Supporting Information 2 Brief tutorial on using the presence/absence Latent Dirichlet Allocation model

Denis Valle

October 2018

Introduction

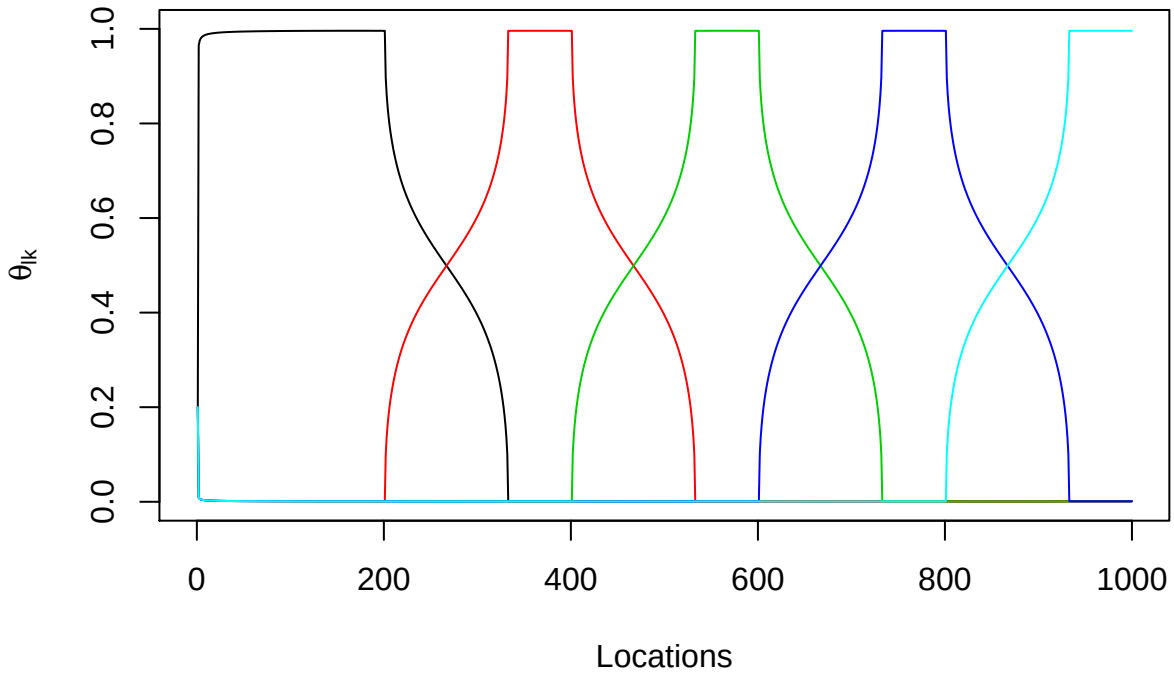
In this tutorial, we reproduce the model comparison results described in the main text for the simulated data with 5 groups. To this end, we first describe how the simulated data were generated to then fit the model and display its results.

Generating parameters

We start by generating the true parameters θ_{lk} and ϕ_{ks}. We assumed a total of 1000 locations, 200 species, and 5 groups.

```
rm(list=ls(all=TRUE))
set.seed(4)

nloc=1000 #number of locations
nspp=200 #number of species
ncommun=5 #number of groups
base=floor(nloc/(ncommun-2))
nobs=5 #number of observation per location

#generate thetas
x=seq(from=-1,to=1,length.out=base)
y=sqrt(1-(x^2))*0.1
min1=0.0001
y[y<min1]=min1

init=floor(nloc/ncommun)
seq1=c(seq(from=1,to=nloc,by=init),nloc)

theta=matrix(min1,nloc,ncommun)
for (i in 1:ncommun){
  seq2=seq1[i]:(seq1[i]+base-1)
  seq3=seq2[seq2<=nloc]
  theta[seq3,i]=y[1:length(seq3)]
}
theta=theta/matrix(apply(theta,1,sum),nloc,ncommun)
theta.true=theta
```

```
#visualize thetas
plot(NA,NA,xlim=c(0,nloc),ylim=c(0,1),xlab='Locations',ylab=expression(theta[lk]))
for (i in 1:ncommun) lines(1:nloc,theta[,i],col=i)
```




```
#generate the id for each location (loc.id)
loc.id=rep(1:nloc,each=nobs) #nobs observations for each location

#generate phi's
phi=matrix(NA,ncommun,nspp)
phi[]=rbeta(ncommun*nspp,0.5,0.5)
phi[,1:ncommun]=diag(1,ncommun) #to ensure that groups have distinct species composition
phi.true=phi
```

Simulating data

In this section, we use the parameters created above to generate the simulated data using the marginal Bernoulli likelihood described in the Supporting Information file 1. In this code, "dat" is a binary matrix where rows are different observations and columns are different species.

```
#calculate probabilities after integrating out the z's
probs=theta%*%phi
# image(probs)

#generate data for each location
```

```
dat=matrix(NA,nloc*nobs,nspp)
oo=1
for (i in 1:nloc){
  for (j in 1:nobs){
    dat[oo,]=rbinom(nspp,size=1,prob=probs[i,])
    oo=oo+1
  }
}
colnames(dat)=paste('spp',1:nspp,sep='')
```

```
setwd('U:\\presence absence model\\appendix tutorial')
dat1=cbind(dat,loc.id)
write.csv(dat1,'fake data.csv',row.names=F)
```

Fitting the model

To fit this model, we rely on the function "lda.presence.absence". This function requires the user to provide:

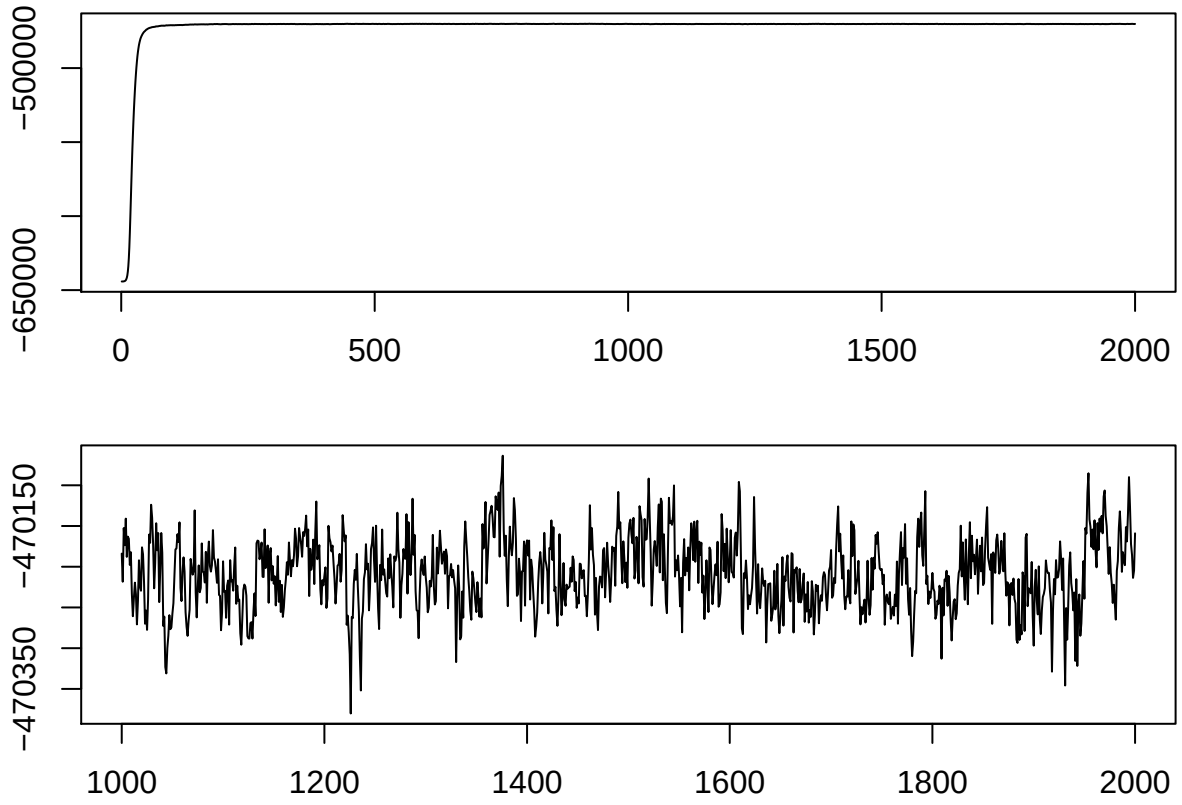
- the data (*dat*): each cell should contain a binary variable x_{isl} indicating if species s was present in unit l during observation i. In this table, rows correspond to different observation and columns correspond to different species,
- a vector with information on which rows are associated with which sampling unit (*id*): this is relevant when multiple observations/rows have arisen from the same sampling unit,
- maximum number of groups (*ncomm*),
- number of Gibbs sampler iterations (*ngibbs*), and
- prior hyper-parameters (*a.phi,b.phi,gamma*).

The different files containing the code that underlies this function are provided in the end of this document.

```
library('Rcpp')
set.seed(4)

#get functions
setwd('U:\\GIT_models\\github-presence_absence-SB')
source('gibbs functions.R')
source('gibbs sampler main function.R')
sourceCpp('aux1.cpp')

#run Gibbs sampler
results=lda.presence.absence(dat=dat,id=loc.id,ncomm=10,
                             a.phi=1,b.phi=1,gamma=0.1,ngibbs=2000)
```

The traceplot of the log-likelihood suggests that the algorithm has converged after iteration 1000.

```
seq1=1000:2000
par(mfrow=c(2,1),mar=c(3,3,1,1))
plot(results$llk,type='l',xlab='Iterations',ylab='Log-likel.')
plot(x=seq1,y=results$llk[seq1],type='l',xlab='Iterations',ylab='Log-likel.')
```

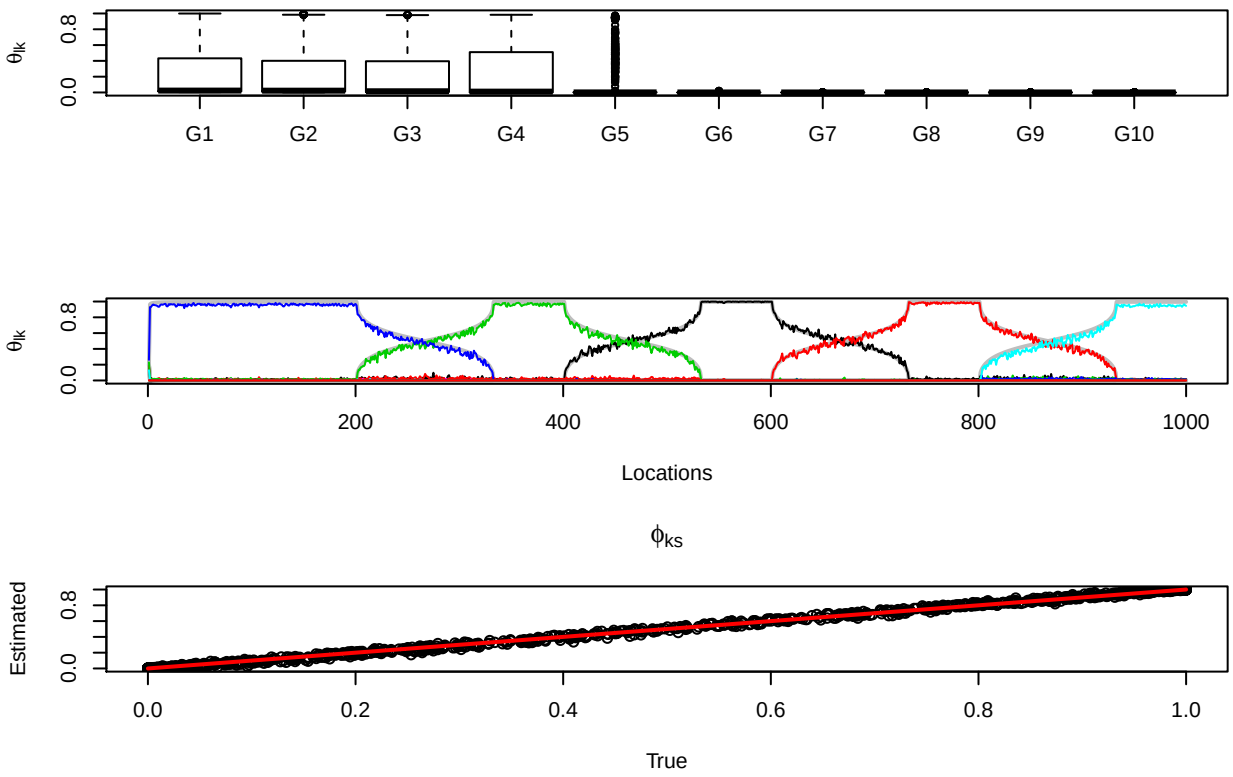


Therefore, we summarize our parameters by averaging the posterior samples from iteration 1000 onwards.

```
#summarize
theta=matrix(colMeans(results$theta[seq1,]),nloc,10)
phi=matrix(colMeans(results$phi[seq1,]),10,nspp)
```

Finally, we compare the estimated parameters to the true parameter values, finding that our algorithm is accurately able to determine the correct number of groups (top panel) and retrieve the original parameters (estimated θ_{lk} and ϕ_{ks} are shown in the middle and bottom panels, respectively).

```
par(mfrow=c(3,1),mar=c(4,4,4,1))
boxplot(theta,xlab='',ylab=expression(theta[lk]),names=paste('G',1:10,sep=''),
        xlab='Groups')
plot(NA,NA,xlim=c(0,1000),ylim=c(0,1),xlab='Locations',ylab=expression(theta[lk]))
for (i in 1:ncol(theta.true)) lines(theta.true[,i],col='grey',lwd=2)
for (i in 1:10) lines(theta[,i],col=i)

#re-order so that the estimated groups match the true groups
order1=c(4,3,1,2,5)
z=c(0,1)
plot(phi.true,phi[order1,],xlim=z,ylim=z,main=expression(phi[ks]),
     xlab='True',ylab='Estimated')
lines(z,z,col='red',lwd=2)
```



Additional code

Here we provide the underlying code behind the function "lda.presence.absence", distributed in three files. These three files are also available in our public GitHub repository https://github.com/drvalle1/github-presence_absence-SB.

1) *File "gibbs sampler main function.R" containing the wrapper function "lda.presence.absence"*

```
lda.presence.absence=function(dat,id,ncomm,a.phi,b.phi,gamma,ngibbs){
  #useful stuff
  dat1=data.matrix(dat)
  loc.id=id
  nloc=max(loc.id)
  nspp=ncol(dat1)
  nlinhas=nrow(dat1)
  hi=0.999999
  lo=0.000001

  #initial values
  theta=matrix(1/ncomm,nloc,ncomm)
  phi=matrix(0.5,ncomm,nspp)
  z=matrix(sample(1:ncomm,size=nlinhas*nspp,replace=T),nlinhas,nspp)

  #to store outcomes from gibbs sampler
  theta.out=matrix(NA,ngibbs,ncomm*nloc)
```

```
  phi.out=matrix(NA,ngibbs,ncomm*nspp)
  llk=rep(NA,ngibbs)

  #run gibbs sampler
  options(warn=2)
  for (i in 1:ngibbs){
    print(i)

    #sample z
    rand.u=matrix(runif(nlinhas*nspp),nlinhas,nspp)
    z=samplez(log(theta), log(1-phi), log(phi), dat1, loc.id,rand.u, ncomm, nloc)

    #calculate summaries that depend on z and the data
    tmp=getks(z=z, ncommun=ncomm, dat=dat1)
    nks1=tmp$nks1
    nks0=tmp$nks0
    nlk=getlk(z=z,locid=loc.id, ncommun=ncomm, nloc=nloc)

    #get theta parameters
    theta=get.theta(nlk,gamma,ncomm,nloc)
    theta[theta>hi]=hi; theta[theta<lo]=lo

    #get phi parameters
    phi=matrix(rbeta(nspp*ncomm,nks1+a.phi,nks0+b.phi),ncomm,nspp)
    phi[phi>hi]=hi; phi[phi<lo]=lo

    #calculate probability after integrating out the latent z's
    prob=theta%*%phi
    prob[prob>hi]=hi; prob[prob<lo]=lo
    prob1=prob[loc.id,]

    #calculate logl and store results
    llk[i]=sum(dat1*log(prob1)+(1-dat1)*log(1-prob1))
    theta.out[i,]=theta
    phi.out[i,]=phi
  }
  list(llk=llk,theta=theta.out,phi=phi.out)
}
```

2) *File "gibbs functions.R" containing some of the auxiliary functions written in R*

```
#this function generates vmat, which is then used to generate the theta matrix
get.theta=function(nlk,gamma,ncomm,nloc){
  vmat=matrix(NA,nloc,ncomm)
  for (i in 1:(ncomm-1)){
    if (i==(ncomm-1)) cumsoma=nlk[,ncomm]
    if (i< (ncomm-1)) cumsoma=rowSums(nlk[,(i+1):ncomm])
    vmat[,i]=rbeta(nloc,nlk[,i]+1,cumsoma+gamma)
  }
  vmat[,ncomm]=1
  convertVtoTheta(vmat,rep(1,nloc))
}
```

3) *File "aux1.cpp" containing some of the auxiliary function written in c++*

```
#include <Rcpp.h>
#include <iostream>
#include <ctime>
#include <fstream>
using namespace Rcpp;

/******************************************************************************************************
/*********************************                      UTILS          ********************************
/******************************************************************************************************

//' This function summarizes the z matrix into 2 K x S matrices
//' where K is the number of groups and S is the number of species.
//' The matrix "res1" holds the values for dat=1
//' The matrix "res0" holds the values for dat=0
// [[Rcpp::export]]
Rcpp::List getks(IntegerMatrix z, int ncommun, IntegerMatrix dat) {
  int nspp=z.ncol();
  int nlinhas=z.nrow();
  IntegerMatrix res1(ncommun,nspp);
  IntegerMatrix res0(ncommun,nspp);

  for(int i=0; i<nlinhas;i++){
    for (int j=0; j<nspp; j++){
      if (dat(i,j)==1) {
        res1(z(i,j)-1,j)=res1(z(i,j)-1,j)+1;
      }
      if (dat(i,j)==0){
        res0(z(i,j)-1,j)=res0(z(i,j)-1,j)+1;
      }
    }
  }

  Rcpp::List resTemp = Rcpp::List::create(Rcpp::Named("nks1") = res1,
                                          Rcpp::Named("nks0") = res0);
  return(resTemp);
}

//' This function summarizes the z matrix into a L x K matrix
//' where K is the number of groups and L is the number of locations
// [[Rcpp::export]]
IntegerMatrix getlk(IntegerMatrix z, IntegerVector locid, int ncommun, int nloc) {
  int nlinhas=z.nrow();
  int nspp=z.ncol();
  IntegerMatrix res(nloc,ncommun);

  for(int i=0; i<nlinhas;i++){
    for (int j=0; j<nspp; j++){
      res(locid[i]-1,z(i,j)-1)=res(locid[i]-1,z(i,j)-1)+1;
    }
  }

  return(res);
}
```

```
// This function helps with multinomial draws
int whichLessDVPresence(double value, NumericVector prob) {
  int res=-1;
  double probcum = 0;

  for (int i = 0; i < prob.length(); i++) {
    probcum = probcum + prob(i);
    if (value < probcum) {
      res = i;
      break;
    }
  }
  return res;
}

//' This function samples z's
// [[Rcpp::export]]
IntegerMatrix samplez(NumericMatrix ltheta, NumericMatrix l1minusphi, NumericMatrix lphi,
                      IntegerMatrix dat1, IntegerVector locid,
                      NumericMatrix randu, int ncommun, int nloc) {

  IntegerMatrix zmat(dat1.nrow(),dat1.ncol());
  NumericVector prob(ncommun);
  int znew;

  for(int i=0; i<dat1.nrow();i++){
    for (int j=0; j<dat1.ncol(); j++){
      for (int k=0; k<ncommun; k++){
        prob(k)=ltheta(locid(i)-1,k)+dat1(i,j)*lphi(k,j)+(1-dat1(i,j))*l1minusphi(k,j);
      }
      prob=prob-max(prob);
      prob=exp(prob);
      prob=prob/sum(prob);

      //multinomial draw
      znew=whichLessDVPresence(randu(i,j),prob);
      zmat(i,j)=znew+1;
    }
  }
  return zmat;
}

//' This function converts vmat into theta
// [[Rcpp::export]]
NumericMatrix convertVtoTheta(NumericMatrix vmat,
                              NumericVector prod) {
  NumericMatrix res(vmat.nrow(),vmat.ncol());

  for(int j=0; j<vmat.ncol();j++){
    res(_,j)=vmat(_,j)*prod;
    prod=prod*(1-vmat(_,j));
  }
```

```
  return (res);
}
```