Single-Cell RNA-Seq visualiZation

Hadrien LORENZO, PhD student, SISTM Tean

16/06/2017

Contents

Load the correct packages

We have chosen the `Rtsne` package to start t-SNE analyses.

```
library(Rtsne)
```

In case you do not have already downloaded that package please unquote and start the following line before loading the package :

```
# install.packages(Rtsne)
```

Also load different fancy common packages

```
library(ggplot2)
```

Introduction to t-SNE capibilities

Please go to the distill website Distill t-SNE. Please play with the first examples according to what have just been discussed.

The **perplexity** parameter is the most important try to see its effect versus the number of iterations.

A real dataset from 10X

Description of the data

The dataset is extracted from the study (Zheng et al. 2017). We have here summarized that dataset to 10 cell types, with 200 cells per type. The cell types are :

- b_cells
- cd14_monocytes
- cd34
- cd4_t_helper
- cd56_nk
- cytotoxic_t
- memory_t
- naive_cytotoxic
- naive_t
- regulatory_t

Initially the dataset is built with 32738 but we have removed a few genes, which are :

- The **0** expression genes,
- The **0** variance genes.

Then, the resulting matrix has been *log*-transformed and standardized, which means :

- **0** mean genes,
- **1** variance genes.

To finally get a dataset of $n_{genes} = 194$ and $n_{cells} = 2000$.

The resulting matrix can be called $\mathbf{X}$ and the cell-type corresponding vector $\mathbf{y}$.

Open dataset

Please open the datasets

- $\mathbf{X}$: in rows the cells and in colums the standardized gene expressions, in *genes.csv*
- $\mathbf{y}$: the cell-type vector, in *cellType.csv*

```r
X <- as.matrix(read.csv2("genes.csv"))
y <- read.csv2("cellType.csv",header = F)
```

PCA visualization

PCA computation

Please compute the PCA of the dataset.With the function *prcomp*, 2 very important values :

- *sdev* : The standard deviation of each axis, if the are r axes. Which is linked to the eigen values of the PCA such as :
$$\forall i \in [\![1, r]\!] | \lambda_i = s_{dev,i}^2,$$
Which is itself linked to the proportion of variance explained by any axis i with :
$$\forall i \in [\![1, r]\!] | VarExpl_i = \frac{\lambda_i}{\sum_{j=1}^{r} \lambda_j} \times 100.$$
- *rotation* : The weights of the PCA for each variable (in rows) over each axis (in columns). So this is a square matrix. One would use that matrix, we call it W to represent the data in the principal axis basis as follows :
$$X_{PC} = XW$$

```r
pca_X <- prcomp(X)
W <- pca_X$rotation
eigenValues <- (pca_X$sdev)^2
varianceExplained <- round(eigenValues*100/sum(eigenValues),digits = 1)
```

Variances explained

The first common way of exploring PCA results is to look at Var Explained for each axis.

The following figure 1 presents the variance explained by the 20 first axes.

```r
df_pca_eigenValues <- data.frame(list(Component=1:length(varianceExplained),
                                      eigenValues=eigenValues))
plotelbow <- ggplot(df_pca_eigenValues[1:20,], aes(x=Component, y=eigenValues)) +
  geom_point() +
  geom_line() +
  ylab("Elbow-Values")
plotkaiser <- ggplot(df_pca_eigenValues[1:20,], aes(x=Component, y=eigenValues)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = mean(df_pca_eigenValues$eigenValues)) +
  ylab("Eigen-Values")

do.call(gridExtra::grid.arrange,  list(plotelbow,plotkaiser,ncol=2) )
```

The horizontal line of the second graph represents the mean of the eigen-values.

Elbow and Kaiser criterions are commonly used to fix the number of components in PCA analyses. Are they reasonnable here to gete good visualizations ? How many components would you keep ?

PCA variates

PCA has been applied and this is now possible to look at the data projected in the subspace defined by the weight matrix W. We have decided to represent the variates of the two first axes on figure 2.

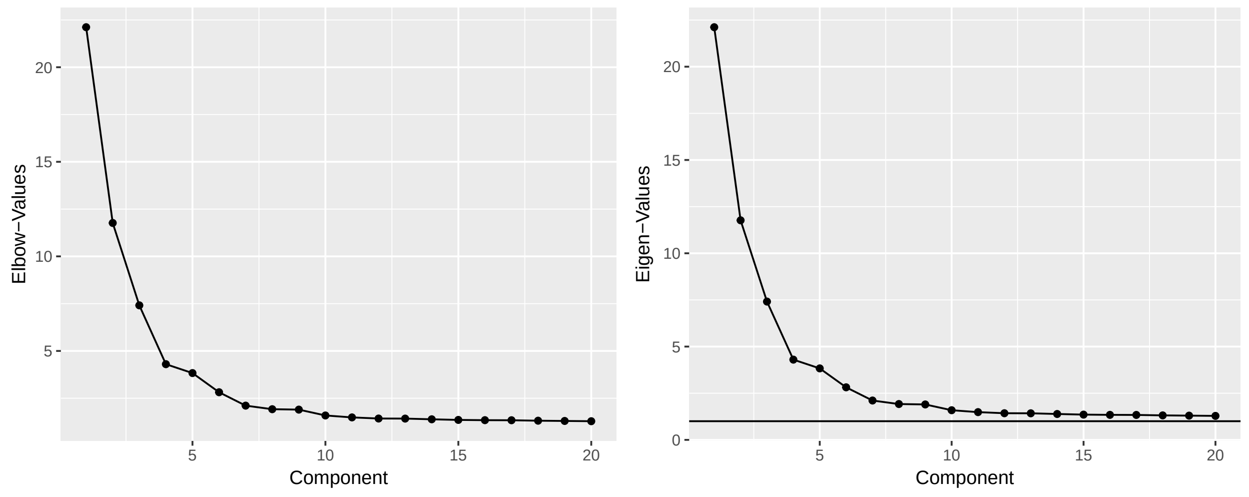

Figure 1: Nine first PCA component variance explained

```
df_pca <- data.frame(list(PC1=X%*%W[,1],
                          PC2=X%*%W[,2],
                          Cell_Type=y$V1))
ggplot()+
  geom_point(data = df_pca, alpha=0.6,cex=1.1,
             aes(x=PC1, y=PC2, group=Cell_Type, col=Cell_Type))+
  xlab(paste("PC 1 (",varianceExplained[1],"% var. expl.)"))+
  ylab(paste("PC 2 (",varianceExplained[2],"% var. expl.)"))+
  scale_colour_brewer(palette="Paired")
```

Are those two first components enough to describe our dataset and its information ? Can you find a good/efficient representation with those axes ?

t-SNE is very efficient to capture the non linear structures. So we will try its performance on that dataset.

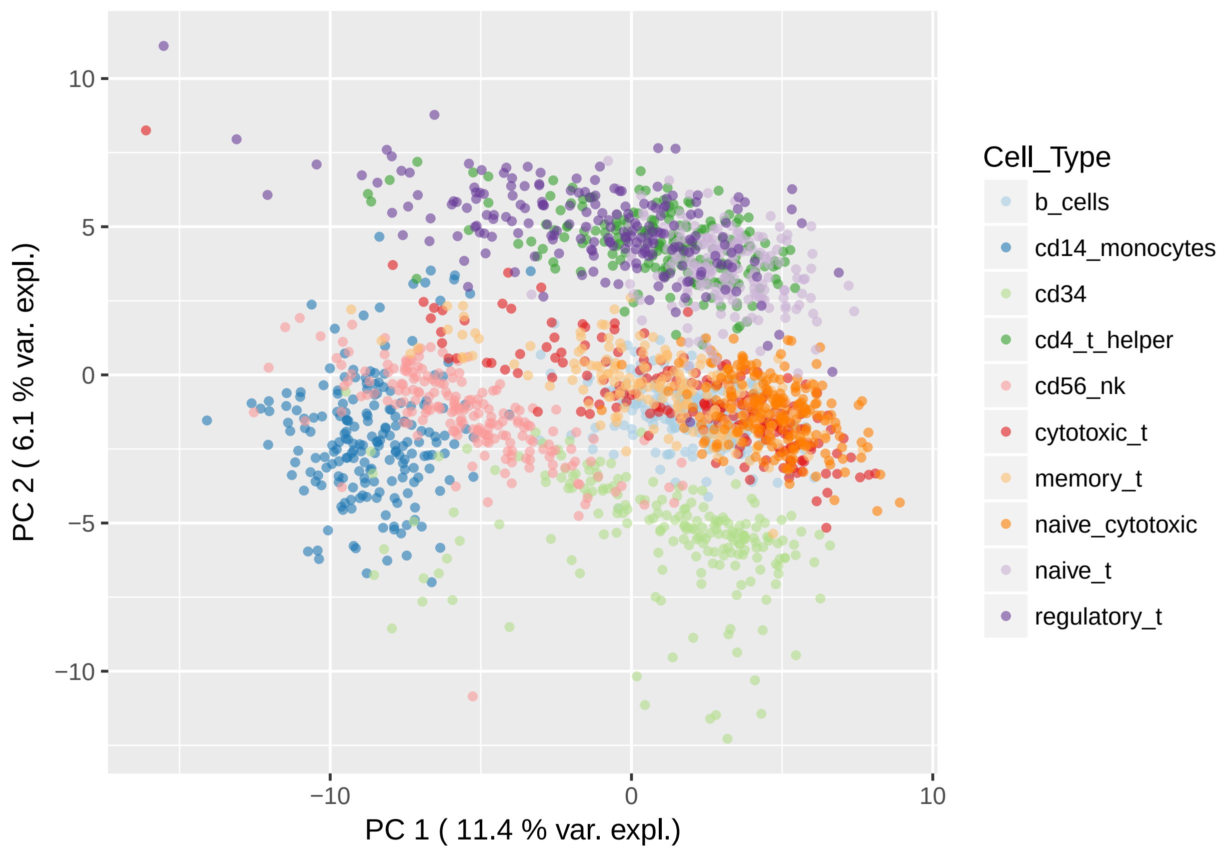

Figure 2: Two first PCA variates

t-SNE visualization

t-SNE is a method which aims to describe in a 2(or 3)-dimensionnal space the similarities of the individues, here cells, of a dataset. This is based on non linear transformations. Whatsmore, this is not supervized, which means that the label of each cell is not known by the algorithm. That method has been introduced by Hinton G. (G. E. Hinton and Roweis 2003) and modified by Van Der Maaten L. (Maaten and Hinton 2008) and then accelerated with tree structures (Van Der Maaten 2014).

The main idea is to find a small dimensionnal space where the data are as close as possible, through the Kullback-Leibler similarity, to the data in the original space. Some advances have been introduced as to tackle the skewed distribution and the crowding problems.

Start 2d t-SNE

t-SNE gives very appealing figures but is often treacky to tune as there are some parameters, sometimes hard to interpret. Here we will use the **Rtsne** function from the **R package** `Rtsne`. Some of the most important parameters are here listed :

- *dims* : The visualization dimension. Most of the times equal to 2 as more accurate to conclude over the efficiency of our parametrization. Default : 2.
- *Perp* : The perplexity. Directly linked to the density of the group of cells desired, through the Shanon entropy as a power of 2. Default : 30.
- *pca* : Does the algorithm has to start a PCA before start t-SNE. We will put it to **FALSE** in our case since we have already produced PCA and we do not want to waste to much time. Default : **TRUE**.

As adviced by the authors, it is convenient to reduce the dimensions thanks to a PCA prior to star t-SNE analysis. We will do so. We have decided to keep

$$k = 30$$

components, which are the k first components of the PCA. So we can now call the `Rtsne` function with the default parameters (appart from `pca`).

Change the *perplexity* and see how the result reacts. Is the result sensible to that parameter ?

```r
k <- 30
set.seed(1)
tsne_sc <- Rtsne(X = X%*%W[,1:k],
                 dims=2,
                 perplexity = 30,
                 pca = F)
```

Visualize t-SNE

The one way to look at t-SNE results is to plot its variates, what we do in figure3.

```r
df_tsne <- data.frame(list(tSNE1=tsne_sc$Y[,1],
                           tSNE2=tsne_sc$Y[,2],
                           Cell_Type=y$V1))
ggplot()+
  geom_point(data = df_tsne, alpha=0.65,cex=1,
             aes(x=tSNE1, y=tSNE2, group=Cell_Type, col=Cell_Type))+
  xlab("t-SNE Comp 1")+
  ylab("t-SNE Comp 2")+
  scale_colour_brewer(palette="Paired")
```

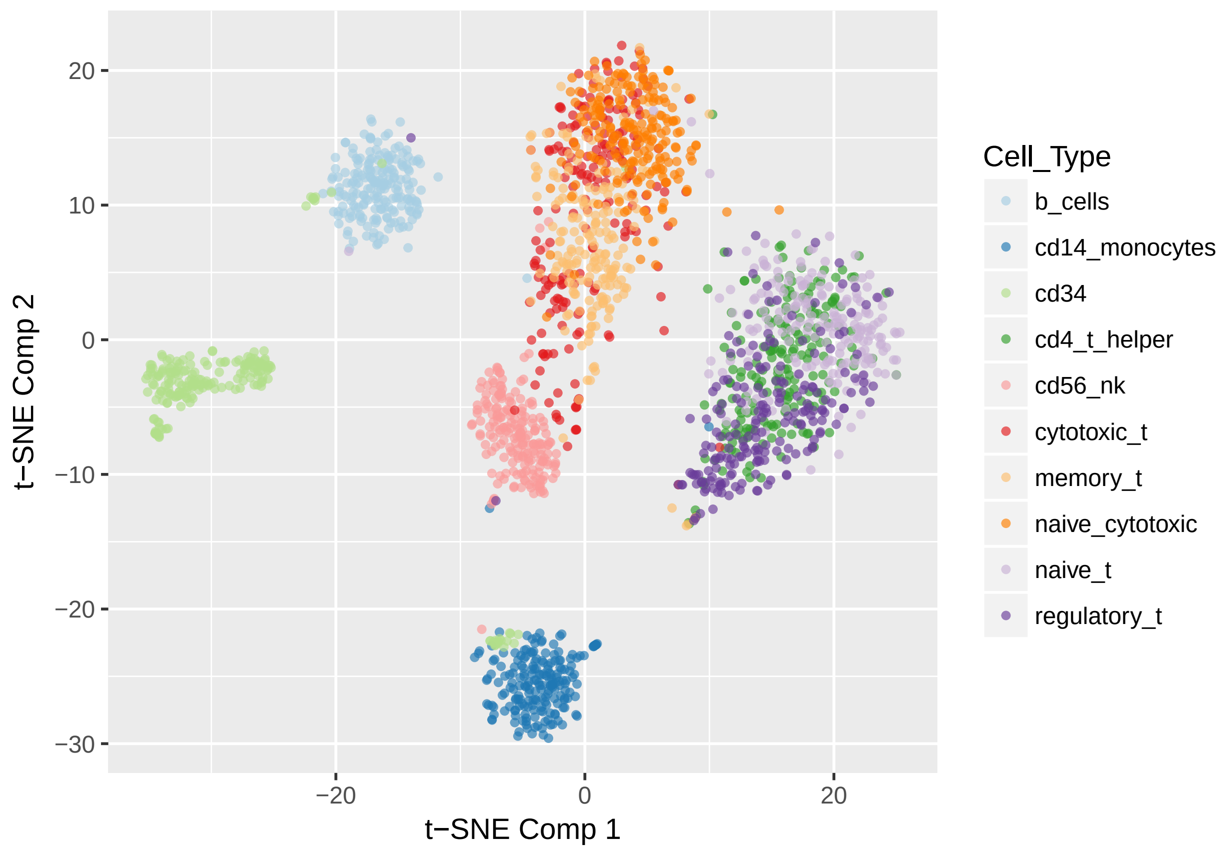

Figure 3: 2d t-SNE representation

How many clusters can you identify ? Is that enough ? Coherent ?

References

Hinton, Geoffrey E, and Sam T Roweis. 2003. "Stochastic Neighbor Embedding." In *Advances in Neural Information Processing Systems*, 857–64.

Maaten, Laurens van der, and Geoffrey Hinton. 2008. "Visualizing Data Using T-Sne." *Journal of Machine Learning Research* 9 (Nov): 2579–2605.

Van Der Maaten, Laurens. 2014. "Accelerating T-Sne Using Tree-Based Algorithms." *Journal of Machine Learning Research* 15 (1): 3221–45.

Zheng, Grace XY, Jessica M Terry, Phillip Belgrader, Paul Ryvkin, Zachary W Bent, Ryan Wilson, Solongo B Ziraldo, et al. 2017. "Massively Parallel Digital Transcriptional Profiling of Single Cells." *Nature Communications* 8. Nature Publishing Group: 14049.