Analysis of blood cell types from human and mouse using OSBF

Amal Thomas

Contents

1 Data

In this workflow, we will analyze gene expression profiles of 10 blood cell types from human and mouse. The count tables of human and mouse blood cell types were generated by processing the raw FASTQ files published in the two studies: (Corces et al. 2016) and (Choi et al. 2019). The genome assembly (FASTA file), GTF files, and orthology annotation were obtained from Ensembl v94. Reads were mapped to the corresponding genome using STAR (v2.7.1a) after trimming adapters using Cutadapt (v1.16). The library preparation protocol strand type of each library was inferred using infer_experiment.py module in RSeQC (v4.0). Genes annotated in the GTF files were quantified using HTSeq-count.

Let us first load the `SBF` library.

```
# load SBF package
library(SBF)
```

Additional packages required for the vignette

```
# install packages
pkgs <- c("data.table", "dplyr", "matrixStats")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install))
  install.packages(require_install)
suppressPackageStartupMessages({
  library(data.table)
  library(dplyr)
  library(matrixStats)
})
```

2 Species and cell types

The list of species and cell types we will be working with:

```
species <- c("Homo_sapiens", "Mus_musculus")
species_short <- sapply(species, getSpeciesShortName)
species_short
```

```
## Homo_sapiens Mus_musculus
##   "hsapiens"  "mmusculus"
```

```
common_celltypes <- c("hsc", "clp", "cmp", "nkcell", "cd8tcell", "cd4tcell",
                      "bcell", "cd14monocytes", "eosinophil", "neutrophil")
```

3 Read gene expression profiles

Download the processed RNA-Seq counts of human and mouse ("human_mouse_blood_counts.tar.gz") from

Uncompress the .tar.gz file and add it to the working directory.

```
# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
counts_list <- metadata_list <- list()
for (sp in species) {
  # read blood logTPM counts for each species
  counts <- read.table(paste0(path, "human_mouse_blood_counts/", sp,
                              "_blood_logtpm.tsv"), header = T, sep = "\t",
                       row.names = 1)
  info <- tstrsplit(colnames(counts), "_")
  metadata <- data.frame(project = info[[1]],
        species = info[[2]],
        tissue = info[[3]],
        gsm = info[[4]],
        name = colnames(counts),
        stringsAsFactors = F)
  metadata$ref <- seq_len(nrow(metadata))
  metadata$key <- paste0(metadata$species, "_", metadata$tissue)
  metadata$tissue_factor <- factor(metadata$tissue)
  counts_list[[sp]] <- counts
  metadata_list[[sp]] <- metadata
}
sapply(counts_list, dim)
```

```
##      Homo_sapiens Mus_musculus
```

```
## [1,]        58676        54446
## [2,]           44           25
```

3.1 Compute mean expression profiles

Now, for each species, let us compute the average expression profile for each cell types.

```
avg_counts <- list()
for (sp in species) {
  avg_counts[[sp]] <- calcAvgCounts(counts_list[[sp]], metadata_list[[sp]])
}
```

```
# check tissue columns are matching in each species
c_tissues <- as.data.frame(sapply(avg_counts, function(x) tstrsplit(colnames(x),"_")[[2]]))
if (!all(apply(c_tissues, 1, function(x) all(x == x[1])))) {
        stop("Error! columns not matching")
}
```

The dimension of mean expression profiles

```
sapply(avg_counts, dim)
```

```
##      Homo_sapiens Mus_musculus
## [1,]        58676        54446
## [2,]           10           10
```

Remove genes not expressed.

```
# remove empty rows
removeZeros <- function(df) {
    return(df[rowSums(df) > 0, ])
}
avg_counts <- lapply(avg_counts, removeZeros)
sapply(avg_counts, dim)
```

```
##      Homo_sapiens Mus_musculus
## [1,]        30330        20851
## [2,]           10           10
```

```
# update counts_list
counts_list_sub <- list()
for (sp in names(avg_counts)) {
  counts_list_sub[[sp]] <- counts_list[[sp]][row.names(avg_counts[[sp]]), ,
                                             drop = F]
}
```

4 OSBF factorization

We will perform OSBF in two ways.

1. Keeping the initial estimate of V the same while updating U_i and Δ_i to minimize the factorization error. By keeping the V same, the initial V estimated based on inter-sample correlation is maintained.
2. Update V, U_i and Δ_i to minimize the factorization error.

```
# first lets compute OSBF without updating the initial estimate of V. U and Delta
# are updated in this case
cat(format(Sys.time(), "%a %b %d %X %Y"),"\n")
```

```
## Fri Jun 17 12:25:47 PM 2022
```

```
osbf_noVupdate <- SBF(avg_counts, orthogonal = TRUE, transform_matrix = TRUE, minimizeError = TRUE,
                      optimizeV = FALSE, tol = 1e-4)
```

```
##
## OSBF optimizing factorization error
```

```
cat(format(Sys.time(), "%a %b %d %X %Y"),"\n")
```

```
## Fri Jun 17 12:25:48 PM 2022
```

```
# Now lets compute OSBF updating all three factors (U, Delta, V)
cat("============================================================\n")
```

```
## ============================================================
```

```
cat("optimizing V = TRUE\n")
```

```
## optimizing V = TRUE
```

```
cat(format(Sys.time(), "%a %b %d %X %Y"),"\n")
```

```
## Fri Jun 17 12:25:48 PM 2022
```

```
osbf <- SBF(avg_counts, orthogonal = TRUE, transform_matrix = TRUE, minimizeError = TRUE,
            optimizeV = TRUE, tol = 1e-4)
```

```
##
## OSBF optimizing factorization error
```

```
cat(format(Sys.time(), "%a %b %d %X %Y"),"\n")
```

```
## Fri Jun 17 12:26:01 PM 2022
```

The final factorization error and number of updates taken:

```
cat("\n", sprintf("%-27s:", "Final error [No V update]"), sprintf("%16.2f",
                                                    osbf_noVupdate$error))
```

```
##
##  Final error [No V update]  :        105405.40
```

```
cat("\n", sprintf("%-27s:", "Final error [With V update]"), sprintf("%16.2f",
                                                              osbf$error))
```

```
##
##  Final error [With V update]:         88096.62
```

```
cat("\n", sprintf("%-27s:", "# of update [No V update]"), sprintf("%16d",
                                                  osbf_noVupdate$error_pos))
```

```
##
##  # of update [No V update]  :                7
```

```
cat("\n", sprintf("%-27s:", "# of update [With V update]"), sprintf("%16d",
                                                        osbf$error_pos))
```

```
##
##  # of update [With V update]:              430
```

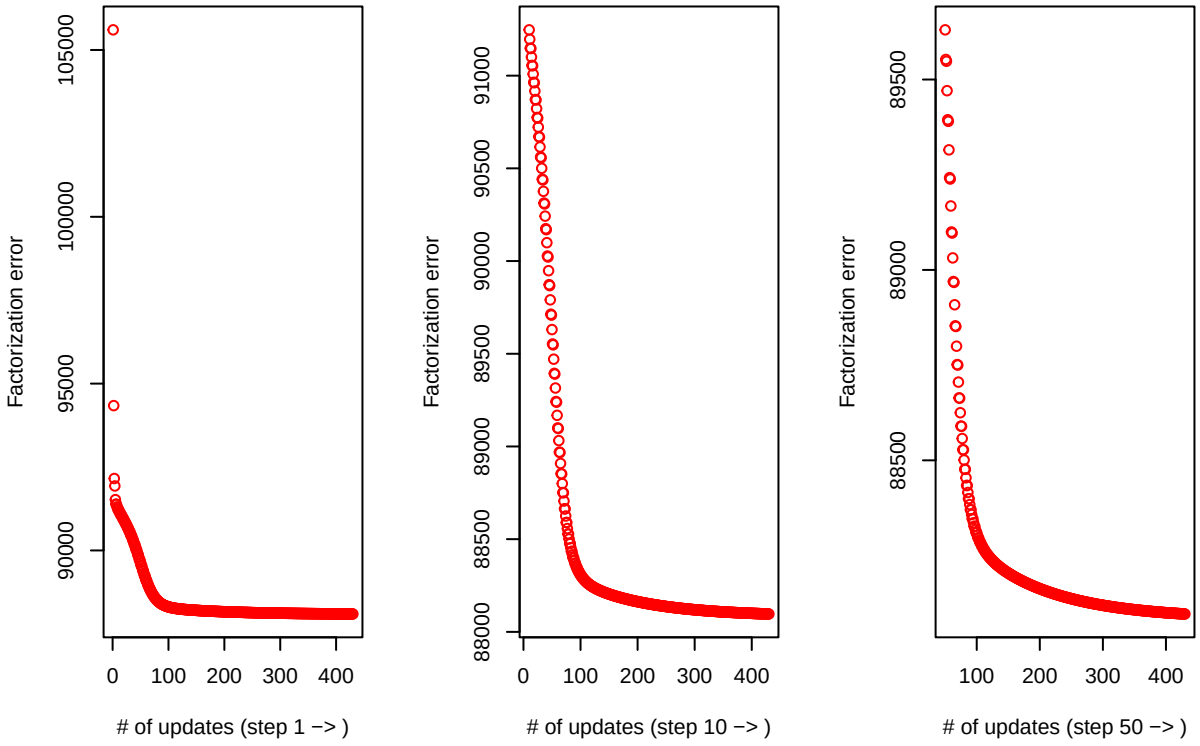
Optimization with updating V achieves a lower decomposition error.

```
osbf_noVupdate$error / osbf$error
```

```
## [1] 1.196475
```

Let us plot the decomposition error vs. updates.

```
par(mfrow = c(1, 3))
plot(x = seq_len(length(osbf$error_vec)), y = osbf$error_vec,
     xlab = "# of updates (step 1 -> )",
     ylab = "Factorization error", col = "red")
plot(x = 10:length(osbf$error_vec),
     y = osbf$error_vec[10:length(osbf$error_vec)],
     xlab = "# of updates (step 10 -> )",
     ylab = "Factorization error", col = "red")
plot(x = 50:length(osbf$error_vec),
     y = osbf$error_vec[50:length(osbf$error_vec)],
     xlab = "# of updates (step 50 -> )",
     ylab = "Factorization error", col = "red")
```



orthogonality of the estimated V

```
zapsmall(osbf_noVupdate$v %*% t(osbf_noVupdate$v))
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    0    0    0    0    0    0    0    0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0
##  [5,]    0    0    0    0    1    0    0    0    0     0
##  [6,]    0    0    0    0    0    1    0    0    0     0
##  [7,]    0    0    0    0    0    0    1    0    0     0
##  [8,]    0    0    0    0    0    0    0    1    0     0
##  [9,]    0    0    0    0    0    0    0    0    1     0
## [10,]    0    0    0    0    0    0    0    0    0     1
```

```
zapsmall(osbf$v %*% t(osbf$v))
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    0    0    0    0    0    0    0    0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0
##  [5,]    0    0    0    0    1    0    0    0    0     0
##  [6,]    0    0    0    0    0    1    0    0    0     0
##  [7,]    0    0    0    0    0    0    1    0    0     0
##  [8,]    0    0    0    0    0    0    0    1    0     0
##  [9,]    0    0    0    0    0    0    0    0    1     0
## [10,]    0    0    0    0    0    0    0    0    0     1
```

Percentage of information (p_{ij}) represented by a common space dimension is defined as $p_{ij} = \delta_{ij}^2 / \sum_{j=1}^{6} \delta_{ij}^2 \times 100$, where $\Delta_i = \text{diag}(\delta_{i1}, \ldots, \delta_{i6})$

```
percentInfo_noVupdate <- lapply(osbf_noVupdate$d, function(x) {
  round(x^2 / sum(x^2) * 100, 2)})
cat("\nPercentage for each delta [No V update]:")
```

```
##
## Percentage for each delta [No V update]:
```

```
for (i in names(osbf_noVupdate$d)) {
  cat("\n", sprintf("%-25s:", i), sprintf("%8.2f", percentInfo_noVupdate[[i]]))
}
```

```
##
##  Homo_sapiens             :    85.93     5.31     3.99     1.18     1.05     0.98     0.62     0.59
##  Mus_musculus             :    91.54     2.35     2.60     1.08     0.69     0.59     0.53     0.46
```

```
percentInfo <- lapply(osbf$d, function(x) {
  round(x^2 / sum(x^2) * 100, 2) })
cat("\nPercentage for each delta:")
```

```
##
## Percentage for each delta:
```

```
for (i in names(osbf$d)) {
  cat("\n", sprintf("%-25s:", i), sprintf("%8.2f", percentInfo[[i]]))
}
```

```
##
##  Homo_sapiens             :    86.37     5.09     3.78     1.16     1.08     1.12     0.51     0.56
##  Mus_musculus             :    91.34     2.48     2.61     1.09     0.70     0.44     0.85     0.32
```

The percentage of information represented by different dimensions of the two approaches looks very similar.

5 Project datasets to the common space

Let us first compute Δ^{-1}

```
d_inv_NoVupdate <- list()
for (sp in names(avg_counts)) {
  if (length(osbf_noVupdate$d[[sp]]) == 1) {
    d_inv_NoVupdate[[sp]] <- as.matrix(diag(as.matrix(1 / osbf_noVupdate$d[[sp]])))
    } else {
```

```
      d_inv_NoVupdate[[sp]] <- as.matrix(diag(1 / osbf_noVupdate$d[[sp]]))
    }
}
d_inv <- list()
for (sp in names(avg_counts)) {
  if (length(osbf$d[[sp]]) == 1) {
    d_inv[[sp]] <- as.matrix(diag(as.matrix(1 / osbf$d[[sp]])))
    } else {
      d_inv[[sp]] <- as.matrix(diag(1 / osbf$d[[sp]]))
    }
}
```

Project individual profiles and average counts to common space by computing $D_i^T U_i \Delta^{-1}$

```
# project using no V update estimates
# we can project both mean expression profiles as well as individual expression
# profiles
avgcounts_projected_noVupdate  <- counts_projected_noVupdate <- list()
n_noVupdate <- n1_noVupdate <- c()
for (sp in names(avg_counts)) {
  avgcounts_projected_noVupdate[[sp]] <- as.matrix(t(avg_counts[[sp]])) %*%
    as.matrix(osbf_noVupdate$u[[sp]]) %*% d_inv_NoVupdate[[sp]]
  n_noVupdate <- c(n_noVupdate, row.names(avgcounts_projected_noVupdate[[sp]]))
  counts_projected_noVupdate[[sp]] <- as.matrix(t(counts_list_sub[[sp]])) %*%
    as.matrix(osbf_noVupdate$u[[sp]]) %*% d_inv_NoVupdate[[sp]]
  n1_noVupdate <- c(n1_noVupdate, row.names(counts_projected_noVupdate[[sp]]))
}

# combine projected profiles
df_proj_avg_noVupdate <- as.data.frame(do.call(rbind, avgcounts_projected_noVupdate))
rownames(df_proj_avg_noVupdate) <- n_noVupdate
colnames(df_proj_avg_noVupdate) <- paste0("Dim", 1:ncol(df_proj_avg_noVupdate))
meta <- tstrsplit(row.names(df_proj_avg_noVupdate), "_")
df_proj_avg_noVupdate$tissue <- factor(meta[[2]])
df_proj_avg_noVupdate$species <- factor(meta[[1]])
df_proj_avg_noVupdate <- df_proj_avg_noVupdate %>% mutate(species = factor(species,
                           levels = species_short))


df_proj_noVupdate <- as.data.frame(do.call(rbind, counts_projected_noVupdate))
rownames(df_proj_noVupdate) <- n1_noVupdate
colnames(df_proj_noVupdate) <- paste0("Dim", 1:ncol(df_proj_noVupdate))
meta1 <- data.table::tstrsplit(row.names(df_proj_noVupdate), "_")
df_proj_noVupdate$tissue <- factor(meta1[[3]])
df_proj_noVupdate$species <- factor(meta1[[2]])
df_proj_noVupdate <- df_proj_noVupdate %>% mutate(species = factor(species,
                           levels = species_short))

# project using V update estimates
avgcounts_projected  <- counts_projected <- list()
n <- n1 <- c()
for (sp in names(avg_counts)) {
  avgcounts_projected[[sp]] <- as.matrix(t(avg_counts[[sp]])) %*%
    as.matrix(osbf$u[[sp]]) %*% d_inv[[sp]]
```

```
  n <- c(n, row.names(avgcounts_projected[[sp]]))
  counts_projected[[sp]] <- as.matrix(t(counts_list_sub[[sp]])) %*%
    as.matrix(osbf$u[[sp]]) %*% d_inv[[sp]]
  n1 <- c(n1, row.names(counts_projected[[sp]]))
}

# combine projected profiles
df_proj_avg <- as.data.frame(do.call(rbind, avgcounts_projected))
rownames(df_proj_avg) <- n
colnames(df_proj_avg) <- paste0("Dim", 1:ncol(df_proj_avg))
meta <- data.table::tstrsplit(row.names(df_proj_avg), "_")
df_proj_avg$tissue <- factor(meta[[2]])
df_proj_avg$species <- factor(meta[[1]])
df_proj_avg <- df_proj_avg %>% mutate(species = factor(species,
                            levels = species_short))


df_proj <- as.data.frame(do.call(rbind, counts_projected))
rownames(df_proj) <- n1
colnames(df_proj) <- paste0("Dim", 1:ncol(df_proj))
meta1 <- tstrsplit(row.names(df_proj), "_")
df_proj$tissue <- factor(meta1[[3]])
df_proj$species <- factor(meta1[[2]])
df_proj <- df_proj %>% mutate(species = factor(species,
                            levels = species_short))
```

6 Two-dimensional projection plots in the common space

Next, we will explore the 2D projection plots in the common space. We will first define a custom theme that we will use for the plots.

```
# install packages
pkgs <- c("grid", "ggthemes", "ggplot2")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install))
  install.packages(require_install)
suppressPackageStartupMessages({
  library(grid)
  library(ggthemes)
  library(ggplot2)
})
```

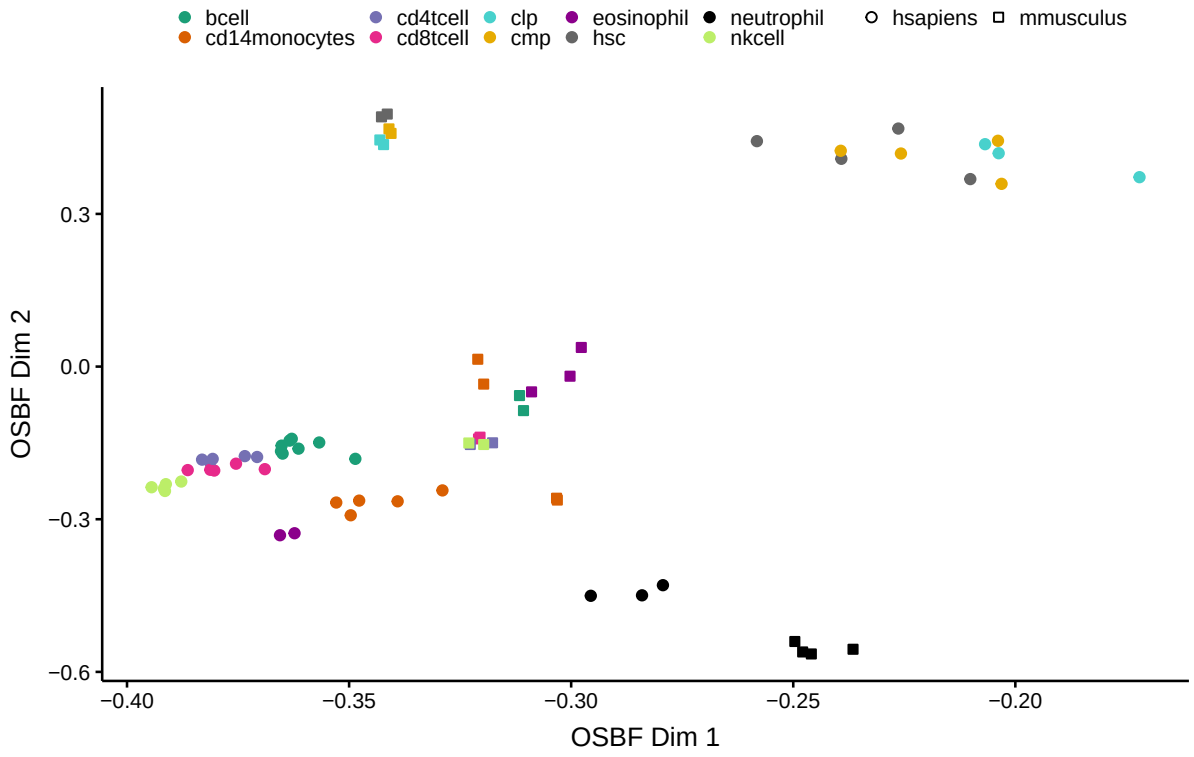
We will use the following custom theme for the ggplots.

```
# custom theme function for ggplot2
customTheme <- function(base_size = 10, base_family = "helvetica") {
   require(grid)
   require(ggthemes)
   (ggthemes::theme_foundation(base_size = base_size)
   + ggplot2::theme(plot.title = element_text(face = "bold",
                                  size = rel(1.2), hjust = 0.5),
         text = element_text(),
         panel.background = element_rect(colour = NA),
         plot.background = element_rect(colour = NA),
```

```
          panel.border = element_rect(colour = NA),
          axis.title = element_text(size = rel(1)),
          axis.title.y = element_text(angle = 90, vjust = 2),
          axis.title.x = element_text(vjust = -0.2),
          axis.text = element_text(),
          axis.line = element_line(colour = "black"),
          axis.ticks = element_line(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.key = element_rect(colour = NA),
          legend.position = "top",
          legend.direction = "horizontal",
          legend.key.size = unit(0.2, "cm"),
          legend.spacing = unit(0, "cm"),
          legend.title = element_text(face = "italic"),
          plot.margin = unit(c(10, 5, 5, 5), "mm"),
          strip.background = element_rect(colour = "#f0f0f0", fill = "#f0f0f0"),
          strip.text = element_text(face = "bold")))
}
```

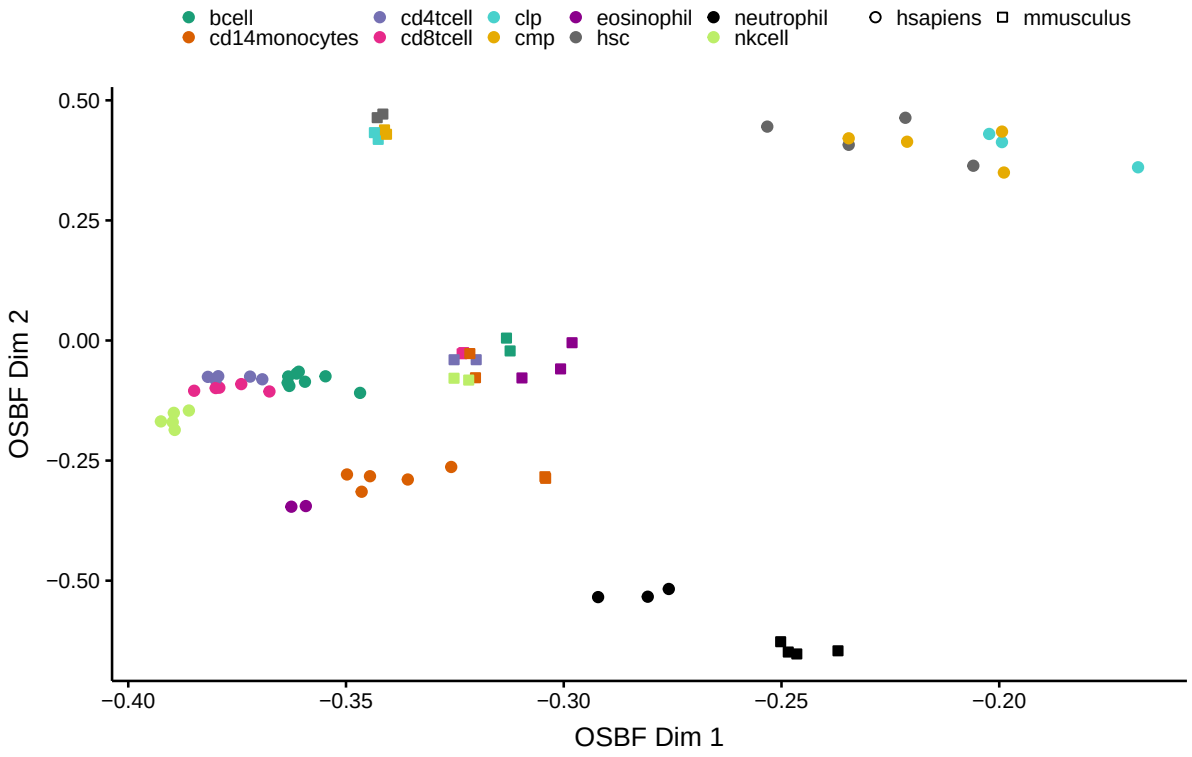
Let us first check the projected libraries in dimension 1 and 2.

```
sel_colors <- c("#1B9E77" ,"#D95F02" ,"#7570B3" ,"#E7298A", "mediumturquoise" ,
                "#E6AB02","darkmagenta", "#666666", "black","darkolivegreen2")
i <- 1
j <- 2
ggplot2::ggplot(df_proj_noVupdate, aes(x = df_proj_noVupdate[, i],
                                y = df_proj_noVupdate[, j], col = tissue,
                                shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```



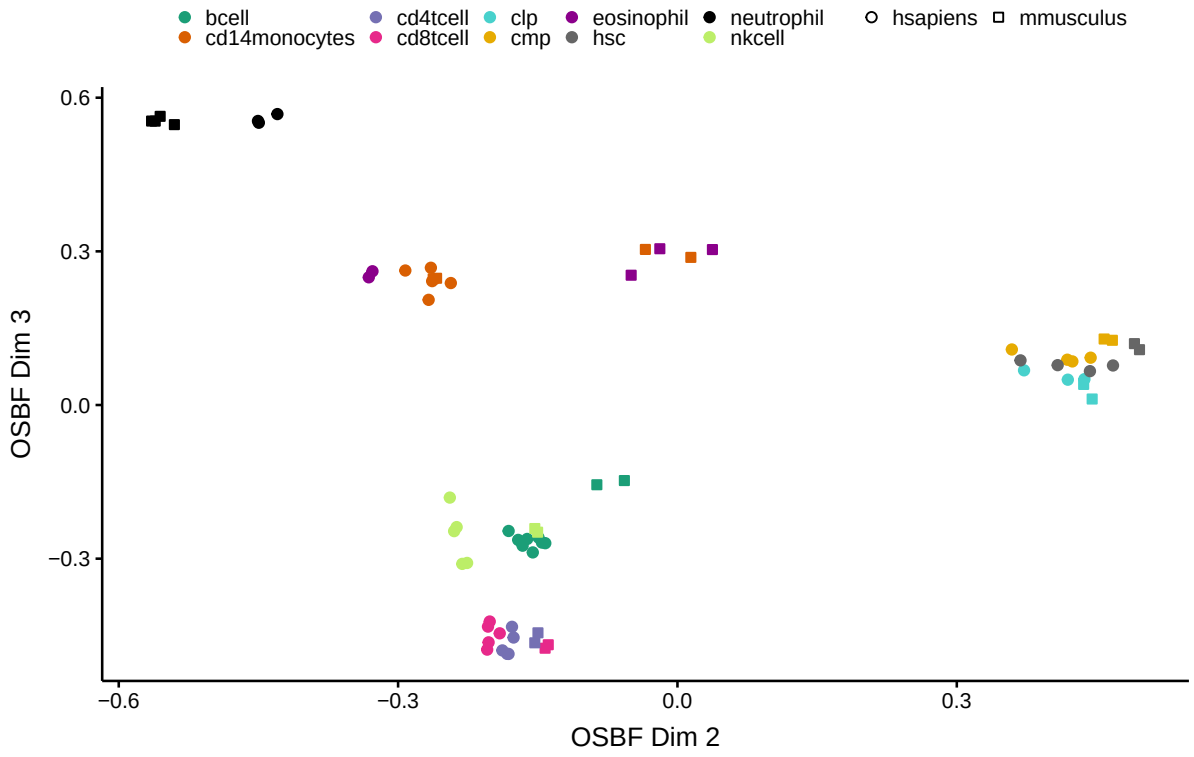
The same with optimized V

```
# 2D plot for Dim1 and Dim2 [With V update]
sel_colors <- c("#1B9E77" ,"#D95F02" ,"#7570B3" ,"#E7298A", "mediumturquoise",
                "#E6AB02","darkmagenta", "#666666", "black","darkolivegreen2")
i <- 1
j <- 2
ggplot2::ggplot(df_proj, aes(x = df_proj[, i],
                             y = df_proj[, j], col = tissue,
                             shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```

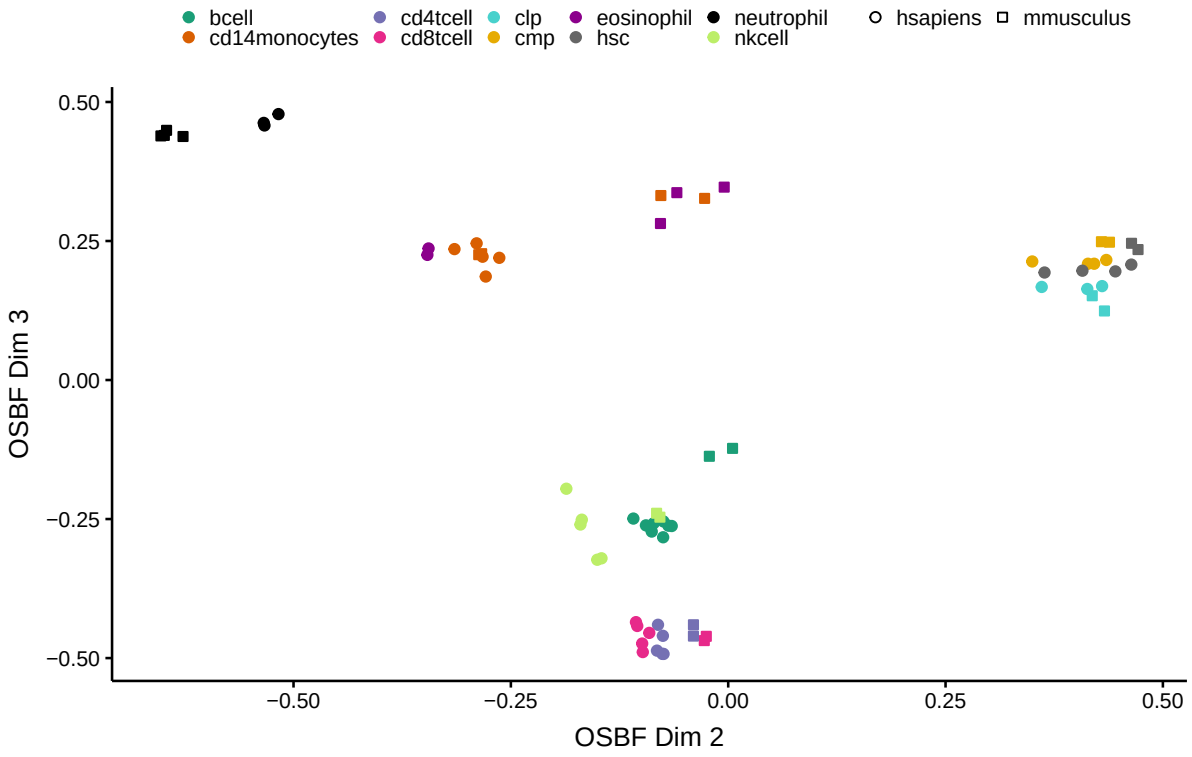


Let us plot the projected libraries in dimensions 2 and 3.

```
# 2D plot for Dim2 and Dim3 [No V update]
sel_colors <- c("#1B9E77" ,"#D95F02" ,"#7570B3" ,"#E7298A", "mediumturquoise",
                "#E6AB02","darkmagenta", "#666666", "black","darkolivegreen2")
i <- 2
j <- 3
ggplot2::ggplot(df_proj_noVupdate, aes(x = df_proj_noVupdate[, i],
                                y = df_proj_noVupdate[, j], col = tissue,
                                shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```




```
# 2D plot for Dim2 and Dim3 [With V update]
sel_colors <- c("#1B9E77" ,"#D95F02" ,"#7570B3" ,"#E7298A", "mediumturquoise",
                "#E6AB02","darkmagenta", "#666666", "black","darkolivegreen2")
i <- 2
j <- 3
ggplot2::ggplot(df_proj, aes(x = df_proj[, i], y = df_proj[, j], col = tissue,
                             shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```



Similarly, we can check for other dimensions of the common space. We observe that the optimized V 2D plots are very similar to the non-optimized V. Since they are similar, we will use the common space with optimized V for all our future analysis.

To save all 2D plots

```
# create output directories. Change this path accordingly
# outdir <- "~/Dropbox/0.Analysis/9.bloodAnalysis/"
# dir.create(file.path(outdir), showWarnings = FALSE)
# subDir <- "2Dplots"
# dir.create(file.path(outdir, subDir), showWarnings = FALSE)

sel_colors <- c("#1B9E77", "#D95F02", "#7570B3", "#E7298A", "mediumturquoise",
                "#E6AB02", "darkmagenta", "#666666", "black", "darkolivegreen2")
outputname <- "human_mouse_blood"
finished <- c()
for (i in 1:(ncol(df_proj)-2)) {
  for (j in 1:(ncol(df_proj)-2)) {
    if (i == j) next
    if (j %in% finished) next
    ggplot(df_proj, aes(x = df_proj[, i], y = df_proj[, j],
                        col = tissue, shape = species, fill = tissue)) +
      xlab(paste0("OSBF Dim ", i)) +
      ylab(paste0("OSBF Dim ", j)) +
      geom_point(size = 1.5) +
      scale_color_manual(values = sel_colors) +
      scale_shape_manual(values = c(21:25,3:7)) +
      scale_fill_manual(values = sel_colors) +
      customTheme(base_size = 12) +
      theme(legend.title = element_blank())
    #ggsave(filename = paste0(outdir, "2Dplots/opt_2Dplot_Dim_", i, "-", j, "_",
```

```
    #                           outputname, ".pdf"), device = "pdf",
    #       width = 7, height = 7, useDingbats = FALSE)
  }
  finished <- c(finished,i)
}
```

7 Clustering in the common space

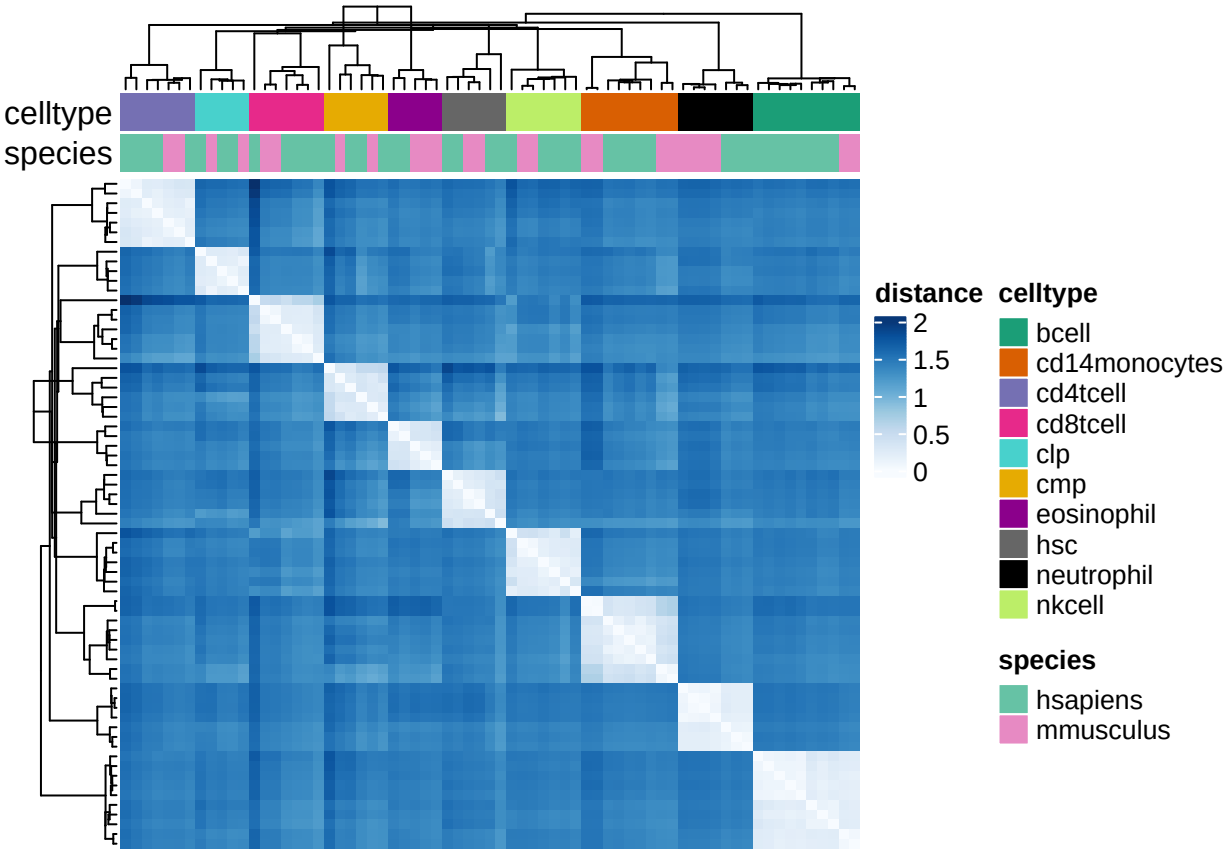
```
# install packages
pkgs <- c("RColorBrewer")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install))
  install.packages(require_install)
pkgs <- c("ComplexHeatmap")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install)) {
  if (!require("BiocManager", quietly = TRUE))
      install.packages("BiocManager")
  BiocManager::install("ComplexHeatmap")
}
suppressPackageStartupMessages({
  library(ComplexHeatmap)
  library(RColorBrewer)
})
```

Compute distances between projected profiles in the common space and perform clustering.

```
data <- df_proj
data$tissue <- NULL
data$species <- NULL
data <- as.matrix(data)
data_dist <- as.matrix(dist(data, method = "euclidean"))
meta <- data.table::tstrsplit(colnames(data_dist), "_")
ht <- ComplexHeatmap::HeatmapAnnotation(celltype = meta[[3]], species = meta[[2]],
                      col = list(species = c("hsapiens" = "#66C2A5",
                                             "mmusculus" = "#E78AC3"),
                                 celltype = c("bcell" = "#1B9E77",
                                              "cd14monocytes" = "#D95F02",
                                              "cd4tcell" ="#7570B3",
                                              "cd8tcell" = "#E7298A",
                                              "clp" = "mediumturquoise",
                                              "cmp" = "#E6AB02",
                                              "eosinophil" = "darkmagenta",
                                              "hsc" = "#666666",
                                              "neutrophil" = "black",
                                              "nkcell" = "darkolivegreen2")),
                                 annotation_name_side = "left")
mypalette <- RColorBrewer::brewer.pal(9, "Blues")
morecolors <- colorRampPalette(mypalette)

myheatmap <- ComplexHeatmap::Heatmap(as.matrix(data_dist), cluster_rows = T,
                     clustering_method_rows = "centroid",
                     cluster_columns = T,
```

```
                    clustering_method_columns = "centroid",
                    top_annotation = ht, col = morecolors(50),
                    show_row_names = F, show_column_names = F,
                    name = "distance")
myheatmap
```

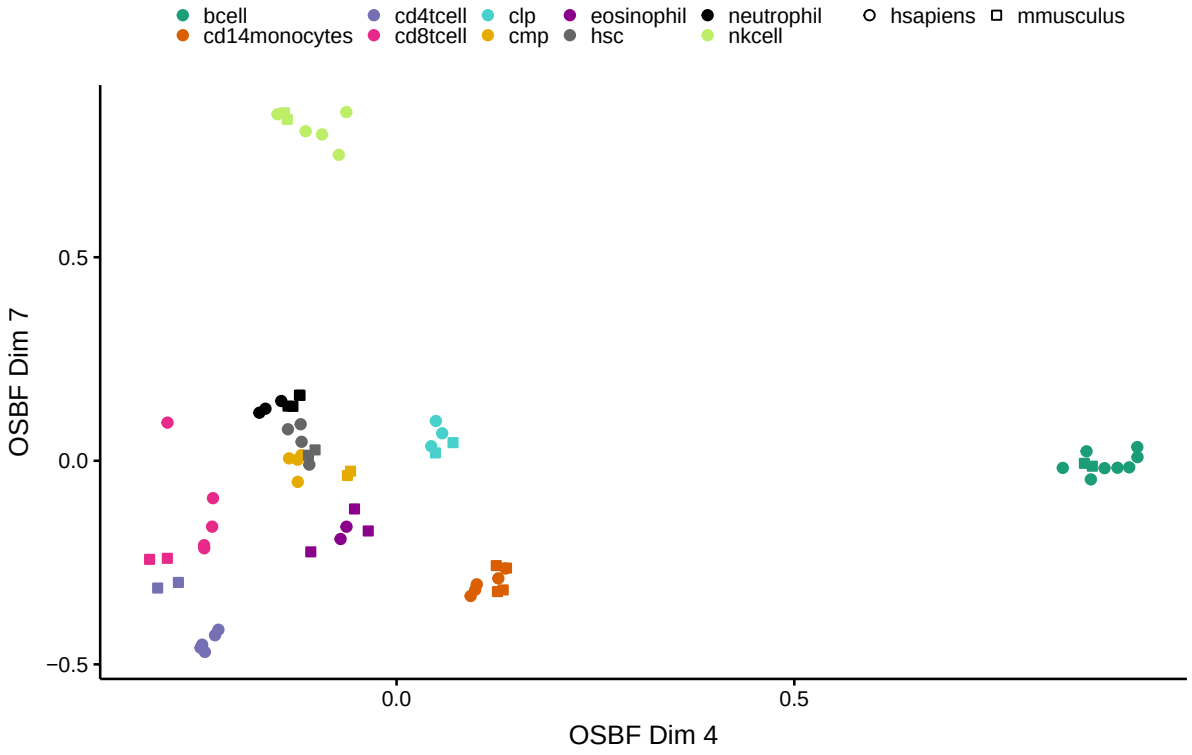


Gene expression profiles cluster by cell type independent of the species of origin.

8 Explore different dimensions

Let us look into individual dimensions to find cell type specific genes

```
# 2D plot for Dim2 and Dim3 [With V update]
sel_colors <- c("#1B9E77" ,"#D95F02" ,"#7570B3" ,"#E7298A", "mediumturquoise",
                "#E6AB02","darkmagenta", "#666666", "black","darkolivegreen2")
i <- 4
j <- 7
ggplot2::ggplot(df_proj, aes(x = df_proj[, i], y = df_proj[, j], col = tissue,
                             shape = species, fill = tissue)) +
  xlab(paste("OSBF Dim", i)) + ylab(paste("OSBF Dim", j)) +
  geom_point(size = 1.5) + scale_color_manual(values = sel_colors) +
  scale_shape_manual(values = c(21:25, 3:7)) +
  scale_fill_manual(values = sel_colors) +
  customTheme() +
  theme(legend.title = element_blank())
```



Dimension 4 +ve axis can be used to to identify bcell specific genes while dimension 7 +ve axis can be used to identify nk specific genes. We will first plot the loading vs cell type expression z-score to confirm this.

8.1 Expression specificity and eigengene loadings

```
# function to compute Tau
calc_tissue_specificity <- function(a) {
    a <- as.matrix(a)
    b <- a / matrixStats::rowMaxs(a)
    return(rowSums(1 - b) / (ncol(b) - 1))
}

Tau <- lapply(avg_counts, function(x) { calc_tissue_specificity(x)})
avg_counts_scaled <- lapply(avg_counts, function(x) { t(scale(t(x)))})

combine_expr <- list()
for (sp in names(avg_counts_scaled)) {
  x <- as.data.frame(avg_counts_scaled[[sp]])
  x[["Tau"]] <- Tau[[sp]]
  combine_expr[[sp]] <- x
}
```

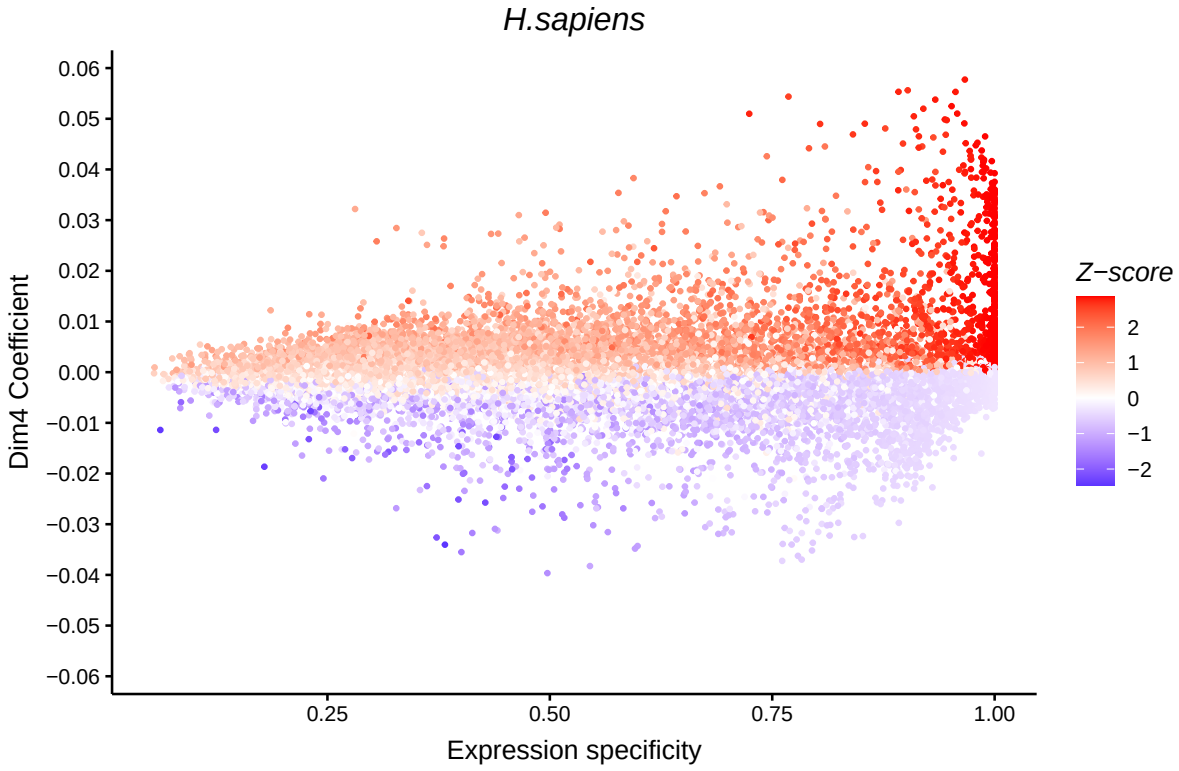
```
sel_dim <- 4
sel_tissue <- "bcell"
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = T]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                  "Tau", "coef")]
```

```
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
head(expr1)
```

```
##                 tissue_zscore       Tau          coef
## ENSG00000000003    -0.5999301 0.8510679 -0.0027397774
## ENSG00000000419     0.5454641 0.2015887 -0.0007854945
## ENSG00000000457     0.3170688 0.3593960 -0.0030531962
## ENSG00000000460     0.1085645 0.5322674 -0.0007646916
## ENSG00000000938     0.1805992 0.5880772  0.0098248748
## ENSG00000000971    -0.7199144 0.8117708 -0.0136224911
```

Dimension 4 U_i loadings vs expression specificity (τ) for humans. Z-score expression of bcell is used for coloring.

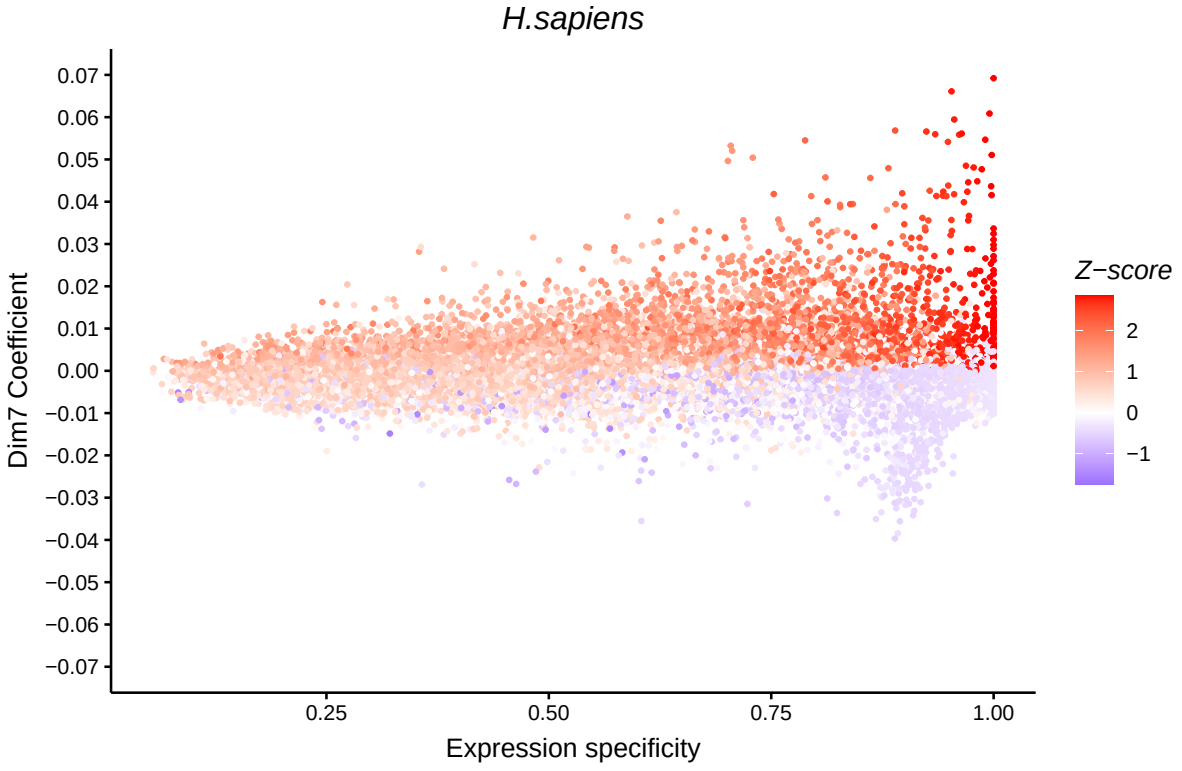
```
# plot scatter
mid <- 0
p1 <- ggplot2::ggplot(expr1, aes(x = Tau, y = coef, col = tissue_zscore)) + theme_bw() +
  geom_point(size = 0.5) + xlab("Expression specificity") +
  ylab(paste0("Dim", sel_dim, " Coefficient")) +
  scale_color_gradient2(midpoint = mid, low = "blue", mid = "white",
                        high = "red", space = "Lab") +
  scale_y_continuous(limits = c(-1 * max(abs(expr1$coef)), max(abs(expr1$coef))),
                     breaks = seq(-1 * round(max(abs(expr$coef)), 2),
                                  round(max(abs(expr$coef)), 2), by = 0.01)) +
  customTheme() +  theme(legend.position = "right",
                         legend.direction = "vertical") +
  labs(title = getScientificName(species), color = "Z-score") +
  theme(legend.key.size = unit(0.5, "cm"),
        plot.title = element_text(face = "italic"))
p1
```



Dimension 7 U_i loadings vs expression specificity (τ) for humans. Z-score expression of nkcell is used for coloring.

```
sel_dim <- 7
sel_tissue <- "nkcell"
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = T]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")

# plot scatter
mid <- 0
p1 <- ggplot2::ggplot(expr1, aes(x = Tau, y = coef, col = tissue_zscore)) + theme_bw() +
  geom_point(size = 0.5) + xlab("Expression specificity") +
  ylab(paste0("Dim", sel_dim, " Coefficient")) +
  scale_color_gradient2(midpoint = mid, low = "blue", mid = "white",
                        high = "red", space = "Lab") +
  scale_y_continuous(limits = c(-1 * max(abs(expr1$coef)), max(abs(expr1$coef))),
                     breaks = seq(-1 * round(max(abs(expr$coef)), 2),
                                  round(max(abs(expr$coef)), 2), by = 0.01)) +
  customTheme() +  theme(legend.position = "right",
                         legend.direction = "vertical") +
  labs(title = getScientificName(species), color = "Z-score") +
  theme(legend.key.size = unit(0.5, "cm"),
        plot.title = element_text(face = "italic"))
p1
```



##

GO analysis

Download the GO files from: https://figshare.com/s/d96c586d5e53199d5370 We will perform the gene ontology analysis for genes with high coefficients. We will use the goseq Bioconductor package to perform GO enrichment analysis.

```
# install packages
pkgs <- c("goseq")
require_install <- pkgs[!(pkgs %in% row.names(installed.packages()))]
if (length(require_install)) {
  if (!require("BiocManager", quietly = TRUE))
      install.packages("BiocManager")
  BiocManager::install("goseq")
}
suppressPackageStartupMessages({
  library(goseq)
})
```

Let us perform GO analysis for top bcell-specific genes in Dimension 4. The bcell-specific genes have positive loadings.

```
sel_dim <- 4
sel_tissue <- "bcell"
top_genes <- 100
# axis positive (pos) or negative (neg)
sel_sign <- "pos"
```

```
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = T]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                          "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
if (sel_sign == "neg") {
  cat("\n selecting negative loadings")
  expr1_selsign <- expr1[expr1$coef < 0, ]
  expr1_bgsign <- expr1[expr1$coef >= 0, ]
} else {
  cat("\n selecting positive loadings")
  expr1_selsign <- expr1[expr1$coef >= 0, ]
  expr1_bgsign <- expr1[expr1$coef < 0, ]
}
```

```
##
##  selecting positive loadings
```

```
expr1_selsign$score <- expr1_selsign$Tau * abs(expr1_selsign$coef)
expr1_selsign$rank <- rank(-1 * expr1_selsign$score)
expr1_selsign <- expr1_selsign[order(expr1_selsign$rank), ]
# gene list of interest
genes_fg <- row.names(expr1_selsign[expr1_selsign$rank <= top_genes, ])
# background genes
# For GO analysis, we will use genes with opposite sign loadings as
# the background.
genes_bg <- row.names(expr1_bgsign)

genes_bg <- genes_bg[!genes_bg %in% genes_fg]
```

```
genome <- "hg38"
total_genes <- unique(c(genes_fg, genes_bg))
up_genes <- as.integer(total_genes %in%  genes_fg)
names(up_genes) <- total_genes
```

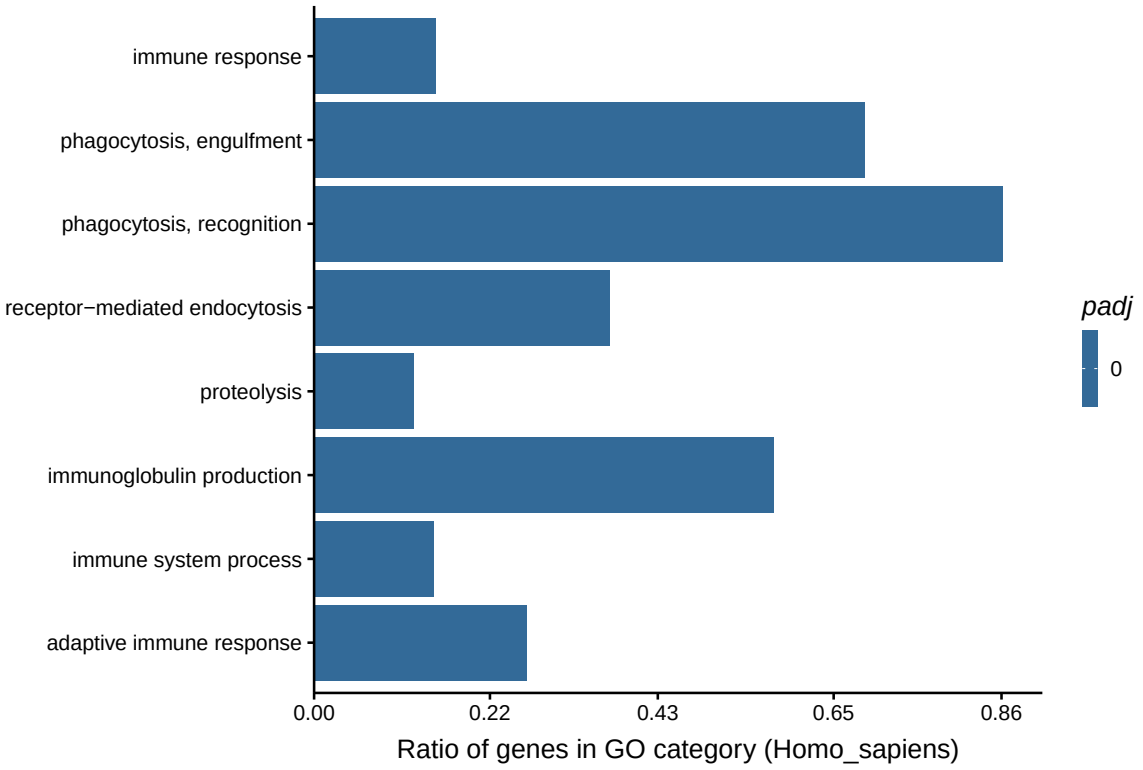
```
## Using manually entered categories.
## For 9390 genes, we could not find any categories. These genes will be excluded.
## To force their use, please run with use_genes_without_cat=TRUE (see documentation).
## This was the default behavior for version 1.15.1 and earlier.
## Calculating the p-values...
## 'select()' returned 1:1 mapping between keys and columns
```

```
head(go.sub)
```

```
##        category numDEInCat numInCat                          term ontology padj
## 686  GO:0002250         55      207      adaptive immune response       BP    0
## 726  GO:0002376         56      373         immune system process       BP    0
## 727  GO:0002377         23       40     immunoglobulin production       BP    0
## 2266 GO:0006508         34      274                   proteolysis       BP    0
## 2481 GO:0006898         34       92 receptor-mediated endocytosis       BP    0
## 2488 GO:0006910         31       36     phagocytosis, recognition       BP    0
##       ratio
## 686  0.2657
## 726  0.1501
## 727  0.5750
## 2266 0.1241
## 2481 0.3696
## 2488 0.8611
```

Barplot with top human GO terms and their p-value.

```
# GO enrichment plot for human
go_out <- head(go.sub, n = 8)
go_out$padj <- as.numeric(go_out$padj)
go_out$term <- factor(go_out$term, levels = go_out$term)
breaks <- round(c(0, 1 / 4, 2 / 4, 3 / 4, 1) * max(go_out[["ratio"]]), 2)
go_plot <- ggplot2::ggplot(go_out, aes(x = term, y = ratio, fill = padj)) + geom_col() +
  scale_y_continuous(expand = c(0, 0), breaks = breaks,
                     limits = c(0, max(go_out[["ratio"]] + 0.05))) +
  scale_x_discrete() + coord_flip() +
  scale_color_gradient(low = "blue", high = "red") +
  ylab(paste0("Ratio of genes in GO category (",
              species, ")")) +
  xlab("") + customTheme() + theme(legend.position = "right",
                                   legend.direction = "vertical",
                                   plot.margin = unit(c(10, 5, 5, 5), "mm"))
go_plot
```



GO analysis for top nkcell-specific genes in Dimension 7.

```
sel_dim <- 7
sel_tissue <- "nkcell"
top_genes <- 100
# axis positive (pos) or negative (neg)
sel_sign <- "pos"

species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = T]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue), "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
if (sel_sign == "neg") {
  cat("\n selecting negative loadings")
  expr1_selsign <- expr1[expr1$coef < 0, ]
  expr1_bgsign <- expr1[expr1$coef >= 0, ]
} else {
  cat("\n selecting positive loadings")
  expr1_selsign <- expr1[expr1$coef >= 0, ]
  expr1_bgsign <- expr1[expr1$coef < 0, ]
}
```

```
##
##  selecting positive loadings
```

```
expr1_selsign$score <- expr1_selsign$Tau * abs(expr1_selsign$coef)
expr1_selsign$rank <- rank(-1 * expr1_selsign$score)
expr1_selsign <- expr1_selsign[order(expr1_selsign$rank), ]
# gene list of interest
```

```
genes_fg <- row.names(expr1_selsign[expr1_selsign$rank <= top_genes, ])
# background genes
# For GO analysis, we will use genes with opposite sign loadings as
# the background.
genes_bg <- row.names(expr1_bgsign)

genes_bg <- genes_bg[!genes_bg %in% genes_fg]
genome <- "hg38"
total_genes <- unique(c(genes_fg, genes_bg))
up_genes <- as.integer(total_genes %in%  genes_fg)
names(up_genes) <- total_genes
```

```
## Using manually entered categories.

## For 6689 genes, we could not find any categories. These genes will be excluded.

## To force their use, please run with use_genes_without_cat=TRUE (see documentation).

## This was the default behavior for version 1.15.1 and earlier.

## Calculating the p-values...

## 'select()' returned 1:1 mapping between keys and columns
```

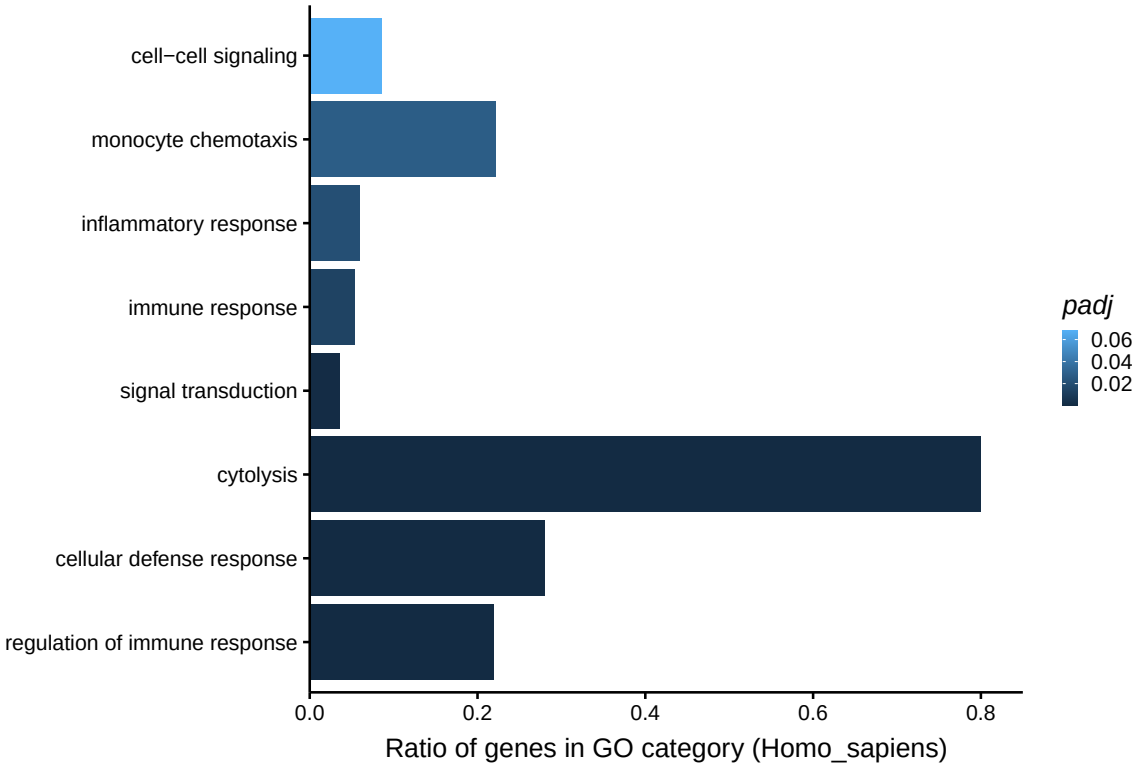
```
head(go.sub)
```

```
##        category numDEInCat numInCat                          term ontology
## 8931 GO:0050776         16       73 regulation of immune response       BP
## 2485 GO:0006968          7       25     cellular defense response       BP
## 4533 GO:0019835          4        5                     cytolysis       BP
## 2577 GO:0007165         20      554           signal transduction       BP
## 2480 GO:0006955         10      188               immune response       BP
## 2479 GO:0006954          9      150         inflammatory response       BP
##              padj  ratio
## 8931 1.297786e-14 0.2192
## 2485 1.503696e-05 0.2800
## 4533 1.057068e-04 0.8000
## 2577 6.989144e-04 0.0361
## 2480 1.363161e-02 0.0532
## 2479 2.014008e-02 0.0600
```

Barplot with top human GO terms and their p-value.

```
# GO enrichment plot for human
go_out <- head(go.sub, n = 8)
go_out$padj <- as.numeric(go_out$padj)
go_out$term <- factor(go_out$term, levels = go_out$term)
breaks <- round(c(0, 1 / 4, 2 / 4, 3 / 4, 1) * max(go_out[["ratio"]]), 2)
go_plot <- ggplot2::ggplot(go_out, aes(x = term, y = ratio, fill = padj)) + geom_col() +
  scale_y_continuous(expand = c(0, 0), breaks = breaks,
                     limits = c(0, max(go_out[["ratio"]] + 0.05))) +
  scale_x_discrete() + coord_flip() +
  scale_color_gradient(low = "blue", high = "red") +
  ylab(paste0("Ratio of genes in GO category (",
              species, ")")) +
  xlab("") + customTheme() + theme(legend.position = "right",
                                   legend.direction = "vertical",
```

```
                                    plot.margin = unit(c(10, 5, 5, 5), "mm"))
go_plot
```



9 Identify cell type specific genes

Next, we will identify those genes with significant contribution to different cell types. We will first create a null distribution of scores for the coefficient and identify genes of interest with respect to the null.

9.1 Create shuffled counts to generate null

We will create mean profiles based on shuffled reads.

```
# set seed
s1 <- 32
s2 <- 135
species <- c("Homo_sapiens", "Mus_musculus")
species_short <- sapply(species, getSpeciesShortName)
# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
counts_list_shuff <- metadata_list_shuff <- avg_counts_shuff <- list()
for (sp in species) {
  # reading raw counts
  counts <- read.table(paste0(path, "human_mouse_blood_counts/", sp,
                              "_blood_rawcounts.tsv"), header = T,
                       sep = "\t", row.names = 1)
  info <- tstrsplit(colnames(counts), "_")
  metadata <- data.frame(project = info[[1]],
        species = info[[2]],
        tissue = info[[3]],
```

```
        gsm = info[[4]],
        name = colnames(counts),
        stringsAsFactors = F)
  metadata$ref <- seq_len(nrow(metadata))
  metadata$key <- paste0(metadata$species, "_", metadata$tissue)
  metadata$tissue_factor <- factor(metadata$tissue)

  counts_avg <- calcAvgCounts(counts, metadata)
  cnames <- colnames(counts_avg)
  rnames <- row.names(counts_avg)
  set.seed(s1)
  counts_avg <- as.data.frame(apply(counts_avg, 2, sample))
  set.seed(s2)
  counts_avg <- as.data.frame(t(apply(counts_avg, 1, sample)))
  colnames(counts_avg) <- cnames
  row.names(counts_avg) <- rnames
  # normalize the shuffled counts to log TPM
  # set the path to the working directory. Change this accordingly
  path <- "~/Dropbox/0.Analysis/0.paper/"
  gene_length <- read.table(paste0(path, "ensembl94_annotation/", sp, "_genelength.tsv"),
                        sep = "\t",header = T, row.names = 1, stringsAsFactors = F)
  if (!all(row.names(counts_avg) %in% row.names(gene_length))) stop("Error")
  gene_length$Length <- gene_length$Length/1e3
  gene_length <- gene_length[row.names(counts_avg), , drop=T]
  names(gene_length) <- row.names(counts_avg)
  counts_tpm <- normalizeTPM(rawCounts = counts_avg, gene_len = gene_length)
  min_tpm <- 1
  counts_tpm[counts_tpm < min_tpm] <- 1
  counts_tpm <- log2(counts_tpm)

  info <- tstrsplit(colnames(counts_tpm),"_")
  metadata <- data.frame(
        species = info[[1]],
        tissue = info[[2]],
        name = colnames(counts_tpm),
        stringsAsFactors = F)
  metadata$key <- paste0(metadata$species,"_",metadata$tissue)
  avg_counts_shuff[[sp]] <- calcAvgCounts(counts_tpm, metadata)
  # write.table(avg_counts_shuff[[sp]],file=paste0(mainDir, "shuffledavgCounts/avg_logTPM_",
  #                                species,"_SHUFF_counts_seed",z,"_",z1,".tsv"),
  #           sep="\t",quote=F,col.names=NA)
  counts_list_shuff[[sp]] <- counts_tpm
  metadata_list_shuff[[sp]] <- metadata
}
```

```
##
## TPM counts returned
## TPM counts returned
```

```
# dims
sapply(counts_list_shuff, dim)
```

```
##      Homo_sapiens Mus_musculus
## [1,]        58676        54446
## [2,]           10           10
```

```
# remove zero counts
avg_counts_shuff <- lapply(avg_counts_shuff, removeZeros)
sapply(avg_counts_shuff, dim)
```

```
##      Homo_sapiens Mus_musculus
## [1,]        49021        47520
## [2,]           10           10
```

9.2 OSBF call for shuffled counts

Depending upon the shuffled counts this could take a while. Decrease `tol` for lower factorization error.

```
cat(format(Sys.time(), "%a %b %d %X %Y"),"\n")
```

```
## Fri Jun 17 12:26:23 PM 2022
```

```
osbf_shuf <- SBF(avg_counts_shuff, transform_matrix = TRUE, orthogonal = TRUE,
                 tol = 1e-2)
```

```
##
## OSBF optimizing factorization error
```

```
cat(format(Sys.time(), "%a %b %d %X %Y"),"\n")
```

```
## Fri Jun 17 12:26:25 PM 2022
```

```
osbf_shuf$error
```

```
## [1] 5531.594
```

Compute Tau and scaled expression for the null datasets

```
Tau_null <- lapply(avg_counts_shuff, function(x) {calc_tissue_specificity(x)})
avg_counts_shuff_scaled <- lapply(avg_counts_shuff, function(x){ t(scale(t(x)))})
combine_expr_null <- list()
for (sp in names(avg_counts_shuff_scaled)) {
  x <- as.data.frame(avg_counts_shuff_scaled[[sp]])
  x[["Tau"]] <- Tau_null[[sp]]
  combine_expr_null[[sp]] <- x
}
```

9.3 Identify b cell specific genes

Now let us find genes with significant loadings in dimension 4

```
sel_dim <- 4
sel_tissue <- "bcell"
# axis positive (pos) or negative (neg)
sel_sign <- "pos"
species <- "Homo_sapiens"
species_short <- "hsapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = T]
expr1 <- expr[, c(paste0(species_short, "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
```

```
# null loadings for the same dimensions
expr_null <- combine_expr_null[[species]]
null_u <- osbf_shuf$u[[species]]
expr_null[["coef"]] <- null_u[, sel_dim, drop = T]
expr1_null <- expr_null[, c(paste0(species_short,"_",sel_tissue),"Tau","coef")]
if (sel_sign == "pos") {
  expr1 <- expr1[expr1$coef >= 0, ]
  expr1_null <- expr1_null[expr1_null$coef >= 0, ]
} else if (sel_dim == "neg") {
  expr1 <- expr1[expr1$coef < 0, ]
  expr1_null <- expr1_null[expr1_null$coef < 0, ]
}
expr1$score <- expr1$Tau * abs(expr1$coef)
expr1$rank <- rank(-1 * expr1$score)
expr1 <- expr1[order(expr1$rank), ]

expr1_null$score <- expr1_null$Tau * abs(expr1_null$coef)
expr1$pvalue <- sapply(expr1$score, function(x) { sum(as.integer(expr1_null$score > x))/length(expr1_nul
head(expr1)
```

```
##                 tissue_zscore       Tau       coef      score rank pvalue
## ENSG00000211594      2.792820 0.9665629 0.05773116 0.05580079    1      0
## ENSG00000242472      2.785474 0.9560529 0.05530281 0.05287241    2      0
## ENSG00000211677      2.785243 0.9332910 0.05377779 0.05019033    3      0
## ENSG00000211900      2.641843 0.9023546 0.05561730 0.05018653    4      0
## ENSG00000211595      2.824715 0.9516554 0.05248654 0.04994909    5      0
## ENSG00000211593      2.737777 0.8919587 0.05532098 0.04934403    6      0
```

Find the number of genes with significant pvalue

```
# cut off for the p-value
alpha <- 1e-3
summary(expr1$pvalue <= alpha)
```

```
##    Mode   FALSE    TRUE
## logical   12882     286
```

Lets check the top 10 genes for dimension 4

```
# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
gene_info <- read.table(paste0(path, "ensembl94_annotation/", species_short, "_genes_completeinfo.tsv")
                                sep = "\t", header = T, quote = "\"")
gene_info <- gene_info[!duplicated(gene_info$ensembl_gene_id), ]
gene_info <- gene_info[gene_info$ensembl_gene_id %in% row.names(expr1), ]
row.names(gene_info) <- gene_info$ensembl_gene_id
gene_info <- gene_info[row.names(expr1), ]
expr1$gene_name <- gene_info$external_gene_name
expr1$biotype <- gene_info$gene_biotype
head(expr1, n = 10)
```

```
##                 tissue_zscore       Tau       coef      score rank pvalue
## ENSG00000211594      2.792820 0.9665629 0.05773116 0.05580079    1      0
## ENSG00000242472      2.785474 0.9560529 0.05530281 0.05287241    2      0
## ENSG00000211677      2.785243 0.9332910 0.05377779 0.05019033    3      0
## ENSG00000211900      2.641843 0.9023546 0.05561730 0.05018653    4      0
```

```
## ENSG00000211595      2.824715 0.9516554 0.05248654 0.04994909    5      0
## ENSG00000211593      2.737777 0.8919587 0.05532098 0.04934403    6      0
## ENSG00000211949      2.818037 0.9579632 0.05102114 0.04887637    7      0
## ENSG00000237111      2.741113 0.9198911 0.05198447 0.04782005    8      0
## ENSG00000264781      2.818623 0.9660110 0.04908614 0.04741775    9      0
## ENSG00000211679      2.805381 0.9440552 0.04986516 0.04707546   10      0
##                  gene_name         biotype
## ENSG00000211594     IGKJ4       IG_J_gene
## ENSG00000242472     IGHJ5       IG_J_gene
## ENSG00000211677     IGLC2       IG_C_gene
## ENSG00000211900     IGHJ6       IG_J_gene
## ENSG00000211595     IGKJ3       IG_J_gene
## ENSG00000211593     IGKJ5       IG_J_gene
## ENSG00000211949  IGHV3-23       IG_V_gene
## ENSG00000237111    IGHJ3P IG_J_pseudogene
## ENSG00000264781   MIR4537           miRNA
## ENSG00000211679     IGLC3       IG_C_gene
```

We see that that the top genes are mostly immunoglobulin genes.

9.4 Identify nk cell specific genes

Now let us find genes with significant loadings in dimension 7 for mouse

```
sel_dim <- 7
sel_tissue <- "nkcell"
# axis positive (pos) or negative (neg)
sel_sign <- "pos"
species <- "Mus_musculus"
species_short <- "mmusculus"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = T]
expr1 <- expr[, c(paste0(species_short, "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")

# null loadings for the same dimensions
expr_null <- combine_expr_null[[species]]
null_u <- osbf_shuf$u[[species]]
expr_null[["coef"]] <- null_u[, sel_dim, drop = T]
expr1_null <- expr_null[, c(paste0(species_short,"_",sel_tissue),"Tau","coef")]
if (sel_sign == "pos") {
  expr1 <- expr1[expr1$coef >= 0, ]
  expr1_null <- expr1_null[expr1_null$coef >= 0, ]
} else if (sel_dim == "neg") {
  expr1 <- expr1[expr1$coef < 0, ]
  expr1_null <- expr1_null[expr1_null$coef < 0, ]
}
expr1$score <- expr1$Tau * abs(expr1$coef)
expr1$rank <- rank(-1 * expr1$score)
expr1 <- expr1[order(expr1$rank), ]

expr1_null$score <- expr1_null$Tau * abs(expr1_null$coef)
expr1$pvalue <- sapply(expr1$score, function(x) { sum(as.integer(expr1_null$score > x))/length(expr1_nul
```

```
#head(expr1)
```

Find the number of genes with significant pvalue

```
# cut off for the p-value
alpha <- 1e-3
summary(expr1$pvalue <= alpha)
```

```
##    Mode   FALSE    TRUE
## logical   10340     220
```

Lets check the top 10 genes for dimension 7

```
# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
gene_info <- read.table(paste0(path, "ensembl94_annotation/", species_short,
                               "_genes_completeinfo.tsv"),
                               sep = "\t", header = T, quote = "\"")
gene_info <- gene_info[!duplicated(gene_info$ensembl_gene_id), ]
gene_info <- gene_info[gene_info$ensembl_gene_id %in% row.names(expr1), ]
row.names(gene_info) <- gene_info$ensembl_gene_id
gene_info <- gene_info[row.names(expr1), ]
expr1$gene_name <- gene_info$external_gene_name
expr1$biotype <- gene_info$gene_biotype
head(expr1, n = 10)
```

```
##                    tissue_zscore       Tau       coef      score rank pvalue
## ENSMUSG00000023132      2.813943 0.9641020 0.07006787 0.06755257    1      0
## ENSMUSG00000062524      2.845611 0.9980411 0.06175483 0.06163385    2      0
## ENSMUSG00000089727      2.834113 0.9898324 0.05885085 0.05825248    3      0
## ENSMUSG00000033024      2.846050 1.0000000 0.05716660 0.05716660    4      0
## ENSMUSG00000079852      2.832204 0.9878951 0.05652688 0.05584263    5      0
## ENSMUSG00000030325      2.788546 0.9745317 0.05619826 0.05476698    6      0
## ENSMUSG00000072721      2.846050 1.0000000 0.05392672 0.05392672    7      0
## ENSMUSG00000067599      2.844010 0.9937175 0.05270266 0.05237155    8      0
## ENSMUSG00000050241      2.843993 0.9957654 0.05142650 0.05120873    9      0
## ENSMUSG00000096176      2.846050 1.0000000 0.05034802 0.05034802   10      0
##                    gene_name                          biotype
## ENSMUSG00000023132      Gzma                   protein_coding
## ENSMUSG00000062524      Ncr1                   protein_coding
## ENSMUSG00000089727     Klra8                   protein_coding
## ENSMUSG00000033024     Klra9                   protein_coding
## ENSMUSG00000079852     Klra4                   protein_coding
## ENSMUSG00000030325    Klrb1c                   protein_coding
## ENSMUSG00000072721 Klra14-ps transcribed_unprocessed_pseudogene
## ENSMUSG00000067599     Klra7                   protein_coding
## ENSMUSG00000050241     Klre1                   protein_coding
## ENSMUSG00000096176     Trdd2                        TR_D_gene
```

We identify common NK cell marker genes

session info

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.4 LTS
```

```
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] goseq_1.48.0           geneLenDataBase_1.32.0 BiasedUrn_1.07
##  [4] ggplot2_3.3.6          ggthemes_4.2.4         RColorBrewer_1.1-3
##  [7] ComplexHeatmap_2.12.0  matrixStats_0.62.0     dplyr_1.0.9
## [10] data.table_1.14.2      SBF_1.0.0.0
##
## loaded via a namespace (and not attached):
##   [1] colorspace_2.0-3            rjson_0.2.21
##   [3] ellipsis_0.3.2              rprojroot_2.0.3
##   [5] circlize_0.4.15             XVector_0.36.0
##   [7] GenomicRanges_1.48.0        GlobalOptions_0.1.2
##   [9] fs_1.5.2                    clue_0.3-61
##  [11] rstudioapi_0.13             farver_2.1.0
##  [13] remotes_2.4.2               bit64_4.0.5
##  [15] AnnotationDbi_1.58.0        fansi_1.0.3
##  [17] xml2_1.3.3                  splines_4.2.0
##  [19] codetools_0.2-18            doParallel_1.0.17
##  [21] cachem_1.0.6                knitr_1.39
##  [23] pkgload_1.2.4               Rsamtools_2.12.0
##  [25] GO.db_3.15.0                dbplyr_2.2.0
##  [27] cluster_2.1.3               png_0.1-7
##  [29] compiler_4.2.0              httr_1.4.3
##  [31] assertthat_0.2.1            Matrix_1.4-1
##  [33] fastmap_1.1.0               cli_3.3.0
##  [35] htmltools_0.5.2             prettyunits_1.1.1
##  [37] tools_4.2.0                 gtable_0.3.0
##  [39] glue_1.6.2                  GenomeInfoDbData_1.2.8
##  [41] rappdirs_0.3.3              tinytex_0.38
##  [43] Rcpp_1.0.8.3                Biobase_2.56.0
##  [45] vctrs_0.4.1                 Biostrings_2.64.0
##  [47] nlme_3.1-157                rtracklayer_1.56.0
##  [49] iterators_1.0.14            xfun_0.31
##  [51] stringr_1.4.0               ps_1.7.0
##  [53] brio_1.1.3                  testthat_3.1.4
##  [55] lifecycle_1.0.1             restfulr_0.0.15
##  [57] devtools_2.4.3              XML_3.99-0.10
##  [59] zlibbioc_1.42.0             scales_1.2.0
```

```
##  [61] hms_1.1.1                  MatrixGenerics_1.8.0
##  [63] parallel_4.2.0             SummarizedExperiment_1.26.1
##  [65] curl_4.3.2                 yaml_2.3.5
##  [67] memoise_2.0.1              biomaRt_2.52.0
##  [69] stringi_1.7.6              RSQLite_2.2.14
##  [71] highr_0.9                  S4Vectors_0.34.0
##  [73] BiocIO_1.6.0               desc_1.4.1
##  [75] foreach_1.5.2              filelock_1.0.2
##  [77] GenomicFeatures_1.48.3     BiocGenerics_0.42.0
##  [79] pkgbuild_1.3.1             BiocParallel_1.30.3
##  [81] shape_1.4.6                GenomeInfoDb_1.32.2
##  [83] rlang_1.0.2                pkgconfig_2.0.3
##  [85] bitops_1.0-7               evaluate_0.15
##  [87] lattice_0.20-45            purrr_0.3.4
##  [89] GenomicAlignments_1.32.0   labeling_0.4.2
##  [91] bit_4.0.4                  processx_3.5.3
##  [93] tidyselect_1.1.2           magrittr_2.0.3
##  [95] R6_2.5.1                   IRanges_2.30.0
##  [97] generics_0.1.2             DelayedArray_0.22.0
##  [99] DBI_1.1.2                  mgcv_1.8-40
## [101] pillar_1.7.0               withr_2.5.0
## [103] KEGGREST_1.36.2            RCurl_1.98-1.7
## [105] tibble_3.1.7               crayon_1.5.1
## [107] utf8_1.2.2                 BiocFileCache_2.4.0
## [109] rmarkdown_2.14             GetoptLong_1.0.5
## [111] progress_1.2.2             usethis_2.1.6
## [113] blob_1.2.3                 callr_3.7.0
## [115] digest_0.6.29              stats4_4.2.0
## [117] munsell_0.5.0              sessioninfo_1.2.2
```

References

Choi, Jarny, Tracey M Baldwin, Mae Wong, Jessica E Bolden, Kirsten A Fairfax, Erin C Lucas, Rebecca Cole, et al. 2019. "Haemopedia RNA-seq: a database of gene expression during haematopoiesis in mice and humans." *Nucleic Acids Research* 47 (D1): D780–85.

Corces, M Ryan, Jason D Buenrostro, Beijing Wu, Peyton G Greenside, Steven M Chan, Julie L Koenig, Michael P Snyder, et al. 2016. "Lineage-Specific and Single-Cell Chromatin Accessibility Charts Human Hematopoiesis and Leukemia Evolution." *Nature Genetics* 48 (10): 1193–203.