

SciLifeLab Bioinformatics LTS / Bioinformatics LTS / scRNAseq labs / Source

pagoda_ilc.md

master ▾



scRNAseq labs / pagoda_ilc.md

Source

Diff

History

c241e2e 2017-10-04 ▾ Full commit

Blame

Embed

Raw

Edit

layout: default title: 'Pagoda'

Pagoda pathway wPCA

Clustering of data using Pagoda in the SCDE package, following tutorial at: <http://hms-dbmi.github.io/scde/pagoda.html>

For this exercise you can either run with your own data or with the example data from Pollen paper that they provide with the package. Below is an example with human innate lymphoid cells (ILCs) from Bjorklund et al. 2016 where you can get some hints on how to run with your own data.

Notes on running SCDE

With a large dataset, it may be a good idea to run all the cpu/memory-intensive steps in the SCDE package on Uppmax, or other HPC center, submitted as an Sbatch job calling an R-script. OBS! The SCDE package uses a lot of memory, so even if you request 16 cpus (a full node) on Uppmax you will have issues with memory overflow if you run the program on all 16 cores. Suggested setting would be to use `n.cores = 10` when running on a normal node or specify nodes with more memory.

But the memory consumption could of course depend on the dataset, so you should monitor your cpu and memory usage using the `jobstats` command.

Example with ILC data

If you want to run this example, all data plus some intermediate files for steps that takes long time, is located in folder:

```
/proj/b2013006/nobackup/scrnaseq_course/data/ILC
```

Load packages

```
suppressMessages(library(scde))
```

Read data

```
# read in meta data table and create pheno data
M <- read.table("data/ILC/Metadata_ILC.csv", sep=";", header=T)

# read count values, SCDE require counts and not normalized rpkm's.
C <- read.table("data/ILC/ensembl_countvalues_ILC.csv", sep=";", header=T)
```

Define some color scales

Define color scales based on celltype and donor.

```
celltype2cols <- c("blue", "cyan", "red", "green")[as.integer(M$Celltype)]
donor2cols <- c("black", "orange", "magenta")[as.integer(M$Donor)]
```

Get GO-terms

In the [Tutorial for biomaRt](#), there is an example on how to fetch go-terms for Ensembl genes. In the file `GO_BP_annotations.Rdata` this was done for all genes in the ILC dataset and selecting for Biological Process entries in GO.

In the tutorial there is an example where they fetch first 100 go-terms in `org.Hs.egGO2ALLEGS`, plus some selected ones related to neurogenesis. In their example they translate gene symbol to NCBI gene id to then fetch go-terms. There are several different ways of getting go-terms for genes, but if you are working with Ensembl IDs the most straight forward is probably to use `biomaRt`.

More examples on how to get GO-terms for gene symbols can be found at: <http://hms-dbmi.github.io/scde/genesets.html>

```
# load go-term annotations for all genes
load("data/ILC/GO_BP_annotations.Rdata", verbose=TRUE)

## Loading objects:
## goBP2gene
```

Filter with SCDE `clean.counts` and go-terms with `clean.gos`

The `clean.counts` function will remove genes and cells based on specified cutoffs:

- `min.lib.size` - minimum number of genes detected per cell (default 1800)
- `min.reads` - minimum number of reads per gene (default 10)
- `min.detected` - minimum number of cells a gene must be detected in (default 5)

```
cd <- clean.counts(C, min.lib.size = 1000, min.reads = 10, min.detected = 5)
genes <- rownames(cd)
```

```
# now filter out all genes that are not included in cd from the go-annotations
go.env <- lapply(goBP2gene, function(x) x[x %in% genes])
```

```
# remove GOs with too few or too many genes
go.env <- clean.gos(go.env, min.size=5, max.size=2000)
```

```
# convert to an environment
go.env <- list2env(go.env)
```

Fit error models

OBS! it takes a while to run on 1 cpu, around an hour with this dataset.

```
# since this step takes a while, save data to a file so that it does not have to be rerun
savefile <- "data/ILC/pagoda_knn.Rdata"
if (file.exists(savefile)){
  load(savefile)
}else {
  pdf("data/ILC/pagoda.cell.models.pdf")
  knn <- knn.error.models(cd, k = ncol(cd)/4, n.cores = 1, min.count.threshold = 2, mi
  dev.off()
  save(knn, file=savefile)
}
```








Normalizing variance and effect of gene detection

In this case, we know that we have a clear batch effect from the 3 donors, so we can also include batch information in the normalization step.

This step also takes a while to run.

In addition, we add in a step where the number of detected genes per cell, which may be a technical artefact.

scRNAseq labs

-  Overview
-  **Source**
-  Commits
-  Branches
-  Pull requests
-  Pipelines
-  Downloads

```
savefile <- "data/ILC/pagoda_varnorm.Rdata"
if (file.exists(savefile)){
  load(savefile)
}else {
  pdf("data/ILC/pagoda_varnorm.pdf")
  varinfo <- pagoda.varnorm(knn, counts = cd, trim = 3/ncol(cd), max.adj.var = 5, n.cc
dev.off()

  # remove effect of detected genes.
  varinfo <- pagoda.subtract.aspect(varinfo, colSums(cd[, rownames(knn)]>0))
  save(varinfo, file=savefile)
}
```

Control for other aspects of heterogeneity

In this example we only remove the aspect of gene detection, but there can be other aspects that you want to remove, an example from there tutorial to remove cell cycle effect.

OBS! This was not run on the ILC dataset

```
# get cell cycle signature and view the top genes
cc.pattern <- pagoda.show.pathways(c("G0:0000280", "G0:0007067"), varinfo, go.env, show.cc)
# subtract the pattern
varinfo.cc <- pagoda.subtract.aspect(varinfo, cc.pattern)
```

Run pagoda pathway wPCA

Instead of using the app, that sometimes is very slow, you can also create each plot with different commands, here are some example plots.

Also a step that takes hours to run.

```
savefile <- "data/ILC/pagoda_pwpca.Rdata"
if (file.exists(savefile)){
  load(savefile)
}else {
  pwpca <- pagoda.pathway.wPCA(varinfo, go.env, n.components = 2, n.cores = 1)
  save(pwpca, file=savefile)
}
```

Evaluate overdispersion of 'de novo' gene sets

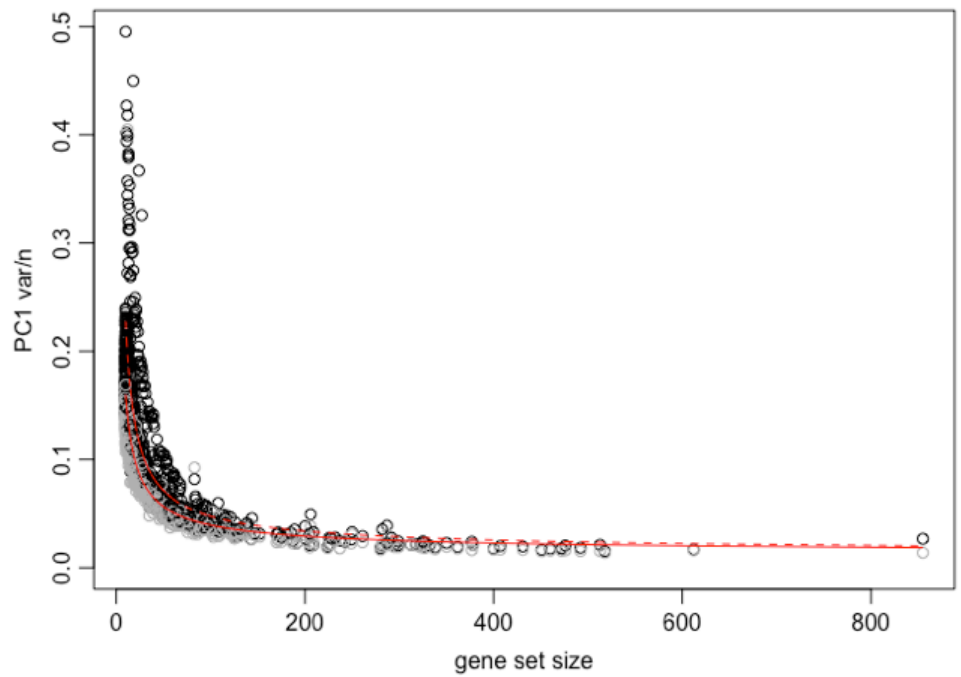
OBS! This takes very long time to run, several hours, so we skip this step in this tutorial and only run the subsequent steps with pwpca.

But it can be run with:

```
clpca <- pagoda.gene.clusters(varinfo, trim = 7.1/ncol(varinfo$mat), n.clusters = 50, n.cc
```

Get top aspects

```
df <- pagoda.top.aspects(pwpca, return.table = TRUE, plot = TRUE, z.score = 1.96)
```



head(df)

```
##                                name npc  n   score
## 1069          GO:0009416;response to light stimulus    1 18 4.241465
## 1797 GO:0032870;cellular response to hormone stimulus    1 24 4.177536
## 2823                                GO:0051591;response to cAMP    1 27 3.997107
## 2889          GO:0060037;pharyngeal system development    1 10 3.097088
## 3141          GO:0071850;mitotic cell cycle arrest    1 12 2.978951
## 1989          GO:0035994;response to muscle stretch    1 13 2.886317
##      z      adj.z sh.z adj.sh.z
## 1069 24.45432 24.16538   NA      NA
## 1797 25.72343 25.40572   NA      NA
## 2823 25.27727 24.98162   NA      NA
## 2889 15.26336 14.84385   NA      NA
## 3141 15.18589 14.77459   NA      NA
## 1989 14.88660 14.47606   NA      NA
```

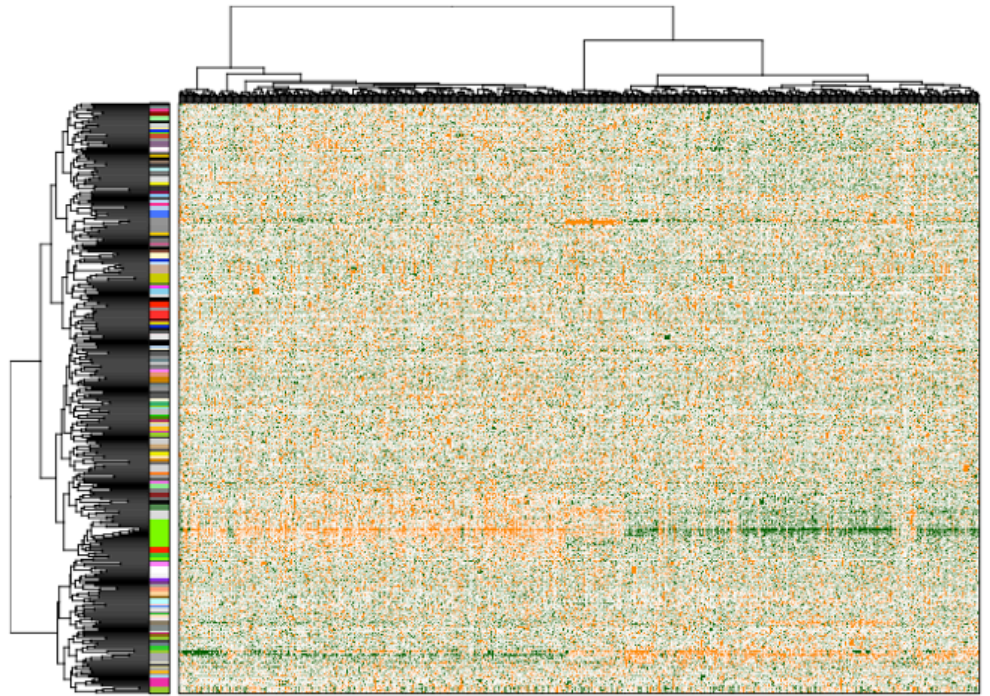
Visualize significant aspects of heterogeneity

```
# get full info on the top aspects
tam <- pagoda.top.aspects(pwpca, n.cells = NULL, z.score = qnorm(0.01/2, lower.tail = FALSE)

# determine overall cell clustering
hc <- pagoda.cluster.cells(tam, varinfo)

# Next, we will reduce redundant aspects in two steps. First we will combine pathways that
tamr <- pagoda.reduce.loading.redundancy(tam, pwpca)

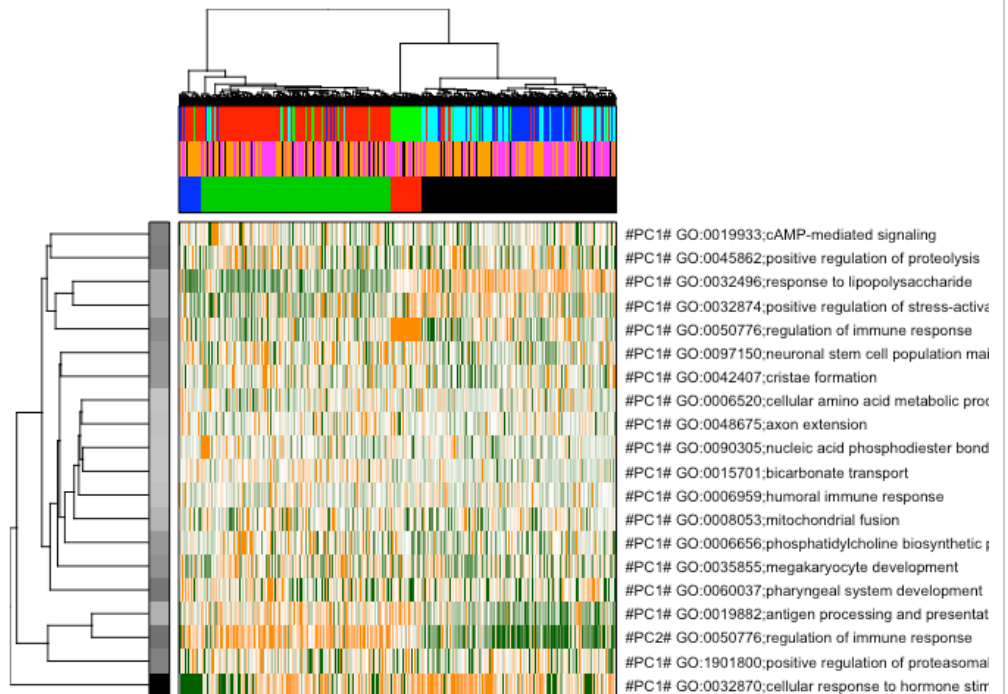
# In the second step we will combine aspects that show similar patterns (i.e. separate the
tamr2 <- pagoda.reduce.redundancy(tamr, distance.threshold = 0.9, plot = TRUE, cell.cluste
```



View top aspects

```
# define colors for clusters, here we split by 4 clusters
col.cols <- rbind(groups = cutree(hc, 4))

# plot top aspects, include also donor/celltype color vectors
pagoda.view.aspects(tamr2, cell.clustering = hc, box = TRUE, labCol = NA, margins = c(0.5,
```



```
# here top 20 aspects are plotted,
```

Launch the pagoda app

The results can be browsed interactively with the pagoda app using commands:

```
app <- make.pagoda.app(tamr2, tam, varinfo, go.env, pwpca, col.cols = rbind(col.cols,donor
# show app in the browser (port 1468)
show.app(app, "ILCs", browse = TRUE, port = 1468)
```

Run tSNE based on pagoda distances.

Instead of basing tSNE on regular PCA distances, you can also use the distances from the pagoda pwpca object as an input to Rtsne.

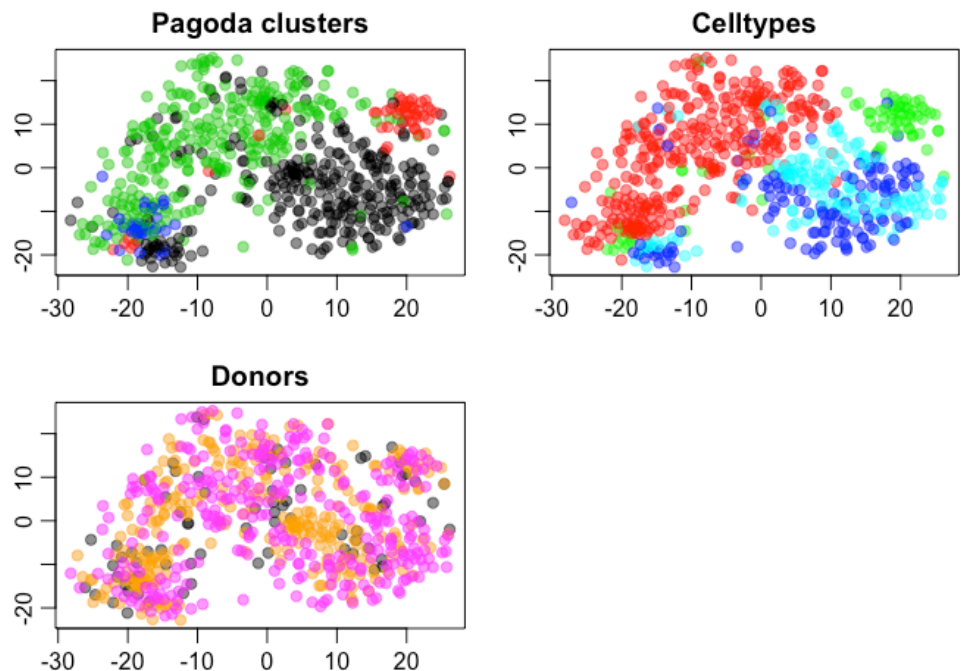
```
suppressMessages(library(Rtsne))
```

```
# recalculate clustering distance, we'll need to specify return.details=T
cell.clustering <- pagoda.cluster.cells(tam,varinfo,include.aspects=TRUE,verbose=TRUE,retu
```

```
## clustering cells based on 1005 genes and 587 aspect patterns
```

```
# fix the seed to ensure reproducible results
```

```
set.seed(0)
tSNE.pagoda <- Rtsne(cell.clustering$distance,is_distance=T,initial_dims=30,perplexity=30,
par(mfrow=c(2,2), mar = c(2.5,2.5,2.0,0.5), mgp = c(2,0.65,0), cex = 1.0);
plot(tSNE.pagoda$Y,col=adjustcolor(col.cols,alpha=0.5),cex=1,pch=19,xlab="",ylab="",main="
plot(tSNE.pagoda$Y,col=adjustcolor(celltype2cols,alpha=0.5),cex=1,pch=19,xlab="",ylab="",n
plot(tSNE.pagoda$Y,col=adjustcolor(donor2cols,alpha=0.5),cex=1,pch=19,xlab="",ylab="",main
```



Compare to tSNE based on counts

Run tsne based on same dataset (cd).

```
set.seed(0)
tSNE.counts <- Rtsne(t(log2(cd+1)),initial_dims=30,perplexity=30,theta=0.1)
par(mfrow=c(2,2), mar = c(2.5,2.5,2.0,0.5), mgp = c(2,0.65,0), cex = 1.0);
plot(tSNE.counts$Y,col=adjustcolor(col.cols,alpha=0.5),cex=1,pch=19,xlab="",ylab="",main="
plot(tSNE.counts$Y,col=adjustcolor(celltype2cols,alpha=0.5),cex=1,pch=19,xlab="",ylab="",n
plot(tSNE.counts$Y,col=adjustcolor(donor2cols,alpha=0.5),cex=1,pch=19,xlab="",ylab="",main
```

