

Analysis of blood cell types from human and mouse using OSBF

Amal Thomas

Contents

1 Data	1
2 Species and cell types	2
3 Load gene expression profiles	2
3.1 Compute mean expression profiles	3
4 OSBF	3
5 Project datasets into common space	6
5.1 Two-dimensional projection plots	7
6 Clustering in the common space	13
7 Explore different dimensions	14
7.1 Expression specificity and eigengene loadings	15
7.2 GO analysis	18
8 Identify cell type specific genes	23
8.1 Create shuffled counts to generate null	23
8.2 OSBF call for shuffled counts	24
8.3 Identify b cell specific genes	25
8.4 Identify nk cell specific genes	27
9 Exploring dimension 1 of the common space	28
10 Session info	31
References	33

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 Load 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 = TRUE, sep = "\t",
                        row.names = 1)
  info <- data.table:::tstrsplit(colnames(counts), "_")
  metadata <- data.frame(project = info[[1]],
                         species = info[[2]],
                         tissue = info[[3]],
                         gsm = info[[4]],
                         name = colnames(counts),
                         stringsAsFactors = FALSE)
  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. We will use `calcAvgCounts` function from the SBF package.

```

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) {
  data.table:::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 = FALSE]
}

```

4 OSBF

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")

## Wed Jun 22 04:08:25 PM 2022
osbf_noVupdate <- SBF(avg_counts, orthogonal = TRUE, transform_matrix = TRUE,
minimizeError = TRUE,
optimizeV = FALSE, tol = 1e-3)

##
## OSBF optimizing factorization error
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")

## Wed Jun 22 04:08:26 PM 2022
# Now lets compute OSBF updating all three factors (U, Delta, V)
cat("optimizing V = TRUE\n")

## optimizing V = TRUE
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")

## Wed Jun 22 04:08:26 PM 2022
osbf <- SBF(avg_counts, orthogonal = TRUE, transform_matrix = TRUE,
minimizeError = TRUE,
optimizeV = TRUE, tol = 1e-3)

##
## OSBF optimizing factorization error
cat(format(Sys.time(), "%a %b %d %X %Y"), "\n")

## Wed Jun 22 04:08:32 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]: 88161.99
cat("\n", sprintf("%-27s:", "# of update [No V update]"), sprintf("%16d",
osbf_noVupdate$error_pos))

##
## # of update [No V update] : 6
```

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

```
##  
## # of update [With V update]: 202
```

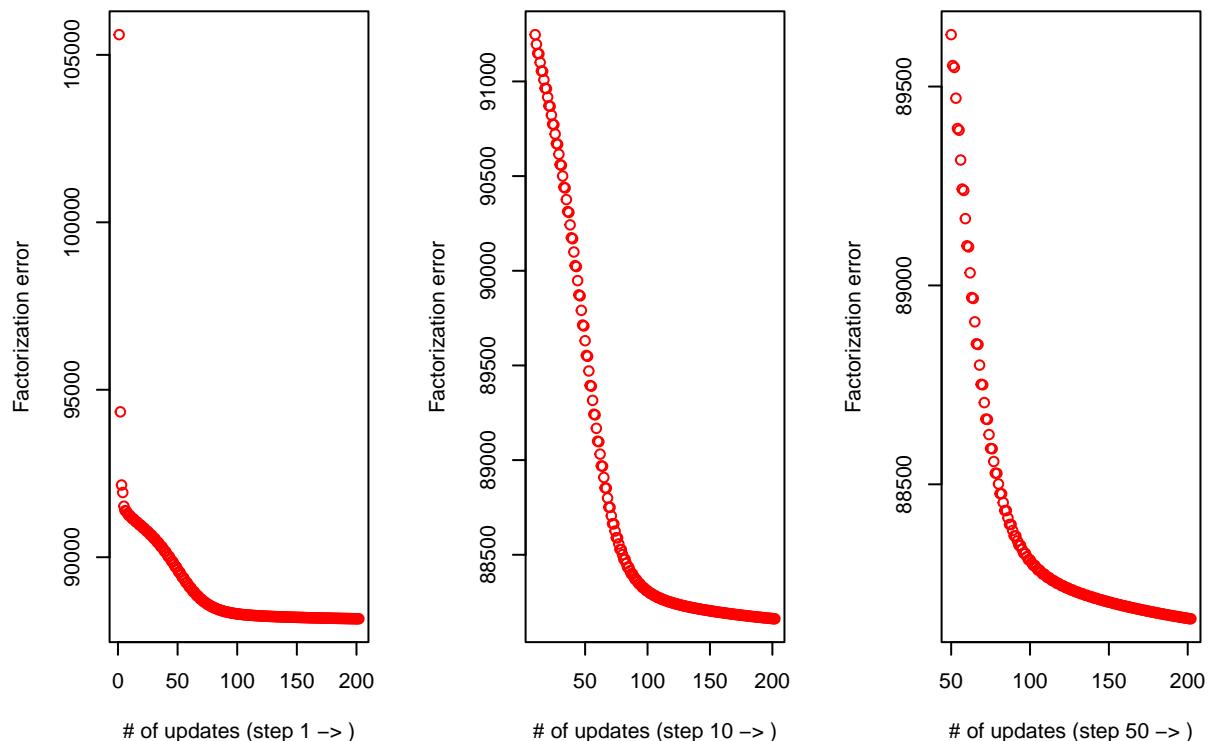
Optimization with updating V achieves a lower decomposition error.

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

```
## [1] 1.195588
```

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 $\Delta_i = \text{diag}(\delta_{i1}, \dots, \delta_{i6})$

```
cat("\nPercentage for each delta [No V update]:")
```

```

##
## Percentage for each delta [No V update]:
percentInfo_noVupdate <- calcPercentInfo(osbf_noVupdate)
for (i in names(osbf_noVupdate$delta)) {
  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 <- calcPercentInfo(osbf)
for (i in names(osbf$delta)) {
  cat("\n", sprintf("%-25s:", i), sprintf("%8.2f", percentInfo[[i]]))
}

```

```

##
## Homo_sapiens : 86.36 5.07 3.80 1.16 1.08 1.12 0.51 0.56
## Mus_musculus : 91.34 2.47 2.62 1.09 0.70 0.45 0.85 0.32

```

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

5 Project datasets into common space

Project individual profiles and average counts to common space by computing $D_i^T U_i \Delta^{-1}$. We will projectCounts function from the SBF package for this.

```

# project profiles using no V update estimates
# we can project both mean expression profiles as well as individual expression
# profiles
df_proj_avg_noVupdate <- projectCounts(avg_counts, osbf_noVupdate)
meta <- data.table::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 <- projectCounts(counts_list_sub, osbf_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))

```

Now let us also project profiles with the updated V

```

# project using V update estimates
df_proj_avg <- projectCounts(avg_counts, osbf)
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 <- projectCounts(counts_list_sub, osbf)
meta1 <- data.table::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))

```

5.1 Two-dimensional projection plots

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")))
}

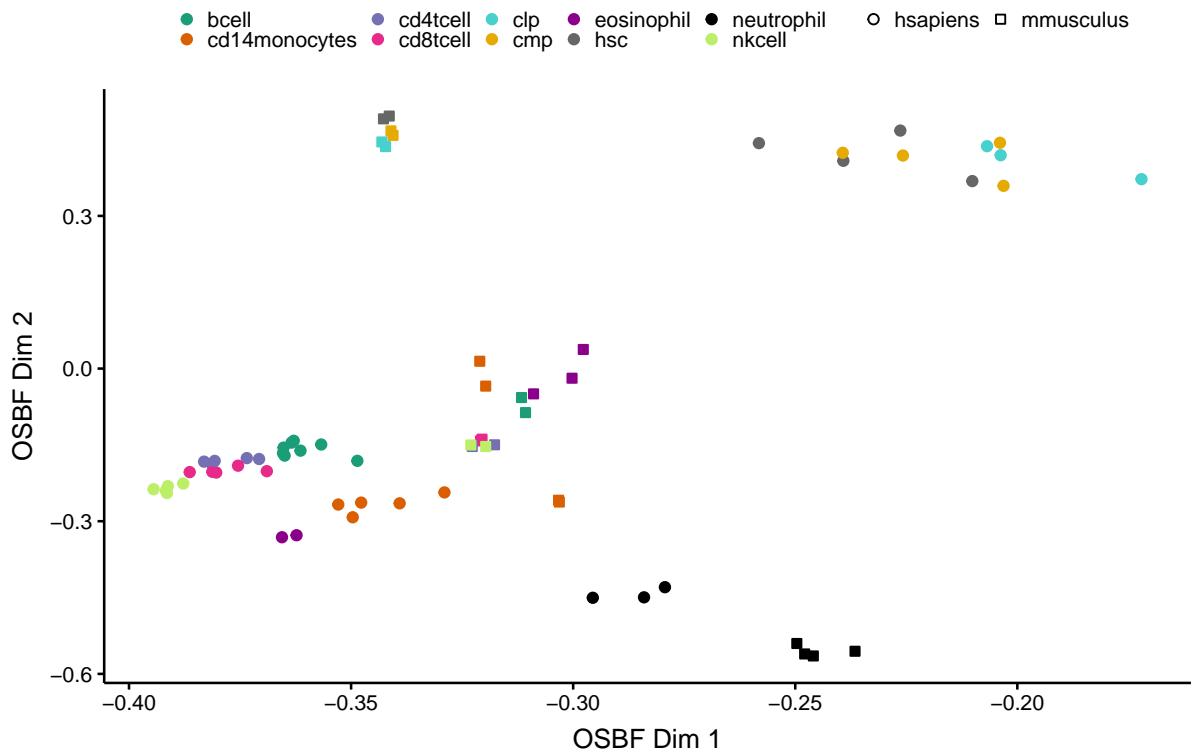
```

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

```

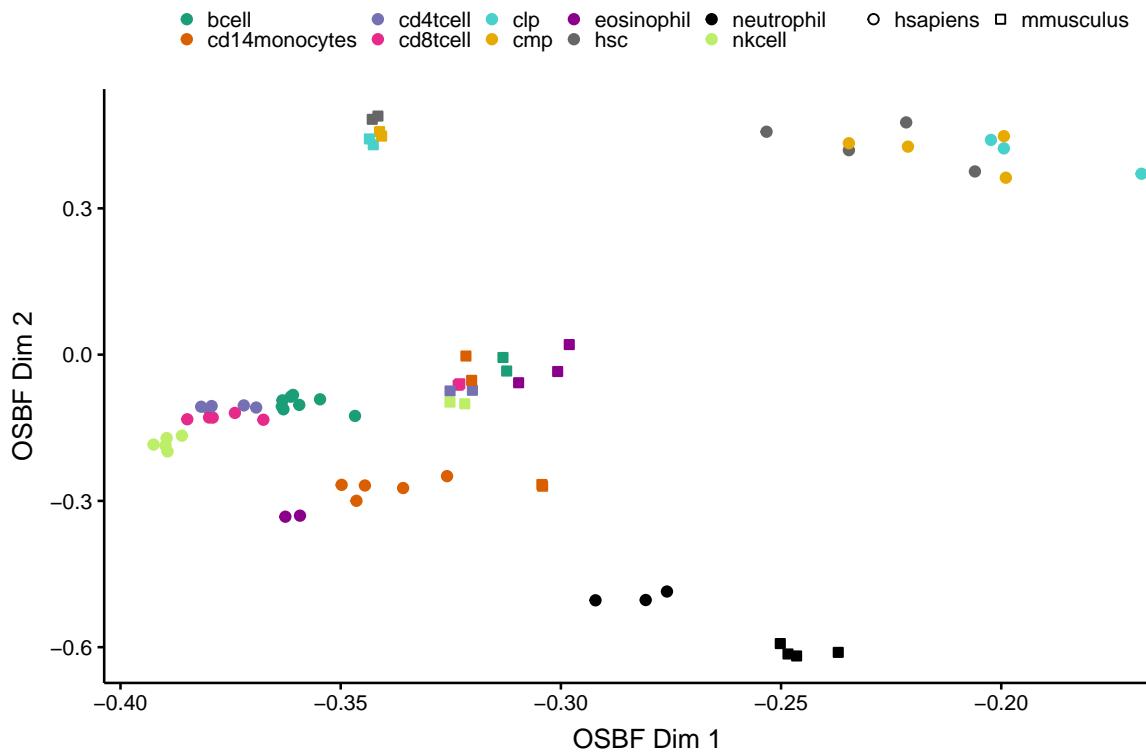
sel_colors <- c("#1B9E77", "#D95F02", "#7570B3", "#E7298A", "mediumturquoise",
                 "#E6AB02", "darkmagenta", "#666666", "black", "darkolivegreen2")
i <- 1
j <- 2
ggplot2::ggplot(df_proj_noVupdate, aes(x = df_proj_noVupdate[, i],
                                         y = df_proj_noVupdate[, 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())

```



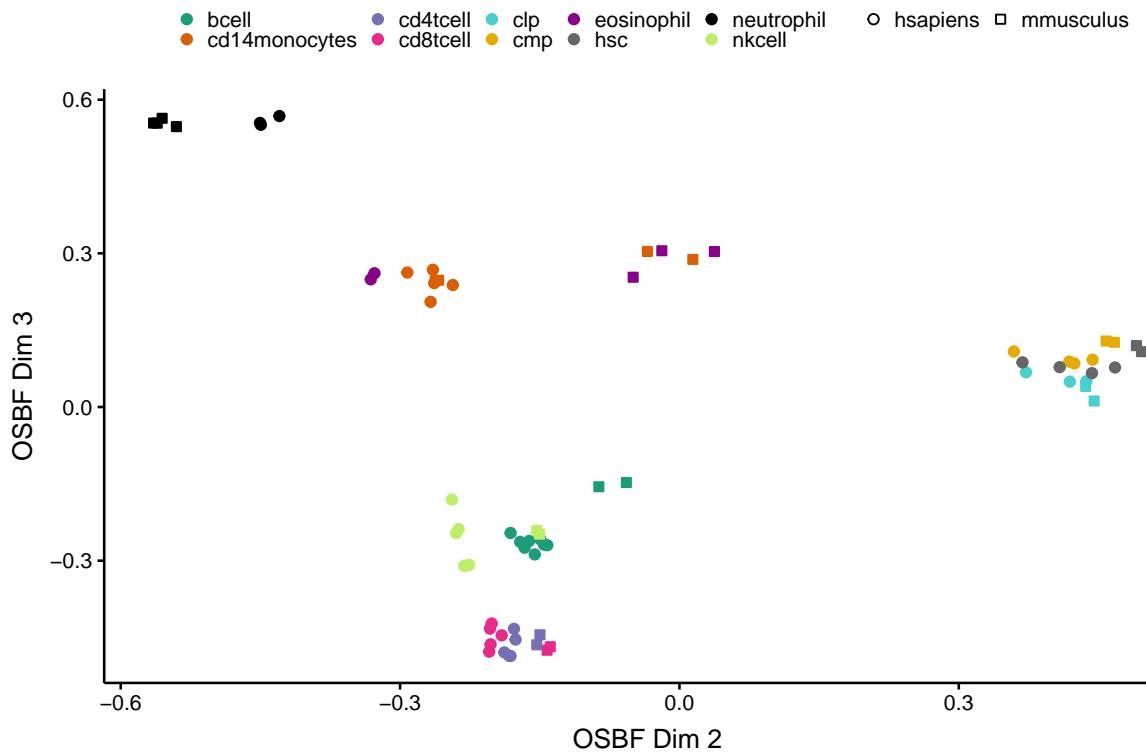
The same with optimized V

```
# 2D plot for Dim1 and Dim2 [With V update]
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())
```

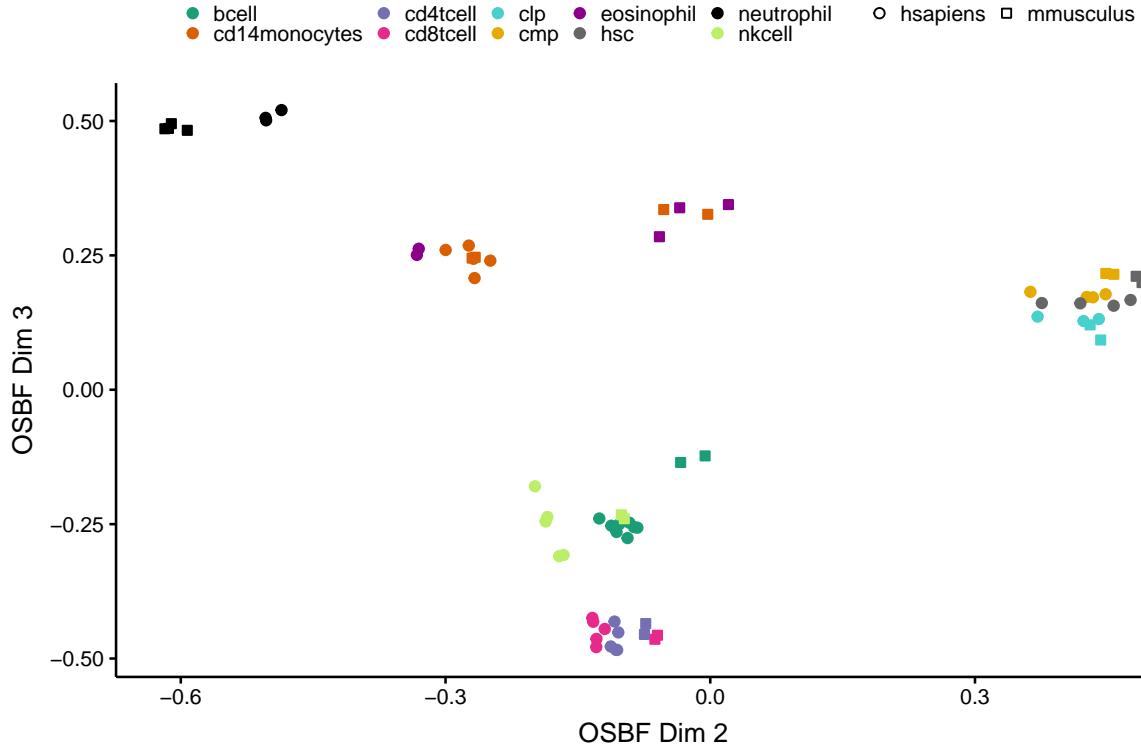


Let us plot the projected libraries in dimensions 2 and 3.

```
# 2D plot for Dim2 and Dim3 [No V update]
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]
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

```

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)
}

```

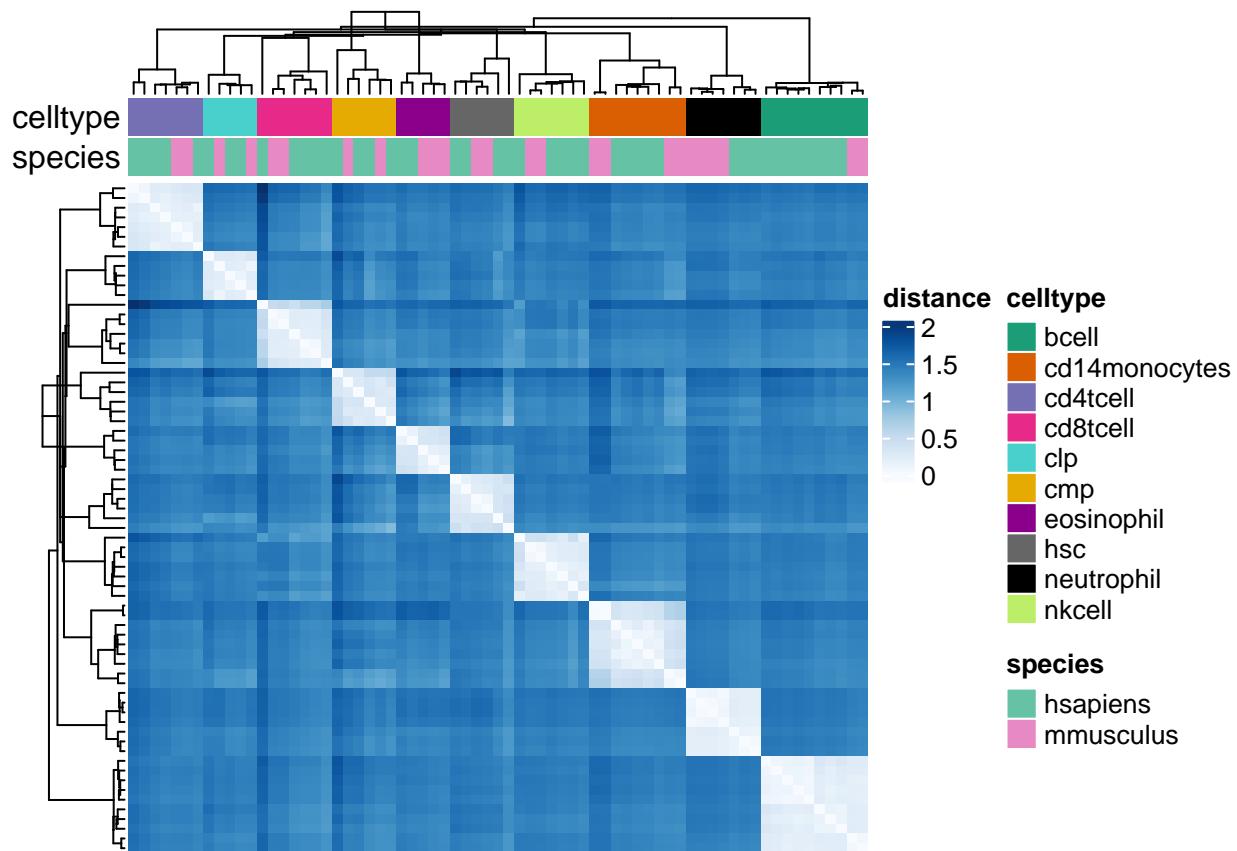
6 Clustering in the common space

```
# 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 = TRUE,
  clustering_method_rows = "centroid",
  cluster_columns = TRUE,
  clustering_method_columns = "centroid",
  top_annotation = ht, col = morecolors(50),
  show_row_names = FALSE, show_column_names = FALSE,
  name = "distance")
myheatmap
```

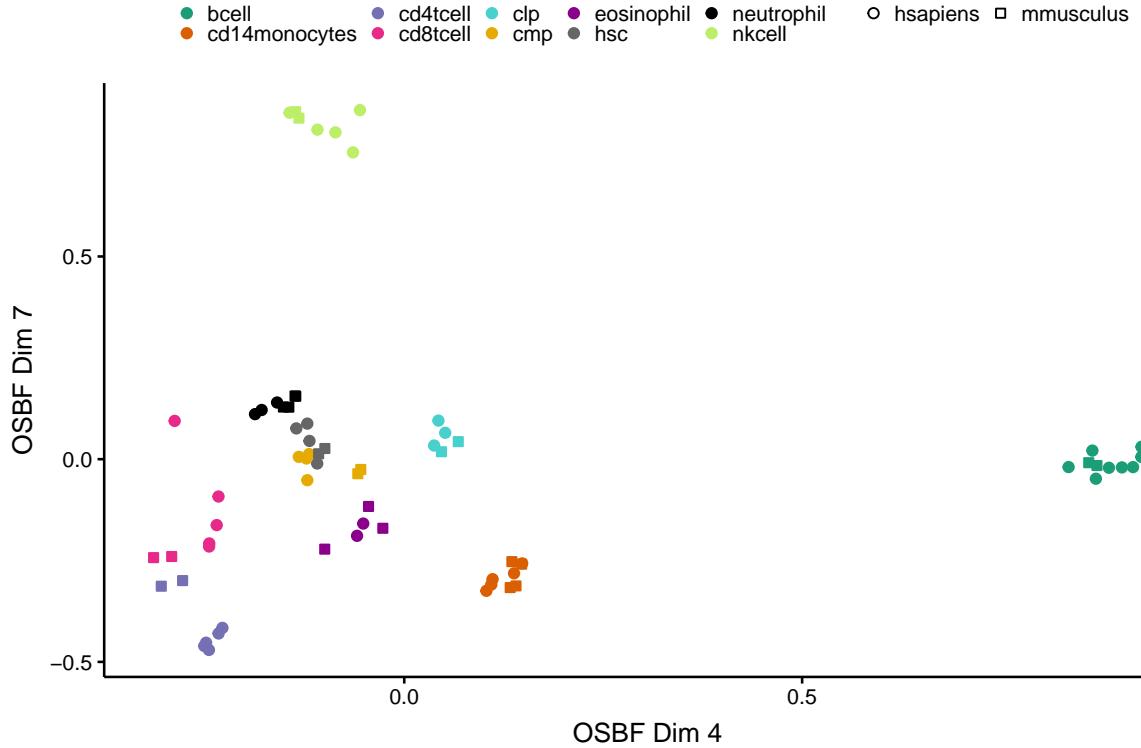


Gene expression profiles cluster by cell type independent of the species of origin.

7 Explore different dimensions

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

```
# 2D plot for Dim2 and Dim3 [With V update]
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 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.

7.1 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 = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                 "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
```

```

head(expr1)

##           tissue_zscore      Tau       coef
## ENSG000000000003 -0.5999301 0.8510679 -0.0027742236
## ENSG00000000419  0.5454641 0.2015887 -0.0009721712
## ENSG00000000457  0.3170688 0.3593960 -0.0031330393
## ENSG00000000460  0.1085645 0.5322674 -0.0007931068
## ENSG00000000938  0.1805992 0.5880772  0.0100492758
## ENSG00000000971 -0.7199144 0.8117708 -0.0137362448

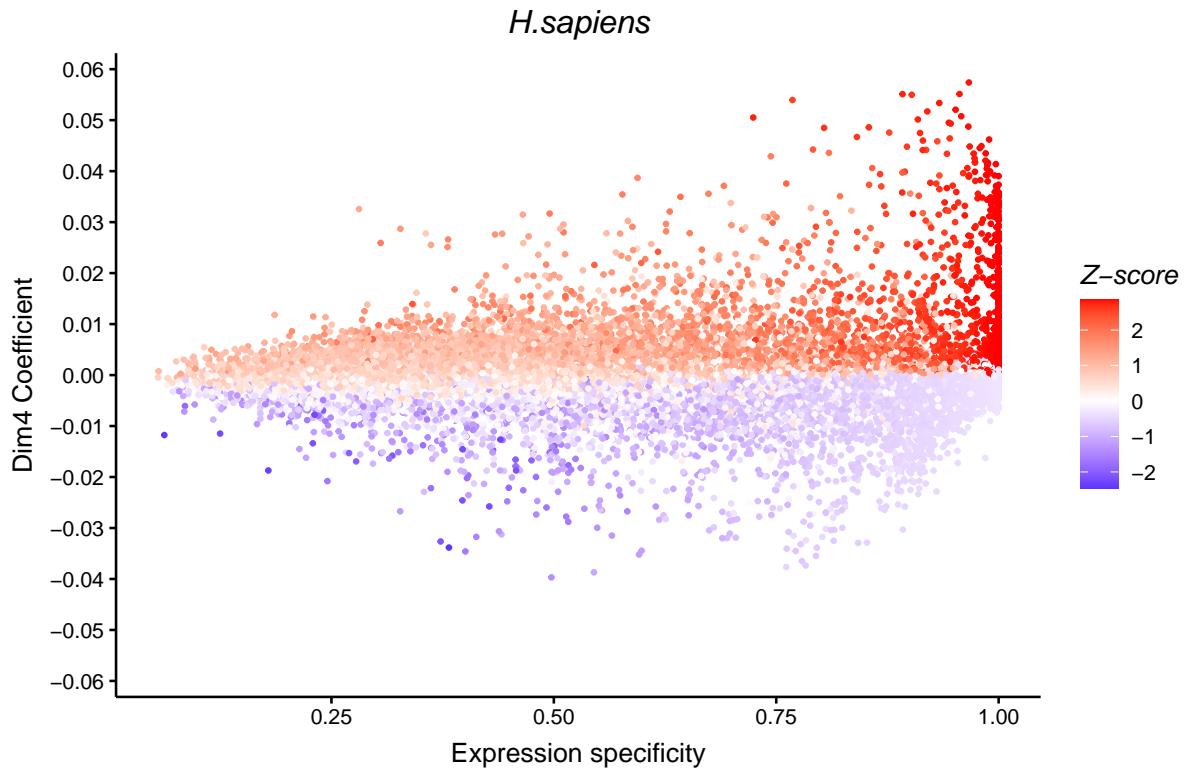
```

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.

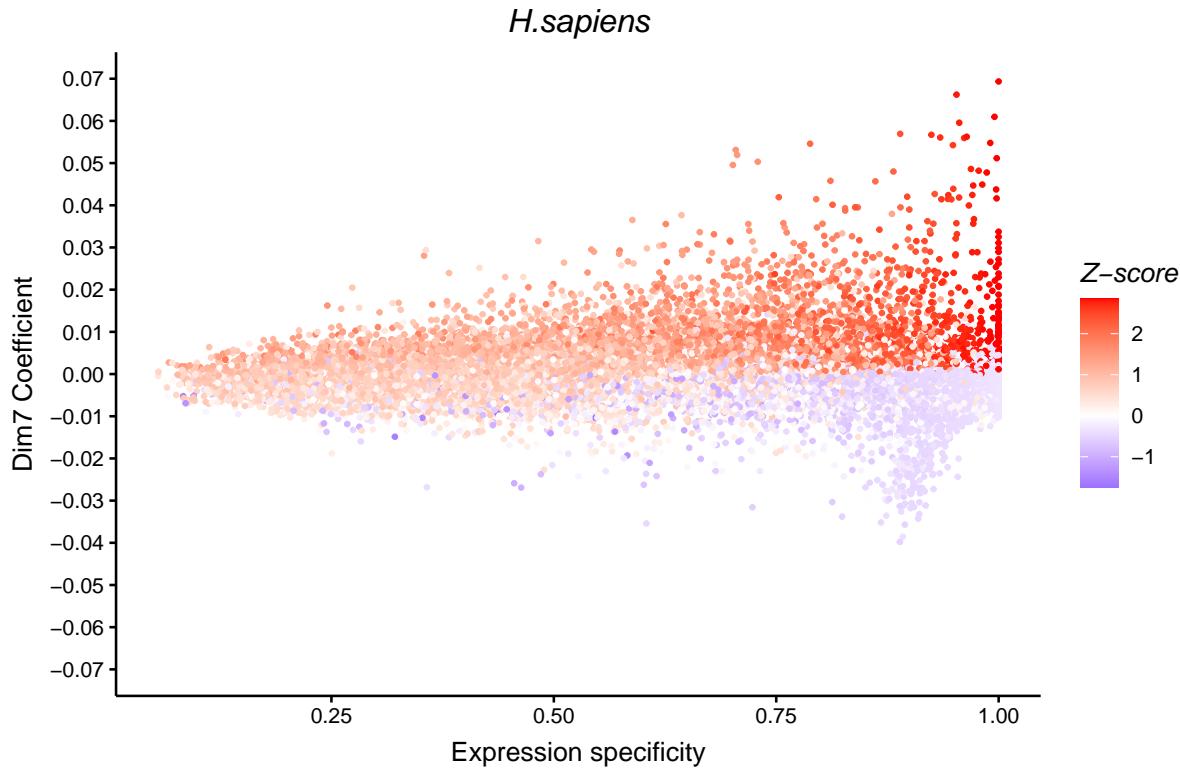
```

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 = TRUE]
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



7.2 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$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
if (sel_sign == "neg") {
  cat("\n selecting negative loadings")
  expr1_selsign <- expr1[expr1$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 9397 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
## 683  G0:0002250       56     208 adaptive immune response      BP    0
## 723  G0:0002376       57     378   immune system process      BP    0
## 724  G0:0002377       23      40 immunoglobulin production      BP    0
## 2260 G0:0006508       34     272           proteolysis      BP    0
## 2474 G0:0006898       34      92 receptor-mediated endocytosis      BP    0
## 2481 G0:0006910       32      37   phagocytosis, recognition      BP    0
##          ratio
## 683  0.2692
## 723  0.1508

```

```

## 724 0.5750
## 2260 0.1250
## 2474 0.3696
## 2481 0.8649

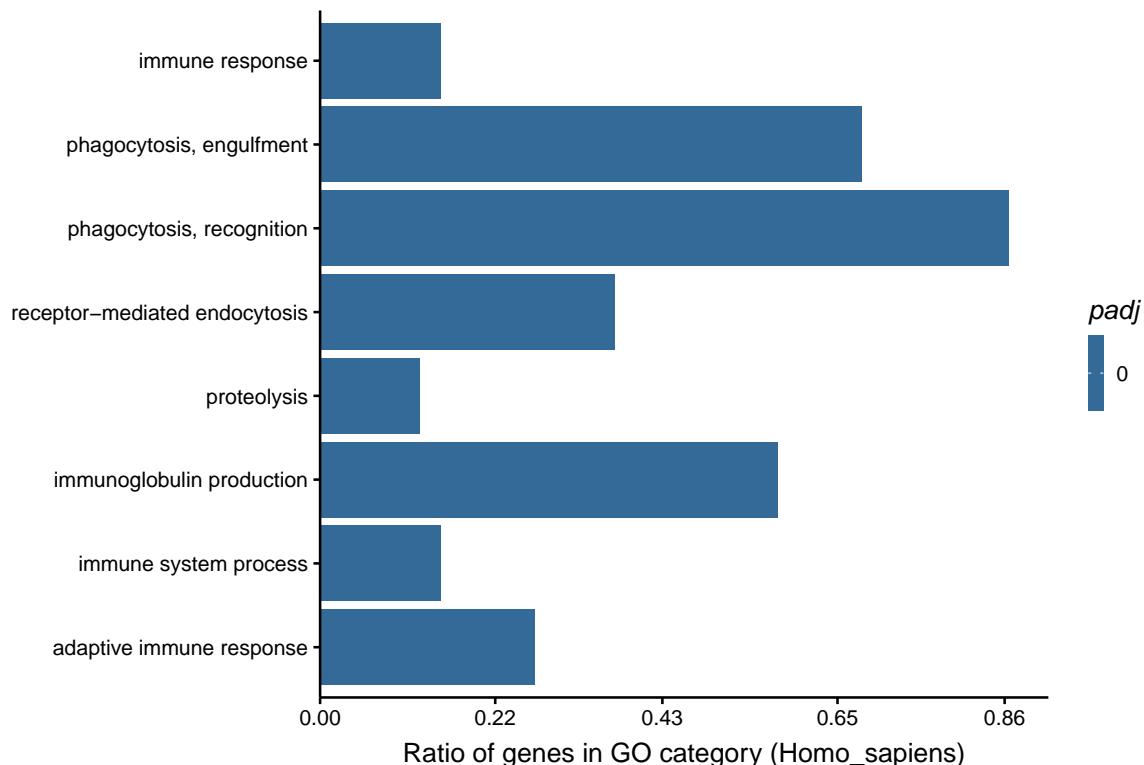
```

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$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(getSpeciesShortName(species), "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
if (sel_sign == "neg") {
  cat("\n selecting negative loadings")
  expr1_selsign <- expr1[expr1$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 7470 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
## 8994 GO:0050776           16      94 regulation of immune response
## 2502 GO:0006968            7      25 cellular defense response
## 4560 GO:0019835           4       5          cytolysis
## 2594 GO:0007165           19     577          signal transduction
## 2595 GO:0007166            7    105 cell surface receptor signaling pathway
## 2496 GO:0006954           8     154          inflammatory response

```

```

##      ontology      padj   ratio
## 8994      BP 1.066539e-13 0.1702
## 2502      BP 1.199485e-05 0.2800
## 4560      BP 9.141476e-05 0.8000
## 2594      BP 2.964592e-03 0.0329
## 2595      BP 9.515304e-02 0.0667
## 2496      BP 9.515304e-02 0.0519

```

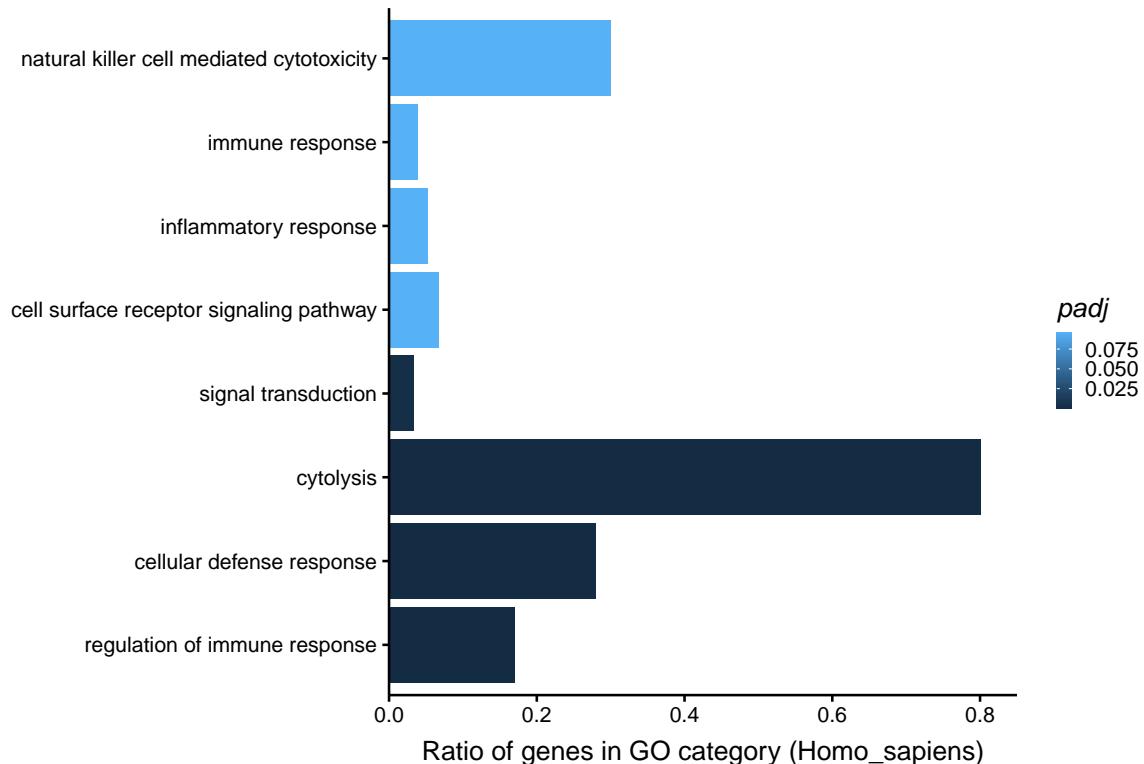
Barplot with top human GO terms and their p-value.

```

# 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

```



8 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.

8.1 Create shuffled counts to generate null

We will create mean profiles based on shuffled reads. Then we will apply TPM normalization for these counts using `normalizeTPM` function from the SBF package.

```
# 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 = TRUE,
                        sep = "\t", row.names = 1)
  info <- data.table::tstrsplit(colnames(counts), "_")
  metadata <- data.frame(project = info[[1]],
                         species = info[[2]],
                         tissue = info[[3]],
                         gsm = info[[4]],
                         name = colnames(counts),
                         stringsAsFactors = FALSE)
  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 = TRUE, row.names = 1,
                            stringsAsFactors = FALSE)
  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 = TRUE]
  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 <- data.table::tstrsplit(colnames(counts_tpm), "_")
metadata <- data.frame(
  species = info[[1]],
  tissue = info[[2]],
  name = colnames(counts_tpm),
  stringsAsFactors = FALSE)
metadata$key <- paste0(metadata$species, "_", metadata$tissue)
avg_counts_shuff[[sp]] <- calcAvgCounts(counts_tpm, metadata)
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

```

8.2 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")

## Wed Jun 22 04:08:52 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")

```

```

## Wed Jun 22 04:08:54 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
}

```

8.3 Identify b cell specific genes

Now let us find genes with significant loadings in dimension 4. We will use the emphirical null generated from the shuffled counts to get the p-value.

```

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 = TRUE]
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 = TRUE]
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_null$score)
})
head(expr1)

##          tissue_zscore      Tau      coef      score rank pvalue
## ENSG00000211594  2.792820 0.9665629 0.05739404 0.05547495    1      0

```

```

## ENSG00000242472      2.785474 0.9560529 0.05513005 0.05270724    2      0
## ENSG00000211677      2.785243 0.9332910 0.05336801 0.04980788    3      0
## ENSG00000211900      2.641843 0.9023546 0.05498137 0.04961269    4      0
## ENSG00000211595      2.824715 0.9516554 0.05205238 0.04953593    5      0
## ENSG00000211593      2.737777 0.8919587 0.05510664 0.04915285    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     12897     281

```

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 = TRUE, 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	0.05739404	0.05547495	1	0
## ENSG00000242472	2.785474	0.9560529	0.05513005	0.05270724	2	0
## ENSG00000211677	2.785243	0.9332910	0.05336801	0.04980788	3	0
## ENSG00000211900	2.641843	0.9023546	0.05498137	0.04961269	4	0
## ENSG00000211595	2.824715	0.9516554	0.05205238	0.04953593	5	0
## ENSG00000211593	2.737777	0.8919587	0.05510664	0.04915285	6	0
## ENSG00000211949	2.818037	0.9579632	0.05076904	0.04863487	7	0
## ENSG00000237111	2.741113	0.9198911	0.05171625	0.04757332	8	0
## ENSG00000264781	2.818623	0.9660110	0.04874001	0.04708338	9	0
## ENSG00000211679	2.805381	0.9440552	0.04949756	0.04672843	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 the top genes are mostly immunoglobulin genes.

8.4 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$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]
expr1 <- expr[, c(paste0(species_short, "_", sel_tissue),
                  "Tau", "coef")]
colnames(expr1) <- c("tissue_zscore", "Tau", "coef")
# null loadings for the same dimensions
expr_null <- combine_expr_null[[species]]
null_u <- osbf_shuf$u[[species]]
expr_null[["coef"]] <- null_u[, sel_dim, drop = TRUE]
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_null$score)
})

```

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   10323     222

```

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 = TRUE, 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 0.07032826 0.06780361 1 0
## ENSMUSG00000062524 2.845611 0.9980411 0.06200972 0.06188825 2 0
## ENSMUSG00000089727 2.834113 0.9898324 0.05909790 0.05849701 3 0
## ENSMUSG00000033024 2.846050 1.0000000 0.05740477 0.05740477 4 0
## ENSMUSG00000079852 2.832204 0.9878951 0.05676019 0.05607312 5 0
## ENSMUSG00000030325 2.788546 0.9745317 0.05640606 0.05496949 6 0
## ENSMUSG00000072721 2.846050 1.0000000 0.05415138 0.05415138 7 0
## ENSMUSG00000067599 2.844010 0.9937175 0.05291999 0.05258752 8 0
## ENSMUSG00000050241 2.843993 0.9957654 0.05164143 0.05142274 9 0
## ENSMUSG00000043932 2.843056 0.9948936 0.05082270 0.05056318 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
## ENSMUSG00000043932      Klri2       protein_coding

```

We identify common NK cell marker genes

9 Exploring dimension 1 of the common space

Lets plot the U_1 loadings vs expression specificity

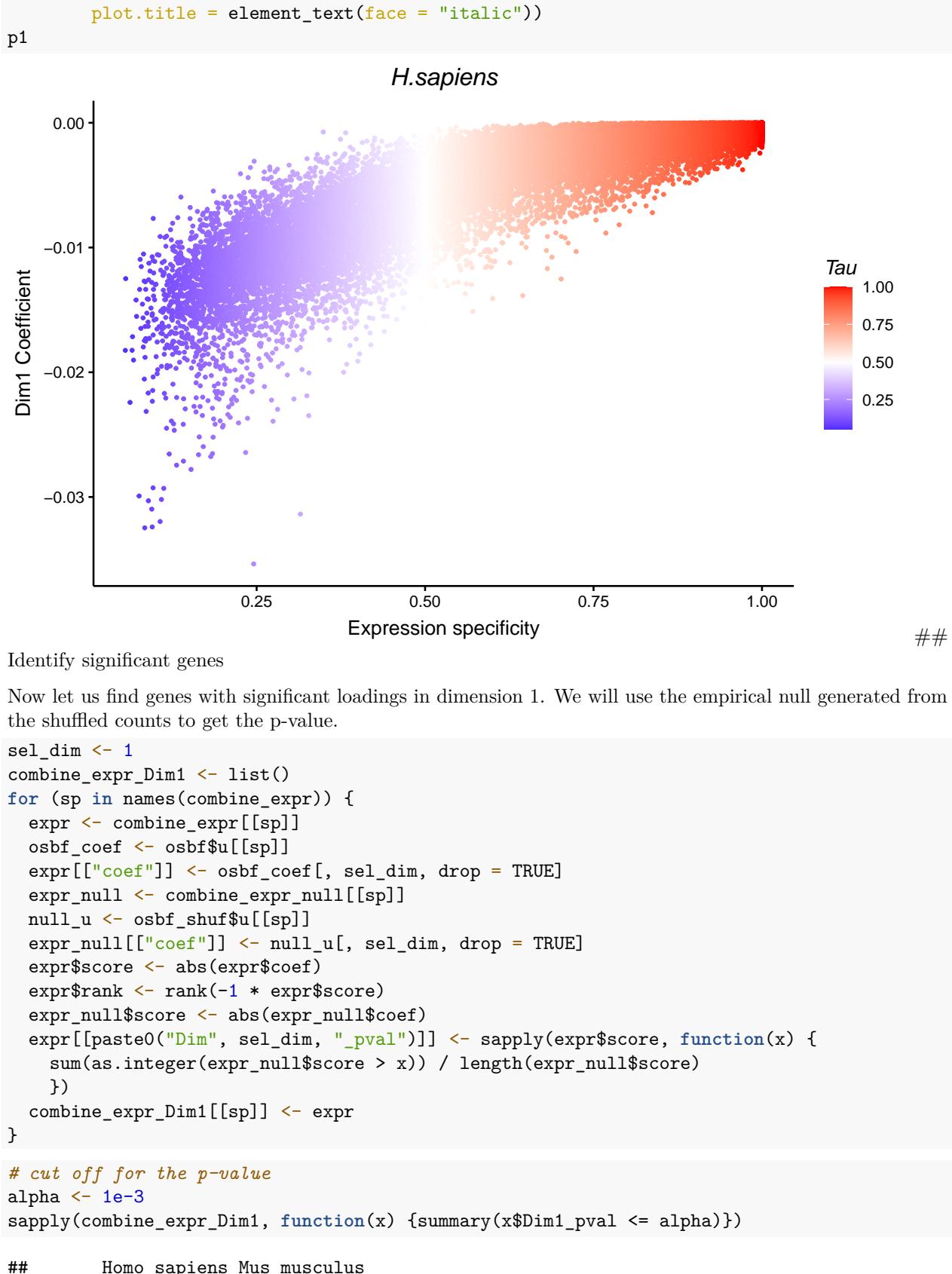
Lets plot for human

```

sel_dim <- 1
species <- "Homo_sapiens"
expr <- combine_expr[[species]]
osbf_coef <- osbf$u[[species]]
expr[["coef"]] <- osbf_coef[, sel_dim, drop = TRUE]

# plot scatter
mid <- 0.5
p1 <- ggplot2::ggplot(expr, aes(x = Tau, y = coef, col = Tau)) +
  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") +
  customTheme() + theme(legend.position = "right",
                        legend.direction = "vertical") +
  labs(title = getScientificName(species), color = "Tau") +
  theme(legend.key.size = unit(0.5, "cm")),

```



```

## Mode "logical"      "logical"
## FALSE "30124"      "20705"
## TRUE  "206"         "146"

```

Now lets find the details of these genes.

- Are these protein coding or not, orthologs?
- How many of these genes are mitochondrial genes?

```

# set the path to the working directory. Change this accordingly
path <- "~/Dropbox/0.Analysis/0.paper/"
# get human mouse orthologs
file <- "allwayOrthologs_hsapiens-mmusculus_ens94.tsv"
hm_orthologs <- read.table(paste0(path, "ensembl94_annotation/"), file),
               header = TRUE, sep = "\t")
Dim1_genes <- list()
for (sp in names(combine_expr_Dim1)) {
  x <- combine_expr_Dim1[[sp]][combine_expr_Dim1[[sp]]$Dim1_pval <= alpha, ]
  x <- x[order(x$rank), c("rank", "score", "Tau", "Dim1_pval"), drop = FALSE]
  # get gene details
  gene_info <- read.table(paste0(path, "ensembl94_annotation/",
                                  getSpeciesShortName(sp),
                                  "_genes_completeinfo.tsv"),
                            sep = "\t", header = TRUE, quote = "\"")
  gene_info <- gene_info[!duplicated(gene_info$ensembl_gene_id), ]
  gene_info <- gene_info[gene_info$ensembl_gene_id %in% row.names(x), ]
  row.names(gene_info) <- gene_info$ensembl_gene_id
  gene_info <- gene_info[row.names(x), ]
  gene_info <- gene_info[, c("ensembl_gene_id", "external_gene_name",
                            "gene_biotype", "chromosome_name")]
  colnames(gene_info) <- c("id", "gene_name", "biotype", "chr")
  orthologs <- hm_orthologs[, getSpeciesShortName(sp), drop = TRUE]
  ortholog_status <- rep(FALSE, nrow(x))
  ortholog_status[row.names(x) %in% orthologs] <- TRUE
  x$orthologs <- as.factor(ortholog_status)
  df <- cbind(gene_info, x)
  row.names(df) <- NULL
  Dim1_genes[[sp]] <- df
}

```

Top Dim1 human genes

```
head(Dim1_genes[["Homo_sapiens"]])
```

```

##           id gene_name      biotype chr rank      score      Tau
## 1 ENSG00000210082  MT-RNR2      Mt_rRNA  MT     1 0.03536681 0.24563693
## 2 ENSG00000198712  MT-C02 protein_coding  MT     2 0.03247328 0.08397371
## 3 ENSG00000198938  MT-C03 protein_coding  MT     3 0.03240265 0.09505659
## 4 ENSG00000198804  MT-C01 protein_coding  MT     4 0.03196621 0.10679948
## 5 ENSG00000211459  MT-RNR1      Mt_rRNA  MT     5 0.03137539 0.31502753
## 6 ENSG00000198886  MT-ND4 protein_coding  MT     6 0.03097495 0.09438006
##   Dim1_pval orthologs
## 1          0    FALSE
## 2          0    TRUE
## 3          0    TRUE
## 4          0    TRUE
## 5          0    FALSE

```

```

## 6          0      TRUE
Top Dim1 mouse genes
head(Dim1_genes[["Mus_musculus"]])

##           id gene_name      biotype chr rank    score      Tau
## 1 ENSMUSG00000064351  mt-Co1 protein_coding MT     1 0.02574092 0.06587759
## 2 ENSMUSG00000037742  Eef1a1 protein_coding 9     2 0.02489218 0.06740533
## 3 ENSMUSG00000060802      B2m protein_coding 2     3 0.02480246 0.04576181
## 4 ENSMUSG00000007892   Rplp1 protein_coding 9     4 0.02446146 0.10895161
## 5 ENSMUSG00000049775   Tmsb4x protein_coding X     5 0.02429125 0.10034603
## 6 ENSMUSG00000064370  mt-Cytb protein_coding MT     6 0.02408973 0.07020361
##   Dim1_pval orthologs
## 1          0      TRUE
## 2          0     FALSE
## 3          0      TRUE
## 4          0     FALSE
## 5          0     FALSE
## 6          0      TRUE

Summary stats of dimension 1 genes
cnames <- c("Dim", "species", "nTotal", "nCoding", "nMt", "nOrthologs")
summary_stats <- data.frame(matrix(NA, nrow = 0, ncol = length(cnames)))
colnames(summary_stats) <- cnames

for (sp in names(Dim1_genes)) {
  df <- Dim1_genes[[sp]]
  stats <- data.frame(matrix(0L, nrow = 1, ncol = length(cnames)))
  colnames(stats) <- cnames
  stats$Dim <- "Dim1"
  stats$species <- getSpeciesShortName(sp)
  stats$nTotal <- nrow(df)
  stats$nCoding <- round(nrow(df[df$biotype == "protein_coding",
                                    , drop = FALSE]) * 100 / nrow(df), 2)
  stats$nMt <- round(nrow(df[tolower(df$chr) == "mt",
                                , drop = FALSE]) * 100 / nrow(df), 2)
  stats$nOrthologs <- round(nrow(df[df$orthologs == TRUE,
                                       , drop = FALSE]) * 100 / nrow(df), 2)
  summary_stats <- rbind(summary_stats, stats)
}
summary_stats

##      Dim species nTotal nCoding nMt nOrthologs
## 1 Dim1 hsapiens    206   96.60  7.77    76.21
## 2 Dim1 mmusculus    146   95.89  5.48    73.97

```

10 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-gnublas/libblas.so.3.9.0
## LAPACK:  /usr/lib/x86_64-linux-gnulapack/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_2.1.1
## [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_0.39
## [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_0.0.13
## [57] devtools_2.4.3            XML_3.99-0.9
## [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_1.48.1    BiocGenerics_0.42.0
## [79] pkgbuild_1.3.1             BiocParallel_1.30.2
## [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_1.36.0            RCurl_1.98-1.6
## [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.