Tutorial per l'uso del codice

Debora Parisi Stefania Perego

29 novembre 2016

Indice

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 cluster 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 cluster (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 cluster si propongono sulla base dei sub-cluster.

Questo algoritmo è utile nel *topic modeling*; come dataset scegliamo dunque delle collezioni di documenti.

Capitolo 1

Prerequisiti

Per la scrittura del codice è stato usato il `C++11`, mentre per le analisi a posteriori e la creazione 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.

1.1 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.

1.2 Le cartelle

Il codice è organizzato come segue:

- *src*: in particolare in *include* sono contenuti tutti gli header file e quindi l'implementazione delle classi, mentre in *RScript* 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 di dataset simulati;
- *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 2

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 data file piuttosto che l'interfaccia con l'utente, che sarebbe diventata un lungo elenco di domande e risposte. Nei paragrafi successivi spiegheremo nel dettaglio come usare i data file e come creare gli eseguibili.

2.1 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 eseguire gli script su `R` basta copiare il contenuto del file e incollarlo sul terminale in cui è aperta la sessione di `R`.

2.2 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*:

```
#Numero di cluster
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 della distribuzione 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 data file;
- *run_dataset* compila il sorgente e lancia l'eseguibile 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 parole;
- *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 visitare questo sito: `http://www.scriptvari.altervista.org/generatore_testo_casuale.php` e copiare le parole nel file togliendo i numeri. Non è necessario che ci sia una parola per riga.

I valori di default per la generazione del dataset sono: $K = 10$, $\gamma = 4.0$, $D = 40$, $\alpha = 0.25$, $W = 100$, $\lambda = 0.5$, $N_j = 50$.

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 = 8.0

#Fixed Gamma
is_gamma = 1
g = 10.0

#Prior on Gamma
is_gamma_prior = 0
ga = 2.0
gb = 0.2

#Lambda
is_lambda = 1
l = 0.5
Dim = 100

#Seed
is_seed = 1
seed = 13

#Initial K
is_K_init = 1
K_init = 30

#Iterations and Sub-cluster Iterations
```

```
Iter = 10000
SubIter = 1

#LPML
is_LPML = 0
burnin  = 90

#Analisi documenti
is_real = 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 di massa
  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 fisso 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 sub-cluster; si consiglia di impostare **SubIter** almeno a 90;
- **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;
- **is_real** serve per l'analisi a posteriori, vale 1 se si vuole associare i documenti ai cluster stimati, 0 altrimenti.

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 al paragrafo 2.3.1.

2.3 PosteriorAnalysis

In questa cartella è presente il sorgente per 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 main.

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

2.3.1 Categorical.cpp

Durante l'esecuzione vengono poste una serie di domande per poter impostare correttamente l'analisi a posteriori dei risultati, nel caso in cui i dati abbiamo verosimiglianza categoriale e la prior sui parametri lateni sia la distribuzione di Dirichlet. In particolare, si chiede di impostare il Burnin e il Thinning per l'analisi delle catene e di controllare gli output nella cartella *R_results*, se si è soddisfatti o si vuole ripetere l'analisi. Nel caso il dataset analizzato sia reale, è necessario che nella cartella *PosteriorAnalysis* ci sia un file *DocNames.txt* che contenga i nomi dei documenti, per associarli ai cluster stimati; viene chiesto se il file è stato messo nella cartella e, successivamente, si chiede di impostare una soglia. Infine, si chiede quanti topic e quante parole per topic visualizzare tramite wordcloud. Al termine dell'esecuzione, l'analisi delle catene e i wordcloud si trovano nella cartella *PosteriorAnalysis/R_results*, mentre l'analisi dei documenti e il

least-square clustering si trovano, rispettivamente, nei file *DocAnalysis.txt* e *LeastSquare.txt* nella cartella *PosteriorAnalysis*.

2.4 MainCategorica

In questa cartella è presente il sorgente che lancia l'algoritmo. 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 parole;
- *Vocabulary.txt*, contiene il vocabolario delle parole distinte.

Abbiamo lasciato un dataset simulato composto da $D = 40$ documenti, $N_j = 50$ parole per documento, quindi $N = 2000$ parole in totale e $W = 100$ parole distinte e anche un data file con le impostazioni per l'algoritmo. Per simulare con altri parametri si deve modificare il data file come in 2.2.

```
#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 = 0.2

#Lambda
is_lambda = 1
```

```
l = 0.5
Dim = 100

#Seed
is_seed = 1
seed = 13

#Initial K
is_K_init = 1
K_init = 30

#Iterations of Algoritm and Subcluster Iterations
Iter = 10000
SubIter = 90

#LPML
is_LPML = 0
burnin  = 0

#Analisi documenti
is_real = 0
```

I risultati dell'algoritmo verrano salvati in *cpp_results*. Anche qui è possibile scegliere se si vuole eseguire 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 al paragrafo 2.3.1.

Capitolo 3

Gestione file binari

Per facilitare un' eventuale estensione del codice descriviamo come sono strutturati i file binari che rimangono comuni a tutti i modelli. Ad ogni iterazione vengono salvate su file le seguenti quantità β, π, θ e le etichette assegnati ai dati, non salvati su un unico file, si inizia a salvare dopo 1000 iterazioni. Su ogni file sono salvate 500 iterazioni. Dopo le 500 iterazioni si salva su un nuovo file.

Beta¡it¿.bin

Il primo file su cui si inizia a salvare è *Beta1000.bin*, dopo 500 iterazione il nuovo file è *Beta1500.bin* e così via. Sul file binario i dati sono salvati nel modo seguente:

```
*vector<double> Beta: vettore dei pesi dei clusters che
                      si sono manifestati all'iterazione It
*vector<double> Beta: vettore dei pesi dei clusters che
                      si sono manifestati all'iterazione It +1
*...
*...
*vector<double> Beta: vettore dei pesi dei clusters che
                      si sono manifestati all'iterazione MaxIt
```

Pi¡it¿.bin

Il primo file su cui si inizia a salvare è *Pi1000.bin*, dopo 500 iterazione il nuovo file è *Pi1500.bin* e così via. Ad ogni iterazione per ogni documento si salvano i pesi dei cluster che si sono manifestati in esse. Sul file binario i dati sono salvati nel modo seguente:

```
*vector<double> Pi1:vettore dei pesi dei cluster del documento 1
che si sono manifestati all'iterazione It
*vector<double> Pi2:vettore dei pesi dei cluster del documento 2
che si sono manifestati all'iterazione It
*...
*vector<double> PiD:vettore dei pesi dei cluster del documento D
che si sono manifestati all'iterazione It
*...
*...
*vector<double> Pi1:vettore dei pesi dei cluster del documento 1
che si sono manifestati all'iterazione It+1
*...
*vector<double> PiD:vettore dei pesi dei cluster del documento D
che si sono manifestati all'iterazione It+1
*...
*vector<double> Pi1:vettore dei pesi dei cluster del documento 1
che si sono manifestati all'iterazione MaxIt
*...
*vector<double> PiD:vettore dei pesi dei cluster del documento D
che si sono manifestati all'iterazione MaxIt
```

Theta¡it¿.bin

Il primo file su cui si inizia a salvare è *Theta1000.bin*, dopo 500 iterazione il nuovo file è *Theta1500.bin* e così via. Ad ogni iterazione, per ogni cluster si salvano i parametri latenti θ_k. Sul file binario i dati sono salvati nel modo seguente:

```
*THETA Theta1: parametro latente del cluster 1 all'iterazione It
*THETA Theta2: parametro latente del cluster 2 all'iterazione It
*...
*THETA ThetaK: parametro latente del cluster K all'iterazione It
*THETA Theta1: parametro latente del cluster 1 all'iterazione It +1
*...
*THETA ThetaK: parametro latente del cluster K all'iterazione It+1
*THETA Theta1: parametro latente del cluster 1 all'iterazione MaxIt
*...
*THETA ThetaK: parametro latente del cluster K all'iterazione MaxIt
```

Labels¡it¿.bin

Il primo file su cui si inizia a salvare è *Labels1000.bin*, dopo 500 iterazione il nuovo file è *Labels1500.bin* e così via. Ad ogni iterazioni sono salvate le

etichette assegnate ad ogni dato. Sul file binario i dati sono salvati nel modo seguente:

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

CPO.bin

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

Capitolo 4

Come aggiungere altre classi

Nel caso in cui si voglia utilizzare l'algoritmo SC-HDP con un altro modello per prima cosa è necessario modificare il file *Type.hpp*. Bisogna aggiungere la classe template per definire la nuova verosimiglianza e i nuovi tipi che saranno utilizzati nelle altre classi. Questa classe template è parametrizzata da `DIM`, che inidica la dimensione del dato.

```
template < unsigned int DIM=1> class <name_of_new_likelihood>{
public:
    //Tipo che gestisce i parametri latenti dei cluster sub-cluster
    using THETA = //inizializzare il tipo dei parametri latenti

    //Tipo del dato singolo
    using Point = //inizializzare il tipo del dato ;

    //Tipo che gestisce gli iperparametri dei parametri lantenti dei cluster
         o sub-cluster
    using HYP = //inizializzare il tipo delle statistiche;

    //Tipo che deve memorizzare e gestire le statistiche, aggiornamenti
        degli iperparametri dei parametri latenti dei cluster o sub-cluster
    using STAT = //tipo che gestisce i conteggi per aggiornare iperparametri
 };
```

Una volta definita questa nuova classe è necessario specializzare le classi *Cluster*, *Model* e *Document*. Per questa classi è stata definita un interfaccia generica in cui tutti i metodi sono dichiarati `null`. Nel momento in cui si aggiunge una classe è necessario implementare tutti i metodi e specializzare la classe.

È presente anche un file *Struct.hpp* in cui sono raccolte le strutture che vengono utilizzate nei metodi di *HDP_MCMC.hpp*. Si è scelto di creare un file a parte con la dichiarazione delle strutture, perchè il loro utilizzo è comune a più metodi. Nel momento in cui si aggiunge un nuovo modello è necessario aggiornare la lista che rinomina i tipi dei dati. Attualemente sono presenti:

```
using POINT = TypeCategorical<1>::Point;
using STAT = TypeCategorical<1>::STAT;
```

Nel caso in cui si aggiunge il nuovo modello, si commentano le dichiarazioni degli altri modelli e si aggiunge quella del modello che si vuole utilizzare.

```
//using POINT = TypeCategorical<1>::Point;
//using STAT = TypeCategorical<1>::STAT;
using POINT = <name_of_new_likelihood><DIM>::Point;
using STAT = <name_of_new_likelihood><DIM>::STAT;
```

Per maggiori informazioni si consiglia di controllare la documentazione del codice.