Semi-Supervised Nonnegative Matrix Factorization for Gene Expression Deconvolution: a Case Study

- *Data Processing and Sample Analysis* -

Renaud Gaujoux

August 30, 2011

Abstract

This document comes as a vignette with the R package `NMFged` to describe step by step the analysis presented in:

Semi-Supervised Nonnegative Matrix Factorization for Gene Expression Deconvolution: a Case Study
Renaud Gaujoux, Cathal Seoighe (2011)
Journal of MEEGID - `http://www.elsevier.com/`

This package provides the methods and functions used in the paper, building upon the package NMF, to adapt standard NMF algorithms to incorporate prior knowledge as marker gene into the fitting process. This significantly improves the estimation of the relative cell-type proportions and the meaningfulness of the cell-type specific signatures. We refer to the paper for more details on this semi-supervised approach.

The package `NMFged` is available at `http://web.cbio.uct.ac.za/~renaud/paper/meegid-deconvolution`.

Contents

1 Installation & Requirements

Beside depending on packages available from the standard repositories like CRAN or Bioconductor, the package `NMFged` requires:

- the package `NMF` version 0.5.99 or higher to be installed. Version 0.5.99 is actually a development version that was not released on CRAN, but is available from `http://web.cbio.uct.ac.za/~renaud/paper/meegid-deconvolution`. This version will be superseeded by version 0.6, soon available from CRAN.
- the package `deconf` available in supplementary data from Repsilber et al. [2].

Installing the package `NMFged` can be done the usual way from the R console. To install and all its dependencies type in the R console:

```
# install annotation package from Bioconductor
source("http://bioconductor.org/biocLite.R")
biocLite("hgu133plus2.db")
# install NMFged
install.packages('NMFged', repos=c("@CRAN@", "http://web.cbio.uct.ac.za/~renaud/paper/meegid-
deconvolution"))
```

This will download and install all the required packages from the respective repositories. The package is then loaded by the standard call:

```
library(NMFged)
```

2 Data

The semi-supervised approach is applied to the GEO dataset GSE11058, which can be loaded directly into R from GEO using the Bioconductor package `GEOquery`. Instead of using GEO's platform definition, we use Bioconductor's relevant annotation package.

```
library(GEOquery)
## LOAD ORIGINAL DATA FROM GEO (ACCESSION: GSE11058)
GSE11058 <- getGEO('GSE11058')[[1]]
annotation(GSE11058) <- 'hgu133plus2.db'
```

The dataset is stored in the R session as an `ExpressionSet` object:

```
GSE11058

ExpressionSet (storageMode: lockedEnvironment)
assayData: 54675 features, 24 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: GSM279589 GSM279590 ... GSM279612 (24 total)
  varLabels: title geo_accession ... data_row_count (33 total)
  varMetadata: labelDescription
featureData
  featureNames: 1007_s_at 1053_at ... AFFX-TrpnX-M_at (54675 total)
  fvarLabels: ID GB_ACC ... Gene.Ontology.Molecular.Function (16 total)
  fvarMetadata: Column Description labelDescription
experimentData: use 'experimentData(object)'
Annotation: hgu133plus2.db
```

We then remove all Affymetrix control probesets, using the function `featureFilter` from the Bioconductor package `genefilter`, which has a default argument `feature.exclude="^AFFX"`.

```
library(genefilter)
# remove affy control porbesets, i.e. those with IDs starting with "AFFX"
GSE11058 <- featureFilter(GSE11058, require.entrez=FALSE, re-
move.dupEntrez=FALSE)
dim(GSE11058)

Features  Samples
   54613       24
```

2.1 Reference NMF model

We create the reference NMF model, that contains the ”true” cell-type specific signature and mixing proportions, by using the functions `GSE11058.basis` and `GSE11058.coef`, which build the respective basis and mixture coefficient matrices.

```
# build the "true" NMF model
model <- nmfModel(W=GSE11058.basis(GSE11058), H=GSE11058.coef(GSE11058))
model
```

```
<Object of class: NMFstd >
features: 54613
basis/rank: 4
samples: 24
```

2.2 Curated dataset

Build the curated dataset using the probesets selected by Abbas et al. [1]. The curated dataset is limited to the 359 probesets that compose the final basis matrix used in [1] to deconvolve white blood cell samples from Systemic Lupus Erythematosus patients, and consists of a set of marker probesets for common immune cell subsets in different states (e.g. resting or activated). The probeset IDs are available as supplementary data of [1], and shipped as data `curated.probes` with the package `NMFged`:

```
## CURATED DATASET
# load the probeset IDs
data(curated.probes)
length(curated.probes)
```

```
[1] 359
```

```
# subset the original dataset
curated.dataset <- GSE11058[curated.probes, ]
dim(curated.dataset)
```

```
Features  Samples
     359       24
```

The reference model for the curated dataset is a subset of the complete reference model:

```
curated.model <- model[curated.probes,]
curated.model
```

```
<Object of class: NMFstd >
features: 359
basis/rank: 4
samples: 24
```

2.3 Full dataset

The full dataset is composed of the probesets that can be mapped to an Entrez Gene identifier. This is achieved by another call to the function `featureFilter`.

```
full.dataset <- featureFilter(GSE11058, require.entrez=TRUE, re-
move.dupEntrez=FALSE)
dim(full.dataset)

Features  Samples
   40791       24
```

The reference model for the full dataset is a subset of the complete reference model:

```
full.model <- model[featureNames(full.dataset),]
full.model

<Object of class: NMFstd >
features: 40791
basis/rank: 4
samples: 24
```

3 Markers

3.1 Parameters

The parameters that influence the marker selection are the following:

```
#line 101 "../../NMFged/inst/doc/markers.Rnw#from line#273#"
# compute p-values on log2 transformed data?
log2.transform <- TRUE
# p-value threshold (upper bound)
p.threshold <- 0.05
# fold-change threshold (lower bound in linear or log2 unit)
f.threshold <- 1.5
# maximum number of markers per cell-type
max.markers <- 300
# use equal variance in t-test
var.equal <- TRUE
```

3.2 Marker extraction

We extract the marker probesets from the pure samples of the complete dataset, using the function `extractMarkers` provided by `NMFged`. The pure samples are the 12 first samples in the dataset and their type is stored in the label variable `'characteristics_ch1'`.

```
## EXTRACT MARKERS
# limit to the pure cell-types, i.e. the 12 first ones
pure.idx <- 1:12
pure <- GSE11058[, pure.idx]
# the cell-types are stored in covariate 'characteristics_ch1'
types <- factor(pure$characteristics_ch1, levels=unique(pure$characteristics_ch1))

types
```

```
    V2     V3     V4     V5     V6     V7     V8     V9    V10    V11    V12
Jurkat Jurkat Jurkat   IM-9   IM-9   IM-9   Raji   Raji   Raji  THP-1  THP-1
   V13
 THP-1
Levels: Jurkat IM-9 Raji THP-1
```

The computation of the p-value for each probesets is relatively long that could certainly be speeded up, but needs to run only once. The result of the computation is a numeric matrix containing p-values and $-\log_2-$ fold changes.

```
# compute discriminative power of each probesets
stats <- extractMarkers(pure, types, log2.transform=log2.transform,
var.equal=var.equal)
# compute qvalues separately for each cell-type and add them to the stats
library(qvalue)
ct.all <- markerStatstoList(stats, types, NULL)
qv <- lapply(ct.all, function(x){ q <- qvalue(x); q$qvalue })
qv <- unlist(setNames(qv, NULL))
stats <- cbind(stats, qvalue=qv[rownames(stats)])
head(stats)
```

```
           top     pvalue2       diff2      mdiff2    fold2    mfold2     qvalue
1007_s_at    3 0.001493108 2.115652914  1.48741306 1.252720 1.1677127 0.02724466
1053_at      2 0.014378392 0.250636015  0.08510588 1.022862 1.0077094 0.11432608
117_at       2 0.050405088 0.651941999  0.27167760 1.085164 1.0350211 0.24813765
121_at       4 0.770748000 0.067581962 -0.40547915 1.006337 0.9627342 0.65150797
1255_g_at    1 0.997026778 0.003676312 -2.22348664 1.000802 0.6175928 0.82904942
1294_at      4 0.047553581 0.907201229  0.11774957 1.099196 1.0121332 0.14162825
```

We then apply a filter on the p-value and fold change, to retain only the probesets that pass the criterion defined in section 3.1. If the p-value were computed on $\log_2$ transformed data, then the fold-change values are given by the column `'diff2'`, and by the column `'fold2'` otherwise.

```
# filter probesets on pvalue
sel <- stats[stats[,'pvalue2'] <= p.threshold, ]
# filter probesets on fold change
fold.var <- if( log2.transform ) 'diff2' else 'fold2'
if( !is.na(f.threshold) )
        sel <- sel[sel[,fold.var] >= f.threshold, ]
```

The selected probesets are used to generate a list of markers, ranked by increasing p-value. The result is a `MarkerList` object, which is in this case a list whose elements are numeric vectors of p-values, named by the respective probeset IDs. The `summary` method for this class of objects gives the number of markers used per cell-type.

```
# generate complete marker list
markers <- markerStatstoList(sel, types, 'pvalue2')
summary(markers)
```

```
<object of class MarkerList>
Length: 4
Markers: 3492
Types:
Jurkat   IM-9   Raji  THP-1
   842    678    530   1442
```

3.3 Full dataset

The `subset` method allows to filter the markers that are present in a reference set, such as the features in an `ExpressionSet` objects. We use it to create a `MarkerList` object containing the markers present in the full dataset.

```
# only take the top markers
full.markers <- subset(markers, full.dataset, lim=max.markers)
summary(full.markers)
```

```
<object of class MarkerList>
Length: 4
Markers: 1200
Types:
Jurkat   IM-9   Raji  THP-1
   300    300    300    300
```

```
# check the qvalues
sapply(full.markers, function(x){ max(stats[names(x), 'qvalue']) })
```

```
     Jurkat        IM-9        Raji       THP-1
0.028277136 0.057891710 0.061530299 0.003984319
```

```
# check the minimum mean fold change
sapply(full.markers, function(x){ min(stats[names(x), fold.var]) })
```

```
  Jurkat     IM-9     Raji    THP-1
1.501352 1.506539 1.501033 1.509159
```

The function `plotMarkers` shows how the expression values of the marker porbesets on their respective cell-types compare to the other cell-types. The resulting plot is shown in Figure 1.

The function `markermap` draws a heatmap that annotates the position of each markers. Figure 2 shows such heatmap of the pure samples, limited to the marker probesets.

```
# plot the markers
plotMarkers(full.markers, pure, types, cex=0.5)
```

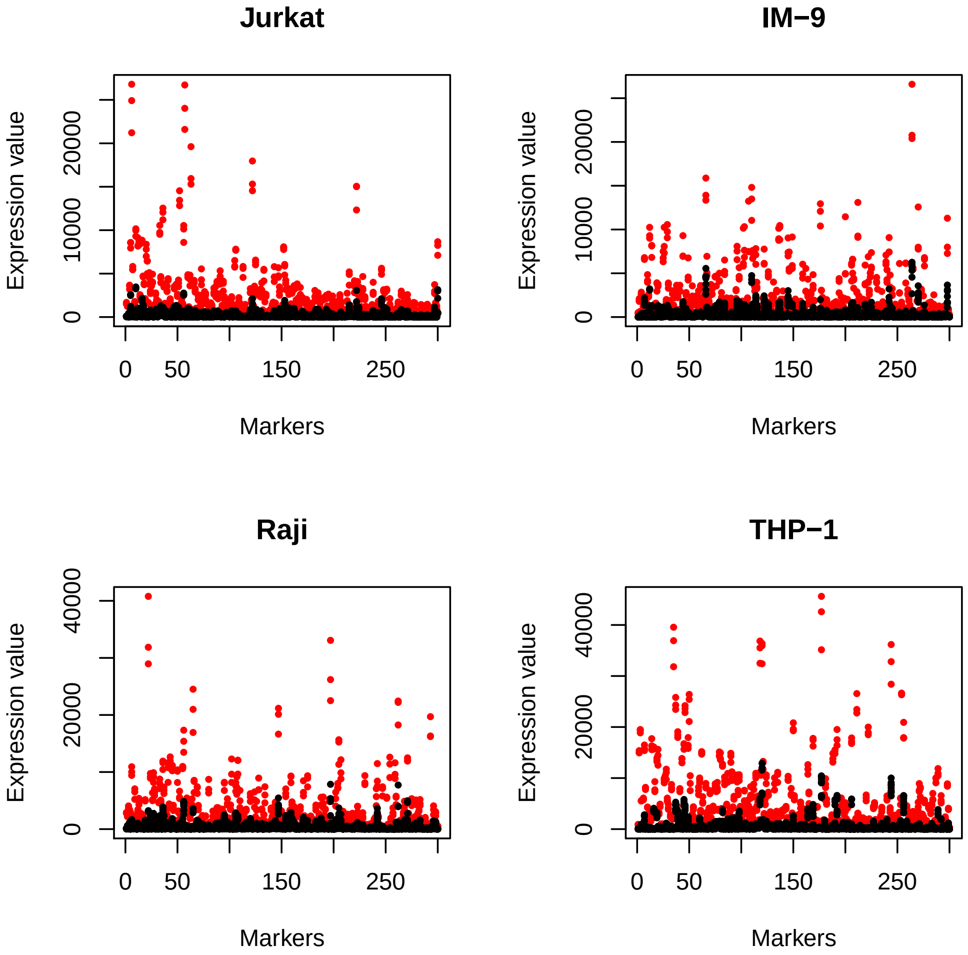

Figure 1: Full dataset: expression values for marker probesets for each cell-type. The markers are ordered by increasing p-values. The red dots show the expression values of the markers on their respective cell-type, the black dots their expression on the three other cell-types.

```
# plot a heatmap of the pure samples (limit to the markers)
markermap(full.markers, full.dataset[, pure.idx], types, show_row=FALSE,
limit=TRUE)
```

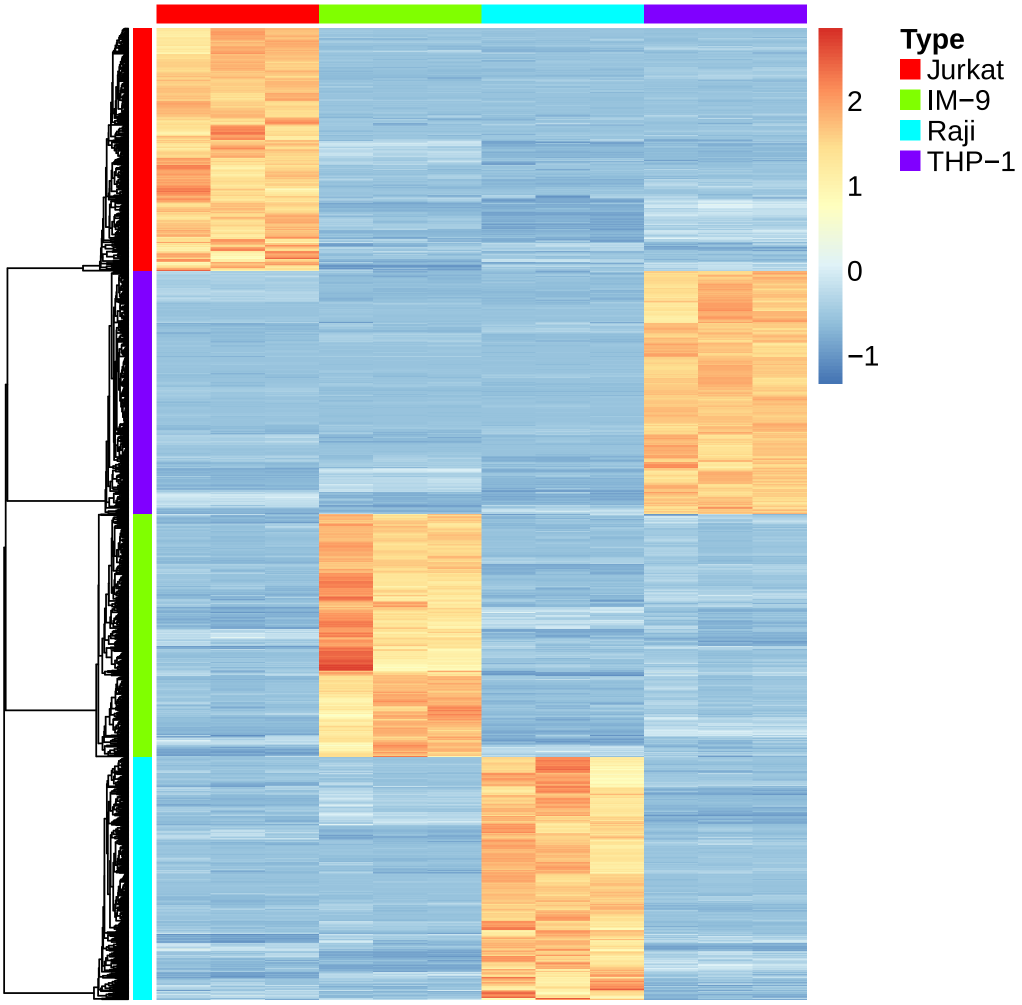

Figure 2: Full dataset: annotated heatmap of the marker probesets for the 12 pure samples.

3.4 Curated dataset

Similarly we subset the full set of markers against the curated dataset and plot their expression values on in Figure 3, as well as annotated heatmaps on Figures 4 and 5.

```
# filter the markers that are present in the curated dataset
curated.markers <- subset(markers, curated.dataset)
summary(curated.markers)
```

```
<object of class MarkerList>
Length: 4
Markers: 84
Types:
Jurkat   IM-9   Raji  THP-1
    21     17     20     26
```

```
# check the minimum mean fold change
sapply(curated.markers, function(x){ min(stats[names(x), fold.var]) })
```

```
  Jurkat     IM-9     Raji    THP-1
1.587066 1.807979 1.598831 1.509159
```

An annotated heatmap can also be generated for the reference NMF model, using the function `markermap` that has a method for `NMF` objects. This latter generates a heatmap of the basis matrix as shown in Figure 6.

```
# plot the markers
plotMarkers(curated.markers, pure, types, cex=0.7)
```

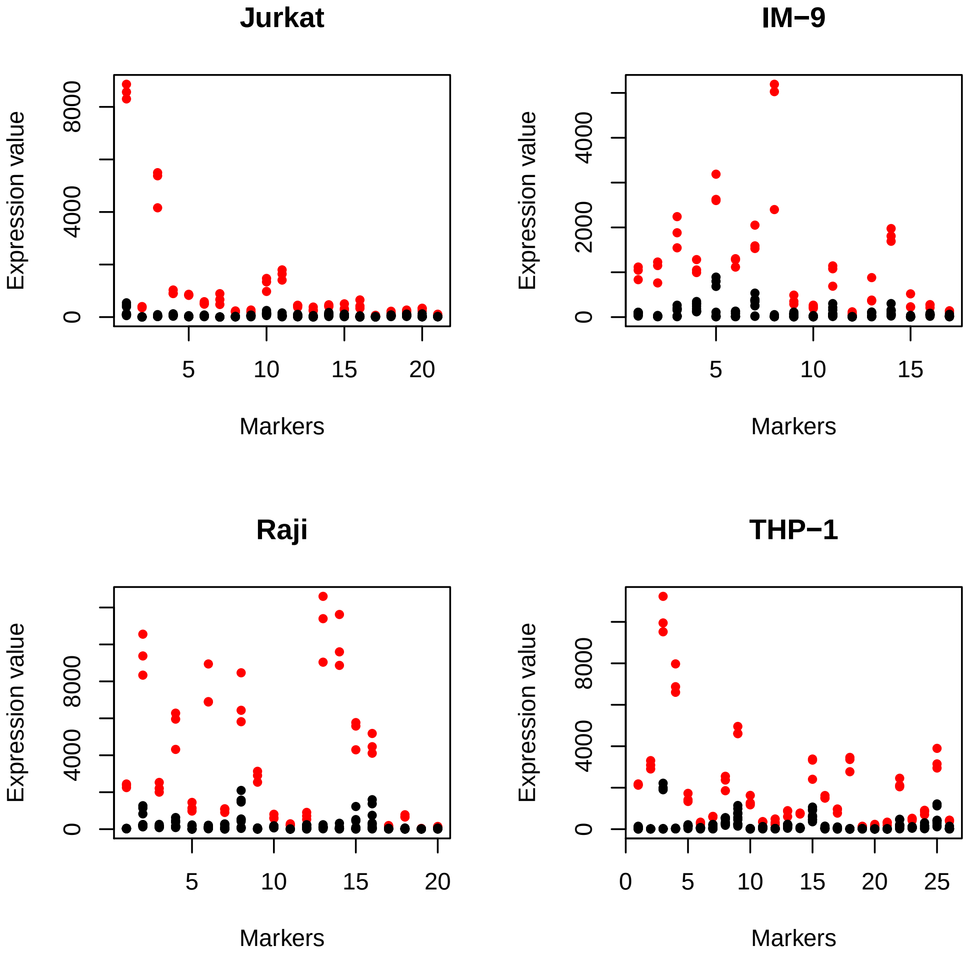

Figure 3: Curated dataset: expression values for marker probesets for each cell-type. The markers are ordered by increasing p-values. The red dots show the expression values of the markers on their respective cell-type, the black dots their expression on the three other cell-types.

```
# plot a heatmap of the curated pure samples (limit to the markers)
markermap(curated.markers, curated.dataset[, pure.idx], types, limit=TRUE,
cex=0.5)
```

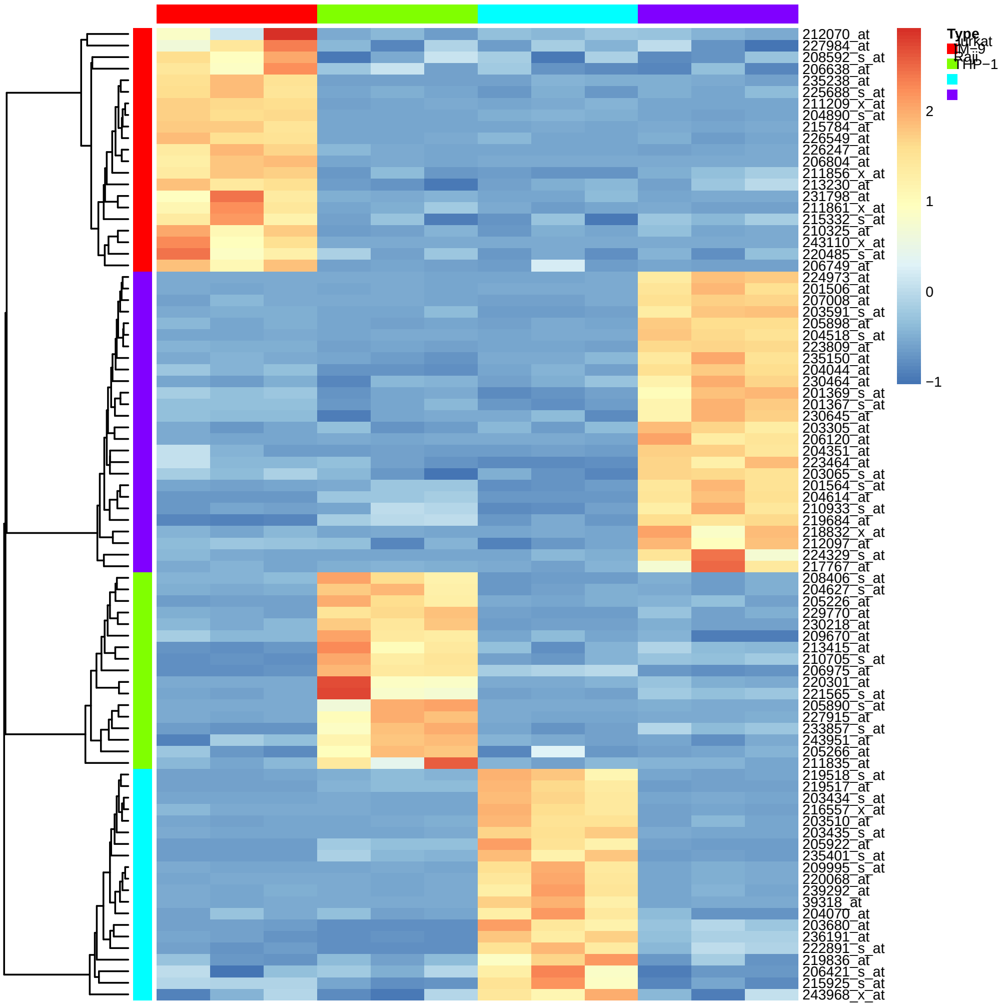

Figure 4: Curated dataset: annotated heatmap of the marker probesets for the 12 pure samples.

```
markermap(curated.markers, curated.dataset[, pure.idx], types, show_row=FALSE)
```

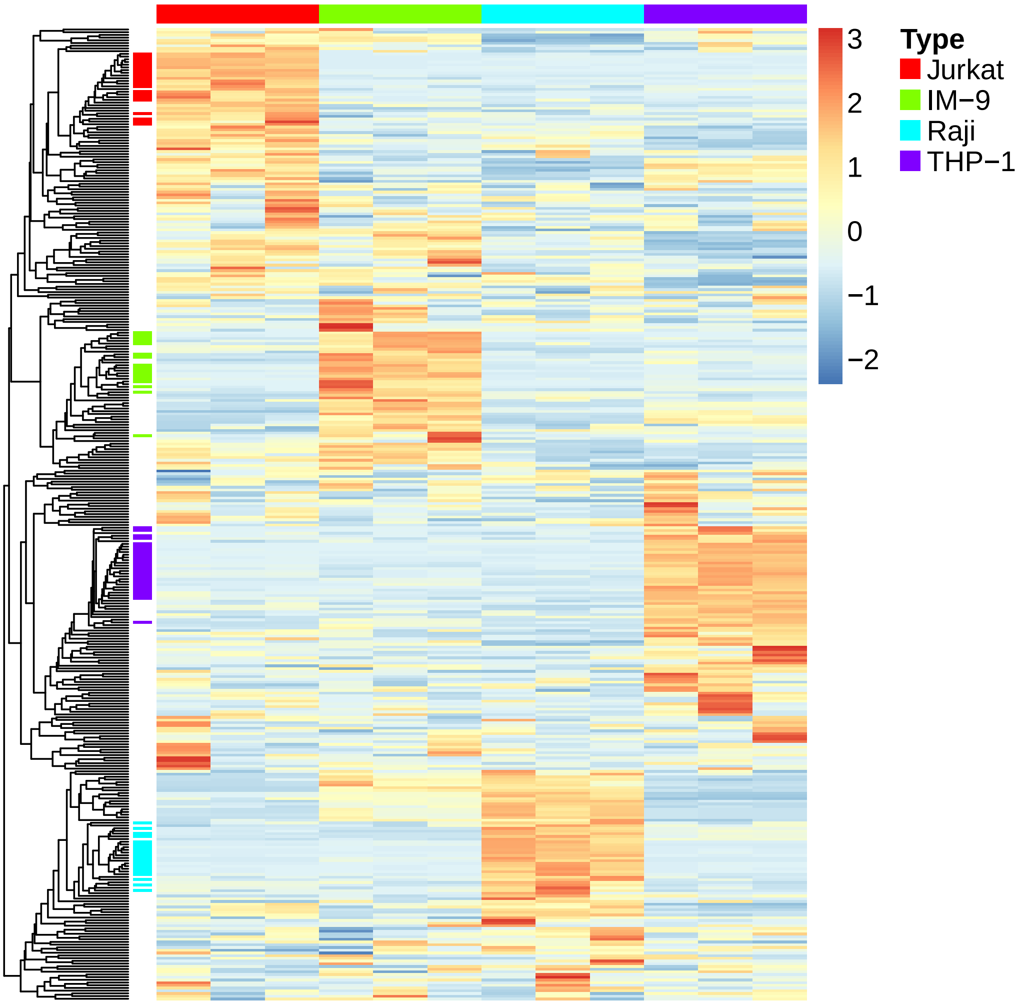

Figure 5: Curated dataset: annotated heatmap of the 12 pure samples, showing all the probesets in this dataset.

```
markermap(curated.markers, curated.model, show_row=FALSE)
```

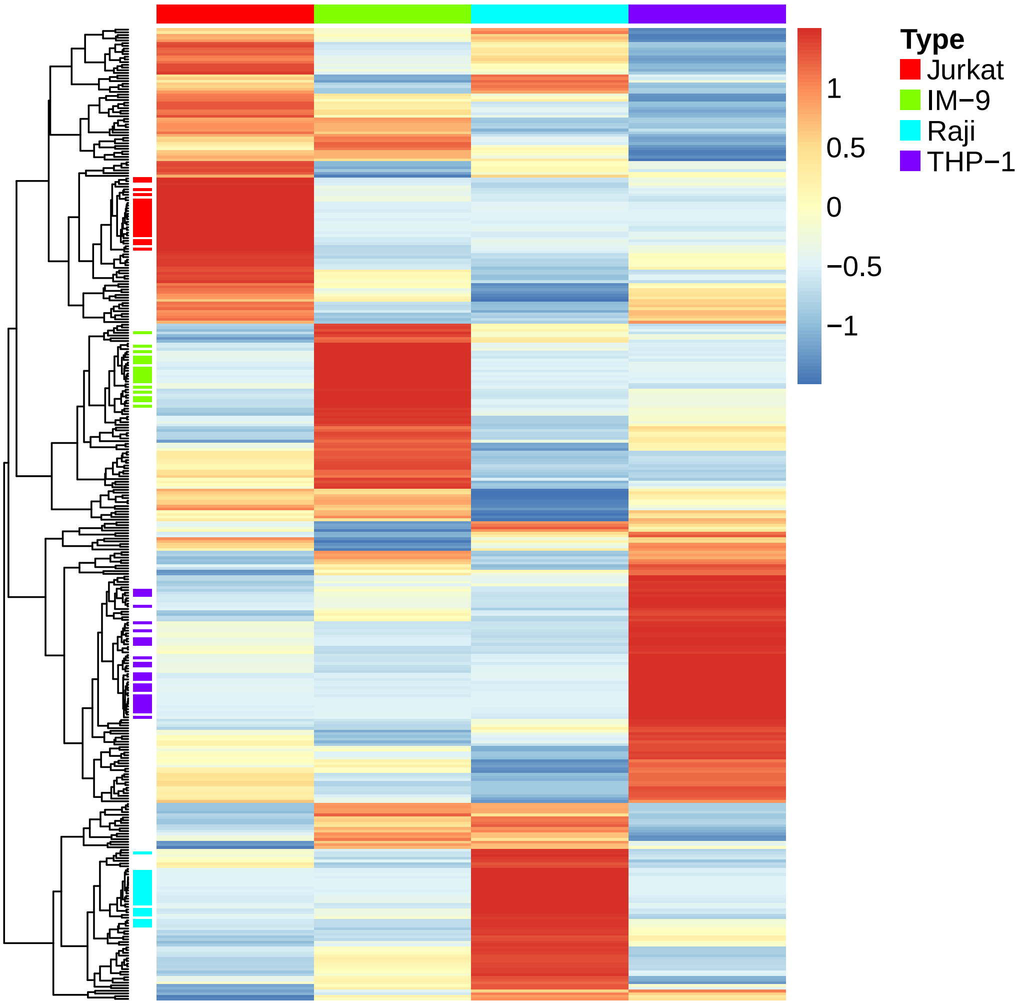

Figure 6: Reference NMF model for the curated dataset: annotated heatmap showing all the probesets in this dataset.

4 Sample analysis

In this section we fit different NMF models, using for some the selected marker probesets to guide the algorithms towards more meaningful results. For the purpose of this vignette we only fit models on the curated dataset. Deconvoluting the full dataset consists essentially in substituting the input and reference data (target matrix, markers and reference NMF model). The estimation is performed using only the mixed samples, removing the 12 first pure samples:

```
curated.dataset <- curated.dataset[, -(1:12)]
curated.dataset
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 359 features, 12 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: GSM279601 GSM279602 ... GSM279612 (12 total)
  varLabels: title geo_accession ... data_row_count (33 total)
  varMetadata: labelDescription
featureData
  featureNames: 242904_x_at 232165_at ... 223809_at (359 total)
  fvarLabels: ID GB_ACC ... Gene.Ontology.Molecular.Function (16 total)
  fvarMetadata: Column Description labelDescription
experimentData: use 'experimentData(object)'
Annotation: hgu133plus2.db
```

```
curated.model <- curated.model[, -(1:12)]
curated.model
```

```
<Object of class: NMFstd >
features: 359
basis/rank: 4
samples: 12
```

We use the seeding method `'rprop'` defined by the package `NMFged`, that seeds the computation with a random NMF model, whose mixture coefficient matrix have been scaled so that the columns sum up to one, like proportions do. Beside the seeding method itself, we fix the seed for the random number generator to 123456, which enable the package `NMF` to generate a reproducible sequence of random streams, each of them being used for a single run. This ensure that the comparisons of the methods is not biased by differences in the random initializations.

4.1 Using standard methods

The package `NMFged` provides modifications of some of the built-in standard algorithms from the package `NMF`, to use a stopping criterion based on the stationarity of the objective function rather than the consensus matrix – which was designed for clustering problems. These are named `Sbrunet`, `Slee` and `SnsNMF` after their respective template versions. A fourth method, `deconf`, is available and was wrapped from the algorithm provide by the package `deconf`. These algorithms do not make use of the marker probesets within the fitting process, but only at posteriori, to assign the estimated signatures to an underlying cell-type.

In this section we illustrate the complete process by fitting an NMF model using the method `Sbrunet` and all the markers to assign the estimated signature. Since NMF algorithms only provide local solutions, one usually perform multiple runs of NMF starting from different random initializations. This is achieved by setting argument `nrun`> 2. For performance reason, we use `nrun=10`, although a greater number of runs would be normally required e.g 100–200, in order to achieve a better estimation accuracy.

```
set.seed(123456)
res0 <- nmf(curated.dataset, 4, 'Sbrunet', seed='rprop', nrun=10, .opt='v2')

 Runs: 2 1* 4 3* 6 5 8 7* 10 9* ... DONE
 System time:
    user  system elapsed
    0.05    0.02   15.58

# scale the mixture coefficients to get proportions
coef(res0) <- scoef(res0)
```

The result is an `NMFfitX1` object, which contains the best fit, that is the fit that achieved the least objective value across the runs.

```
res0

 <Object of class: NMFfitX1 >
   Method: Sbrunet
   Runs:   10
   RNG:
    rstream.mrg32k3a - 797784, 753565, 391256, 341557, 361294, 198345
 [bfcf0066cffada4cf8b348165e962c2f]
   Total timing:
    user  system elapsed
    0.05    0.02   15.58
```

Because the estimated signatures have no a priori predictable order, one needs to assign them a posteriori, to make possible pairwise comparisons with the known signatures and proportions. The method **`match.nmf`** assigns and reorder the basis components according to their predicted porportions of markers from each cell-type. The assigned signatures are named after their respective cell-type.

```
res <- match.nmf(res0, curated.markers)
```

The agreement of this fit with the known signatures and proporitions (i.e. the reference NMF model) can be assessed by plotting

1. the scatter plot of each the estimated proportions for each cell-type (i.e. estimated basis profiles) against the reference;
2. the heatmap of the cell-type specific signatures (i.e. the basis signatures) annotated with the predicted cell-type for each marker probeset, which is their highest-expressing estimated cell-type.

These plots are shown in Figure 7.

```
opar <- par(no.readonly=TRUE)
par(mfrow=c(1, 2))
profplot(curated.model, res, main="Method: Sbrunet")
basismarker(res, curated.markers, add=TRUE)
par(opar)
```

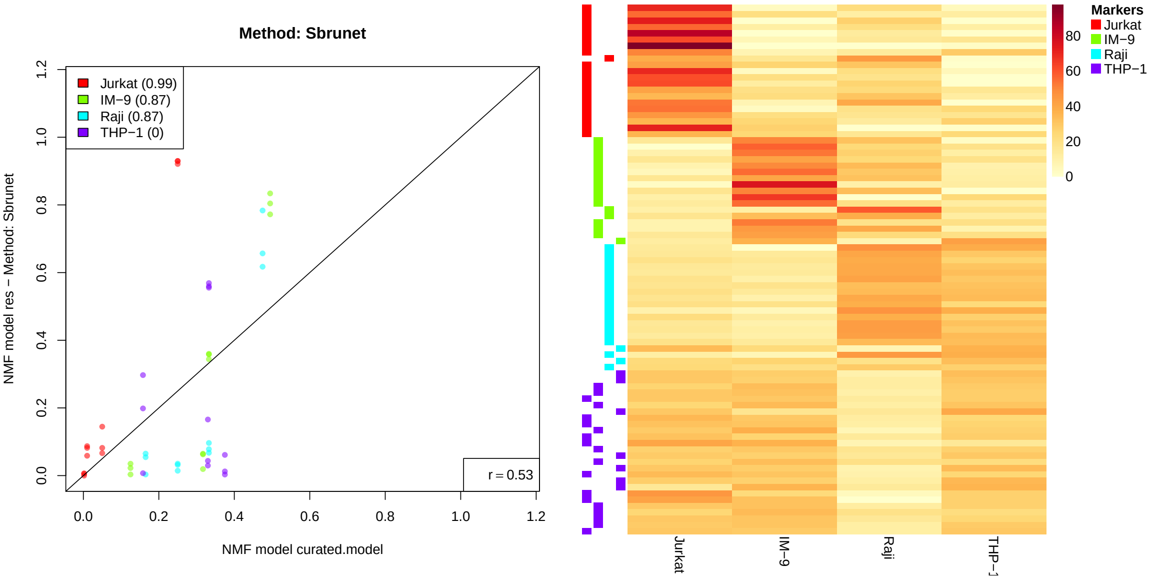

Figure 7: Standard method `Sbrunet`: agreement between the estimated proportions and cell-type signatures.

4.2 Using the semi-supervised approach

The semi-supervised methods use the maker probesets to enforce block patterns in the signatures, that improves the meaningfulness of the estimated model. Moreover, there is no need to a posteriori assign the estimated signatures to a cell-type, because each basis component is a priori assigned to the cell-type whose markers are enforced on. The package `NMFged` provides the semi-supervised (guided) methods `Gbrunet`, `Glee` and `GnsNMF`, named after their respective standard versions.

In this section we use the method `Gbrunet` and enforce a single marker.

```
set.seed(123456)
res <- nmf(curated.dataset, 4, 'Gbrunet', markers=subset(curated.markers, 1),
seed='rprop', nrun=10, .opt='v2')

Runs: 1 2 3 4 6 5 7 8 9 10 ... DONE
System time:
   user  system elapsed
 29.020   0.670  15.277

# scale the mixture coefficients to get proportions
coef(res) <- scoef(res)
```

As previously, the result is assessed with the profile plot and the annotated basis heatmap, as shown in Figure 8. The advantage of enforcing the marker block patterns is clear, in both the estimation of the proportions and the recovery of cell-type specific signatures that are consistent with the known markers.

```
opar <- par(no.readonly=TRUE)
par(mfrow=c(1, 2))
profplot(curated.model, res, main="Method: Gbrunet - 1 marker")
basismarker(res, curated.markers, add=TRUE)
par(opar)
```

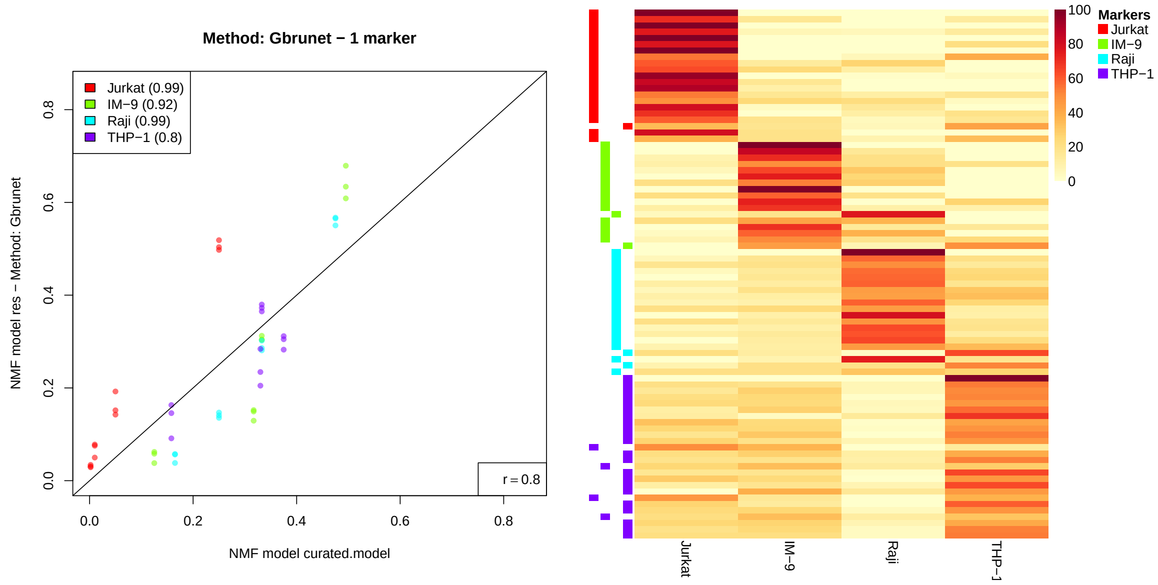

Figure 8: Semi-supervised method `Gbrunet` using a single marker: agreement between the estimated proportions and cell-type signatures.

5 Session Information

- R version 2.12.1 (2010-12-16), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_ZA.utf8`, `LC_NUMERIC=C`, `LC_TIME=en_ZA.utf8`, `LC_COLLATE=en_ZA.utf8`, `LC_MONETARY=C`, `LC_MESSAGES=en_ZA.utf8`, `LC_PAPER=en_ZA.utf8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_ZA.utf8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, tools, utils
- Other packages: AnnotationDbi 1.12.0, bigmemory 4.2.11, Biobase 2.10.0, codetools 0.2-8, colorspace 1.1-0, DBI 0.2-5, deconf 1.0, digest 0.5.0, doMC 1.2.2, foreach 1.3.2, genefilter 1.32.0, gridBase 0.4-3, hgu133plus2.db 2.4.5, highlight 0.2-5, iterators 1.0.5, multicore 0.1-5, NMF 0.5.99, NMFged 1.0, org.Hs.eg.db 2.4.6, parser 0.0-13, RColorBrewer 1.0-5, Rcpp 0.9.6, RSQLite 0.9-4, rstream 1.3.1, stringr 0.5, synchronicity 1.0.9, xtable 1.5-6
- Loaded via a namespace (and not attached): annotate 1.28.0, plyr 1.6, splines 2.12.1, survival 2.36-2

References

[1] Alexander R Abbas, Kristen Wolslegel, Dhaya Seshasayee, Zora Modrusan, and Hilary F Clark. Deconvolution of blood microarray data identifies cellular activation patterns in systemic lupus erythematosus. *PloS one*, 4(7):e6098, January 2009.

[2] Dirk Repsilber, Sabine Kern, Anna Telaar, Gerhard Walzl, Gillian F Black, Joachim Selbig, Shreemanta K Parida, Stefan H E Kaufmann, and Marc Jacobsen. Biomarker discovery in heterogeneous tissue samples -taking the in-silico deconfounding approach. *BMC bioinformatics*, 11:27, January 2010.