笔记 I：fastTopics 包

Jonathan Chow

2022-09-29

目录

1 Rcpp 包的一般结构

```
DemoRcpp
  DESCRIPTION
  NAMESPACE
  R
     RcppExports.R
  Read-and-delete-me
  man
     DemoRcpp-package.Rd
     rcpp_hello_world.Rd
  src
      RcppExports.cpp
      rcpp_hello_world.cpp
```

R 目录下存放 R 代码，编写包函数。其中 RcppExports.R 存放调用 C++ 代码的 R 接口函数。

src 目录下存放 C++ 代码，编写库函数。其中 RcppExports.cpp 用于接收 R 的对象，并转化成 C++ 的对象。

2 fastTopics 包

fastTopics 包通过 R 实现使用期望极大算法（EM）和顺序坐标下降算法（CCD、SCD）拟合 Poisson 非负矩阵分解模型和多项主题模型。

模型拟合函数：`fit_multinom_model.R`，`fit_poisson_nmf.R`，`fit_topic_model.R`。

评价指标函数：`likelihood.R`。

绘图函数：`structure_plot.R`，`other_plots.R`，`embedding_plots.R`，`volcano_plots.R`。

数据模拟函数：`datasim.R`。

辅助函数：`R` 目录下的其他函数和 `src` 目录下的函数主要用于辅助上述主要函数。或者实现较为普遍的功能，或者加速核心算法。

3 辅助函数

3.1 src 目录

`src` 目录下的代码使用 C++ 编写，然后利用 `Rcpp` 实现与 R 的交互，以加速核心算法。

`poismix.cpp` 和 `pnmfem.cpp` 主要用于实现 EM 算法的核心参数迭代。`ccd.cpp` 主要用于实现 CCD 算法的核心参数迭代。`scd.cpp` 主要用于实现 SCD 算法的核心参数迭代。`cost.cpp` 主要用于计算评价指标，比如损失函数、一阶 KKT 条件。

主要使用矩阵库 `RcppArmadillo`。并使用 `RcppParallel` 进行并行运算。

3.2 verify_args.R 和 misc.R

`verify_args.R` 是验证输入合法性的文件。需要注意的是用 `deparse(substitute(x))` 来标记代码。实现的函数如下表。

参数	描述
verify.positive.vector	验证 x 是具有正元素的向量。
verify.nonnegative.matrix	验证 x 是非负矩阵。
verify.count.matrix	验证 x 是有效的计数矩阵。
verify.fit	验证 x 是有效的多项式主题模型拟合或 Poisson 非负矩阵分解。
verify.fit.and.count.matrix	验证 x 是有效的计数矩阵，且 fit 是有效的多项式主题模型拟合或 Poisson 非负矩阵分解。
is.scalar	验证 x 是没有缺失项的有限标量。

misc.R 是实现基本运算的文件。实现的函数如下表。需要使用 Matrix、stats 等软件包。

参数	描述
pfromz	根据 z-score 计算双尾 p 值。。
lpfromz	从 z-score 计算 log10 双尾 p 值。
is.sparse.matrix	验证 x 是否为稀疏矩阵。
get.nonzeros	从稀疏矩阵的第 j 列中高效提取非零元素。
any.allzero.cols	检查矩阵是否包含一个或多个全零列。
remove.allzero.cols	移除全零列。
apply.nonzeros	对稀疏矩阵的所有非零项应用 f 算子。
x_over_tcrossprod	当 X 是稀疏矩阵时高效计算 X/(crossprod(A,B) + e)。使用了 Rcpp 代码。
repmat	利用 rbind(x,...,x) 返回 m*n 矩阵。
scale.cols	将 A[,i] 列用 b[i] 缩放。

参数	描述
normalize.rows	对 A 的每一行缩放使行和为 1。
normalize.cols	对 A 的每一列缩放使列和为 1。
normalize.rows.by.max	对 A 的每一行缩放使每一行最大项为 1。
le.diff	向量 x 的等长差分。
rescale.factors	重新缩放 F 和 L 使 F 的第 k 列和 L 的第 k 列均值相同。
rand	产生均匀分布的随机数。
initialize.multithreading	预先指定初始化 RcppParallel 的多线程数目。使用了 Rcpp 代码。
fit_pnmf_rank1	处理秩为 1 的 Poisson NMF。
hpd	计算最高后验密度（HPD）区间。
min_kl_poisson	计算最小 KL 散度。
le_lfc	计算 “最小极限”LFC 统计 LFC(j) = log2(fj/fk)。

4 评价指标函数

likelihood.R 计算 Poisson NMF 模型与主题模型的评价指标。

输入参数表。

参数	描述
X	n*m 的非负矩阵。可以是稀疏矩阵 dgCMatrix 或稠密矩阵 matrix。
fit	Poisson NMF 或多项式主题模型的拟合。比如 fit_poisson_nmf 或 fit_topic_model 的输出。
A	n*k 维的 L 矩阵。是稠密矩阵。
B	k*m 维的 F 矩阵。是稠密矩阵。
e	为避免计算零的对数而添加的小正常数。代价是产生小的不精确。

参数	描述
family	模型选择。实现了两种模型：poisson 计算 Poisson NMF 的相应的值，multinom 计算多项式主题模型的相应的值。
version	计算引擎。实现了两种选择：R 表示全部在 R 中计算，对于 X 稠密矩阵计算速度快；Rcpp 表示使用了 Rcpp 进行计算，对于 X 大且稀疏矩阵计算速度快且内存占用小。未指定此选项时，根据 X 的类型调用最优的选项

实现的主要函数如下。

```
loglik_poisson_nmf <- function (X, fit, e = 1e-8)
  loglik_helper(X,fit,"loglik.poisson",e)

loglik_multinom_topic_model <- function (X, fit, e = 1e-8)
  loglik_helper(X,fit,"loglik.multinom",e)

deviance_poisson_nmf <- function (X, fit, e = 1e-8)
  loglik_helper(X,fit,"deviance.poisson",e)
```

Poisson 非负矩阵分解模型：计数矩阵 $X \in \mathbb{R}^{n\times m}$，非负矩阵参数 $L \in \mathbb{R}_+^{n\times K}$，$F \in \mathbb{R}_+^{m\times K}$，对 $K \geq 1$，满足

$$x_{ij}|L,F \sim Poisson(\lambda_{ij})$$

$$\lambda_{ij} = (LF^T)_{ij} = \sum_{k=1}^{K} l_{ik}f_{jk}$$

多项生成模型：计数矩阵 $X \in \mathbb{R}^{n\times m}$，非负矩阵参数 $L^* \in \mathbb{R}_+^{n\times K}$，$F^* \in \mathbb{R}_+^{m\times K}$，其中 l_{ik}^*，f_{jk}^* 要求和一性，即

$$\sum_{k=1}^{K} l_{ik}^* = 1, \sum_{j=1}^{m} f_{jk}^* = 1$$

对 $K \geq 2$，满足

$$x_{i1},\ldots,x_{im} \sim Mult(\sum_{j=1}^{m} x_{ij}; \pi_{i1},\ldots,\pi_{im})$$

$$\pi_{ij} = (L^*(F^*)^T)_{ij} = \sum_{k=1}^{K} l^*_{ik} f^*_{jk}$$

构造映射 $\phi : L, F \to L^*, F^*$

$$u_k \leftarrow \sum_{j=1}^{m} f_{jk}, \forall k; \quad f^*_{jk} \leftarrow \frac{f_{jk}}{u_k}, \forall j, k$$

$$s_i \leftarrow \sum_{k=1}^{K} l_{ik} u_k, \forall i; \quad l^*_{ik} \leftarrow \frac{l_{ik} u_k}{s_i}, \forall i, k$$

注意到

$$\mathcal{L}_{Poisson}(X|L, F) = \prod_{i,j} \frac{\lambda_{ij}^{x_{ij}}}{x_{ij}!} e^{-\lambda_{ij}}, \quad \lambda_{ij} = \sum_k l_{ik} f_{jk}$$

$$\mathcal{L}_{Topic}(X|L^*, F^*) = \prod_i \frac{t_i!}{\prod_j x_{ij}!} \prod_j \pi_{ij}^{x_{ij}}, \quad \pi_{ij} = \sum_k l^*_{ik} f^*_{jk}, \quad t_i = \sum_j x_{ij}$$

又由于

$$\begin{aligned}\prod_{j=1}^{m} Poisson(x_j : \lambda_j) =& Mult(x_1, \dots, x_m : \sum_{j=1}^{m} x_{ij}; \frac{\lambda_1}{\sum_{j=1}^{m} \lambda_j}, \dots, \frac{\lambda_m}{\sum_{j=1}^{m} \lambda_j}) \\ & \times Poisson(\sum_{j=1}^{m} x_{ij} : \sum_{j=1}^{m} \lambda_j)\end{aligned}$$

于是

$$\mathcal{L}_{Poisson}(X|L, F) = \mathcal{L}_{Topic}(X|L, F) \prod_i Poisson(t_i : s_i)$$

于是 Poisson NMF 问题相当于求解

$$\begin{aligned} \underset{L,F}{argmax} \quad & \mathcal{L}(X|L, F) = x_{ij} log(l_i^T f_j) - \sum_{i,j} l_i^T f_j \\ s.t. \quad & L \geq 0, F \geq 0 \end{aligned}$$

为了实现函数 `loglik_helper`，我们需要以下辅助函数，列表如下。需要使用软件包 `Matrix`、`stats`。

函数	描述
loglik_poisson_const	计算 Poisson 对数似然中的常数项。
loglik_multinom_const	计算多项对数似然中的常数项。
deviance_poisson_const	计算 Poisson 偏差中的常数项。
poisson_nmf_kkt	计算 Poisson NMF 最优解处的一阶 KKT 残差。
loglik_size_factors 和 ldpois	尺寸指标。

函数 *loglik_helper*【核心代码】

```
  if (output.type == "loglik.poisson" | output.type == "loglik.multinom") {
    f <- loglik_poisson_const(X) - cost(X,L,t(F),e,"poisson")
    if (output.type == "loglik.multinom")
      f <- f - loglik_size_factors(X,fit$F,fit$L)
  } else if (output.type == "deviance.poisson")
    f <- deviance_poisson_const(X) + 2*cost(X,L,t(F),e,"poisson")
```

cost() 是在迭代过程中作为收敛指标的函数，主要部分调用了 C++ 代码。

5 模型拟合函数

5.1 pnmfem.R、ccd.R 和 scd.R

pnmfem.R 是利用 C++ 函数，利用 EM 更新 Poisson 回归模型的参数。

```
pnmfem_update_factors <- function (X, F, L, j = seq(1,ncol(X)),
                                   numiter = 1, nc = 1) {
  F <- t(F)
  if (nc == 1) {
```

```
    if (is.matrix(X))
      F <- pnmfem_update_factors_rcpp(X,F,L,j-1,numiter)
    else if (is.sparse.matrix(X))
      F <- pnmfem_update_factors_sparse_rcpp(X,F,L,j-1,numiter)
  } else if (nc > 1) {
    if (is.matrix(X))
      F <- pnmfem_update_factors_parallel_rcpp(X,F,L,j-1,numiter)
    else if (is.sparse.matrix(X))
      F <- pnmfem_update_factors_sparse_parallel_rcpp(X,F,L,j-1,numiter)
  }
  return(t(F))
}

pnmfem_update_loadings <- function (X, F, L, i = seq(1,nrow(X)),
                                    numiter = 1, nc = 1) {
  X <- t(X)
  L <- t(L)
  if (nc == 1) {
    if (is.matrix(X))
      L <- pnmfem_update_factors_rcpp(X,L,F,i-1,numiter)
    else if (is.sparse.matrix(X))
      L <- pnmfem_update_factors_sparse_rcpp(X,L,F,i-1,numiter)
  } else if (nc > 1) {
    if (is.matrix(X))
      L <- pnmfem_update_factors_parallel_rcpp(X,L,F,i-1,numiter)
    else if (is.sparse.matrix(X))
      L <- pnmfem_update_factors_sparse_parallel_rcpp(X,L,F,i-1,numiter)
  }
  return(t(L))
}
```

`ccd.R` 是利用 C++ 函数，利用 CCD 更新 Poisson 回归模型的参数。

`scd.R` 是利用 C++ 函数，利用 SCD 更新 Poisson 回归模型的参数。

5.2 poisson2multinom.R

利用 Poisson NMF 和多项主题模型之间的关系，从 Poisson NMF 拟合中恢复多项式主题模型拟合。

输入参数是类 poisson_nmf_fit 的对象，比如 init_poisson_nmf 或 fit_poisson_nmf 的输出。输出是类 multinom_topic_model_fit 的对象。

实现的主要函数是 poisson2multinom。

```
poisson2multinom <- function (fit) {

  if (inherits(fit,"multinom_topic_model_fit"))
    return(fit)
  if (!inherits(fit,"poisson_nmf_fit"))
    stop("Input argument \"fit\" should be an object of class ",
         "\"poisson_nmf_fit\"")
  verify.fit(fit)
  if (ncol(fit$F) < 2 | ncol(fit$L) < 2)
    stop("Input matrices \"fit$F\" and \"fit$L\" should have 2 or more",
         "columns")

  out   <- get_multinom_from_pnmf(fit$F,fit$L)
  fit$F <- out$F
  fit$L <- out$L
  fit$s <- out$s

  class(fit) <- c("multinom_topic_model_fit","list")
  return(fit)
}
```

为了实现函数 poisson2multinom,我们需要辅助函数 get_multinom_from_pnmf。

```
get_multinom_from_pnmf <- function (F, L) {
  u <- colSums(F)
  F <- scale.cols(F,1/u)
```

```
  L <- scale.cols(L,u)
  s <- rowSums(L)
  L <- L / s
  return(list(F = F,L = L,s = s))
}
```

5.3 fit_poisson_nmf.R

Poisson NMF 问题：计数矩阵 $X \in \mathbb{R}^{n\times m}$，非负矩阵参数 $L \in \mathbb{R}_+^{n\times K}$，$F \in \mathbb{R}_+^{m\times K}$，对 $K \geq 1$，满足

$$x_{ij}|L,F \sim Poisson(\lambda_{ij}),$$

$$\lambda_{ij} = (LF^T)_{ij} = \sum_{k=1}^{K} l_{ik}f_{jk},$$

并分别使用 EM、CCD、SCD 解决该问题。

使用三种方式来衡量模型的进展：对数似然（或偏差）的改进、模型参数的变化、KKT 系统的残差。当迭代接近损失函数的（局部）稳定点时，模型参数应当趋于不变、KKT 系统的残差应当消失。使用 `plot_progress` 查看模型进程。

输入参数表。

参数	描述
`X`	n*m 的非负矩阵。可以是稀疏矩阵 `dgCMatrix` 或稠密矩阵 `matrix`，但有一些例外。
`k`	矩阵的秩。是预先设定的大于等于 2 的整数。设定初始值（`fit0` 或 `F`、`L`）时忽略此选项。
`fit0`	初始模型拟合。是类 `poisson_nmf_fit` 的对象，比如 `init_poisson_nmf` 或 `fit_poisson_nmf` 的输出。
`numiter`	迭代次数。是 `F` 和 `L` 的更新次数。
`update.`	更新特定的 `F` 行。默认更新所有的 `F` 行。此选项只针对方法 `em` `factors` 和 `scd`。

参数	描述
update.loadings	更新特定的 L 行。默认更新所有的 L 行。此选项只针对方法 em 和 scd。
method	更新方法。实现了四种方法：乘法更新 mu，期望最大化 em，顺序坐标下降 scd，循环坐标下降 ccd。
init.method	初始化 F 和 L 的方法。实现了两种方法：random 表示均匀随机初始化 F 和 L，topicscore 表示使用（非常快的）Topic SCORE 算法初始化 F、通过运行少量 SCD 初始化 L。设定初始值（fit0 或 F、L）时忽略此选项。
control	控制算法的参数列表。用于控制优化算法和 Topic SCORE 算法（若选择 topicscore）。详见 control 表。
verbose	进度信息控件。实现了三个选项：detail 表示打印每次迭代的信息，progressbar 表示使用进度条显示算法进程，none 表示不显示进度信息。

control 表。

参数	描述
numiter	内部循环的迭代次数。此选项在方法 mu 和 ccd 下必须设置为 1。
nc	用于更新的 RcppParallel 线程数。nc 为 NA 时线程数调用 RcppParallel 的 defaultNumThreads。此选项忽略乘法更新 mu。
minval	用来保护乘法更新的小正常数。具体代码为 F <- pmax(F1,minval) 和 L <- pmax(L1,minval)。允许设置 minval = 0 此时将给出一个警告。
eps	为避免计算零的对数而添加的小正常数。增大此选项的值会加快收敛速度但也会影响优化的精度。
zero.threshold	判定是否为 0 的小正常数。将小于或等于阈值 zero.threshold 的项判定为 0。

init_poisson_nmf 和 fit_poisson_nmf 返回值表。

参数	描述
F	F 的最佳估计值。
L	L 的最佳估计值。
loss	最佳估计值处的损失函数值。
iter	已完成的迭代数量。
progress	算法进度详细信息的数据表。数据表有 numiter 行。数据表的列详见 progress 表。

progress 表。

参数	描述
iter	迭代次数
loglik	当前最佳点的 Poisson NMF 模型的对数似然。
loglik. multinom	当前最佳点的多项式主题模型的对数似然。
dev	当前最佳点的偏差。
res	当前最佳点的 KKT 一阶最优条件的残差。
delta.f	F 的最大变化。
delta.l	L 的最大变化。
nonzeros.f	F 中非零元素所占的比例。
nonzeros.l	L 中非零元素所占的比例。
timing	以秒为单位的运行时间。

实现的主要函数是 fit_poisson_nmf。

首先，输入相应参数。

```
fit_poisson_nmf <- function (X, k, fit0, numiter = 100,
                             update.factors = seq(1,ncol(X)),
                             update.loadings = seq(1,nrow(X)),
                             method = c("scd","em","mu","ccd"),
```

```
                                init.method = c("topicscore","random"),
                                control = list(),
                                verbose = c("progressbar","detailed","none")) {
```

其次，数据检查与预处理。

再次，初始化估计值。

```
  fit <- rescale.fit(fit)
  fit <- safeguard.fit(fit,control$minval)
```

最后，更新参数值。

```
  if (verbose == "detailed")
    cat("iter loglik(PoisNMF) loglik(multinom) res(KKT)  |F-F'|  |L-L'|",
        "nz(F) nz(L) beta\n")
  fit <- fit_poisson_nmf_main_loop(X,fit,numiter,update.factors,
                                   update.loadings,method,control,
                                   verbose)

  # Output the updated "fit".
  fit$progress     <- rbind(fit0$progress,fit$progress)
  dimnames(fit$F)  <- dimnames(fit0$F)
  dimnames(fit$L)  <- dimnames(fit0$L)
  dimnames(fit$Fn) <- dimnames(fit0$Fn)
  dimnames(fit$Ln) <- dimnames(fit0$Ln)
  dimnames(fit$Fy) <- dimnames(fit0$Fy)
  dimnames(fit$Ly) <- dimnames(fit0$Ly)
  class(fit) <- c("poisson_nmf_fit","list")
  return(fit)
}
```

为了实现函数 `fit_poisson_nmf_main_loop`，我们需要以下辅助函数，列表如下。

函数	描述
update_poisson_nmf	F 和 L 的单次更新。
update_factors_poisson_nmf	F 的单次更新。
update_loadings_poisson_nmf	L 的单次更新。
rescale.fit	重新调整当前最佳的 F 和 L。
safeguard.fit	强制使 F 和 L 是正的。
fit_poisson_nmf_control_default	control 的默认值。

具体代码如下。

函数 *update_poisson_nmf*

```
update_poisson_nmf <- function (X, fit, update.factors, update.loadings,
                                method, control) {

  # Compute L.
  if (length(update.loadings) > 0) {
    fit$L <- update_loadings_poisson_nmf(X,fit$F,fit$L,update.loadings,
                                         method,control)
    fit$L <- pmax(fit$L,control$minval)
  }

  # Compute F.
  if (length(update.factors) > 0) {
    fit$F <- update_factors_poisson_nmf(X,fit$F,fit$L,update.factors,
                                        method,control)
    fit$F <- pmax(fit$F,control$minval)
  }
```

```
  # fit$Fy <- fit$F
  # fit$Ly <- fit$L
  # fit$Fn <- fit$F
  # fit$Ln <- fit$L

  fit <- rescale.fit(fit)

  # Compute loss.
  fit$loss <- sum(cost(X,fit$L,t(fit$F),control$eps))
  # fit$loss.fnly <- fit$loss

  return(fit)
}
```

函数 *update_factors_poisson_nmf*

```
update_factors_poisson_nmf <- function (X, F, L, j, method, control) {
  numiter <- control$numiter
  nc      <- control$nc
  eps     <- control$eps
  if (method == "mu")
    F <- t(betanmf_update_factors(X,L,t(F)))
  else if (method == "em")
    F <- pnmfem_update_factors(X,F,L,j,numiter,nc)
  else if (method == "ccd")
    F <- t(ccd_update_factors(X,L,t(F),nc,eps))
  else if (method == "scd")
    F <- t(scd_update_factors(X,L,t(F),j,numiter,nc,eps))
  return(F)
}
```

函数 *update_loadings_poisson_nmf*

```
update_loadings_poisson_nmf <- function (X, F, L, i, method, control) {
  numiter <- control$numiter
  nc      <- control$nc
  eps     <- control$eps
  if (method == "mu")
    L <- betanmf_update_loadings(X,L,t(F))
  else if (method == "em")
    L <- pnmfem_update_loadings(X,F,L,i,numiter,nc)
  else if (method == "ccd")
    L <- ccd_update_loadings(X,L,t(F),nc,eps)
  else if (method == "scd")
    L <- scd_update_loadings(X,L,t(F),i,numiter,nc,eps)
  return(L)
}
```

函数 *rescale.fit*

```
rescale.fit <- function (fit) {

  # Rescale the "current best" factors and loadings.
  out    <- rescale.factors(fit$F,fit$L)
  fit$F  <- out$F
  fit$L  <- out$L

  # # Rescale the non-extrapolated factors and loadings.
  # out    <- rescale.factors(fit$Fn,fit$Ln)
  # fit$Fn <- out$F
  # fit$Ln <- out$L
  #
  # # Rescale the extrapolated factors and loadings.
  # out    <- rescale.factors(fit$Fy,fit$Ly)
  # fit$Fy <- out$F
  # fit$Ly <- out$L
```

```
  return(fit)
}
```

函数 *safeguard.fit*

```
safeguard.fit <- function (fit, minval) {
  fit$F  <- pmax(fit$F,minval)
  fit$L  <- pmax(fit$L,minval)
  fit$Fn <- pmax(fit$Fn,minval)
  fit$Ln <- pmax(fit$Ln,minval)
  fit$Fy <- pmax(fit$Fy,minval)
  fit$Ly <- pmax(fit$Ly,minval)
  return(fit)
}
```

函数 *fit_poisson_nmf_control_default*

```
fit_poisson_nmf_control_default <- function()
  list(numiter           = 4,
       init.numiter      = 10,
       minval            = 1e-10,
       eps               = 1e-8,
       zero.threshold    = 1e-6,
       nc                = 1,
       extrapolate       = FALSE,
       extrapolate.reset = 20,
       beta.increase     = 1.1,
       beta.reduce       = 0.75,
       betamax.increase  = 1.05)
```

函数 *fit_poisson_nmf_main_loop*【核心代码】

```
fit_poisson_nmf_main_loop <- function (X, fit, numiter, update.factors,
                                       update.loadings, method, control,
                                       verbose) {
  if (verbose == "progressbar")
    pb <- progress_bar$new(total = numiter)

  # Pre-compute quantities and set up data structures used in the loop below.
  loglik.const    <- sum(loglik_poisson_const(X))
  dev.const       <- sum(deviance_poisson_const(X))
  progress        <- as.data.frame(matrix(0,numiter,13))
  names(progress) <- c("iter","loglik","loglik.multinom","dev","res",
                       "delta.f","delta.l","nonzeros.f","nonzeros.l",
                       "extrapolate","beta","betamax","timing")

  # Iterate the updates of the factors and loadings.
  for (i in 1:numiter) {
    fit0 <- fit
    t1   <- proc.time()
    if (verbose == "progressbar")
      pb$tick()

    # Update the factors and loadings.
    if (control$extrapolate &
        fit$beta > 0 &
        i %% control$extrapolate.reset != 0) {

      # Perform an "extrapolated" update of the factors and loadings.
      extrapolate <- TRUE
      fit <- update_poisson_nmf_extrapolated(X,fit,update.factors,
                                             update.loadings,method,
                                             control)
    } else {
```

```
    # Perform a basic coordinate-wise update of the factors and loadings.
    extrapolate <- FALSE
    fit <- update_poisson_nmf(X,fit,update.factors,update.loadings,
                              method,control)
  }
  t2 <- proc.time()

  # Update the iteration number.
  fit$iter <- fit$iter + 1

  # Update the "progress" data frame.
  progress[i,"iter"]        <- fit$iter
  progress[i,"loglik"]      <- loglik.const - fit$loss
  progress[i,"loglik.multinom"] <-
    loglik.const - fit$loss - sum(loglik_size_factors(X,fit$F,fit$L))
  progress[i,"dev"]         <- dev.const + 2*fit$loss
  progress[i,"res"]         <- with(poisson_nmf_kkt(X,fit$F,fit$L),
                                    max(abs(rbind(F[update.factors,],
                                                  L[update.loadings,]))))
  progress[i,"delta.f"]     <- max(abs(fit$F - fit0$F))
  progress[i,"delta.l"]     <- max(abs(fit$L - fit0$L))
  progress[i,"beta"]        <- fit$beta
  progress[i,"betamax"]     <- fit$betamax
  progress[i,"timing"]      <- t2["elapsed"] - t1["elapsed"]
  progress[i,"nonzeros.f"]  <- mean(fit$F > control$zero.threshold)
  progress[i,"nonzeros.l"]  <- mean(fit$L > control$zero.threshold)
  progress[i,"extrapolate"] <- extrapolate
  if (verbose == "detailed")
    cat(sprintf("%4d %+0.8e %+0.9e %0.2e %0.1e %0.1e %0.3f %0.3f %0.2f\n",
                fit$iter,progress[i,"loglik"],progress[i,"loglik.multinom"],
                progress[i,"res"],progress[i,"delta.f"],
                progress[i,"delta.l"],progress[i,"nonzeros.f"],
                progress[i,"nonzeros.l"],extrapolate * progress[i,"beta"]))
```

```
  }

  # Output the updated "fit".
  fit$progress <- progress
  return(fit)
}
```

5.4 fit_topic_model.R

适合大或者复杂的数据集。

模型拟合为分四个步骤完成：使用型 `init_poisson_nmf` 初始化 Poisson NMF 模型，使用 `fit_poisson_nmf` 运行 100 次 EM 更新，使用 `fit_poisson_nmf` 运行运行 100 次外推 SCD 更新，使用 `poisson2multinom` 恢复多项式主题模型。

EM 能快速到达最优值附近，SCD 能快速收敛到最优值。对于较大的数据集可能需要更多次 EM 迭代。

输入参数表。

参数	描述
`X`	n*m 的非负矩阵。可以是稀疏矩阵 `dgCMatrix` 或稠密矩阵 `matrix`。
`k`	矩阵的秩。是预先设定的大于等于 2 的整数。
`numiter.main`	主要步骤迭代次数。
`numiter.refine`	细化步骤迭代次数。
`method.main`	主要步骤更新方法。
`method.refine`	细化步骤更新方法。
`init. method`	初始化 `F` 和 `L` 的方法。
`control.init`	控制初始化算法的参数列表。
`control.main`	控制主要步骤算法的参数列表。
`control.refine`	控制细化步骤算法的参数列表。
`verbose`	进度信息控件。

输出是由函数 poisson2multinom 生成的多项主题模型的拟合。

```
fit_topic_model <-
  function (X, k, numiter.main = 100, numiter.refine = 100, method.main = "em",
            method.refine = "scd", init.method = c("topicscore","random"),
            control.init = list(), control.main = list(numiter = 4),
            control.refine = list(numiter = 4,extrapolate = TRUE),
            verbose = c("progressbar","detailed","none")) {

  # Check the input data matrix.
  verify.count.matrix(X)

  # Check and process input argument "verbose".
  verbose <- match.arg(verbose)

  # If necessary, remove all-zero columns from the counts matrix.
  if (any.allzero.cols(X)) {
    X <- remove.allzero.cols(X)
    warning(sprintf(paste("One or more columns of X are all zero; after",
                          "removing all-zero columns, %d columns will be",
                          "used for model fitting"),ncol(X)))
  }

  # Initialize the Poisson NMF model fit.
  fit <- init_poisson_nmf(X,k = k,init.method = init.method,
                          control = control.init,
                          verbose = ifelse(verbose == "none",
                                           "none","detailed"))

  # Perform the main model fitting step.
  fit <- fit_poisson_nmf(X,fit0 = fit,numiter = numiter.main,
                         method = method.main,control = control.main,
                         verbose = verbose)
```

```
  # Perform the model refinement step.
  if (numiter.refine > 0) {
    if (verbose != "none")
      cat("Refining model fit.\n")
    fit <- fit_poisson_nmf(X,fit0 = fit,numiter = numiter.refine,
                           method = method.refine,control = control.refine,
                           verbose = verbose)
  }

  # Output the multinomial topic model fit.
  return(poisson2multinom(fit))
  }
```

6 绘图函数

6.1 structure_plot.R

绘制结构图。

输入参数表。

参数	描述
fit	poisson_nmf_fit 或 multinom_topic_model_fit 类的对象。若提供前者，则利用 poisson2multinom 恢复为后者。
topics	从上到下的主题排序。若不特别指定，默认最大的主题在最底下。
grouping	可选的分类变量。对样本行进行分组。
loadings_order	加载矩阵 L 的行顺序。
n	最大样本数。
colors	结构图中用于绘制主题的颜色。
gap	组之间的水平间距。默认无间距。
embed_method	用于计算一维嵌入的函数。

参数	描述
ggplot_call	用于创建图形的函数。

实现的主要函数如下。

```
plot.poisson_nmf_fit <- function (x, ...)
  structure_plot(x,...)

plot.multinom_topic_model_fit <- function (x, ...)
  structure_plot(x,...)
```

为了实现函数 structure_plot，我们需要以下辅助函数，列表如下。

函数	描述
structure_plot_default_embed_method	默认方法。
structure_plot_ggplot_call	绘制图像。
compile_structure_plot_data	创建适合 ggplot 的数据表。
compile_grouped_structure_plot_data	分组情况下创建适合 ggplot 的数据表。

我们逐个分析。

函数 *structure_plot_default_embed_method*

```
structure_plot_default_embed_method <- function (fit,...) {
  if (nrow(fit$L) < 20)
    return(rnorm(nrow(fit$L)))
  else {
    d <- dim(fit$L)
    message(sprintf("Running tsne on %s x %s matrix.",d[1],d[2]))
    return(drop(suppressMessages(tsne_from_topics(fit,dims = 1,...))))
  }
}
```

函数 *structure_plot_ggplot_call*

需要载入包 ggplot2。

输入参数表。

函数	描述
dat	ggplot 所要求的数据表。至少包括三列：sample 包含样本（L 矩阵的行）在水平轴上的位置，topic 包含一个主题（L 的一列），prop 包含每个样本的主题比例。
ticks	沿水平轴放置的组标签名称。若数据未进行分组则为 NULL。
font.size	图中使用的字体大小。

```
structure_plot_ggplot_call <- function (dat, colors, ticks = NULL,
                                        font.size = 9)
  ggplot(dat,aes_string(x = "sample",y = "prop",color = "topic",
                        fill = "topic")) +
    geom_col() +
    scale_x_continuous(limits = c(0,max(dat$sample) + 1),breaks = ticks,
                       labels = names(ticks)) +
    scale_color_manual(values = colors) +
    scale_fill_manual(values = colors) +
    labs(x = "",y = "topic proportion") +
    theme_cowplot(font.size) +
    theme(axis.line    = element_blank(),
          axis.ticks   = element_blank(),
          axis.text.x = element_text(angle = 45,hjust = 1))
```

函数 *compile_structure_plot_data*

输入参数表。

函数	描述
L	主题比例矩阵。

函数	描述
topics	所选主题的向量。即所选 L 列。

输出是包含三列的数据表。

函数	描述
sample	数值型的 L 的行。
topic	主题。
prop	数值型的给定样本的主题比例。

```
compile_structure_plot_data <- function (L, topics) {
  n <- nrow(L)
  k <- length(topics)
  dat <- data.frame(sample = rep(1:n,times = k),
                    topic  = rep(topics,each = n),
                    prop   = c(L[,topics]))
  dat$topic <- factor(dat$topic,topics)
  return(dat)
}
```

函数 compile_grouped_structure_plot_data

相对于函数 compile_structure_plot_data 需要再输入 gap。

```
compile_grouped_structure_plot_data <- function (L, topics, grouping,
                                                 gap = 0) {
  ticks <- rep(0,nlevels(grouping))
  names(ticks) <- levels(grouping)
  dat <- NULL
  m <- 0
  for (j in levels(grouping)) {
    i          <- which(grouping == j)
    out        <- compile_structure_plot_data(L[i,,drop = FALSE],topics)
```

```
    out$sample <- out$sample + m
    n          <- length(i)
    dat        <- rbind(dat,out)
    ticks[j]   <- m + n/2
    m          <- m + n + gap
  }
  return(list(dat = dat,ticks = ticks))
}
```

函数 *structure_plot*【核心代码】

```
structure_plot <-
  function (fit, topics, grouping, loadings_order = "embed", n = 2000,
            colors = c("#e41a1c","#377eb8","#4daf4a","#984ea3","#ff7f00",
                       "#ffff33","#a65628","#f781bf","#999999"),
            gap = 1, embed_method = structure_plot_default_embed_method,
            ggplot_call = structure_plot_ggplot_call, ...) {
```

首先，检查输入变量。

其次，处理数据并绘制结构图。

```
  # Prepare the data for plotting and create the structure plot.
  fit$L <- fit$L[loadings_order,]
  grouping <- grouping[loadings_order,drop = TRUE]
  if (nlevels(grouping) == 1) {
    dat <- compile_structure_plot_data(fit$L,topics)
    return(ggplot_call(dat,colors))
  } else {
    out <- compile_grouped_structure_plot_data(fit$L,topics,grouping,gap)
    return(ggplot_call(out$dat,colors,out$ticks))
  }
  }
```

6.2 other_plots.R

利用 fit 结果的输出 progress 绘制评价指标的曲线。

实现的主要函数是 plot_progress 和 plot_loglik_vs_rank。

为了实现上述主要函数，我们需要以下辅助函数，列表如下。

函数	描述
prepare_progress_plot_data	为绘制评价指标与迭代次数的曲线图制作适合 ggplot 的数据集。
create_progress_plot	绘制评价指标与迭代次数的曲线图。
loglik_vs_rank_ggplot_call	绘制评价指标与主题数目 K 的曲线图。

具体代码如下。

函数 *prepare_progress_plot_data*

```
prepare_progress_plot_data <- function (fits, e) {
  n      <- length(fits)
  labels <- names(fits)
  for (i in 1:n) {
    y         <- fits[[i]]$progress
    y         <- cbind(data.frame("method" = labels[i]),y)
    y$timing  <- cumsum(y$timing)
    fits[[i]] <- y
  }
  out                 <- do.call(rbind,fits)
  out$method          <- factor(out$method,labels)
  out$loglik          <- max(out$loglik) - out$loglik + e
  out$loglik.multinom <- max(out$loglik.multinom) - out$loglik.multinom + e
  out$dev             <- out$dev - min(out$dev) + e
  return(out)
}
```

函数 *create_progress_plot*

```
create_progress_plot <- function (pdat, x, y, add.point.every, colors,
                                  linetypes, linesizes, shapes, fills,
                                  theme) {
  rows <- which(pdat$iter %% add.point.every == 1)
  if (x == "timing")
    xlab <- "runtime (s)"
  else if (x == "iter")
    xlab <- "iteration"
  if (y == "res")
    ylab <- "max KKT residual"
  else if (y == "dev")
    ylab <- paste("distance from best deviance")
  else if (y == "loglik" | y == "loglik.multinom")
    ylab <- paste("distance from best loglik")
  return(ggplot(pdat,aes_string(x = x,y = y,color = "method",
                                linetype = "method",size = "method")) +
         geom_line(na.rm = TRUE) +
         geom_point(data = pdat[rows,],
                    mapping = aes_string(x = x,y = y,color = "method",
                                         fill = "method",shape = "method"),
                    inherit.aes = FALSE,na.rm = TRUE) +
         scale_y_continuous(trans = "log10") +
         scale_color_manual(values = colors) +
         scale_linetype_manual(values = linetypes) +
         scale_size_manual(values = linesizes) +
         scale_shape_manual(values = shapes) +
         scale_fill_manual(values = fills) +
         labs(x = xlab,y = ylab) +
         theme())
}
```

函数 *loglik_vs_rank_ggplot_call*

```
loglik_vs_rank_ggplot_call <- function (dat, font.size = 9)
  return(ggplot(dat,aes_string(x = "x",y = "y")) +
         geom_line() +
         geom_point() +
         scale_x_continuous(breaks = dat$x) +
         labs(x = "rank, or number of topics (k)",
              y = "log-likelihood difference") +
         theme_cowplot(font.size))
```

6.3 embedding_plots.R 和 volcano_plots.R

embedding_plots.R 实现基于 PCA 的分析文本主题的算法，并依据 PCA 将文本根据主题聚合在一起以绘制结构图。

volcano_plots.R 实现绘制文本的主题分布的火山图。

7 数据模拟函数

datasim.R 实现模拟基因数据，这仅仅针对单细胞的 RNA 有效，当然与二倍体的 DNA 基因也有密切联系。主要实现了模拟基于 Poisson NMF 模型和主题模型的基因数据。使用了库 stats 和 MCMCpack 以调用分布函数。比如 runif、rnorm、rmultinom、rpois、rdirichlet。