scDEcrypter: Detecting latent viral states in scRNA-seq

Luer Zhong

2025-11-20

Introduction

Detecting infected cells in viral scRNA-seq data is challenging because viral transcripts are often sparse or undetected, leaving most infection labels unknown and limiting downstream differential expression analyses. scDEcrypter is a statistical framework we developed to address this problem by modeling infection status, and optionally other partitioning variables such as cell type. The method uses a penalized multiway mixture model, anchored by a small set of confidently labeled cells, to recover latent infection states with high accuracy even when viral reads are extremely limited. To avoid double-dipping, scDEcrypter employs a data-splitting strategy, using one subset of cells for parameter estimation and another independent subset for differential expression testing. A fast approximate likelihood ratio test is then used to identify infection-associated genes within each cell type. scDEcrypter can improve power, robustness, and biological interpretability. If you use scDEcrypter in published research, please cite:

Run scDEcrypter

Install the package

Before analysis can be performed, scDEcrypter must be installed. The easiest way to install scDEcrypter is through github:

```
library(devtools)
devtools::install_github("https://github.com/LZHONG25/scDEcrypter")
library(scDEcrypter)
```

Required inputs

Load the Seurat object. We analyzed the COVID-19 nasal wash dataset described in
Gao, Kevin M., *et al.* (2021). *Human nasal wash RNA-Seq reveals distinct cell-specific innate immune responses in influenza versus SARS-CoV-2.* **JCI Insight**, 6(22): e152288.

```
seed <- sample(63321232,1)
set.seed(seed)
load(".../covid_full_raw_seu.Rdata")
# the seurat object covid_full_raw_seu.Rdata is saved as seurat_obj
```

scDEcrypter requires one additional cell-level label, which we refer to as the partitioning variable. This variable defines groups of cells that are expected to differ in their baseline expression patterns, and it helps

the model separate biological heterogeneity unrelated to infection status. Importantly, the partitioning variable is flexible and does not need to be cell type. It can represent any categorical cell-level attribute that the user believes is relevant.

In the example shown here, we use broad cell-type categories as the partitioning variable. Fine-grained annotations from the Seurat object are collapsed into six coarse groups (T cells, macrophages, B cells, epithelial cells, dendritic cells, and neutrophils). Users may adapt or replace this mapping entirely depending on the structure of their own dataset. Make sure that the partitioning variable is numerically coded.

```
seurat_obj$CellType_Label <- ifelse(seurat_obj$CellType_Fine %in%
    c("CD4_T", "CD8_T", "prolifT"), 1, ifelse(seurat_obj$CellType_Fine %in%
    c("alveol-mac", "IFNexp-mac", "M1-mac",
        "M1-mac-exp", "M2-mac"), 2, ifelse(seurat_obj$CellType_Fine %in%
    c("memoryB1", "naiveB", "transB", "plasma",
        "ABC"), 3, ifelse(seurat_obj$CellType_Fine %in%
    c("basal", "ciliated", "goblet+club",
        "hillock"), 4, ifelse(seurat_obj$CellType_Fine %in%
    c("pDC"), 5, ifelse(seurat_obj$CellType_Fine %in%
    c("G5a_naive", "G5b", "G5c_aged", "G5c_naive"),
    6, NA))))))
```

scDEcrypter requires a second cell-level variable that indicates the infection state of each cell. In many single-cell studies, this information is only partially observed, some cells can be confidently labeled as infected, some cells can be confidently labeled as uninfected, and the rest is unknown.

scDEcrypter is designed to model this partially observed structure: users provide the labels they know, leave the rest as NA. Make sure to label the "infected" cells as 1, and "uninfected" cells as 2, and the rest unknown as NA.

```
C.obs <- seurat_obj$CellType_Label

V.obs <- rep(NA, ncol(seurat_obj))
to_label_I <- which(seurat_obj$Infection_byViralCounts == "Infected")
to_label_U <- which(seurat_obj$Group == "Healthy")
V.obs[to_label_I] <- 1
V.obs[to_label_U] <- 2
```

Assign the total number of unique infection statuses and cell types. Note: for this Covid data, we observed potential "bystander" cells besides "infected" and "uninfected", and we also wanted to know which cells were bystander, so we assigned the number of infection status as 3.

```
V.star <- 3
C.star <- 6
```

Run the pre-processing function

After running this step, scDEcrypter will split the data into one generation set - for parameter estimation, and one test set - for inference. The inputs are already gathered from the previous steps, and users can choose the variance stablization methods, we use "shifted_log" here.

```
prep <- preprocess_scDEcrypter(seurat_obj, C.obs, V.obs, seed, vs_method = "shifted_log")

Y_generation_stbl <- prep$Y_generation
```

```
Y_test_stbl <- prep$Y_test
splitted_data <- prep$splitted
```

Current method: optimizing for highly variable genes (HVG) across cells within covid samples, within cell types. Users can choose other HVG selection methods, but users must perform HVG selection only on the generation set.

```
vargroup <- na.omit(seurat_obj$CellType_Label[splitted_data$generation_idx])
covgroup <- names(which(seurat_obj$Group[splitted_data$generation_idx] == "COVID-19"))

top_genesA <- sapply(unique(vargroup), function(x)
  apply(Y_generation_stbl[intersect(covgroup, names(which(vargroup==x))),], 2, var))

topidx <- apply(top_genesA, 2, function(x) order(x, decreasing = T)[1:500])
topidx_g <- apply(topidx, 2, function(x) rownames(top_genesA)[x])
top_genes.HVG <- unique(as.vector(apply(topidx_g, 2, function(x) x[1:50])))
```

Select the best tuning parameter

Users can self-define a range of potential lambdas, and fit them altogether in the lambda selection function. Users can also choose the maximal number of iterations, and the tolerance for the function to converge. The rest information that the lambda selection function needed is already provided from the previous steps.

```
lambda.vec <- c(1e2, 1, 0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8)
cv_lambda <- cross_validate_lambda(Y_generation_stbl[,top_genes.HVG],
                                   splitted_data$C.obs_generation,
                                   splitted_data$V.obs_generation,
                                   lambda.vec, max.iter = 200,
                                   tol = 1e-8, C.star, V.star, seed,
                                   NCORES = 5)
best_lambda <- cv_lambda$best_lambda
```

Run scDEcrypter

We run scDEcrypter on the pre-processed generation set, with the best tuning parameter. All the results will be saved for downstream infection status prediction and differential expressed gene analysis.

```
results_generation <- MultiwayMixture(Y = Y_generation_stbl[,top_genes.HVG],
                                      C.obs = splitted_data$C.obs_generation,
                                      V.obs = splitted_data$V.obs_generation,
                                      max.iter = 200, tol = 1e-8,
                                      C.star, V.star, seed,
                                      lambda.vec=dim(splitted_data$Y_generation)[1]
                                      *best_lambda)
```

```
cutoffs <- .9
cell_thresh <- .75
```

```r
M.1 <- results_generation$M_list[[1]]
sigma2.1 <- results_generation$sigma2_list[[1]]
probs.1 <- results_generation$probs_list[[1]]
W.1 <- results_generation$weights_list[[1]]

W.2.test <- E.step1(Y_test_stbl[, top_genes.HVG], splitted_data$C.obs_test,
    splitted_data$V.obs_test, M.1, probs.1, sigma2.1)

W_max_byC <- apply(W.2.test, c(1, 2), max, na.rm = TRUE)
CellType_pred <- max.col(W_max_byC, ties.method = "first")
CellType_pred[which(apply(W_max_byC, 1, max) < cell_thresh)] <- 0

temp1 <- as.data.frame(W.2.test[, , 1, drop = F])  # infected
temp2 <- as.data.frame(W.2.test[, , 2, drop = F])  # uninfected
temp3 <- as.data.frame(W.2.test[, , 3, drop = F])  # bystander

temp1$Prediction_Infected <- (ifelse(temp1$V1 >= cutoffs |
    temp1$V2 >= cutoffs | temp1$V3 >= cutoffs | temp1$V4 >=
    cutoffs | temp1$V5 >= cutoffs | temp1$V6 >= cutoffs,
    "Infected", "Unknown"))
temp2$Prediction_Uninfected <- (ifelse(temp2$V1 >=
    cutoffs | temp2$V2 >= cutoffs | temp2$V3 >= cutoffs |
    temp2$V4 >= cutoffs | temp2$V5 >= cutoffs | temp2$V6 >=
    cutoffs, "Uninfected", "Unknown"))
temp3$Prediction_Bystander <- (ifelse(temp3$V1 >= cutoffs |
    temp3$V2 >= cutoffs | temp3$V3 >= cutoffs | temp3$V4 >=
    cutoffs | temp3$V5 >= cutoffs | temp3$V6 >= cutoffs,
    "Bystander", "Unknown"))
temp1$Full_Prediction <- "Unknown"
temp1$Full_Prediction[which(temp1$Prediction_Infected ==
    "Infected")] <- "Infected"
temp1$Full_Prediction[which(temp2$Prediction_Uninfected ==
    "Uninfected")] <- "Uninfected"
temp1$Full_Prediction[which(temp3$Prediction_Bystander ==
    "Bystander")] <- "Bystander"
```