# Deploying mutSignatures on Computational Clusters

*Damiano Fantini*

*April 2nd, 2018*

This document provides a guide to analyze and extract mutational signatures using high-performance computational clusters. This tutorial will cover the following points:

- Launching an Amazon EC2 instance (Linux, Rstudio), installing, and loading all required software
- Retrieve TCGA data (MAF files) from the Broad Institute Servers
- Retrieve TCGA clinical data from cBioPortal using TCGAretriever
- Extract, prepare, and count tri- and tetra-nucleotide mutation types
- Different flavors of mutational signature extraction
- Simplification of complex mutational signatures, signature comparison, sorting, and data visualization

Deploying *mutSignatures* on a computational clusters enables exploiting the built-in parallelization capacity of this analytical framework. In this example, an Amazon EC2 c4.4xlarge instance is used. To get started with R and RStudio on an EC2 instance, it is possible to use one of the RStudio AMIs maintained by Dr. Louis Aslett (Durham University, UK), at the following URL: http://www.louisaslett.com/RStudio_AMI/. Instructions to launch an EC2 instance and log-in into the RStudio server are provided at the same URL.

## Initialization Routine

Before starting with the analyses, it is important to install all required software on the system. This can be done from RStudio, by executing the following initialization routine that installs all required R extensions. These lines are supposed to be executed line-by-line. Make sure that all steps are completed without errors. Depending on the operating system (here, Ubuntu is used), RStudio version, and intended analysis, the installation of additional packages may be required.

```r
# Install Bioconductor and Bioc packages
source("http://www.bioconductor.org/biocLite.R")
biocLite()
biocLite("BSgenome.Hsapiens.UCSC.hg19")
biocLite("BSgenome.Mmusculus.UCSC.mm10")
biocLite("copynumber")

# CRAN packages
install.packages("devtools")
install.packages("survival")
install.packages("corpcor")
install.packages("TCGAretriever")
install.packages("sequenza")
install.packages("ggplot2")

# Install the latest version of mutSignatures from GitHub
library(devtools)
install_github("dami82/mutSignatures")
```

At the end of the installation, we recommend to re-start the RStudio session.

## Data retrieval

The *mutSignatures* analytic framework is aimed at processing DNA mutation variants, typically from cancer datasets. A great resource of cancer genomic data is *The Cancer Genome Atlas* (TCGA, https://cancergenome.nih.gov/). Some pre-processed TCGA data are mirrored or stored on the *Broad Institute* servers (https://www.broadinstitute.org/), and can be accessed at the following URL: http://gdac.broadinstitute.org/runs/analyses___2016_01_28/reports/cancer/.

Here, we are downloading Mutational data from six TCGA cancers: adrenocortical cancer (ACC), bladder cancer (BLCA), esophageal cancer (ESCA), liver cancer (LIHC), lung adenocarcinoma (LUAD), and cervical/endocervical carcinoma (UCEC). The corresponding MAF files (primary tumors) from the Broad repository are downloaded on the fly, using the `read.delim()` function. Mutation data are stored in a list. A total of 598,239 mutations from 1,796 patients were retrieved.

```
## Get MAF files
all_dsets <- c("ACC", "BLCA", "ESCA", "LIHC", "LUAD", "UCEC")

url_01 <- paste("http://gdac.broadinstitute.org/runs/",
                "analyses__2016_01_28/reports/cancer/",
                sep = "")
url_02 <- "-TP/MutSigNozzleReport2CV/"
url_03 <- "-TP.final_analysis_set.maf"

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")

# Retrieve the data
# This may take several minutes, depending on the Internet connection speed
maf.list <- sapply(all_dsets, function(fi) {

  # Compose URL and read
  myUrl <- paste(url_01, fi, url_02, fi, url_03, sep = "")
  TMPmaf <- tryCatch({read.delim(myUrl, as.is = TRUE)},
                     error = function(e) NULL)

  TMPmaf <- TMPmaf[, names(TMPmaf) %in% keep.MAFfields]
  TMPmaf$case_id <- substr(TMPmaf$patient, 1, 15)

  # Return
  TMPmaf
}, USE.NAMES = TRUE, simplify = FALSE)
```

```
# Total number of mutations per dataset
sapply(maf.list, nrow)
```

```
##    ACC   BLCA   ESCA   LIHC   LUAD   UCEC
##  13912 125708  46326  51600 187450 173243
```

```
# Total number of patients per dataset
sapply(maf.list, function(x) {length(unique(x$case_id))})
```

```
##  ACC BLCA ESCA LIHC LUAD UCEC
##   62  395  185  373  533  248
```

## Mutation Type Preparation

After retrieval, mutation data are prepared according to the standard *mutSignatures* pipeline. Briefly, non-SNV mutations are filteredd out, nucleotide context is attached, mutation types are computed, and then formatted according to the standard *Sanger Institute* style. These steps are throughly described in the vignette entitled: "**Getting Started with mutSignatures**". Here, we are retrieving the 5-nucleotide context, and then trimming the resulting mutation types to obtain 3-nt and 4-nt mutations (4-nt mutation types as shown by Wormald et al, 2018, https://doi.org/10.1093/carcin/bgx133). For a total number of ~600,000 variants, all operations should only take about 10 minutes, without the need of parallelization.

```
# load mutSignatures
library(mutSignatures)

# load hg19 - reference genome
hg19 <- BSgenome.Hsapiens.UCSC.hg19::BSgenome.Hsapiens.UCSC.hg19

# Filter single nucleotide variants, attach context, compute mutTypes
maf.list <- sapply(maf.list, function(x){
  x <- filterSNV(dataSet = x,
                      seq_colNames = c("Reference_Allele",
                                       "Tumor_Seq_Allele1",
                                       "Tumor_Seq_Allele2"))

  # Attach 3-nucleotide context
  x <- attachContext(mutData = x,
                     chr_colName = "Chromosome",
                     start_colName = "Start_Position",
                     end_colName = "End_position",
                     nucl_contextN = 5,
                     BSGenomeDb = hg19)

  # Remove mismatched positions
  x <- removeMismatchMut(mutData = x,
                         refMut_colName = "Reference_Allele",
                         context_colName = "context",
                         refMut_format = "N")

  # Attach mutation Type
  x <- attachMutType(mutData = x,
                     ref_colName = "Reference_Allele",
                     var_colName = "Tumor_Seq_Allele1",
                     var2_colName = "Tumor_Seq_Allele2",
                     context_colName = "context")

  # Clean mut types
  x$mutType.3 <- substr(x$mutType, 2, 8)
  x$mutType.4 <- substr(x$mutType, 1, 8)
  # Return
  x
}, simplify = FALSE)
```

Let's have a look at an excerpt from the mutation data at the end of processing.

```
# Visualize an excerpt from the data
maf.list[[1]][1:6, c(1, 16, 18, 21, 22)]
```

```
##   Hugo_Symbol Protein_Change         case_id mutType.3 mutType.4
## 1        A1BG         p.R94H TCGA-OR-A5KB-01   G[C>T]G  AG[C>T]G
## 2        A1CF        p.G397G TCGA-OR-A5KB-01   G[C>A]G  GG[C>A]G
## 3       A2ML1        p.A481T TCGA-OR-A5J4-01   G[C>T]A  TG[C>T]A
## 4      A4GALT        p.P301P TCGA-OR-A5JY-01   C[C>G]G  CC[C>G]G
## 5        AACS        p.A102A TCGA-OR-A5LD-01   C[T>G]G  TC[T>G]G
## 6        AACS         p.A35P TCGA-PK-A5HB-01   G[C>G]C  AG[C>G]C
```

In the next step (last step of data preparation), mutation types are counted by patient (*case_id*). This operation is performed by the `countMutTypes()` function. Before counting mutations, all samples with less than 30 total mutations are removed (*n=30* is an arbitrary threshold to discriminate between genetic-stable or instable genomes). This operation is performed in agreement with the hypothesis that genomes with few total mutations may have no active sources of genetic instability. Therefore, the corresponding samples are considered unsuitable for extraction of mutational signatures. The following lines of code return a list of Mutation Count objects.

```
# Define a threshold
mutBurden_thresh <- 30

# Count mutation types
mutCount.list <- sapply(maf.list, function(x){

  # Filter by threshold
  mBySmpl <- table(x$case_id)
  KEEPid <- names(mBySmpl >= mutBurden_thresh)

  # COunt mutations
  mt4c <- countMutTypes(mutTable = x[x$case_id %in% KEEPid, ],
                        sample_colName = "case_id",
                        mutType_colName = "mutType.4")

  mt3c <- countMutTypes(mutTable = x[x$case_id %in% KEEPid, ],
                        sample_colName = "case_id",
                        mutType_colName = "mutType.3")

  # return list
  list(mt3c = mt3c, mt4c = mt4c)
}, simplify = FALSE, USE.NAMES = TRUE)
```

Let's visualize (by printing to console) one of the 'Mutation Counts' objects that were computed. These objects can be analyzed by NMF, and used to extract mutational signatures and compute mutation singature exposures.

```
print(mutCount.list[[1]]$mt3c)
```

```
##  Mutation Counts object - mutSignatures
##
```

```
##  Total num of MutTypes: 96
##  MutTypes: A[C>A]A, A[C>A]C, A[C>A]G, A[C>A]T, A[C>G]A ...
##
##  Total num of Samples: 62
##  Sample Names: TCGA-OR-A5KB-01, TCGA-OR-A5J4-01, TCGA-OR-A5JY-01, TCGA-OR-A5LD-01, TCGA-PK-A5HB-01 .
```

## Clinical Data Retrieval

Before starting with signature extraction, let's download TCGA clinical information. While these data are not used for signature extraction, they may be examined later on, with respect to signature exposures. For example, we may want to study the association between smoking status and specific mutational signatures. To retrieve clinical data, the fastest way is to use the *TCGAretriever* package (https://cran.r-project.org/web/packages/TCGAretriever/index.html). This will download TCGA data from cBioPortal (http://www.cbioportal.org/). Correlations between clinical features and mutational signatures will be investigated in the section entitled: *"Correlation between Mutational Signatures and Clinical Features"*. Execution of this code requires connection to Internet.

```r
# Collect clinical data corresponding to the five datasets
# that were specified above
all_dsets <- c("ACC", "BLCA", "ESCA", "LIHC", "LUAD", "UCEC")

# load TCGAretriever library
library(TCGAretriever)

# collect clinical data
clin.list <- sapply(all_dsets, function(x){
  x <- paste(tolower(x), "_tcga_all", sep = "")
  suppressWarnings(get_clinical_data(x))
}, simplify = FALSE)
```

## Standard Signature Extraction Pipeline

The basic mutational signature extraction analysis is executed by aligning to the parameters originally set by the Sanger Institute for operating the WTSI MATLAB framework. This means using:

1. Brunet's NMF algorithm
2. Raw counts as input
3. Total number of NMF iterations should be 1000 (use 100-200 iterations for exploratory analyses, similar to the example shown in this vignette; however, use 500-1000 iterations for production)

Parallelization can considerably speed up this analysis. Parallelization requires a multi-core CPU system and is enabled by setting the argument `num_parallelCores = 2` or higher number (indicating how many cores to use for parallelization). In the following example, we are using 12 cores at the same time for analyzing data in parallel.

A second very important parameter is the number of signatures to extract. Finding the optimal number $k$ of processes to extract is often achieved by trial-and-error. Start with a reasonable $k$ (much smaller than the number of genomes), extract signatures, review the results, update $k$ and repeat. Continue iterating until you find the most reliable $k$ signatures. Specifically:

- Reduce $k$ if you get unreliable signatures showing overall low (~0) or ultra-low (<<0) silhouette values (examine silhouette plots, disregard outliers, and assess how consistent were the NMF results. Silhouette plots are returned automatically, at the end of the NMF procedure)

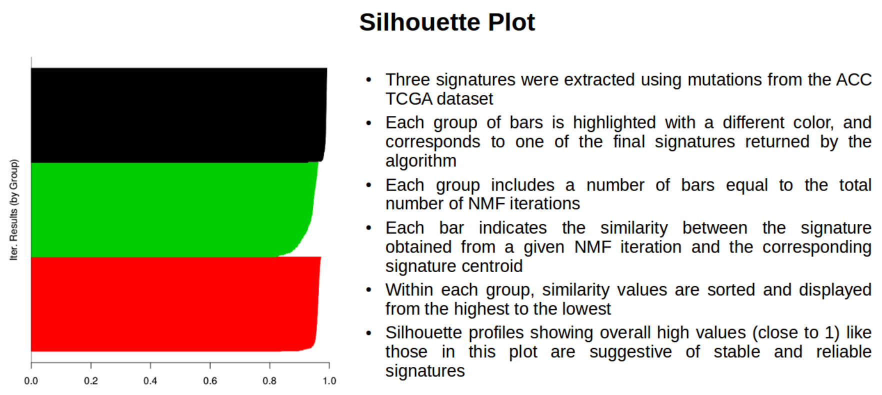


Figure 1: A representative Silhouette plot returned at the end of signature extraction

- Increase $k$ as long as the extracted signatures have distinct non-overlapping mutational profiles/patterns (if many signatures have overlapping mutational profile with minimal changes, consider reducing $k$)

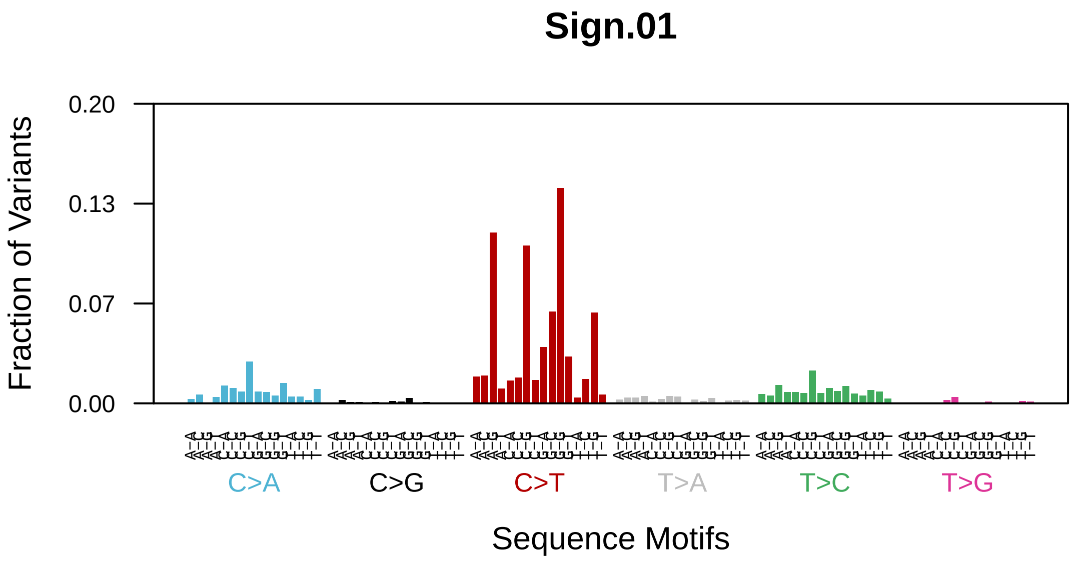
In this example, we extract 4 mutational signatures for the BLCA, ESCA, LUAD, and UCEC TCGA datasets, and 3 signatures for the ACC, and LIHC TCGA datasets. Parameters are set by the `setMutClusterParams()` function. Extraction is executed by the `decipherMutationalProcesses()` function. Depending on the number of cores, number of genomes, complexity of the data, and the total number of iterations, the following operation may take several minutes to complete. Set the framework parameter `debug=TRUE` for printing to console information about the progress status (note: verbose untidy output; text will flood. Useful option to make sure about job advancements). Finally, note the use of `sapply()` for executing the same operation for each dataset and collecting all NMF results in the same output list.

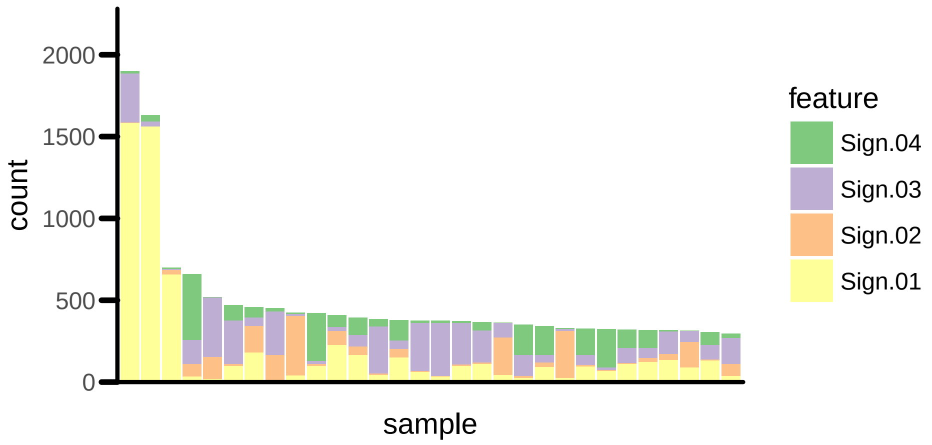
```
# Define k (num. of signatures to extract) by dataset
k.byDset <- c(ACC=3, BLCA=4, ESCA=4, LIHC=3, LUAD=4, UCEC=4)

# Execute analysis
std.results <- sapply(names(mutCount.list), function(x){
  # Get K
  my.k <- k.byDset[x]

  # Get the 3-nucleotide mutCounts
  xx <- mutCount.list[[x]]$mt3c

  # Params
  std.params <- setMutClusterParams(num_processesToExtract = my.k,
                                    num_totIterations = 200,
                                    num_parallelCores = 12,
                                    debug = TRUE,
                                    approach = "counts",
                                    algorithm = "brunet")
  # Analyze and return
  decipherMutationalProcesses(input = xx,
                              params = std.params)
}, simplify = FALSE)
```

## Review signature extraction results

Let's inspect results from signature extraction. Each signature extraction analysis returns a list including different elements. The `$RunSpecs` element includes the input, ie the 'Mutation Counts' and the 'mutFrameworkParams' objects that were used for the extraction. The `$Results` element, contains the 'Mutation Signatures' and 'MutSignature Exposures' objects returned by the analysis. To visualize the mutation profile of a mutational signature, you should use the `plot()` method, and specify the argument `signature` (for example, `signature=1`), indicating the index of the signature to plot. Here, `plot()` invokes the `plotMutTypeProfile()` function included in the `mutSignatures` package. Similarly, the `plot()` method can be used to visualize signature exposures. Here, you can specify the `top` argument (for example, `top=30`), which indicates how many samples to plot (the samples with the highest mutation burden are selected). Here, `plot()` invokes the `plotMutCount()` function which also is included in the `mutSignatures` package. Exposures visualizations are built on the top of `ggplot2`, and hence functions from `ggplot2` can be applied to change colors or format the resulting plots.

```
library(ggplot2)
plot(std.results$ESCA$Results$signatures, signature=1)
plot(std.results$ESCA$Results$exposures, top=30) + scale_fill_brewer(type="qual")
```

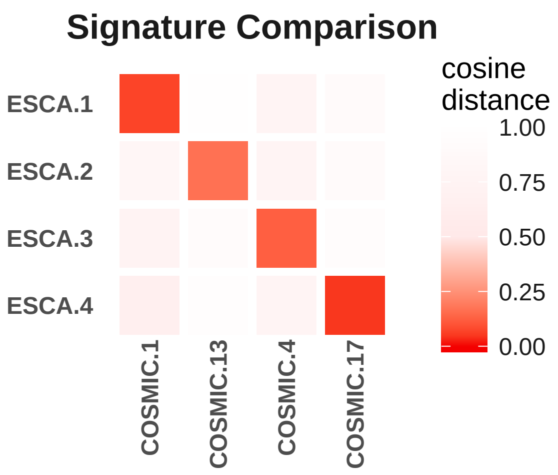


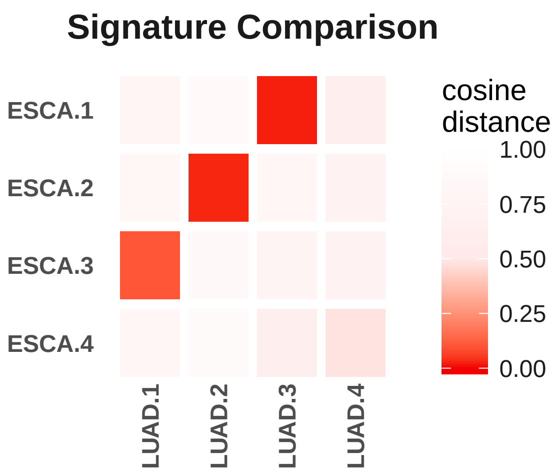


## Mutational Signature Matching

A very important type of analysis is the comparison of mutational signatures. Briefly, it may be insightful to compare newly extracted mutational signatures with other mutational signatures, for example well-established human cancer signatures from COSMIC (http://cancer.sanger.ac.uk/cosmic/signatures), or signatures that were extracted using different parameters or from a different input dataset. The *mutSignatures* framework comes with a dedicated function for this type of analysis: `matchSignatures()`. Briefly, this function takes two 'Mutation Signatures' objects and computes the cosine distance between each signature from the first object and each signature from the latter. A distance matrix is returned, together with an optional heatmap-representation of the same results. In the heatmap, red boxes correspond to cosine distances ~ 0, indicative of high similarity between signatures (good match). White boxes display cosine distances ~ 1, indicative of high dissimilarity (poor match) between signatures. Below, signatures extracted from ESCA are compared against COSMIC signatures that were previously identified in the same type of cancer, namely COSMIC 1, 13, 4, and 17. Also, similarity between signatures derived from ESCA (y-axis) and LUAD (x-axis) is assessed. Interestingly, newly extracted signatures from ESCA match COSMIC signatures 1, 13, 4, and 17. Three of these signatures are found in both ESCA and LUAD cancers, and have high similarity (signatures corresponding to COSMIC 1, 13, and 4). On the contrary, the fourth signature (matching COSMIC.17) is ESCA-specific. Note the use of `getSignatureIdentifiers()` and `setSignatureIdentifiers()` to obtain and update the names of mutational signatures included in Mutational Signatures-objects.

```r
# Get COSMIC signatures (reference human signatures)
cosmix <- getCosmicSignatures()

# Rename mutational signatures, for easy comparison
for (x in names(std.results)) {
  tmp <- std.results[[x]]$Results$signatures
  sigRange <- 1:length(getSignatureIdentifiers(tmp))
  std.results[[x]]$Results$signatures <-
    setSignatureIdentifiers(tmp, paste(x, sigRange, sep="."))
}

# ESCA vs. COSMIC 1, 2, 4, and 17
mtch1 <- matchSignatures(mutSign = std.results$ESCA$Results$signatures,
                         reference = cosmix[c(1,13,4,17)])
print(mtch1$plot)

# LUAD vs. ESCA
# three shared signatures
mtch2 <- matchSignatures(mutSign = std.results$ESCA$Results$signatures,
                         reference = std.results$LUAD$Results$signatures)
print(mtch2$plot)
```

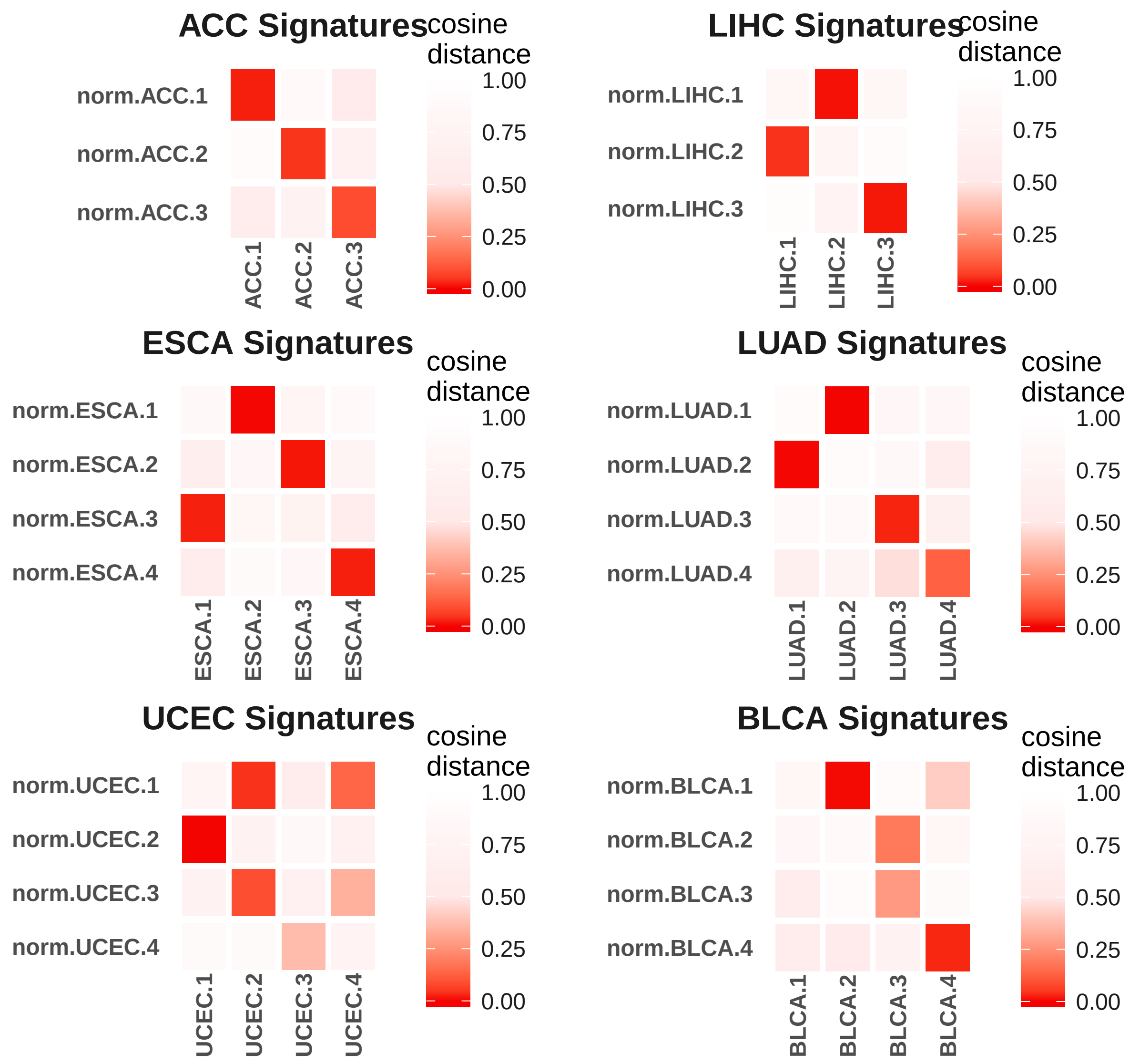




## Signature Extraction using Normalized Counts

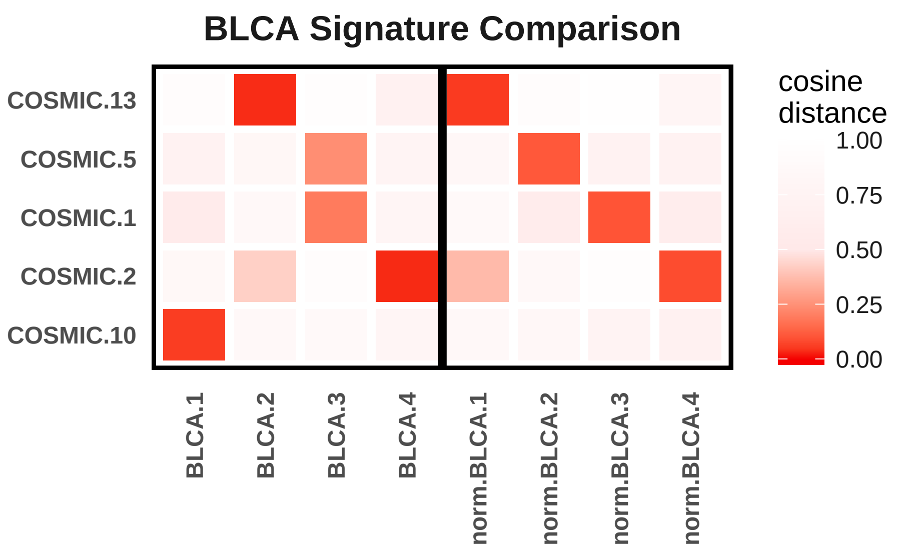
If mutational signatures are extracted from raw mutation counts, the presence of high mutation burden samples in a dataset may prevent precise identification of mutational signatures that are relevant in a number of low-mutation burden tumor genomes. Therefore, it may be desirable to level the weight of all samples in the dataset. This can be achieved by sample-wise mutation count normalization. In mutSignatures, normalization can be applied by setting the argument `approach="freq"`. Here, we analyze the same TCGA datasets as before, and compare the results obtained using normalized counts (here) and raw counts (above).

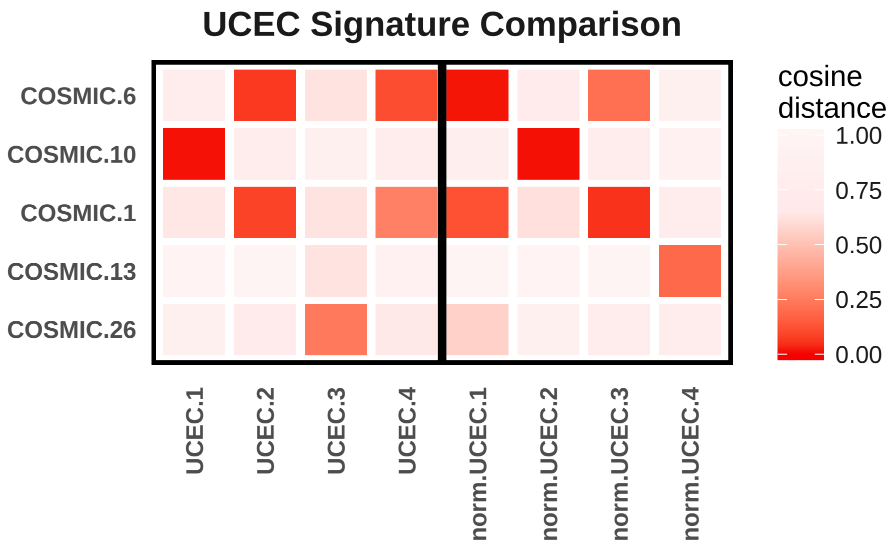
```r
# Norm count NMF analysis
norm.results <- sapply(names(mutCount.list), function(x){

  # Get k, same as defined before
  my.k <- k.byDset[x]

  # Get the 3-nucleotide mutCounts & set params
  xx <- mutCount.list[[x]]$mt3c

  norm.params <- setMutClusterParams(num_processesToExtract = my.k,
                                     num_totIterations = 200,
                                     num_parallelCores = 12,
                                     debug = TRUE,
                                     approach = "freq", # <----- norm!
                                     algorithm = "brunet")
  # Analyze and return
  decipherMutationalProcesses(input = xx,
                              params = norm.params)
}, simplify = FALSE)

# Rename resulting signatures consistently
for (x in names(norm.results)) {
  tmp <- norm.results[[x]]$Results$signatures
  sigRange <- 1:length(getSignatureIdentifiers(tmp))
  norm.results[[x]]$Results$signatures <-
    setSignatureIdentifiers(tmp, paste("norm", x, sigRange, sep = "."))
}
# match signatures identified in norm vs. raw counts
for(x in names(norm.results)) {
  yy <- matchSignatures(norm.results[[x]]$Results$signatures,
                        std.results[[x]]$Results$signatures)
  print(yy$plot + ggtitle(paste(x, "Signatures")))
}
```



In many cases, mutational signatures extracted from normalized mutation counts were a very good match to those from raw counts (ACC, LIHC, ESCA, and LUAD TCGA datasets). In some instances (BLCA or UCEC TCGA datasets), signatures extracted using raw mutation counts matched only in part (3 out of 4 signatures) those from normalized counts. This suggests that signatures specifically found in high-mutation burden tumors may prevent identification of signatures active in a number of low-mutation burden samples. In BLCA datasets, the signature specifically identified from raw mutaitons matched COSMIC 10, a mutational pattern found in tumors with hyper-mutator phenotype. This signature prevented precise identification of signatures matching COSMIC 1 and 5. Interestingly, in the UCEC TCGA dataset, a COSMIC 10-like signature was identified using either approach. This is suggestive that COSMIC 10 is not just an outlier in UCEC, but is present in a considerable fraction of UCEC tumors. Analysis of normalized counts from UCEC facilitated the correct identification of an APOBEC-related signature (COSMIC 13), which was not revealed by raw counts analysis.

```r
# BLCA Signature Comparison
print(matchSignatures(
    cosmix[c(13, 5, 1, 2, 10)],
    cbind(std.results[["BLCA"]]$Results$signatures,
          norm.results[["BLCA"]]$Results$signatures))$plot +
      geom_vline(xintercept = 4.5, colour = "black", size = 1.5) +
      theme(panel.border = element_rect(colour = "black", fill=NA, size=1.5),
            axis.text.x = element_text(margin = margin(2, 0, 0, 0, unit = 'mm'))) +
      ggtitle("BLCA Signature Comparison")
    )

# UCEC Signature Comparison
print(matchSignatures(
    cosmix[c(6, 10, 1, 13)],
    cbind(std.results[["UCEC"]]$Results$signatures,
          norm.results[["UCEC"]]$Results$signatures))$plot +
      geom_vline(xintercept = 4.5, colour = "black", size = 1.5) +
      theme(panel.border = element_rect(colour = "black", fill=NA, size=1.5),
            axis.text.x = element_text(margin = margin(2, 0, 0, 0, unit = 'mm'))) +
            ggtitle("UCEC Signature Comparison")
    )
```



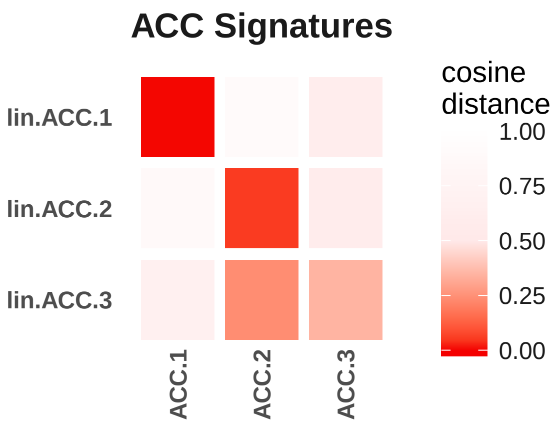


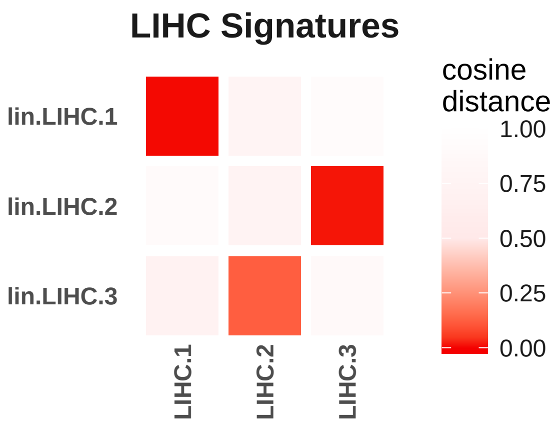
## Signatures derived using Lin's alternative NMF method

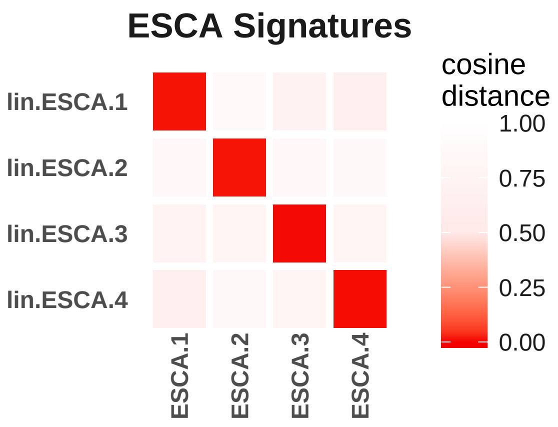
The `mutSignatures` software implements the same Brunet's algorithm that was used in the WTSI framework for performing NMF. However, an alternative algorithm is also available in `mutSignatures`. This is the multiplicative update NMF method described by Lin (Lin, 2007; https://doi.org/10.1109/tnn.2007.895831). Lin's modified multiplicative update algorithm enforces convergence, has similar computational complexity per iteration as the original NMF algorithm, and was previously applied to the analysis of genomic and biomedical data. The use of this algorithm may support identification of reliable and consistent mutational signatures (highly similar mutational patterns that are identified by two alternative algorithms), and may help extracting novel signatures with different clinical predictive power as compared to the COSMIC signatures. To make use ot the Lin's algorithm, it is sufficient to set the argument `algorithm = "lin"`. Here's an example using Lin's algorithm and raw mutation counts. Note, that the Lin's NMF procedure is compatible with normalized counts as well.

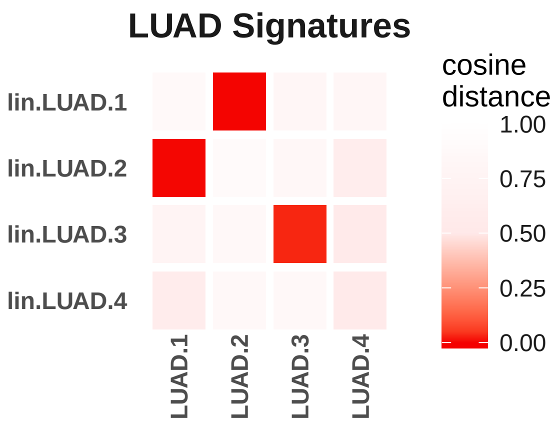
```
# Norm count NMF analysis
lin.results <- sapply(names(mutCount.list), function(x){

  # Get K, same as defined before
  my.k <- k.byDset[x]

  # Get the 3-nucleotide mutCounts
  xx <- mutCount.list[[x]]$mt3c

  # Params
  lin.params <- setMutClusterParams(num_processesToExtract = my.k,
                                    num_totIterations = 200,
                                    num_parallelCores = 12,
                                    debug = TRUE,
                                    approach = "counts",
                                    algorithm = "lin" # <------ Lin!
                                    )
  # Analyze and return
  decipherMutationalProcesses(input = xx,
                              params = lin.params)
}, simplify = FALSE)
```

```r
# Rename resulting signatures consistently
for (x in names(norm.results)) {
  tmp <- lin.results[[x]]$Results$signatures
  sigRange <- 1:length(getSignatureIdentifiers(tmp))
  lin.results[[x]]$Results$signatures <-
    setSignatureIdentifiers(tmp, paste("lin", x, sigRange, sep = "."))
}

# match signatures identified in norm vs. raw counts
for(x in names(lin.results)) {
  yy <- matchSignatures(lin.results[[x]]$Results$signatures,
                        std.results[[x]]$Results$signatures)
  print(yy$plot + ggtitle(paste(x, "Signatures")))
}
```



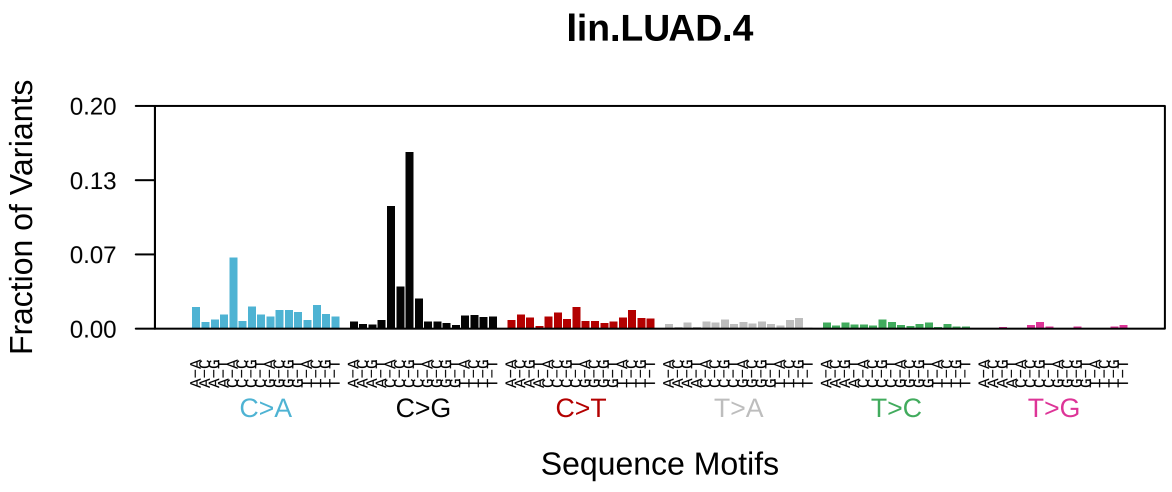






Notably, most of the mutational signatures identified before (Brunet's algorithm) were found also by running the alternaitve update algorithm proposed by Lin. This is indicative of the reliability of mutational signatures characterized in datasets such as ESCA, LIHC, and ACC. However, not all signatures returned by the Lin's approach were a match to previously characterized mutational signatures. For example, one of the signatures extracted from LUAD (*lin.LUAD.4*), was very different compared to any other COSMIC signature. It may be interesting to further analyze this signature, or better characterize signatures specific to the Lin's extraction approach to determine their relationship with clinical features and patient outcomes.

```r
# Visualize Lin's signature #4 from LUAD
plot(lin.results$LUAD$Results$signatures, signature  = 4)
```



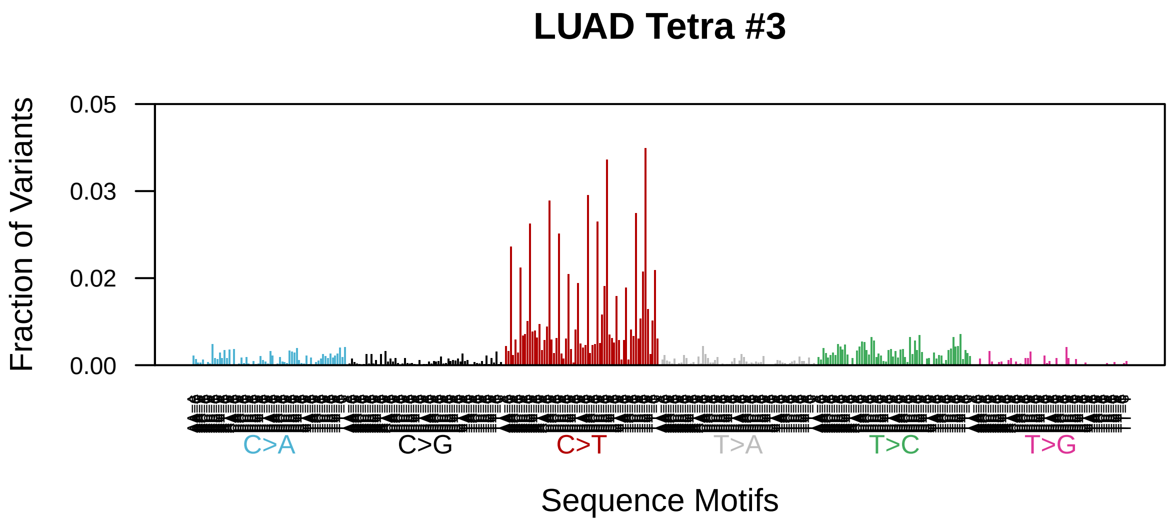
## Tetra-nucleotide mutational signatures from Lung Cancer

The analysis of tetra-nucleotide mutational signatures was recently reported by Wormald et al, 2018 (https://doi.org/10.1093/carcin/bgx133). Authors demonstrated that mutational signatures have context-specificity that is not just limited to the 3-nucleotide context, but extends further. Our framework is compatible with the analysis of extended non-standard mutation types, and comes with a set of tools for signature semplification and comparison. Here, we are analyzing tetra-nucleotide mutational signatures from the LUAD TCGA dataset. The tetra-nucleotide mutational signatures are computed using tetra-nuleotide 'Mutation Counts'-objects, and by using the same parameters as the previous run. Because of the increased number of mutation types (n=384), this analysis usually takes much longer than a standard tri-nucleotide signature extraction.

```
# Parameters are the same as before
tetra.params <- setMutClusterParams(num_processesToExtract = 4,
                                    num_totIterations = 200,
                                    num_parallelCores = 5,
                                    debug = TRUE,
                                    approach = "counts",
                                    algorithm = "brunet")

# Get tetra-mut LUAD counts
tetra.counts <- mutCount.list$LUAD$mt4c

# Extract signatures
tetra.luad <- decipherMutationalProcesses(input = tetra.counts,
                                          params = tetra.params)
```

It is possible to visualize the tetra-nucleotide signatures using the `plot()` method as before. This generates a barplot where extended mutation types are grouped by the corresponding single nucleotide variant. Interestingly, comparing tetra- and tri-nucleotide mutational signatures extracted from the same dataset and using the same parameters shows consistent mutational profiles. However, a direct comparison is not possible since the mutation types are different. Likewise, signature matching using standard and tetra-mutations counts is not allowed because of non-compatible mutation types.

```
plot(tetra.luad$Results$signatures, signature = 3, main = "LUAD Tetra #3")
```

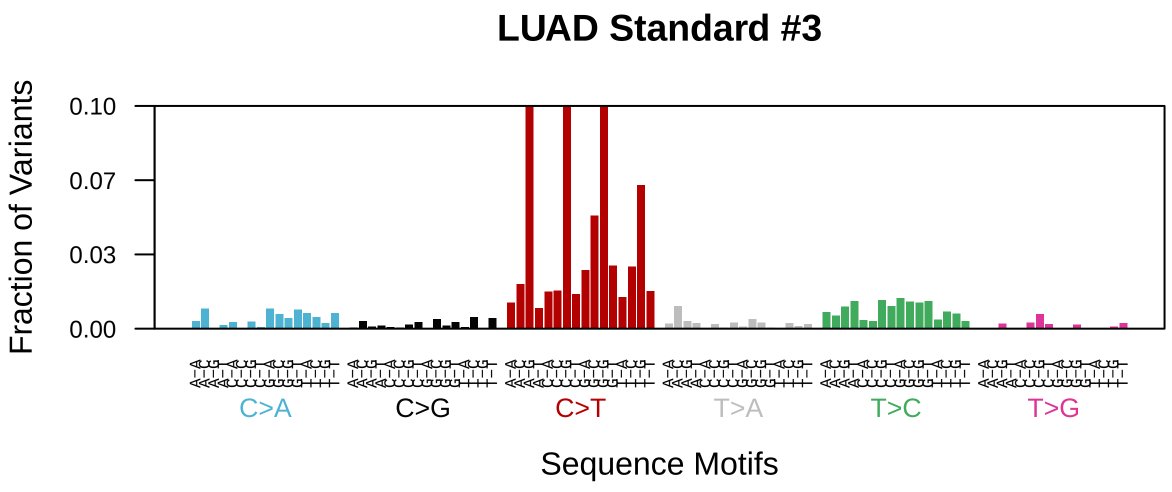


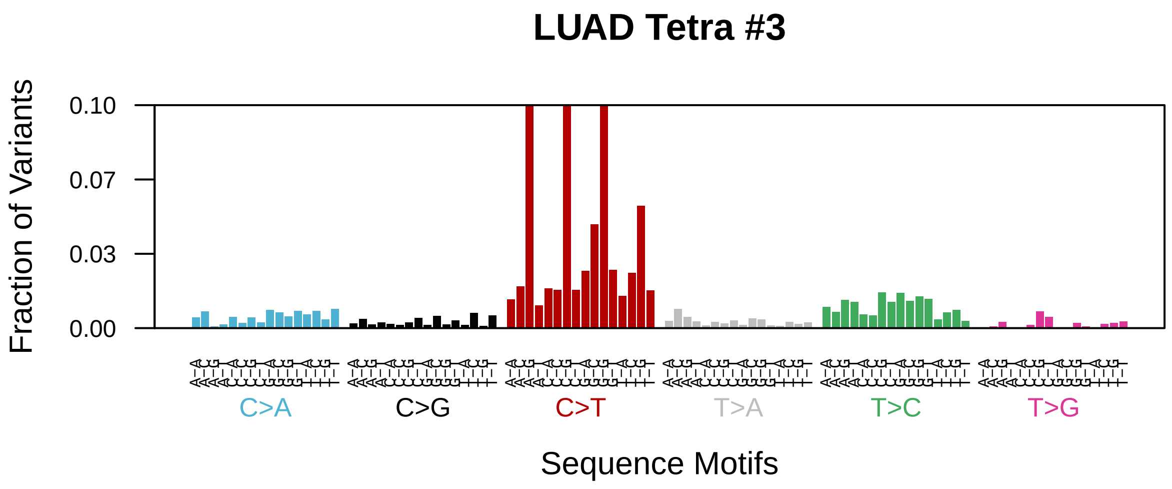
In order to facilitate signature comparison, extended mutation types can be simplified, or coerced to the corresponding tri-nucleotide format, using the `simplifySignatures()` function. By default, this returns a 'Mutation signatures' object, which in turn can be used for visualization or signature matching. Notably, extraction of either tri- or tetra-nucleotide mutation types returned very similar results in this case.

```
print(tetra.luad$Results$signatures)
```

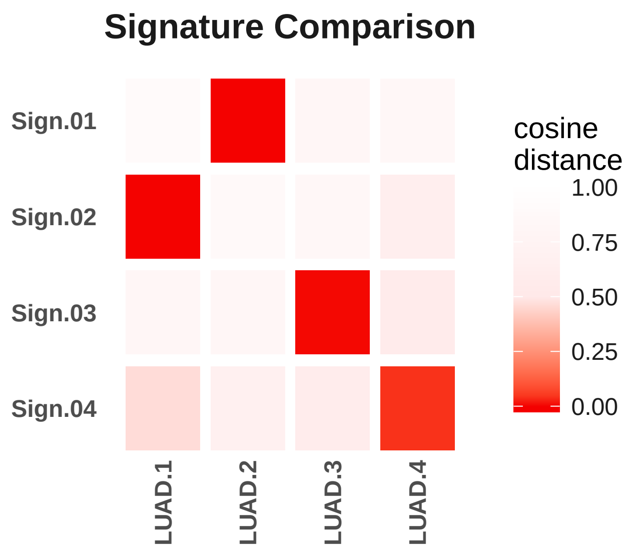
```
##  Mutation Signatures object - mutSignatures
##
##  Total num of Signatures: 4
##  Total num of MutTypes: 384
##
##     Sign.1   Sign.2   Sign.3   Sign.4
##     ------   ------   ------   ------
##   + 0.0003   0.0069   0.0018   0.0032  +  AA[C>A]A
##   + 0.0010   0.0052   0.0012   0.0045  +  AA[C>A]C
##   + 0.0006   0.0038   0.0005   0.0028  +  AA[C>A]G
##   + 0.0004   0.0045   0.0005   0.0022  +  AA[C>A]T
##   + 0.0007   0.0016   0.0003   0.0041  +  AA[C>G]A
##   + 0.0004   0.0009   0.0013   0.0011  +  AA[C>G]C
##   + 0.0003   0.0006   0.0006   0.0010  +  AA[C>G]G
##   + 0.0004   0.0009   0.0003   0.0035  +  AA[C>G]T
##   + 0.0015   0.0016   0.0037   0.0029  +  AA[C>T]A
##   + 0.0001   0.0015   0.0027   0.0003  +  AA[C>T]C
##     ......   ......   ......   ......
```

```
plot(std.results$LUAD$Results$signatures, ylim = c(0, 0.1),
     signature = 3, main = "LUAD Standard #3")
plot(simplifySignatures(tetra.luad$Results$signatures),
      ylim = c(0, 0.1), signature = 3, main = "LUAD Tetra #3")
```

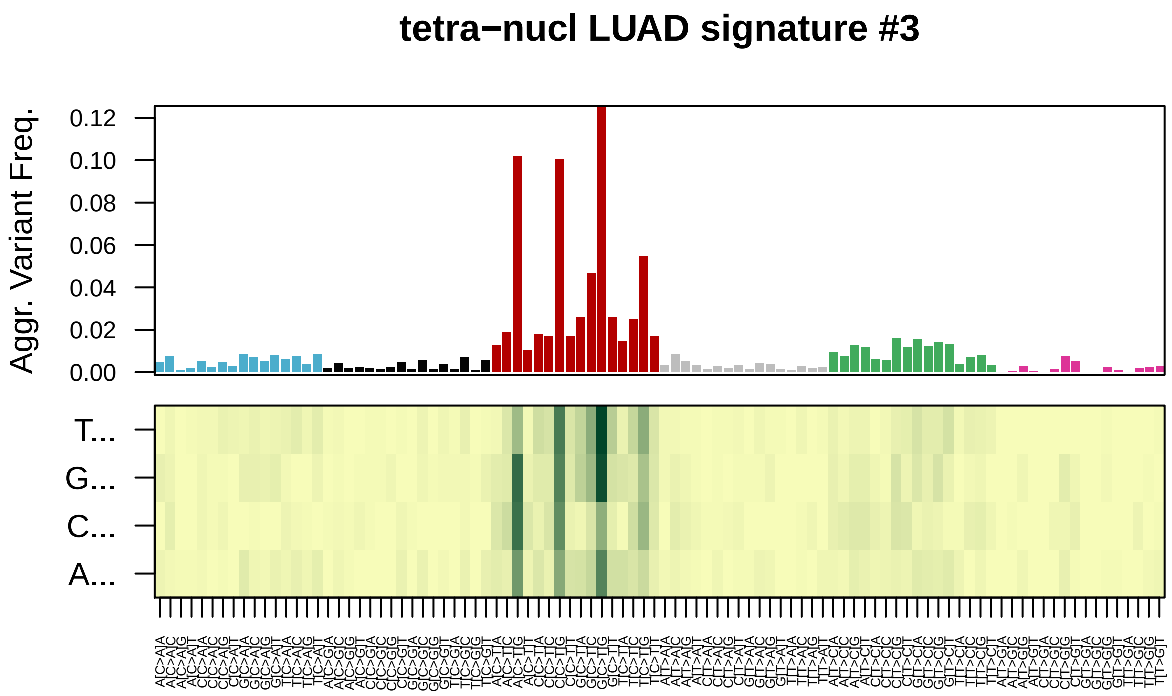




```
# Match LUAD signatures
mtch3 <- matchSignatures(simplifySignatures(tetra.luad$Results$signatures),
                         std.results$LUAD$Results$signatures)
print(mtch3$plot)
```



To further explore nucleotide specificity in the DNA regions flanking somatic mutations, it is possible to look into tetra-nucleotide signatures at higher resolution. When information about single mutation types is required, we recommend simplyfing mutational signatures setting the argument `asMutationSignatures = FALSE`. This returns a list of data.frames. Each element of the list corresponds to a simplified signature, including frequencies for each core tri-nucleotide variant (rows) and each possible flanking nucleotide (columns). Each *cell* in the data.frame indicates the relative frequency of a specific mutation. Note that rows are alphabetically sorted and not grouped by mutation type. Before proceeding, it is recommended to sort the rows in these data.frames by mutation type, using the `sortByMutations()` function. A heat-map like visualization of the tetra-nucleotide mutational signature can be obtained using the `image()` function from the `graphics` package, or via *ggplot2*.



## Correlation between Mutational Signatures and Clinical Features

Mutational signatures may be prognostic of patient outcomes and clinical features. Here, we are examining the correlation between mutational signatures extracted from the LUAD TCGA dataset and smoking status. Patient smoking status corresponds to the variable named *TOBACCO_SMOKING_HISTORY_INDICATOR*, which is included in the clinical data.frames downloaded before. Briefly, smoking history is encoded as follows:

- value = **'1'**, means **life-long non-smoker**
- value = **'2'**, means **current smoker**
- value %in% c(**'3'**, **'4'**, **'5'**), means **reformed smoker**

Let's retrieve the TCGA identifiers corresponding to tumors from smokers and non-smokers, and then analyze mutational signature exposures with respect to smoking status (reformed smokers and missing values will be excluded from this analysis).

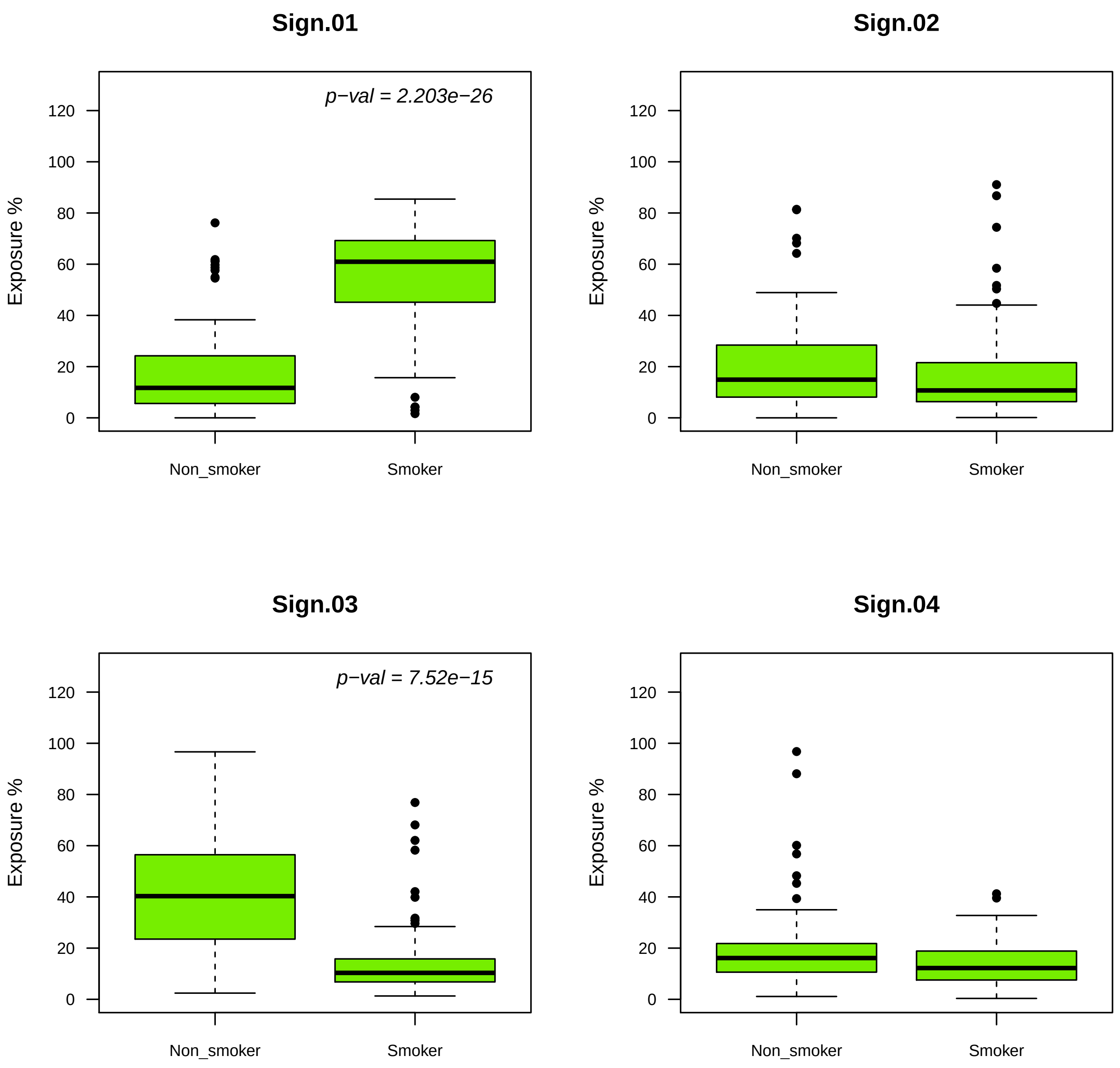
```
# Smoking status
head(clin.list$LUAD$TOBACCO_SMOKING_HISTORY_INDICATOR, n = 20)
```

```
##  [1] "4" "3" "3" "4" "2" "4" "3" "4" "4" "2" "3" "3" "4" "4" "4" "3" "1"
## [18] "4" "4" "4"
```

```r
# Define smoking status
luad_by_smoking <- list(
  nonsmok = clin.list$LUAD$CASE_ID[clin.list$LUAD$TOBACCO_SMOKING_HISTORY_INDICATOR == "1"],
  smokers = clin.list$LUAD$CASE_ID[clin.list$LUAD$TOBACCO_SMOKING_HISTORY_INDICATOR == "2"])

# Getexposures in LUAD
luad.df <- as.data.frame(std.results$LUAD$Results$exposures)

# Percentize
luad.df <- 100 * luad.df / apply(luad.df, 1, sum)

# Attach smoking status
luad.df$smoking <- sapply(rownames(luad.df), function(id) {
  if (id %in% luad_by_smoking$nonsmok) {
    "Non_smoker"
  } else if (id %in% luad_by_smoking$smokers) {
    "Smoker"
  } else {
    NA
  }
})

# Remove excluded cases
luad.df <- luad.df[!is.na(luad.df$smoking),]

# Check patients included
table(luad.df$smoking)
```

```
##
## Non_smoker     Smoker
##         72        107
```

```r
# Plot exposures vs smoking status
par(mfrow = c(2,2))
for (sig in grep("Sign", colnames(luad.df), value = TRUE)) {
  x <- split(luad.df[,sig], f = factor(luad.df$smoking))
  tx <- t.test(x$Non_smoker, x$Smoker)$p.value
  boxplot(x, pch = 19, col = "chartreuse2",
          ylim = c(0, 130), main = sig, las = 1,
          cex.axis = 0.85, ylab = "Exposure %")
  if (tx < 0.00001) {
  text(2.45, 125,
       labels = paste("p-val =", format(tx, digits = 4, nsmall = 4)),
       pos = 2, font = 3)
  }
}
par(mfrow = c(1,1))
```



Interestingly, we identified two mutational signatures whose exposures had a very strong correlation with smoking status. Specifically, LUAD signature #1 (matching COSMIC #4, see pag. 8) was enriched in tumors from smokers; conversely, LUAD signature #3 (matching COSMIC signature #1) had a more important contribution to the total number of mutations found in tumors from never-smokers. This example showed a stron link between mutational signatures and clinical features, confirming that signatures can help identifying the molecular causes of cancer (for example, cigarette smoke carcinogens), and may help predicting or revealing clinical features, such as smoking status.

## Conclusions

Thank you for using *mutSignatures*. For questions and information, please check the official website, at: http://www.mutsignatures.org/

## Session Info

```
## 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] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] gridExtra_2.3       ggplot2_2.2.1       mutSignatures_1.3.7
## [4] Biobase_2.38.0      BiocGenerics_0.24.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.15          pracma_2.1.4          XVector_0.18.0
##  [4] GenomeInfoDb_1.14.0   compiler_3.4.3        pillar_1.1.0
##  [7] RColorBrewer_1.1-2    plyr_1.8.4            zlibbioc_1.24.0
## [10] bitops_1.0-6          iterators_1.0.9       tools_3.4.3
## [13] rngtools_1.2.4        digest_0.6.15         evaluate_0.10.1
## [16] tibble_1.4.2          gtable_0.2.0          gridBase_0.4-7
## [19] NMF_0.21.0            rlang_0.1.6           foreach_1.4.4
## [22] registry_0.5          yaml_2.1.16           GenomeInfoDbData_1.0.0
## [25] stringr_1.3.0         pkgmaker_0.22         cluster_2.0.6
## [28] knitr_1.19            S4Vectors_0.16.0      IRanges_2.12.0
## [31] stats4_3.4.3          rprojroot_1.3-2       grid_3.4.3
## [34] rmarkdown_1.8         reshape2_1.4.3        magrittr_1.5
## [37] GenomicRanges_1.30.1  backports_1.1.2       scales_0.5.0
## [40] codetools_0.2-15      htmltools_0.3.6       colorspace_1.3-2
## [43] xtable_1.8-2          labeling_0.3          stringi_1.1.7
## [46] proxy_0.4-21          RCurl_1.95-4.10       lazyeval_0.2.1
## [49] doParallel_1.0.11     munsell_0.4.3
```