

Analyzing flow Cytometry Data in Bioconductor

Nishant Gopalakrishnan

30 July, 2010

1 Introduction

The advent of high throughput techniques has resulted in the generation of large flow cytometry (FCM) data sets. Along with the task of acquiring the data comes the task of storing, managing, quality control, data analysis and data summarization to a condensed form that can be interpreted by the researcher.

Open source Bioconductor packages for analysis of flow cytometry data provides a unified framework for bioinformaticians to develop methods to analyze and interpret flow cytometry data. The packages relevant to flow cytometry include `flowCore`, `flowViz`, `flowStats`, `flowQ`, `flowUtils`, `flowClust`, `flowFP`, `flowMeans` and `flowMerge`.

- `flowCore` - handles importing, storing, preprocessing and assessment of data from flow cytometry experiments.
- `flowViz` - provides graphical methods for visualization of flow cytometry data.
- `flowQ` - provides quality control and quality assessment tools for flow cytometry data.
- `flowStats` - provides tools and methods to analyze flow cytometry data that is beyond the basic infrastructure provided in the `flowCore` package.
- `flowUtils` - provides utilities, mainly to integrate foreign FCM analysis tools.
- `flowClust` - implements mixture model based clustering algorithms for FCM data.
- `flowMerge` - provides merging of mixture components for model-based automated gating of flow cytometry data using the `flowClust` framework.
- `flowMeans` - identifies cell populations in flow cytometry data using non-parametric clustering and segmented-regression-based change point detection.

- `flowFP` - fingerprint generation of flow cytometry data, used to facilitate the application of machine learning and data mining tools for flow cytometry

This session introduces the data structures and functions implemented in the core flow related Bioconductor packages for handling flow data. Simple exercises are provided to get familiar with basic operations such as transformations, gating and visualization of data. We then proceed to sequentially gate a simple data set using workflows implemented in the `flowCore` to get exposure to the more advanced functionalities offered by these packages. Our sample data consists of 14 patient samples from two groups treated with either drug A or B. Each sample has been stained with CD3, CD4, CD8, CD69 and HLADr fluorescence markers.

2 Data structures for flow cytometry data

2.1 *flowFrame*

The package `flowCore` has several data structures implemented for storing and manipulating flow cytometry data. The basic container for storing flow data in the `flowCore` package is a *flowFrame*.

The *flowFrame* data structure has three main methods for accessing and modifying the information stored in the *flowFrame*.

- The `exprs` method can be used to access and modify the matrix containing fluorescence intensity information. The column names of the matrix correspond to the fluorescence parameter names and each row of the matrix corresponds to a single recorded event.
- The `parameters` method can be used to access and modify information regarding the stains and the range of the fluorescence parameters that were recorded by the flow cytometer.
- The `description` method can be used to access the FCS keywords and the related information that was produced by the instrument during the measurement.

2.2 *flowSet*

Flow cytometry experiments typically involve data from several patients. It is useful to have data from an experiment organized along with the meta data information. The `flowCore` package provides the `flowSet` container that organizes several *flowFrames* together.

The *flowSet* data structure has several methods for accessing and modifying the meta data and fluorescence parameter related information stored in the *flowSet*.

- `sampleNames` can be used to access/ update the names of the samples stored in the *flowSet*.
- `colnames` can be used to access/update the fluorescence parameter names.
- The `pData` can be used to access and update the meta data information for the experiment.
- `transform` can be used to apply a transformation to all the frames in a *flowSet*.

3 Importing our example data set into a flowSet

For our example, we proceed by reading in FCS files from 14 patients and creating a *flowSet*.

Exercise 1

Install the *flowTrack* package. Copy the files included in the data folder of the package to your working directory. If you have installed the *flowTrack* package, you can find the location of the data files as follows:

```
> library(flowCore)
> library(flowTrack)
> library(flowViz)
> system.file("extdata", package = "flowTrack")
```

To read in the *flowSet* we make use of the `read.flowSet` function. The *phenoData* information is provided as a tab delimited file. The file *annotation.txt* contains a matrix with row names corresponding to the sample names and column names corresponding to names of the meta data variables.

In a clinical data collection process, the parameter information fields for each *flowFrame* may not always be updated with the appropriate fields especially when the samples have been run by different lab personnel or when an error was made in the data entry process. In such cases, the corresponding fields can be updated by using the `pData` and `parameters` methods.

Exercise 2

- Create a *flowSet* by reading in the supplied *fcs* files and annotation information using the `read.flowSet` function.
- Observe the *phenoData* information stored in the *flowSet* using the `pData` function.
- Update the *sampleNames* of the *flowSet* with the "PatientID" information from the *phenoData* information provided.
- Observe the *parameters* information for the `flowData[[1]] flowFrame`.

- Update the description field for each *flowFrame* in the *flowSet* with the stain names `c(NA, NA, "CD8", "CD69", "CD4", "CD3", "HLADr", NA)` using the `pData` and `parameters` update methods. The description fields for parameters like forward/side scatter and Time have are to be updated with `NA`, since they are not associated with a specific staining marker.

```
> flowData <- read.flowSet( phenoData="annotation.txt",
+                           transformation=FALSE)

> head(pData(flowData))
> sampleNames(flowData) <- pData(flowData)[,"PatientID"]

> for(i in seq_len(length(flowData))){
+   pData(parameters(flowData[[i]]))[, "desc"] <- c(NA, NA, "CD8", "CD69",
+                                                    "CD4", "CD3", "HLADr", "NA")
+ }
```

4 Visualize and transform the data

The package `flowViz` has several functions available for visualizing flow cytometry data.

4.1 Visualizing data

A typical call to the `flowViz` `xyplot` and `densityplot` function `xyplot` is shown below.

```
> xyplot(y ~ x , data, xlab, ylab, main)
> densityplot( ~ x, data, xlab, ylab, main)
```

The arguments to the functions can be summarized in terms of

1. formula: The first argument to a lattice method is a formula. The formula for our example is `y ~ x`. If the lattice method takes only a single vector as input, the formula can be expressed as `~ x`.
2. primary variables: Variables `y` (Y axis of the plot) and `x` (X axis of the plot) that defines the lattice display separated by the `~` character.
3. `xlab`: label for the x axis.
4. `ylab`: label for the y axis.
5. `main`: title for the plot.

4.2 Transforming data

A scatter plot matrix of all the parameters of the `flowData[[1]]` visualized using the `splom` function can be observed in the Figure 1. From the figure, it is clear that the channels FITC-A, PE-A, FL3-A, PE-CY7-A and APC-A need some form of transformation for better visualization of the data.

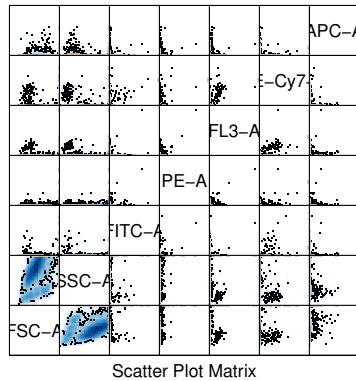


Figure 1: Scatter plot for untransformed flowData

The `flowCore` package offers several functions for transforming flow data. The most widely used transformations are the `asinh` and `logicleTransform` functions. Data in `flowFrames` and `flowSets` can be transformed using the `transform` function. The `transformList` object is useful for transforming several parameters of a `flowSet` in one operation.

Exercise 3

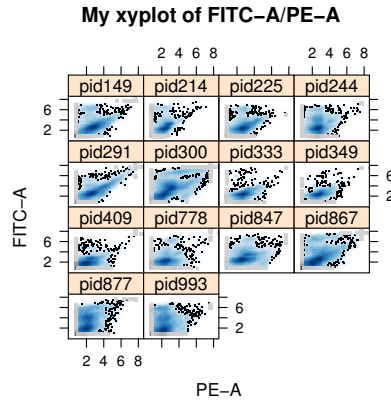
- Create an object of class `transformList` for transforming the the fluorescence channels FITC-A, PE-A, FL3-A, PE-CY7-A and APC-A using the `asinh` transformation.
- Transform the flowData flowSet that we created using the `transform` method.
- Create a scatter plot of the transformed FITC-A and PE-A channels using the `xyplot` function.
- Create a density plot of the transformed FL3-A channel using the `densityplot` function.

```
> tf <- transformList(colnames(flowData)[3:7], asinh)
> tData <- transform(flowData, tf)
> myplot <- xyplot(`FITC-A` ~ `PE-A`, tData,
+                 main=" My xyplot of FITC-A/PE-A")
```

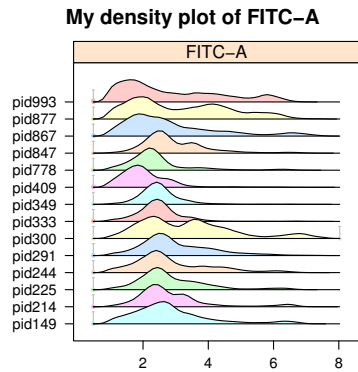
```

> myplot <- densityplot( ~ `FITC-A`, tData,
+                         main=" My density plot of FITC-A")

```



(a) FITC-A PE-A xyplot



(b) Density plot for FITC-A

Figure 2: xyplot and density plots for transformed data.

5 Workflows in flowCore

workFlows implemented in the flowCore handles the gating operations, intermediate results and naming schemes for the variables in a sequential gating

process and thereby provides a unified software managed interface to access and summarize the operations performed.

Workflows can be created using the `workFlow` function. Objects such as gates, transformations etc can be added to the workflow using the `add` function. Data for a particular view in the workflow can be retrieved using the `Data` function. The `undo` function can be used to undo an operation performed on the workflow.

Exercise 4

- Create a workflow for the transformed data `tData` called `myWork` using the `workFlow` function.
- Create a rectangle gate using the `rectangleGate` function to include the parameters `FSC-A` and `SSC-A` between the values of 100 and 600
- Add the rectangle gate to the workflow using the `add`.
- Create a scatter plot of `FSC-A` and `SSC-A` for the events included in the rectangle gate using the `xyp1ot` function. The events included in the gate can be accessed from the workflow using the `Data` function and the `[[`.

```
> mw <- workFlow(tData, name = "myWork")
> rg <- rectangleGate("FSC-A"=c(100, 600), "SSC-A"=c(100, 600),
+   filterId="rectangle")
> add(mw, rg, parent="base view")
> mw
```

A flow cytometry workflow called 'myWork'
The following data views are provided:

```
Basic view 'base view'
on a flowSet
not associated to a particular action item
```

```
View 'rectangle+'
on a flowSet linked to
gate action item 'action_rectangle'
```

```
View 'rectangle-'
on a flowSet linked to
gate action item 'action_rectangle'
```

```
> myplot <- xyp1ot(`FSC-A` ~ `SSC-A`, Data( mw[["rectangle+"]]))
```

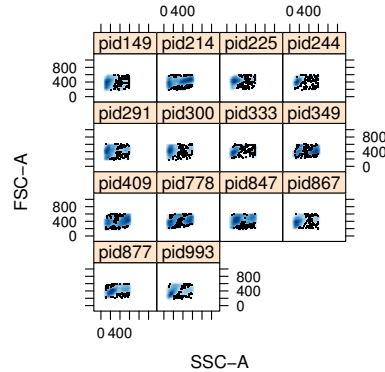


Figure 3: Scatter plot for rectangle gated data

6 Detailed analysis of our data set

Our data set consists of 14 flowFrames consisting of patients from two groups that underwent treatments using two drugs A and B. Each sample has been stained for CD3,CD4,CD8, CD69 and HLADR. We begin by reading in flow cytometry data from raw FCS files and storing them in appropriate data structures. We proceed to identify sub populations of helper and cytotoxic T lymphocytes using sequential gating strategies. Our goal is to calculate the proportion of helper and cytotoxic T cells that exhibit the HLADr activation marker amongst the two groups of patients in our data.

We make use of workflows implemented in the flowCore package to keep track of operations performed on our data set, organize the intermediate results generated as well as handle the naming schemes of intermediate variables that are created during the analysis.

Our goal in this data analysis process is to separate out the activated T helper/inducer cells and cytotoxic/suppressor T cells from the rest of the population and compare the results amongst the two groups of patients (Drugs A and B). Figure 4 summarizes our gating strategy for identifying the cells of interest.

The column names of the data are updated with the stain names before it is added to the workflow.

```
> colnames(flowData) <- c("FSC", "SSC", "CD8", "CD69", "CD4", "CD3",
+                          "HLADr", "Time")
> wf <- workflow(flowData, name="biocExample")
```

The data is first transformed for better visualization of data using the `asinh` transformation. We make use of a `transformList` object to transform the fluorescence channels stained for CD8, CD69, CD4, CD3 and HLADr channels.

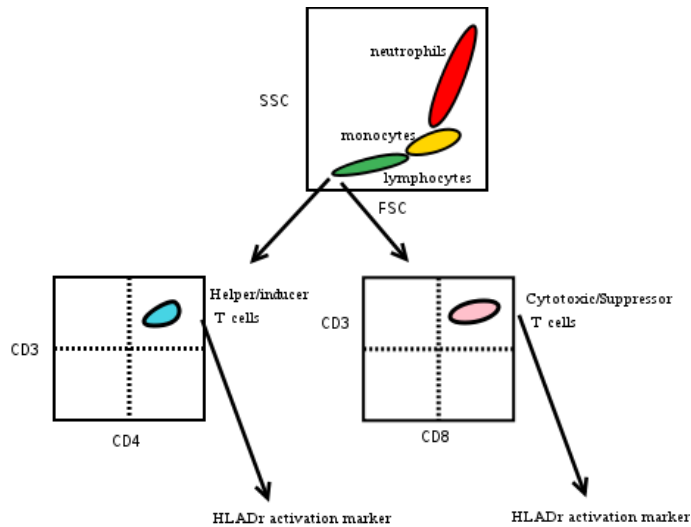


Figure 4: Sequential gating strategy for our sample data set

The transformList object created can then be added to the workflow using the add function.

```
> tf <- transformList(colnames(Data(wf[["base view"]]))[3:7], asinh,
+                     transformationId="asinh")
> add(wf, tf)
> wf
```

A flow cytometry workflow called 'biocExample'
The following data views are provided:

```
Basic view 'base view'
on a flowSet
not associated to a particular action item
```

```
View 'asinh'
on a flowSet linked to
transform action item 'action_asinh'
```

Before we proceed with the gating operations, we first remove the boundary events for the FSC and SSC channels. This is done by use of the boundaryFilter function. We proceed to create a boundary filter object and add it to our workflow

```

> boundFilt <- boundaryFilter(filterId="boundFilt", x=c("FSC","SSC"))
> add(wf,boundFilt,parent="asinh")

```

6.1 T lymphocyte population

The first step in our gating strategy is to identify the lymphocyte subpopulation using the `lymphGate` function. It selects elliptical cell subpopulations from two dimensional projections by fitting a bivariate normal distribution to a preselected rectangular area.

We make use of the fact that the CD3 reagent binds specifically to the T lymphocytes for the pre-election process. The `lymphGate` identifies a rectangular area containing the lymphocytes by gating areas that are CD3 positive and then fits a bivariate normal distribution to this selected area using the `norm2Filter` function.

After the lymph gate gets added to the workflow, our initial population gets split into two groups - one containing T lymphocytes(Tcells+) and the other containing all the other cell groups(Tcells-)

The T cell population selected by the lymph Gate can be observed in Figure 5.

```

> lg <- lymphGate(Data(wf[["boundFilt+"]]), channels=c("FSC", "SSC"),
+                 preselection="CD3", filterId="TCells", eval=FALSE,
+                 scale=2.5)
> add(wf, lg$n2gate, parent="boundFilt+")
> myplot <- xyplot(SSC ~ FSC | PatientID, wf[["TCells+"]],
+                 par.settings=list(gate=list(col="red",
+                 fill="red", alpha=0.3)))

```

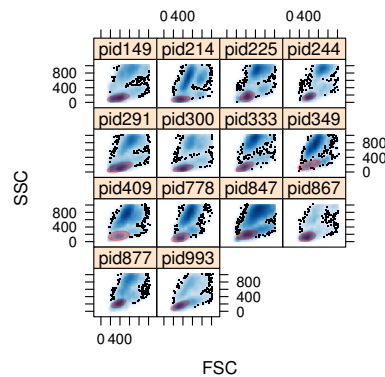


Figure 5: Lymph gate

6.2 T helper inducer cells and Cytotoxic suppressor populations

The dyes used in flow cytometry are chosen so that they can distinguish between sub populations when used in combination. For our example, CD4 binds to helper/inducer T lymphocytes and monocytes. A combination of CD3/CD4 reagents can be used to separate the helper/inducer T cells from the rest of the T cells and monocytes (if any are included by use of a more liberal lymph gate).

Similarly, CD8 binds to cytotoxic/suppressor cells and NK cells. A combination of CD3/CD8 can be used to separate the cytotoxic/suppressor T cells from the remaining T cells and NK cells.

6.2.1 Need for data normalization

The density plots for the CD3, CD4 and CD8 channels are shown below. Our goal is to identify the cell population groups that are CD3+/CD4+ and CD3+/CD8+. While this can be achieved by making use of the `quadGate` function on each `flowFrame`, this process can be tedious as the gate dimensions required for each sample could vary considerably especially in case of CD3 from our example because the peaks in the density plot are not aligned.

```
> myplot <-densityplot(PatientID~., Data(wf[["TCells+"]]),
+   channels=c("CD3","CD4","CD8"), groups=GroupID,
+   scales=list(y=list(draw=F)),
+   filter=lapply(c("CD3","CD4","CD8"), curv1Filter),
+   layout=c(3,1))
```

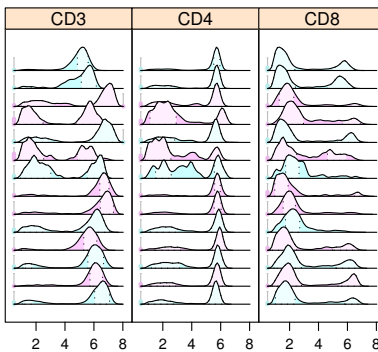


Figure 6: Data before normalization

A single quad gate could be used if the data could be normalized so that the respective peaks align.

6.2.2 Data normalization

The `warpSet` function could be used to normalize the data based on the identification of high density areas (termed "landmarks") and the subsequent computation of appropriate transformation functions for each `flowFrame` so that the identified landmarks are aligned.

The density plot for the CD3,CD4 and CD8 fluorescence parameters after normalization can be observed in Figure 7.

```
> pars <- colnames(Data(wf[["base view"]]))[c(3,4,5,6)]
> norm <- normalization(normFun=function(x, parameters, ...)
+                       warpSet(x, parameters,...),
+                       parameters=pars,
+                       normalizationId="Warping")
> add(wf, norm, parent="TCells+")

Estimating landmarks for channel CD8 ...
Estimating landmarks for channel CD69 ...
Estimating landmarks for channel CD4 ...
Estimating landmarks for channel CD3 ...
Registering curves for parameter CD8 ...
Registering curves for parameter CD69 ...
Registering curves for parameter CD4 ...
Registering curves for parameter CD3 ...

> myplot <- densityplot(PatientID~., Data(wf[["Warping"]]),
+                       channels=c("CD3","CD4","CD8"), groups=GroupID,
+                       scales=list(y=list(draw=F)),
+                       filter=lapply(c("CD3","CD4","CD8"), curv1Filter),
+                       layout=c(3,1))
```

6.2.3 Quadrant gates to identify T helper/inducer cell and cytotoxic/suppressor T cell populations

The `quadrantGate` function can be used to separate two dimensional data into positive and negative quadrants based on the density estimates of the two parameters under consideration. Essentially, this creates a single quadrant gate based on the joint data of all `flowFrames`.

We apply the `quadrantGate` function to the normalized data to identify the CD3+CD4+ sub population.

```
> qgate <- quadrantGate(Data(wf[["Warping"]]), stains=c("CD3", "CD4"),
+                       plot=FALSE, filterId="CD3CD4")
> add(wf, qgate, parent="Warping")
```

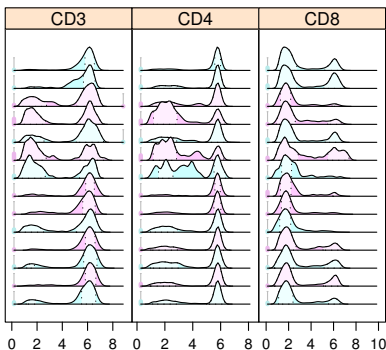


Figure 7: Data after normalization

```
> myplot <- xyplot(CD3 ~ CD4 | PatientID, wf[["CD3+CD4+"]] )
```

Similarly, we apply the `quadrantGate` function to the CD3 and CD8 channels to identify the CD3+CD8+ population corresponding to the cytotoxic/suppressor T cells.

```
> qgate <- quadrantGate(Data(wf[["Warping"]]), stains=c("CD3", "CD8"),
+                       plot=FALSE, filterId="CD3CD8")
> add(wf, qgate, parent="Warping")
```

```
> myplot <- xyplot(CD3 ~ CD8 | PatientID, wf[["CD3+CD8+"]])
```

6.3 T Cell activation marker HLADr

We are interested in the proportion of T cells that exhibit the activation marker HLADr for the helper and cytotoxic T cells. We make use of the `rangeGate` to identify T cells that express the HLADr marker from a one dimensional density estimate of the data.

```
> HLADr1 <- rangeGate(Data(wf[["CD3+CD4+"]]), stain="HLADr", plot=FALSE,
+                     alpha=0.75, filterId="CD3+CD4+HLAct")
> add(wf, HLADr1, parent="CD3+CD4+")
```

```
> myplot <- densityplot(PatientID ~ HLADr, Data(wf[["CD3+CD4+"]]),
+                       refline=HLADr1@min)
```

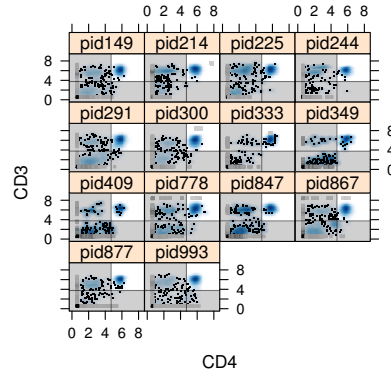


Figure 8: CD3+CD4+ quadrant gate

```
> HLADr2 <- rangeGate(Data(wf[["CD3+CD8+"]]), stain="HLADr", plot=FALSE,
+                       alpha=0.75, filterId="CD3+CD8+HLAct")
> add(wf, HLADr2, parent="CD3+CD8+")

> myplot <- densityplot(PatientID ~ HLADr, Data(wf[["CD3+CD8+"]]),
+                       reffline=HLADr2@min)
```

The proportion of helper T cells that exhibit the HLADr activation marker to the total count of events in the original sample is calculated and plotted in Figure 12.

```
> pr <- fsApply(Data(wf[["CD3+CD4+HLAct-"]]),nrow)*100 /
+           fsApply(Data(wf[["boundFilt+"]]),nrow)
> res <- data.frame(pr, pData(Data(wf[["CD3+CD4+"]]))[c("GroupID", "PatientID")])

> myplot <- barchart(reorder(PatientID, as.numeric(factor(GroupID))) ~ pr,
+                   data=res, groups=GroupID, stack=TRUE,
+                   auto.key=list(points = FALSE, rectangles = TRUE, space = "right"),
+                   main="Activated CD3+CD4+ T cells",
+                   xlab="Percentage of activated cells")
```

Similarly, the proportion of cytotoxic T Cells that express the activation marker HLADr to the total count of events in the original sample is calculated and plotted in Figure 13.

```
> pr <- fsApply(Data(wf[["CD3+CD8+HLAct-"]]),nrow)*100 /
+           fsApply(Data(wf[["boundFilt+"]]),nrow)
> res <- data.frame(pr, pData(Data(wf[["CD3+CD8+"]]))[c("GroupID", "PatientID")])
```

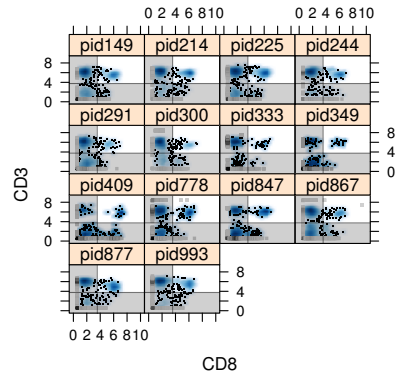


Figure 9: CD3+CD8+ quadrant gate

```

> myplot <- bargraph(reorder(PatientID,as.numeric(factor(GroupID)))) ~ pr,
+   data=res, groups=GroupID, stack=TRUE,
+   auto.key = list(points = FALSE, rectangles = TRUE, space = "right"),
+   main="Activated CD3+CD8+ T cells",
+   xlab=" Percentage of activated cells")

```

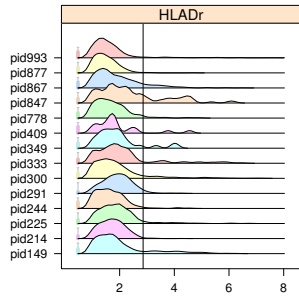


Figure 10: HLADr activated CD3+CD4+

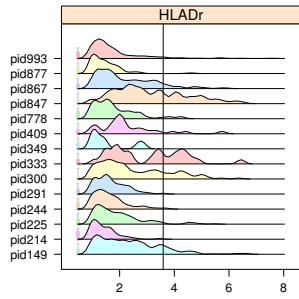


Figure 11: HLADr activated CD3+CD8+

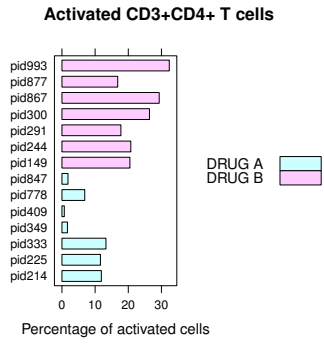


Figure 12: HLADr activated CD3+CD4+ Summary

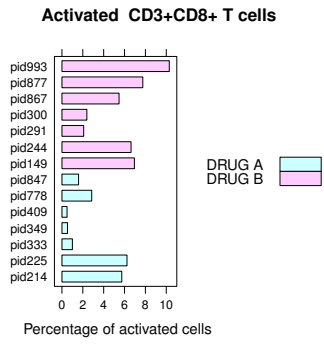


Figure 13: HLADr activated CD3+CD8+ Summary

7 Session information

- R version 2.11.1 (2010-05-31), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=en_US.UTF-8, LC_MONETARY=C, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, graphics, grDevices, methods, splines, stats, utils
- Other packages: Biobase 2.8.0, cluster 1.12.3, fda 2.2.2, flowCore 1.14.1, flowStats 1.6.0, flowTrack 0.0.1, flowViz 1.12.0, lattice 0.18-8, mvoutlier 1.4, mvtnorm 0.9-92, pcaPP 1.8-2, robustbase 0.5-0-1, rrcov 1.0-01, zoo 1.6-4
- Loaded via a namespace (and not attached): feature 1.2.4, graph 1.27.16, grid 2.11.1, KernSmooth 2.23-3, ks 1.6.13, latticeExtra 0.6-11, MASS 7.3-6, RColorBrewer 1.0-2, stats4 2.11.1, tools 2.11.1