Tutorial per l'uso del codice

Debora Parisi, Stefania Perego

Marzo 2016

Indice

Capitolo 1

Introduzione

In questo codice è implementato l'algoritmo proposto nel rapporto tecnico di J. Chang, J. W. Fisher III "Parallel Sampling of HDPs using sub-clusters splits", NIPS, 2014 in cui si alternano passi di Gibbs sampler a passi di Metropolis-Hastings.
Nei passi di Gibbs sampler si considerano soltanto i clusters non vuoti e si aggiornano le quantità di interesse (β, π, θ) usando le *full-conditionals*; queste ultime non sono altro che alcuni step dell'algoritmo. Durante questi passi il numero di cluster rimane invariato.
Con i passi di Metropolis-Hastings, invece, si propone l'unione di due clusters (mosse di merge) oppure la divisione di un cluster (mosse di split). Per ogni cluster k si individuano kl e kr, che rispettivamente corrispondono al sub-cluster *sinistro* e *destro*; i nuovi clusters si propongono sulla base dei sub-clusters.
Questo algoritmo è utile nel *topic modeling*; come dataset scegliamo dunque delle collezioni di documenti.

Capitolo 2

Prerequisiti

Per la scrittura del codice è stato usato il C++11, mentre per le analisi a posteriori e la costruzione dei grafici è stato usato R. Per questo motivo è necessario installare:

- una versione di *R* superiore a 3.0
- i pacchetti di *R* *RInside*, *Rcpp*, *coda*, *wordcloud* e *RColorBrewer*

Per installare un pacchetto R è sufficiente aprire una sessione di R da terminale e digitare *install.packages("nome_pacchetto ")*.
R richiede che sia impostato correttamente il *workspace* per l'esecuzione degli script, quindi abbiamo fatto in modo che il codice sia funzionante solo se la cartella *APSC_Parisi_Perego* è posizionata nella home directory di Linux/Ubuntu.
Nel caso si decida di non mettere la cartella nella home, è necessario modificare la variabile **wd**, che contiene la working directory, nei seguenti file: */MainCategorica/main.cpp, /PosteriorAnalysis/Categorical.cpp, /CategoricalDatasetGenerator/main.cpp*; ad esempio:
wd="setwd(\"~/APSC_Parisi_Perego/MainCategorica/R_results ") "
diventa
wd="setwd(\"~/*nuovo_percorso*/APSC_Parisi_Perego/MainCategorica/R_results ") "
A causa dell'interfaccia con R, i *makefile* sono costruiti in modo da poter caricare le opportune librerie.

2.1 Parallelizzazione

Per la parallelizzazione del codice abbiamo scelto OpenMP. Per fissare il numero di thread occorre impostare al valore voluto la variabile d'ambiente

OMP_NUM_THREADS. Il programma si occuperà in automatico della sua lettura negli eseguibili parallelizzati. Se non viene specificato alcun valore di *OMP_NUM_THREADS* il valore di default è 1 e quindi l'algoritmo viene eseguito in seriale. Se il dataset è molto grande (ordine di 100000 dati) consigliamo di scegliere il numero massimo di thread a disposizione; nel caso di dataset meno numerosi è preferibile impostare un numero di thread pari al numero di clusters atteso o poco più.

2.2 Le cartelle

Il codice è organizzato come segue:

- *src*: in particolare in *include* sono contenuti tutti gli header file e le relative implementazioni dell'algoritmo, mentre in *RScripts* sono contenuti tutti i file per le analisi a posteriori con R e il file *PriorK.txt* che permette di fare una stima a priori del numero di cluster.
- *CategoricalDatasetGenerator*: contiene un file sorgente per l'esecuzione dell'algoritmo e un file sorgente per la creazione del dataset sintetico;
- *MainCategorica*: contiene il file sorgente per l'esecuzione dell'algoritmo, un dataset in *Dataset.txt*, il file *Variables.txt* che contiene informazioni sul dataset e il vocabolario delle parole distinte in *Vocabulary.txt*
- *PosteriorAnalysis*: contiene un file sorgente per eseguire le analisi a posteriori dei risultati prodotti dall'algoritmo
- Ogni cartella che contiene i file sorgente a sua volta contiene due cartelle, *cpp_results* e *R_results*: la prima conterrà i risultati dell'algoritmo e la seconda i risultati delle analisi a posteriori.

Capitolo 3

Come eseguire l'algoritmo

Nelle cartelle dove sono contenuti i file sorgente sono presenti anche i Makefile per la creazione degli eseguibili nella stessa cartella. Inoltre sono presenti i *DataFile.txt*, che servono per inserire le informazioni necessarie all'esecuzione dell'algoritmo.
Poichè dobbiamo inserire molte informazioni iniziali per lanciare l'algoritmo, ci è sembrato più veloce ed agevole utilizzare i DataFile piuttosto che l'interfaccia con l'utente, che sarebbe diventata un lungo elenco di domande e risposte. Nelle sezioni successive spiegheremo nel dettaglio come usare i Datafile e come creare gli eseguibili.

3.1 Scelta dei parametri

Prima di eseguire l'algoritmo è bene scegliere in modo corretto i parametri. Per aiutare l'utente, nella cartella *src/Rscript* è presente lo script R PriorK.txt. Ricordiamo che per compilare i file su R basta copiare il contenuto del file e incollarlo sul terminale in cui è aperta la sessione di R.

3.2 CategoricalDatasetGenerator

In questa cartella è necessario creare un dataset per testare l'algoritmo, se non è già stato fatto. Abbiamo lasciato i file *GeneratorDataFile.txt* per generare il dataset e *DataFile.txt* per far partire l'algoritmo già compilati con dei valori ragionevoli, per testare il codice ed avere degli output ragionevoli, altrimenti se si vogliono cambiare le informazioni inserite seguire le istruzioni sotto elencate. Per creare un dataset con le caratteristiche scelte, è necessario modificare *GeneratorDataFile.txt*:

```
#Cluster Veri
K = 10

#Valore di Gamma
gamma = 10.0

#Numero di documenti
D = 5

#Valore di Alpha
alpha = 1.0

#Dimensione dell'iperparametro della distribuzione del parametro latente
#Nel topic modeling, numero di parole distinte
W = 10

#Iperparametro dei parametri latenti
lambda = 0.5

#Numero di parole in ogni documento
Nj = 50
```

Per creare il dataset, nel Makefile sono presenti le seguenti regole:

- *build_dataset* compila il sorgente; a questo punto, se si desidera creare un dataset con i valori di default basta scrivere sul terminale **./dataset**, mentre **./dataset -f** per crearlo con i valori scelti da DataFile
- *run_dataset* crea e compila il sorgente con le impostazioni scelte.

L'algoritmo produce tre file:

- *Dataset.txt*, che contiene il dataset
- *Variables.txt*, che contiene informazioni sul dataset, cioè numero documenti, numero parole distinte, numero totale di dati
- *Vocabulary.txt*, che contiene il vocabolario delle parole distinte. L'algoritmo genera solo gli id delle parole, che sono dei numeri, se si vogliono le parole vere e proprie consigliamo di andare in questo sito: `http://www.scriptvari.altervista.org/generatore_testo_casuale.php` e copiare le parole nel file togliendo i numeri. Non è necessario che su ogni riga ci sia una parola.

I valori di default per la generazione del dataset sono: $K = 5$, $\gamma = 10.0$, $D = 5$, $\alpha = 1.0$, $W = 100$, $\lambda = 0.5$, $N_j = 50$.

Per eseguire l'algoritmo, per prima cosa è necessario modificare *DataFile.txt*:

```
#Fixed Alpha
is_alpha = 1
a = 5.0

#Prior on Alpha
is_alpha_prior = 0
aa = 2.0
ab = 2.0

#Fixed Gamma
is_gamma = 1
g = 10.0

#Prior on Gamma
is_gamma_prior = 0
ga = 2.0
gb = 0.1

#Lambda
is_lambda = 1
l = 0.5
Dim = 20

#Seed
is_seed = 1
seed = 23

#Initial K
is_K_init = 1
K_init = 30

#Iterations of Algoritm and Subcluster Iterations
Iter = 1000
SubIter = 1

#LPML
is_LPML = 0
burnin  = 0
```

Facciamo qualche esempio di scelta dei parametri :

- Parametri fissi
  is_alpha=1
  is_gamma=1
  is_alpha_prior=0
  is_gamma_prior=0

is_lambda=1
e scegliere valori da assegnare ad α,γ e λ

- Prior sui parametri
  is_alpha=0
  is_gamma=0
  is_alpha_prior=1
  is_gamma_prior=1
  is_lambda=1
  e scegliere i valori **aa** e **bb** di $\alpha \sim \mathcal{G}(aa, bb)$,**ga** e **gb** di $\gamma \sim \mathcal{G}(ga, gb)$ e il valore costante di λ

- Parametri di default
  is_alpha=0
  is_gamma=0
  is_alpha_prior=0
  is_gamma_prior=0
  is_lambda = 0
  si consiglia di non usarla, è sempre meglio impostare i parametri in base al dataset.

Gli altri parametri da impostare sono

- **Dim** è la dimensione dell'iperparametro nella distribuzione del parametro latente, in questo caso corrisponde al numero di parole distinte

- **K** è il numero di cluster iniziali

- **Seed**, è il seme, una volta scelto rimane fissato durante l'esecuzione dell'algoritmo. Se invece si decide di utilizzare un seme aleatorio basta impostare **is_seed**=0

- **Iter** e **SubIter** sono rispettivamente il numero di iterazioni dell'algoritmo e il numero di iterazioni per campionare i subcluster; si consiglia di impostare **SubIter** almeno a 100

- **is_LPML** si imposta ad 1 se si vuole monitorare il modello con l'indice LPML, 0 altrimenti

- **burnin** numero di iterazioni iniziali da scartare nel calcolo di LPML.

Per creare l'eseguibile dell'algoritmo ci sono due possibilità: fare l'analisi a posteriori al termine dell'algoritmo oppure eseguirla in un secondo momento nella directory *PosteriorAnalysis*. La scelta viene fatta con la variabile **IFR**.

Se **IFR**=no, l'analisi a posteriori non viene eseguita subito.
Nel Makefile sono presenti le seguenti regole:

- *build_main*, compila il sorgente
- *IFR=no build_main*, compila il sorgente
- *run_main*, compila ed esegue
- *IFR=no run_main*, compila ed esegue
- *run_all*, compila i sorgenti del dataset e dell'algoritmo e li esegue con i valori scelti
- *clean*, cancella l'eseguibile
- *distclean*, cancella l'eseguibile e tutti i file che sono stati creati dall'algoritmo e dall'eventuale analisi

Per la spiegazione dell'analisi a posteriori rimandiamo alla sottosezione 3.3.1

3.3 PosteriorAnalysis

In questa cartella è presente il sorgente per poter fare solo l'analisi a posteriori. In pratica si spostano i risultati dell'algoritmo in *cpp_results* di PosteriorAnalysis. Nel Makefile vi sono le seguenti regole:

- *DIRECTORY=nome_directory* build_<nome_modello> compila il sorgente
- *DIRECTORY=nome_directory* run_<nome_modello> compila ed esegue
- *distclean* cancella l'eseguibile e tutti i file che sono in *cpp_results* e *R_results*

dove con *nome_directory* intendiamo la directory da dove prendere gli output dell'algoritmo, mentre con <*nome_modello*> si mette il nome del main che si vuole compilare. Nell'eventualità di inserimento di un nuovo modello, quando si crea il file per fare l'analisi a posteriori è consigliabile aggiungere una regola nel Makefile che abbia lo stesso nome del sorgente.

Nella cartella *Example* abbiamo lasciato degli output per testare PosteriorAnalysis, perciò basta scrivere:
make DIRECTORY=Example run_categorical

3.3.1 Categorical.cpp

Durante l'esecuzione vengono poste una serie di domande per poter impostare correttamente l'analisi a posteriori del modello Dirichlet-Categorica. In particolare, si chiede di impostare il Burnin e il Thinning per l'analisi delle catene, il numero di cluster da visualizzare e il numero di parole per creare i wordcloud. Inoltre si chiede di controllare gli output nella cartella *R_results* e se si è soddisfatti o si vuole ripetere l'analisi.
Abbiamo notato che, alcune volte, i wordcloud non vengono generati; pensiamo sia un problema del pacchetto *wordcloud*. E' comunque sufficiente rispondere no alla domanda *Would you like to change something?* e ripetere la generazione.

3.4 MainCategorica

In questa cartella è presente il sorgente per testare l'algoritmo con il dataset reale KOS, che consiste in 750 documenti, 6366 parole distinte, 96942 parole in totale. Inoltre sono presenti i file:

- *Dataset.txt*, contiene i dati nel formato: DocID WordID nID, rispettivamente l'id del documento, l'id della parola distinta, il numero di volte che WordID è presente in DocID
- *Variables.txt*, contiene le seguenti informazioni: numero di documenti, numero di parole distinte e numero totale di dati
- *Vocabulary.txt*, contiene il vocabolario delle parole distinte

Abbiamo lasciato DataFile compilato con i valori che usati per testare l'algoritmo. Per simulazioni con altri dati si deve modificare il DataFile come in sezione 3.2.

```
#Fixed Alpha
is_alpha = 1
a = 5.0

#Prior on Alpha
is_alpha_prior = 0
aa = 2.0
ab = 2.0

#Fixed Gamma
is_gamma = 1
g = 10.0
```

```
#Prior on Gamma
is_gamma_prior = 0
ga = 2.0
gb = 0.1

#Lambda
is_lambda = 1
l = 0.5
Dim = 20

#Seed
is_seed = 1
seed = 23

#Initial K
is_K_init = 1
K_init = 30

#Iterations of Algoritm and Subcluster Iterations
Iter = 1000
SubIter = 1

#LPML
is_LPML = 0
burnin  = 0
```

I risultati dell'algoritmo verrano salvati in *cpp_results*. Anche qui è possibile scegliere se si vuole subito l'analisi a posteriori oppure no. Nel Makefile sono presenti le seguenti regole:

- *build*, compila il sorgente con l'analisi a posteriori inclusa
- *IFR=no build*, compila il sorgente senza analisi a posteriori
- *run*, compila ed esegue
- *IFR=no run*, compila ed esegue
- *clean*, cancella l'eseguibile
- *distclean* cancella l'eseguibile e tutti i file che sono stati creati dall'algoritmo e dall'eventuale analisi

Per l'interazione con l'utente nell'analisi a posteriori, rimandiamo a sottosezione 3.3.1. Per poter testare il funzionamento dell'analisi a posteriori in *PosteriorAnalysis*, abbiamo lasciato i risultati prodotti da una simulazione con il dataset KOS.

Capitolo 4

Gestione file binari

Per facilitare un' eventuale estensione del codice descriviamo come sono strutturati i file binari che rimangono comuni a tutti i modelli.
LastBeta.bin

```
*It: Iterazione a cui inizia il salvataggio
*K: Numero di clusters che si sono manifestati all'iterazione It
*vector<double> B: vettore dei pesi dei clusters che
                   si sono manifestati all'iterazione It
*K: Numero di clusters che si sono manifestati all'iterazione It+1
*vector<double> B: vettore dei pesi dei clusters che
                   si sono manifestati all'iterazione It +1
*...
*...
*K: Numero di cluster che si sono manifestati all'iterazione MaxIt
*vector<double> B: vettore dei pesi dei clusters che
                   si sono manifestati all'iterazione MaxIt
```

CPO.bin

```
*vector<double> CPO: vettore di dimensione pari alla dimensione
                     del dataset.
```

Labels.bin
La dimensione di questo file dipende dal numero di iterazioni dell'algoritmo. Se si eseguono più di 10000 iterazioni, salviamo le etichette dei dati delle ultime 10000 iterazioni, altrimenti iniziamo a salvare fin da subito. Quindi salvo 10000 oppure un numero di vettori pari al numero di iterazioni eseguite, questi vettori hanno dimensione pari a N.

```
*unsigned int labels: etichetta del primo dato all'iterazione It
*unsigned int labels: etichetta del secondo dato all'iterazione It
*...
*unsigned int labels: etichetta del N-esimo dato all'iterazione It
*unsigned int labels: etichetta del primo dato all'iterazione It+1
*unsigned int labels: etichetta del secondo dato all'iterazione It+1
*...
*unsigned int labels: etichetta del N-esimo dato all'iterazione It+1
```

```
*...
*...
*unsigned int labels: etichetta del primo dato all'iterazione MaxIt
*unsigned int labels: etichetta del secondo dato all'iterazione MaxIt
*...
*unsigned int labels: etichetta del N–esimo dato all'iterazione MaxIt
```

I metodi che producono i file **LastTheta.bin** e **Data.bin** dipendono invece dal modello scelto, perciò tali file avranno una struttura specifica per il tipo di dato e di parametro latente.