Università degli Studi di Milano Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

MCS-Progetto-1

Relazione del progetto d'esame di Metodi del Calcolo Scientifico

Relazione di progetto a cura di di:
Nicolò Ripamonti
Simone Ritucci
Lorenzo Rovida

Anno Accademico 2019–2020

Sommario

Lo scopo di questo progetto è di studiare l'implementazione in ambienti di programmazione open source del metodo di **Choleski** per la risoluzione sistemi lineari per matrici sparse, simmetriche e definite positive e successivamente confrontarli con l'implementazione nell'ambiente di programmazione di MatLab.

Immaginiamo che un'azienda abbia la necessità di munirsi di un ambiente di programmazione per risolvere sistemi lineari con matrici sparse e definite positive di grandi dimensioni utilizzando il metodo di Choleski.

L'alternativa è tra software proprietario (MatLab) oppure open source e anche tra Windows oppure Linux.

La richiesta è quindi quella di confrontare, in ambiente Linux e Windows, e sulla stessa macchina, l'ambiente di programmazione MatLab con una libreria open source.

In altre parole bisogna porsi le seguenti domande:

1. È meglio affidarsi alla sicurezza di MatLab pagando oppure vale la pena di avventurarsi nel mondo open source?

2. È meglio lavorare in ambiente Linux oppure in ambiente Windows?

Indice

1 | Introduzione

1.1 Introduzione

Lo scopo di questo progetto è quello di valutare l'implementazione, in ambienti di programmazione differenti, del metodo di Cholesky per la risoluzione di sistemi lineari per matrici sparse. La scelta degli ambienti, da parte del team, si è focalizzata su:

Matlab **C++** **R**

Le matrici che sono state considerate per la realizzazione del progetto sono quelle del gruppo *SuiteSparse Matrix Collection* che colleziona matrici sparse derivanti da applicazioni di problemi reali (ingegneria strutturale, fluidodinamica, elettromagnetismo, termodinamica, computer graphics/vision, network e grafi).

In particolare le matrici simmetriche e definite positive che sono state analizzate sono le seguenti:

Flan_1565 **StocF-1465** **cfd2** **cfd1** **G3_circuit**
parabolic_fem **apache2** **shallow_water1** **ex15**

Un sistema lineare si può rappresentare in forma matriciale come $A \cdot x = b$ dove:

- A è la matrice dei coefficienti del sistema ed è rappresentata da una delle matrici del gruppo *SuiteSparse Matrix Collection* elencate precedentemente.
- x è il vettore colonna delle incognite.
- b è il vettore colonna dei termini noti; b è scelto in modo che la soluzione esatta sia il vettore $xe = [111 \ldots 11]$ avente tutte le componenti uguali a 1, tale che $b = A \cdot xe$.

Il confronto dei vari ambienti di programmazione sui sistemi operativi di Windows e Linux deve avvenire in termini di:

Tempo: Calcolo dei tempi di esecuzione del metodo di Cholesky;

Accuratezza: Calcolo dell'errore assoluto tra la soluzione esatta e la soluzione del sistema lineare;

Impiego della memoria: Quantità di memoria utilizzata per il calcolo del sistema lineare con il metodo di Cholesky.

1.1.1 Matrici sparse

Le matrici sparse sono una classe di matrici con la caratteristica di contenere un significativo numero di elementi uguali a zero. Spesso il numero di elementi diversi da zero su ogni riga è un numero piccolo (per esempio del- l'ordine di 10^1) indipendente dalla dimensione della matrice, che può essere anche dell'ordine di 10^8.

Le matrici sparse si possono memorizzare in modo compatto, tenendo solo conto degli elementi diversi da zero; per esempio, è sufficiente per ogni elemento diverso da zero memorizzare solo la sua posizione ij e il suo valore a_{ij}, ignorando gli elementi uguali a zero. Ciò consente di ridurre il tempo di calcolo eliminando operazioni su elementi nulli.

1.2 Descrizione delle librerie

1.2.1 MatLab

Versione: 9.7.0

Referenze: Sito - Documentazione

MatLab (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica scritto in C, che comprende anche l'omonimo linguaggio di programmazione creato dalla MathWorks. MATLAB consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente, e interfacciarsi con altri programmi. MatLab è usato da milioni di persone nell'industria e nelle università per via dei suoi numerosi strumenti a supporto dei più disparati campi di studio applicati e funziona su diversi sistemi operativi, tra cui Windows, Mac OS, GNU/Linux e Unix.

1.2.2 Eigen C++

Versione: 3.7.7, Novembre 2019

Referenze: Sito - Documentazione

Eigen è una libreria di modelli C++ per l'algebra lineare, ovvero che supporta il calcolo e la risoluzione di matrici, vettori, solutori numerici e algoritmi correlati.

Supporta matrici di qualsiasi dimensione, dalle piccole matrici di dimensioni fisse alle matrici dense arbitrariamente grandi, fino alle matrici sparse e il suo ecosistema offre molte funzionalità specializzate come l'ottimizzazione non lineare, le funzioni di matrice, un solutore polinomiale e molto altro.

"L'implementazione di un algoritmo su Eigen è come copiare semplicemente lo pseudocodice."

Eigen ha un buon supporto per il compilatore mentre si esegue una suite di test per garantire affidabilità e aggirare eventuali bug del compilatore. Eigen è anche standard C ++ 98 e mantiene tempi di compilazione molto ragionevoli.

Eigen è un software open source concesso in licenza con Mozilla Public License 2.0 dalla versione 3.1.1.

In particolare di Eigen sono state utilizzate le funzioni:

`SimplicialLDLT< >` Questa classe fornisce fattorizzazioni LDL^T Cholesky di matrici sparse che sono definite positive. La fattorizzazione consente di risolvere $AX = B$ dove X e B possono essere densi o sparsi. Al fine di ridurre il riempimento, viene applicata una permutazione simmetrica P prima della fattorizzazione in modo tale che la matrice fattorizzata sia PAP^{-1};

`solve()` Funzione utilizzata per trovare la soluzione di x.

1.2.3 Matrix R

Versione: 1.2-18, Novembre 2018

Referenze: Sito - Documentazione

Il pacchetto di R *Matrix* fornisce un set di classi per matrici dense e sparse che estendono le classi base delle matrici già presenti in R. I metodi per un'ampia gamma di funzioni e operatori applicati agli oggetti di queste classi forniscono un accesso efficiente alla soluzione di sistemi lineari, matrici dense e matrici sparse.

Una caratteristica notevole del pacchetto è che ogni volta che una matrice viene fattorizzata, la fattorizzazione viene memorizzata come parte della matrice originale in modo che ulteriori operazioni sulla matrice possano riutilizzare questa fattorizzazione.

In particolare del pacchetto Matrix, sono state usate due funzioni:

`readMM()` Funzione utilizzata per leggere un file di tipo MatrixMarket. All'interno della funzione basta inserire il percorso del file;

`chol()` Funzione che calcola la fattorizzazione di Choleski di una matrice quadrata simmetrica e definita positiva. Gli argomenti che si possono inserire all'interno della funzione sono:

x Una matrice quadrata (sparsa o densa); se x non è definito positivo, viene segnalato un errore. per la nostra implementazione abbiamo utilizzato come argomento della funzione solamente la matrice;

pivot Valore logico che indica se si deve usare il pivot. Attualmente, questo non viene utilizzato per matrici dense.

cache Valore logico che indica se il risultato deve essere memorizzato nella cache; si noti che questo argomento è sperimentale e disponibile solo per alcune matrici sparse.

2 | Analisi dei risultati

2.1 Analisi del dataset

Analizzando la struttura del nostro dataset, possiamo notare che il minimo valore per il glucosio, per la pressione del sangue, per lo spessore della pelle, per il BMI e per l'insulina è uguale a 0; siccome è impossibile, per un essere umano, avere anche solo uno di questi valori a 0, questi ultimi vengono trasformati in NA.
A seguito di questa trasformazione, noteremo, attraverso il grafico seguente, la percentuale di valori NA presenti all'interno del nostro dataset.

Una volta effettuata tale sostituizione, il dataset viene pulito eliminando tutti i valori mancanti, con il pacchetto "mice" scaricabile all'interno di RStudio.

Di seguito vengono proposti i grafici relativi a determinate variabili predittiva del dataset per capire meglio il significato di ciascuna di esse.

Di seguito è riportato, invece, un grafico riassuntivo di ciascuna variabile predittiva con relative correlazioni tra di loro.

2.2 Descrizione della PCA

La *Principal Component Analysis* (PCA) è una procedura statistica che utilizza una trasformazione ortogonale per convertire un insieme di osservazioni di variabili numeriche eventualmente correlate in un insieme di valori di variabili linearmente non correlate chiamate componenti principali.
Questa trasformazione è definita in modo tale che il primo componente principale abbia la maggiore varianza possibile (vale a dire, tiene conto della maggior variabilità possibile nei dati) e ogni componente successivo ha a sua volta la maggiore varianza possibile sotto il vincolo che è ortogonale ai componenti precedenti.
I vettori risultanti sono un insieme di basi ortogonali non correlate.
PCA è sensibile al relativo ridimensionamento delle variabili originali.
La PCA viene utilizzata principalmente come strumento di analisi dei dati esplorativi e per la creazione di modelli predittivi, come nel caso in esame.
La PCA può essere eseguita mediante decomposizione di autovalori di una matrice di covarianza dei dati (o correlazione) o decomposizione di valori singolari di una matrice di dati, di solito dopo una fase di **standardizzazione** dei dati iniziali.
La standardizzazione di ciascun attributo consiste nel centrare la media (sottraendo ogni valore di dati dalla media misurata della sua variabile in modo che la sua media sia zero) e, possibilmente, normalizzando la varianza di ciascuna variabile per renderla uguale a 1.
I risultati di una PCA sono di solito discussi in termini di punteggi dei componenti, se tali punteggi sono standardizzati per la varianza unitaria, i caricamenti devono contenere la varianza dei dati (e questa è la grandezza degli autovalori).
PCA è la più semplice delle vere analisi multivariate basate su autovettori.
Spesso, il suo funzionamento può essere considerato come rivelazione della struttura interna dei dati in un modo che spiega meglio la varianza nei dati.

La PCA, applicata al nostro dataset, restituisce i seguenti risultati:

Da tali risultati, si nota che le dimensioni più importanti nella valutazione del target sono *n_pregnant*, *age* e *glucosio*.
Successivamente, notiamo il grafico delle varianze cumulate delle varie dimensioni restituite dalla PCA:

Mentre questo è il risultato restituito su console:

Per ottenere un risultato accettabile sono sufficienti solamente 5 delle 8 dimensioni; infatti, con queste ultime, si arriva ad una varianza cumulata del 83% (che è un ottimo risultato).

Nonostante ciò, dato il dataset, non c'è un vero bisogno di ridurre il numero di dimensioni in quanto 8 non è un valore così alto.

Infine, viene qui di seguito presentato il grafico che rappresenta il valore *cos2* per ogni individuo, maggiore è questo valore, migliore è la rappresentazione dell'individuo dai component principali.

3 | Confronto dei risultati

3.1 Descrizione dei modelli di Machine Learning scelti

3.1.1 Descrizione del modello Decision Tree

Un albero di decisione è un sistema con n variabili in input e m variabili in output; le variabili in input, ossia gli attributi, sono derivate dall'osservazione dell'ambiente.
Le ultime variabili in output, invece, identificano la decisione, o l'azione, da intraprendere.
Il processo decisionale è rappresentato con un albero logico rovesciato dove ogni nodo è una funzione condizionale.
Ogni nodo verifica una condizione, ossia un test, su una particolare proprietà dell'ambiente e ha due o più diramazioni verso il basso.
Il processo consiste in una sequenza di test: essa comincia sempre dal nodo radice, ovvero il nodo genitore situato più in alto della struttura, e procede verso il basso; la decisione finale si trova nei nodi foglia terminali, quelli più in basso.
In questo modo, dopo aver analizzato le varie condizioni, l'agente giunge alla decisione finale.
In altre parole, nel machine learning un albero di decisione è un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, che nell'albero è rappresentato dal cammino dal nodo radice al nodo foglia.
Normalmente un albero di decisione viene costruito utilizzando tecniche di apprendimento a partire dall'insieme dei dati iniziali (dataset), il quale può essere diviso in due sottoinsiemi: il training set sulla base del quale si crea la struttura dell'albero e il test set che viene utilizzato per testare l'accuratezza del modello predittivo così creato.
É stato scelto tale modello nello sviluppo del progetto perchè è adeguato per la predizione di valori binari e, nonostante la sua complessità rispetto ad altri

modelli sia elevata, il dataset non è eccessivamente pesante, per cui la sua esecuzione sul medesimo non crea problemi.

Si presenta di seguito un piccolo plot che rappresenta un esempio di albero di decisione applicato al dataset esaminato:

Questo invece rappresenta l'output su console:

Da qui si deduce come, in seguito ad una 10-Fold cross validation (discussa nel paragrafo 4.1), l'albero viene costruito su diversi livelli di complessità, ognuno con delle statistiche diverse.
L'algoritmo sceglie in modo automatico il livello migliore in base alle statistiche e restituisce l'albero costruito con questo livello di complessità.

3.1.2 Descrizione del modello SVM

Il Support Vector Machine (SVM) è un algoritmo di apprendimento automatico supervisionato che può essere utilizzato sia per scopi di classificazione che di regressione.
L'algoritmo SVM ottiene la massima efficacia nei problemi di classificazione binari, e questo rappresenta uno dei motivi per cui si è scelto di utilizzare tale modello per il nostro progetto: il nostro problema, infatti, è binario.
L'SVM è basato sull'idea di trovare un iperpiano che divida al meglio un set di dati in due classi.
Per un'attività di classificazione con solo due dimensioni spaziali x e y, un iperpiano è raffigurato come una linea che separa e classifica un insieme di dati.

I support vector, invece, sono i punti dati più vicini all'iperpiano.
Tali punti dipendono dal set di dati che si sta analizzando e, se vengono rimossi o modificati, alterano la posizione dell'iperpiano divisorio.
Per questo motivo, possono essere considerati gli elementi critici di un set di dati.

Il margine, infine, è definito come la distanza tra i vettori di supporto di due classi differenti più vicini all'iperpiano.

Alla metà di questa distanza viene tracciato l'iperpiano, o retta nel caso si stia lavorando a due dimensioni.
Il Support Vector Machine ha l'obiettivo di identificare l'iperpiano che meglio divide i vettori di supporto in classi.
Per farlo esegue i seguenti step:

- Cerca un iperpiano linearmente separabile o un limite di decisione che separa i valori di una classe dall'altro; se ne esiste più di uno, cerca quello che ha margine più alto con i vettori di supporto, per migliorare l'accuratezza del modello.
- Se tale iperpiano non esiste, SVM utilizza una mappatura non lineare per trasformare i dati di allenamento in una dimensione superiore (se siamo a due dimensioni, valuterà i dati in 3 dimensioni); in questo modo, i dati di due classi possono sempre essere separati da un iperpiano, che sarà scelto per la suddivisione dei dati.

Qui di seguito il risultato del lancio di SVM sul nostro training set:

3.1.3 Descrizione del modello delle reti neurali

Le reti neurali artificiali sono modelli matematici composti da neuroni artificiali di ispirazione alle reti neurali biologiche, quella umana o animale, e vengono utilizzate per risolvere problemi ingegneristici di Intelligenza Artificiale legati a diversi ambiti tecnologici come l'informatica, l'elettronica, la simulazione o altre discipline.
Volendo dare una definizione più dettagliata potremmo dire che le reti neurali sono modelli di calcolo matematico-informatici basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni; tali interconnessioni derivano da neuroni artificiali e processi di calcolo basati sul modello delle scienze cognitive chiamato "connessionismo".
Le reti neurali artificiali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione: ricevono segnali esterni su uno strato di nodi (che rappresenta l'unità di elaborazione, il processore); ognuno di questi nodi è collegato a svariati nodi interni della rete che sono organizzati a più livelli, in modo che ogni singolo nodo possa elaborare i segnali ricevuti trasmettendo ai livelli successivi il risultato delle sue elaborazioni.
Affinché questo processo risulti performante è necessario "addestrare" le reti neurali, ossia fare in modo che apprendano come comportarsi nel momento in cui andrà risolto un problema.
Vi sono tre grandi paradigmi di apprendimento, ciascuno corrispondente ad un particolare compito astratto di apprendimento.
Si tratta dell'apprendimento supervisionato, apprendimento non supervisionato e l'apprendimento per rinforzo; di solito, un tipo di architettura di rete può essere impiegato in qualsiasi di tali compiti.
Nel nostro caso, si parla di apprendimento supervisionato, in cui si dispone di un insieme di dati per l'addestramento (o training set) comprendente esempi tipici d'ingressi con le relative uscite loro corrispondenti: in tal modo la rete può imparare ad inferire la relazione che li lega.
Successivamente, la rete è addestrata mediante un opportuno algoritmo, il quale usa tali dati allo scopo di modificare i pesi ed altri parametri della rete stessa in modo tale da minimizzare l'errore di previsione relativo all'insieme d'addestramento.
L'obiettivo finale dell'apprendimento supervisionato è la previsione del valore dell'uscita per ogni valore valido dell'ingresso, basandosi soltanto su un numero limitato di esempi di corrispondenza.
Per fare ciò, la rete deve essere dotata di un'adeguata capacità di generalizzazione, con riferimento a casi ad essa ignoti; ciò consente di risolvere problemi di regressione o classificazione.
E' stato utilizzato questo modello perchè esso è estremamente efficiente per quanto riguarda problemi di classificazione binaria.

Qui di seguito il risultato del lancio di Rete neurale sul nostro training set:

4 | Codici sorgente

4.1 10-Fold Cross Validation

Che cos'è?

Sono state eseguite delle 10-Fold Cross Validation sui diversi modelli utilizzati.
La k-fold cross-validation consiste nella suddivisione del dataset totale in k parti di uguale numerosità e, ad ogni passo, la k-esima parte del dataset viene ad essere il validation dataset, mentre la restante parte costituisce il training dataset.
Così, per ognuna delle k parti si allena il modello, evitando quindi problemi di overfitting, ma anche di campionamento asimmetrico del training dataset, tipico della suddivisione del dataset in due sole parti (ovvero training e validation dataset).

In altre parole, si suddivide il campione osservato in gruppi di egual numerosità, si esclude iterativamente un gruppo alla volta e lo si cerca di predire con i gruppi non esclusi. Tutto ciò al fine di verificare la bontà del modello di predizione utilizzato.

Implementazione

All'interno dello script R, basta aggiungere un oggetto di tipo *trainControl* alla funzione *train()*.

Un oggetto *trainControl* è cosi definito:

Listing 4.1: Definizione oggetto trainControl

```
1  control = trainControl(method = "repeatedcv",
2      number = 10,
3      repeats = 3, classProbs = TRUE,
4      summaryFunction = twoClassSummary)
```

A questo punto, basterà aggiungerlo come attributo *trControl* ad una funzione *train()*, come di seguito:

Listing 4.2: Training con train control

```
model = train(targetColumn ~ .,
  data = trainset,
  method = 'svmRadial',
  metric = "ROC",
  trControl = control)
```

Utilizzando questa tecnica, abbiamo addestrato i diversi modelli (ovvero: Decision Tree, SVM e Neural Network).

4.2 Stima delle misure di performance

Le misure di performance vengono utilizzate per poter comprendere in che modo un modello di machine learning effettua una predizione su un set di test, valutandone i risultati rispetto a delle labels definite a priori.

Nel progetto abbiamo utilizzato diverse misure di performance per i modelli, nei prossimi paragrafi vengono descritte.

4.2.1 Matrice di confusione

La matrice di confusione è una rappresentazione dell'accuratezza di classificazione statistica.

Da una matrice di confusione si possono dedurre diverse statistiche: **sensitività** (ovvero la true positive rate, TPR), **specificità** (ovvero true negative rate, TNR) e **accuratezza** (vicinanza delle misurazioni a un valore specifico).

- Modello Albero decisionale:

- Modello SVM:

- Modello Rete neurale:

4.2.2 Precision, Recall, F-Measure

Queste tre misure, ricavabili da una matrice di confusione, vanno a dare ulteriori informazioni circa la predizione del modello.
La **precision** è la frazione di istanze rilevanti fra le istanze, mentre la **recall** è la frazione fra il numero di istanze rilevanti totali e il numero di istanze rilevanti trovate.
La **F-Measure** infine è la media armonica fra **precision** e **recall**.

- Modello SVM:

- Modello Rete neurale:

- Modello Albero decisionale:

Da queste statistiche si può notare che i modelli abbiano dei livelli (o statistiche) molto simili, differiscono al più di 0.1 unità.
Essendo questi valori superiori all'80% possiamo dire che i modelli hanno alti valori di precision, recall e F-M, e che tra questi non ci sia un migliore in termini assoluti.

4.2.3 ROC e AUC

Una curva ROC (receiver operating characteristic curve) è un grafico che mostra le performances di un modello di classificazione in ogni soglia di classificazione.

Si potrebbe dire che questa curva ci da informazioni su *quanto il modello sia in grado di distinguere fra le classi* (in questo caso T o F).

Il valore di AUC, compreso tra 0 e 1, equivale infatti alla probabilità che il risultato del classificatore applicato ad un individuo estratto a caso dal gruppo dei malati sia superiore a quello ottenuto applicandolo ad un individuo estratto a caso dal gruppo dei sani.

Questa curva mostra due parametri:

Come leggere la curva ROC

Per leggere la curva ROC, ha senso mostrare questo esempio:

Le curve rappresentano le distribuzioni di TN (true negative) e TP (true positive), la situazione ideale è quando queste distribuzioni non si interesecano, quindi la ROC è una curva che passa da (0,0), (0, 1) e (1, 1).
Quando queste si intersecano in parte, la ROC non sarà quella ideale, ma ciò dipende sempre da quanto l'intersezione sia grande.

SVM Per quanto riguarda il modello SVM, la curva ROC è rappresentata come segue:

Il valore di AUC è molto buono (0.82)

Rete neurale Per quanto riguarda il modello Rete neurale, la curva ROC è rappresentata come segue:

Anche in questo modello il valore di AUC è molto buono (0.83), SVM e NNet si comportano in maniera molto simile.

Albero decisionale Per quanto riguarda il modello Albero decisionale, la curva ROC è rappresentata come segue:

Per quanto riguarda l'albero, il valore di AUC è comunque buono (0.77) ma non all'altezza degli altri due modelli.
Vengono ora mostrati i tre modelli sullo stesso grafico, per avere un confronto netto:

Da tale grafico si può dedurre che Rete neurale e SVM si comportano in modo molto simile nel riconoscimento di TP e TN, mentre l'Albero di decisione, nonostante si trovi ad alti livelli, effettua una classificazione leggermente inferiori agli altri due.

Conclusioni

Dopo aver letto, analizzato e pulito il dataset, abbiamo iniziato i due processi di training su SVM e Rete neurale. Dopo aver analizzato le varie statistiche abbiamo notato che i due modelli, seppur molto buoni, non differiscono di molto. Abbiamo quindi deciso di implementare anche un modello di Albero di decisione. La scelta, però, non ha portato risultati significativamente migliori in quanto questo modello ha avuto statistiche addirittura inferiori a quelle dei predenti. Se si volesse stabilire il "migliore" fra i tre, probabilmente sarebbe la Rete neurale, che ha delle statistiche leggermente superiori agli altri.