ImmuneREF Documentation

Release 0.5

CRW

Jan 24, 2022

Contents:

immuneREF allows the analysis of repertoire similarity on a one-to-one, one-to-many and many-to-many scale across repertoire features ranging from fully sequence- to fully frequency-dependent ones.

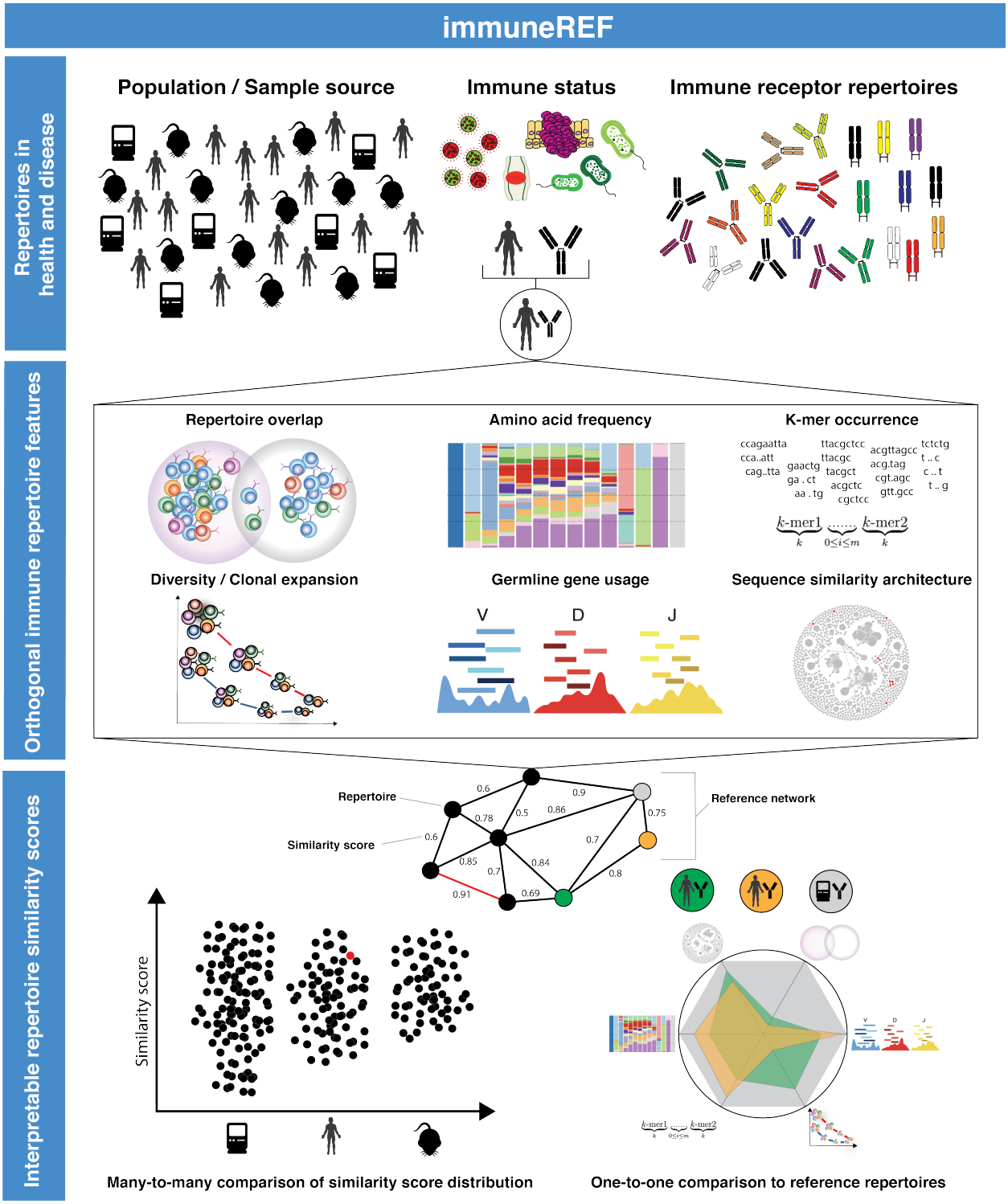


Fig. 1: The immuneREF workflow combines the analysis of a range of features to calculate a multidimensional similarity network of immune repertoires. This enables the identifciation of both, global repertoire similarity (many-to-many, one-to-many) and similarity to a reference repertoires (many-to-one, one-to-one)

CHAPTER 1

Installing immuneREF

1.1 Prerequisites

To be able to run the code, the following prerequisites should be fulfilled:

1. R >= 3.4.0.
2. The following packages need to be installed:
    - ggplot2
    - igraph
    - Biostrings
    - stringdist
    - vegan
    - doParallel
    - foreach
    - dplyr
    - grid
    - kebabs
    - ComplexHeatmap

```
# Check R version
    version[['version.string']]

# Install required R packages hosted on CRAN
    install.packages(c("ggplot2","igraph","stringdist","vegan","doParallel","foreach",
↪"dplyr","grid"))

# Install required R packages hosted on Bioconductor
```

(continues on next page)

(continued from previous page)

```
    #If R version  "4.0""
    if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

    BiocManager::install(c("Biostrings","kebabs","ComplexHeatmap"))

    #If R version < "4.0"
    source("https://bioconductor.org/biocLite.R")
    biocLite(c("Biostrings","kebabs","ComplexHeatmap"))
```

1.2 Install immuneREF

The package can be installed in R (via GitHub):

1. Check if all the prerequisites are fulfilled/installed.
2. Execute the following lines in R:

```
# Install the devtools package
install.packages("devtools")

# Load devtools and install immuneREF from GitHub
library(devtools)
install_github("GreiffLab/immuneREF")

# Test if installation was successful by loading immuneREF
library(immuneREF)
```

CHAPTER 2

Quickstart

2.1 Quickstart

In the `immuneREF_quickstart.R` script (available at https://github.com/GreiffLab/immuneREF), we provide a simple example of the analysis of the included dataset of tutorial repertoires (`tutorial_repertoires`). A more in-depth tutorial is also available (See: *In-depth Tutorial*)

```
library(immuneREF)

# Calculate simularity networks for single features (Step 1 and 2)
similarity_networks <- immuneREF_quickstart(repertoire_list = tutorial_repertoires)

# Calculate network features and plot heatmap of repertoire similarities (condensed
↪network) (Step 4)
network_features <- analyze_similarity_network(similarity_networks[["Condensed"]])

pheatmap::pheatmap(similarity_networks[["Condensed"]],scale='row')
```

CHAPTER 3

In-depth Tutorial

This tutorial summarizes how immuneREF may be used to analyze a set of four simulated immune repertoires.

3.1 Overview

ImmuneREF allows the analysis of repertoire similarity on a one-to-one, one-to-many and many-to-many scale across repertoire features ranging from fully sequence- to fully frequency-dependent ones. For information on package availability and installation refer to *Installing immuneREF*.

3.2 The workflow of the quickstart analysis

The immuneREF repertoire analysis workflow consists of the following steps:

1. Preparation: Ensuring correct format of input repertoires.
2. Analysis of six immune repertoire features for each repertoire.
3. Similarity score calculation for each repertoire pair and feature.
4. Condensing resulting similarity layers into a multi-layer network.

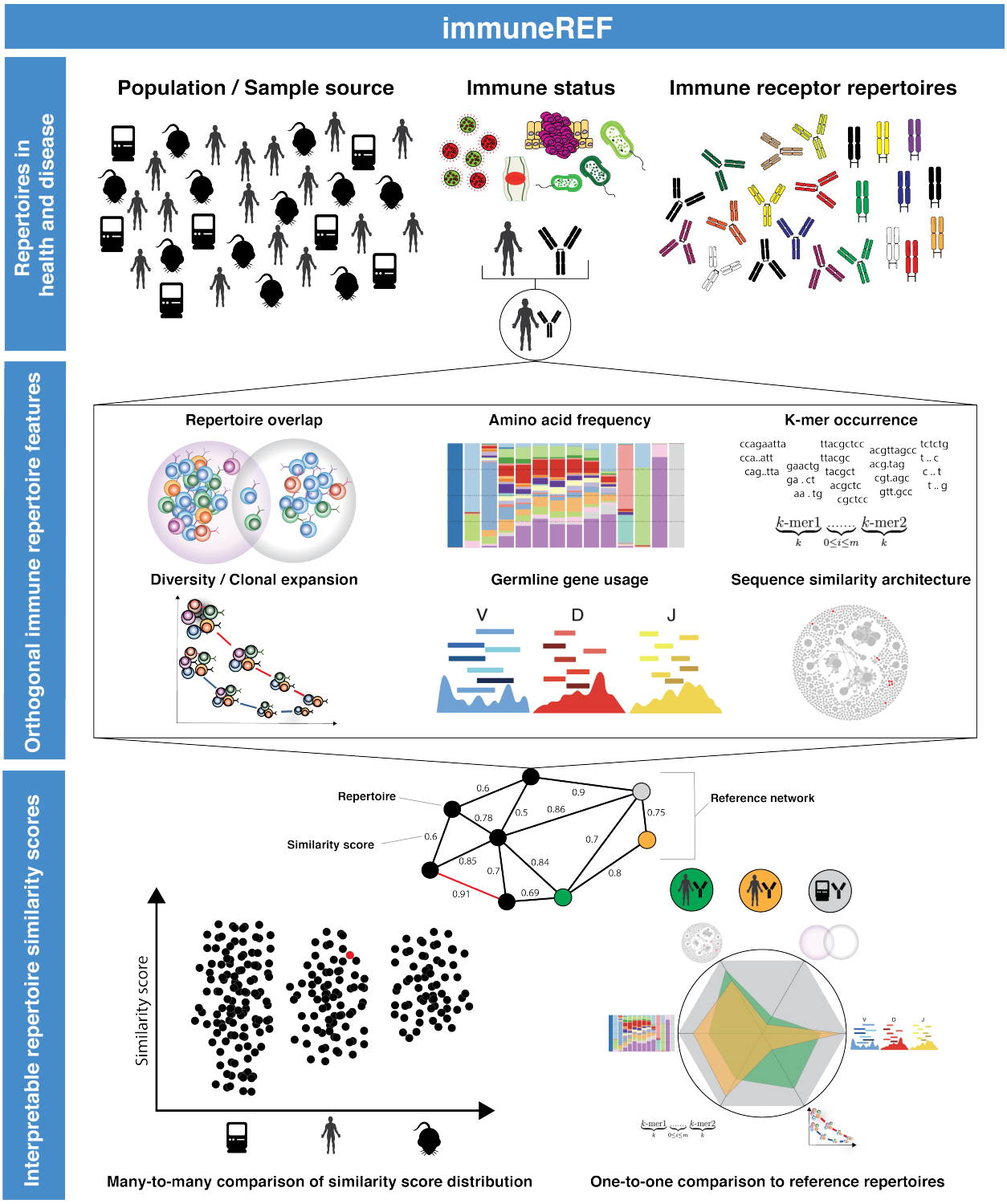


Fig. 1: The immuneREF workflow combines the analysis of a range of features to calculate a multidimensional similarity network of immune repertoires.

5. Analysis of similarity dimensions.

In the `immuneREF_tutorial.R`, we provide a step by step example of a standard immuneREF analysis.

3.3 1. Input format

immuneREF requires your data to be in an R data.frame format with AIRR-standard[1] column names:

- "sequence_aa": Full amino acid VDJ sequence
- "sequence": Full nucleotide VDJ sequence
- "junction_aa": amino acid CDR3 sequence
- "junction": nucleotide CDR3 sequence
- "freqs": occurrence of each sequence in sequencing (column has to sum up to 1)
- "v_call": V-gene annotation (IMGT format, compare naming to provided `list_germline_genes`)
- "d_call": D-gene annotation (IMGT format, compare naming to provided `list_germline_genes`)
- "j_call": J-gene annotation (IMGT format, compare naming to provided `list_germline_genes`)

To test the compatibility of the input format, we provide the `compatibility_check()` function which checks for potential compatibility issues and notifies the user where adjustments might be necessary. Note that this function only provides insight into potential compatibility and does not necessarily indicate that the analysis will fail.

```
compatibility_check(repertoire=tutorial_repertoires[[1]], species="mm", receptor="igh
↪")
```

3.4 2. Analysis of single features

3.4.1 A. Setting up of reference dataframe.

First, we set up a reference dataframe that contains meta-information for each repertoire that is to be analyzed. This dataframe will summarize the datasets and serves as a reference for the analysis of isolated categories downstream.

```
# Load immuneREF package
library(immuneREF)

# Set working directory
setwd(PATH)

# Load datasets (check for compatibility,
# Compatible repertoires have:
#       1) AIRR standard column naming.
#       2) VJ naming compatible with the included list_germline_genes dataset.
#       3) Clonal frequencies that sum up to 1.
list_simulated_repertoires <- tutorial_repertoires

# Set number of repertoires to be analyzed and names for reference dataframe
repertoire_names <- names(list_simulated_repertoires)
```

(continues on next page)

[1] AIRR Community Standardized Representations for Annotated Immune Repertoires. Vander Heiden JA et al., Front Immunol (2018), doi: 10.3389/fimmu.2018.02206, https://github.com/airr-community/airr-standards

(continued from previous page)

```
repertoire_lengths <- sapply(list_simulated_repertoires,nrow)
repertoire_species <- sapply(strsplit(repertoire_names, "\\_"),function(x) x[[1]])
repertoire_receptor <- sapply(strsplit(repertoire_names, "\\_"),function(x) x[[2]])
repertoire_chain <- sapply(strsplit(repertoire_names, "\\_"),function(x) x[[3]])

# Create dataframe containing all relevant metadata on repertoires
input_data_ref<-data.frame(sample_id = repertoire_names,
                           nb_sequences = repertoire_lengths,
                           species = repertoire_species,
                           receptor = repertoire_receptor,
                           chain = repertoire_chain,row.names=c(1:length(repertoire_
↪names)))
```

3.4.2 B. Calculate repertoire overlap

To determine the "Convergence" feature layer, we calculate the repertoire overlap of the full repertoires. In the standard immuneREF workflow, a simple 100% CDR3 amino acid match is used `basis = "CDR3_aa"`. However, the user is free to change the parameter `basis` to:

- CDR3_nt (100% CDR3 nucleotide match)
- VDJ_aa (100% amino acid match of full VDJ sequence),
- VDJ_nt (100% nucleotide sequence match of full VDJ sequence),
- V_CDR3_J_aa (100% CDR3 amino acid sequence match and same v_call and j_call)
- V_CDR3_J_nt (100% CDR3 nucleotide match and same v_call and j_call)

```
# Calculate overlap layer on full datasets for Convergence layer
overlap_layer<-repertoire_overlap(list_simulated_repertoires,basis="CDR3_aa")
```

The resulting `overlap_layer` is a symmetrical $n \times n$ matrix, where $n = number\ of\ repertoires$. Each entry represents the overlap between a repertoire pair. For this layer, the user is free to choose other overlap measures (e.g., Morisita Horn) as long as the result is a nxn matrix with similarity scores in the range [0,1].

3.4.3 C. Subsampling of repertoires

Since multiple immuneREF steps contain computationally intensive calculations, users are encouraged to subsample them to a user-defined `subsample_size` (at least for the calculation of the Architecture and k-mer occurrence layers). A repertoire size of 10'000 sequences is suggested. Subsampling may be performed in one of two ways:

- Picking the x top clones (`random = FALSE`)
- Randomly (x random rows are chosen for the analysis) (`random = TRUE`)

```
# Subsample for the ones that are not 10000
subsample_size<-10000

list_simulated_repertoires<-subset_input_repertoires(list_repertoires=list_simulated_
↪repertoires,
                           subset_size=subsample_size,
                           random=FALSE)
```

3.4.4 D. Calculation of the remaining 5 features

The prepared list of repertoires is analyzed using the `calc_characteristics()` function and returns a list containing the extracted features for each layer.

```
# Calculate all features for each repertoire
repertoires_analyzed<-list()
for(i in 1:length(list_simulated_repertoires)){
  repertoires_analyzed[[repertoire_names[i]]]<-calc_characteristics(
                                repertoire_df=list_simulated_repertoires[[i]],
                                species=repertoire_species[i],
                                receptor=repertoire_receptor[i],
                                chain=repertoire_chain[i],
                                identifier_rep=repertoire_names[i])
}
```

Parallelization: For a larger number of repertoires, it is suggested to parallelize this step. Parallelization may for example be achieved using the R packages foreach, doParallel as shown in the code example below.

```
library(foreach)
library(doParallel)
registerDoParallel(10) #register multicore backend

# Calculate all features for each repertoire in parallel
repertoires_analyzed<-foreach(i=1:length(list_simulated_repertoires)) %dopar% {#
  repertoires_analyzed_loop<-calc_characteristics(
                                repertoire_df=list_simulated_repertoires[[i]],
                                species=repertoire_species[i],
                                receptor=repertoire_receptor[i],
                                chain=repertoire_chain[i],
                                identifier_rep=repertoire_names[i])

  return(repertoires_analyzed_loop)
}
names(repertoires_analyzed)<-repertoire_names
```

3.5 3. Similarity score calculation

Having analyzed the repertoire features, the similarity scores may be calculated for each layer. This is done using the `calculate_similarities()` function for all layers except the `overlap_layer` which is already in a similarity matrix format. Further details on the similarity calculation are given in a dedicated chapter (See: *Determining repertoire similarities*). The required input is:

- repertoires_analyzed: The output of the previous section
- vdj_vj_weights: For the germline usage similarity calculation the user can choose which gene usage information to take into account: c(V, D, J, VJ). Default is set to V and J usage (i.e., c(1,0,1,0))
- convergence: Here the user can choose, between overlap (default) and immunosignature layer.
- overlap_layer: In case overlap is used as a convergence measure, the calculated overlap layer is introduced here
- cor_methods: A named vector of length six containing information on which correlation measure should be used (Default: Pearson for all layers)

```
# Calculate similarities between repertoire for each layer
list_single_layers<-calculate_similarities(repertoires_analyzed=repertoires_analyzed,
↪overlap_layer=overlap_layer)
```

This step can again be **parallelized** using the `calculate_similarities_parallel()` function:

```
# Calculate similarities in parallel per layer.
list_single_layers<-calculate_similarities_parallel(repertoires_analyzed,overlap_
↪layer)
```

3.6 4. Condensing layers into multi-layer network

After having calculated the similarity relationships between repertoires for each layer, the layers may be combined into a multi-layer network by condensing layers. The default method (standard) takes a weighted mean of the similarity scores for each repertoire pair (future versions will include additional methods). The layer weights are determined by the user via the `weights` parameter. (See also: *Multi-layer analysis*)

```
# Calculate condensed network (here equal weights for each layer)
cormat <- condense_layers(list_single_layers,
    weights = c(1,1,1,1,1,1),
    method = "standard")
```

3.7 5. Visualization of results

The resulting immuneREF layers can be analyzed using various tools provided in the package and described in the manuscript. These include:

- Drawing clustered heatmaps for each layer and the condensed network
- Determine network features of similarity layers
- Analyze global similarity score distribution (many-to-many)
- Identify most and least similar repertoires per category via local similarity
- Six-dimensional many-to-one comparison of repertoires to reference repertoires.
- Classical repertoire analysis of repertoires based on the analysis in the repertoire feature extraction (*D. Calculation of the remaining 5 features*).

A detailed overview of the output produced by the tutorial code below is found at *Analysis of immuneREF output*

```
###
# Draw heatmap of immuneREF layers
###

# Create folder to save figures in
dir.create("figures")

# Make list of all layers you want to plot heatmaps for
list_all_layers <- list_single_layers
list_all_layers[["Condensed"]] <- cormat
```

(continues on next page)

(continued from previous page)

```
# Prepare list with heatmap annotations containing categories and colors
annotation_list<-list()
annotation_list[["categories"]]<-data.frame(Species=input_data_ref$species,
                                            Receptor = input_data_ref$receptor)

annotation_list[["colors"]]<-list(Species=c(mm='#ffffbf',hs='#fc8d59'),
                                  Receptor=c(ig='#91bfdb'))


# For each entry (immuneREF layer) plot a heatmap
print_heatmap_sims(list_similarity_matrices=list_all_layers,
                   annotation_list=annotation_list,
                   path_figure="figures")


## Bonus: calculate network features of condensed immuneREF layer
network_features <- analyze_similarity_network(cormat)

###
# Draw Global Similarity Plots of immuneREF layers
###

# Define relevant subsets for splits
categories_list<-list()
categories_list[["categories"]]<-input_data_ref
categories_list[["color"]]<-c("white",'#91bfdb','#ffffbf')
categories_list[["subset"]]<-"species"

#Plot global similarity
print_global_similarity(list_similarity_matrices=list_all_layers,
                        categories_list = categories_list,
                        path_figure="figures")

###
# Plot local similarity per category and identify max and min locally similar␣
↪repertoires
##
max_min_reps<-print_local_similarity(list_similarity_matrices=list_all_layers,
                        categories_list = categories_list,
                        path_figure="figures")


###
# Radar plot to visualize similarity across all 6 layers
##
radar_list<-list()
radar_list[["mm_ig_h_2_0__0_0_0_A"]]<-repertoires_analyzed[["mm_ig_h_2_0__0_0_0_A"]]
radar_list[["mm_ig_h_4_0__0_0_0_A"]]<-repertoires_analyzed[["mm_ig_h_4_0__0_0_0_A"]]
radar_list[["hs_ig_h_2_0__0_0_0_A"]]<-repertoires_analyzed[["hs_ig_h_2_0__0_0_0_A"]]
radar_list[["hs_ig_h_4_0__0_0_0_A"]]<-repertoires_analyzed[["hs_ig_h_4_0__0_0_0_A"]]

comparison_list<-list(roi=names(radar_list),
  roi_names=c(
    "Murine A",
    "Murine B",
    "Human A",
    "Human B"),
```

(continues on next page)

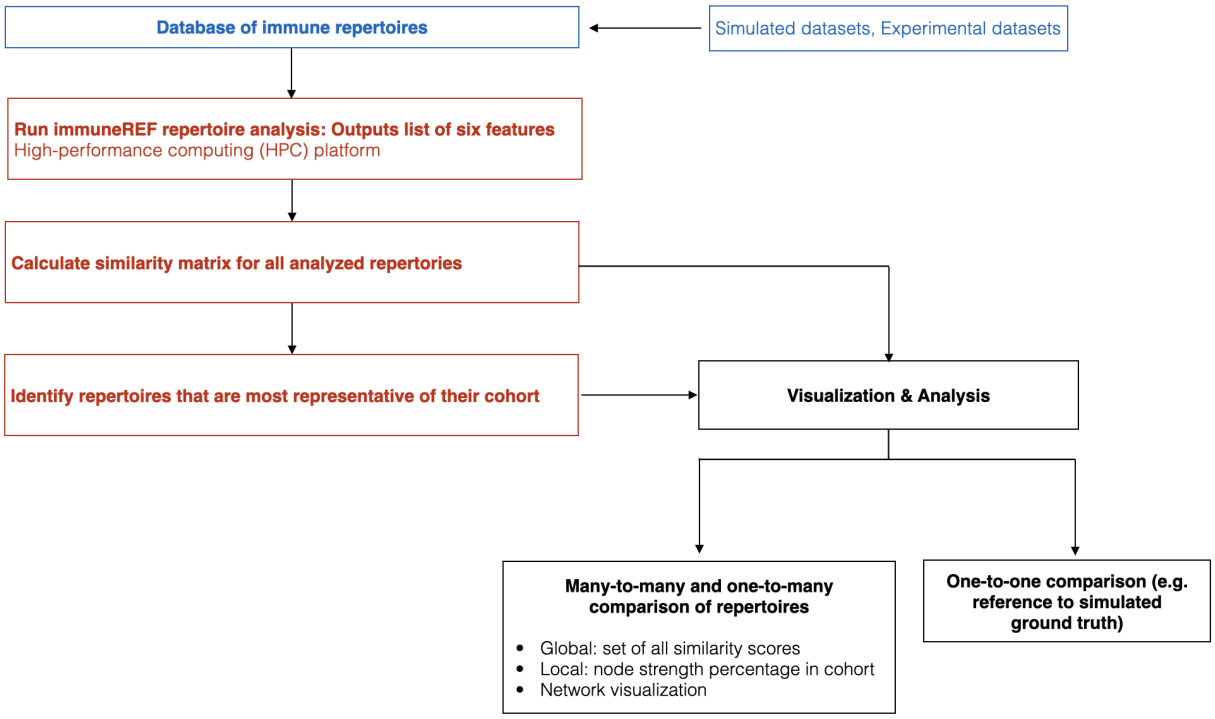
(continued from previous page)

```
  ref="mm_ig_h_2_0__0_0_0_A",
  plot_names=c("Murine A", "Murine B","Human A","Human B"),
  colors=c("grey","blue",'red',"green"))


print_repertoire_radar(list_similarity_matrices=list_single_layers,
  to_compare=comparison_list,
  path_figure="figures",
  name_plot="tutorial")


####
# Classical repertoire analysis of maximally and minimally similar repertoires per␣
↪category
####

# Print classic repertoires comparing max and min locally similar plots for:
# Simulated murine igh repertoires
mm_igh<-list()
#add max locally similar repertoire
mm_igh[["mm_ig_h_2_0__0_0_0_A"]]<-repertoires_analyzed[["mm_ig_h_2_0__0_0_0_A"]]
#add min locally similar repertoire
mm_igh[["mm_ig_h_4_0__0_0_0_A"]]<-repertoires_analyzed[["mm_ig_h_4_0__0_0_0_A"]]

print_repertoire_comparison(list_repertoires=mm_igh,name_plots="mm_igh",aa_freq_
↪length=14,path_figure="figures")


# Simulated human igh repertoires
hs_igh<-list()
#add max locally similar repertoire
hs_igh[["hs_ig_h_2_0__0_0_0_A"]]<-repertoires_analyzed[["hs_ig_h_2_0__0_0_0_A"]]
#add min locally similar repertoire
hs_igh[["hs_ig_h_4_0__0_0_0_A"]]<-repertoires_analyzed[["hs_ig_h_4_0__0_0_0_A"]]

print_repertoire_comparison(list_repertoires=hs_igh,
  name_plots="hs_igh",
  aa_freq_length=17,
  path_figure="figures")
```

3.8 6. Flowchart immuneREF workflow

A step-by-step breakdown of the immuneREF analysis workflow.

3.9 References


Database of immune repertoires
Simulated datasets, Experimental datasets
Run immuneREF repertoire analysis: Outputs list of six features
High-performance computing (HPC) platform
Calculate similarity matrix for all analyzed repertories
Identify repertoires that are most representative of their cohort
Visualization & Analysis
Many-to-many and one-to-many comparison of repertoires
Global: set of all similarity scores
Local: node strength percentage in cohort
Network visualization
One-to-one comparison (e.g. reference to simulated ground truth)

CHAPTER 4

immuneREF feature layers

4.1 The feature layers of immuneREF (Overview)

This section provides an overview over the six immuneREF feature layers:

- *Diversity*
- *Germline gene usage*
- *Positional amino acid frequency*
- *Sequential occurrence of gapped k-mers*
- *Network architecture*
- *Convergence (Repertoire overlap, Immunosignatures)*
- *Gene Expression*
- *Running the feature layer analysis*

4.1.1 Diversity

The diversity profiles were calculated as previously described[1]. Briefly, we calculated the Hill-diversity for alphas 0–10 in steps of 0.1 with alpha = 1 being defined as the Shannon entropy. Each entry in the profile varies between 0 and 1, where higher values indicate an increasingly uniform frequency distribution.

$$^{\alpha}D = (\sum_{i=1}^{n} f^{\alpha})^{\frac{1}{1-\alpha}}$$

[1] A bioinformatic framework for immune repertoire diversity profiling enables detection of immunological status, Greiff et al., Genome Medicine, 2015, https://genomemedicine.biomedcentral.com/articles/10.1186/s13073-015-0169-8

4.1.2 Germline gene usage

The frequency of occurrence of IMGT germline genes[2] in each repertoire was calculated for each repertoire depending on species and immune receptor class (Ig, TCR). The frequency of germline genes (defined by the ImmunoGenetics Database, IMGT)[2] across clones in each repertoire was calculated for each repertoire depending on species and immune receptor class (Ig, TR). The germline gene usage allows insight into deviations from a baseline recombinational likelihood and thereby captures the potential impact of disease, vaccine or other events on the immune state. To determine germline gene usage similarities, germline gene frequencies across clones are extracted for each individual.

4.1.3 Positional amino acid frequency

The positional amino acid frequencies were calculated separately for each CDR3 sequence length present in the repertoire. To decrease the chance of bias by small sets of short and long CDR3 sequences, we limit this analysis to a range of most common lengths (i.e., between 8–20 amino acids)[3]. This range can be adapted via the `calc_characteristics()` parameter `aa_range` which takes in a vector containing the range of lengths (e.g. `aa_range = c(10:22)` to consider AA frequenceis of lengths 10-22)

4.1.4 Sequential occurrence of gapped k-mers

For a given (nucleotide) k-mer size k and maximal gap length m , optimized to maximize information via parameter search, the gapped-pair-k-mer occurrences were counted for all gap sizes <= m. The counts were normalized by the total number of found patterns across all gap sizes such that short range patterns were weighted higher than large gap sizes. Users are free to introduce a new k-mer dictionary to analyze different k-mer lengths and gap sizes using the function `make_kmer_dictionary()`.

```
# Constructs a gapped kmer dictionary used to count k-mers
# The arguments determine the k_nt and gap_size_nt determine the pattern length and␣
↪gap size
dictionary_counts <- make_kmer_dictionary(
            k_nt = 3,
            gap_size_nt = c(0,1,2,3))
```

4.1.5 Network architecture

immuneREF constructs a similarity network for each repertoire[4] where nodes represent CDR3 sequences connected by Levenshtein Distance LD=1 similarity-edges using the igraph package[5]. The resulting networks were analyzed as previously described with respect to four measures: (i) cumulative degree distribution, (ii) mean hub score (Kleinberg hub centrality score), (iii) fraction of unconnected clusters and nodes and (iv) percent of sequences in the largest connected component.

4.1.6 Convergence (Repertoire overlap, Immunosignatures)

To compare repertoire similarity with respect to the presence of immunosignatures, immuneREF provides two options.

[2] IMGT/JunctionAnalysis: IMGT Standardized Analysis of the V-J and V-D-J Junctions of the Rearranged Immunoglobulins (IG) and T Cell Receptors (TR), Giudicelli et al., Cold Spring Harbor Protocols, 2011, http://cshprotocols.cshlp.org/content/2011/6/pdb.prot5634

[3] Systems Analysis Reveals High Genetic and Antigen-Driven Predetermination of Antibody Repertoires throughout B Cell Development, Greiff et al., Cell Reports, 19(7), 2017, https://www.sciencedirect.com/science/article/pii/S221112471730565X

[4] Large-scale network analysis reveals the sequence space architecture of antibody repertoires, Miho et al., Nature Communications, 2019, https://www.nature.com/articles/s41467-019-09278-8

[5] The igraph software package for complex network research, Csardi G, Nepusz T, InterJournal, Complex Systems 1695, 2006, http://igraph.org

The first (and standard) option is to measure the convergence of repertoires by calculating a simple overlap measure according to:

$$\text{overlap} = \frac{\text{A} \cap \text{B}}{\min(|\text{A}|, |\text{B}|)}$$

For this, we provide the `repertoire_overlap()` function which includes the `basis` option that determines based on which sequence the overlap is calculated. The options are: "CDR3_aa","CDR3_nt","VDJ_aa","VDJ_nt","V_CDR3_J_aa" and "V_CDR3_J_nt"

```
# Calculate the overlap across all repertoire pairs. Default basis is CDR3_aa
overlap_layer <- repertoire_overlap(list_simulated_repertoires, basis = "CDR3_aa")
```

The second option integrated in immuneREF is to evaluate the repertoires using pre-trained machine learning models. In this approach, the models are used to predict presence of sequences with given characteristics (public clones, antigen-specificity). For each repertoire, the percentage of positively predicted sequences is then taken as the repertoire immunosignature feature. The base immuneREF package provides an SVM model trained to classify public and private CDR3 sequences (murine Ig)[6] using the KeBABS R-package[7].

Once a larger number of robust pretrained ML models are available future versions of immuneREF will enable their rapid application for immune repertoire diagnostics.

The `calc_characteristics` already calculates a basic immunosignature layer based on the contained SVM model (predicting similarity with respect to presence of of murine Ig public clones). The associated similarity layer can be calculated using:

```
immunosignature_layer <- make_cormat(repertoires_analyzed, weights_overall = c(0,0,0,
↪1,0,0))
```

Additionally, the `calc_characterstics()` function has an additional parameter `models` which allows the user to read-in a list of ML models compatible with the kebabs function''predict()'' for the calculation of the immunosignature feature.

4.1.7 Gene Expression

immuneREF allows integration of immune repertoires and gene expression, which is of high interest to experiments that include both receptor and global transcript sequences (i.e. RNA-seq or repertoire experiments paired with transcriptomics). The analysis of omics data is handled by `calculate_omics_layer()`, which takes a preprocessed and normalized gene expression matrix and sample info data as input.

Firstly, there is a low variation filter to keep the most informative genes from the thousand genes obtained in transcriptomics. Standard deviation (SD) is calculated per gene across samples and all the genes over a certain threshold (default, SD>1) are preserved for subsequent analysis.

Second, users can choose between three correlation-based methods to construct the immuneREF gene expression layer: (1) Pairwise Pearson correlation between samples (method="PC"); (2) Mutual Rank defined as

$$\sqrt{Rank_{AB} \times Rank_{BA}}$$

which is the geometric average between the rank of sample A from sample B and the rank of sample B from sample A after pairwise Pearson correlation (`method="MR"`); and (3) Principal Component Analysis (PCA) is applied to the gene expression matrix, the PCA scores matrix that explains enough variability (default, 80%) is retained and pairwise Pearson correlation between samples is applied (`method="PCA"`).

[6] Learning the High-Dimensional Immunogenomic Features That Predict Public and Private Antibody Repertoires, Greiff et al., Journal of Immunology, 99(8), 2017, http://www.jimmunol.org/content/199/8/2985

[7] KeBABS: an R package for kernel-based analysis of biological sequences. Palme et al., Bioinformatics, 31, 2015, https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btv176

4.1.8 Running the feature layer analysis

immuneREF analyzes the similarity between repertoires across six major features. Before the similarity calculation, each repertoire is analyzed with respect to all of these features. This analysis can be performed all in one or in separate steps per single layer (especially useful for large datasets). Below is an example of the analysis across all features:

```
# Feature calculation

# Extract repertoire sizes and names
repertoire_lengths <- sapply(1:length(tutorial_repertoires),function(x) nrow(tutorial_
↪repertoires[[x]]))
repertoire_names <- sapply(1:length(tutorial_repertoires),function(x) as.
↪character(unique(tutorial_repertoires[[x]]$name_repertoire)))


# Calculate all features for each repertoire
repertoires_analyzed <- list()
for(i in 1:length(tutorial_repertoires)){
  repertoires_analyzed[[repertoire_names[i]]] <- calc_characteristics(
        repertoire_df = tutorial_repertoires[[i]],
        species = strsplit(repertoire_names[i],"_")[[1]][2],
        receptor = strsplit(repertoire_names[i],"_")[[1]][3],
        chain = strsplit(repertoire_names[i],"_")[[1]][4],
        identifier_rep = repertoire_names[i])
}

save(repertoires_analyzed,file = "repertoires_analyzed")
```

4.1.9 Running immuneREF on species not yet included

immuneREF is currently limited to the analysis of human ("hs") and mouse ("mm") repertoires. For the analysis of additional species the extension of `list_germline_genes` needs to be extend to include germline genes of these species. Once the `list_germline_genes` has been extended by information on the species, the species can be specified in the `calc_characteristics()` function via the `species` parameter.

```
#Example process to include new VDJ information for the analysis of IgH repertoires␣
↪of speciesX

#Dataframe containing V-gene information on new species
df_species_VDJ<-list()
df_species_VDJ[["V"]] <- data.frame(gene=c("IGHV1","IGHV2"),allele="01",sequence=c(
↪"gattaca","acattag"),species="speciesX",frequency_uniform=c(0.5,0.5), frequency=c(0.
↪6,0.4))
df_species_VDJ[["D"]] <- data.frame(gene=c("IGHD1","IGHD2"),allele="01",sequence=c(
↪"gat","tac"),species="speciesX",frequency_uniform=c(0.5,0.5), frequency=c(0.6,0.4))
df_species_VDJ[["J"]] <- data.frame(gene=c("IGHJ1","IGHJ2"),allele="01",sequence=c(
↪"aca","tta"),species="speciesX",frequency_uniform=c(0.5,0.5), frequency=c(0.6,0.4))

# Extending list_germline_genes
list_germline_genes[["speciesX"]][["ig"]][["h"]] <- df_species_VDJ
```

4.2 References

CHAPTER 5

Determining repertoire similarities

5.1 Similarity per feature

Having calculated the features of each repertoire (See: *immuneREF feature layers*), the similarity value between each repertoire pair may be calculated. This is done on a per feature basis as laid out in the table below. These measures are the default settings of immuneREF. However, depending on application realm and repertoire characteristics, other methods might be preferred (JS-divergence, Morisita-Horn overlap, etc.) The user is free to deviate from these default settings.

5.2 Table similarity calculation per feature

Table 1: Similarity Calculation

Layer	Similarity calculation
Diversity	Pearson correlation, [1-3].
VDJ Usage	Pearson correlation coefficient was determined for each frequency vectors (Default V-, J-gene usage vectors) [3,4].
Positional AA Frequency	Pearson correlation of positional frequency distributions between repertoires.
Kmer occurrence	Pearson correlation of gapped k-mer occurrence (k = 3, m 3) (Weber et al., 2019).
Network Architecture	Mean of four measures, (i) Pearson correlated cumulative degree distribution, (ii) absolute mean hub score difference (complement), (iii) complement of absolute difference in fraction of unconnected clusters and nodes and (iv) complement of absolute difference of percent of sequences in the largest connected component.
Repertoire overlap	The clonal sequence overlap measure represents the similarity score between repertoires with respect to clonal convergence [3].

5.3 Similarity calculation for all features

Having calculated the features *immuneREF feature layers* of each repertoire, the similarity value between each repertoire pair may be calculated. The easiest way is to use either `calculate_similarities()` or `calculate_similarities_parallel()`. Alternatively, the user may choose to calculate each layer separately using the `make_cormat()` or `make_cormat_parallel()` functions:each repertoire the similarity score between each repertoire pair can be calculated.

```
# Calculate similarities across layers
list_single_layers<-list()

# Calculate diversity similarity
list_single_layers[["Diversity"]] <- make_cormat(
        repertoires_analyzed= repertoires_analyzed,      # list of analyzed␣
↪repertoires containing results for all features
        weights_overall = c(1,0,0,0,0,0),                        # weighting for each␣
↪feature. since we calculate per feature set weight to 1 for diversity
        correlation_method='pearson')                            # correlation method␣
↪is set.

# Calculate AA frequency similarity
list_single_layers[["AAfreq"]] <- make_cormat(repertoires_analyzed, weights_overall =␣
↪c(0,1,0,0,0,0))

# Calculate Architecture similarity (uses pearson correlation)
list_single_layers[["Architecture"]] <- make_cormat(repertoires_analyzed, weights_
↪overall = c(0,0,1,0,0,0), correlation_method='pearson')

# Add repertoire overlap as additional similarity layer
list_single_layers[["repertoire_overlap"]] <- overlap_layer

# Calculate VDJ usage similarity (includes weighting of germlines weights_vdj )
list_single_layers[["VDJ_usage"]] <- make_cormat(repertoires_analyzed, weights_VDJ=c(
↪"V"=1,"D"=0,"J"=1,"VJ"=0), weights_overall = c(0,0,0,0,1,0))

# Calculate k-mer occurrence similarity
list_single_layers[["k-mers"]] <- make_cormat(repertoires_analyzed, weights_overall =␣
↪c(0,0,0,0,0,1))

## Alternatively to overlap the user may choose to calculate Immunosignatures␣
↪similarity
#list_single_layers[["Immunosignatures"]] <- make_cormat(repertoires_analyzed,␣
↪weights_overall = c(0,0,0,1,0,0))
```

5.4 References

[1] The TCR Repertoire Reconstitution in Multiple Sclerosis: Comparing One-Shot and Continuous Immunosuppressive Therapies, Amoriello et al., Frontiers in immunology, 2020, https://www.frontiersin.org/articles/10.3389/fimmu.2020.00559/full

[2] A bioinformatic framework for immune repertoire diversity profiling enables detection of immunological status, Greiff et al., Genome Medicine, 2015, https://genomemedicine.biomedcentral.com/articles/10.1186/s13073-015-0169-8

[3] Learning the High-Dimensional Immunogenomic Features That Predict Public and Private Antibody Repertoires,

Greiff et al., Journal of Immunology, 99(8), 2017, http://www.jimmunol.org/content/199/8/2985

[4] immuneSIM: tunable multi-feature simulation of B- and T-cell receptor repertoires for immunoinformatics benchmarking, Weber et al., Bioinformatics, 2020, https://academic.oup.com/bioinformatics/article/36/11/3594/5802461

CHAPTER 6

Multi-layer analysis

6.1 Condensing multiple layers

The immuneREF package provides the `condense_layer` function to combine single layers into a multi-layer network. It allows the user to set weights for the different layers using the `weight` option, thereby enabling the analysis of repertoire datasets based on problem-specific feature combinations. Additionally, immuneREF has a method option, by which method the multi-layer network is calculated. The default method is 'standard', which takes a weighted mean across layers. Other methods, such as majority voting (that is conversion into a boolean network where edges are drawn if they are above a certain weihgt threshold in a majority of layers), will be added in the future.

```
cormat <- condense_layers(list_single_layers = list_single_layers,
                weights = c(1,1,1,1,1,1),
                method = "standard")
```

CHAPTER 7

Analysis of immuneREF output

7.1 Analyzing Similarity Networks

Be it a single-layer or multi-layer similarity network, the analysis of the network structure is a central goal of immuneREF. Here, we provide an overview of several approaches to analyze the immuneREF similarity networks, including

- Draw clustered heatmaps for each layer and the condensed network
- Determine network features of similarity layers
- Analyze global similarity score distribution (many-to-many)
- Identify most and least similar repertoires per category via local similarity
- Six-dimensional many-to-one comparison of repertoires to reference repertoires
- Classical repertoire analysis of repertoires based on the immuneREF feature extraction.

7.1.1 A. Heatmaps of similarity layers

Representing the similarity layers by a hierarchically clustered Heatmap (R-package ComplexHeatmap[1]) allows easy to interpret analysis of repertoire similarity and category groupings.

```
###
# Draw heatmap of immuneREF layers
###

# Prepare list of all layers you want to plot heatmaps for (here 6 single layers and␣
↪1 condensed layer)
list_all_layers <- list_single_layers
list_all_layers[["Condensed"]] <- cormat
```

(continues on next page)

[1] Gu Z, Eils R, Schlesner M (2016). "Complex heatmaps reveal patterns and correlations in multidimensional genomic data." Bioinformatics., 10.18129/B9.bioc.ComplexHeatmap

(continued from previous page)

```
# Prepare list with heatmap annotations containing categories and colors
annotation_list<-list()
annotation_list[["categories"]]<-data.frame(Species=input_data_ref$species,
                                            Receptor = input_data_ref$receptor)

annotation_list[["colors"]]<-list(Species=c(mm='#ffffbf',hs='#fc8d59'),
                                  Receptor=c(ig='#91bfdb'))

# For each entry (immuneREF layer) plot a heatmap (path_figure folder needs to exist␣
↪already)
print_heatmap_sims(list_similarity_matrices=list_all_layers,
                   annotation_list=annotation_list,
                   path_figure="figures")
```

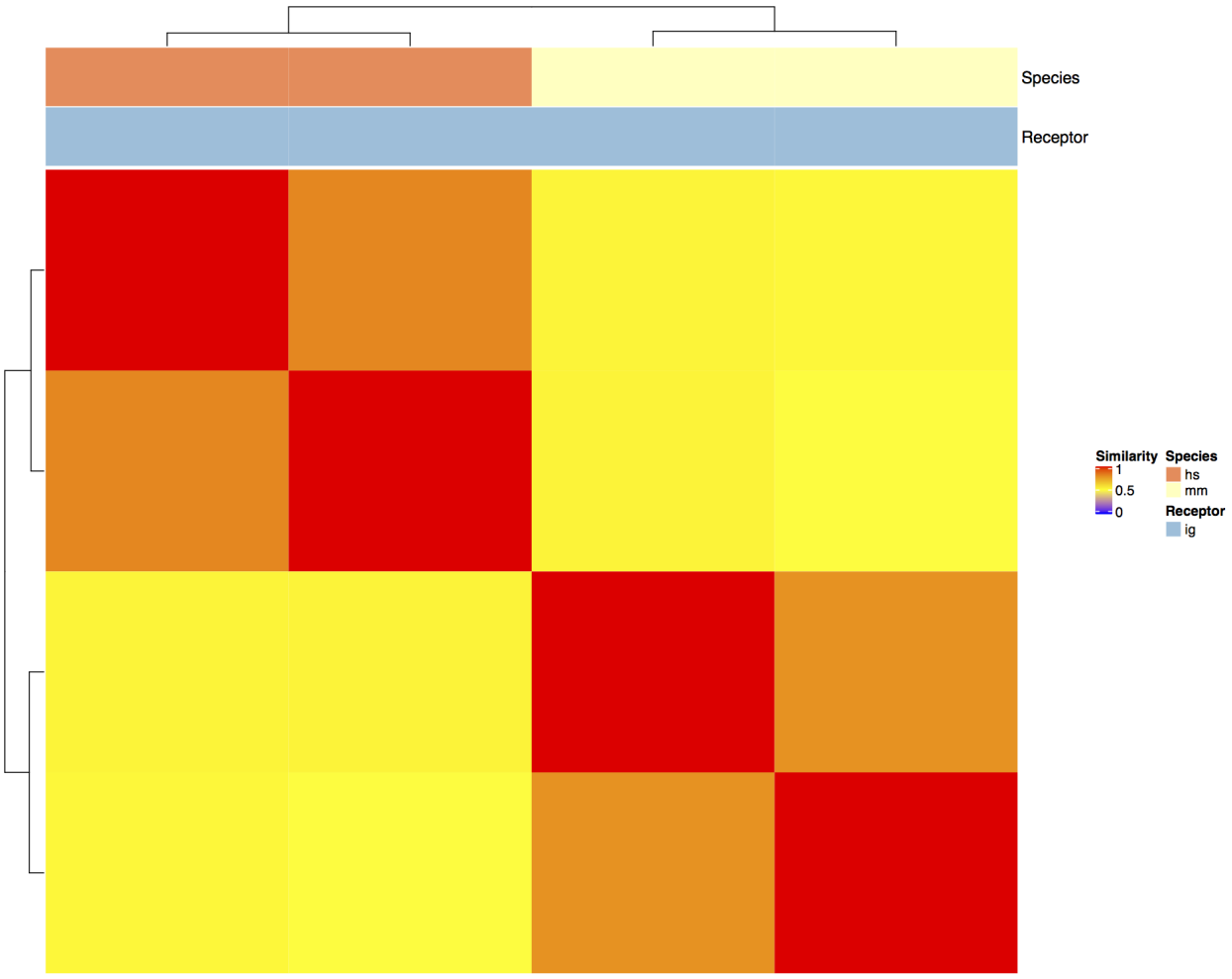


Fig. 1: Visualizing similarity scores across network in a hierarchically clustered heatmap

7.1.2 B. Global similarity scores

The global similarity represents the set of all similarity scores (i.e., weighted edges) in a given repertoire similarity network (many-to-many). Global similarity may be analyzed across the entire network (all repertoires) or on subgraphs of repertoires of a given category.

$$\text{global similarity}_{\text{network}} = \bigcup_{category} similarity\ scores_{repertoire}$$

For easier comparability of overall and per category similarity score distribution, immuneREF visualizes the data using violin plots.

```
# Define how the data should be subset into categories for plotting.
categories_list<-list()
categories_list[["categories"]]<-input_data_ref
categories_list[["color"]]<-c("white",'#91bfdb','#ffffbf')
categories_list[["subset"]]<-"species"

# Plot global similarity
print_global_similarity(list_similarity_matrices=list_all_layers,
                        categories_list = categories_list,
                        path_figure="figures")
```

7.1.3 C. Local similarity scores

To determine how representative single nodes we developed local similarity based on the strength of network nodes. It is calculated by dividing the node strength given node (sum of edge weights of the adjacent edges of a node) by the sum of all node strengths in the subgraph of the category of interest (for example the sum of node strengths of the subgraph of all human repertoires.)

$$\text{local similarity}_{\text{repertoire}} = \frac{Node\ Strength_{repertoire}}{\sum_{category} Node\ Strength}$$

Local similarity gives insight into which repertoires most similar to repertoires of the same condition/species/category. This allows the identification of condition-representative repertoires.

```
###
# Plot local similarity per category and identify max and min locally similar␣
↪repertoires
##

max_min_reps<-print_local_similarity(list_similarity_matrices=list_all_layers,
                        categories_list = categories_list,
                        path_figure="figures")
```

7.1.4 D. Many-to-one analysis across 6 features

Radar plots enable the visualization of similarity of one, or a group of repertoires to a reference repertoire across 6 features.

```
###
# Radar plot to visualize similarity across all 6 feature layers
##
radar_list<-list()
```

(continues on next page)

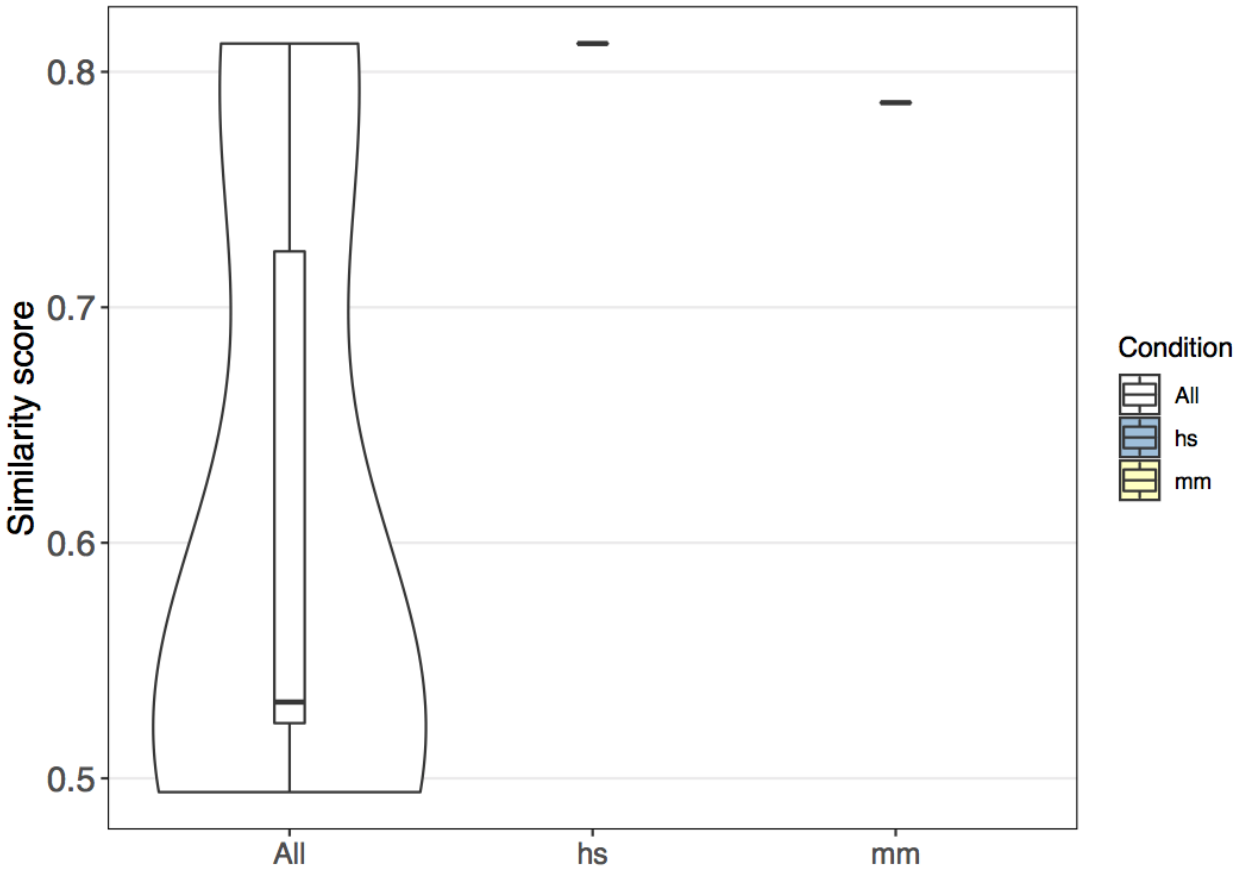


Fig. 2: Comparing similarity scores across network and by species.

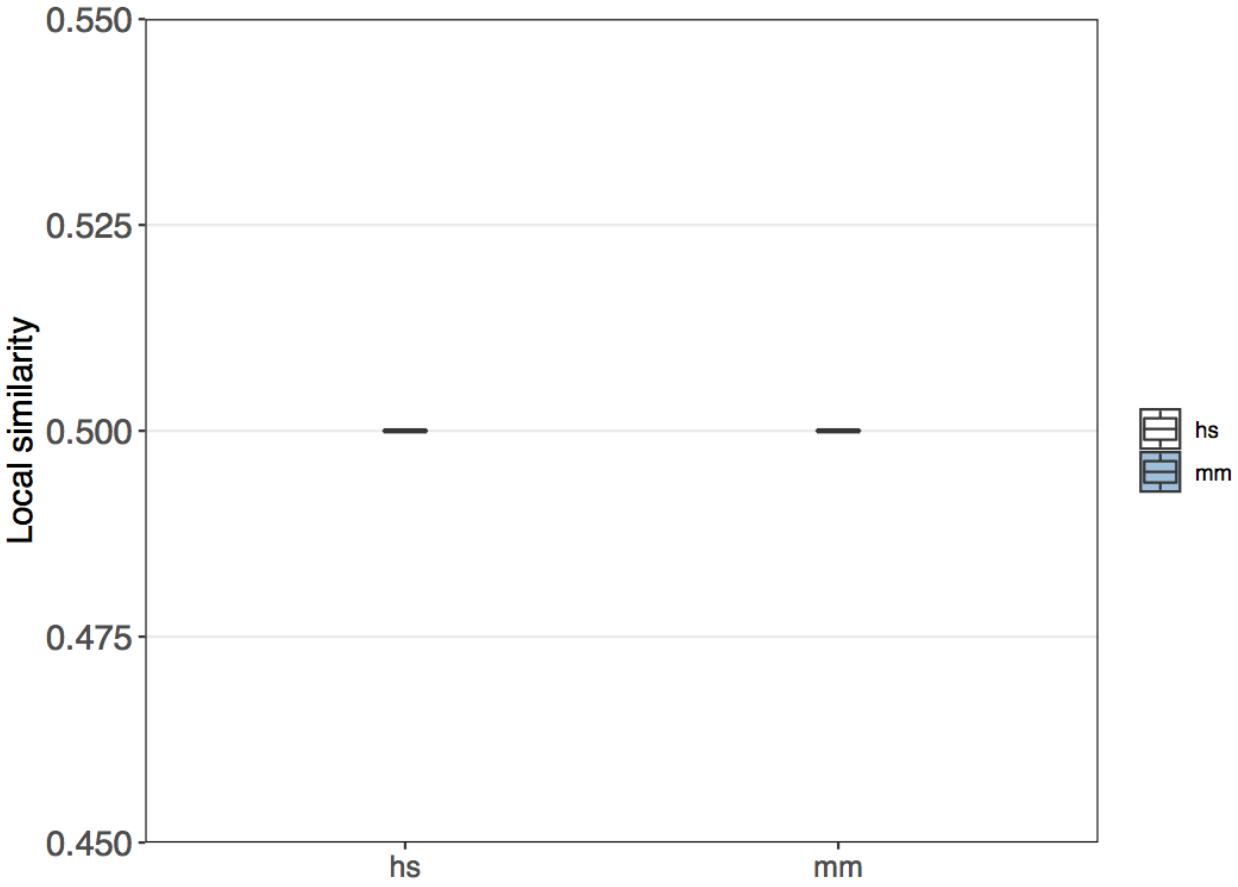


Fig. 3: Comparing local similarity between species. As the tutorial only analyzes two repertoires per category the local similarity is 0.5 for each repertoire.

(continued from previous page)

```
radar_list[["category"]]<-c("Murine A","Murine B","Human A","Human B")
radar_list[["roi"]]<-c("mm_ig_h_2_0__0_0_0_A","mm_ig_h_4_0__0_0_0_A","hs_ig_h_2_0__0_
↪0_0_A","hs_ig_h_4_0__0_0_0_A")
radar_list[["label"]]<-c("Murine A","Murine B","Human A","Human B")
radar_list[["colors"]]<-c("grey","blue",'red',"green")

comparison_list<-list(roi=radar_list[["roi"]],
  roi_names=radar_list[["category"]],
  ref="mm_ig_h_2_0__0_0_0_A",
  plot_names=radar_list[["label"]],
  colors=radar_list[["colors"]])

print_repertoire_radar(list_similarity_matrices=list_single_layers,
  to_compare=comparison_list,
  path_figure="figures",
  name_plot="tutorial")
```

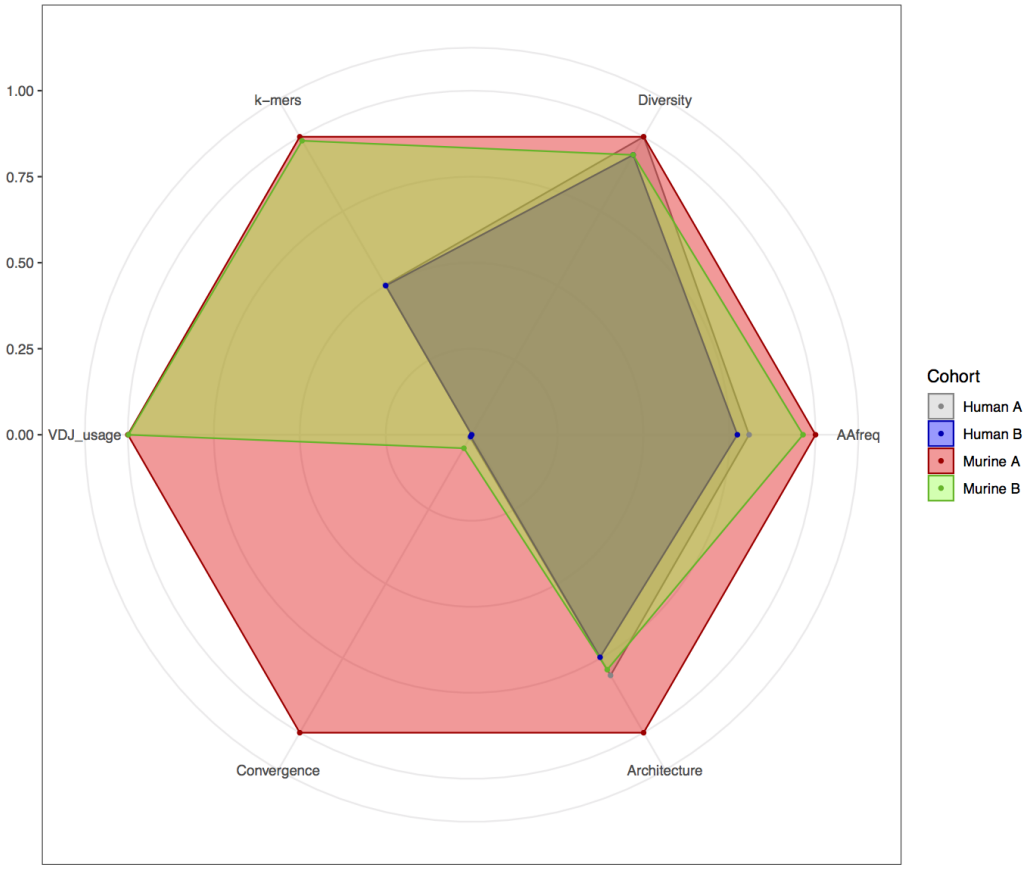


Fig. 4: Comparing 4 repertoires across six features (Reference repertoire: murine IgH).

7.1.5 E. Classical repertoire analysis on a one-to-one scale

The immuneREF function print_repertoire_comparison() outputs a comparison between two repertoires across three classical repertoire features: Amino acid frequency, Germline gene usage and Clonal expansion. The example code

below takes advantage of the results of the per-feature analysis saved in the list `repertoires_analyzed`.

```
####
# Classical repertoire analysis of maximally and minimally similar repertoires per␣
↪category
####

#print classic repertoires comparing max and min locally similar plots for:
# simulated human igh repertoires
hs_igh<-list()
hs_igh[["hs_ig_h_2_0__0_0_0_A"]]<-repertoires_analyzed[["hs_ig_h_2_0__0_0_0_A"]]
hs_igh[["hs_ig_h_4_0__0_0_0_A"]]<-repertoires_analyzed[["hs_ig_h_4_0__0_0_0_A"]]

print_repertoire_comparison(list_repertoires=hs_igh,
    name_plots="hs_igh",
    aa_freq_length=17,
    path_figure="figures")
```

7.2 Bonus: Network characteristics

Additionally, the immuneREF similarity networks may be analyzed with respect to their network features using the `analyze_similarity_network()` function, which returns a list of network features.

```
#Calculate Network Characteristics
network_features <- analyze_similarity_network(cormat)
```

7.3 References

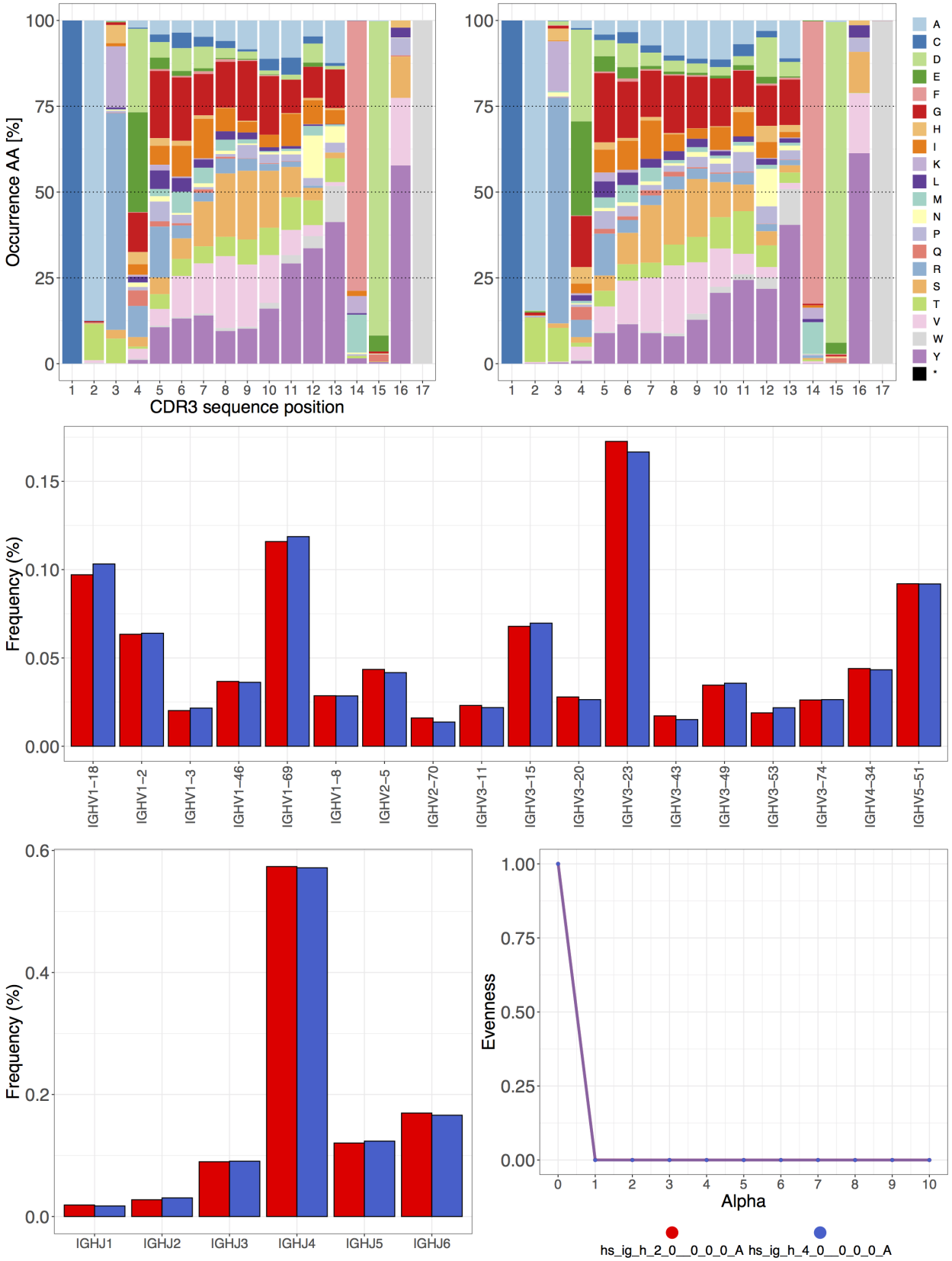


Fig. 5: Comparing two human IgH repertoires with respect to Amino acid frequency, VJ (germline gene) usage and Clonal expansion.

CHAPTER 8

Additional Exploratory Analysis

8.1 Gene expression exploratory analysis

immuneREF assumes that the input RNA-seq gene expression matrix is already (1) preprocessed with the appropriate methods for correcting possible technical noise and (2) normalized to make samples comparable. Briefly, RNA-seq technology biases include transcript length, GC content, PCR artifacts, uneven transcript read coverage, off-target transcript contamination and differences in transcript distribution. These potential biases can be corrected sequentially, or there are methods that perform in combination. For instance, the R Bioconductor package `NOISeq` includes exploratory plots to detect these errors and methods to reduce noise at each step.

8.1.1 Count data quality control and pre-processing

Once the gene expression matrix in read count format has been obtained from the raw fasta/fastq files, immuneREF assumes that the user has checked/corrected mainly (a) low count genes, (b) sequencing bias (gene length and GC content) and (c) different count distribution per sample.

Genes with low counts are generally less reliable and increase data noise, making it harder to extract relevant information. The R package NOISeq incorporates procedures to filter out low counts genes in a given dataset with the function `filtered.data`. We recommend CPM (method 1) when sample size in each condition is small, previously choosing a CPM threshold from the Sensitivity plot included NOISeq (`explo.plot` function). If the number of replicates per condition is at least five, the Wilcoxon test (method 2) is more appropriate since it does not need to set any threshold. NOISeq uses readData function to create a NOISeq object necessary for further functions. The dat function performs different calculations according to the type argument (see NOISeq manual for adequate input format). The plots can be generated for each experimental condition using the argument factor. We provide a code example for low count detection and filtering:

Low count filter detection and correction

Genes with low counts are generally less reliable and increase data noise, making it harder to extract relevant information. The R package NOISeq incorporates procedures to filter out low counts genes in a given dataset with the function `filtered.data`. We recommend CPM (method 1) when sample size in each condition is small, previously choosing a CPM threshold from the "Sensitivity plot" included NOISeq (`explo.plot` function). If the number of replicates per condition is at least five, the Wilcoxon test (method 2) is more appropriate since it does not need to

set any threshold. NOISeq uses readData function to create a NOISeq object necessary for further functions. The dat function performs different calculations according to the type argument (see NOISeq manual for adequate input format). The plots can be generated for each experimental condition using the argument factor. We provide a code example for low count detection and filtering:

```
library(NOISeq)
mydata = readData(data = RNAseq, factors = myfactors)
mycounts = dat(mydata, factor = "Group", type = "countsbio")
explo.plot(mycounts, toplot = 1, samples = NULL, plottype = "barplot")
myfilt = filtered.data(RNAseq, factor = "Group", norm = FALSE, depth = NULL, method =␣
↪1, cv.cutoff = 100, cpm = 1, p.adj = "fdr")
```

Sequencing bias detection and correction

The length bias and the GC bias plots describe the relationship between the expression values and the feature length and the GC content, respectively. Two vectors,matrices containing transcript length and GC content for each gen have to be provided (see NOISeq manual for adequate format). As before, the plots can be generated for each experimental condition using the argument factor. Code example:

```
library(NOISeq)
mydata = readData(data = myfilt, factors = myfactors, length = mylength, gc = mygc)
mylengthbias = dat(mydata, factor = "Group", type = "lengthbias")
explo.plot(mylenthbias, samples = NULL, toplot = "global")
myGCbias = dat(mydata, factor = factors$group, type = "GCbias")
explo.plot(myGCbias, samples = NULL, toplot = "global")
```

If length and GC bias are detected, there exist different normalization methods to correct them. Transcript length bias is corrected by dividing the expression matrix by a numeric vector containing the length of each feature. NOISeq allows internal length correction in the normalization step (point 2). There exist normalization methods external to NOISeq that include length gene and GC content correction, such as CQN (conditional quantile normalization) or the R/Bioconductor package EDASeq.

Different count distribution per sample

The count distribution for all samples can be visualized using a boxplot: `explo.plot(mydata, samples = NULL, plottype = "boxplot")` The normalization methods homogenize, to a greater or lesser extent, inter-sample distributions to make samples comparable for subsequent analysis. The following paragraph defines some of the available normalization methods for RNA-seq data.

8.1.2 Normalization

NOISeq performs three types of normalizations (RPKM[1], UQUA[2] and TMM[3]) that can be applied as follows:

The parameters long, k and lc are used for optional length bias correction (see NOISeq manual for more details).

One of the lightest normalization methods is RPKM (Reads Per Kilobase Million)[1], that count the total reads in a sample and divide that number by 106 ("per million" scaling factor), then divide the read counts by the "per million" scale factor, which normalizes for sequencing depth (RPM), and finally divide the RPM values by the length of the gene in kilobases (RPKM).

[1] Mapping and quantifying mammalian transcriptomes by RNA-Seq, Mortazavi et al., Nature Methods, 2008, https://www.nature.com/articles/nmeth.1226

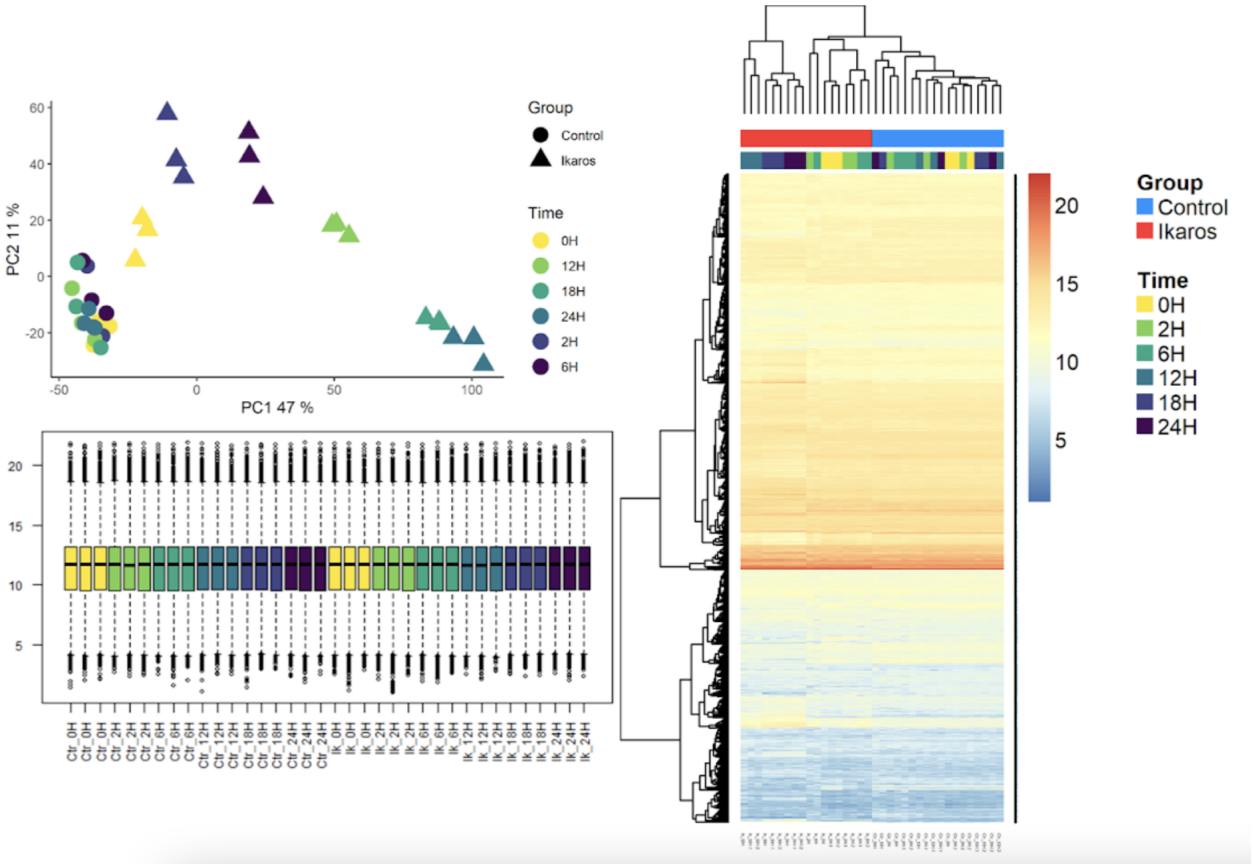
[2] Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments, Bullard et al., BMC Bioinformatics, 2010, https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-94

[3] A scaling normalization method for differential expression analysis of RNA-seq data, Robinson et al., Genome Biology, 2010, https://genomebiology.biomedcentral.com/articles/10.1186/gb-2010-11-3-r25#:~:text=TMM%20normalization%20is%20a%20simple,statistical%20methods%20for%20DE%20analysis.

Two commonly applied normalization methods are UQUA (Upper Quantile)[2] and TMM (Trimmed Mean of M values)[3], both scaling the samples to have the same upper quartile or median distribution, respectively.

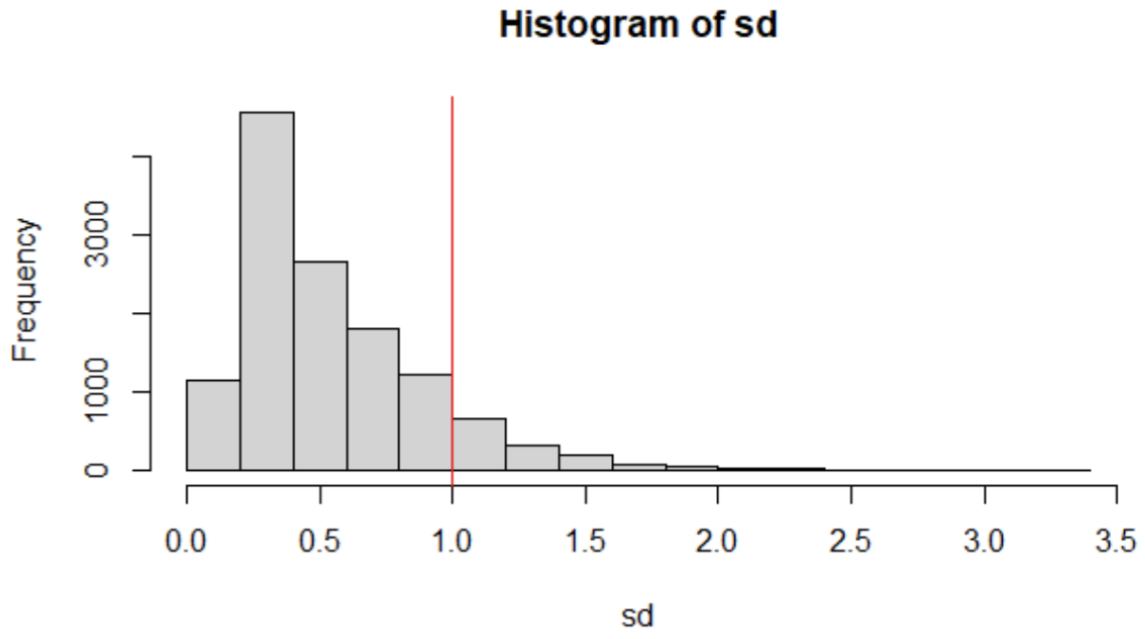
Finally, two normalization methods have been mentioned in the previous section in case sequencing bias were detected and they allow correcting gene length, GC content or both. CQN (conditional quantile normalization) applies full-quantile normalization across samples, being one of the more exaggerated normalization methods. The EDASeq R Bioconductor package includes loess normalization that transforms the data by regressing the counts on a gene feature (GC content, gene length) and subtracting the loess fit from the counts to remove the dependence.

To support quality control of the provided data, the immuneREF function `print_omics_data()` generates plots for the exploratory analysis. This allows an initial screening of sample quality to decide whether it is necessary to discard any of the samples (outliers) as well as previous check for differences in gene expression between experimental conditions. The implemented plots were PCA, boxplot and heatmap. All of them are useful for outlier detection (extreme points in the PCA score plot, boxes of the boxplot or branches in the dendrogram will point to them). Boxplots show inter-sample distributions, helping to check if the samples were comparable or the data need to be previously normalized. PCA and heatmap help to check whether gene expression differences exists in the dataset coloring by experimental condition.



Finally, immuneREF also generates a histogram for the standard deviation (SD) of the gene expression in the provided dataset. This plot helps to select the most appropriate SD threshold for immuneREF. The immuneREF function `print_sd_histogram()` ask for the expression matrix as input and generates the following plot:

8.2 References


Histogram of sd
Frequency
0
1000
3000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
sd

CHAPTER 9

Acknowledgments

This work is a close collaboration between the groups of Prof. Victor Greiff and Prof. Geir Kjetil Sandve (University of Oslo, Norway), Prof. Sai Reddy ETH Zurich, Switzerland), and the group of Dr. Xiao Liu (Beijing Genomics Institute in Shenzhen, China). In addition to all co-authors, we would like to thank in particular Milena Pavlovic, Andrei Slabodkin, Mariia Chernigovskaya, Lonneke Scheffer for package testing and feedback on the manuscript.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search