Deconvoluting Catalogs of Mutation Counts Against Known Mutational Signatures

Damiano Fantini

March 26, 2018

DNA mutations accumulate in the genomes of cancer cells as result of genetic instability processes. These processes are often associated with cognate mutational signatures. Efforts form the Sanger Institute (http://cancer.sanger.ac.uk) resulted in a comprehensive list of resources for exploring the impact of somatic mutations in human cancer, namely the Catalogue of Somatic Mutations in Cancer (COSMIC, http://cancer.sanger.ac.uk/cosmic). Among others, COSMIC lists a series of 30 Mutational Signatures operative in Human Cancer. The *mutSignatures* and *deconstructSigs* R packages provide tools for deconvoluting cancer mutation counts against these mutational signatures (as well as other mutational signatures). This vignette shows how to perform such analysis, and compares the results obtained by `deconstructSigs` and `mutSignatures`.

Deconvolution of Mutations from the TCGA BLCA Dataset

Importing, Preparing, and Counting Bladder Cancer Mutations

The first step in this analysis is **data retrieval**. Here, we downloaded and imported mutation data from a MAF file. This included info about 395 bladder cancer patients enrolled in the BLCA TCGA provisional study (https://cancergenome.nih.gov/). Initial data processing was performed using functions included in the `mutSignatures` suite.

```
# Load required libraries
library(deconstructSigs)
library(mutSignatures)
library(microbenchmark)
library(ggplot2)
library(gridExtra)

# Retrieve BLCA provisional dataset
fi <- "BLCA-TP"
url_01 <- "http://gdac.broadinstitute.org/runs/analyses__2016_01_28/reports/cancer/"
url_02 <- "/MutSigNozzleReport2CV/"
url_03 <- ".final_analysis_set.maf"

myUrl <- paste(url_01, fi, url_02, fi, url_03, sep = "")
tryCatch({download.file(myUrl, destfile=paste(fi, ".maf", sep = ""))},
         error = function(e) NULL)

# Columns to keep
keep.MAFfields <- c("Hugo_Symbol", "Entrez_Gene_Id", "NCBI_Build",
                    "Chromosome", "Start_Position", "End_position",
                    "Strand", "Variant_Classification", "Variant_Type",
                    "Reference_Allele", "Tumor_Seq_Allele1", "Tumor_Seq_Allele2",
                    "dbSNP_RS", "Mutation_Status", "Protein_Change",
                    "patient", "Tumor_Sample_Barcode")
```

```
# Read and prep BLCA dataset
BLCAmaf <- read.delim(paste(fi, ".maf", sep = ""), as.is = TRUE)
BLCAmaf <- BLCAmaf[, names(BLCAmaf) %in% keep.MAFfields]
BLCAmaf$case_id <- substr(BLCAmaf$patient, 1, 15)
head(BLCAmaf)

# Filter single nucleotide variants
BLCAdf <- filterSNV(dataSet = BLCAmaf,
                    seq_colNames = c("Reference_Allele",
                                     "Tumor_Seq_Allele1", "Tumor_Seq_Allele2"))
# Attach 3-nucleotide context
hg19 <- BSgenome.Hsapiens.UCSC.hg19::BSgenome.Hsapiens.UCSC.hg19
BLCAdf <- attachContext(mutData = BLCAdf,
                        chr_colName = "Chromosome",
                        start_colName = "Start_Position",
                        end_colName = "End_position",
                        nucl_contextN = 3,
                        BSGenomeDb = hg19)

BLCAdf <- removeMismatchMut(mutData = BLCAdf,
                            refMut_colName = "Reference_Allele",
                            context_colName = "context",
                            refMut_format = "N")
# Attach mutation Type
BLCAdf <- attachMutType(mutData = BLCAdf,
                        ref_colName = "Reference_Allele",
                        var_colName = "Tumor_Seq_Allele1",
                        var2_colName = "Tumor_Seq_Allele2",
                        context_colName = "context")
```

Next, we had a look at the data. As shown below, all mutations were stored in a `data.frame`, that included mutation types, as well as patient unique identifier. Mutations were counted using the `countMutTypes()` function from the `mutSignatures` package. Also, the counts were extracted from the `BLCA_counts` object, transposed, and then coerced to `data.frame` before being used for the *deconstructSigs* analysis.

```
head(BLCAdf[,c("Hugo_Symbol", "case_id", "context", "mutType")])
```

```
##   Hugo_Symbol         case_id context mutType
## 1        A1BG TCGA-FD-A3SJ-01     GGG C[C>T]C
## 2        A1BG TCGA-FD-A62P-01     CCG C[C>T]G
## 3        A1BG TCGA-GC-A3I6-01     TCC T[C>G]C
## 4        A1BG TCGA-SY-A9G5-01     TGA T[C>G]A
## 5        A1BG TCGA-UY-A9PD-01     TGA T[C>T]A
## 6        A1CF TCGA-CF-A9FF-01     TCT T[C>G]T
```

```
# Count Mutations
BLCA_counts <- countMutTypes(mutTable = BLCAdf,
                             sample_colName = "case_id",
                             mutType_colName = "mutType")
# How many samples are there?
print(length(getSampleIdentifiers(BLCA_counts)))
```

```
## [1] 395
```

```
# Convert to data frame, suitable for deconstructSigs analysis
BLCA_counts_df <- as.data.frame(t(as.data.frame(BLCA_counts)))
```

Importing COSMIC signatures

After preparing mutation counts, we retrieved mutational signatures. In this example, we imported COSMIC signatures that were found important in bladder cancer (doi:10.1016/j.cell.2017.09.007).

```
# Retrieve COSMIC signatures, extract all that are important in bladder cancer tcga
data(cosmix) #alternatively, try: `getCosmicSignatures()`
print(cosmix)
```

```
##  Mutation Signatures object - mutSignatures
##
##  Total num of Signatures: 30
##  Total num of MutTypes: 96
##
##     Sign.1   Sign.2   Sign.3   Sign.4   Sign.5
##     ------   ------   ------   ------   ------
##   + 0.0111   0.0007   0.0222   0.0365   0.0149  +  A[C>A]A
##   + 0.0091   0.0006   0.0179   0.0309   0.0090  +  A[C>A]C
##   + 0.0015   0.0001   0.0021   0.0183   0.0022  +  A[C>A]G
##   + 0.0062   0.0003   0.0163   0.0243   0.0092  +  A[C>A]T
##   + 0.0018   0.0003   0.0240   0.0097   0.0117  +  A[C>G]A
##   + 0.0026   0.0003   0.0122   0.0054   0.0073  +  A[C>G]C
##   + 0.0006   0.0002   0.0053   0.0031   0.0023  +  A[C>G]G
##   + 0.0030   0.0006   0.0233   0.0054   0.0117  +  A[C>G]T
##   + 0.0295   0.0074   0.0179   0.0120   0.0218  +  A[C>T]A
##   + 0.0143   0.0027   0.0089   0.0075   0.0128  +  A[C>T]C
##     ......   ......   ......   ......   ......
```

```
blca.cosmic <- as.data.frame(t(as.data.frame(cosmix[c(1, 2, 5, 13, 10)])))
```

Approach 1: deconstructSigs

The `deconstructSigs` approach is based on the `whichSignatures()` function. We ran this function for each patient included in the study and multiplied each imputed weight by the total number of mutations found in the corresponding sample. Results were stored in the `run_01` variable, which was in turn coerced to a *Mutation Exposure* object before visualization and results comparison.

```
# Deconvolute, approach 'deconstructSigs'
run_01 <- do.call(rbind, lapply(1:nrow(BLCA_counts_df), function(i) {
  TMP <- whichSignatures(tumor.ref = BLCA_counts_df,
                  sample.id = rownames(BLCA_counts_df)[i],
                  signatures.ref = blca.cosmic,
                  contexts.needed = T)$weights
  sum(BLCA_counts_df[i,]) * TMP}))

# Collect and convert results
run_01_out <- mutSignatures::as.mutsign.exposures(run_01, samplesAsCols = F)
```

Approach 2: mutSignatures

The `mutSignatures` approach is very straightforward, and is based on the `resolveMutSignatures()` function. This function, in turn, relies on the `fcnnls()` function from the `NMF` package. Results are returned as a list of lists. Imputed mutation counts by signatures and by sample are included in the `results$count.result` element of the output.

```
# Deconvolute, approach 'mutSignatures'
run_02 <- mutSignatures::resolveMutSignatures(mutCountData = BLCA_counts,
                                              signFreqData = cosmix[c(1,2,5,13,10)])

run_02_out <- run_02$results$count.result
```

Comparison and Conclusions

Despite the differences in syntax and data formats, both approaches were rather simple to implement, and reasonably fast. Most important of all, they produced very consistent results.

- **deconstructSigs** implements a series of optional controls for normalization and removal of under-represented signatures. These were not further explored here, since default parameters were used.
- **mutSignatures** and the `resolveMutSignatures()` function were lightning fast, simpler to use, and returned results that tracked with those of `deconstructSigs`

Below, the two barplots summarize signature exposures of the top 50 samples (samples with highest mutation load) in the BLCA TCGA provisional dataset imputed by the *deconstructSigs* and the *mutSignatures* method. These results were overlapping.

```
# Custom colors for the plot
my_cols <- c("#ffd92f", "#e7298a", "#386cb0", "#fbb4ae", "#67b867")

# Display results (top 50 samples) - deconstructSigs
p1 <- mutSignatures::plot(run_01_out, top = 50) +
  scale_fill_manual(values = my_cols) +
  ggtitle(label = "deconstructSigs") +
  scale_y_continuous(limits = c(0, 5100), expand = c(0, 0))

# Display results (top 50 samples) - mutSignatures
p2 <- mutSignatures::plot(run_02_out, top = 50) +
  scale_fill_manual(values = my_cols) +
  ggtitle(label = "mutSignatures / fcnnls") +
  scale_y_continuous(limits = c(0, 5100), expand = c(0, 0))
```

An interesting functionality that comes with `mutSignatures` is that plots generated as shown above, carry information about the identity of each sample. Specifically, the sample identifier (`inputLabel`) is attached to the data.frame returned as part of the `ggplot2` object. This enables sample selection and sorting via `ggplot2::scale_x_discrete()`. However, note that the original plots returned by `plotMutCount()` and plot methods invoking `plotMutCount()` use the `sample` feature as x-axis value. Below, we are sorting the data in `p2` so that the sample order is identical to `p1`.

```
# Show identifiers in the ggplot2-object data
head(p1$data)
```

```
##   sample   feature     count      inputLabel
## 1 100001  COSMIC.1  752.2043 TCGA-DK-A6AW-01
## 2 100001  COSMIC.2    0.0000 TCGA-DK-A6AW-01
## 3 100001  COSMIC.5    0.0000 TCGA-DK-A6AW-01
## 4 100001 COSMIC.13    0.0000 TCGA-DK-A6AW-01
## 5 100001 COSMIC.10 3702.7957 TCGA-DK-A6AW-01
## 6 100002  COSMIC.1    0.0000 TCGA-MV-A51V-01
```

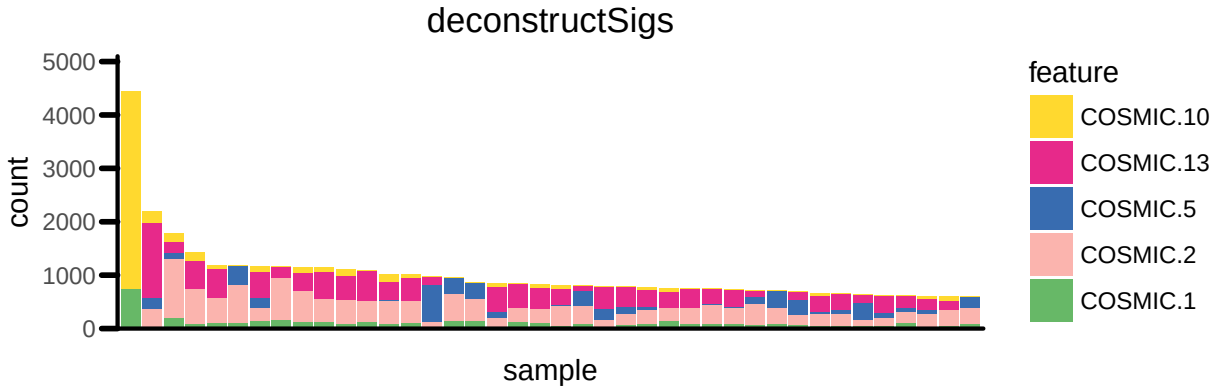
```
# Select shared samples based on original identifiers
commonSamples <- unique(p1$data$inputLabel)
```

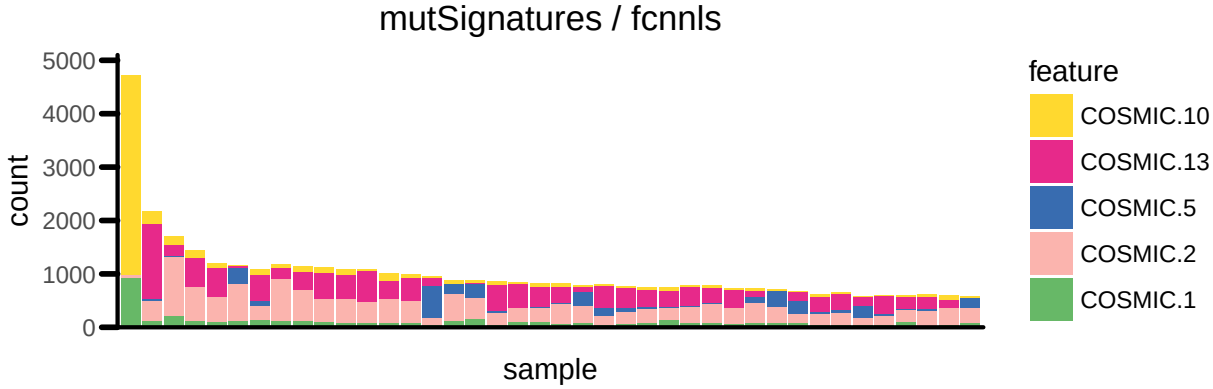
```r
commonSamples <- commonSamples[commonSamples %in% p2$data$inputLabel]

# Generate limits for scaling p1 and p2 x-axis based on `commonSamples`
# Limit to 40 samples
p1_xscale <- as.character(sapply(commonSamples[1:40], function(id) {
  p1$data$sample[p1$data$inputLabel == id][1]
}))
p2_xscale <- as.character(sapply(commonSamples[1:40], function(id) {
  p2$data$sample[p2$data$inputLabel == id][1]
}))

# Scale
p1 <- p1 + scale_x_discrete(limits = p1_xscale)
p2 <- p2 + scale_x_discrete(limits = p2_xscale)

# Plot
blank<-rectGrob(gp=gpar(col="white")) # make a white spacer grob
grid.arrange(p1, blank, p2, heights=c(4.5, 0.5, 4.5), nrow=3)
```





A final test was performed to compare the speed of these two approaches. This was achieved using the `microBenchmark` R package and repeating each operation (as described above) 20 times. As shown below, the `mutSignatures` approach overperformed the other method, with a median time for processing 395 samples of less-than 1s (median elapsed time ~ 0.5s, compared to ~ 27s of `deconstructSigs`).

```r
# Compare elapsed time
mbr <- microbenchmark(
  decS = {do.call(rbind, lapply(1:nrow(BLCA_counts_df), function(i) {
```

```
  TMP <- whichSignatures(tumor.ref = BLCA_counts_df,
                         sample.id = rownames(BLCA_counts_df)[i],
                         signatures.ref = blca.cosmic,
                         contexts.needed = T)$weights
  sum(BLCA_counts_df[i,]) * TMP}))},

  mutS = {mutSignatures::resolveMutSignatures(mutCountData = BLCA_counts,
                                              signFreqData = cosmix[c(1,2,5,13,10)])},

  times = 20)

# Print results
mbr <- summary(mbr)
print(mbr[,-c(3, 6)])
```

```
##   expr        min       mean     median       max neval
## 1 decS 23988.5613 25417.8347 25118.5508 28030.509    20
## 2 mutS   468.7155   525.2159   500.0299   668.322    20
```

Thanks for using the `mutSignatures` package.

Session Info

```
## Elapsed time for building this Vignette and Running all examples
```

```
## ~ 12 mins.
```

```
# Session Info
print(sessionInfo())
```

```
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.3 LTS
##
## Matrix products: default
## BLAS: /usr/lib/libblas/libblas.so.3.6.0
## LAPACK: /usr/lib/lapack/liblapack.so.3.6.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
## [1] corpcor_1.6.9         gridExtra_2.3         ggplot2_2.2.1
## [4] microbenchmark_1.4-4  mutSignatures_1.3.7   Biobase_2.38.0
## [7] BiocGenerics_0.24.0   deconstructSigs_1.8.0
##
```

```
## loaded via a namespace (and not attached):
##  [1] SummarizedExperiment_1.8.1        reshape2_1.4.3
##  [3] lattice_0.20-35                   colorspace_1.3-2
##  [5] BSgenome.Hsapiens.UCSC.hg19_1.4.0 htmltools_0.3.6
##  [7] stats4_3.4.3                      rtracklayer_1.38.3
##  [9] yaml_2.1.16                       NMF_0.21.0
## [11] XML_3.98-1.9                      rlang_0.1.6
## [13] pracma_2.1.4                      pillar_1.1.0
## [15] BiocParallel_1.12.0               RColorBrewer_1.1-2
## [17] registry_0.5                      rngtools_1.2.4
## [19] matrixStats_0.53.0                GenomeInfoDbData_1.0.0
## [21] foreach_1.4.4                     plyr_1.8.4
## [23] pkgmaker_0.22                     stringr_1.3.0
## [25] zlibbioc_1.24.0                   Biostrings_2.46.0
## [27] munsell_0.4.3                     gtable_0.2.0
## [29] codetools_0.2-15                  evaluate_0.10.1
## [31] labeling_0.3                      knitr_1.19
## [33] IRanges_2.12.0                    doParallel_1.0.11
## [35] GenomeInfoDb_1.14.0               Rcpp_0.12.15
## [37] xtable_1.8-2                      scales_0.5.0
## [39] backports_1.1.2                   BSgenome_1.46.0
## [41] DelayedArray_0.4.1                S4Vectors_0.16.0
## [43] XVector_0.18.0                    Rsamtools_1.30.0
## [45] digest_0.6.15                     stringi_1.1.7
## [47] GenomicRanges_1.30.1              rprojroot_1.3-2
## [49] tools_3.4.3                       bitops_1.0-6
## [51] magrittr_1.5                      lazyeval_0.2.1
## [53] RCurl_1.95-4.10                   proxy_0.4-21
## [55] tibble_1.4.2                      cluster_2.0.6
## [57] Matrix_1.2-11                     gridBase_0.4-7
## [59] rmarkdown_1.8                     iterators_1.0.9
## [61] GenomicAlignments_1.14.1          compiler_3.4.3
```