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

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.

3 Confronto dei risultati

3.1 Descrizione dei modelli di Machine Learning scelti

3.1.1 Descrizione del modello Decision Tree

3.1.2 Descrizione del modello SVM

4 | Codici sorgente

4.1 10-Fold Cross Validation

Che cos'è?

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: 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

4.2.1 Matrice di confusione

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

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.