=200000**	**p=300000**	**p=4583776**
Ciclo su tutti i β	311	311	310
Ciclo solo sui β già modificati	425	405	387
Numero di volte che viene calcolata la probabilità $P(x\|y=0)$	201	200	199
Numero di β entrati in soluzione	2556	2655	2886

Tabella 4.7: Numeri medi di esecuzione dei blocchi principali all'interno del ciclo su λ e numero medio di β entrati in soluzione con $n = 4411$ per il diabete di tipo 1 senza normalizzazione della matrice X cambiando il numero di SNP osservati (p).

Blocco di codice	**Con norm.**	**Senza norm.**	**Diff.**
Cicli su tutti i β	310	702	392
Cicli solo sui β già modificati	387	3587	3200
Numero di volte che viene calcolata la probabilità $P(x\|y=0)$	199	206	7
Numero di β entrati in soluzione	2886	3729	843

Tabella 4.8: Numeri medi di esecuzione dei blocchi principali all'interno del ciclo su λ e numero medio di β entrati in soluzione con $p = 458376$ e $n = 4411$ per il diabete di tipo 1 con e senza normalizzazione della matrice X.

Blocco di codice	**p=200000**	**p=300000**	**p=4583776**
Ciclo su tutti i β	309	309	309
Ciclo solo sui β già modificati	309	294	281
Numero di volte che viene calcolata la probabilità $P(x\|y=0)$	201	200	199
Numero di β entrati in soluzione	3967	4120	4479

Tabella 4.9: Numeri medi di esecuzione dei blocchi principali all'interno del ciclo su λ e numero medio di β entrati in soluzione con $n = 4376$ per il diabete di tipo 2 senza normalizzazione della matrice X cambiando il numero di SNP osservati (p).

Blocco di codice	**Con norm.**	**Senza norm.**	**Diff.**
Cicli su tutti i β	309	545	236
Cicli solo sui β già modificati	281	1359	1078
Numero di volte che viene calcolata la probabilità $P(x\|y=0)$	199	193	-6
Numero di β entrati in soluzione	4479	7131	2652

Tabella 4.10: Numeri medi di esecuzione dei blocchi principali all'interno del ciclo su λ e numero medio di β entrati in soluzione con $p = 458376$ e $n = 4376$ per il diabete di tipo 2 con e senza normalizzazione della matrice X.

Numero di cachemiss utilizzando il tool valgrind Il terzo e ultimo test di esperimenti è volto a capire quante sono le occorrenze di cachemiss durante l'intera esecuzione del programma, tenendo distinte quelle in lettura e quelle in scrittura. Anche qui vengono effettuati i test modificando la taglia di input, il tipo di diabete dato in input e normalizzando o meno la matrice X.

Descrizione	**Totale**	**Lettura**	**Scrittura**
Numero di istruzioni eseguite	245,671,880,634		
Cachemiss in lettura della cache I1	44,843		
Cachemiss in lettura della cache IL	24,660		
Miss-rate della cache I1	0.000%		
Miss-rate della cache IL	0.000%		
Numero di accessi alle cache dati	90,124,476,970	77,448,768,300	12,675,708,670
Cachemiss della cache dati D1	361,533,231	356,697,600	4,835,631
Cachemiss della cache dati DL	295,820,156	292,489,232	3,330,924
Miss-rate della cache D1	0.401%	0.461%	0.038%
Miss-rate della cache DL	0.328%	0.378%	0.026%
Numero di accessi alla cache di ultimo livello (LL)	361,578,074	356,742,443	4,835,631
Cachemiss della cache LL	295,814,816	292,513,892	3,300,924
Miss-rate della cache LL rispetto al totale degli accessi in memoria	0.088%	0.091%	0.026%

Tabella 4.11: Cachemiss e miss-rate delle diverse cache utilizzando come input il diabete di tipo 1 con $p = 10000$ e $n = 4411$ e normalizzando la matrice X.

Descrizione	**Totale**	**Lettura**	**Scrittura**
Numero di istruzioni eseguite	2,986,482,720,810		
Cachemiss in lettura della cache I1	64,087		
Cachemiss in lettura della cache IL	45,010		
Miss-rate della cache I1	0.000%		
Miss-rate della cache IL	0.000%		
Numero di accessi alle cache dati	1,089,081,725,755	930,008,205,075	159,073,520,680
Cachemiss della cache dati D1	7,225,620,568	7,177,791,113	47,829,455
Cachemiss della cache dati DL	6,130,349,261	6,092,868,115	37,481,146
Miss-rate della cache D1	0.663%	0.772%	0.030%
Miss-rate della cache DL	0.563%	0.655%	0.024%
Numero di accessi alla cache di ultimo livello (LL)	7,225,684,655	7,177,855,200	47,829,455
Cachemiss della cache LL	6,130,394,271	6,092,913,125	37,481,146
Miss-rate della cache LL rispetto al totale degli accessi in memoria	0.150%	0.156%	0.024%

Tabella 4.12: Cachemiss e miss-rate delle diverse cache utilizzando come input il diabete di tipo 1 con $p = 100000$ e $n = 4411$ e normalizzando la matrice X.

Descrizione	**Totale**	**Lettura**	**Scrittura**
Numero di istruzioni eseguite	250,881,113,773		
Cachemiss in lettura della cache I1	44,280		
Cachemiss in lettura della cache IL	24,358		
Miss-rate della cache I1	0.000%		
Miss-rate della cache IL	0.000%		
Numero di accessi alle cache dati	91,858,666,713	78,824,012,059	13,034,654,654
Cachemiss della cache dati D1	399,679,443	394,890,083	4,789,360
Cachemiss della cache dati DL	298,227,032	294,898,151	3,328,881
Miss-rate della cache D1	0.435%	0.501%	0.037%
Miss-rate della cache DL	0.325%	0.374%	0.026%
Numero di accessi alla cache di ultimo livello (LL)	399,723,723	394,934,363	4,789,360
Cachemiss della cache LL	298,251,390	294,922,509	3,328,881
Miss-rate della cache LL rispetto al totale degli accessi in memoria	0.087%	0.089%	0.026%

Tabella 4.13: Cachemiss e miss-rate delle diverse cache utilizzando come input il diabete di tipo 2 con $p = 10000$ e $n = 4376$ e normalizzando la matrice X.

Descrizione	**Totale**	**Lettura**	**Scrittura**
Numero di istruzioni eseguite	8,231,602,181,530		
Cachemiss in lettura della cache I1	332,992		
Cachemiss in lettura della cache IL	297,901		
Miss-rate della cache I1	0.000%		
Miss-rate della cache IL	0.000%		
Numero di accessi alle cache dati	2,556,723,297,131	2,099,400,828,417	457,322,468,714
Cachemiss della cache dati D1	12,668,748,105	12,587,036,021	81,712,084
Cachemiss della cache dati DL	7,251,775,831	7,242,548,921	9,226,910
Miss-rate della cache D1	0.496%	0.600%	0.018%
Miss-rate della cache DL	0.284%	0.345%	0.002%
Numero di accessi alla cache di ultimo livello (LL)	12,669,081,097	12,587,369,013	81,712,084
Cachemiss della cache LL	7,252,073,732	7,242,846,822	9,226,910
Miss-rate della cache LL rispetto al totale degli accessi in memoria	0.067%	0.070%	0.002%

Tabella 4.14: Cachemiss e miss-rate delle diverse cache utilizzando come input il diabete di tipo 1 con $p = 10000$ e $n = 4411$ senza normalizzare la matrice X.

4.4.5.3 Conclusioni

Collo di bottiglia Come si può vedere dalle tabelle nella sezione 4.4.5.2, il blocco di codice che rappresenta il collo di bottiglia del programma è la determinazione di fine ciclo sui β, in cui, se nessun β è stato significativamente modificato, bisogna scandire tutta la matrice X. Ovviamente il tempo di esecuzione di questo blocco è proporzionale al numero di SNPs presi in esame e rappresenta circa il 70-80% del tempo totale di esecuzione dell'intero programma.

Cachemiss Le tabelle nella sezione 4.4.5.2 evidenziano che la miss-rate dipende quasi esclusivamente dal numero di SNPs osservati. Infatti, anche variando il file di input o evitando di eseguire la normalizzazione della matrice X, la miss-rate non varia di molto, mentre sale all'aumentare del valore p.

Il numero assoluto di cachemiss però, oltre che aumentare all'aumenta di p, è molto maggiore se non si effettua la normalizzazione. Questo sia per la cache di primo livello che per quella di ultimo livello, soprattutto in fase di lettura.

Normalizzazione: pro e contro Senza la normalizzazione della matrice X si possono memorizzare i dati utilizzando solo 2 bit al posto dei 4 byte (32 bit) necessari per memorizzare un float. Si riuscirebbe quindi a diminuire di una fattore 16 l'occupazione di memoria da parte della matrice X. Dall'altro verso però se non si effettua la normalizzazione il tempo di esecuzione aumenta di circa 5-6 minuti (con $p = 458376$), il numero assoluto di cachemiss aumenta anche di un fattore 30-35 e, cosa più importante, usando la tecnica statistica della k-fold cross-validation l'accuracy del programma diminuisce in media del 2,5%.

Numero di iterazioni su λ Come evidenziato dalla figura 4.4.1 la percentuale di accuracy del modello trovata cambia nelle varie iterazioni di λ.

In molti casi, come nell'esempio di figura 4.4.1, la percentuale maggiore non si ha nell'ultima iterazione ma si ha alle iterazioni numero 65 e 66. Si potrebbe quindi fermare l'esecuzione del programma una volta trovato il valore massimo, cioè quando la percentuale comincia a scendere, per ridurre il tempo di esecuzione. Da notare però che questo non si può fare in quanto la curva non è monotona crescente fino al valore massimo. Per ovviare a questo problema si può fermare l'esecuzione del programma quando la percentuale di accuracy nell'iterazione corrente è inferiore di una certa soglia rispetto alla percentuale massima corrente. Fissando ipoteticamente la soglia a 1,5% nell'esempio di figura 4.4.1 l'ultima iterazione su λ che verrebbe eseguita sarebbe la numero 75, riducendo così notevolmente il tempo di esecuzione.

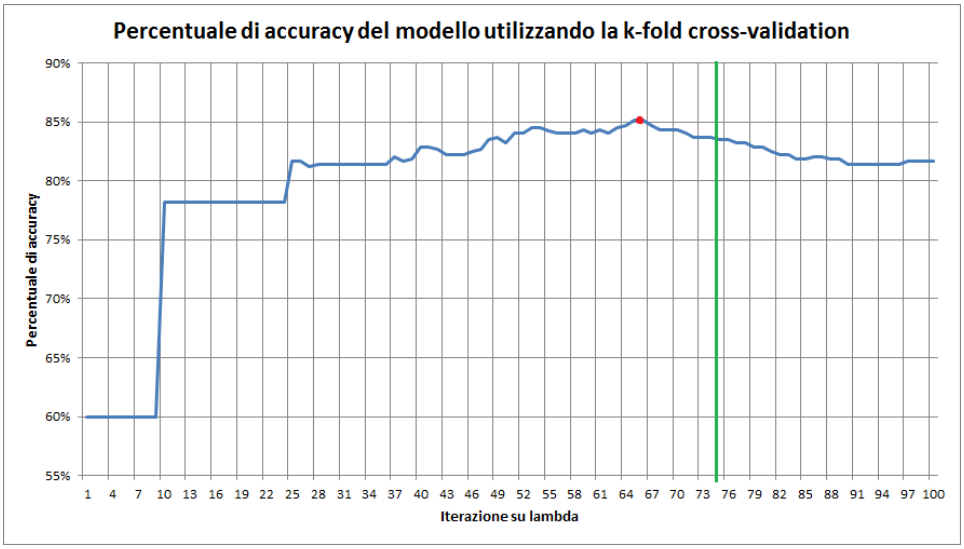

Figura 4.4.1: Esempio di percentuale di accuracy di un modello per ogni iterazione su λ utilizzando la cross-validazione. Il punto rosso indica dove si raggiunge l'accuracy massima, mentre il segmento verde indica dove si potrebbero fermare le iterazioni fissando la soglia a 1,5%.

4.5 Scelta del parametro α

Il primo parametro di cui fare il tuning è il parametro α, che assume valori reali compresi tra 0 e 1. Per scegliere qual è il valore adatto per ogni dataset

si esegue l'algoritmo più volte cambiando tale valore. L'esecuzione che darà un'accuracy massima, calcolata sulla media di 3 test, ci dirà quale sarà il valore migliore per il parametro α.

Ciascun test si effettua variando il numero di SNPs, in particolare si utilizzeranno i valori 1000, 10000, 100000 e 458376. Per selezionare gli n SNPs si ordineranno in base alla statistica data dall'Armitage Test for Trend prendendone i primi n.

4.5.1 Diabete di tipo 1

Numero di SNPs	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$p = 1000$	75.68%	76.15%	76.01%	75.81%	75.93%
$p = 10000$	77.23%	77.83%	77.85%	78.01%	78.11%
$p = 100000$	79.76%	80.50%	80.52%	80.38%	80.58%
$p = 458376$	80.01%	80.56%	80.54%	80.31%	80.56%

Tabella 4.15: Accuracy media su 3 test per il diabete di tipo 1 con $n = 3267$ variando il numero di SNPs e il valore di α.

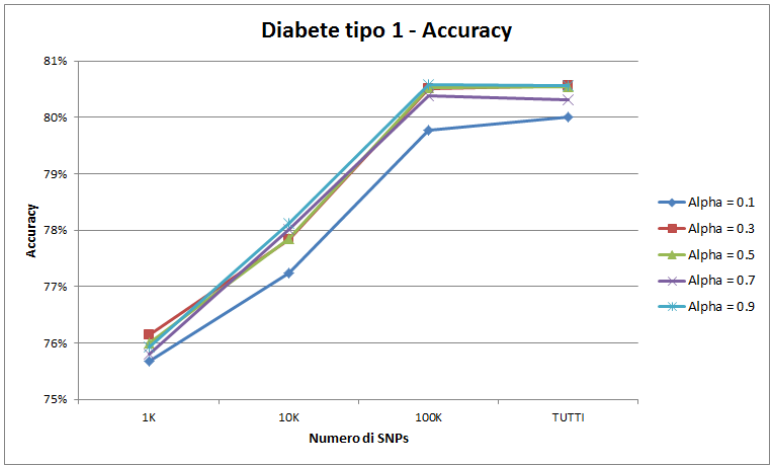

Figura 4.5.1: Grafico rappresentante l'accuracy media su 3 test per il diabete di tipo 1 con $n = 3267$ variando il numero di SNPs e il valore di α.

Numero di SNPs	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$p = 1000$	0.4893	0.4991	0.4961	0.4915	0.4941
$p = 10000$	0.5189	0.5325	0.5335	0.5373	0.5394
$p = 100000$	0.5736	0.5884	0.5908	0.5884	0.5927
$p = 458376$	0.5788	0.5911	0.5915	0.5872	0.5925

Tabella 4.16: MCC medio su 3 test per il diabete di tipo 1 con $n = 3267$ variando il numero di SNPs e il valore di α.

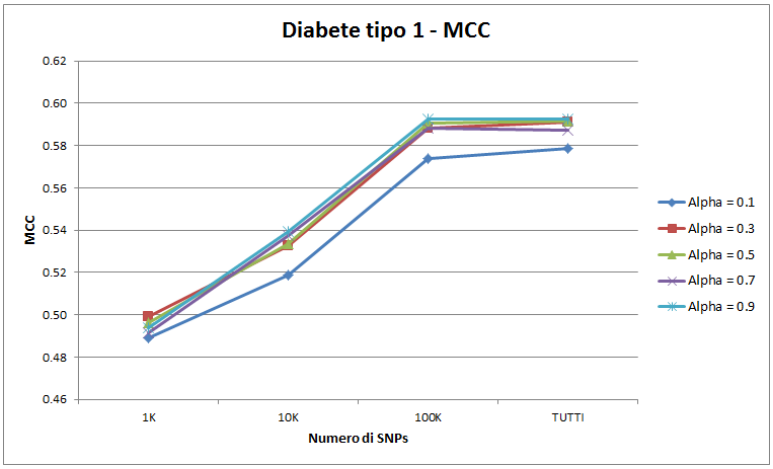

Figura 4.5.2: Grafico rappresentante l'MCC medio su 3 test per il diabete di tipo 1 con $n = 3267$ variando il numero di SNPs e il valore di α.

In base alle tabelle 4.15 e 4.16 il valore migliore del parametro α per il diabete di tipo 1 è 0.9. Questo è un valore che porta la penalità elastic-net vicina alla lasso, cioè alla norma ℓ_1, la quale tende a fissare molti β pari a 0 come si può vedere dalla tabella 4.17.

Il motivo per cui il numero di β in soluzione diminuisce nel passare da $p = 10000$ a $p = 100000$ è dovuto al valore di λ ottimo. Infatti, se si guarda la tabella 4.34, le iterazioni con $p = 10000$ sono il doppio rispetto agli altri valori di p e quindi in si ha un λ ottimo molto minore rispetto agli altri.

Numero di SNPs	λ ottimo medio	Numero di β in soluzione
$p = 1000$	0.003184	427
$p = 10000$	0.002785	1673
$p = 100000$	0.021786	372
$p = 458376$	0.020945	432

Tabella 4.17: λ ottimo medio e numero medio di β in soluzione su 3 test per il diabete di tipo 1 con $n = 3267$ e $\alpha = 0.9$ variando il numero di SNPs.

4.5.2 Diabete di tipo 2

Numero di SNPs	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$p = 1000$	59.02%	58.79%	58.77%	58.77%	58.69%
$p = 10000$	61.06%	60.89%	60.23%	60.28%	60.17%
$p = 100000$	62.60%	61.53%	60.91%	60.63%	60.07%
$p = 458376$	62.74%	61.37%	60.79%	60.50%	60.38%

Tabella 4.18: Accuracy media su 3 test per il diabete di tipo 2 con $n = 3241$ variando il numero di SNPs e il valore di α.

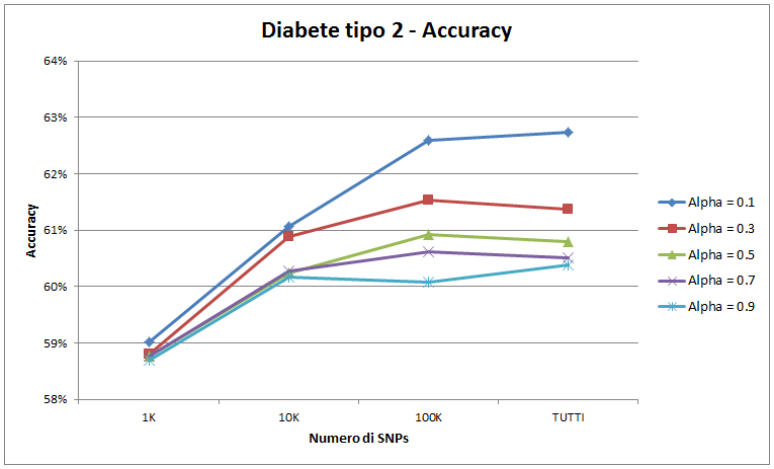

Figura 4.5.3: Grafico rappresentante l'accuracy media su 3 test per il diabete di tipo 2 con $n = 3241$ variando il numero di SNPs e il valore di α.

Numero di SNPs	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$p = 1000$	0.1140	0.1120	0.1152	0.1147	0.1137
$p = 10000$	0.1388	0.1410	0.1302	0.1313	0.1291
$p = 100000$	0.1733	0.1018	0.0938	0.0860	0.0588
$p = 458376$	0.1734	0.0981	0.0903	0.0833	0.0712

Tabella 4.19: MCC medio su 3 test per il diabete di tipo 2 con $n = 3241$ variando il numero di SNPs e il valore di α.

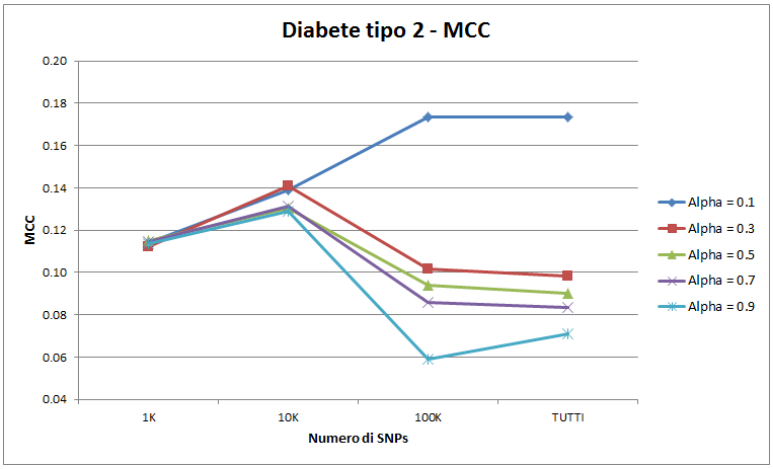

Figura 4.5.4: Grafico rappresentante l'MCC medio su 3 test per il diabete di tipo 2 con $n = 3241$ variando il numero di SNPs e il valore di α.

In base alle tabelle 4.18 e 4.19 il valore migliore del parametro α per il diabete di tipo 2 è 0.1. Questo è un valore che porta la penalità elastic-net vicina alla ridge-regression, cioè alla norma ℓ_2, la quale tende a fissare meno β pari a 0 come si può dalla tabella 4.20.

In modo analogo al diabete 1, anche qui il numero di β in soluzione diminuisce passando da $p = 100000$ a $p = 458376$ a causa dell'iterazione in cui si trova il λ ottimo, vedere tabella 4.35, e del suo valore.

Numero di SNPs	λ ottimo medio	Numero di β in soluzione
$p = 1000$	0.015509	905
$p = 10000$	0.005014	6180
$p = 100000$	0.005723	9140
$p = 458376$	0.032317	7462

Tabella 4.20: λ ottimo medio e numero medio di β in soluzione su 3 test per il diabete di tipo 1 con $n = 3241$ e $\alpha = 0.1$ variando il numero di SNPs.

4.6 Parallelizzare il codice

Sulla base delle conclusioni dei test descritte nella sezione 4.4.5.3 si intende applicare la k-fold cross-validation in modo parallelo per trovare il λ ottimo e poi parallelizzare alcune parti di codice nella creazione del modello finale.

Come spiegato nella sezione 4.5 il valore del parametro α sarà pari a 0.9 per il diabete di tipo 1 e pari a 0.1 per il diabete di tipo 2.

4.6.1 MPI

4.6.1.1 Cos'è

Message Passing Interface (MPI) [10] è una libreria per la programmazione di applicazione parallele. L'obiettivo di MPI è quello di sviluppare uno standard utilizzato per la scrittura di applicazioni parallele, in quanto tale l'interfaccia dovrebbe stabilire uno standard pratico, portatile, efficiente e flessibile per lo scambio di messaggi. MPI non è uno standard de iure, ma è uno standard de facto per la programmazione a scambio di messaggi.

I principali vantaggi dello standard MPI sono portabilità, scalabilità ed efficienza; infatti è indipendente da implementazione, linguaggio e architettura di calcolo e di rete. MPI è quindi un protocollo di comunicazione indipendente dal linguaggio utilizzato e supporta comunicazioni sia point-to-point che collettive.

Lo standard MPI definisci la sintassi e la semantica di un insieme di funzioni per lo scambio di messaggi in programmi, principalmente scritti in C/C++ e Fortran. MPI-2.2 è la versione attuale di questo standard.

4.6.1.2 Istruzioni principali

Per fare in modo che i processi comunicano tra loro è necessario un comunicatore. Il comunicatore di default è MPI_COMM_WORLD, che comprende tutti gli n processi lanciati dal sistema run-time. Per utilizzare lo standard MPI è ovviamente anche necessario includere la libreria *mpi.h* dove sono definite tutte le seguenti funzioni.

Inizializzare MPI Deve essere chiamata da tutti i processi prima di ogni altra istruzione MPI

```
int MPI_Init(int *argc,char ***argv);
```

Terminare MPI Deve essere chiamata da tutti i processi come ultima istruzione MPI

```
int MPI_Finalize(void);
```

Quanti siamo? *size* è il valore restituito

```
int MPI_Comm_size(MPI_Comm comm, int *size);
```

Chi sono io? *rank* è l'identificativo del processo corrente, da 0 a $size - 1$

```
int MPI_Comm_rank(MPI_Comm comm, int *rank);
```

Spedire un messaggio Ritorna quando il messaggio è stato copiato dal sistema

```
int MPI_Send(void* mes, int lenght, MPI_Datatype
    type, int dest, int tag, MPI_Comm comm);
```

Broadcast di un messaggio Replica il messaggio da *root* a tutti gli altri processi

```
int MPI_Bcast(void* mes, int lenght, MPI_Datatype
    type, int root, MPI_Comm comm);
```

Ricevere un messaggio Ritorna quando il messaggio è stato ricevuto nel buffer nello stazio utente

```
int MPI_Recv(void* mes, int lenght, MPI_Datatype
    type, int source, int tag, MPI_Comm comm,
    MPI_Status, *status);
```

Sincronizzare i processi Blocca il chiamante finché tutti i processi effettuano la chiamata

```
int MPI_Barrier(MPI_Comm comm);
```

4.6.2 Parallelizzazione della k-fold cross-validation

Attraverso il parallelismo è possibile eseguire la k-fold cross-validation utilizzando k nodi distinti. In questo modo i nodi lavorano separatamente nella costruzione dei modelli. Utilizzando la k-fold cross-validation in parallelo è anche possibile ridurre il numero delle iterazione su λ, come spiegato nella sezione 4.4.5.3.

Al termine di ogni iterazione su λ, con il vettore β appena trovato, ogni nodo calcola la percentuale di accuracy con il proprio validation set. Il nodo 0 si occupa poi di combinarle calcolando l'accuracy media. Se questa percentuale è più piccola di 1.5% rispetto alla percentuale di accuracy massima trovata finora si interrompono le iterazioni a quella attuale. Le successive iterazioni sarebbero inutili, in quanto non porterebbero ad un aumento della percentuale di accuracy massima e quindi troverebbero una funzione di classificazione che descrive in maniera peggiore il modello dando origine al

problema dell'overfitting. Nella versione originaria del programma il numero delle iterazioni su λ è fissato a 100.

Numero di SNPs	Numero di cicli effettuati	Accuracy massima
$p = 1000$	67	87.98%
$p = 10000$	100	85.62%
$p = 100000$	85	80.75%
$p = 458376$	78	80.79%

Tabella 4.21: Numero medio di cicli effettuati su λ su 3 test con relativa accuracy massima per il diabete di tipo 1 variando il numero p di SNPs con $n = 3267$, $size = k = 10$ e $\alpha = 0.9$.

Numero di SNPs	Numero di cicli effettuati	Accuracy massima
$p = 1000$	58	87.64%
$p = 10000$	100	86.78%
$p = 100000$	100	63.02%
$p = 458376$	100	61.92%

Tabella 4.22: Numero medio di cicli effettuati su λ su 3 test con relativa accuracy massima per il diabete di tipo 2 variando il numero p di SNPs con $n = 3241$, $size = k = 10$ e $\alpha = 0.1$.

Come si può vedere dalle tabelle 4.21 e 4.22 soprattutto per il diabete di tipo 1 il numero di iterazioni da 100 si è ridotto. Le differenze fra il diabete di tipo 1 e 2 sono dovute alla correlazione dei SNPs e al valore di α scelto. Infatti in media per il diabete di tipo 1 il numero di β nella soluzione ottima sarà minore rispetto a quello per il diabete di tipo 1. A causa di questa correlazione il diabete di tipo 1 da un'accuracy migliore rispetto a quello di tipo 2.

4.6.2.1 Tempi di esecuzione

Numero di SNPs	Versione originaria	Versione parallela	Speed Up
$p = 1000$	00:30:34	00:06:15	4.89
$p = 10000$	00:26:35	00:02:44	9.71
$p = 100000$	02:17:25	00:11:22	12.09
$p = 458376$	08:43:25	00:41:24	12.64

Tabella 4.23: Tempi medi di esecuzione espressi nella forma hh:mm:ss effettuati per k-fold cross-validation su 3 test nella versione originaria e parallela e relativo speed up per il diabete di tipo 1 variando il numero p di SNPs con $n = 3267$, $size = k = 10$ e $\alpha = 0.9$.

Come evidenziato nella tabella 4.23 per il diabete di tipo 1 utilizzando come input 100000 o 458376 SNPs si ottiene uno speed up superiore a $k = 10$, questo perché in questi due casi la parallelizzazione riesce a ridurre il numero di iterazioni su λ portando quindi lo speed up ad un valore superiore a 12. Con $p = 10000$ invece lo speed up si aggira intorno al valore 10 proprio perché non si riesce a ridurre il numero di iterazioni che resta a 100. Con $p = 1000$ infine lo speed up medio è pari a 4.89 in quanto al termine di ogni iterazione su λ i 10 nodi si devono sincronizzare per comunicare al nodo 0 l'accuracy appena calcolata, se uno o più nodi sono quindi in ritardo gli altri devono aspettarli.

Numero di SNPs	Versione originaria	Versione parallela	Speed Up
$p = 1000$	00:23:07	00:01:43	13.41
$p = 10000$	01:32:40	00:09:36	9.64
$p = 100000$	04:08:29	00:25:43	9.66
$p = 458376$	10:37:25	01:05:20	9.76

Tabella 4.24: Tempi medi di esecuzione espressi nella forma hh:mm:ss effettuati per k-fold cross-validation su 3 test nella versione originaria e parallela e relativo speed up per il diabete di tipo 2 variando il numero p di SNPs con $n = 3241$, $size = k = 10$ e $\alpha = 0.1$.

Per il diabete di tipo 2 invece la tabella 4.24 mostra uno speed up pari a circa $k = 10$ per 1000, 10000 o 458376 SNPs dato che la parallelizzazione non riduce il numero di iterazioni su λ. Cosa che invece accade per $p = 1000$ dove lo speed up è di 13.41.

4.6.3 Parallelizzazione della costruzione del modello finale

Sulla base del λ ottimo trovato tramite la k-fold cross-validation verrà costruito il modello finale.

In questa parte del programma i blocchi di codice che si potranno parallelizzare sono la normalizzazione della matrice X e ovviamente il collo di bottiglia. In questi blocchi i nodi saranno divisi in due gruppi: nel primo ci sarà solo il nodo con $rank$ 0 (nodo master), che si occuperà di calcolare parte delle informazioni, delle comunicazioni con gli altri nodi e di raggruppare le informazioni che gli vengono passate; mentre nel secondo ci saranno tutti gli altri nodi (nodi slave), che effettueranno solo codice operazionale e comunicheranno esclusivamente con il nodo master. Quindi si hanno un nodo master e $size - 1$ nodi slave.

I blocchi di codice non parallelizzati saranno eseguiti tutti dal nodo master.

4.6.3.1 La normalizzazione della matrice X

Il primo blocco di codice che si può parallelizzare è la normalizzazione della matrice X. Esso presenta due cicli innestati: quello esterno sul numero p di SNPs e quello interno sul numero n di soggetti. Viene quindi suddiviso il ciclo esterno in $size$ parti tra il nodo master e i nodi slave che, dopo aver calcolato la loro parte di matrice, gliela passeranno al nodo master, che si occuperà di unire le varie sottomatrici nella matrice iniziale X.

Insieme alla normalizzazione si effettua in parallelo anche il calcolo del vettore ju. Tale vettore era l'output della funzione chkvars (vedi sezione 3.5.3.1) presente nella versione originaria del programma.

Numero di SNPs	Versione originaria	Versione parallela
$p = 1000$	0.40	0.09
$p = 10000$	3.98	0.86
$p = 100000$	39.68	8.56
$p = 458376$	174.47	38.03

Tabella 4.25: Tempi medi espressi in secondi su 3 test necessari per il calcolo del vettore ju e della normalizzazione della matrice X per il diabete di tipo 1 variando il numero p di SNPs con $n = 3267$, $size = 10$ e $\alpha = 0.9$.

Numero di SNPs	Versione originaria	Versione parallela
$p = 1000$	0.39	0.08
$p = 10000$	3.94	0.86
$p = 100000$	39.84	8.53
$p = 458376$	173.14	37.46

Tabella 4.26: Tempi medi espressi in secondi su 3 test necessari per il calcolo del vettore ju e della normalizzazione della matrice X per il diabete di tipo 2 variando il numero p di SNPs con $n = 3241$, $size = 10$ e $\alpha = 0.1$.

Le tabelle 4.25 e 4.26 mostrano come si è ridotto il tempo per effettuare la normalizzazione della matrice X. Utilizzando 10 nodi per entrambi i tipi di diabete si riscontra un significativo miglioramento: si risparmia circa l'80% indipendente dalla taglia della matrice X data in input.

4.6.3.2 Il collo di bottiglia

Il secondo blocco di codice da parallelizzare è il collo di bottiglia descritto nella sezione 4.4.5.3. Come nella normalizzazione della matrice X, anche qui sono presenti i due cicli innestati e verrà suddiviso il ciclo esterno tra i vari nodi. I nodi slave passeranno poi al nodo master le informazioni necessarie.

Qui però a causa della cross-validazione non è noto a priori quale sarà il numero delle iterazioni su λ. Per cui il nodo master dovrà provvedere anche all'invio di un particolare segnale per far terminare i nodi slave quando uscirà dal ciclo su λ.

Numero di SNPs	Versione originaria	Versione parallela
$p = 1000$	0.0203	0.0024
$p = 10000$	0.2412	0.0271
$p = 100000$	2.7117	0.2920
$p = 458376$	11.8848	1.2599

Tabella 4.27: Tempi medi espressi in secondi su 3 test necessari per la determinazione del fine ciclo su β per il diabete di tipo 1 variando il numero p di SNPs con $n = 3267$, $size = 10$ e $\alpha = 0.9$.

Numero di SNPs	Versione originaria	Versione parallela
$p = 1000$	0.0035	0.0007
$p = 10000$	0.1151	0.0192
$p = 100000$	2.4323	0.2806
$p = 458376$	11.5535	1.2122

Tabella 4.28: Tempi medi espressi in secondi su 3 test necessari per la determinazione del fine ciclo su β per il diabete di tipo 2 variando il numero p di SNPs con $n = 3241$, $size = 10$ e $\alpha = 0.1$.

Le tabelle 4.27 e 4.28 mostrano invece come si è ridotto il collo di bottiglia. Grazie alla parallelizzazione il tempo necessario per l'esecuzione di questo blocco ad ogni iterazione è diventato quasi irrisorio. Infatti, utilizzando tutta la matrice X come input, si riesce a ridurre il collo di bottiglia per il diabete di tipo 1 e 2 di quasi il 90%.

4.6.3.3 Tempi di esecuzione

Numero di SNPs	Versione originaria	Versione parallela	Speed Up
$p = 1000$	00:00:17	00:00:13	1.23
$p = 10000$	00:02:08	00:01:41	1.26
$p = 100000$	00:03:23	00:00:37	5.56
$p = 458376$	00:14:59	00:02:30	6.00

Tabella 4.29: Tempi medi di esecuzione espressi nella forma hh:mm:ss effettuati per la creazione del modello finale su 3 test nella versione originaria e parallela e relativo speed up per il diabete di tipo 1 variando il numero p di SNPs con $n = 3267$, $size = k = 10$ e $\alpha = 0.9$.

Per il diabete di tipo 1 si ha un buono speed up per 10000 o 458476 SNPs come si può vedere dalla tabella 4.29. Questo perché negli altri due casi il tempo per la determinazione del fine ciclo sul vettore β è irrisorio rispetto al tempo impiegato dagli altri blocchi di codice.

Numero di SNPs	Versione originaria	Versione parallela	Speed Up
$p = 1000$	00:00:32	00:00:32	1.01
$p = 10000$	00:09:10	00:08:57	1.02
$p = 100000$	00:18:11	00:14:33	1.25
$p = 458376$	00:26:15	00:11:25	2.30

Tabella 4.30: Tempi medi di esecuzione espressi nella forma hh:mm:ss effettuati per la creazione del modello finale su 3 test nella versione originaria e parallela e relativo speed up per il diabete di tipo 2 variando il numero p di SNPs con $n = 3241$, $size = k = 10$ e $\alpha = 0.1$.

La tabella 4.30 mostra invece che per il diabete di tipo 2 non si ha un significativo miglioramento. Questo perché utilizzando $\alpha = 0.1$ molti β entrano in soluzione e quindi il tempo per il calcolo e l'aggiornamento di tali valori risulta essere superiore al tempo necessario per la determinazione del fine ciclo sul vettore β.

4.7 Risultati finali

Tutti i tempi e percentuali di questa sezione sono stati calcolati sulla base di 3 test senza tener conto del tempo necessario per leggere da file il vettore Y e la matrice X. I tempi di lettura sono in ogni caso sempre pari a circa 270 secondi, indipendentemente dal numero di SNPs presi in considerazione per la creazione del modello.

4.7.1 Tempi complessivi

Numero di SNPs	Versione originaria	Versione parallela	Speed Up
$p = 1000$	00:30:51	00:06:29	4.76
$p = 10000$	00:28:42	00:04:25	6.50
$p = 100000$	02:20:48	00:11:58	11.76
$p = 458376$	08:58:24	00:43:53	12.27

Tabella 4.31: Tempi medi, espressi in nella forma hh:mm:ss, necessari per il calcolo della funzione di classificazione $Y = f(X, \beta)$ per il diabete di tipo 1 variando il numero p di SNPs con $n = 3267$, $size = 10$ e $\alpha = 0.9$.

Numero di SNPs	Versione originaria	Versione parallela	Speed Up
$p = 1000$	00:23:39	00:02:15	10.50
$p = 10000$	01:41:50	00:18:34	5.49
$p = 100000$	04:26:40	00:40:15	6.62
$p = 458376$	11:03:40	01:16:44	8.65

Tabella 4.32: Tempi medi, espressi in nella forma hh:mm:ss, necessari per il calcolo della funzione di classificazione $Y = f(X, \beta)$ per il diabete di tipo 2 variando il numero p di SNPs con $n = 3241$, $size = 10$ e $\alpha = 0.1$.

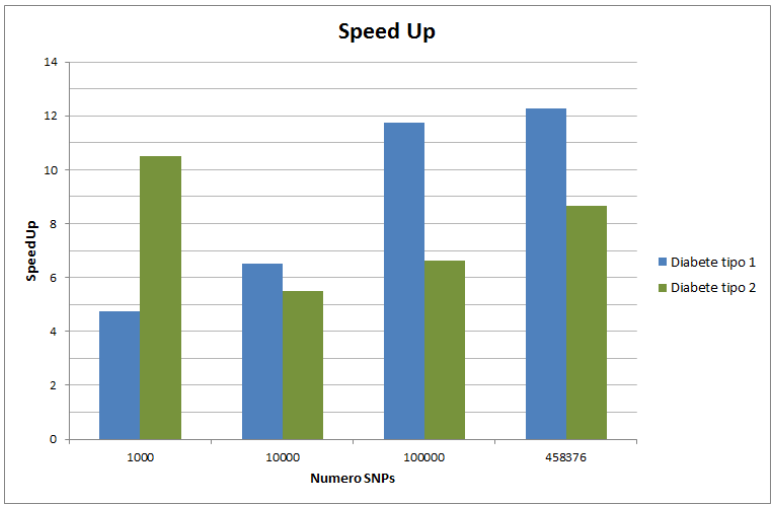

Figura 4.7.1: Speed Up per il diabete di tipo 1 e2

Come si può vedere dalle tabelle 4.31 e 4.32 e dalla figura 4.7.1 per entrambi i diabete si riscontrano miglioramenti complessivi dei tempi di esecuzione.

In particolare per il diabete di tipo 1 all'aumentare del numero di SNPs in input si ha un aumento dello speed up a causa principalmente della riduzione del numero di iterazioni su λ durante la k-fold cross-validation. A causa dello stesso motivo nel diabete di tipo 2 invece lo speed up maggiore si ha con $p = 1000$.

Utilizzando tutta la matrice X come input si ha quindi un valore di speed up pari a 12.27 per il diabete di tipo 1 e pari a 8.65 per il diabete di tipo 2.

4.7.2 Tempi di comunicazione e attesa

Numero di SNPs	Diabete di tipo 1	Diabete di tipo 2
$p = 1000$	35.39%	31.31%
$p = 10000$	36.10%	45.39%
$p = 100000$	5.48%	34.18%
$p = 458376$	4.88%	14.05%

Tabella 4.33: Percentuale di comunicazione e attesa utilizzando 10 nodi per il diabete di tipo 1 e 2.

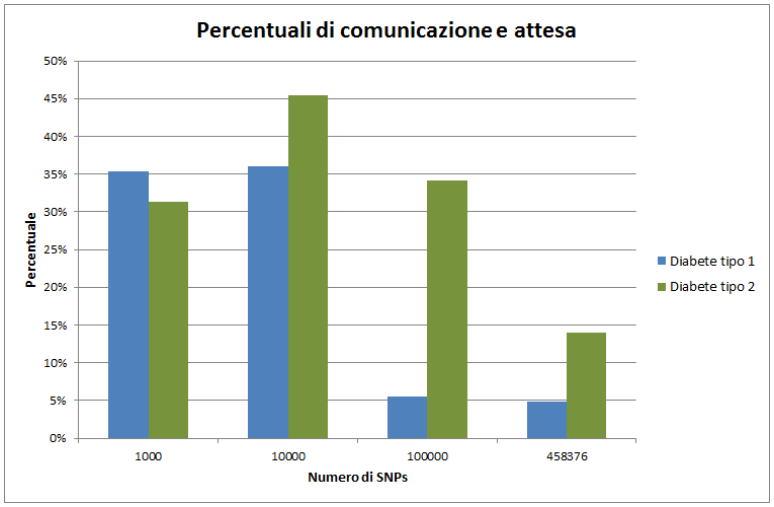

Figura 4.7.2: Percentuale di comunicazione e attesa utilizzando 10 nodi per il diabete di tipo 1 e 2.

La tabella 4.33 e la figura 4.7.2 descrivono le percentuali di tempo in cui i 10 nodi sono in comunicazione o in attesa di dati rispetto al tempo totale di esecuzione del programma.

Le percentuali maggiori sono dovute al maggior tempo di attesa dei 10 nodi. Queste attese si riscontrano principalmente in due punti durante l'esecuzione: nella parte della k-fold cross-validation quanto i 10 nodi si devono sincronizzare per passare al nodo 0 la percentuale di accuracy e nella parte della costruzione del modello finale quando devono i nodi slave devono passare al nodo master le sottomatrici calcolate. Ovviamente maggiori sono le iterazioni su λ nelle due parti è maggiore sarà la somma dei tempi di attesa.

Si ha quindi una bassa percentuale, intorno al 5%, per il diabete di tipo 1 con 100000 e 458376 SNPs dato che nella prima parte i 10 nodi lavorano senza grossi tempi di attesa e nella seconda parte il numero delle iterazioni su λ è dimezzato.

Per il diabete di tipo 2 invece si riesce ad avere una medio-bassa percentuale di comunicazione e attesa solo per $p = 459376$ dove i 10 nodi sono molto sincronizzati nella prima parte e le iterazioni su λsi riducono a 72 nella seconda.

In entrambi i tipi di diabete con $p = 10000$ non si riesce a ridurre le iterazioni su λ su nessuna delle due fasi. Per questo motivo la percentuale di comunicazione e attesa è maggiore degli altri tre valori di p.

Utilizzando tutta la matrice X come input si ha quindi una percentuale del tempo di comunicazione e attesa rispetto al tempo totale di esecuzione, escludendo il tempo necessario a leggere il vettore Y e la matrice X da file, pari a 4.88% per il diabete di tipo 1 e pari a 14.05% per il diabete di tipo 2.

4.8 Accuracy con iterazioni su λ ridotte

Bisogna infine controllare se la riduzione delle iterazioni su λ non peggiore l'accuracy finale, ma apporta solamente un'ottimizzazione in termini di tempo di esecuzione.

Si è quindi disabilito il comando per uscire anticipatamente dal ciclo su λ per controllare che il λ fosse lo stesso.

Numero di SNPs	Iterazione in cui l'esecuzione sarebbe terminata	Iterazione con accuracy massima
$p = 1000$	67	50
$p = 10000$	100	100
$p = 100000$	85	56
$p = 458376$	78	57

Tabella 4.34: Numero medio di cicli effettuati su λ su 3 test per il diabete di tipo 1 variando il numero p di SNPs disabilitando il comanda per uscire dal ciclo su λ, con $n = 3267$, $size = k = 10$ e $\alpha = 0.9$.

Numero di SNPs	Iterazione in cui l'esecuzione sarebbe terminata	Iterazione con accuracy massima
$p = 1000$	58	38
$p = 10000$	100	100
$p = 100000$	100	97
$p = 458376$	100	72

Tabella 4.35: Numero medio di cicli effettuati su λ su 3 test per il diabete di tipo 2 variando il numero p di SNPs disabilitando il comanda per uscire dal ciclo su λ, con $n = 3241$, $size = k = 10$ e $\alpha = 0.1$.

Come si può vedere dalle tabelle 4.34 e 4.35 l'uscita anticipata dal ciclo su λ determina solo un'ottimizzazione positiva in quanto se non ci fosse il modello finale sarebbe lo stesso ma il tempo di esecuzione totale aumenterebbe.

Capitolo 5

Conclusioni

In questa relazione è stato analizzato il pacchetto GLMNET che viene utilizzato per il calcolo delle funzioni di regressione logistica. Funzione utilizzata nei Genome-Wide Association Studies per l'identificazione del migliore insieme di SNPs che descrivono una malattia.

Dopo aver analizzato l'algoritmo è stata adattata la parte di codice presente nel pacchetto GLMNET per risolvere questo problema. Il primo passo è stato quello di identificare il collo di bottiglia e poi sono stati trovati altri possibili miglioramenti come la riduzione della iterazioni su λ. Tutto questo utilizzando due dataset di prova rappresentati dai diabeti di tipo 1 e 2.

Prima di effettuare le ottimizzazioni vere e proprie è stato individuato il valore di α migliore per ciascun tipo di diabete confrontando le accuratezze date da ogni modello ottenuto variando α sul test set. Per il diabete di tipo 1 è stato quindi scelto il valore $\alpha = 0.9$, mentre per il diabete di tipo 2 $\alpha = 0.1$. Il programma è stato infine ottimizzato attraverso la parallelizzazione della k-fold cross-validation per trovare il λ ottimo riducendone anche il numero di iterazioni, del collo di bottiglia individuato e del blocco di codice che calcola la normalizzazione della matrice X.

I risultati finali mostrano un'effettiva ottimizzazione del codice in termini di efficienza. Infatti, utilizzando 10 nodi e dando in input l'intera matrice X si ha un fattore di speed up pari a 12.27 per il diabete di tipo 1 e pari a 8.65 per il diabete di tipo 2.

Questi miglioramenti sono legati anche al fatto che la matrice X sta progressivamente aumentando di dimensioni e quindi in analisi future se si utilizzeranno matrice di input molto più grandi rispetto a quelle viste in questa relazione un miglioramento di 12 o 8 volte risulterà essere molto significativo.

Una fase successiva di questo progetto potrebbe essere il miglioramento di questo algoritmo in termini di efficacia per avere un'accuracy migliore.

Attualmente le percentuali di accuracy e i valori di MCC che si riescono ad ottenere con questo programma sono rispettivamente 80.56% e 0.5927 per il diabete di tipo 1 mentre 62.74% e 0.1734 per il diabete di tipo 2. Questi risultati sono in linea con quelli previsti, infatti per il diabete di tipo 2 è molto difficile stabilire se un soggetto è a rischio di avere quella malattia sulla base del suo DNA mentre è più facile ottenere un risultato corretto per il diabete di tipo 1.

Ringraziamenti

This study makes use of data generated by the Wellcome Trust Case-Control Consortium. A full list of the investigators who contributed to the generation of the data is available from http://www.wtccc.org.uk. Funding for the project was provided by the Wellcome Trust under award 076113 and 085475.

Bibliografia

[1] *Valgrind Documentation*. http://www.valgrind.org/.

[2] K. L. Ayers and H. J. Cordell. Snp selection in genome-wide and candidate gene studies via penalized logisic regression. *Genetic Epidemiology*, 34(8):879–891, 2010.

[3] G. Bruscagin. Studio di associazione genome-wide: preprocessing e selezione snps. Master's thesis.

[4] B. Di Camillo, I. Castiglioni, F. Sambo, M. C. Girardi, and G. M. Toffolo. Methods for discovery and integration of genetic and neuroimaging biomarkers. Dipartimento di Ingegneria dell'Informazione, Università di Padova. Istituto di Bioimmagini e Fisiologia Molecolare IBFM CNR, Segrate (MI).

[5] The Wellcome Trust Case Control Consortium. Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. *Nature*, 447(7145):661–678, 2007.

[6] K. Cook. Glmnet - ml4bio software presentation. University of Toronto.

[7] K. Dowd. *High performance computing*. O'Reilly, 2nd edition, 1998.

[8] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[9] S. Goedecker and A. Hoisie. *Performance Optimization of numerically intensive codes.* SIAM, 2001.

[10] W. Groop, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message passing interface.* MIT Press, 2^{nd} edition, 1999.

[11] T. Hastie. Fast regularization paths via coordinate descent. Stanford University.

[12] Q. He and D.-Y. Lin. A variable selection method for genome-wide association studies. *Bioinformatics*, 27(1):1–8, 2011.

[13] C. J. Hoggart, J. C. Whittaker, M. De Iorio, and D. J. Balding. Simultaneous analysis of all snps in genome-wide and re-sequencing association studies. *Plos Genetics*, 4(7), 2008.

[14] T. A. Manolio. Genome-wide association studies and assessment of the risk of disease. *The New England Journal of Medicine*, 2010.

[15] T. M. Mitchell. Generative and discriminative classifiers: naive bayes and logistic regression. In *Machine learning.* 2005.

[16] A. Y. Ng. Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance. *Proceedings of the* 21^{st} *International Conference on Machine Learning*, pages 78–85, 2004.

[17] A. Nuzzo and A. Malovini. Studi di associazione genetica. Università degli Studi di Pavia.

[18] T. A. Pearson and T. A. Manolio. How to interpret a genome-wide association study. *JAMA*, 299(11):1335–1344, 2008.

[19] T. Strachan and A. P. Read. *Genetica umana molecolare.* UTET, 3^{rd} edition, 2006.

[20] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2006.

[21] T. T. Wu, Y. F. Chen, T. Hastie, E. Sobel, and K. Lange. Genomewide association analysis by lasso penalized logistic regression. *Bioinformatics Advance Access*, 2009.