BIAS.jl Documentation

Release 0.1.1

Adham Beyki

April 05, 2016

Contents

Version 0.1.1

Requires julia releases 0.4.0 or later

Date April 05, 2016

Author Adham Beyki (abeyki@deakin.edu.au)

Web site http://github.com/adham/BIAS.jl

CHAPTER 1

Purpose

BIAS is an open source implementation of several Bayesian parametric and non-parametric hierarchical mixture models in julia. The implemented models include:

1. Bayesian Mixture Model (BMM)
2. Latent Dirichlet Allocation (LDA)
3. Dirichlet Process Mixture Model (DPM)
4. Hierarchical Dielectric Mixture Model (HDP)
5. dynamic Hierarchical Dirichelt Mixture Model (dHDP)
6. Recurrent Chinese Restaurant Process (RCRP) also known as Temporal Dirichelt Process Mixture model (TDPM)

CHAPTER 2

Getting Started

The following **julia** command will install the package:

```
julia> Pkg.clone("git://github.com/adham/BIAS.jl.git")
```

CHAPTER 3

Contents

3.1 Introduction

A few sentences on

- mixture modeling
- Bayesian n treatment of mixture models
- why hierarchical mixture models
- why temporal

Mixture modelling is a probabilistic representations of subpopulations (or clusters) within a population. It provides a flexible framework for statistical modeling and analysis.

3.1.1 Mixture Models

The pdf/pmf a mixture distributino is given by convex combination of the pdf/pmf of its individual components. We say a distribution f is a mixture of K component distributions if

$$f\left(x;\Theta,\pi\right)=\sum_{k=1}^{K}\pi_k f\left(x;\theta_k\right)$$

A mixture model is characterized by a set of component parameters $\Theta=\{\theta_1,\ldots,\theta_K\}$ and a prior distribution π over these components.

3.2 Distributions

All distributions provided in this package are organized in a type hierarchy as follows:

```
abstract Distribution
distribution <: Distribution
```

Here is the list of implemented distribution. This list will grow as I continue to develop this package.

- Gaussian1D
- Dirichlet

3.2.1 Gaussian1D

Gaussian1D is the univariate Gaussian distribution. It is parametrized by its mean and variance.

```
julia> using BIAS
julia> srand(123)

julia> qq = Gaussian1D(5, 2)
Gaussian1D distribution
mean: mu=5.0, variance: vv=2.0

julia> sample(qq)
6.683292

julia> sample(qq, 100)
100-element Array{Float64, 1}:
7.89656
6.61595
...

julia> pdf(qq, 4.5)
0.265003

julia> logpdf(qq, 4.5)
-1.328012
```

3.2.2 Dirichlet

Dirichlet distribution is a probability distribution on a probability simplex where its draws are multinomial random variables.

```
julia> qq = Dirichlet([0.2, 0.2, 0.2, 0.2])
Dirichlet distribution
cardinality = 4
alpha       = [0.2,0.2,0.2,0.2]

julia> qq = Dirichlet(5, 4)
Dirichlet distribution
cardinality = 4
alpha       = [1.25,1.25,1.25,1.25]

julia> mean(qq)
4-elemnt Array{Float64, 1}
0.25
0.25
0.25
0.25
```

Caution: pdf and logpdf need to be revised.

3.3 Conjugates

`conjugates` are types equivalent to likelihood-prior pairs in Bayesian paradigm. They comprise of a likelihood distribution and a prior distribution where the parameters of the likelihood are drawn from the prior. Apart from the

parameters of the likelihood and the prior distributions, a conjugate component also contains the data points which are associated with it.

All conjugates provided in this package are organized in a type hierarchy as follows:

```
abstract Conjugates
conjugate <: Conjugates
```

Conjugates names follow the convention of LikelihoodPrior. Therefore MultinomialDirichlet is a Bayesian conjugate component with a Multinomial likelihood where the Multinomial distribution parameter is drawn from Dirichelt distribution.

3.3.1 Conjugate distributions

Here is a list of implemented conjugates. This list will grow as I continue to develop this package.

- Gaussian1DGaussian1D
- MultinomialDirichlet

Gaussian1DGaussian1D

It is a conjugate component with univariate Gaussian likelihood. The variance of the likelihood is known. The mean parameter of the likelihood is drawn from another univariate Gaussian distribution.

$$\mu|\mu_0, \sigma_0^2 \sim \text{Gaussian}\left(\mu_0, \sigma_0^2\right)$$
$$x_1, \ldots, x_n|\mu, \sigma^2 \sim \text{Gaussian}\left(\mu, \sigma^2\right)$$

It is parametrized by the mean and variance of the prior, `m0` and `v0`, and the variance of the likelihood `vv`.

```
julia> qq = Gaussian1DGaussian1D(m0, v0, vv)
```

```
julia> qq = Gaussian1DGaussian1D(2, 10, 0.5)
Gaussian1DGaussian1D conjugate component
likelihood parameters: mu=2.0, vv=0.5
prior parameters     : m0=2.0, v0=10.0
number of data points: nn=0
```

MultinomialDirichlet

It is a conjugate component with Multinmomial likelihood where the likelihood parameter (i.e. Multinomial vector) is drawn from a Dirichlet distirbution.

$$\theta|\alpha \sim \text{Dirichlet}\left(\alpha\right)$$
$$x_{1:n}|\theta \sim \text{Multinomial}\left(\theta\right)$$

where $\theta = [\theta_1, \ldots, \theta_K]$ and $\alpha = [\alpha_1, \ldots, \alpha_K]$. It is parametrized by the Dirichlet concentration parametr. You can create a `MultinomialDirichlet` object by passing the cardinality and prior concentration parameter.

```
julia> qq = MultinomialDirichlet(5, 2.0)
MultinomialDirichlet component
cardinality: dd=5, Dirichlet prior parameter: aa=0.4
data: mm=0, nn=0
```

Note: write a few sentences about nn and mm and the two different forms that Multinomial Dirichelty can be used.

3.3.2 Interface

In this section the functions that operate on conjugates are described.

Adding and removing data points

Note: Since these functions are not meant to be used directly by the user, they are not exported.

Use `additem!` or `delitem!` functions to add or delete a data point.

```
julia> qq = Gaussian1DGaussian1D(2, 10, 0.5)
Gaussian1DGaussian1D conjugate component
likelihood parameters: mu=2.0, vv=0.5
prior parameters     : m0=2.0, v0=10.0
number of data points: nn=0

julia> BIAS.additem!(qq, 3.0)
julia> BIAS.additem!(qq, 3.5)
julia> BIAS.additem!(qq, 2.8)
likelihood parameters: mu=3.099999.0, vv=0.5
prior parameters     : m0=2.0, v0=10.0
number of data points: nn=3

julia> BIAS.delitem!(qq, 2.8)
Gaussian1DGaussian1D conjugate component
likelihood parameters: mu=3.249999, vv=0.5
prior parameters     : m0=2.0, v0=10.0
number of data points: nn=2
```

Posterior distribution

Use `posterior` to find the posterior probability distribution of the likelihood parameters given the observations. Since we use conjugate priors, the posterior will have the same form as prior.

$$p\left(\theta|x_{1:n}\right) \propto p\left(\theta\right) p\left(x_{1:n}|\theta\right)$$

```
julia> qq = Gaussian1DGaussian1D(2, 10, 0.5)
julia> BIAS.additem!(qq, 3.0)
julia> BIAS.additem!(qq, 3.5)
julia> BIAS.additem!(qq, 2.8)
likelihood parameters: mu=3.099999.0, vv=0.5
prior parameters     : m0=2.0, v0=10.0
number of data points: nn=3

julia> posterior(qq)
Gaussian1D distribution
mean: mu=3.081967, variance: vv=0.163934
```

Posterior predictive likelihood

Note: Since these functions are not meant to be used directly by the user, they are not exported.

Use `logpredictive` to compute the posterior predictive likelihood of a component given a data point.

$$p\left(\tilde{x}|x_{1:n}\right) = \int_{\theta} p\left(\tilde{x}|\theta\right) p\left(\theta|x_{1:N}\right) d\theta$$

```
julia> qq = Gaussian1DGaussian1D(2, 10, 0.5)
julia> BIAS.additem!(qq, 3.0)
julia> BIAS.additem!(qq, 3.5)
julia> BIAS.additem!(qq, 2.8)
julia> posterior(qq)
Gaussian1D distribution
mean: mu=3.081967, variance: vv=0.163934

julia> BIAS.logpredictive(qq, 3.2)
-0.724644
julia> BIAS.logpredictive(qq, 4.2)
-1.655550
```

3.4 Models

3.4.1 Bayesian Finite Mixture Model

3.4.2 Latent Dirichlet Allocation

3.4.3 Dirichlet Process Mixture Model

3.4.4 Hierarchical Dirichlet Process Mixture Model

3.4.5 dynamic Hierarchical Dirichlet Process Mixture Model

3.4.6 Temporal Dirichlet Process Mixture Model

3.5 Examples

The following list shows several examples on how to use **BIAS** package:

3.5.1 Bayesian Mixture Model

BMM for univariate Gaussian likelihood

Here we look at the problem of fitting a Bayesian Mixture Model to a series of data points that are drawn from a number of Gaussian distributions. We assume:

- the variance of these Gaussian components is fixed and known
- the mean of these Gaussian components are drawn from another Gaussian distribution

What we are looking for is the mean of inferred Gaussian components after observing the data.

Problem

We have N data points generated by K univariate Gaussian distributions with known variance and unknown means. Infer the means from the data.

Model

$$\begin{aligned}
\boldsymbol{\pi}|\alpha &\sim \text{Dirichlet}\,(\alpha) \\
\mu_k|\mu_0, \sigma_0^2 &\sim \mathcal{N}\left(\mu_0, \sigma_0^2\right) \\
z_n|\boldsymbol{\pi} &\sim \boldsymbol{\pi} \\
x_n|z_n, \mu_{1:K}, \sigma^2 &\sim \mathcal{N}\left(\mu_{z_n}, \sigma^2\right)
\end{aligned}$$

Solution

Simulate the data First we need to simulate a dataset. For this, we have to specify the parameters of some "true" clusters and use the generative process mentioned above to obtain the observations. Each cluster is specified with a univariate Gaussian distribution. Below, we assume 5 Gaussian distributions with means [1, 2, 3, 4, 5] and variance equal to 0.01. The total number of observations is 500.

```
using BIAS
srand(123)

true_KK = 5                                 # number of components
NN = 500                                    # number of data points

vv = 0.01                                    # fixed variance
true_atoms = [Gaussian1D(kk, vv) for kk = 1:true_KK]

mix = ones(Float64, true_KK)/true_KK
xx = ones(Float64, NN)
true_zz = ones(Int, NN)
true_nn = zeros(Int, true_KK)

for n=1:NN
    kk = BIAS.sample(mix)
    true_zz[n] = kk
    xx[n] = sample(true_atoms[kk])
    true_nn[kk] += 1
end
```

Let's look at the true cluster parameters. `true_nn[kk]` shows the number of data points generate by cluster `kk`.

```
julia> true_atoms
5-element Array{BIAS.Gaussian1D,1}:
 Gaussian1D distribution
mean: mu=1.0, variance: vv=0.1

 Gaussian1D distribution
mean: mu=2.0, variance: vv=0.1

 Gaussian1D distribution
mean: mu=3.0, variance: vv=0.1
```

```
 Gaussian1D distribution
mean: mu=4.0, variance: vv=0.1

 Gaussian1D distribution
mean: mu=5.0, variance: vv=0.1

julia> true_nn
5-elemet Array{Int64, 1}
 102
  88
 116
  99
  95
```

Model construction The prior-likelihood pair of this model can be seen as a `Gaussian1DGaussian1D` component.

```
m0 = mean(xx)
v0 = 2.0
q0 = Gaussian1DGaussian1D(m0, v0, vv)
```

Now we construct and instantiate the model:

```
KK = true_KK
bmm_aa = 1.0
bmm = BMM(q0, KK, bmm_aa)

zz = zeros(Int64, length(xx))
init_zz!(bmm, zz)
```

Inferecne Now it is time to run the inference routine:

```
n_burnins   = 100
n_lags      = 2
n_samples   = 200
store_every = 100
filename    = "demo_BMM_Gaussian1DGaussian1D_"

collapsed_gibbs_sampler!(bmm, xx, zz, n_burnins, n_lags, n_samples, store_every, filename)
```

to obtain the posterior distributions:

```
julia> posterior_components, nn = posterior(bmm, xx, zz)
julia> posterior_components
5-element Array{BIAS.Gaussian1D,1}:
 Gaussian1D distribution
mean: mu=4.990567102836185, variance: vv=0.00010525761802010421

 Gaussian1D distribution
mean: mu=3.00416499910953, variance: vv=8.620318089737512e-5

 Gaussian1D distribution
mean: mu=1.9939209647504217, variance: vv=0.00011362990739162548

 Gaussian1D distribution
mean: mu=1.0153304673990435, variance: vv=9.803441007793736e-5
```

```
 Gaussian1D distribution
mean: mu=4.000720712951586, variance: vv=0.00010100499974748751

julia> nn
5-element Array{Int64,1}:
  95
 116
  88
 102
  99
```

As it is readily seen `nn` is equal to `true_nn` by permutation and the posterior distribution of clusters are very close to `true_atoms`.

BMM for Multinomial likelihood

Here we look at the problem of fitting a Bayesian mixture model to a series of observations made from Multinomial distributions.

Here we assume:

- the probability vector of the Multinomial distribution is drawn from a Dirichlet distribution

Problem

We have N observation generated by K Multinomial distributions (i.e. topics). Infer the topics from the data.

Model

$$\begin{aligned}
\boldsymbol{\pi}|\alpha &\sim \text{Dirichlet}\,(\alpha) \\
\phi_k|\beta &\sim \text{Dirichlet}\,(\beta) \\
z_n|\boldsymbol{\pi} &\sim \boldsymbol{\pi} \\
x_n|z_n, \phi_{1:K} &\sim \phi_{z_n}
\end{aligned}$$

Solution

Simulate the data First we need to simulate a dataset. For this, we have to specify the parameters of some "true" mixture components and use the generative process mentioned above, to obtain the observations. Each component is a Multinomial distribution over a vocabulary that we call it a topic. Below, we create 4 topics over 25 words.

```
using BIAS
srand(123)

true_KK    = 4
vocab_size = 25

true_topics = BIAS.gen_bars(true_KK, vocab_size, 0.0)
4x25 Array{Float64,2}:
0.2  0.0  0.0  0.0  0.0  0.2  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.2  0.0  0.0  0.0  0.0
0.0  0.2  0.0  0.0  0.0  0.0  0.2  0.0  0.0  0.0      0.2  0.0  0.0  0.0  0.0  0.2  0.0  0.0  0.0
```

```
0.2  0.2  0.2  0.2  0.2  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.2  0.2  0.2  0.2  0.2     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Looking at the numerical values of the topics is not very convenient. Instead we can think of each topic as a 5x5 image and plot it.

Fig. 3.1: true topics

Now we are ready to draw observations from the simulated topics. We assume each observation is a sentence with 15 words. We have 200 observations in total.

```
n_sentences = 200
n_tokens    = 15

mix = ones(true_KK) / true_KK
xx = Array(Sent, n_sentences)
true_zz = zeros(Int, n_sentences)
true_nn = zeros(Int, true_KK)
for ii = 1:n_sentences
    kk = sample(mix)
    true_zz[ii] = kk
    true_nn[kk] += 1
    sentence = sample(true_topics[kk, :][:], n_tokens)
    xx[ii] = BIAS.sparsify_sentence(sentence)
end
```

`xx` is a vector of type `Sent`.

```
julia> xx[1]
BIAS.Sent([10,9,7,8,6],[2,3,4,3,3])
```

Model construction The prior-likelihood pair of this model can be seen as a `MultinomialDirichlet` component.

```
d = vocab_size
aa = 1.0
q0 = MultinomialDirichlet(dd, aa)
```

Now we construct and instantiate the model:

```
bmm_KK = true_KK
bmm_aa = 0.1
bmm = BMM(q0, bmm_KK, bmm_aa)
```

```
# Sampling
zz = zeros(Int, length(xx))
init_zz!(bmm, zz)
```

Inferecne Now it is time to run the inference routine:

```
n_burnins   = 100
n_lags      = 2
n_samples   = 200
store_every = 100
filename    = "demo_BMM_MultinomialDirichlet_"

collapsed_gibbs_sampler!(bmm, xx, zz, n_burnins, n_lags, n_samples, store_every, filename)
```

to obtain the posterior distributions:

```
posterior_components, nn = posterior(bmm, xx, zz)
inferred_topics = zeros(Float64, bmm.K, vocab_size)
for kk = 1:length(posterior_components)
    inferred_topics[kk, :] = mean(posterior_components[kk])
end

visualize_bartopics(inferred_topics)
```

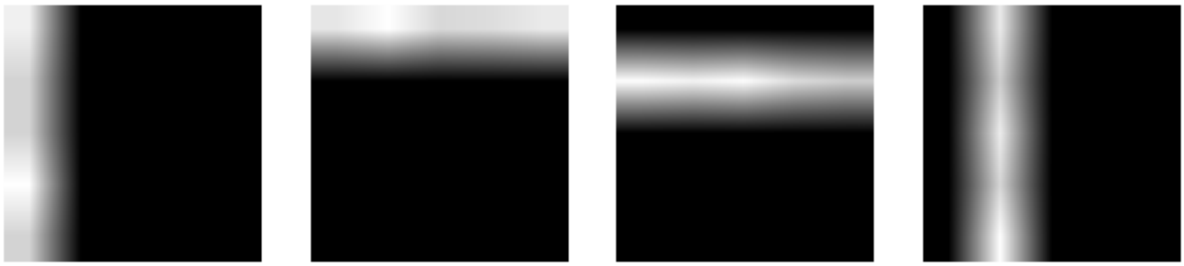

Fig. 3.2: inferred topics

As it is readily seen from two figures, the model has successfully inferred the topics. Also:

```
julia> true_nn
4-element Array{Int64, 1}
 51
 55
 49
 45

julia> nn
4-element Array{Int64, 1}
 49
 51
 55
 45
```

3.5.2 Latent Dirichlet Allocation

LDA for univariate Gaussian likelihood

We have already looked at the problem of modeling a group of data points that are drawn from a mixture of Gaussian distributions in *BMM for univariate Gaussian likelihood*. This is the simplest case of Bayesian mixture modeling. Now what if instead of one group of data points, we have J groups? This adds one level of hierarchy on the representation of the data that needs to be addressed in the model as well. LDA was proposed to solve this problem.

Here we Here we look at the problem of modeling J groups of data points where each grouop is generated by a mixture of Gaussian distributions where it is desired that groups share their mixing components.

We assume:

- the variance of these Gaussian components is fixed and known
- the mean of these Gaussian components are drawn from another Gaussian distribution

What we are looking for is the mean of inferred Gaussian components after observing the data.

Problem

We have J groups of data points. Each group is generated by a mixture of K univariate Gaussian distributions with known variance and unknown means. Assuming J groups share their mixing components, infer the unknown means from the data.

Model

$$
\begin{aligned}
\boldsymbol{\pi}_j|\alpha &\sim \text{Dirichlet}\left(\alpha\right) \\
\mu_k|\mu_0, \sigma_0^2 &\sim \mathcal{N}\left(\mu_0, \sigma_0^2\right) \\
z_{ji}|\boldsymbol{\pi} &\sim \boldsymbol{\pi} \\
x_{ji}|z_{ji}, \mu_{1:K}, \sigma^2 &\sim \mathcal{N}\left(\mu_{z_{ji}}, \sigma^2\right)
\end{aligned}
$$

Solution

Simulate the data First we need to simulate a dataset. For this, we have to specify the parameters of some "true" mixture components and then use the generative process mentioned above to obtain the observations. Each component is a univariate Gaussian distribution. Below, we assume 5 Gaussian distributions with means [1, 2, 3, 4, 5] and variance equal to 0.01.

```
using BIAS
srand(123)

true_KK = 5                                 # number of components
vv = 0.01                                   # fixed variance
true_atoms = [Gaussian1D(kk, vv) for kk = 1:true_KK]
```

Now we simualte the observtaioas from the components.

```
n_groups  = 1000                                    # number of groups in the corpus, J
n_group_j = 100 * ones(Int, n_groups)               # number of observations in each group
```

```
alpha = 0.1
xx = Array(Vector{Float64}, n_groups)
true_zz = Array(Vector{Int}, n_groups)
true_nn = zeros(Int, n_groups, true_KK)
for jj = 1:n_groups
    xx[jj] = zeros(Float64, n_group_j[jj])
    true_zz[jj] = ones(Int, n_group_j[jj])
    theta = BIAS.rand_Dirichlet(alpha .* ones(Float64, true_KK))
    for ii = 1:n_group_j[jj]
        kk = sample(theta)
        true_zz[jj][ii] = kk
        true_nn[jj, kk] += 1
    xx[jj][ii] = sample(true_atoms[kk])
    end
end
```

Therefore we have 1000 groups, each consisting of 100 data points drawn from 5 univariate Gaussian distributions.

Model construction The prior-likelihood pair of this model is a `Gaussian1DGaussian1D` component.

```
m0 = mean(mean(xx))
v0 = 10.0
q0 = Gaussian1DGaussian1D(m0, v0, vv)
```

Now we construct and instantiate the model:

```
KK = true_KK
lda_aa = 1.0
lda = LDA(q0, KK, lda_aa)

zz = Array(Vector{Int}, n_groups)
for jj = 1:n_groups
    zz[jj] = ones(Int, n_group_j[jj])
end
init_zz!(lda, zz)
```

Inferecne Now it is time to run the inference routine:

posterior distributions posterior_components, nn = posterior(lda, xx, zz)

```
n_burnins   = 100
n_lags      = 2
n_samples   = 200
store_every = 100
filename    = "demo_LDA_Gaussian1DGaussian1D_"

collapsed_gibbs_sampler!(lda, xx, zz, n_burnins, n_lags, n_samples, store_every, filename)
```

to obtain the posterior distributions:

```
julia> posterior_components, nn = posterior(lda, xx, zz)
julia> posterior_components
5-element Array{BIAS.Gaussian1D,1}:
 Gaussian1D distribution
mean: mu=0.9997691504265703, variance: vv=4.827884703354245e-7

 Gaussian1D distribution
mean: mu=4.45205906026365, variance: vv=2.632063454838595e-7
```

```
 Gaussian1D distribution
mean: mu=3.009697893083578, variance: vv=9.67585405135266e-7

 Gaussian1D distribution
mean: mu=2.990988787760691, variance: vv=9.2712732842234e-7

 Gaussian1D distribution
mean: mu=2.000400973353594, variance: vv=4.957119675526774e-7
```

As it is seen, `posterior_components` are very close to ''true" components.

LDA for Multinomial likelihood

We have already looked at modeling one group of data points that are drawn from a mixture Multinomial distribution in *BMM for Multinomial likelihood*. Now we aim to look at the modeling `J` groups of data where each group is drawn from a mixture of Multinomial distributions while it is desired for these groups to share their mixture components.

Here we assume:

- the probability vector of the Multinomial distribution is drawn from a Dirichlet distribution

Since usually `MultinomialDirichlet` conjugate is used in the context of text processing, we also adopt the same jargon. In this context, each mixing component is called a topic as it is a multinomial distribution over a vocabulary. There is a subtle difference in how a sample is drawn from these sentences in different papers. Some works assume that each word in a sentence is drawn from a different topic and draw one word at a time from a topic. Some other assume each sentence is drawn from one topic, and draw a sentence (i.e. n words) at a time from a topic.

Both methods are implemented in **BIAS** package and the relevant examples are given in the following sections.

Problem

We have J groups of data points. Each group is generated by a mixture of K topics. Assuming `J` groups share the topics, infer them from the data.

Model

$$
\begin{aligned}
\boldsymbol{\pi}|\alpha &\sim \text{Dirichlet}\,(\alpha) \\
\phi_k|\beta &\sim \text{Dirichlet}\,(\beta) \\
z_{ji}|\boldsymbol{\pi}_j &\sim \boldsymbol{\pi}_j \\
x_{ji}|z_{ji}, \phi_{1:K} &\sim \phi_{z_{ji}}
\end{aligned}
$$

Simulated data

First we need to specify the parameters of some "true" topics. Below, we create 10 topics over 25 words.

```
using BIAS
srand(123)

true_KK = 10
vocab_size = 25
```

```
true_topics = BIAS.gen_bars(true_KK, vocab_size, 0)
visualize_bartopics(true_topics)
```

Fig. 3.3: true topics

Scenario 1: MultinomialDirichelt with single observations Now we are ready to simulate the corpus. In this section, we assume the observations are single words. We construct a corpus of 100 documents (groups), each document consisting of 50 words (observations).

```
n_groups = 100
n_group_j = 50 * ones(Int, n_groups)
max_n_topic_doc = 4
n_groups = 100

xx = Array(Vector{Int}, n_groups)
true_zz = Array(Vector{Int}, n_groups)
true_nn = zeros(Int, n_groups, true_KK)
n_topic_doc_p = cumsum(fill(1/max_n_topic_doc, max_n_topic_doc))

for jj = 1:n_groups
  n_topic_doc = sum(rand() .< n_topic_doc_p)
  topic_doc = randperm(true_KK)[1:n_topic_doc]

  xx[jj] = zeros(Int, n_group_j[jj])
  true_zz[jj] = ones(Int, n_group_j[jj])
  for ii = 1:n_group_j[jj]
    kk = topic_doc[rand(1:n_topic_doc)]
    true_zz[jj][ii] = kk
    xx[jj][ii] = BIAS.sample(true_topics[kk, :][:])
    true_nn[jj, kk] += 1
  end
end
```

Model construction The prior-likelihood pair of this model can be seen as a `MultinomialDirichlet` component.

```
dd = vocab_size
aa = 0.1 * dd
q0 = MultinomialDirichlet(dd, aa)
```

Now we construct and instantiate the model:

```
lda_KK = true_KK
lda_aa = 0.1
lda = LDA(q0, lda_KK, lda_aa)

zz = Array(Vector{Int}, n_groups)
for jj = 1:n_groups
    zz[jj] = ones(Int, n_group_j[jj])
end
init_zz!(lda, zz)
```

Inference Now it is time to run the inference routine:

```
n_burnins   = 100
n_lags      = 2
n_samples   = 400
store_every = 100
filename    = "demo_LDA_Multinomial_Dirichlet"

collapsed_gibbs_sampler!(lda, xx, zz, n_burnins, n_lags, n_samples, store_every, filename)

# posterior distributions
posterior_components, nn = posterior(lda, xx, zz)
```

to obtain the posterior distributions:

```
inferred_topics = zeros(Float64, length(posterior_components), vocab_size)
for kk = 1:length(posterior_components)
    inferred_topics[kk, :] = mean(posterior_components[kk])
end

visualize_bartopics(inferred_topics)
```

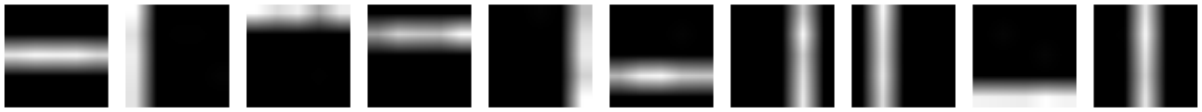

Fig. 3.4: inferred topics

As it is readily seen from two figures, the model has successfully inferred the topics. Also:

Scenario 2: MultinomialDirichelt with multiple observations In this section we assume instead of drawing one word at a time form a topic, we draw a sentence i.e. n words. We construct a corpus of 25 documents (groups), each document consisting of 100 sentences, and each sentence consisting of 230 words.

```
n_groups = 25
n_group_j = 100 * ones(Int, n_groups)
n_tokens = 20

alpha = 0.1
xx = Array(Vector{Sent}, n_groups)
true_zz = Array(Vector{Int}, n_groups)
true_nn = zeros(Int, n_groups, true_KK)
for jj = 1:n_groups
    xx[jj] = Array(Sent, n_group_j[jj])
    true_zz[jj] = ones(Int, n_group_j[jj])
    theta = BIAS.rand_Dirichlet(fill(alpha, true_KK))
    for ii = 1:n_group_j[jj]
        kk = sample(theta)
    sentence = sample(true_topics[kk, :][:], n_tokens)
    xx[jj][ii] = BIAS.sparsify_sentence(sentence)
    true_zz[jj][ii] = kk
    true_nn[jj, kk] += 1
  end
end
```

Model construction Similar to scenario 1, we construct the conjugate object.

```
dd = vocab_size
aa = 0.1 * dd
q0 = MultinomialDirichlet(dd, aa)
```

Now we construct and instantiate the model:

```
lda_KK = true_KK
lda_aa = 0.1
lda = LDA(q0, lda_KK, lda_aa)

zz = Array(Vector{Int}, n_groups)
for jj = 1:n_groups
    zz[jj] = ones(Int, n_group_j[jj])
end
init_zz!(lda, zz)
```

Inference Now it is time to run the inference routine:

```
n_burnins   = 100
n_lags      = 2
n_samples   = 100
store_every = 100
filename    = "demo_LDA_Multinomial_Dirichlet"

collapsed_gibbs_sampler!(lda, xx, zz, n_burnins, n_lags, n_samples, store_every, filename)

# posterior distributions
posterior_components, nn = posterior(lda, xx, zz)
```

to obtain the posterior distributions:

```
inferred_topics = zeros(Float64, length(posterior_components), vocab_size)
for kk = 1:length(posterior_components)
    inferred_topics[kk, :] = mean(posterior_components[kk])
end

visualize_bartopics(inferred_topics)
```

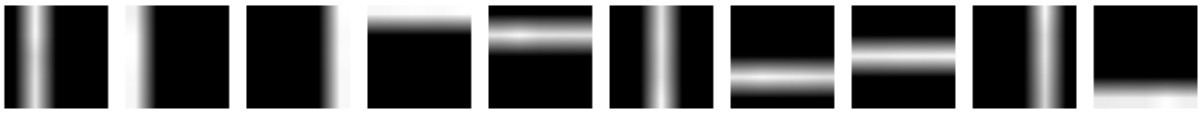

Fig. 3.5: inferred topics

As it is readily seen from two figures, the model has successfully inferred the topics. Also:

Real data

In this section we fit LDA to a corpus of nematode biology abstracts (see https://web.archive.org/web/20040328153507/http://elegans.swmed.edu/wli/cgcbib). Yee Why Teh has used this dataset to validate HDP and showed that a model with 50 to 80 topics has the least perplexity *[1]*. We choose to fit a model with 70 topics on this dataset. The provided data, it taken from the link above and preprocessed. It consists of 5957 abstracts and the vocabulary size is 3793. Please consider that we are using the first representation of a MultinomialDirichelt object for this example.

First we read the dataset.

```
f = open("datasets\\cgcbib\\cgcbib_abs_bow.txt")
lines = readlines(f)
close(f)

n_lines = length(lines)
n_groups = 5957
n_group_j = zeros(Int, n_groups)
n_vocab = 3793

lines_mat = zeros(Int, n_lines, 4)
for i = 1:n_lines
  line = split(strip(lines[i]))
  line = [parse(Int, s) for s = line]
  lines_mat[i, :] = line
end

xx = Array(Vector{Int}, n_groups)

for jj = 1:n_groups
  print("$jj ")
  mask = lines_mat[:, 1] .== jj-1
  xx_jj_mat = lines_mat[mask, :]
  xx_jj = Int[]

  for ii=1:size(xx_jj_mat, 1)
    for rr=1:xx_jj_mat[ii, 4]
      push!(xx_jj, xx_jj_mat[ii, 3]+1)
    end
  end
  xx[jj] = xx_jj
end

n_group_j = zeros(Int, n_groups)
for jj = 1:n_groups
  n_group_j[jj] = length(xx[jj])
end
```

The rest is similar to previous exampes.

```
dd = n_vocab
aa = 0.5 * n_vocab
q0 = MultinomialDirichlet(dd, aa)

# constructing the model
lda_KK = 70
lda_aa = 1.5
lda = LDA(q0, lda_KK, lda_aa)

# sampling
zz = Array(Vector{Int}, n_groups)
for jj = 1:n_groups
    zz[jj] = ones(Int, n_group_j[jj])
end

init_zz!(lda, zz)

n_burnins   = 200
```

```
n_lags       = 1
n_samples    = 300
sample_hyperparam = true
store_every = 1000
filename     = "cgc_LDA_results_"
collapsed_gibbs_sampler!(lda, xx, zz, n_burnins, n_lags, n_samples, store_every, filename)
posterior_components, nn = posterior(lda, xx, zz)
```

Looking at the numerical values of a topic with cardinality of 3793 is not very informative. One better way is plotting the wordcloud of the topic. A wordcloud is a graphical representation of the topic where the font size of each word is linearly related to its weight in the probability vector.

The following code will create a directory and saves the topics as csv files and wordclouds in it.

```
f = open("datasets\\cgcbib\\cgcbib_abs_vocab.txt")
vocab = readlines(f)
close(f)
vocab = [strip(vv) for vv in vocab]


dirname = "datasets\\cgcbib\\LDA_results"
mkdir(dirname)

for kk = 1:length(posterior_components)

    filename = join([dirname, "topic$(kk).csv"], "/")
    topic2csv(filename, vocab, posterior_components[kk].alpha)

    filename = join([dirname, "topic$(kk).png"], "/")
    topic2png(filename, vocab, posterior_components[kk].alpha)
end
```

One of the topics inferred from the data is shown here:

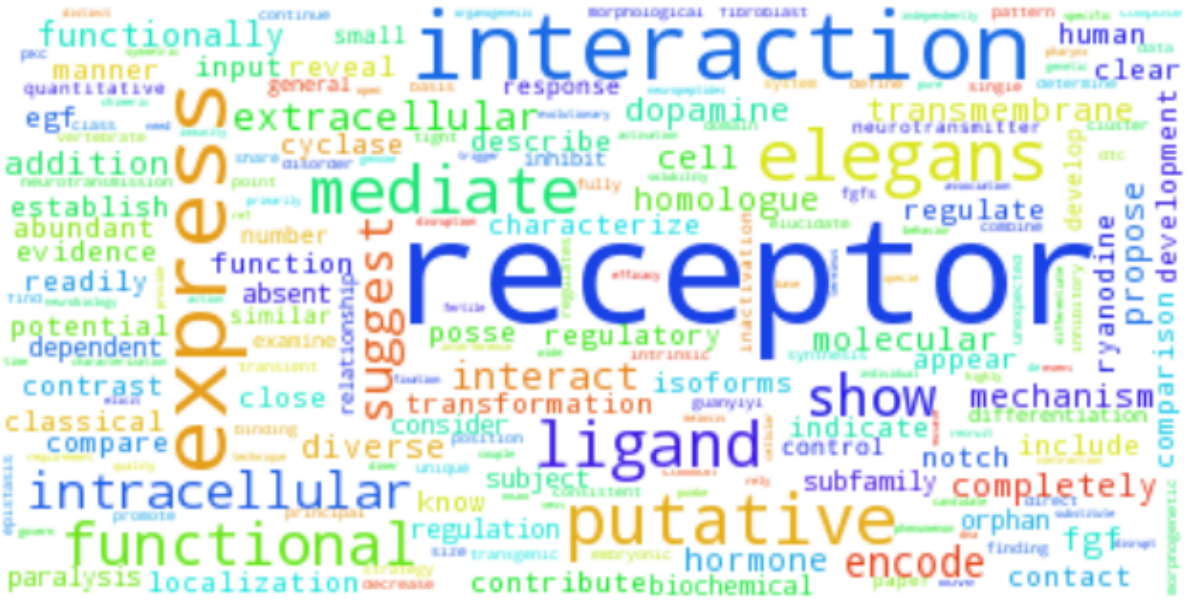

Fig. 3.6: a topic inferred from cgcbib dataset using LDA

3.5.3 Recurrent Chinese Restaurant Process

RCRP for univariate Gaussian likelihood

Scenario

	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
Epoch 1	O	O	O			
Epoch 2	O	O	O	O		
Epoch 3		O	O	O	O	
Epoch 4			O	O	O	O
Epoch 5			O	O	O	O

```
true_KK = 6
TT = 5
N_t = fill(300, TT)

vv = 0.01
true_atoms = [Gaussian1D(kk, vv) for kk = 1:true_KK]

xx = Array(Vector{Float64}, TT)
true_zz = Array(Vector{Int64}, TT)
true_nn = zeros(Int, TT, true_KK)

epoch_chains = Array(Vector{Int}, TT)
epoch_chains[1] = [1, 2, 3]
epoch_chains[2] = [1, 2, 3, 4]
epoch_chains[3] = [2, 3, 4, 5]
epoch_chains[4] = [3, 4, 5, 6]
epoch_chains[5] = [3, 4, 5, 6]

for tt = 1:TT
    n_chains = length(epoch_chains[tt])
    mix = ones(Float64, n_chains)/n_chains
    xx_ = ones(Float64, N_t[tt])
    true_zz_ = ones(Int64, N_t[tt])

    for n = 1:N_t[tt]
        kk = sample(mix)
        kk = epoch_chains[tt][kk]
        true_zz_[n] = kk
        xx_[n] = sample(true_atoms[kk])
        true_nn[tt, kk] += 1
    end

    xx[tt] = xx_
    true_zz[tt] = true_zz_
end

m0= mean(xx[1])
v0 = 2
q0 = Gaussian1DGaussian1D(m0, v0, vv)

init_KK = 3
rcrp_aa = 1
rcrp_a1 = 1
rcrp_a2 = 1
```

```
rcrp = RCRP(q0, init_KK, rcrp_aa, TT, rcrp_a1, rcrp_a2)

# sampling
zz = Array(Vector{Int64}, TT)
for tt = 1:TT
  zz[tt] = rand(1:rcrp.K, N_t[tt])
end

n_burnins    = 100
n_lags       = 1
n_samples    = 400
sample_hyperparam = true
n_internals = 10
store_every = 10000
filename     = "demo_RCRP_Gaussian1DGaussian1D_"
nn, components, zz2table = collapsed_gibbs_sampler!(rcrp, xx, zz,
    n_burnins, n_lags, n_samples, sample_hyperparam, n_internals, store_every, filename)
```

Result

```
julia> nn
5x12 Array{Int64,2}:
 106  93  100   0   0   0  0  1  0  0  0  0
  76  70   75  78   0   0  0  0  1  0  0  0
  73   1   84  74  64   0  2  0  0  1  1  0
   1   0   67  80  84  68  0  0  0  0  0  0
   0   0   77  69  81  72  0  0  0  0  0  1

julia> true_nn
5x6 Array{Int64,2}:
 93  106  101   0   0   0
 70   76   75  79   0   0
  0   77   84  75  64   0
  0    0   68  80  84  68
  0    0   77  70  81  72

julia> zz2table
5x12 Array{Int64,2}:
 1  2  3  0  0  0  0  4  0  0  0   0
 1  2  3  5  0  0  0  0  6  0  0   0
 1  6  3  2  4  0  5  0  0  7  8   0
 6  0  4  1  3  2  0  0  0  0  0   0
 0  0  8  4  9  3  0  0  0  0  0  10
```

RCRP for Multinomial likelihood

Scenario

	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
Epoch 1	O	O	O			
Epoch 2	O	O	O	O		
Epoch 3		O	O	O	O	
Epoch 4			O	O	O	O
Epoch 5			O	O	O	O

```
true_KK = 6
TT = 5

N_t = fill(500, TT)
n_tokens = 25
vocab_size = 9

true_topics = BIAS.gen_bars(true_KK, vocab_size, 0.0)



xx = Array(Vector{Sent}, TT)
true_zz = Array(Vector{Int}, TT)
true_nn = zeros(Int, TT, true_KK)

epoch_chains = Array(Vector{Int}, TT)
epoch_chains[1] = [1, 2, 3]
epoch_chains[2] = [1, 2, 3, 4]
epoch_chains[3] = [2, 3, 4, 5]
epoch_chains[4] = [3, 4, 5, 6]
epoch_chains[5] = [3, 4, 5, 6]

for tt = 1:TT
    n_chains = length(epoch_chains[tt])
    mix = ones(Float64, n_chains)/n_chains
    xx_ = Array(Sent, N_t[tt])
    true_zz_ = ones(Int64, N_t[tt])

    for n = 1:N_t[tt]
        kk = sample(mix)
        kk = epoch_chains[tt][kk]
        true_zz_[n] = kk
        sentence = sample(true_topics[kk, :][:], n_tokens)
        xx_[n] = BIAS.sparsify_sentence(sentence)
        true_nn[tt, kk] += 1
    end

    xx[tt] = xx_
    true_zz[tt] = true_zz_
end




dd = vocab_size
aa = 1.0
q0 = MultinomialDirichlet(dd, aa)

init_KK = 3
rcrp_aa = 1
rcrp_a1 = 1
rcrp_a2 = 1
rcrp = RCRP(q0, init_KK, rcrp_aa, TT, rcrp_a1, rcrp_a2)

# sampling
zz = Array(Vector{Int64}, TT)
for tt = 1:TT
  zz[tt] = rand(1:rcrp.K, N_t[tt])
```

```
end

n_burnins    = 100
n_lags       = 3
n_samples    = 400
sample_hyperparam = true
n_internals = 10
store_every = 10000
filename     = "demo_RCRP_MultinomialDirichlet"
nn, components, zz2table = collapsed_gibbs_sampler!(rcrp, xx, zz,
    n_burnins, n_lags, n_samples, sample_hyperparam, n_internals, store_every, filename)
```

Result

```
```

julia> z2table 5x8 Array{Int64,2}:

0 1 2 3 0 0 0 0 0 1 2 3 4 0 0 0 0 1 2 0 3 4 0 0 0 0 1 0 2 3 4 0 1 0 4 0 2 6 3 5)

julia> nn 5x8 Array{Int64,2}:

0 162 170 168 0 0 0 0 0 136 123 120 121 0 0 0 0 122 127 0 108 143 0 0 0 0 133 0 125 131 111 0

126 0 115 0 118 6 133 2

julia> true_nn 5x6 Array{Int64,2}:

168 162 170 0 0 0 120 136 123 121 0 0

0 122 127 108 143 0 0 0 133 125 131 111 0 0 115 118 132 135

3.6 References

3.7 Indices

- genindex

[1] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the american statistical association*, 2006.

E