BioC 2010 Automated Gating and Metaclustering for Flow Cytometry Data

Greg Finak

Computational Biology Unit.
Gottardo Laboratory
IRCM, Montreal, Canada

July 29, 2010

Goals

Purpose of the workshop

- Learn how to use `flowClust` and `flowMerge` effectively for automated gating.
- Understand the effects of `flowClust` parameters on gating.
- Learn to metacluster discovered cell populations.

Software Requirements

Software you should have installed

- R 2.11.1
- Bioconductor 2.6.0
- `flowMerge` 1.3.7
- `flowClust` 2.7.0
- `snowfall`
- `flowViz`
- `flowStats`
- `fpc`

If you're missing the development versions of flowClust or flowMerge, there is a hard drive being passed around that has the development versions of flowClust and flowMerge.

Goals

After working through these examples you should be able to:

- Run `flowClust` and `flowMerge` in a fully automated manner
- Run `flowClust` and `flowMerge` in a semi-automated manner
- Run `flowClust` model fitting in parallel using `snow` and `snowfall` packages
- Understand the role of different model parameters and how changing them impacts gated cell pouplations
- Be able to manipulate and evaluate `flowClust` and `flowMerge` model objects in R
- Perform metaclustering of cell populations across samples using `flowMetaCluster`

Overview

Why Automated Gating with BioConductor?

- Manual gating has drawbacks for large data sets.
- Analysis using BioConductor can be automated
- Customizable

Data Set

Analysis Overview

Analysis of a data set comparing two drug treatment protocols. Goal is gating and measuring the fraction of T–helper and cytotoxic T–cells expressing HLADr activation marker. We'll use different tools, including `flowClust`, `flowMerge` and gating functionality from `flowCore` to walk through the analysis of this data.

Preliminaries

Loading Required Libraries

BioConductor is modular. Different functionality comes from different user–supplied packages. To utilize the functionality provided by a given package, the package must be loaded.

```
require(flowMerge)
require(flowViz)
require(flowQ)
require(flowStats)
require(snowfall)
require(fpc)
require(flowTrack)
```

Loading Data

Reading into a flowSet:

To load the data we run:

```
file.loc <- system.file("extdata",
   package = "flowTrack")
data <- read.flowSet(path = file.loc,
   pattern = "fcs", phenoData = "annotation.txt",
   transformation = FALSE)
```

Inspecting Data

We can get additional information about the flowSet by entering the name of the flowSet at the R command line:

```
 data
```

```
A flowSet with 14 experiments.

An object of class "AnnotatedDataFrame"
  rowNames: 01126211_NB8_I025.fcs, 01247181
  _NB06_I025.fcs, ..., 12015151_NB8_I025.fc
  s  (14 total)
  varLabels and varMetadata description:
    PatientID:
    GroupID:
    ...: ...
    name: Filename
    (6 total)

  column names:
  FSC-A SSC-A FITC-A PE-A FL3-A PE-Cy7-A APC-A Time
```

We see it is composed of 14 sample (experiments), and we see some additional information about the AnnotatedDataFrame associated with the flowSet (we see file names, and that there are 6 pieces of metadata assocaited with each sample).

Inspecting Data

Similarly, by calling:

```
data[[1]]
```

```
flowFrame object '01126211_NB8_I025.fcs'
with 4000 cells and 8 observables:
        name         desc range minRange
$P1    FSC-A         <NA>  1024        0
$P2    SSC-A         <NA>  1024        0
$P3   FITC-A   CD8 FITC-A  1024        1
$P4     PE-A    CD69 PE-A  1024        1
$P5    FL3-A          CD4  1024        1
$P6 PE-Cy7-A CD3 PE-Cy7-A  1024        1
$P7    APC-A  HLADr APC-A  1024        1
$P8     Time         <NA>  1024        0
    maxRange
$P1     1023
$P2     1023
$P3     1023
$P4     1023
$P5     1023
$P6     1023
$P7     1023
$P8     1023
98 keywords are stored in the 'description' slot
```

we get some additional information about the first flowFrame. We see it has 4000 events and 8 channels. It is named using its associated FCS file name. The channels are named by their associated dye ("names" column). The assocaited markers are in the "desc" column.

Inspecting the Data

If we examine the default sample and marker names associated with the flowSet, we see that they are not very informative:

```
colnames(data)
```

```
[1] "FSC-A"    "SSC-A"    "FITC-A"
[4] "PE-A"     "FL3-A"    "PE-Cy7-A"
[7] "APC-A"    "Time"
```

```
sampleNames(data)
```

```
 [1] "01126211_NB8_I025.fcs"
 [2] "01247181_NB06_I025.fcs"
 [3] "02276161_NB8_I025.fcs"
 [4] "03157141_NB06_I025.fcs"
 [5] "05107251_NB06_I025.fcs"
 [6] "06226101_NB8_I025.fcs"
 [7] "07115141_NB8_I025.fcs"
 [8] "07215281_NB8_I025.fcs"
 [9] "08045071_NB8_I025.fcs"
[10] "09225121_NB8_I025.fcs"
[11] "10175181_NB8_I025.fcs"
[12] "10276181_NB8_I025.fcs"
[13] "11145351_NB8_I025.fcs"
[14] "12015151_NB8_I025.fcs"
```

The default sample and channel names appear on any generated plots and graphics. We want to assign the more informative `"PatientID"` from the annotation metadata as sample names:

We can view the metadata via:

```
                        PatientID GroupID
01126211_NB8_I025.fcs      pid349  DRUG A
01247181_NB06_I025.fcs     pid300  DRUG B
02276161_NB8_I025.fcs      pid778  DRUG A
03157141_NB06_I025.fcs     pid291  DRUG B
05107251_NB06_I025.fcs     pid214  DRUG A
06226101_NB8_I025.fcs      pid867  DRUG B
07115141_NB8_I025.fcs      pid877  DRUG B
07215281_NB8_I025.fcs      pid409  DRUG A
08045071_NB8_I025.fcs      pid993  DRUG B
09225121_NB8_I025.fcs      pid847  DRUG A
10175181_NB8_I025.fcs      pid244  DRUG B
10276181_NB8_I025.fcs      pid149  DRUG B
11145351_NB8_I025.fcs      pid225  DRUG A
12015151_NB8_I025.fcs      pid333  DRUG A
                        Technician  Project
01126211_NB8_I025.fcs         Jill BIOC2009
01247181_NB06_I025.fcs        Jill BIOC2009
```

```
02276161_NB8_I025.fcs        Jill BIOC2009
03157141_NB06_I025.fcs      Peter BIOC2009
05107251_NB06_I025.fcs      Peter BIOC2009
06226101_NB8_I025.fcs        Jill BIOC2009
07115141_NB8_I025.fcs        Jill BIOC2009
07215281_NB8_I025.fcs        Jill BIOC2009
08045071_NB8_I025.fcs        Jill BIOC2009
09225121_NB8_I025.fcs        Mark BIOC2009
10175181_NB8_I025.fcs        Mark BIOC2009
10276181_NB8_I025.fcs        Mark BIOC2009
11145351_NB8_I025.fcs        Mark BIOC2009
12015151_NB8_I025.fcs        Mark BIOC2009
                                             Stain
01126211_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
01247181_NB06_I025.fcs CD8/CD69/CD4/CD3/HLADR
02276161_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
03157141_NB06_I025.fcs CD8/CD69/CD4/CD3/HLADR
05107251_NB06_I025.fcs CD8/CD69/CD4/CD3/HLADR
06226101_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
```

```
07115141_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
07215281_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
08045071_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
09225121_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
10175181_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
10276181_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
11145351_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
12015151_NB8_I025.fcs  CD8/CD69/CD4/CD3/HLADR
                                           name
01126211_NB8_I025.fcs   01126211_NB8_I025.fcs
01247181_NB06_I025.fcs 01247181_NB06_I025.fcs
02276161_NB8_I025.fcs   02276161_NB8_I025.fcs
03157141_NB06_I025.fcs 03157141_NB06_I025.fcs
05107251_NB06_I025.fcs 05107251_NB06_I025.fcs
06226101_NB8_I025.fcs   06226101_NB8_I025.fcs
07115141_NB8_I025.fcs   07115141_NB8_I025.fcs
07215281_NB8_I025.fcs   07215281_NB8_I025.fcs
08045071_NB8_I025.fcs   08045071_NB8_I025.fcs
09225121_NB8_I025.fcs   09225121_NB8_I025.fcs
```

```
10175181_NB8_I025.fcs   10175181_NB8_I025.fcs
10276181_NB8_I025.fcs   10276181_NB8_I025.fcs
11145351_NB8_I025.fcs   11145351_NB8_I025.fcs
12015151_NB8_I025.fcs   12015151_NB8_I025.fcs
```

Preparing the Data

Renaming Samples

The following code assigns the "PatientID" columns as the sample names:

```
 sampleNames(data) <- as.character(pData(data)[,
    "PatientID"])
```

Renaming Channels

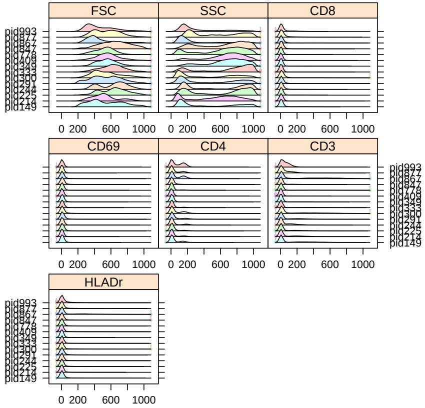
In order to use the more informative marker names as the channel names, rather than the names of the dyes, we run the following:

```
 for (i in seq_len(length(data))) {
    pData(parameters(data[[i]]))[,
        "desc"] <- c("NA", "NA",
        "CD8", "CD69", "CD4", "CD3",
        "HLADr", "NA")
}
 colnames(data) <- c("FSC", "SSC",
    "CD8", "CD69", "CD4", "CD3",
    "HLADr", "Time"))
```

Transformation and Preprocessing

Before Transformation - 1D Density Plots

densityplot(˜., data)



- Forward and side scatter cover the range of data.
- Fluorescence channels need to be transformed.
- Many to choose from in `flowCore`

Data Transformation

Logicle Transform

We'll use the logicle transformation from `flowCore` to transform the fluorescence channels.

```
tData <- transform(data, transformList(colnames(data)[3:7],
   logicleTransform()))
```

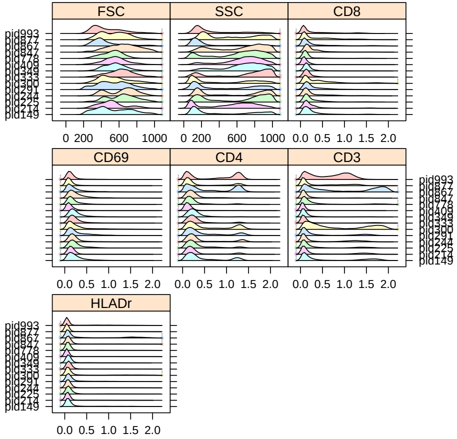
- `transformList()` constructs a list of channels and transformation functions.
- `transform()` applies the transformation list to data.

Data Transformation

Data after transformation

We can now plot the transformed data for comparison.

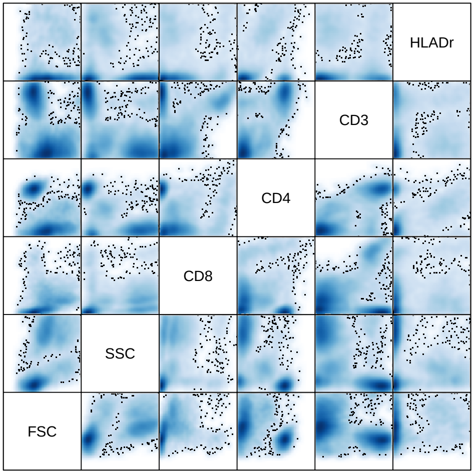
```
densityplot(~., tData)
```



Looking at the CD3 and CD4 dimensions we can see that the different populations are readily visible.

All Channels Scatterplot

```
splom(tData[[2]][, c(1, 2, 3, 5,
   6, 7)], smooth = TRUE)
```



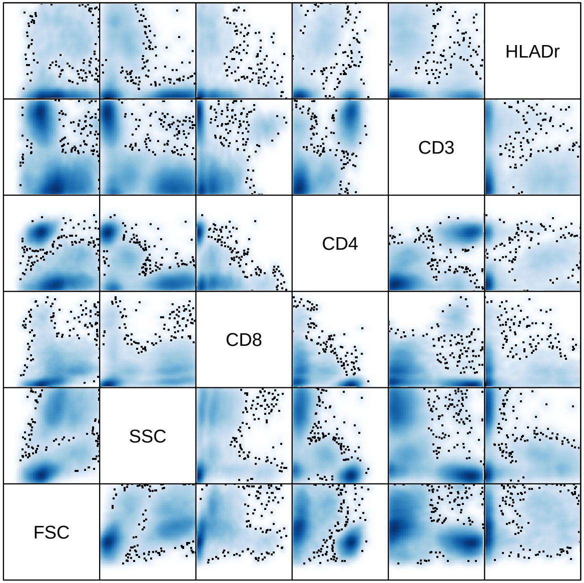
Scatter Plot Matrix

- Different populations are resolved in different channels following transformation.
- Need to remove boundary events in FSC and SSC (details in the handout).

```
tData.f <- Subset(tData, filter(tData,
           boundaryFilter(c("FSC", "SSC"))))
```

The above code constructs a boundary filter object via `boundaryFilter()`, taking the names of the channels to be filtered. Next, via `filter()` the filtering operation is performed on `tData`. Finally, the filtered data are extracted into a new flowFrame object via `Subset()`. Let's visualize some of the filtered data:

```
splom(tData.f[[2]][, c(1, 2, 3, 5,
           6, 7)], smooth = TRUE)
```



Scatter Plot Matrix

Normalization

We'll normalize the fluorescence channels so that populations are lined up at the same intensities across replicates. Details on normalization are in the `flowSet` package.

```
 tData.n <- normalize(tData.f, normalization(parameters = colnames(tData.f)[3:
    normFun = function(x, parameters,
        ...) warpSet(x, parameters,
        ...)))
```

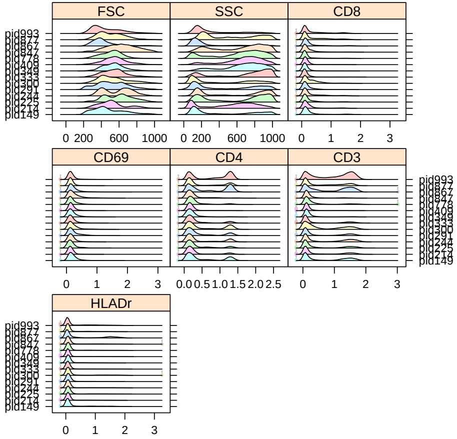
```
Estimating landmarks for channel CD8 ...
Estimating landmarks for channel CD69 ...
Estimating landmarks for channel CD4 ...
Estimating landmarks for channel CD3 ...
Estimating landmarks for channel HLADr ...
Registering curves for parameter CD8 ...
Registering curves for parameter CD69 ...
Registering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

Normalization

Plot Normalized Data

Let's plot the normalized data, just to check.

```
densityplot(~., tData.n)
```



With the preprocessing completed, we move on to automated gating with flowClust and flowMerge.

Gating Strategy

Target Population

Fraction of CD3+/CD4+ and CD3+/CD8+ T-cells that express activation marker HLADr following treatment with two different drugs.

Options

- All–at–once gating (scatter + fluorescence dimensions)
- Sequential gating: scatter channels first, followed by fluorescence channels.
- Gaussian vs t-mixture models.
- With and without transformation.

We'll begin by comparing sequential vs all–at–once gating.

Since this may take some time to compute, you can load pre–computed data with:

Sequential vs All–at-Once Gating

Compare Different Parameterization Options in flowClust

Note: these results can be loaded with:

```
data(flowMerge1)
```

We'll look at just the first sample for now. Compare sequential vs all–at–once gating, with and without transformation..

```
fc.all <- flowClust(tData.n[[1]],
    K = 1:10, varNames = colnames(tData.n[[1]])[1:7],
    trans = 1, nu = 4, nu.est = 1)
fc.all.notrans <- flowClust(tData.n[[1]],
    K = 1:10, varNames = colnames(tData.n[[1]])[1:7],
    trans = 0, nu = 4, nu.est = 1)
fc.sequential <- flowClust(tData.n[[1]],
    K = 1:10, varNames = colnames(tData.n[[1]])[c(1,
        2)], trans = 1, nu = 4,
    nu.est = 1)
fc.sequential.notrans <- flowClust(tData.n[[1]],
    K = 1:10, varNames = colnames(tData.n[[1]])[c(1,
```

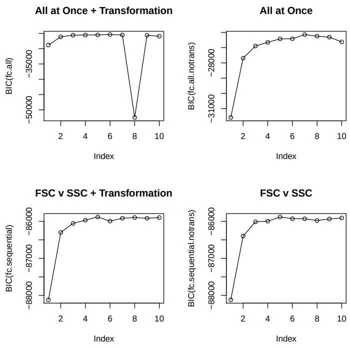
We can examine the BIC plots for each set of models and we can choose the model with the max BIC from each fitting procedure and then we can proceed to plot the best fitting models to get an idea how they differ.

Model Selection

BIC

We use the Bayesian Information Criterion (BIC) to choose the best fitting model within each model class.

```
par(mfrow = c(2, 2))
plot(BIC(fc.all), main = "All at Once + Transformation",
   type = "o")
plot(BIC(fc.all.notrans), main = "All at Once",
   type = "o")
plot(BIC(fc.sequential), main = "FSC v SSC + Transformation",
   type = "o")
plot(BIC(fc.sequential.notrans),
   main = "FSC v SSC", type = "o")
```



Automate Model Selection

Model selection can be automated, without examining the BIC plots.

```
fc.all <- fc.all[[which.max(BIC(fc.all))]]
fc.all.notrans <- fc.all.notrans[[which.max(BIC(fc.all.notrans))]]
fc.sequential.notrans <- fc.sequential.notrans[[which.max(BIC(fc.sequential.notrans))]]
fc.sequential <- fc.sequential[[which.max(BIC(fc.sequential))]]
```

flowClust Object Details

getSlots("flowClust")

```
      expName      varNames             K 
  "character"   "character"     "numeric" 
            w            mu         sigma 
     "vector"      "matrix"       "array" 
       lambda            nu             z 
    "numeric"     "numeric"      "matrix" 
            u         label   uncertainty 
     "matrix"      "vector"      "vector" 
 ruleOutliers  flagOutliers        rm.min 
     "vector"      "vector"     "numeric" 
       rm.max       logLike           BIC 
    "numeric"     "numeric"     "numeric" 
          ICL 
    "numeric" 
```

Some Helper and Accessor Methods

slots

You can access the data in any slot using the `@` method.

```
fc.all@mu
fc.all@sigma
```

This would allow you to inspect the means and covariances of each cell population (cluster) in the model.

Mapping Events to Cell Subpopulations

Mapping from events to cell populations is done via the `z` matrix.

- Probability of membership of each element to each of the model clusters.
- Get the mapping using the `Map()` method.

```
Map(fc.all)
```

Linking Data and Models

`flowObj` objects

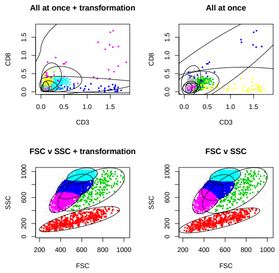
flowClust generates multiple models for the same set of data. In order to keep track of which data set corresponds to which model, we can combine them in a `flowObj` object. Plotting `flowClust` models is also simplified for this type of object.

```
fc.all <- flowObj(fc.all, tData.n[[1]])
fc.all.notrans <- flowObj(fc.all.notrans,
   tData.n[[1]])
fc.sequential <- flowObj(fc.sequential,
   tData.n[[1]])
fc.sequential.notrans <- flowObj(fc.sequential.notrans,
   tData.n[[1]])
```

- flowObj combined model and data
- Memory efficient: keep only one copy of the data for multiple models
- Extract data via `getData(object)`

Plotting flowClust Models

```
par(mfrow = c(2, 2))
plot(as(fc.all, "flowClust"), data = tData.n[[1]],
   subset = c("CD3", "CD8"), pch = 20,
   main = "All at once + transformation")
plot(as(fc.all.notrans, "flowClust"),
   data = tData.n[[1]], subset = c("CD3",
       "CD8"), pch = 20, main = "All at once")
plot(as(fc.sequential, "flowClust"),
   data = tData.n[[1]], pch = 20,
   main = "FSC v SSC + transformation")
plot(as(fc.sequential.notrans,
   "flowClust"), data = tData.n[[1]],
   pch = 20, main = "FSC v SSC")
```



- All–at–once clustering with or without transformation doesn't identify the CD8/CD3+ cell population correctly.
- Optimal parameters for scatter dimensions and fluorescence dimensions are different.
- Could use gaussian mixtures, but it would affect the clustering of scatter dimensions.
- Best option is sequential gating of scatter followed by fluorescence dimensions.
- Single cell population modeled by three clusters in the scatter channels.
- `flowMerge` can help model it as distinct cell population.

flowMerge Objects

getSlots("flowMerge")

```
         merged       entropy
      "numeric"     "numeric"
           DATA       expName
  "environment"   "character"
       varNames             K
    "character"     "numeric"
              w            mu
       "vector"      "matrix"
          sigma        lambda
        "array"     "numeric"
             nu             z
      "numeric"      "matrix"
              u         label
       "matrix"      "vector"
    uncertainty  ruleOutliers
       "vector"      "vector"
   flagOutliers        rm.min
       "vector"     "numeric"
         rm.max       logLike
      "numeric"     "numeric"
            BIC           ICL
      "numeric"     "numeric"
```

flowMerge

Purpose

Merge overlapping clusters to model distinct cell populations.

Usage

flowMerge requires that each cluster is modeled with a common degrees of freedom and a common transformation parameter. Merging clusters is trivial, given the `flowObj` model object:

```
m1 <- merge(fc.sequential.notrans)
m2 <- merge(fc.sequential)
```

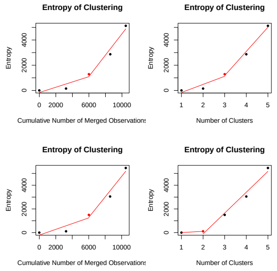
The code above merges clusters in the sequentially gated model, with and without transformation. The output is a list of merged models. The next step is to identify the optimal merged model.

flowMerge

Evaluating Merged Models

The entropy of clustering is a measure of how much clusters overlap with each other. By examining plots of the entropy vs the number of clusters, or vs the cumulative number of merged observations, we can identify the point where further merging doesn't significantly improve the entropy (i.e. clusters are well separated).

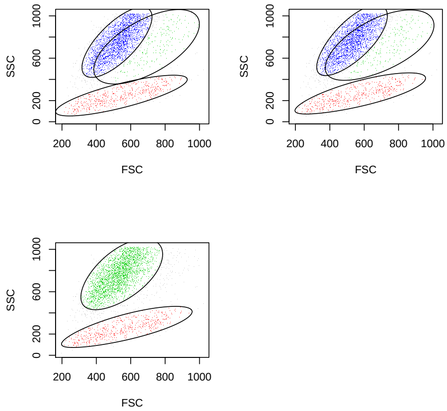
```
par(mfrow = c(2, 2))
fitPiecewiseLinreg(m1, plot = T,
   normalized = TRUE, )
fitPiecewiseLinreg(m1, plot = T,
   normalized = FALSE)
fitPiecewiseLinreg(m2, plot = T,
   normalized = TRUE)
fitPiecewiseLinreg(m2, plot = T,
   normalized = FALSE)
```



The `flowMerge` package has functionality to automatically identify the changepoint in these plots, which can be used to extract the optimal model. Plots of the different model fits are shown in the vignette.

flowMerge Model Plots

```
par(mfrow = c(2, 2))
plot(m1[[3]])
plot(m2[[3]])
plot(m2[[2]])
```



In this particular case, the different model fits are comparable, except for an extra population to handle outliers. We choose model 2, the 3-cluster model with transformation

flowMerge

Model Selection and Data Extraction

We can identify the optimal merged model from the model 2 series of fits using `fitPiecewiseLinreg` and then extract the gated populations:

```
m <- m2[[fitPiecewiseLinreg(m2,
   normalize = FALSE)]]
fl <- split(f = m)
```

Outlier Threshold

ruleOutliers

We can modify the threshold for outlier detection when plotting or extracting a cell population. The `ruleOutliers` function modifies the outlier detection rule. Most commonly, we'll use a certain quantile of the distribution, beyond which events are considered outliers. The default is the 90% quantile. When we extract the population, we see the number of included events changes based on the outlier threshold.

```
ruleOutliers(m)
```

```
Rule of identifying outliers: 90% quantile
```

```
dim(split(f = m)[[1]])
```

```
    events parameters
       481          8
```

```
ruleOutliers(m) <- list(level = 0.99)
```

```
Rule of identifying outliers: 99% quantile
```

```
dim(split(f = m)[[1]])
```

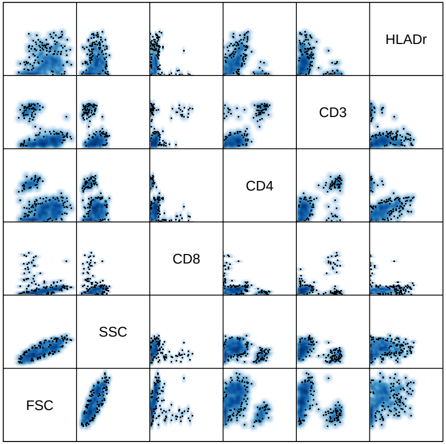
```
    events parameters
       510          8
```

Plotting Subpopulations

Second Stage Clustering

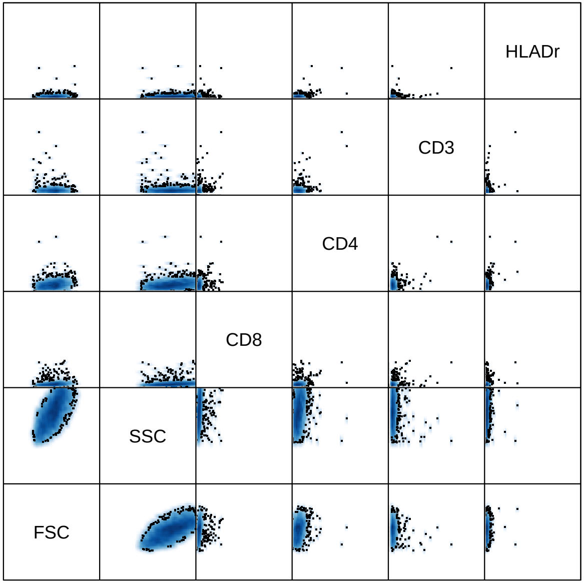
We have extracted the clustered data, and plot subpopulation 1. This is the subpopulation that has CD3+/CD4+ and CD3+/CD8+ cells.

```
splom(fl[[1]][, c(1, 2, 3, 5, 6,
   7)])
```



Scatter Plot Matrix

```
print(splom(fl[[2]][, c(1, 2, 3,
                5, 6, 7)]))
```



Scatter Plot Matrix

Second Stage Clustering

```
fl.trans <- flowClust(fl[[1]],
    varNames = colnames(fl[[1]])[c(3,
         5, 6)], K = 1:10, trans = 1,
    nu = Inf, nu.est = 0)
fl.notrans <- flowClust(fl[[1]],
    varNames = colnames(fl[[1]])[c(3,
         5, 6)], K = 1:10, trans = 0,
    nu = Inf, nu.est = 0)
```

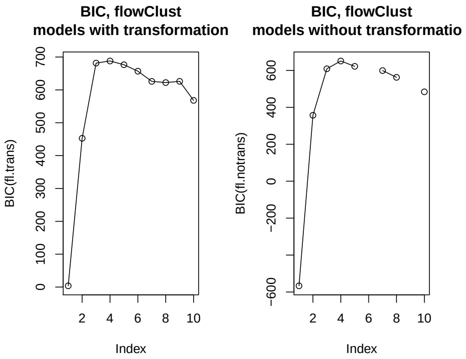
We run the second stage clustering on the CD3/CD4/CD8 fluorescence channels, using gaussian mixtures and *t*–mixtures. Populations are relatively symmetric in the fluorescence channels, so we opt for no transformation.
Again, the data has been pre-computed to save time. You can load it with:

```
data(flowMerge2)
```

Second Stage Clustering

Compare BIC plots

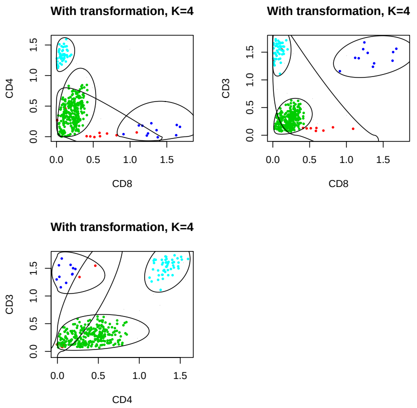
```
par(mfrow = c(1, 2))
plot(BIC(fl.trans), type = "o",
   main = "BIC, flowClust\nmodels with transformation")
plot(BIC(fl.notrans), type = "o",
   main = "BIC, flowClust\nmodels without transformation")
```

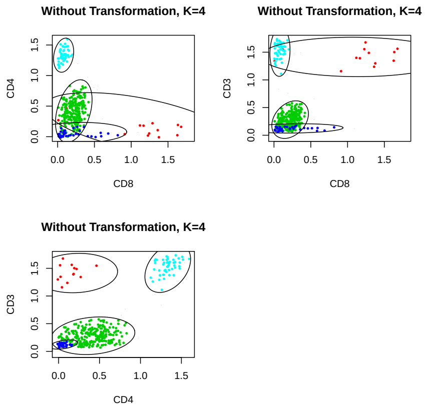


The best fitting models have four clusters, independent of transformation.

Model Comparison

```
plot(flowObj(fl.trans[[4]], fl[[1]]),
   pch = 20, new = T, main = "With transformation, K=4")
plot(flowObj(fl.notrans[[4]], fl[[1]]),
   pch = 20, new = T, main = "Without Transformation, K=4")
```





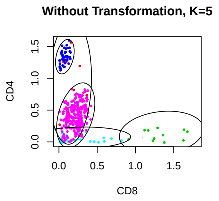
It's interesting to note that the CD3+/CD8+ cell population identified in the model without transformation, and K=4 clusters picks up a couple of CD3+/CD8- events. This is due to the low density nature of this cell population and our use of *t*–mixtures for modeling. We could have a look at the K=5 solution (even though it doesn't have the highest BIC value) to see if we get further resolution of this cell population.

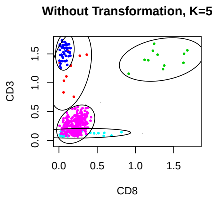
Model Comparison

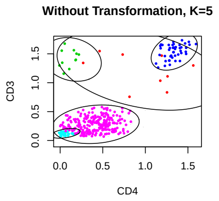
K=5 Solution, With Transformation

```
 plot(flowObj(fl.notrans[[5]], fl[[1]]),
     pch = 20, new = T, main = "Without Transformation, K=5")
```

```
NULL
```







The CD3+/CD8+ and CD3+/CD4+ cell populations are now well defined. There are two additional populations that pick up some dim events in the CD4 and CD8 dimensions.

Subpopulation Selection

Using the estimated population means

We want the CD8+/CD3+ and CD4+/CD3+ subpopulations. We can make use of model parameters estimated by `flowClust` to select these programmatically.

```
fl.2 <- flowObj(fl.notrans[[5]],
    fl[[1]])
ruleOutliers(fl.2) <- list(level = 0.99)
```

```
Rule of identifying outliers: 99% quantile
```

```
order(fl.2@mu[, 1], fl.2@mu[, 3],
    decreasing = TRUE)[1]
```

```
[1] 2
```

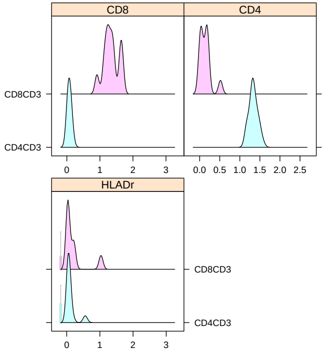
```
order(fl.2@mu[, 2], fl.2@mu[, 3],
    decreasing = TRUE)[1]
```

```
[1] 3
```

- Raise the outlier threshold.
- Use `order` to sort populations on their CD8/CD3, and CD4/CD3 columns.
- The CD8+/CD3+ and CD4+/CD3+ populations are the first in the list returned by the above code.

Subpopulation Selection

```
hladr <- split(x = getData(fl.2),
   f = as(fl.2, "flowClust"),
   population = list(CD8CD3 = order(fl.2@mu[,
       1], fl.2@mu[, 3], decreasing = TRUE)[1],
       CD4CD3 = order(fl.2@mu[,
           2], fl.2@mu[, 3], decreasing = TRUE)[1]))
print(densityplot(~CD8 + CD4 +
   HLADr, as(hladr, "flowSet")))
```



The HLADr+ cells are rare. There's no point running `flowClust` on these populations as there aren't enough data points to fit a good model. We can use the `rangeGate` function from `flowCore` for 1D gating instead.

Gating HLADr+ Cells

RangeGate

```
 hladr.cd3.cd4 <- Subset(hladr[[1]],
    rangeGate(hladr[[1]], stain = "HLADr",
        plot = FALSE, alpha = 0.5,
        filterId = "CD3+CD4+HLAct"))
 hladr.cd3.cd8 <- Subset(hladr[[2]],
    rangeGate(hladr[[2]], stain = "HLADr",
        plot = FALSE, alpha = 0.5,
        filterId = "CD3+CD8+HLAct"))
 100 * dim(hladr.cd3.cd4)[1]/dim((data[[1]]))[1]
```

```
events
 0.025
```

```
 100 * dim(hladr.cd3.cd8)[1]/dim((data[[1]]))[1]
```

```
events
 0.075
```

- 0.02 % of cells are CD3+/CD8+/HLADr+
- 0.05 % of cells are CD4+/CD3+/HLADr+

Now that we've worked out the gating, we'd like to quickly analyze all the samples in the same manner.

Running flowClust in Parallel

The snowfall package

flowClust has support for parallel processing via the snow and snowfall packages. We initialize snowfall by running:

Initialization

```
sfInit(parallel = TRUE, cpus = 4)
```

Parallel flowClust

The following code performs the forward and side scatter gating on the full set of samples. fsApply cycles through the samples in the flowSet. We'll use the parameter settings we decided upon from the first run.

```
result <- fsApply(tData.n, function(x) flowClust(x,
    K = 1:10, trans = 1, nu = 4,
    nu.est = 1, varNames = colnames(x)[1:2]))
sfStop()
```

The data takes some time to process. A pre-computed version has been provided to save time:

```
data(flowMerge3)
```

Next we extract the max BIC solution for each sample, construct flowObj objects, and do the merging.

Running `flowClust` in Parallel

Model Selection

Essentially, we proceed as before, but wrapping calls in `lapply`, and `sapply` statements to loop through the different samples.

```
 result <- lapply(result, function(x) x[[which.max(BIC(x))]])
 result <- sapply(1:length(data),
    function(i) flowObj(result[[i]],
        tData.n[[i]]))
 result <- lapply(result, function(x) {
    m <- merge(x)
    m[[fitPiecewiseLinreg(m)]]
})
```

Semiautomated Gating

```
 indices <- unlist(lapply(result,
    function(x) {
        plot(x)
        z <- unlist(locator(n = 1))
        inc <- which.min(sapply(1:x@K,
            function(i) mahalanobis(box(z,
                lambda = x@lambda),
                x@mu[i, 1:2], x@sigma[i,
                  , ])))
        plot(x, include = inc)
        inc
    }))
```

The code above will allow you to select the lymphocyte population from each gated sample interactively by clicking on the center of the desired population in a plot. This is required because *a priori* we don't know the index of the lymphocyte populations we're interested in.
We load the pre-gated data for this analysis. We then proceed to exract the populations of interest (see vignette for details), and run the next stage of clustering.

The code above will compute the mahalanobis distance between the location you select and each cell population on the transformed scale. The `mahalanobis()` function is provided by the stats package. The Box–Cox transformation `box()` is provided by flowClust, and takes the transformation parameter specific to the sample under consideration as an argument, as well as the data being transformed.
Again, we extract the populations of interest using `split()`, and run the next stage of clustering on the fluorescence channels.

```
result.split <- sapply(1:length(result),
            function(i) split(f = result[[i]])[[indices[i]]
```

Stage 2 Clustering

We'll run `flowClust` with cluster–specific transformation parameters. We won't be able to run `flowMerge` in this case, but it will give us more model flexibility.

```
sfInit(parallel = TRUE, cpus = 2)
result.fl <- lapply(result.split,
   function(x) flowClust(x, B.init = 100,
       K = 1:10, varNames = colnames(x)[c(3,
           5, 6)], nu = Inf, nu.est = 0,
       trans = 2))
sfStop()
```

And, again, we can load the pre-computed data to save time here:

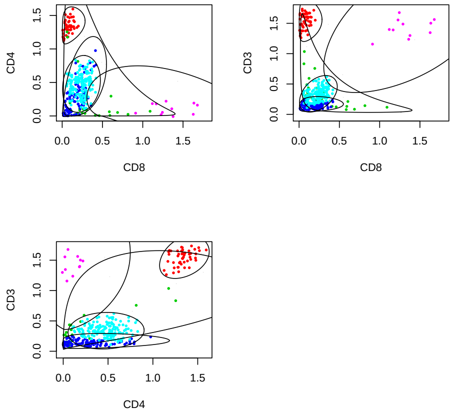
```
data(flowMerge4)
```

```
result.fl.bic <- sapply(1:length(result.fl),
   function(i) flowObj(result.fl[[i]][[which.max(BIC(result.fl[[i]])
       result.split[[i]]))
```

Automated Gating

Recall that the CD8+/CD3+ population is not well defined for sample 1. We manually select a model with five clusters rather than the model with 4 clusters.

```
plot(result.fl.bic[[1]], pch = 20)
result.fl.bic[[1]] <- flowObj(result.fl[[1]][[5]],
   result.split[[1]])
plot(result.fl.bic[[1]], pch = 20)
```



Extracting HLADr+ Population

Extract the CD4+/CD3+ and CD8+/CD3+ populations and look at the HLADr markers.

```
fl.split.cd8 <- sapply(1:length(result.fl.bic),
   function(i) split(x = result.split[[i]],
       f = as(result.fl.bic[[i]],
           "flowClust"), population = list(order(result.fl.bic[[i]]@
           1], result.fl.bic[[i]]@mu[,
           3], decreasing = TRUE)[1])))
fl.split.cd4 <- sapply(1:length(result.fl.bic),
   function(i) split(x = result.split[[i]],
       f = as(result.fl.bic[[i]],
           "flowClust"), population = list(order(result.fl.bic[[i]]@
           2], result.fl.bic[[i]]@mu[,
           3], decreasing = TRUE)[1])))
```

Range Gate

1D Gating of HLADr+ Cells

Finally, we can gate the HLADr positive populations using the rangeGate function again.

```
all.hladr.cd3.cd8 <- lapply(fl.split.cd8,
   function(x) Subset(x, rangeGate(x,
       stain = "HLADr", plot = FALSE,
       borderQuant = 0.5, alpha = 0.5,
       filterId = "CD3+CD8+HLAct")))
all.hladr.cd3.cd4 <- lapply(fl.split.cd4,
   function(x) Subset(x, rangeGate(x,
       stain = "HLADr", plot = FALSE,
       borderQuant = 0.5, alpha = 0.5,
       filterId = "CD3+CD4+HLAct")))
```

Getting Cell Proportions

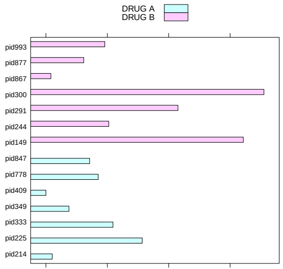
We can get the percentage of cells from the counts.

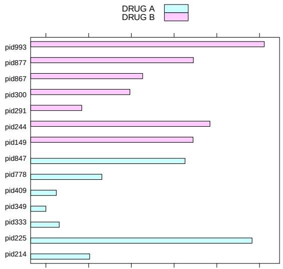
```
CD4.hladr <- sapply(1:length(all.hladr.cd3.cd4),
   function(i) 100 * dim(all.hladr.cd3.cd4[[i]])[1]/dim(data[[i]])[1])
CD8.hladr <- sapply(1:length(all.hladr.cd3.cd8),
   function(i) 100 * dim(all.hladr.cd3.cd8[[i]])[1]/dim(data[[i]])[1])
```

Plotting the Proportions

```
print(barchart(reorder(PatientID,
   as.numeric(factor(GroupID))) ~
   pr, data = data.frame(pr = as.vector(CD4.hladr),
   pData(data)[, c("GroupID",
       "PatientID")]), groups = GroupID,
   auto.key = T))
```

```
print(barchart(reorder(PatientID,
   as.numeric(factor(GroupID))) ~
   pr, data = data.frame(pr = as.vector(CD8.hladr),
   pData(data)[, c("GroupID",
       "PatientID")]), groups = GroupID,
   auto.key = T))
```





Metaclustering

Metaclustering is the term for classifying common cell populations across multiple samples.

flowMetaCluster

- `flowMetaCluster`: a package for metaclustering cell populations from `flowClust` and `flowMerge` models.
- Still under active development, not yet ready for release.
- We provide some examples of its functionality.

Approaches

- Two different algorithms for metaclustering.
- Clustering cell populations based on means and covariances using Mahalanobis distance.
- Clustering cell populations based on entropy due to overlap, analogous to `flowMerge`.

Both algorithms are implemented in the package.

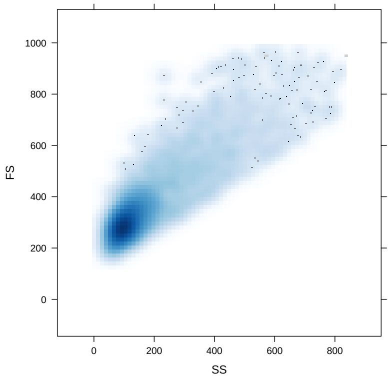
flowMetaCluster

We'll metacluster some lymphoma data for this example. Although the package is not distributed for this workshop, we want to provide some examples of the types of tools that are in the works.

```
require(flowMetaCluster)
require(snowfall)
data(lymphoma)
```

- Load the lymphoma data and required packages.
- Plot the data to examine the distribution. The scatter channels indicate a single population.
- Run `flowClust`/`flowMerge` on the fluorescence channels.

```
print(xyplot(FS ~ SS, lymph[[1]]))
```



The above plot is representative of the data distribution in the forward and side scatter dimensions of this data set. Therefore there is no need to run flowClust on FSC and SSC. We choose the default parameterization of an estimated transformation, and an estimated degrees of freedom, and fit models with one through ten components on the fluorescence channels.

flowMetaCluster

We run `flowClust` on the data, followed by `flowMerge`.

Parallel flowClust

```
result.lymph <- fsApply(lymph,
   function(x) flowClust(x, varNames = colnames(x)[-c(1:2)],
       K = 1:10, nu = 4, trans = 1,
       nu.est = 1))
```

Parallel flowMerge

```
sfInit(parallel = TRUE, cpus = 4)
clusterEvalQ(sfGetCluster(), library(flowMerge))
m <- sfLapply(result.lymph.opt,
   merge)
k <- sfLapply(m, fitPiecewiseLinreg)
sfStop()
m <- sapply(1:length(k), function(i) m[[i]][[k[[i]]]])
```

Metaclustering `flowMerge` objects

With a series of fitted `flowMerge` objects, it is trivial to run the metaclustering algorithms.

```
#Mahalanobis distance metaclustering
meta<-metaCluster(m);
#Entropy-based metaclustering
meta2<-metaClusterByMerging(m);
```

Two different data structures, `meta` and `meta2` store information required to generate metacluster memberships for entropy–based metaclustering and metaclustering based on Mahalanobis distance between populations. The `flowMetaCluser` provides functionality for extracting and plotting the metaclustered cell populations using these structures. More information is provided in the Vignette.

flowMetaCluster Objects

Mahalanobis distance based metaclustering

```
 getSlots("flowMetaCluster") #S4 object

         flow     Indicator
"environment"  "data.frame"
```

Entropy Based Metaclustering

```
 names(meta2); #S3 object

[1] "z"         "adjacency" "entropy"
```

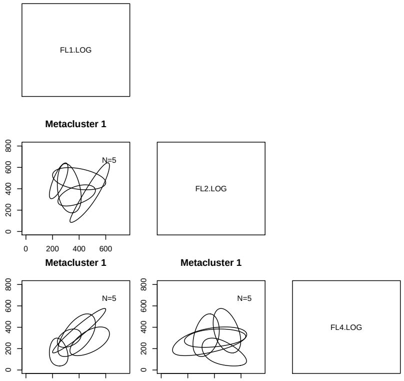
Plotting MetaClusters

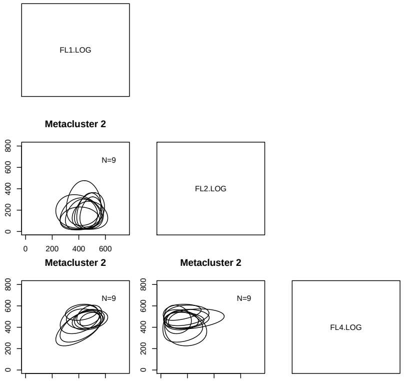
We can plot relevant projections of individual metaclusters:

Plot by Metacluster

```
print(plot(meta, by = "metacluster",
   KK = 1, main = "Metacluster 1"))
```

```
print(plot(meta, by = "metaclus
   KK = 2, main = "Metacluster
```





Plotting MetaClusters

Metaclustering - Helper Methods

There are a number of helper methods for working with the metaclustered data.

```
# Return metacluster membership of each cell population.
metaClusterPopulations(meta)
# Return the transformed or untransformed means of the populatio
getClassMeans(meta)
# Return the list of flowMerge objects being clustered
getFlow(meta)
#Return clustering statistics about each metacluster
getClusterStats(meta)
# Return the distance matrix used for clustering the data
getDist(meta)
```

These methods can be used to extract information about each metacluster for further analysis.

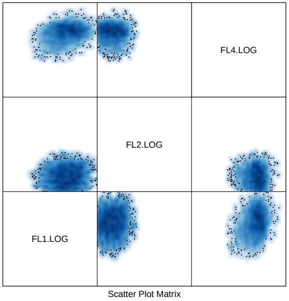
Extracting Metaclustered Populations

Splitting Populations

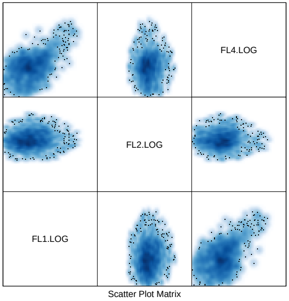
We can use the `split` method to extract individual cell populations within each metacluster. From the plots above, we we want to extract metaclusters two and four. `flowMetaCluster` implements the `split` method for this purpose.

```
metacluster2 <- split(meta, metaK = 2)
metacluster4 <- split(meta, metaK = 4)
```

```
print(splom(metacluster2[[1]][,
   c(3, 4, 5)]))
```



```
print(splom(metacluster4[[1]][,
   c(3, 4, 5)]))
```

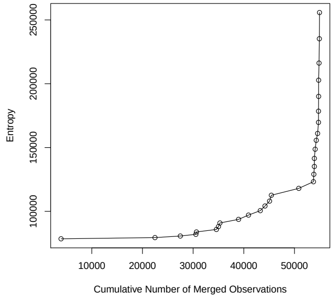


Entropy Based Metaclustering

Entropy Based Metaclustering

We can also have a look at the entropy vs cumulative number of merged observations plot for the entropy based metaclustering.

```
screePlot(meta2)
```



- The entropy vs cumulative number of merged observations identifies when sufficient populations have been merged across samples.
- Localization of the changepoint can be done automatically, as in `flowMerge`.

```
cp <- findChangePoint(screePlot(meta2,
   main = "Entropy vs Cum. # Merged Obs."))
print(cp)
```

```
[1] 16
```

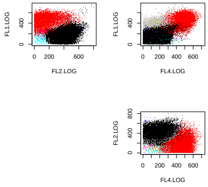
The changepoint corresponds to the model with 16 populations. We can plot this model to see how it looks.

Plotting Merged Metaclusters

Plotting Entropy Based Metaclusters

We can plot the populations identified by entropy based metaclustering.

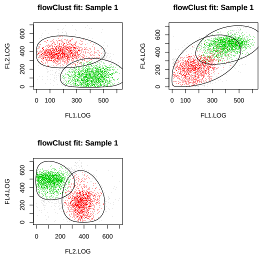
```
plot(meta2)
```



There are four major populations with the remaining 12 populations corresponding to various outlier points.

```
print(plot(m[[1]], pch = ".", new = T,
   main = "flowClust fit: Sample 1"))
```

```
NULL
```



Ongoing Work

Work on the metaclustering package is ongoing. We hope to have development version ready for submission in the near future. We are in the process of developing greater depth and functionality for the package.

Coming Updates

flowClust/flowMerge

- Integrate `flowMerge` more closely with `workFlows`.
- Add explicit parallelization support
- Expand plotting and graphing capabilities
- Implement bayesian priors for rare cell populations.
- ...

flowMetaCluster

- Add support for user provided distance matrix.
- Add more robust support for labeling of cell populatons.
- Entropy-based merging: map metacluster labelled events back to original populations.
- ...

Selected References

- *Merging Mixture Components for Cell Population Identification in Flow Cytometry* Greg Finak, Ali Bashashati, Ryan Brinkman, Raphael Gottardo. (2009) Adv. In Bioinformatics.
- *Automated Gating of Flow Cytometry Data via Robust Model-Based Clustering* Kenneth Lo, Ryan Brinkman, Raphael Gottardo. (2008) Cytometry A.