An Introduction to MAST

Andrew McDavid and Greg Finak

April 20, 2016

1 Philosophy

`MAST`is an R/Bioconductor package for managing and analyzing qPCR and sequencing-based single–cell gene expression data, as well as data from other types of single–cell assays. Our goal is to support assays that have multiple *features* (genes, markers, etc) per *well* (cell, etc) in a flexible manner. Assays are assumed to be mostly *complete* in the sense that most *wells* contain measurements for all features.

1.1 Internals

A `SingleCellAssay` object can be manipulated as a matrix, with rows giving features and columns giving cells. It derives from `SummarizedExperiment`.

1.2 Statistical Testing

Apart from reading and storing single–cell assay data, the package also provides functionality for significance testing of differential expression using a Hurdle model, gene set enrichment, facilities for visualizing patterns in residuals indicative of differential expression, and power calculations (soon).

There is also some facilities for inferring background thresholds, and filtering of individual outlier wells/libraries. These methods are described our papers.

2 Examples

With the cursory background out of the way, we'll proceed with some examples to help understand how the package is used.

2.1 Reading Data

Data can be imported in a Fluidigm instrument-specific format (the details of which are undocumented, and likely subject-to-change) or some derived, annotated format, or in "long" (melted) format, in which each row is a measurement, so if there are N wells and M cells, then the `data.frame` should contain $N \times M$ rows.

For example, the following data set was provided in as a comma-separated value file. It has the cycle threshold (ct) recorded. Non-detected genes are recorded as NAs. For the Fluidigm/qPCR single cell expression functions to work as expected, we must use the *expression threshold*, defined as $et = c_{\max} - ct$, which is proportional to the log-expression.

Below, we load the package and the data, then compute the expression threshold from the ct, and construct a `FluidigmAssay`.

```
#load_all('..')
library(data.table)
library(plyr)
data(vbeta)
colnames(vbeta)
```

```
##  [1] "Sample.ID"         "Subject.ID"        "Experiment.Number"
##  [4] "Chip.Number"       "Stim.Condition"    "Time"
##  [7] "Population"        "Number.of.Cells"   "Well"
## [10] "Gene"              "Ct"
```

```
vbeta <- computeEtFromCt(vbeta)
vbeta.fa <- FromFlatDF(vbeta, idvars=c("Subject.ID", "Chip.Number", "Well"),
                           primerid='Gene', measurement='Et', ncells='Number.of.Cells',
                           geneid="Gene",  cellvars=c('Number.of.Cells', 'Population'),
                           phenovars=c('Stim.Condition','Time'), id='vbeta all', class='FluidigmAssay')
show(vbeta.fa)

## class: FluidigmAssay
## dim: 75 456
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(456): Sub01 1 A01 Sub01 1 A02 ... Sub02 3 H10 Sub02 3 H11
## colData names(9): Number.of.Cells Population ... Time wellKey
```

We see that the variable `vbeta` is a `data.frame` from which we construct the `FluidigmAssay` object. The `idvars` is the set of column(s) in `vbeta` that uniquely identify a well (globally), the `primerid` is a column(s) that specify the feature measured at this well. The `measurement` gives the column name containing the log-expression measurement, `ncells` contains the number of cells (or other normalizing factor) for the well. `geneid`, `cellvars`, `phenovars` all specify additional columns to be included in the `featureData`, `phenoData` and `cellData` (TODO: wellData). The output is a `FluidigmAssay` object with 456 wells and 75 features.

We can access the feature–level metadata and the cell–level metadata using the `mcols` and `colData` accessors.

```
head(mcols(vbeta.fa),3)

## DataFrame with 3 rows and 2 columns
##          Gene    primerid
##   <character> <character>
## 1      B3GAT1      B3GAT1
## 2         BAX         BAX
## 3        BCL2        BCL2

head(colData(vbeta.fa),3)

## DataFrame with 3 rows and 9 columns
##             Number.of.Cells            Population    ncells Subject.ID
##                   <integer>           <character> <integer>   <factor>
## Sub01 1 A01               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A02               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A03               1 CD154+VbetaResponsive         1      Sub01
##             Chip.Number        Well Stim.Condition     Time     wellKey
##               <integer> <character>    <character> <factor> <character>
## Sub01 1 A01           1         A01      Stim(SEB)       12 Sub01 1 A01
## Sub01 1 A02           1         A02      Stim(SEB)       12 Sub01 1 A02
## Sub01 1 A03           1         A03      Stim(SEB)       12 Sub01 1 A03
```

We see this gives us the set of genes measured in the assay, or the cell-level metadata (i.e. the number of cells measured in the well, the population this cell belongs to, the subject it came from, the chip it was run on, the well id, the stimulation it was subjected to, and the timepoint for the experiment this cell was part of). The wellKey are concatenated idvars columns, helping to ensure consistency when splitting and merging `MAST`objects.

2.2 Importing Matrix Data

TBD

2.3 Subsetting, splitting, combining, melting

It's possible to subset `MAST`objects by wells and features. Square brackets ("[") will index on the first index (features) and by features on the second index (cells). Integer and boolean and indices may be used, as well as character vectors naming the wellKey or the feature (via the primerid). There is also a `subset` method, which will evaluate its argument in the frame of the `colData`, hence will subset by wells.

```
sub1 <- vbeta.fa[,1:10]
show(sub1)
```

```
## class: FluidigmAssay
## dim: 75 10
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(10): Sub01 1 A01 Sub01 1 A02 ... Sub01 1 A09 Sub01 1 A10
## colData names(9): Number.of.Cells Population ... Time wellKey
```

```
sub2 <- subset(vbeta.fa, Well=='A01')
show(sub2)
```

```
## class: FluidigmAssay
## dim: 75 5
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(5): Sub01 1 A01 Sub01 2 A01 Sub02 1 A01 Sub02 2 A01 Sub02
##   3 A01
## colData names(9): Number.of.Cells Population ... Time wellKey
```

```
sub3 <- vbeta.fa[6:10, 1:10]
show(sub3)
```

```
## class: FluidigmAssay
## dim: 5 10
## metadata(0):
## assays(1): Et
## rownames(5): CCL4 CCL5 CCR2 CCR4 CCR5
## metadata column names(2): Gene primerid
## colnames(10): Sub01 1 A01 Sub01 1 A02 ... Sub01 1 A09 Sub01 1 A10
## colData names(9): Number.of.Cells Population ... Time wellKey
```

```
colData(sub3)
```

```
## DataFrame with 10 rows and 9 columns
##             Number.of.Cells            Population    ncells Subject.ID
##                   <integer>           <character> <integer>   <factor>
## Sub01 1 A01               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A02               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A03               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A04               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A05               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A06               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A07               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A08               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A09               1 CD154+VbetaResponsive         1      Sub01
```

```
## Sub01 1 A10               1 CD154+VbetaResponsive         1      Sub01
##             Chip.Number        Well Stim.Condition     Time     wellKey
##               <integer> <character>    <character> <factor> <character>
## Sub01 1 A01           1         A01      Stim(SEB)       12 Sub01 1 A01
## Sub01 1 A02           1         A02      Stim(SEB)       12 Sub01 1 A02
## Sub01 1 A03           1         A03      Stim(SEB)       12 Sub01 1 A03
## Sub01 1 A04           1         A04      Stim(SEB)       12 Sub01 1 A04
## Sub01 1 A05           1         A05      Stim(SEB)       12 Sub01 1 A05
## Sub01 1 A06           1         A06      Stim(SEB)       12 Sub01 1 A06
## Sub01 1 A07           1         A07      Stim(SEB)       12 Sub01 1 A07
## Sub01 1 A08           1         A08      Stim(SEB)       12 Sub01 1 A08
## Sub01 1 A09           1         A09      Stim(SEB)       12 Sub01 1 A09
## Sub01 1 A10           1         A10      Stim(SEB)       12 Sub01 1 A10

mcols(sub3)

## DataFrame with 5 rows and 2 columns
##          Gene    primerid
##   <character> <character>
## 1        CCL4        CCL4
## 2        CCL5        CCL5
## 3        CCR2        CCR2
## 4        CCR4        CCR4
## 5        CCR5        CCR5
```

The cellData and featureData `AnnotatedDataFrames` are subset accordingly as well.

A `MAST`may be split into a list of `MAST`, which is known as an `SCASet`. The split method takes an argument which names the column (factor) on which to split the data. Each level of the factor will be placed in its own `MAST`within the SCASet.

```
sp1 <- split(vbeta.fa, 'Subject.ID')
show(sp1)

## $Sub01
## class: FluidigmAssay
## dim: 75 177
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(177): Sub01 1 A01 Sub01 1 A02 ... Sub01 2 H09 Sub01 2 H10
## colData names(9): Number.of.Cells Population ... Time wellKey
##
## $Sub02
## class: FluidigmAssay
## dim: 75 279
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(279): Sub02 1 A01 Sub02 1 A02 ... Sub02 3 H10 Sub02 3 H11
## colData names(9): Number.of.Cells Population ... Time wellKey
```

The splitting variable can either be a character vector naming column(s) of the `MAST`, or may be a `factor` or `list` of `factors`.

It's possible to combine `MAST`objects or an `SCASet` with the `cbind` method.

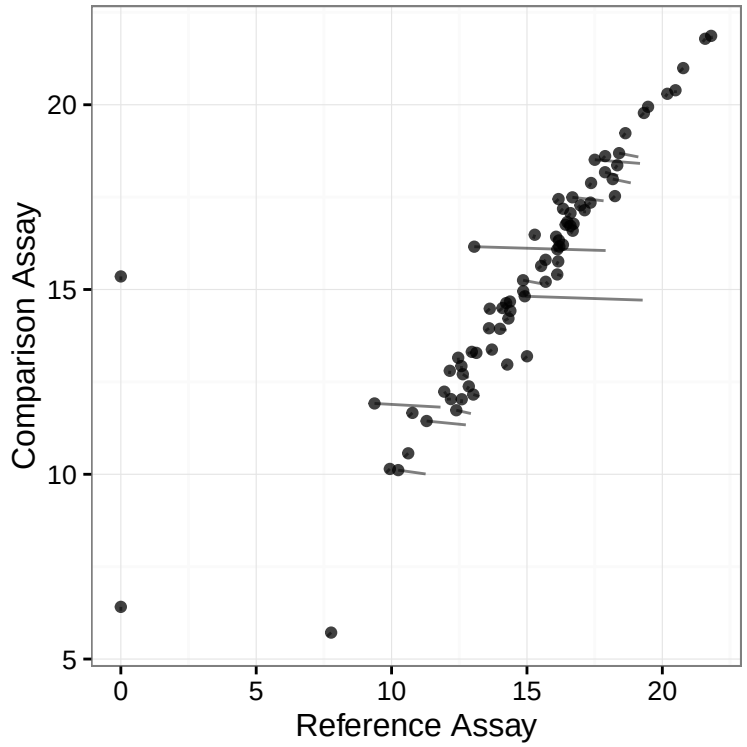
```
## class: FluidigmAssay
## dim: 75 456
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(456): Sub01 1 A01 Sub01 1 A02 ... Sub02 3 H10 Sub02 3 H11
## colData names(9): Number.of.Cells Population ... Time wellKey
```

2.4 Filtering

We can filter and perform some significance tests on the `MAST`. We may want to filter any wells with at least two outlier cells where the discrete and continuous parts of the signal are at least 9 standard deviations from the mean. This is a very conservative filtering criteria. We'll group the filtering by the number of cells.

We'll split the assay by the number of cells and look at the concordance plot after filtering.

```
vbeta.split<-split(vbeta.fa,"Number.of.Cells")
#see default parameters for plotSCAConcordance
plotSCAConcordance(vbeta.split[[1]],vbeta.split[[2]],
                   filterCriteria=list(nOutlier = 1, sigmaContinuous = 9,
                       sigmaProportion = 9))

## Sum of Squares before Filtering:  14.95
##  After filtering:  12.4
##  Difference:  2.54
```



The filtering function has several other options, including whether the filter shuld be applied (thus returning a new SingleCellAssay object) or returned as a matrix of boolean values.

```
vbeta.fa

## class: FluidigmAssay
```

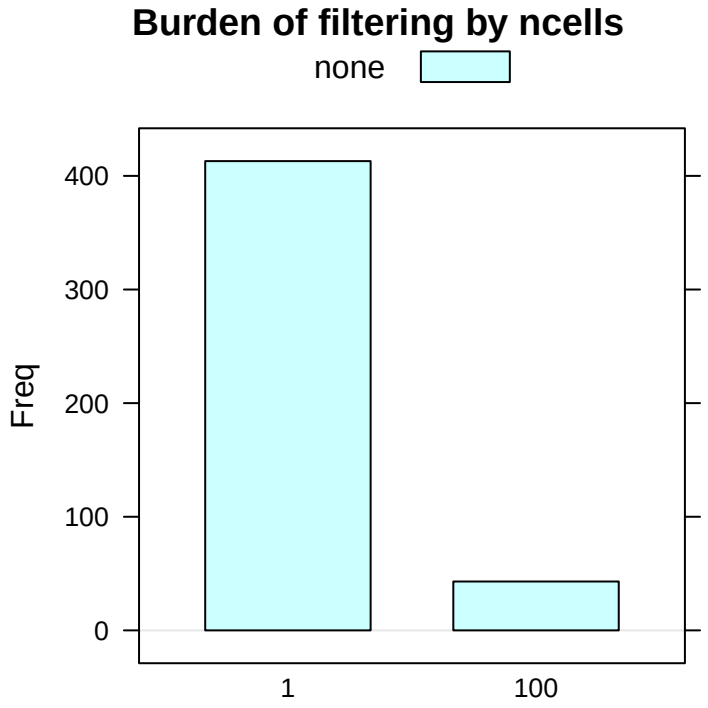
```
## dim: 75 456
## metadata(0):
## assays(1): Et
## rownames(75): B3GAT1 BAX ... TNFRSF9 TNFSF10
## metadata column names(2): Gene primerid
## colnames(456): Sub01 1 A01 Sub01 1 A02 ... Sub02 3 H10 Sub02 3 H11
## colData names(9): Number.of.Cells Population ... Time wellKey
```

```
## Split by 'ncells', apply to each component, then recombine
vbeta.filtered <- filter(vbeta.fa, groups='ncells')
## Returned as boolean matrix
was.filtered <- filter(vbeta.fa, apply_filter=FALSE)
## Wells filtered for being discrete outliers
head(subset(was.filtered, pctout))
```

```
##              intout null pctout
## Sub01 1 D05  FALSE TRUE   TRUE
## Sub01 1 D06  FALSE TRUE   TRUE
## Sub01 1 D07  FALSE TRUE   TRUE
## Sub01 1 D08  FALSE TRUE   TRUE
## Sub01 1 D10  FALSE TRUE   TRUE
## Sub01 1 D11  FALSE TRUE   TRUE
```

There's also some functionality for visualizing the filtering.

```
burdenOfFiltering(vbeta.fa, 'ncells', byGroup=TRUE)
```



2.5 Thresholding

TBD

3 Significance testing under the Hurdle model

There are two frameworks available in the package. The first framework `zlm` offers a full linear model to allow arbitrary comparisons and adjustment for covariates. The second framework `LRT` can be considered essentially performing t-tests (respecting the discrete/continuous nature of the data) between pairs of groups. `LRT` is subsumed by the first framework, but might be simpler for some users, so has been kept in the package.

We'll describe `zlm`. Models are specified in terms of the variable used as the measure and covariates present in the `cellData` using symbolic notation, just as the `lm` function in R.

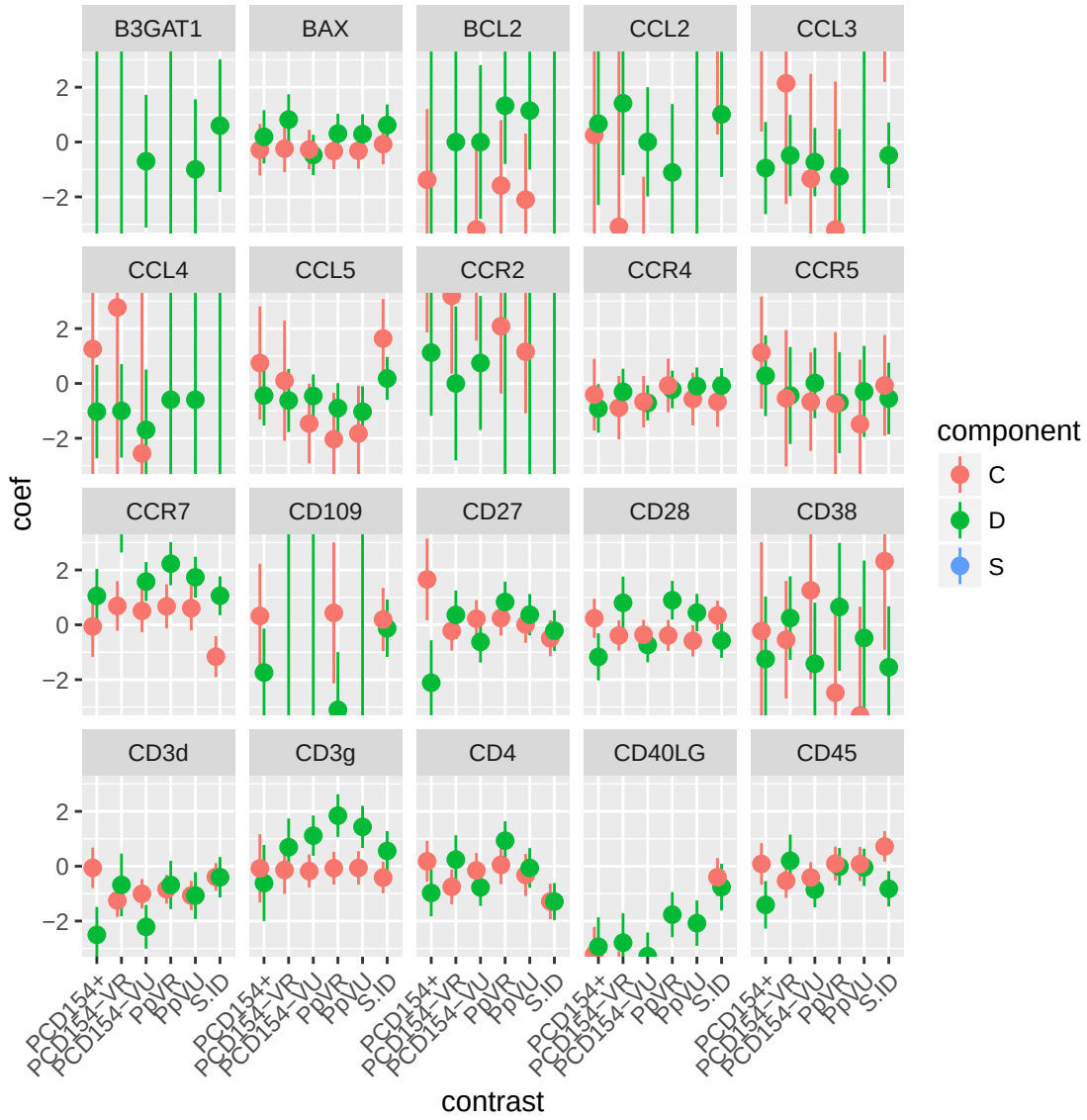
```
vbeta.1 <- subset(vbeta.fa, ncells==1)
## Consider the first 20 genes
vbeta.1 <- vbeta.1[1:20,]
head(colData(vbeta.1))

## DataFrame with 6 rows and 9 columns
##             Number.of.Cells            Population    ncells Subject.ID
##                   <integer>           <character> <integer>   <factor>
## Sub01 1 A01               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A02               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A03               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A04               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A05               1 CD154+VbetaResponsive         1      Sub01
## Sub01 1 A06               1 CD154+VbetaResponsive         1      Sub01
##             Chip.Number        Well Stim.Condition     Time     wellKey
##               <integer> <character>    <character> <factor> <character>
## Sub01 1 A01           1         A01      Stim(SEB)       12 Sub01 1 A01
## Sub01 1 A02           1         A02      Stim(SEB)       12 Sub01 1 A02
## Sub01 1 A03           1         A03      Stim(SEB)       12 Sub01 1 A03
## Sub01 1 A04           1         A04      Stim(SEB)       12 Sub01 1 A04
## Sub01 1 A05           1         A05      Stim(SEB)       12 Sub01 1 A05
## Sub01 1 A06           1         A06      Stim(SEB)       12 Sub01 1 A06
```

Now, for each gene, we can regress on `Et` the factors `Population` and `Subject.ID`.

In each gene, we'll fit a Hurdle model with a separate intercept for each population and subject. A an S4 object of class "ZlmFit" is returned, containing slots with the genewise coefficients, variance-covariance matrices, etc.

```
library(ggplot2)
library(reshape)
library(abind)
zlm.output <- zlm.SingleCellAssay(~ Population + Subject.ID, vbeta.1, method='glm',
                                  ebayes=TRUE)
show(zlm.output)

## Fitted zlm on 413 genes and 20 cells.
##  Using GLMlike ~ Population + Subject.ID

## returns a data.table with a summary of the fit
coefAndCI <- summary(zlm.output, logFC=FALSE)
coefAndCI <- coefAndCI[contrast != '(Intercept)',]
coefAndCI[,contrast:=abbreviate(contrast)]

## Fitted zlm with top 2 genes per contrast:
## ( Wald Z-scores on discrete )
##  primerid PCD154+ PCD154-VR PCD154-VU PpVR    PpVU    S.ID
##  CCR7        2.1     6.3*      4.3       5.5*    4.6*    2.9*
##  CD3d       -4.8*   -1.2      -5.4*     -1.5    -2.5    -1.1
##  CD3g       -0.9     1.3       3.0       4.7*    3.7     1.5
##  CD4        -2.2     0.5      -2.2       2.6    -0.2    -3.7*
##  CD40LG     -5.4*   -5.1*     -7.5*     -4.2    -4.9*   -1.8
```

```
ggplot(coefAndCI, aes(x=contrast, y=coef, ymin=ci.lo, ymax=ci.hi, col=component))+
    geom_pointrange(position=position_dodge(width=.5)) +facet_wrap(~primerid) +
    theme(axis.text.x=element_text(angle=45, hjust=1)) + coord_cartesian(ylim=c(-3, 3))
```



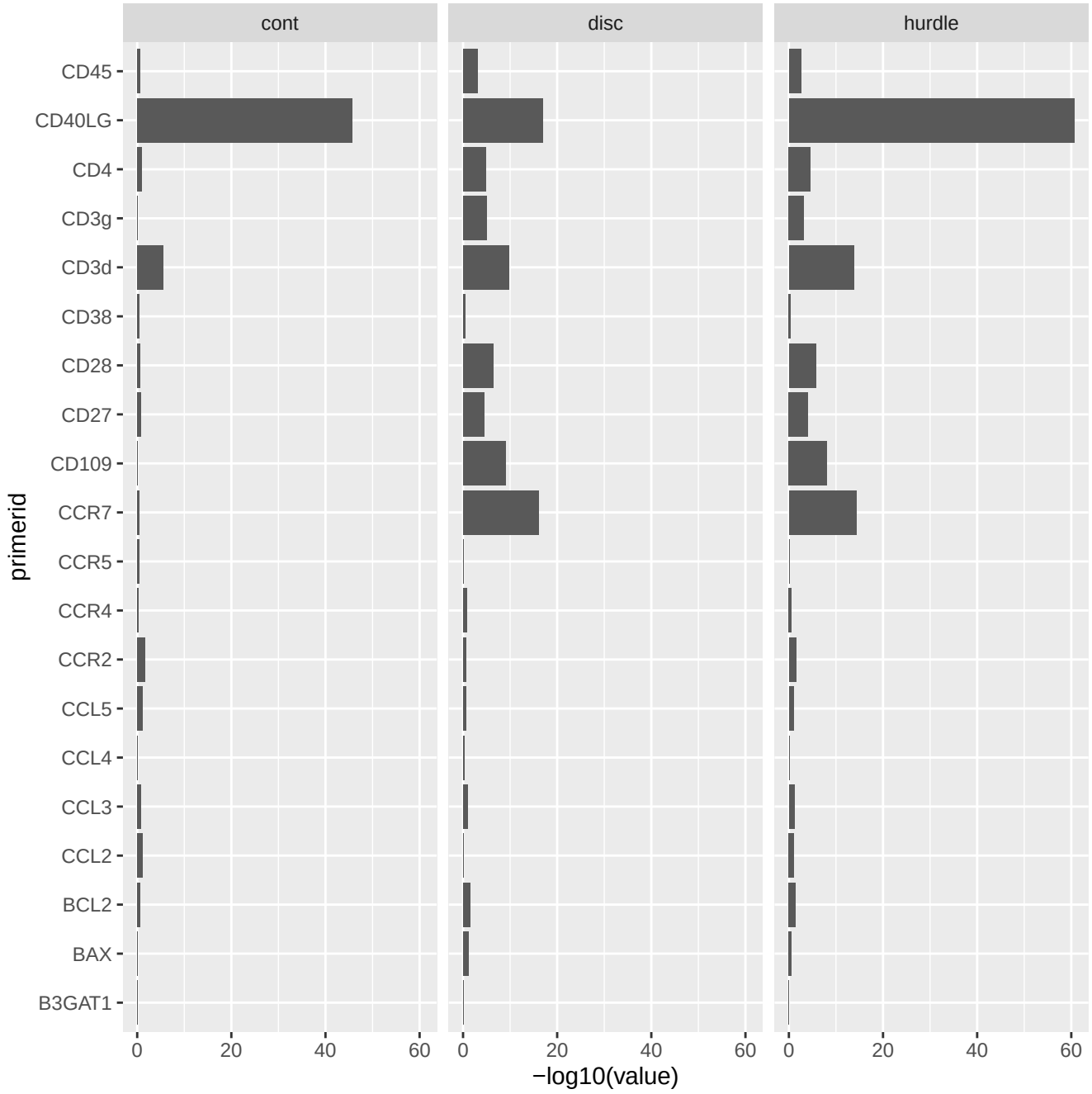
Try `?ZlmFit-class` or `showMethods(classes='ZlmFit')` to see a full list of methods.

The combined test for differences in proportion expression/average expression is found by calling a likelihood ratio test on the fitted object. An array of genes, metrics and test types is returned. We'll plot the -log10 P values by gene and test type.

```
zlm.lr <- lrTest(zlm.output, 'Population')

## Refitting on reduced model...
## .
##
## Done!
## Warning in fData(zlmfit@sca):  Deprecated:  use mcols

dimnames(zlm.lr)

## $primerid
##  [1] "B3GAT1" "BAX"    "BCL2"   "CCL2"   "CCL3"   "CCL4"   "CCL5"
```

```
##  [8] "CCR2"   "CCR4"   "CCR5"   "CCR7"   "CD109"  "CD27"   "CD28"
## [15] "CD38"   "CD3d"   "CD3g"   "CD4"    "CD40LG" "CD45"
##
## $test.type
## [1] "cont"   "disc"   "hurdle"
##
## $metric
## [1] "lambda"     "df"         "Pr(>Chisq)"
```

```
pvalue <- ggplot(melt(zlm.lr[,,'Pr(>Chisq)']), aes(x=primerid, y=-log10(value)))+
    geom_bar(stat='identity')+facet_wrap(~test.type) + coord_flip()
print(pvalue)
```



In fact, the `zlm` framework is quite general, and has wrappers for a variety of modeling functions that accept `glm`-like

arguments to be used, such as mixed models (using `lme4`) and Bayesian regression models (using `arm`). Multicore support is offered by setting `options(mc.cores=4)`, or however many cores your system has.

```
library(lme4)

lmer.output <- zlm.SingleCellAssay(~ Stim.Condition +(1|Subject.ID), vbeta.1,
                                   method='glmer')
```

3.1 Two-sample Likelihood Ratio Test

Another way to test for differential expression is available through the `LRT` function, which is analogous to two-sample T tests.

```
two.sample <- LRT(vbeta.1, 'Population', referent='CD154+VbetaResponsive')
car::some(two.sample)
```

```
##                 Population test.type primerid direction    lrstat
## 8  CD154+VbetaUnresponsive      comb     CCR2         1 14.698609
## 12 CD154+VbetaUnresponsive      comb    CD109        -1  6.738768
## 13 CD154+VbetaUnresponsive      comb     CD27        -1 16.336193
## 17 CD154+VbetaUnresponsive      comb     CD3g        -1  2.323000
## 45 CD154-VbetaUnresponsive      comb     CCL3        -1  1.469168
## 69         VbetaResponsive      comb     CCR4        -1  1.730697
## 74         VbetaResponsive      comb     CD28         1  4.262893
## 78         VbetaResponsive      comb      CD4        -1 14.061485
## 80         VbetaResponsive      comb     CD45         1  6.355646
## 85       VbetaUnresponsive      comb     CCL3        -1 11.895953
##         p.value
## 8  0.0006430396
## 12 0.0344108287
## 13 0.0002835572
## 17 0.3130163494
## 45 0.4797050789
## 69 0.4209049170
## 74 0.1186654966
## 78 0.0008842751
## 80 0.0416762891
## 85 0.0026111182
```

Here we compare each population (CD154-VbetaResponsive, CD154+VbetaUnresponsive, CD154-VbetaUnresponsive, VbetaResponsive, VbetaUnresponsive) to CD154+VbetaResponsive. The `Population` column shows which population is being compared, while `test.type` is `comb` for the combined normal theory/binomial test. Column `primerid` gives the gene being tested, `direction` shows if the comparison group mean is greater (1) or less (-1) than the referent group, and `lrtstat` and `p.value` give the test statistic and χ^2 p-value (two degrees of freedom).

Other options are whether additional information about the tests are returned (`returnall=TRUE`) and if the testing should be stratified by a character vector naming columns in `colData` containing grouping variables (`groups`).

These tests have been subsumed by `zlm.SingleCellAssay` but remain in the package for user convenience.

4 Use with single cell RNA-sequencing data

In RNA-sequencing data is essentially no different than qPCR-based single cell gene expression, once it has been aligned and mapped, if one is willing to reduce the experiment to counts or count-like data for a fixed set of genes/features. We assume that suitable tools (eg, SAMseq or TopHat) have been applied to do this.

More details on convenience functions for RNAseq data

5 Implementation Details

Here we provide some background on the implementation of the package.

There are several fundamental new object types provided by the package. `SummarizedExperiment` is the base class, which is provides an array-like object to store tabular data that might have multiple derived representations. A `SingleCellAssay` object contains a `DataLayer`, plus cell and feature data. New types of single cell assays can be incorportated by extending `SingleCellAssay`.

Different derived classes of `MAST`require different fields to be present in the `cellData` and `featureData` These requirements are set for each class by the slots `cmap` and `fmap`, giving required columns in cell and feature data, respectively.

We have found it useful to enforce naming conventions to reduce confusion when combining data across projects, so the constructor will rename the fields the user provides to match the values specifed in `cmap` and `fmap`.

Sets of single cell assays are stored in the `SCASet` class. A constructor for SCASet is provided to construct an SCASet directly from a data frame. Alternatively, a SingleCellAssay or derived class can be `split` on an arbitray variable to produce an SCASet.

On construction of a `SingleCellAssay` object, the package tests for completeness, and will fill in the missing data (with NA) if it is not, so assays with lots of missing data can make reading marginally slower.