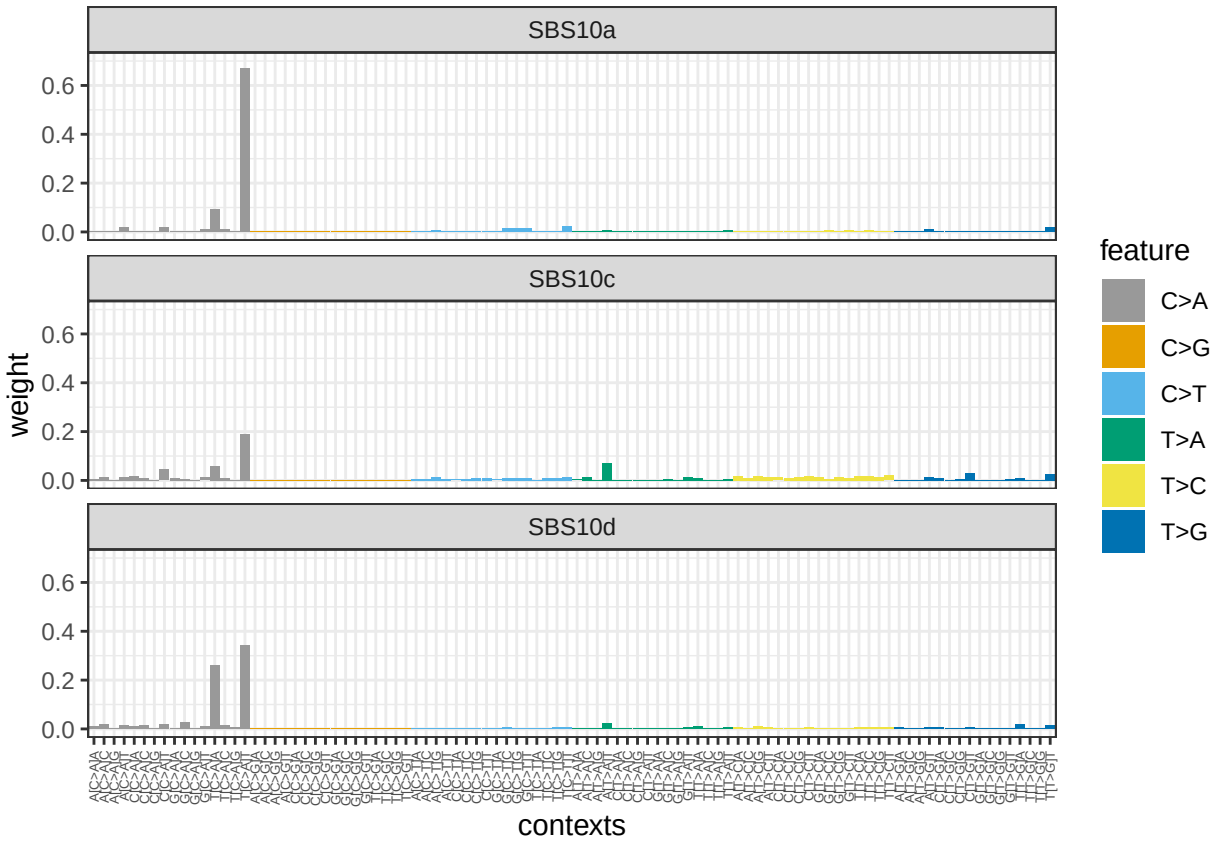
Generate input vectors according to definitions of SBS signatures

In this Rmarkdown document (which contains results dynamically generated from the simulation code displayed) we model input vectors according to SBS signatures and then extract signatures using NMF.

1 Load definitions of SBS signatures representing C>A substitutions

We focused on three signatures which represent C>A substitutions in a T[C>A]T context (SBS10a, SBS10c and SBS10d). First, we get the definition of the three signatures that we are going to focus on.

```
SBSsigs<-readRDS(file=paste0(path_to_data, "/cosmic.sbs.v3.2.march2021.GRCh37.rds"))

contexts <- colnames(SBSsigs)
contexts <- factor(contexts, levels=colnames(SBSsigs))
feature <- c(rep("C>A",16), rep("C>G",16) ,rep("C>T",16), rep("T>A",16), rep("T>C",16), rep("T>G",16))
feature <- factor(feature, levels=c("C>A","C>G","C>T","T>A","T>C","T>G"))

# Assign weights to the components of each signature
pdat <- data.frame(sig = c(rep("SBS10a", 96), rep("SBS10c", 96), rep("SBS10d", 96)),
                   contexts,
                   feature,
                   weight = c(t(as.vector(SBSsigs[13,])), # Signature SBS10a
                            t(as.vector(SBSsigs[15,])), # Signature SBS10c
                            t(as.vector(SBSsigs[16,]))) # Signature SBS10d
                   )

pdat$sig <- factor(pdat$sig, levels=c("SBS10a", "SBS10c", "SBS10d"))

# Plot signature definitions
ymax <- round(max(pdat$weight)+0.05,digits = 1)
ggplot(pdat, aes(x = contexts, y = weight, fill = feature)) +
  geom_bar(stat = "identity") + facet_wrap(. ~ sig, ncol = 1) + theme_bw() +
  scale_y_continuous(limits = c(0, ymax)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
  scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                        "C>T" = "#56B4E9", "T>A" = "#009E73",
                                        "T>C" = "#F0E442", "T>G" = "#0072B2"))
```



1.1 Simulate samples

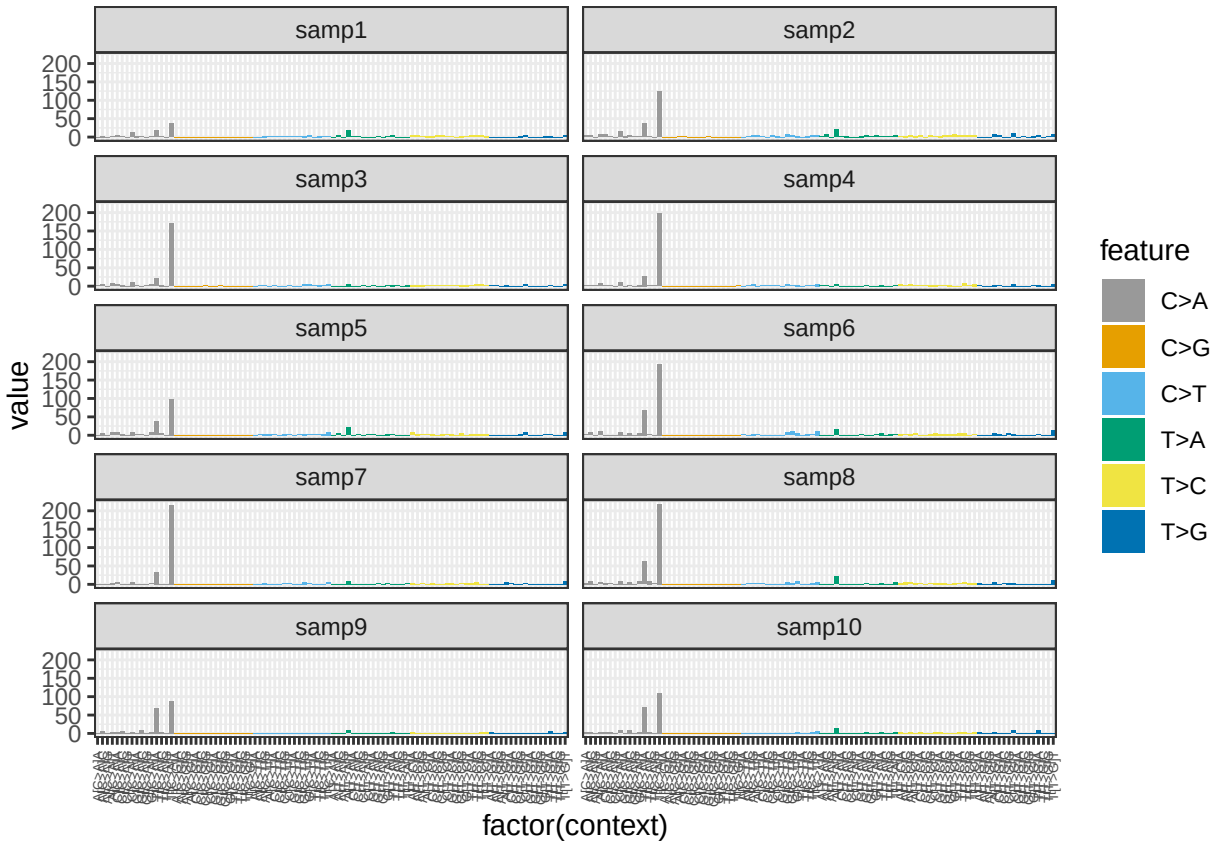
Here we simulate 250 input matrix with 500 samples from these signatures. For simulations, we randomly indicate if one, two or three of the signatures are acting on the sample. If more than one signature affects the sample profile, we assume that one of them has a higher activity. In 80% of cases, the SBS10c (which has higher divergence in mutation types) is selected as the one with highest activity. That's for having a good representation of this signature in our dataset.

```
# For reproducibility
set.seed(3)

# List of contexts
contexts <- colnames(SBSsigs)
contexts <- factor(contexts, levels=colnames(SBSsigs))

# List with probability of each trinucleotide context in each signature
sbs10a <- as.vector(t(SBSsigs[13,]))
sbs10c <- as.vector(t(SBSsigs[15,]))
sbs10d <- as.vector(t(SBSsigs[16,]))
prob <- list(sbs10a,sbs10c,sbs10d)
names(prob)<-c("sbs10a","sbs10c","sbs10d")

# Input matrix is declared (will be filled in the for loop)
input_matrix <- c()
```

```r
# A for loop where each loop a new sample is created
lSimulations <- list()
lSimulations <- lapply(1:Nsim, function(thisSim){
  for(i in 1:500) {

    ## Number of signatures acting in the sample
    nsig <- sample(c(1:3), 1)

    if (nsig == 1){
      # signature
      sig <- sample(c("sbs10a","sbs10c","sbs10d"), nsig)
      # number of mutations
      nmut <- sample(c(seq(200,300,1)),1)
      # context of mutations
      smut <- sample(contexts, nmut, replace = TRUE, prob = prob[[sig]])
      # Result is added to input matrix
      samp <- table(smut)
      input_matrix <- rbind(input_matrix, samp)
    }
    else if (nsig == 2){
      # signatures
      sig <- sample(c("sbs10a","sbs10c","sbs10d"), nsig, prob = c(0.1,0.8,0.1))
      # number of mutations
      nmut1 <- sample(c(seq(200,300,1)),1)
      nmut2 <- sample(c(seq(50,150,1)),1)
      # context of mutations
      sig1 <- sample(sig,1)
      smut1 <- sample(contexts, nmut1, replace = TRUE, prob = prob[[sig1]])
      smut1 <- table(smut1)
      sig2 <- sig[sig!=sig1]
      smut2 <- sample(contexts, nmut2, replace = TRUE, prob = prob[[sig2]])
      smut2 <- table(smut2)
      # Result is added to input matrix
      samp<-c()
      for (i in 1:96){
        s <- sum(smut1[i],smut2[i])
        samp <- c(samp,s)
      }
      input_matrix <- rbind(input_matrix, samp)
    }
    else if (nsig == 3) {
      # signature with highest activity
      sig <- sample(c("sbs10a","sbs10c","sbs10d"), 1, prob = c(0.1,0.8,0.1))
      # number of mutations
      nmut <- sample(c(seq(200,300,1)),1)
      # context of mutations
      smut <- sample(contexts, nmut, replace = TRUE, prob = prob[[sig]])
      smut <- table(smut)
      # other signatures
      sigs <- names(prob)[names(prob) != sig]
      # number of mutations
      nmuts <- sample(c(seq(50,150,1)),1)
      # context of mutations
```

```
      smut1 <- sample(contexts, nmuts, replace = TRUE, prob = prob[[sigs[1]]])
      smut2 <- sample(contexts, nmuts, replace = TRUE, prob = prob[[sigs[2]]])
      smut1 <- table(smut1)
      smut2 <- table(smut2)
      ## Result is added to input matrix
      samp<-c()
      for (i in 1:96){
        s <- sum(smut[i],smut1[i],smut2[i])
        samp <- c(samp,s)
      }
      input_matrix <- rbind(input_matrix, samp)
    }
  }

  # Sample names are given
  rownames(input_matrix) <- paste0("samp", seq_len(nrow(input_matrix)))
  colnames(input_matrix) <- contexts
  lSimulations[[thisSim]] <- input_matrix
})

saveRDS(lSimulations, file = paste0(path_to_data, "/1_InputMatrices_", Nsim, "xSimulations_500samples_S
```

```
lSimulations <- readRDS(paste0(path_to_data,"/1_InputMatrices_", Nsim, "xSimulations_500samples_SBS10c.

# Plots first ten samples of the 1st simulated input matrix
input_matrix <- lSimulations[[1]]
pdat <- reshape2::melt(input_matrix[1:10,])
colnames(pdat) <- c("sample", "context", "value")
pdat$feature <- c(rep("C>A",160), rep("C>G",160) ,rep("C>T",160), rep("T>A",160),
                  rep("T>C",160), rep("T>G",160))

ggplot(pdat, aes(x = factor(context), y = value, fill = feature)) +
  geom_bar(stat = "identity") + facet_wrap(. ~ sample, ncol = 2) + theme_bw() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
  scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                        "C>T" = "#56B4E9", "T>A" = "#009E73",
                                        "T>C" = "#F0E442", "T>G" = "#0072B2"))
```



1.2 Estimate signatures from samples

1.2.1 Brunet NMF

Next we run NMF with k=3 to see if we can reconstruct the signatures:

```
#lSimulations <- readRDS(paste0(path_to_data,"/1_InputMatrices",Nsim,"xSimulations_500samples_SBS10c.rd
lNMF <- list()
input_matrix<-0
lNMF <- lapply(1:Nsim, function(thisSim){
  print(thisSim)
  input_matrix <- lSimulations[[thisSim]]

  # I have added 0.001 in all samples in this row
  col<-which(colSums(input_matrix) == 0)
  input_matrix[,col]<-0.00001

  # Run the NMF.
  sigs <- NMF::nmf(t(input_matrix), rank = 3, seed = 222222, nrun = 100, method = "brunet")

  # Format for plotting counts of each context per signature
  sigmat <- basis(sigs)

  # Normalize signature matrix
  sigmat<-apply(sigmat,2,function(x){x/sum(x)})
```

```
  lNMF[[thisSim]] <- sigmat
})
saveRDS(lNMF, file = paste0(path_to_data, "/2_NMF_", Nsim,"xSimulations_500samples_SBS10c.rds"))
```

Now, we match definitions of the three signatures obtained from the different NMF runs and calculated the mean and standard error.

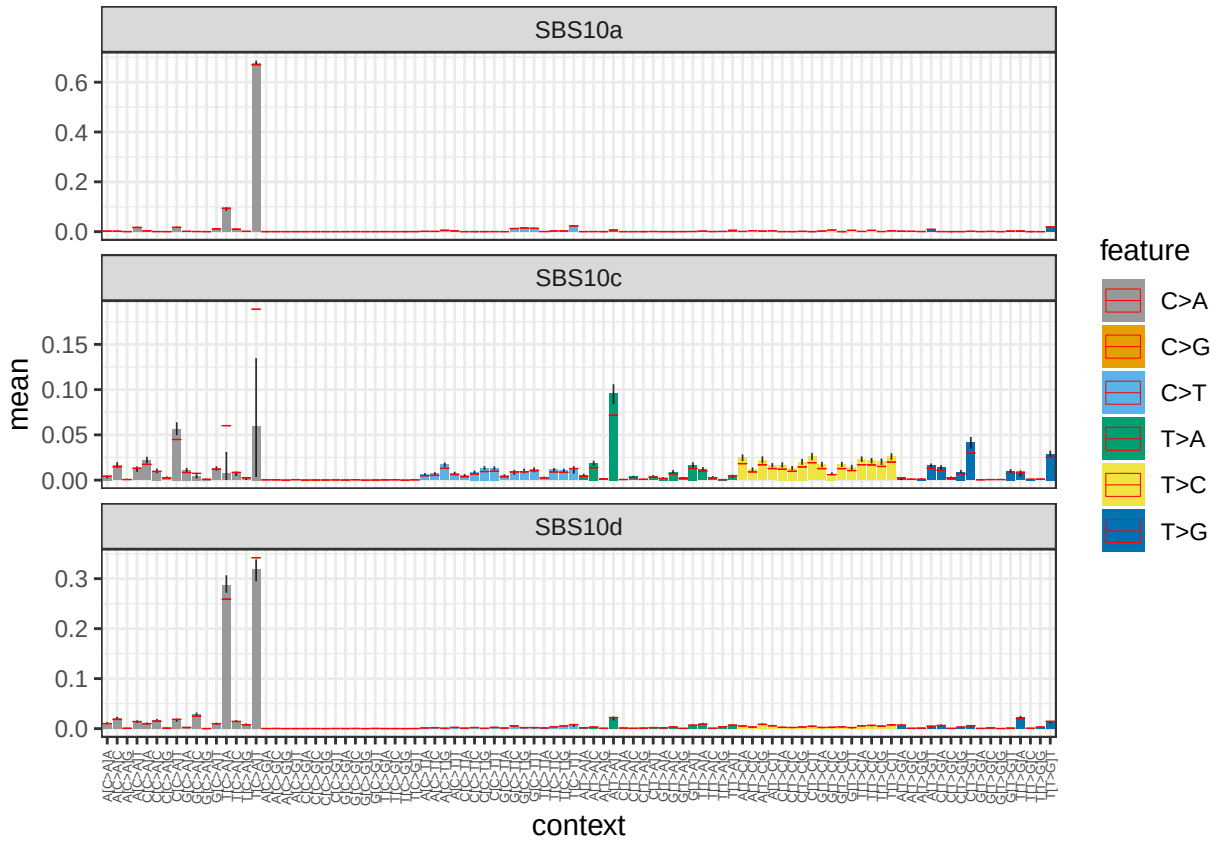
```
lNMF <- readRDS(paste0(path_to_data,"/2_NMF_", Nsim,"xSimulations_500samples_SBS10c.rds"))

# Definition from the original matrix
Sigs <- c("SBS10a", "SBS10c", "SBS10d")
sigs <- t(SBSsigs[Sigs,])

# Match signatures and check that only matches with one
lmatches <- lapply(1:Nsim, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
    d <- length(unique(match))
    match <- c(match, d)
    return(match)
})

# Get definitions of matched signatures
lSimDefinitions <- lapply(1:Nsim, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
    colnames(sigmat) <- match
    sigmat <- sigmat[,Sigs]
    return(sigmat)
})

lSimDefinitions <- lapply(Sigs, function(thisSig){
  sdefs <- lapply(1:Nsim, function(thisSim){
    s <- lSimDefinitions[[thisSim]][,thisSig]})
  sdefs <- as.data.frame(do.call(cbind, sdefs))
  return(sdefs)
})
names(lSimDefinitions)<-Sigs

# Calculate the standard error of simulated definitions
lVariabilityDefinitions <- lapply(Sigs, function(thisSig) {
```

```
    defs <- lSimDefinitions[[thisSig]]

    # Statistics
    min <- apply(defs, 1, min)
    max <- apply(defs, 1, max)
    median <- apply(defs, 1, median)
    mean <- rowMeans(defs)
    sd <- apply(defs, 1, sd)
    se <- sd/sqrt(ncol(defs))
    nVar <- as.data.frame(cbind(median=median, min=min, max=max, mean=mean, sd=sd, se=se))
    return(nVar)
})
names(lVariabilityDefinitions)<-Sigs
```

Finally, we plot the definitions of SBS signatures. Error bars show the range of values obtained for each component. Horizontal red lines indicate the original weigth of each component.

```
## Prepare data for plotting and produce the figure
# Get mean values for bars and melt
mean <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "mean")))
colnames(mean) <- colnames(SBSsigs)
mean <- reshape2::melt(t(mean))
colnames(mean) <- c("context", "sig", "mean")

# Get min for error bars and melt
min <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "min")))
colnames(min) <- colnames(SBSsigs)
min <- reshape2::melt(t(min))
colnames(min) <- c("context", "sig", "min")

# Get max for error bars and melt
max <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "max")))
colnames(max) <- colnames(SBSsigs)
max <- reshape2::melt(t(max))
colnames(max) <- c("context", "sig", "max")

# Join standard errors
pdat <- left_join(mean, min, by=c("context", "sig"))
pdat <- left_join(pdat, max, by=c("context", "sig"))

# Add original signatures
sbs <- reshape2::melt(sigs)
colnames(sbs) <- c("context", "sig", "weights")
pdat <- left_join(pdat, sbs, by=c("context", "sig"))

# Prepare for plotting
pdat$feature <- c(rep(c(rep("C>A",16), rep("C>G",16), rep("C>T",16), rep("T>A",16),
                        rep("T>C",16), rep("T>G",16)),3))
pdat$feature <- factor(pdat$feature, levels=c("C>A","C>G","C>T","T>A","T>C","T>G"))

yaxis.max <- round(max(pdat$max)+0.05,digits = 1)
yaxis.min <- round(min(pdat$min)-0.05,digits = 1)
```

```
p <- ggplot(pdat, aes(x=context, y=mean, fill=feature)) +
    geom_bar(stat = "identity") + facet_wrap(. ~ sig, ncol = 1, scales="free_y") + theme_bw() +
    geom_linerange(aes(ymin=min, ymax=max), colour="black", alpha=0.8, size=0.3) +
    geom_crossbar(aes(y=weights, ymin=weights, ymax=weights), colour="red", alpha=0.6, size=0.1) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
    #scale_y_continuous(limits = c(yaxis.min, yaxis.max)) +
    scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                          "C>T" = "#56B4E9", "T>A" = "#009E73",
                                          "T>C" = "#F0E442", "T>G" = "#0072B2"))
print(p)
```




```
## Save output
# cairo_pdf(file.path(OUTPUT_DIR, "SuppFig43_2_SBSSimulations_250xInputMatrices_SBS10c.pdf"), height =
# print(p); dev.off()
```

Here we can observe that the weights of T[C>A]A & T[C>A]T significantly decrease in the SBS10c signature. Specifically, the weight of T[C>A]A mutations drops from 0.06 to 0.0071 [9.67e-7, 0.031]; while the weight of T[C>A]T decays from 0.19 to 0.059 [0.0035, 0.13]. That's because these mutations are counted for the SBS10a & SBS10d signatures.

1.2.2 SigProfilerExtractor

We first prepare data for SigProfilerExtractor, and run it. We performed this analysis to confirm what we observed with brunet NMF. Therefore, only 10 simulated input matrices has been used for this analysis.

```
lSimulations <- readRDS(paste0(path_to_data,"/1_InputMatrices",Nsim,"xSimulations_500samples_SBS10c.rds
for (i in 1:10){
    M <- t(lSimulations[[i]])
    M <- cbind(Mutation_type=rownames(M), M)
    write.table(M, paste0(path_to_data,"/SBS10c_simulations/", i,"_SBS10c_simulations.txt"), sep="\t",
}
```

After running SigProfilerExtractor (in the cluster), we match definitions of the three signatures obtained from the 10 runs and calculated the mean and standard error.

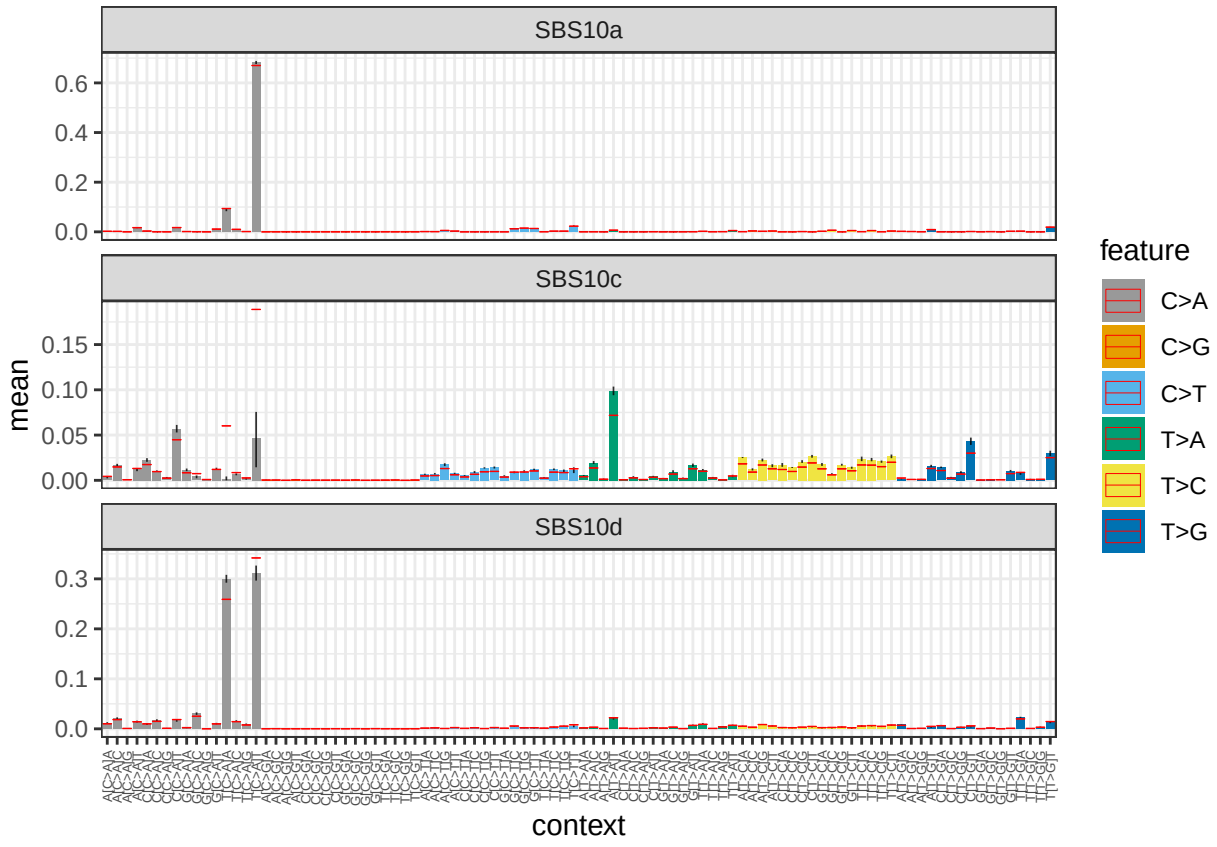
```
# Get paths of signature definition matrices
path <- paste0(path_to_sigprof, "/SBS10c")
allSigFiles <- unlist(lapply(path, function(thisPath){
    file <- list.files(thisPath, pattern="*.txt",full.names = TRUE)}))
numM <- length(allSigFiles)

# Load signature definition matrices
lNMF <- lapply(allSigFiles, function(file){
    defs <- read.table(file, sep="\t", header = T, check.names=FALSE)
    row.names(defs) <- defs[,1]
    defs <- defs[,-1]
})
names(lNMF)<-1:numM

# Definition from the original matrix
Sigs <- c("SBS10a", "SBS10c", "SBS10d")
sigs <- t(SBSsigs[Sigs,])

# Match signatures and check that only matches with one
lmatches <- lapply(1:10, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
    d <- length(unique(match))
    match <- c(match, d)
    return(match)
})

# Get definitions of matched signatures
lSimDefinitions <- lapply(1:10, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
```

```
    colnames(sigmat) <- match
    sigmat <- sigmat[,Sigs]
    return(sigmat)
})

lSimDefinitions <- lapply(Sigs, function(thisSig){
  sdefs <- lapply(1:10, function(thisSim){
    s <- lSimDefinitions[[thisSim]][,thisSig]})
  sdefs <- as.data.frame(do.call(cbind, sdefs))
  return(sdefs)
})
names(lSimDefinitions)<-Sigs

# Calculate the standard error of simulated definitions
lVariabilityDefinitions <- lapply(Sigs, function(thisSig) {
    defs <- lSimDefinitions[[thisSig]]

    # Statistics
    min <- apply(defs, 1, min)
    max <- apply(defs, 1, max)
    median <- apply(defs, 1, median)
    mean <- rowMeans(defs)
    sd <- apply(defs, 1, sd)
    se <- sd/sqrt(ncol(defs))
    nVar <- as.data.frame(cbind(median=median, min=min, max=max, mean=mean, sd=sd, se=se))
    return(nVar)
})
names(lVariabilityDefinitions)<-Sigs
```

Finally, we plot the definitions of SBS signatures. Error bars show the range of values obtained for each component. Horizontal red lines indicate the original weight of each component.

```
## Prepare data for plotting and produce the figure
# Get mean values for bars and melt
mean <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "mean")))
colnames(mean) <- colnames(SBSsigs)
mean <- reshape2::melt(t(mean))
colnames(mean) <- c("context", "sig", "mean")

# Get min for error bars and melt
min <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "min")))
colnames(min) <- colnames(SBSsigs)
min <- reshape2::melt(t(min))
colnames(min) <- c("context", "sig", "min")

# Get max for error bars and melt
max <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "max")))
colnames(max) <- colnames(SBSsigs)
max <- reshape2::melt(t(max))
colnames(max) <- c("context", "sig", "max")

# Join standard errors
pdat <- left_join(mean, min, by=c("context", "sig"))
```

```
pdat <- left_join(pdat, max, by=c("context", "sig"))

# Add original signatures
sbs <- reshape2::melt(sigs)
colnames(sbs) <- c("context", "sig", "weights")
pdat <- left_join(pdat, sbs, by=c("context", "sig"))

# Prepare for plotting
pdat$feature <- c(rep(c(rep("C>A",16), rep("C>G",16), rep("C>T",16), rep("T>A",16),
                        rep("T>C",16), rep("T>G",16)),3))
pdat$feature <- factor(pdat$feature, levels=c("C>A","C>G","C>T","T>A","T>C","T>G"))

yaxis.max <- round(max(pdat$max)+0.05,digits = 1)
yaxis.min <- round(min(pdat$min)-0.05,digits = 1)

p <- ggplot(pdat, aes(x=context, y=mean, fill=feature)) +
    geom_bar(stat = "identity") + facet_wrap(. ~ sig, ncol = 1, scales="free_y") + theme_bw() +
    geom_linerange(aes(ymin=min, ymax=max), colour="black", alpha=0.8, size=0.3) +
    geom_crossbar(aes(y=weights, ymin=weights, ymax=weights), colour="red", alpha=0.6, size=0.1) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
    #scale_y_continuous(limits = c(yaxis.min, yaxis.max)) +
    scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                          "C>T" = "#56B4E9", "T>A" = "#009E73",
                                          "T>C" = "#F0E442", "T>G" = "#0072B2"))
print(p)
```

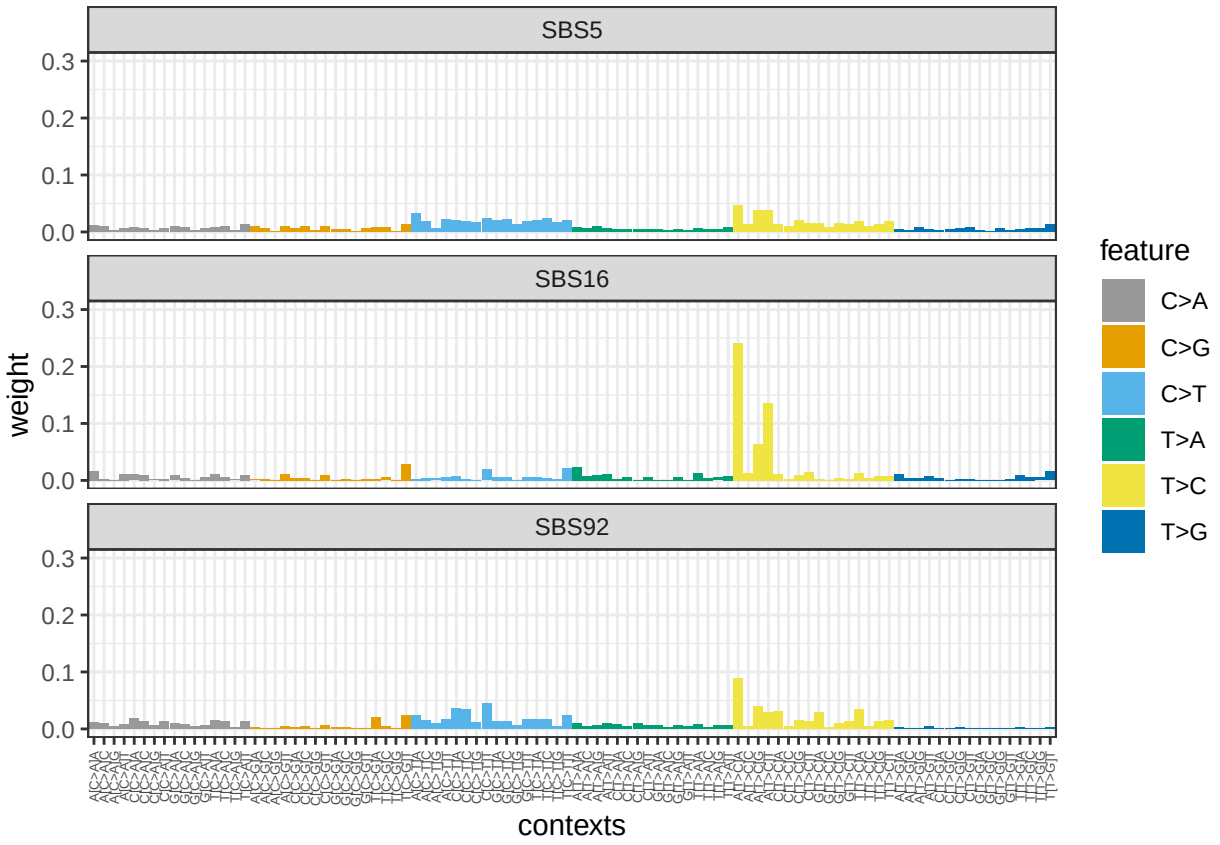


```
## Save output
# cairo_pdf(file.path(OUTPUT_DIR, "SuppFig51_2_SigProfilerExtractor_SBSSimulations_10xInputMatrices_SBS
# print(p); dev.off()
```

2 Load definitions of SBS signatures representing T>C substitutions

In this second example, we focused on three signatures which represent T>C substitutions in a A[T]A context (SBS5, SBS16 and SBS92). First, we get the definition of the three signatures that we are going to focus on.

```
SBSsigs<-readRDS(file=paste0(path_to_data, "/cosmic.sbs.v3.2.march2021.GRCh37.rds"))

contexts <- colnames(SBSsigs)
contexts <- factor(contexts, levels=colnames(SBSsigs))
feature <- c(rep("C>A",16), rep("C>G",16) ,rep("C>T",16), rep("T>A",16), rep("T>C",16), rep("T>G",16))
feature <- factor(feature, levels=c("C>A","C>G","C>T","T>A","T>C","T>G"))

# Assign weights to the components of each signature
pdat <- data.frame(sig = c(rep("SBS5", 96), rep("SBS16", 96), rep("SBS92", 96)),
                   contexts,
                   feature,
                   weight = c(t(as.vector(SBSsigs[5,])), # Signature SBS5
                            t(as.vector(SBSsigs[22,])), # Signature SBS16
                            t(as.vector(SBSsigs[76,]))) # Signature SBS92
                   )

pdat$sig <- factor(pdat$sig, levels=c("SBS5", "SBS16", "SBS92"))

# Plot signature definitions
ymax <- round(max(pdat$weight)+0.05,digits = 1)
ggplot(pdat, aes(x = contexts, y = weight, fill = feature)) +
  geom_bar(stat = "identity") + facet_wrap(. ~ sig, ncol = 1) + theme_bw() +
  scale_y_continuous(limits = c(0, ymax)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
  scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                        "C>T" = "#56B4E9", "T>A" = "#009E73",
                                        "T>C" = "#F0E442", "T>G" = "#0072B2"))
```



2.1 Simulate samples

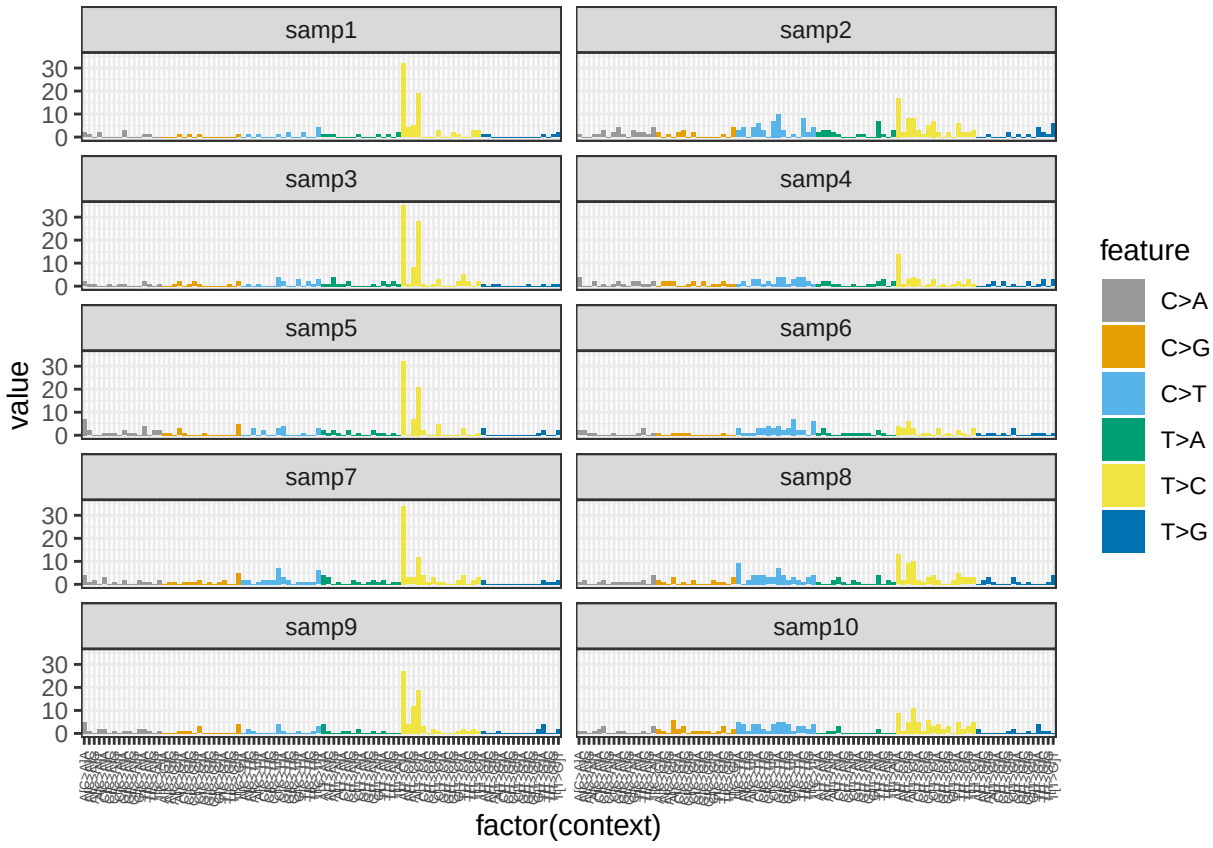
Here we simulate 250 input matrices which include 500 samples from these signatures and display the first 10. For simulations, we randomly indicate if one, two or three of the signatures are acting on the sample. If more than one signature affects the sample profile, we assume that one of them has a higher activity. In 80% of cases, the SBS10c (which has higher divergence in mutation types) is selected as the one with highest activity. That's for having a good representation of this signature in our dataset.

```
# For reproducibility
set.seed(3)

# List of contexts
contexts <- colnames(SBSsigs)
contexts <- factor(contexts, levels=colnames(SBSsigs))

# List with probability of each trinucleotide context in each signature
sbs5 <- as.vector(t(SBSsigs[5,]))
sbs16 <- as.vector(t(SBSsigs[22,]))
sbs92 <- as.vector(t(SBSsigs[76,]))
prob <- list(sbs5,sbs16,sbs92)
names(prob)<-c("sbs5","sbs16","sbs92")

# Input matrix is declared (will be filled in the for loop)
input_matrix <- c()
```

```r
# A for loop where each loop a new sample is created
lSimulations <- list()
lSimulations <- lapply(1:Nsim, function(thisSim){
  for(i in 1:500) {

    ## Number of signatures acting in the sample
    nsig <- sample(c(1:3), 1)

    if (nsig == 1){
      # signature
      sig <- sample(c("sbs5","sbs16","sbs92"), nsig)
      # number of mutations
      nmut <- sample(c(seq(100,150,1)),1)
      # context of mutations
      smut <- sample(contexts, nmut, replace = TRUE, prob = prob[[sig]])
      # Result is added to input matrix
      samp <- table(smut)
      input_matrix <- rbind(input_matrix, samp)
    }
    else if (nsig == 2){
      # signatures
      sig <- sample(c("sbs5","sbs16","sbs92"), nsig, prob=c(0.8,0.1,0.1))
      # number of mutations
      nmut1 <- sample(c(seq(100,150,1)),1)
      nmut2 <- sample(c(seq(25,50,1)),1)
      # context of mutations
      sig1 <- sample(sig,1)
      smut1 <- sample(contexts, nmut1, replace = TRUE, prob = prob[[sig1]])
      smut1 <- table(smut1)
      sig2 <- sig[sig!=sig1]
      smut2 <- sample(contexts, nmut2, replace = TRUE, prob = prob[[sig2]])
      smut2 <- table(smut2)
      # Result is added to input matrix
      samp<-c()
      for (i in 1:96){
        s <- sum(smut1[i],smut2[i])
        samp <- c(samp,s)
      }
      input_matrix <- rbind(input_matrix, samp)
    }
    else if (nsig == 3) {
      # signature with highest activity
      sig <- sample(c("sbs5","sbs16","sbs92"), 1, prob=c(0.8,0.1,0.1))
      # number of mutations
      nmut <- sample(c(seq(100,150,1)),1)
      # context of mutations
      smut <- sample(contexts, nmut, replace = TRUE, prob = prob[[sig]])
      smut <- table(smut)
      # other signatures
      sigs <- names(prob)[names(prob) != sig]
      # number of mutations
      nmuts <- sample(c(seq(25,50,1)),1)
      # context of mutations
```

```
      smut1 <- sample(contexts, nmuts, replace = TRUE, prob = prob[[sigs[1]]])
      smut2 <- sample(contexts, nmuts, replace = TRUE, prob = prob[[sigs[2]]])
      smut1 <- table(smut1)
      smut2 <- table(smut2)
      ## Result is added to input matrix
      samp<-c()
      for (i in 1:96){
        s <- sum(smut[i],smut1[i],smut2[i])
        samp <- c(samp,s)
      }
      input_matrix <- rbind(input_matrix, samp)
    }
  }
  # Sample names are given
  rownames(input_matrix) <- paste0("samp", seq_len(nrow(input_matrix)))
  colnames(input_matrix) <- contexts
  lSimulations[[thisSim]] <- input_matrix
})
saveRDS(lSimulations, file = paste0(path_to_data, "/1_InputMatrices_", Nsim, "xSimulations_500samples_S
```

```
lSimulations <- readRDS(paste0(path_to_data,"/1_InputMatrices_", Nsim, "xSimulations_500samples_SBS5.rds

# Plots first ten samples of the 1st simulated input matrix
input_matrix <- lSimulations[[1]]
pdat <- reshape2::melt(input_matrix[1:10,])
colnames(pdat) <- c("sample", "context", "value")
pdat$feature <- c(rep("C>A",160), rep("C>G",160) ,rep("C>T",160), rep("T>A",160),
                  rep("T>C",160), rep("T>G",160))

ggplot(pdat, aes(x = factor(context), y = value, fill = feature)) +
  geom_bar(stat = "identity") + facet_wrap(. ~ sample, ncol = 2) + theme_bw() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
  scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                        "C>T" = "#56B4E9", "T>A" = "#009E73",
                                        "T>C" = "#F0E442", "T>G" = "#0072B2"))
```



2.2 Estimate signatures from samples

2.2.1 Brunet NMF

Next we run NMF with k=3 to see if we can reconstruct the signatures:

```
#lSimulations <- readRDS(paste0(path_to_data,"/1_InputMatrices_", Nsim, "xSimulations_500samples_SBS5.r
lNMF <- list()
input_matrix<-c()
lNMF <- lapply(1:Nsim, function(thisSim){
  print(thisSim)
  input_matrix <- lSimulations[[thisSim]]

  # I have added 0.001 in all samples in this row
  col<-which(colSums(input_matrix) == 0)
  input_matrix[,col]<-0.00001

  # Run the NMF.
  sigs <- NMF::nmf(t(input_matrix), rank = 3, seed = 222222, nrun = 100, method = "brunet")

  # Format for plotting counts of each context per signature
  sigmat <- basis(sigs)

  # Normalize signature matrix
  sigmat<-apply(sigmat,2,function(x){x/sum(x)})
```

```
  lNMF[[thisSim]] <- sigmat
})
saveRDS(lNMF, file = paste0(path_to_data, "/2_NMF_",Nsim,"xSimulations_500samples_SBS5.rds"))
```

Now, we match definitions of the three signatures obtained from the different NMF runs and calculated the mean.

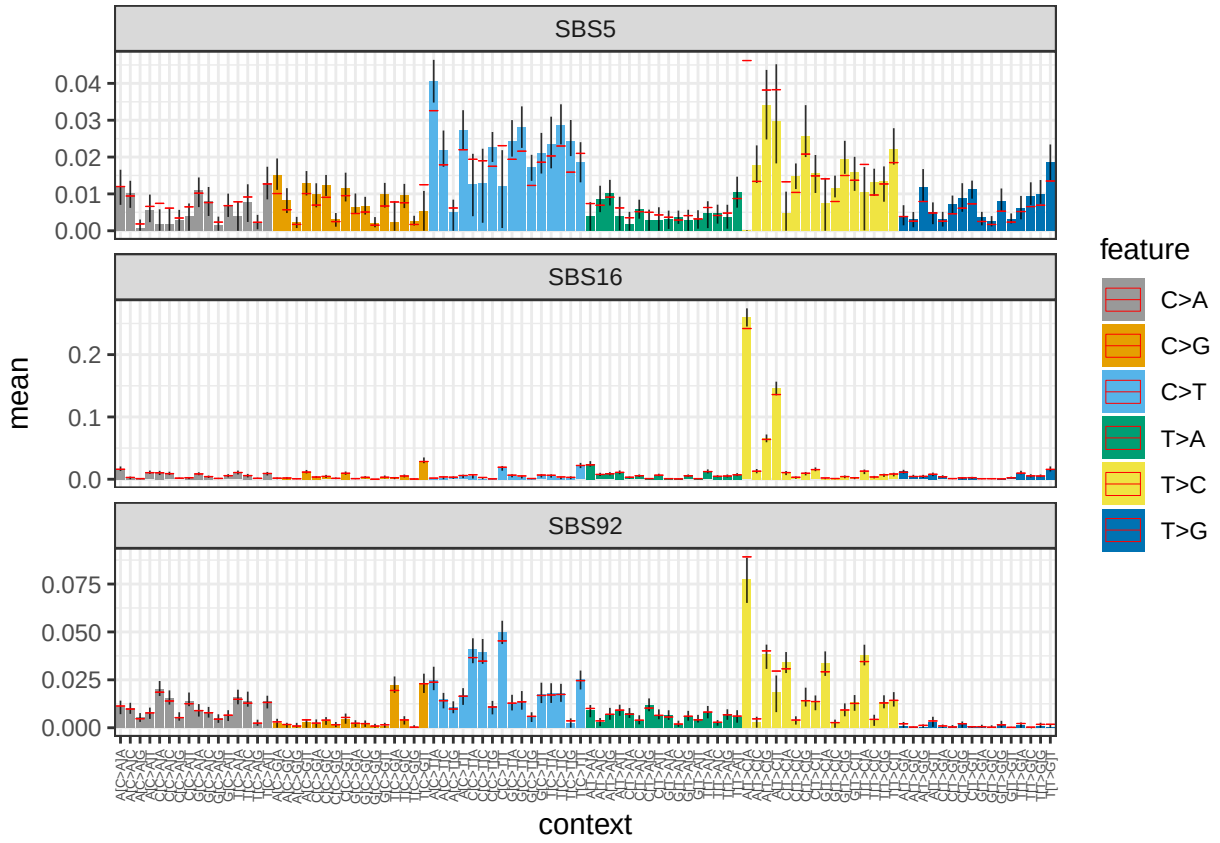
```
lNMF <- readRDS(paste0(path_to_data,"/2_NMF_", Nsim,"xSimulations_500samples_SBS5.rds"))

# Definition from the original matrix
Sigs <- c("SBS5", "SBS16", "SBS92")
sigs <- t(SBSsigs[Sigs,])

# Match signatures and check that only matches with one
lmatches <- lapply(1:Nsim, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
    d <- length(unique(match))
    match <- c(match, d)
    return(match)
})

# Get definitions of matched signatures
lSimDefinitions <- lapply(1:Nsim, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
    colnames(sigmat) <- match
    sigmat <- sigmat[,Sigs]
    return(sigmat)
})

lSimDefinitions <- lapply(Sigs, function(thisSig){
  sdefs <- lapply(1:Nsim, function(thisSim){
    s <- lSimDefinitions[[thisSim]][,thisSig]})
  sdefs <- as.data.frame(do.call(cbind, sdefs))
  return(sdefs)
})
names(lSimDefinitions)<-Sigs

# Calculate the standard error of simulated definitions
lVariabilityDefinitions <- lapply(Sigs, function(thisSig) {
```

```
    defs <- lSimDefinitions[[thisSig]]

    # Statistics
    min <- apply(defs, 1, min)
    max <- apply(defs, 1, max)
    median <- apply(defs, 1, median)
    mean <- rowMeans(defs)
    sd <- apply(defs, 1, sd)
    se <- sd/sqrt(ncol(defs))
    nVar <- as.data.frame(cbind(median=median, min=min, max=max, mean=mean, sd=sd, se=se))
    return(nVar)
})
names(lVariabilityDefinitions)<-Sigs
```

Finally, we plot the definitions of SBS signatures

```
## Prepare data for plotting and produce the figure
# Get mean values for bars and melt
mean <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "mean")))
colnames(mean) <- colnames(SBSsigs)
mean <- reshape2::melt(t(mean))
colnames(mean) <- c("context", "sig", "mean")

# Get min for error bars and melt
min <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "min")))
colnames(min) <- colnames(SBSsigs)
min <- reshape2::melt(t(min))
colnames(min) <- c("context", "sig", "min")

# Get max for error bars and melt
max <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "max")))
colnames(max) <- colnames(SBSsigs)
max <- reshape2::melt(t(max))
colnames(max) <- c("context", "sig", "max")

# Join standard errors
pdat <- left_join(mean, min, by=c("context", "sig"))
pdat <- left_join(pdat, max, by=c("context", "sig"))

# Add original signatures
sbs <- reshape2::melt(sigs)
colnames(sbs) <- c("context", "sig", "weights")
pdat <- left_join(pdat, sbs, by=c("context", "sig"))

# Prepare for plotting
pdat$feature <- c(rep(c(rep("C>A",16), rep("C>G",16), rep("C>T",16), rep("T>A",16),
                        rep("T>C",16), rep("T>G",16)),3))
pdat$feature <- factor(pdat$feature, levels=c("C>A","C>G","C>T","T>A","T>C","T>G"))

yaxis.max <- round(max(pdat$max)+0.05,digits = 1)
yaxis.min <- round(min(pdat$min)-0.05,digits = 1)

p <- ggplot(pdat, aes(x=context, y=mean, fill=feature)) +
```

```
    geom_bar(stat = "identity") + facet_wrap(. ~ sig, ncol = 1, scales="free_y") + theme_bw() +
    geom_linerange(aes(ymin=min, ymax=max), colour="black", alpha=0.8, size=0.3) +
    geom_crossbar(aes(y=weights, ymin=weights, ymax=weights), colour="red", alpha=0.6, size=0.1) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
    #scale_y_continuous(limits = c(yaxis.min, yaxis.max)) +
    scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                          "C>T" = "#56B4E9", "T>A" = "#009E73",
                                          "T>C" = "#F0E442", "T>G" = "#0072B2"))
print(p)
```




```
## Save output
# cairo_pdf(file.path(OUTPUT_DIR, "SuppFig43_1_SBSSimulations_250xInputMatrices_SBS5.pdf"), height = 14
# print(p); dev.off()
```

This is a clear example where NMF shrinks to zero some components of a signature because they are better represented by another signature. We can observe that, in the SBS5, the weight of A[T>C]A significantly decrease from 0.046 to 8.83e-06 [1.76e-12, 1.86e-04].

2.2.2 SigProfilerExtractor

We first prepare data for SigProfilerExtractor, and run it. We performed this analysis to confirm what we observed with brunet NMF. Therefore, only 10 simulation input matrices has been used for this analysis.

```
#lSimulations <- readRDS(paste0(path_to_data,"/1_InputMatrices",Nsim,"xSimulations_500samples_SBS5.rds"
for (i in 1:10){
    M <- t(lSimulations[[i]])
    M <- cbind(Mutation_type=rownames(M), M)
    write.table(M, paste0(path_to_data,"/SBS5_simulations/", i,"_SBS5_simulations.txt"), sep="\t", row.
}
```

After running SigProfilerExtractor, we match definitions of the three signatures obtained from the 10 runs and calculated the mean and standard error.

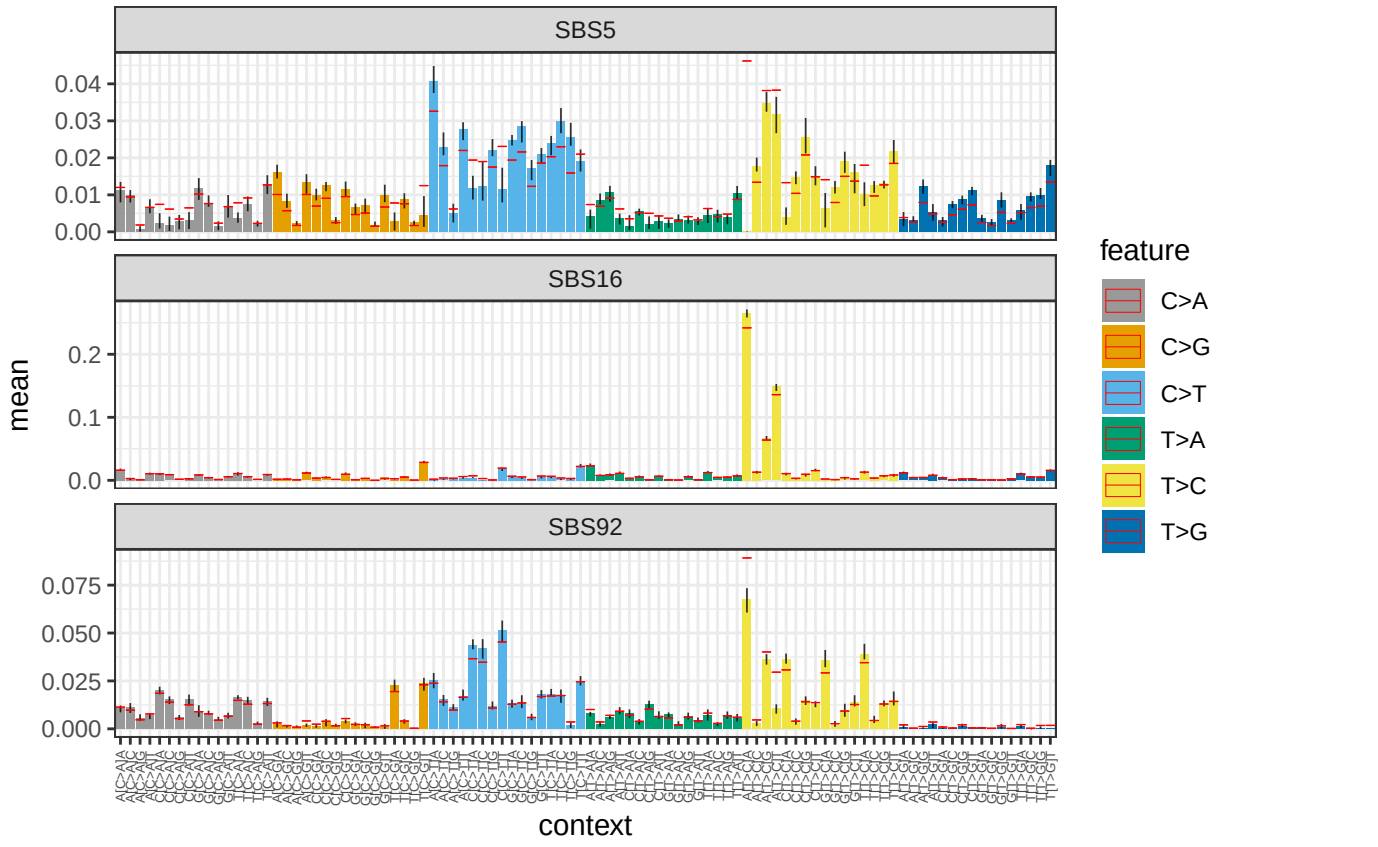
```
# Get paths of signature definition matrices
path <- paste0(path_to_sigprof, "/SBS5")
allSigFiles <- unlist(lapply(path, function(thisPath){
    file <- list.files(thisPath, pattern="*.txt",full.names = TRUE)}))
numM <- length(allSigFiles)

# Load signature definition matrices
lNMF <- lapply(allSigFiles, function(file){
    defs <- read.table(file, sep="\t", header = T, check.names=FALSE)
    row.names(defs) <- defs[,1]
    defs <- defs[,-1]
})
names(lNMF)<-1:numM

# Definition from the original matrix
Sigs <- c("SBS5", "SBS16", "SBS92")
sigs <- t(SBSsigs[Sigs,])

# Match signatures and check that only matches with one
lmatches <- lapply(1:10, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
    d <- length(unique(match))
    match <- c(match, d)
    return(match)
})

# Get definitions of matched signatures
lSimDefinitions <- lapply(1:10, function(thisSim){
    sigmat <- lNMF[[thisSim]]

    # Found noisy signatures that match
    match <- unlist(lapply(1:ncol(sigmat), function(i){
      cos <- cosine(sigmat[,i], sigs)
      max <- max(cos)
      sig <- names(which(cos == max))
    }))
```

```
    colnames(sigmat) <- match
    sigmat <- sigmat[,Sigs]
    return(sigmat)
})

lSimDefinitions <- lapply(Sigs, function(thisSig){
  sdefs <- lapply(1:10, function(thisSim){
    s <- lSimDefinitions[[thisSim]][,thisSig]})
  sdefs <- as.data.frame(do.call(cbind, sdefs))
  return(sdefs)
})
names(lSimDefinitions)<-Sigs

# Calculate the standard error of simulated definitions
lVariabilityDefinitions <- lapply(Sigs, function(thisSig) {
    defs <- lSimDefinitions[[thisSig]]

    # Statistics
    min <- apply(defs, 1, min)
    max <- apply(defs, 1, max)
    median <- apply(defs, 1, median)
    mean <- rowMeans(defs)
    sd <- apply(defs, 1, sd)
    se <- sd/sqrt(ncol(defs))
    nVar <- as.data.frame(cbind(median=median, min=min, max=max, mean=mean, sd=sd, se=se))
    return(nVar)
})
names(lVariabilityDefinitions)<-Sigs
```

Finally, we plot the definitions of SBS signatures. Error bars show the range of values obtained for each component. Horizontal red lines indicate the original weight of each component.

```
## Prepare data for plotting and produce the figure
# Get mean values for bars and melt
mean <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "mean")))
colnames(mean) <- colnames(SBSsigs)
mean <- reshape2::melt(t(mean))
colnames(mean) <- c("context", "sig", "mean")

# Get min for error bars and melt
min <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "min")))
colnames(min) <- colnames(SBSsigs)
min <- reshape2::melt(t(min))
colnames(min) <- c("context", "sig", "min")

# Get max for error bars and melt
max <- as.data.frame(t(sapply(lVariabilityDefinitions, `[[`, "max")))
colnames(max) <- colnames(SBSsigs)
max <- reshape2::melt(t(max))
colnames(max) <- c("context", "sig", "max")

# Join standard errors
pdat <- left_join(mean, min, by=c("context", "sig"))
```

```
pdat <- left_join(pdat, max, by=c("context", "sig"))

# Add original signatures
sbs <- reshape2::melt(sigs)
colnames(sbs) <- c("context", "sig", "weights")
pdat <- left_join(pdat, sbs, by=c("context", "sig"))

# Prepare for plotting
pdat$feature <- c(rep(c(rep("C>A",16), rep("C>G",16), rep("C>T",16), rep("T>A",16),
                        rep("T>C",16), rep("T>G",16)),3))
pdat$feature <- factor(pdat$feature, levels=c("C>A","C>G","C>T","T>A","T>C","T>G"))

yaxis.max <- round(max(pdat$max)+0.05,digits = 1)
yaxis.min <- round(min(pdat$min)-0.05,digits = 1)

p <- ggplot(pdat, aes(x=context, y=mean, fill=feature)) +
    geom_bar(stat = "identity") + facet_wrap(. ~ sig, ncol = 1, scales="free_y") + theme_bw() +
    geom_linerange(aes(ymin=min, ymax=max), colour="black", alpha=0.8, size=0.3) +
    geom_crossbar(aes(y=weights, ymin=weights, ymax=weights), colour="red", alpha=0.6, size=0.1) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1, size = 5)) +
    #scale_y_continuous(limits = c(yaxis.min, yaxis.max)) +
    scale_fill_manual("feature", values=c("C>A" = "#999999", "C>G" = "#E69F00",
                                          "C>T" = "#56B4E9", "T>A" = "#009E73",
                                          "T>C" = "#F0E442", "T>G" = "#0072B2"))
print(p)
```



```
## Save output
# cairo_pdf(file.path(OUTPUT_DIR, "SuppFig51_1_SigProfilerExtractor_SBSSimulations_10xInputMatrices_SBS
# print(p); dev.off()
```