Characterization and correction of ATAC-seq Tn5 sequence bias vignette

Jacob B. Wolpe* Michael J. Guertin†

Abstract

Applying a correction for Tn5 insertion sequence bias to ATAC-seq data.

Contents

*University of Virginia

†University of Connecticut

1 Foreword

Organization of this vignette assumes that each section contains all files in a folder specific to that section and that scripts are initiated within the original section folder. ie. Figure1, Figure2 are individual folders. Therefore, calls to data made in a previous section will usually involve accessing that folder.

2 Figure 1. Tn5 sequence bias is uniquely wide amongst nucleases.

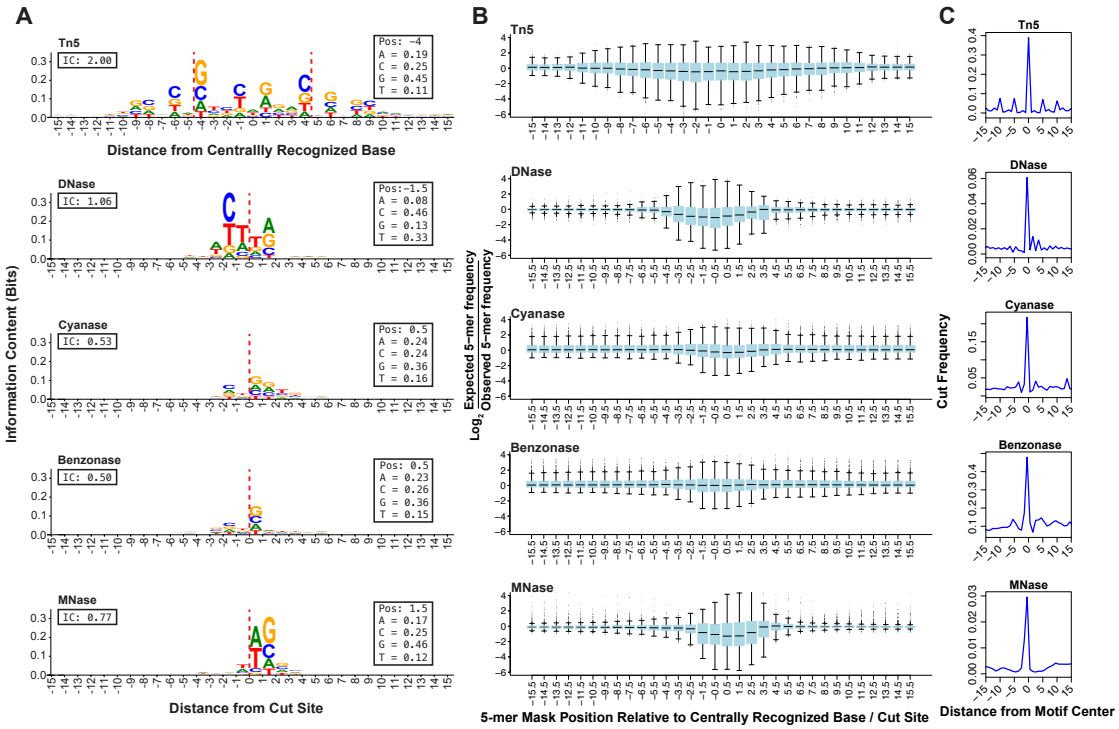


Figure 1: Tn5 sequence bias is uniquely wide amongst nucleases. A) Background corrected sequence bias motifs for Tn5 and other commonly used enzymes for assaying chromatin accessibility. Nucleotide frequencies listed in the inset to the right correspond to the highest information content position. Nuclease cleavage sites are indicated by dashed red line. Information content (IC) from positions -10 to 10 is listed in the inset to the upper left. B) Box and whisker plots for the log2 inverse of all possible 5-mer scaling factors as a function of mask position relative to the cutsite for chromatin accessibility assay enzymes. Scaling factors are the percent of each k-mer in the genome divided by that k-mer's observed percent in the data, for a given position relative to the central base. The inverse of this value is the enzyme's bias for this k-mer. C) Composite profiles of sites which best adhere to a given enzyme's bias motif, as seen in A, from data sets for each respective enzyme, separated by strand and using only reads aligned to that strand (strand specific).

2.1 Downloading genomic FASTA files

This step will download all of the necessary genomic reference files for figure 1.

```
#You will need to download the human and mouse genomes:
wget http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz
wget https://hgdownload.cse.ucsc.edu/goldenpath/hg38/chromosomes/chrM.fa.gz
wget http://hgdownload.cse.ucsc.edu/goldenPath/mm39/bigZips/mm39.fa.gz
#unzip
gunzip hg38.fa.gz
gunzip chrM.fa.gz
gunzip mm39.fa.gz
```

2.2 Python script: bedToOneEntryBed.py

The file is at https://raw.githubusercontent.com/guertinlab/Tn5bias/master/bedToOneEntryBed.py and it converts a bed to a bed with one entry per instance, so that a FASTA entry is present for each ATAC insertion event.

```
#! /usr/bin/env
import re
import string
import sys
import getopt
import os
import itertools
import glob



def function(fqFileName):
    infile=open(fqFileName, 'r')
    outfile=open(str.split(fqFileName,'.bed')[0]+'.oneentry.bed', 'w')
    while 1:
        line=infile.readline()
        if not line: break
        for i in range(int(line.split()[4])):
            outfile.write('%s'%(line))
    infile.close()
    outfile.close()



def main(argv):
    try:
        opts, args = getopt.getopt(argv, "i:h", ["infile=","help"])
    except getopt.GetoptError as err:
        print(str(err))
        sys.exit(2)
    infile = False
    for opt, arg in opts:
        if opt in ('-i', '--infile'):
            infile = arg
        elif opt in ('-h', '--help'):
            print('\n./bedToOneEntryBed.py -i test_PE1_plus_not_scaled.bed')
            sys.exit()
    if infile:
        print(infile)
        function(infile)

if __name__ == "__main__":
    main(sys.argv[1:])
```

2.3 Retrieving then processing FASTQ data files

This step will download all of the data used in figure 1. These data sets will then be aligned to their respective genomes, plus and minus data will be separated, seqOutBias will be run on both separated and unseparated data, bed entries will be converted into an entry for each read, minus strand sequences are reverse complemented, and finally concatenated with the plus strand. This will be done for DNase and Tn5 using human (hg38 reference genome) data while Cyanase, Benzonase and MNase use mouse (mm39 reference genome) data.

Software installations:

sra-tools: https://github.com/ncbi/sra-tools

fastp: https://github.com/OpenGene/fastp/blob/master/README.md

bowtie2: http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

samtools: http://www.htslib.org

seqOutBias: https://github.com/guertinlab/seqOutBias

```
#Make a directory for mm39 data
mkdir mm39_data
cd mm39_data

#Download Benzonase, Cyanase, MNase mm39 (mouse) data
fasterq-dump SRR535737
fasterq-dump SRR535738
fasterq-dump SRR535739
fasterq-dump SRR535740
fasterq-dump SRR535741
fasterq-dump SRR535742
fasterq-dump SRR535743
fasterq-dump SRR535744
fasterq-dump SRR5723785

#Rename Benzonase, Cyanase, MNase data from SRR number
mv SRR535737.fastq mm39_liver_Benzonase0.25U.fastq
mv SRR535738.fastq mm39_liver_Benzonase1U_1.fastq
mv SRR535739.fastq mm39_liver_Benzonase1U_2.fastq
mv SRR535740.fastq mm39_liver_Benzonase4U.fastq
mv SRR535741.fastq mm39_liver_Cyanase0.25U.fastq
mv SRR535742.fastq mm39_liver_Cyanase1U_1.fastq
mv SRR535743.fastq mm39_liver_Cyanase1U_2.fastq
mv SRR535744.fastq mm39_liver_Cyanase4U.fastq
mv SRR5723785_1.fastq mm39_liver_MNase_1.fastq
mv SRR5723785_2.fastq mm39_liver_MNase_2.fastq

#Concatenate each enzyme
cat *Benz* > mm39_liver_Benzonase.fastq
cat *Cyan* > mm39_liver_Cyanase.fastq
cat *MNase* > mm39_liver_MNase.fastq

#Zip each file
gzip *ase.fastq

#build index genome for mm39
bowtie2-build mm39.fa mm39

#Align each dataset to mm39 genome
for fq in mm39_liver_*.fastq.gz
do
    name=$(echo $fq | awk -F".fastq.gz" '{print $1}')
    echo $name
    bowtie2 -p 6 --maxins 500 -x mm39 -U ${name}.fastq.gz  \
       | samtools view -bS - | samtools sort - -o $name.bam
done

#First, split each dataset into plus and minus aligned reads.
#Then, run seqOutBias on plus/minus and unseparated strands to get
##the bedfile which has chromosome coordinates for each read.
#From the bedfile, make each read a unique entry using bedToOneEntryBed.py
#Using awk commands, shift each position back 10bp then forward 21 in order to
##make a 'window' around the cut site.
#In order to get each sequence, we then convert from fasta to bed format for each data set
for i in mm39*ase.bam
do
    name=$(echo $i | awk -F".bam" '{print $1}')
    echo $name
    samtools view -bh -F 20 ${name}.bam > ${name}_plus.bam
    samtools view -bh -f 0x10 ${name}.bam > ${name}_minus.bam
    seqOutBias mm39.fa ${name}.bam --no-scale --shift-counts \
```

```
                                --strand-specific --bed=${name}.bed \
                                 --bw=${name}.bigWig --read-size=76
    seqOutBias mm39.fa ${name}_plus.bam --no-scale --shift-counts \
                                --strand-specific --bed=${name}_plus.bed \
                                 --bw=${name}_plus.bigWig --read-size=76
    seqOutBias mm39.fa ${name}_minus.bam --no-scale --shift-counts \
                                --strand-specific --bed=${name}_minus.bed \
                                 --bw=${name}_minus.bigWig --read-size=76
#Make each read its own entry in a bed file
#e.g. 2 of the same read becomes 2 entries of the same read
    python bedToOneEntryBed.py -i ${name}.bed
    python bedToOneEntryBed.py -i ${name}_plus.bed
    python bedToOneEntryBed.py -i ${name}_minus.bed
#These bed files can be used to get the sequence flanking ALL Tn5 insertion sites,
#so we can see the precise nature of the sequence bias for each PE/strand combination
#we shift the window coordinatess for each bed file to accomodate the 31bp for the figure
    awk '{$2 = $2 - 10; print}' ${name}.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 21; print}' | grep -v - | \
       fastaFromBed -fi mm39.fa -s -bed stdin -fo ${name}.fasta
    awk '{$2 = $2 - 10; print}' ${name}_plus.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 21; print}' | grep -v - | \
       fastaFromBed -fi mm39.fa -s -bed stdin -fo ${name}_plus.fasta
    awk '{$2 = $2 - 11; print}' ${name}_minus.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 21; print}' |  grep -v - | \
       fastaFromBed -fi mm39.fa -s -bed stdin -fo ${name}_minus.fasta
#First convert all sequences in the minus fasta into uppercase letters,
#then reverse complement all minus strand sequences.
    awk 'BEGIN{FS=" "}{if(!/>/){print toupper($0)}else{print $1}}' ${name}_minus.fasta | \
       fastx_reverse_complement -o ${name}_minus_RC.fasta
#Concatenate the plus and minus strand fasta files together
    cat ${name}_minus_RC.fasta ${name}_plus.fasta > ${name}_sepcat.fasta
done



#Make a directory for hg38 data
mkdir hg38_data
cd hg38_data

#Download Tn5 dataset
fasterq-dump SRR5123141

#Zip Tn5 data
gzip *fastq

#Rename Tn5 data
mv SRR5123141_1.fastq.gz C1_gDNA_rep1_PE1.fastq.gz
mv SRR5123141_2.fastq.gz C1_gDNA_rep1_PE2.fastq.gz


#build index genome
bowtie2-build hg38.fa hg38
bowtie2-build chrM.fa chrM

#First align Tn5 data to mitochondrial chromosome, then sort by chr/start
#Convert bam file to fastq file, then align to hg38 genome
for fq in *PE1.fastq.gz
do
    name=$(echo $fq | awk -F"_PE1.fastq.gz" '{print $1}')
    echo $name
    bowtie2 -p 6 -x chrM -1 ${name}_PE1.fastq.gz -2 ${name}_PE2.fastq.gz \
        | samtools view -b - | samtools sort - -o $name.sorted.chrM.bam
    samtools index $name.sorted.chrM.bam
    samtools view -b $name.sorted.chrM.bam '*' | samtools sort -n - \
```

```
        | bamToFastq -i - -fq ${name}_PE1.chrM.fastq -fq2 ${name}_PE2.chrM.fastq
    gzip *.chrM.fastq
    rm $name.sorted.chrM.bam
    bowtie2 -p 6 --maxins 500 -x hg38 -1 ${name}_PE1.chrM.fastq.gz -2 ${name}_PE2.chrM.fastq.gz \
        | samtools view -bS - | samtools sort -n - | samtools fixmate -m - - \
        | samtools sort - | samtools markdup -r - ${name}.bam

done


#First, split data into plus and minus aligned reads
#Run seqOutBias with no scale and shift of 4,-4 to get each read centered
#on cut site. Then process same as other enzymes above.
for i in C1*.bam
do
    name=$(echo $i | awk -F".bam" '{print $1}')
    echo $name


        samtools view -bh -F 20 ${name}.bam > ${name}_plus.bam
    samtools view -bh -f 0x10 ${name}.bam > ${name}_minus.bam
    seqOutBias hg38.fa ${name}.bam --no-scale --custom-shift=4,-4 \
                               --strand-specific --bed=${name}.bed \
                                --bw=${name}.bigWig --read-size=76
    seqOutBias hg38.fa ${name}_plus.bam --no-scale --custom-shift=4,-4 \
                               --strand-specific --bed=${name}_plus.bed \
                                --bw=${name}_plus.bigWig --read-size=76
    seqOutBias hg38.fa ${name}_minus.bam --no-scale --custom-shift=4,-4 \
                               --strand-specific --bed=${name}_minus.bed \
                                --bw=${name}_minus.bigWig --read-size=76
    python bedToOneEntryBed.py -i ${name}.bed
        python bedToOneEntryBed.py -i ${name}_plus.bed
    python bedToOneEntryBed.py -i ${name}_minus.bed
#we shift the window coordinatess for each bed file to accomodate the 31bp for the figure
    awk '{$2 = $2 - 15; print}' ${name}.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 31; print}' | grep -v - | \
       fastaFromBed -fi hg38.fa -s -bed stdin -fo ${name}.fasta
        awk '{$2 = $2 - 15; print}' ${name}_plus.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 31; print}' | grep -v - | \
       fastaFromBed -fi hg38.fa -s -bed stdin -fo ${name}_plus.fasta
    awk '{$2 = $2 - 15; print}' ${name}_minus.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 31; print}' |  grep -v - | \
       fastaFromBed -fi hg38.fa -s -bed stdin -fo ${name}_minus.fasta

        awk 'BEGIN{FS=" "}{if(!/>/){print toupper($0)}else{print $1}}' ${name}_minus.fasta | \
         fastx_reverse_complement -o ${name}_minus_RC.fasta
#Concatenate the plus and minus strand fasta files together
      cat ${name}_RC.fasta ${name}_plus.fasta > ${name}_sepcat.fasta

done



#Download naked DNase data
fasterq-dump SRR769954


#Rename DNase data from SRR number
mv SRR769954.fastq DNase_Naked.fastq

#Zip
gzip DNase_Naked.fastq
```

```
#Align each dataset to hg38 genome
for fq in DNase_*.fastq.gz
do
    name=$(echo $fq | awk -F".fastq.gz" '{print $1}')
    echo $name
    bowtie2 -p 6 --maxins 500 -x hg38 -U ${name}.fastq.gz  \
       | samtools view -bS - | samtools sort - -o $name.bam
done

#First, split each dataset into plus and minus aligned reads.
#Then, run seqOutBias on plus/minus and unseparated strands to get
##the bedfile which has chromosome coordinates for each read.
#From the bedfile, make each read a unique entry using bedToOneEntryBed.py
#Using awk commands, shift each position back 10bp then forward 21 in order to
##make a 'window' around the cut site.
#In order to get each sequence, we then convert from fasta to bed format for each data set
for i in DNase_*.bam
do
    name=$(echo $i | awk -F".bam" '{print $1}')
    echo $name
    samtools view -bh -F 20 ${name}.bam > ${name}_plus.bam
    samtools view -bh -f 0x10 ${name}.bam > ${name}_minus.bam
    seqOutBias hg38.fa ${name}.bam --no-scale --shift-counts \
                               --strand-specific --bed=${name}_unscaled.bed \
                                --bw=${name}_unscaled.bigWig --read-size=76
    seqOutBias hg38.fa ${name}_plus.bam --no-scale --shift-counts \
                               --strand-specific --bed=${name}_plus_unscaled.bed \
                                --bw=${name}_plus_unscaled.bigWig --read-size=76
    seqOutBias hg38.fa ${name}_minus.bam --no-scale --shift-counts \
                               --strand-specific --bed=${name}_minus_unscaled.bed \
                                --bw=${name}_minus_unscaled.bigWig --read-size=76
#Make each read its own entry in a bed file
#e.g. 2 of the same read becomes 2 entries of the same read
    python bedToOneEntryBed.py -i ${name}.bed
    python bedToOneEntryBed.py -i ${name}_plus.bed
    python bedToOneEntryBed.py -i ${name}_minus.bed
#these bed files can be used to get the sequence flanking ALL DNase insertion sites,
#so we can see the precise nature of the sequence bias for each PE/strand combination
#we shift the window coordinatess for each bed file to accomodate the 31bp for the figure
    awk '{$2 = $2 - 15; print}' ${name}.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 31; print}' | grep -v - | \
       fastaFromBed -fi hg38.fa -s -bed stdin -fo ${name}.fasta
    awk '{$2 = $2 - 15; print}' ${name}_plus.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 31; print}' | grep -v - | \
       fastaFromBed -fi hg38.fa -s -bed stdin -fo ${name}_plus.fasta
    awk '{$2 = $2 - 16; print}' ${name}_minus.oneentry.bed | \
       awk '{OFS="\t";} {$3 = $2 + 31; print}' |  grep -v - | \
       fastaFromBed -fi hg38.fa -s -bed stdin -fo ${name}_minus.fasta
#First convert all sequences in the minus fasta into uppercase letters,
#then reverse complement all minus strand sequences.
    awk 'BEGIN{FS=" "}{if(!/>/){print toupper($0)}else{print $1}}' ${name}_minus.fasta | \
       fastx_reverse_complement -o ${name}_minus_RC.fasta
#Concatenate the plus and minus strand fasta files together
    cat ${name}_minus_RC.fasta ${name}_plus.fasta > ${name}_sepcat.fasta
done
```

2.4 Counting nucleotides for sequence logos of each enzyme

Here, each enzyme's data set (FASTA file) is used to count the occurrence of each nucleotide at each position within 15bp around the cutsite.

```
library(ggseqlogo)

#Count all instances of each nucleotide at each position, data frame to be counted is
#x.ligation, and posnum is the size of the window counted
transfac.func.2 <- function(x.ligation, posnum) {
   col.matrix = matrix()
   for (g in 1:posnum){
    itnum = lapply(strsplit(as.character(x.ligation), ''), "[", g)
    if (g == 1) {
    col.matrix = itnum
    } else {
    col.matrix = cbind(col.matrix, itnum)
    }
    }

  a.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "A"))
  t.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "T"))
  c.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "C"))
  g.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "G"))

  transfac = cbind(a.nuc, c.nuc, g.nuc, t.nuc)
  print(transfac)
  return(transfac)
}

#Split up data sets into groups of desired size to reduce RAM load
seqlast <- function (from, to, by)
{
  vec <- do.call(what = seq, args = list(from, to, by))
  if ( tail(vec, 1) != to ) {
    return(c(vec, to))
  } else {
    return(vec)
  }
}

#Make all instaces in a table uppercase
uppercase <- function(tableinput){
                      upperdf <- read.table(tableinput, comment.char = '>')
                      upperdf[,1] = as.character(upperdf[,1])
                      upperdf = data.frame(lapply(upperdf, function(v) {
                        if (is.character(v)) return(toupper(v))
                        else return(v)}))
                        }

#############################################################################################
system('mkdir output')
setwd('output')
#############################################################################################
#Benzonase plus, minus, unseparated, sepcat FASTA files:
uppercasenames <- list('../mm39_data/mm39_liver_Benzonase_plus.fasta',
                       '../mm39_data/mm39_liver_Benzonase_RC.fasta',
                       '../mm39_data/mm39_liver_Benzonase.fasta',
                       '../mm39_data/mm39_liver_Benzonase_sepcat.fasta')

#Make sure all FASTA entries are uppercase
uplist <- lapply(uppercasenames, uppercase)

#Each list object of uplist is the corresponding file:
```

```
Benzonase_plus = as.data.frame(uplist[1])
Benzonase_minus = as.data.frame(uplist[2])
Benzonase = as.data.frame(uplist[3])
Benzonase_sepcat = as.data.frame(uplist[4])

rm(uplist, uppercasenames)
#
#Take each data frame and count each nucleotide at each position:
##Benzonase_plus
Benzonase_plus.transfac = vector("list", length = ceiling(nrow(Benzonase_plus)/10000000))
Benzonase_plus.split = seqlast(from=1,to=(nrow(Benzonase_plus)+1),by=10000000)

for (i in 1:length(Benzonase_plus.transfac)) {
Benzonase_plus.transfac[[i]] =
  transfac.func.2(Benzonase_plus[Benzonase_plus.split[i]:(Benzonase_plus.split[i+1]-1),1],31)
}
#Combine all splits of data
transfac.Benzonase_plus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Benzonase_plus.transfac)) {
  transfac.Benzonase_plus = transfac.Benzonase_plus + Benzonase_plus.transfac[[i]]
}
#Convert matrix back into DF
transfac.Benzonase_plus = cbind(1:nrow(transfac.Benzonase_plus), transfac.Benzonase_plus)
colnames(transfac.Benzonase_plus) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Benzonase_plus_bias",
             "BF Benzonase_plus_bias"), 'Benzonase_plus_bias.transfac')
write.table(transfac.Benzonase_plus, file = "Benzonase_plus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
##Benzonase_minus
Benzonase_minus.transfac = vector("list", length = ceiling(nrow(Benzonase_minus)/10000000))
Benzonase_minus.split = seqlast(from=1,to=(nrow(Benzonase_minus)+1),by=10000000)

for (i in 1:length(Benzonase_minus.transfac)) {
Benzonase_minus.transfac[[i]]=
  transfac.func.2(Benzonase_minus[Benzonase_minus.split[i]:(Benzonase_minus.split[i+1]-1),1], 31)
}
#Combine all splits of data
transfac.Benzonase_minus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Benzonase_minus.transfac)) {
  transfac.Benzonase_minus = transfac.Benzonase_minus + Benzonase_minus.transfac[[i]]
}
#Convert matrix back into DF
transfac.Benzonase_minus = cbind(1:nrow(transfac.Benzonase_minus), transfac.Benzonase_minus)
colnames(transfac.Benzonase_minus) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Benzonase_minus_bias",
             "BF Benzonase_minus_bias"), 'Benzonase_minus_bias.transfac')
write.table(transfac.Benzonase_minus, file = "Benzonase_minus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
##Benzonase_nosep
Benzonase.transfac = vector("list", length = ceiling(nrow(Benzonase)/10000000))
Benzonase.split = seqlast(from=1,to=(nrow(Benzonase)+1),by=10000000)

for (i in 1:length(Benzonase.transfac)) {
Benzonase.transfac[[i]]=transfac.func.2(Benzonase[Benzonase.split[i]:(Benzonase.split[i+1]-1),1],
                                'Benzonase_bias_plus',31,FALSE)
}
#Combine all splits of data
transfac.Benzonase <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Benzonase.transfac)) {
  transfac.Benzonase = transfac.Benzonase + Benzonase.transfac[[i]]
```

```
}

#Convert matrix back into DF
transfac.Benzonase = cbind(1:nrow(transfac.Benzonase), transfac.Benzonase)
colnames(transfac.Benzonase) = c('PO', 'A', 'C', 'G', 'T')

#Write transfac file for input into weblogo
writeLines(c("ID Benzonase_bias",
             "BF Benzonase_bias"), 'Benzonase_bias.transfac')
write.table(transfac.Benzonase, file = "Benzonase_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##Benzonase_sepcat
Benzonase_sepcat.transfac = vector("list", length = ceiling(nrow(Benzonase_sepcat)/10000000))
Benzonase_sepcat.split = seqlast(from=1,to=(nrow(Benzonase_sepcat)+1),by=10000000)

for (i in 1:length(Benzonase_sepcat.transfac)) {
Benzonase_sepcat.transfac[[i]] =
  transfac.func.2(Benzonase_sepcat[Benzonase_sepcat.split[i]:(Benzonase_sepcat.split[i+1]-1),1],
                                          'Benzonase_sepcat_bias', 31, FALSE)
}
#Combine all splits of data
transfac.Benzonase_sepcat <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Benzonase_sepcat.transfac)) {
  transfac.Benzonase_sepcat = transfac.Benzonase_sepcat + Benzonase_sepcat.transfac[[i]]
}
#Convert matrix back into DF
transfac.Benzonase_sepcat = cbind(1:nrow(transfac.Benzonase_sepcat), transfac.Benzonase_sepcat)
colnames(transfac.Benzonase_sepcat) = c('PO', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Benzonase_sepcat_bias",
             "BF Benzonase_sepcat_bias"), 'Benzonase_sepcat_bias.transfac')
write.table(transfac.Benzonase_sepcat, file = "Benzonase_sepcat_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
################################################################################
#Cyanase plus, minus, unseparated, sepcat FASTA files:
uppercasenames <- list('../mm39_data/mm39_liver_Cyanase_plus.fasta',
                       '../mm39_data/mm39_liver_Cyanase_RC.fasta',
                       '../mm39_data/mm39_liver_Cyanase.fasta',
                       '../mm39_data/mm39_liver_Cyanase_sepcat.fasta')

#Make sure all FASTA entries are uppercase
uplist <- lapply(uppercasenames, uppercase)

#Each list object of uplist is the corresponding file:
Cyanase_plus = as.data.frame(uplist[1])
Cyanase_minus = as.data.frame(uplist[2])
Cyanase = as.data.frame(uplist[3])
Cyanase_sepcat = as.data.frame(uplist[4])

rm(uplist, uppercasenames)
#
#Take each data frame and count each nucleotide at each position:
##Cyanase_plus
Cyanase_plus.transfac = vector("list", length = ceiling(nrow(Cyanase_plus)/10000000))
Cyanase_plus.split = seqlast(from=1,to=(nrow(Cyanase_plus)+1),by=10000000)

for (i in 1:length(Cyanase_plus.transfac)) {
Cyanase_plus.transfac[[i]] =
  transfac.func.2(Cyanase_plus[Cyanase_plus.split[i]:(Cyanase_plus.split[i+1]-1),1],31)
}
#Combine all splits of data
transfac.Cyanase_plus <- matrix(0,nrow = 31, ncol = 4)
```

```
for (i in 1:length(Cyanase_plus.transfac)) {
  transfac.Cyanase_plus = transfac.Cyanase_plus + Cyanase_plus.transfac[[i]]
}
#Convert matrix back into DF
transfac.Cyanase_plus = cbind(1:nrow(transfac.Cyanase_plus), transfac.Cyanase_plus)
colnames(transfac.Cyanase_plus) = c('PO', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Cyanase_plus_bias",
             "BF Cyanase_plus_bias"), 'Cyanase_plus_bias.transfac')
write.table(transfac.Cyanase_plus, file = "Cyanase_plus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##Cyanase_minus
Cyanase_minus.transfac = vector("list", length = ceiling(nrow(Cyanase_minus)/10000000))
Cyanase_minus.split = seqlast(from=1,to=(nrow(Cyanase_minus)+1),by=10000000)

for (i in 1:length(Cyanase_minus.transfac)) {
Cyanase_minus.transfac[[i]]=
  transfac.func.2(Cyanase_minus[Cyanase_minus.split[i]:(Cyanase_minus.split[i+1]-1),1], 31)
}
#Combine all splits of data
transfac.Cyanase_minus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Cyanase_minus.transfac)) {
  transfac.Cyanase_minus = transfac.Cyanase_minus + Cyanase_minus.transfac[[i]]
}
#Convert matrix back into DF
transfac.Cyanase_minus = cbind(1:nrow(transfac.Cyanase_minus), transfac.Cyanase_minus)
colnames(transfac.Cyanase_minus) = c('PO', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Cyanase_minus_bias",
             "BF Cyanase_minus_bias"), 'Cyanase_minus_bias.transfac')
write.table(transfac.Cyanase_minus, file = "Cyanase_minus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##Cyanase_nosep
Cyanase.transfac = vector("list", length = ceiling(nrow(Cyanase)/10000000))
Cyanase.split = seqlast(from=1,to=(nrow(Cyanase)+1),by=10000000)

for (i in 1:length(Cyanase.transfac)) {
Cyanase.transfac[[i]]=transfac.func.2(Cyanase[Cyanase.split[i]:(Cyanase.split[i+1]-1),1],
                                'Cyanase_bias_plus',31,FALSE)
}
#Combine all splits of data
transfac.Cyanase <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Cyanase.transfac)) {
  transfac.Cyanase = transfac.Cyanase + Cyanase.transfac[[i]]
}

#Convert matrix back into DF
transfac.Cyanase = cbind(1:nrow(transfac.Cyanase), transfac.Cyanase)
colnames(transfac.Cyanase) = c('PO', 'A', 'C', 'G', 'T')

#Write transfac file for input into weblogo
writeLines(c("ID Cyanase_bias",
             "BF Cyanase_bias"), 'Cyanase_bias.transfac')
write.table(transfac.Cyanase, file = "Cyanase_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##Cyanase_sepcat
Cyanase_sepcat.transfac = vector("list", length = ceiling(nrow(Cyanase_sepcat)/10000000))
Cyanase_sepcat.split = seqlast(from=1,to=(nrow(Cyanase_sepcat)+1),by=10000000)

for (i in 1:length(Cyanase_sepcat.transfac)) {
Cyanase_sepcat.transfac[[i]] =
```

```
  transfac.func.2(Cyanase_sepcat[Cyanase_sepcat.split[i]:(Cyanase_sepcat.split[i+1]-1),1],
                                          'Cyanase_sepcat_bias', 31, FALSE)
}
#Combine all splits of data
transfac.Cyanase_sepcat <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Cyanase_sepcat.transfac)) {
  transfac.Cyanase_sepcat = transfac.Cyanase_sepcat + Cyanase_sepcat.transfac[[i]]
}
#Convert matrix back into DF
transfac.Cyanase_sepcat = cbind(1:nrow(transfac.Cyanase_sepcat), transfac.Cyanase_sepcat)
colnames(transfac.Cyanase_sepcat) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Cyanase_sepcat_bias",
             "BF Cyanase_sepcat_bias"), 'Cyanase_sepcat_bias.transfac')
write.table(transfac.Cyanase_sepcat, file = "Cyanase_sepcat_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
################################################################################
#MNase plus, minus, unseparated, sepcat FASTA files:
uppercasenames <- list('../mm39_data/mm39_liver_MNase_plus.fasta',
                       '../mm39_data/mm39_liver_MNase_RC.fasta',
                       '../mm39_data/mm39_liver_MNase.fasta',
                       '../mm39_data/mm39_liver_MNase_sepcat.fasta')

#Make sure all FASTA entries are uppercase
uplist <- lapply(uppercasenames, uppercase)

#Each list object of uplist is the corresponding file:
MNase_plus = as.data.frame(uplist[1])
MNase_minus = as.data.frame(uplist[2])
MNase = as.data.frame(uplist[3])
MNase_sepcat = as.data.frame(uplist[4])

rm(uplist, uppercasenames)
#
#Take each data frame and count each nucleotide at each position:
##MNase_plus
MNase_plus.transfac = vector("list", length = ceiling(nrow(MNase_plus)/10000000))
MNase_plus.split = seqlast(from=1,to=(nrow(MNase_plus)+1),by=10000000)

for (i in 1:length(MNase_plus.transfac)) {
MNase_plus.transfac[[i]] =
  transfac.func.2(MNase_plus[MNase_plus.split[i]:(MNase_plus.split[i+1]-1),1],31)
}
#Combine all splits of data
transfac.MNase_plus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(MNase_plus.transfac)) {
  transfac.MNase_plus = transfac.MNase_plus + MNase_plus.transfac[[i]]
}
#Convert matrix back into DF
transfac.MNase_plus = cbind(1:nrow(transfac.MNase_plus), transfac.MNase_plus)
colnames(transfac.MNase_plus) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID MNase_plus_bias",
             "BF MNase_plus_bias"), 'MNase_plus_bias.transfac')
write.table(transfac.MNase_plus, file = "MNase_plus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##MNase_minus
MNase_minus.transfac = vector("list", length = ceiling(nrow(MNase_minus)/10000000))
MNase_minus.split = seqlast(from=1,to=(nrow(MNase_minus)+1),by=10000000)

for (i in 1:length(MNase_minus.transfac)) {
MNase_minus.transfac[[i]]=
```

```
  transfac.func.2(MNase_minus[MNase_minus.split[i]:(MNase_minus.split[i+1]-1),1], 31)
}
#Combine all splits of data
transfac.MNase_minus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(MNase_minus.transfac)) {
  transfac.MNase_minus = transfac.MNase_minus + MNase_minus.transfac[[i]]
}
#Convert matrix back into DF
transfac.MNase_minus = cbind(1:nrow(transfac.MNase_minus), transfac.MNase_minus)
colnames(transfac.MNase_minus) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID MNase_minus_bias",
             "BF MNase_minus_bias"), 'MNase_minus_bias.transfac')
write.table(transfac.MNase_minus, file = "MNase_minus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
##MNase_nosep
MNase.transfac = vector("list", length = ceiling(nrow(MNase)/10000000))
MNase.split = seqlast(from=1,to=(nrow(MNase)+1),by=10000000)

for (i in 1:length(MNase.transfac)) {
MNase.transfac[[i]]=transfac.func.2(MNase[MNase.split[i]:(MNase.split[i+1]-1),1],
                                'MNase_bias_plus',31,FALSE)
}
#Combine all splits of data
transfac.MNase <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(MNase.transfac)) {
  transfac.MNase = transfac.MNase + MNase.transfac[[i]]
}

#Convert matrix back into DF
transfac.MNase = cbind(1:nrow(transfac.MNase), transfac.MNase)
colnames(transfac.MNase) = c('P0', 'A', 'C', 'G', 'T')

#Write transfac file for input into weblogo
writeLines(c("ID MNase_bias",
             "BF MNase_bias"), 'MNase_bias.transfac')
write.table(transfac.MNase, file = "MNase_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
##MNase_sepcat
MNase_sepcat.transfac = vector("list", length = ceiling(nrow(MNase_sepcat)/10000000))
MNase_sepcat.split = seqlast(from=1,to=(nrow(MNase_sepcat)+1),by=10000000)

for (i in 1:length(MNase_sepcat.transfac)) {
MNase_sepcat.transfac[[i]] =
  transfac.func.2(MNase_sepcat[MNase_sepcat.split[i]:(MNase_sepcat.split[i+1]-1),1],
                                        'MNase_sepcat_bias', 31, FALSE)
}
#Combine all splits of data
transfac.MNase_sepcat <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(MNase_sepcat.transfac)) {
  transfac.MNase_sepcat = transfac.MNase_sepcat + MNase_sepcat.transfac[[i]]
}
#Convert matrix back into DF
transfac.MNase_sepcat = cbind(1:nrow(transfac.MNase_sepcat), transfac.MNase_sepcat)
colnames(transfac.MNase_sepcat) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID MNase_sepcat_bias",
             "BF MNase_sepcat_bias"), 'MNase_sepcat_bias.transfac')
write.table(transfac.MNase_sepcat, file = "MNase_sepcat_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
###############################################################################
```

```
#DNase plus, minus, unseparated, sepcat FASTA files:
uppercasenames <- list('../hg38_data/DNase_Naked_unscaled_plus.fasta',
                       '../hg38_data/DNase_Naked_unscaled_RC.fasta',
                       '../hg38_data/DNase_Naked_unscaled.fasta',
                       '../hg38_data/DNase_Naked_unscaled_sepcat.fasta')

#Make sure all FASTA entries are uppercase
uplist <- lapply(uppercasenames, uppercase)

#Each list object of uplist is the corresponding file:
DNase_plus = as.data.frame(uplist[1])
DNase_minus = as.data.frame(uplist[2])
DNase = as.data.frame(uplist[3])
DNase_sepcat = as.data.frame(uplist[4])

rm(uplist, uppercasenames)
#
#Take each data frame and count each nucleotide at each position:
##DNase_plus
DNase_plus.transfac = vector("list", length = ceiling(nrow(DNase_plus)/10000000))
DNase_plus.split = seqlast(from=1,to=(nrow(DNase_plus)+1),by=10000000)

for (i in 1:length(DNase_plus.transfac)) {
DNase_plus.transfac[[i]] =
  transfac.func.2(DNase_plus[DNase_plus.split[i]:(DNase_plus.split[i+1]-1),1],31)
}
#Combine all splits of data
transfac.DNase_plus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(DNase_plus.transfac)) {
  transfac.DNase_plus = transfac.DNase_plus + DNase_plus.transfac[[i]]
}
#Convert matrix back into DF
transfac.DNase_plus = cbind(1:nrow(transfac.DNase_plus), transfac.DNase_plus)
colnames(transfac.DNase_plus) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID DNase_plus_bias",
             "BF DNase_plus_bias"), 'DNase_plus_bias.transfac')
write.table(transfac.DNase_plus, file = "DNase_plus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##DNase_minus
DNase_minus.transfac = vector("list", length = ceiling(nrow(DNase_minus)/10000000))
DNase_minus.split = seqlast(from=1,to=(nrow(DNase_minus)+1),by=10000000)

for (i in 1:length(DNase_minus.transfac)) {
DNase_minus.transfac[[i]]=
  transfac.func.2(DNase_minus[DNase_minus.split[i]:(DNase_minus.split[i+1]-1),1], 31)
}
#Combine all splits of data
transfac.DNase_minus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(DNase_minus.transfac)) {
  transfac.DNase_minus = transfac.DNase_minus + DNase_minus.transfac[[i]]
}
#Convert matrix back into DF
transfac.DNase_minus = cbind(1:nrow(transfac.DNase_minus), transfac.DNase_minus)
colnames(transfac.DNase_minus) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID DNase_minus_bias",
             "BF DNase_minus_bias"), 'DNase_minus_bias.transfac')
write.table(transfac.DNase_minus, file = "DNase_minus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
################################################################################
##DNase_nosep
DNase.transfac = vector("list", length = ceiling(nrow(DNase)/10000000))
```

```
DNase.split = seqlast(from=1,to=(nrow(DNase)+1),by=10000000)

for (i in 1:length(DNase.transfac)) {
DNase.transfac[[i]]=transfac.func.2(DNase[DNase.split[i]:(DNase.split[i+1]-1),1],
                                    'DNase_bias_plus',31,FALSE)
}
#Combine all splits of data
transfac.DNase <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(DNase.transfac)) {
  transfac.DNase = transfac.DNase + DNase.transfac[[i]]
}

#Convert matrix back into DF
transfac.DNase = cbind(1:nrow(transfac.DNase), transfac.DNase)
colnames(transfac.DNase) = c('P0', 'A', 'C', 'G', 'T')

#Write transfac file for input into weblogo
writeLines(c("ID DNase_bias",
             "BF DNase_bias"), 'DNase_bias.transfac')
write.table(transfac.DNase, file = "DNase_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
##############################################################################
##DNase_sepcat
DNase_sepcat.transfac = vector("list", length = ceiling(nrow(DNase_sepcat)/10000000))
DNase_sepcat.split = seqlast(from=1,to=(nrow(DNase_sepcat)+1),by=10000000)

for (i in 1:length(DNase_sepcat.transfac)) {
DNase_sepcat.transfac[[i]] =
  transfac.func.2(DNase_sepcat[DNase_sepcat.split[i]:(DNase_sepcat.split[i+1]-1),1],
                                          'DNase_sepcat_bias', 31, FALSE)
}
#Combine all splits of data
transfac.DNase_sepcat <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(DNase_sepcat.transfac)) {
  transfac.DNase_sepcat = transfac.DNase_sepcat + DNase_sepcat.transfac[[i]]
}
#Convert matrix back into DF
transfac.DNase_sepcat = cbind(1:nrow(transfac.DNase_sepcat), transfac.DNase_sepcat)
colnames(transfac.DNase_sepcat) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID DNase_sepcat_bias",
             "BF DNase_sepcat_bias"), 'DNase_sepcat_bias.transfac')
write.table(transfac.DNase_sepcat, file = "DNase_sepcat_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
##############################################################################
##############################################################################
#Tn5 plus, minus, unseparated, sepcat FASTA files:
uppercasenames <- list('../hg38_data/C1_gDNA_rep1_plus.fasta',
                       '../hg38_data/C1_gDNA_rep1_RC.fasta',
                       '../hg38_data/C1_gDNA_rep1.fasta',
                       '../hg38_data/C1_gDNA_rep1_sepcat.fasta')
#Make sure all FASTA entries are uppercase
uplist <- lapply(uppercasenames, uppercase)
#Each list object of uplist is the corresponding file:
Tn5_plus = as.data.frame(uplist[1])
Tn5_minus = as.data.frame(uplist[2])
Tn5 = as.data.frame(uplist[3])
Tn5_sepcat = as.data.frame(uplist[4])

rm(uplist, uppercasenames)
#
#Take each data frame and count each nucleotide at each position:
##Tn5_plus
Tn5_plus.transfac = vector("list", length = ceiling(nrow(Tn5_plus)/10000000))
```

```
Tn5_plus.split = seqlast(from=1,to=(nrow(Tn5_plus)+1),by=10000000)

for (i in 1:length(Tn5_plus.transfac)) {
Tn5_plus.transfac[[i]] =
  transfac.func.2(Tn5_plus[Tn5_plus.split[i]:(Tn5_plus.split[i+1]-1),1],31)
}
#Combine all splits of data
transfac.Tn5_plus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Tn5_plus.transfac)) {
  transfac.Tn5_plus = transfac.Tn5_plus + Tn5_plus.transfac[[i]]
}
#Convert matrix back into DF
transfac.Tn5_plus = cbind(1:nrow(transfac.Tn5_plus), transfac.Tn5_plus)
colnames(transfac.Tn5_plus) = c('PO', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Tn5_plus_bias",
             "BF Tn5_plus_bias"), 'Tn5_plus_bias.transfac')
write.table(transfac.Tn5_plus, file = "Tn5_plus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
##Tn5_minus
Tn5_minus.transfac = vector("list", length = ceiling(nrow(Tn5_minus)/10000000))
Tn5_minus.split = seqlast(from=1,to=(nrow(Tn5_minus)+1),by=10000000)

for (i in 1:length(Tn5_minus.transfac)) {
Tn5_minus.transfac[[i]]=
  transfac.func.2(Tn5_minus[Tn5_minus.split[i]:(Tn5_minus.split[i+1]-1),1], 31)
}
#Combine all splits of data
transfac.Tn5_minus <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Tn5_minus.transfac)) {
  transfac.Tn5_minus = transfac.Tn5_minus + Tn5_minus.transfac[[i]]
}
#Convert matrix back into DF
transfac.Tn5_minus = cbind(1:nrow(transfac.Tn5_minus), transfac.Tn5_minus)
colnames(transfac.Tn5_minus) = c('PO', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Tn5_minus_bias",
             "BF Tn5_minus_bias"), 'Tn5_minus_bias.transfac')
write.table(transfac.Tn5_minus, file = "Tn5_minus_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
###############################################################################
##Tn5_nosep
Tn5.transfac = vector("list", length = ceiling(nrow(Tn5)/10000000))
Tn5.split = seqlast(from=1,to=(nrow(Tn5)+1),by=10000000)

for (i in 1:length(Tn5.transfac)) {
Tn5.transfac[[i]]=transfac.func.2(Tn5[Tn5.split[i]:(Tn5.split[i+1]-1),1],
                                  'Tn5_bias_plus',31,FALSE)
}
#Combine all splits of data
transfac.Tn5 <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Tn5.transfac)) {
  transfac.Tn5 = transfac.Tn5 + Tn5.transfac[[i]]
}

#Convert matrix back into DF
transfac.Tn5 = cbind(1:nrow(transfac.Tn5), transfac.Tn5)
colnames(transfac.Tn5) = c('PO', 'A', 'C', 'G', 'T')

#Write transfac file for input into weblogo
writeLines(c("ID Tn5_bias",
             "BF Tn5_bias"), 'Tn5_bias.transfac')
write.table(transfac.Tn5, file = "Tn5_bias.transfac",
```

```
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
##############################################################################
##Tn5_sepcat
Tn5_sepcat.transfac = vector("list", length = ceiling(nrow(Tn5_sepcat)/10000000))
Tn5_sepcat.split = seqlast(from=1,to=(nrow(Tn5_sepcat)+1),by=10000000)

for (i in 1:length(Tn5_sepcat.transfac)) {
Tn5_sepcat.transfac[[i]] =
  transfac.func.2(Tn5_sepcat[Tn5_sepcat.split[i]:(Tn5_sepcat.split[i+1]-1),1],
                                          'Tn5_sepcat_bias', 31, FALSE)
}
#Combine all splits of data
transfac.Tn5_sepcat <- matrix(0,nrow = 31, ncol = 4)
for (i in 1:length(Tn5_sepcat.transfac)) {
  transfac.Tn5_sepcat = transfac.Tn5_sepcat + Tn5_sepcat.transfac[[i]]
}
#Convert matrix back into DF
transfac.Tn5_sepcat = cbind(1:nrow(transfac.Tn5_sepcat), transfac.Tn5_sepcat)
colnames(transfac.Tn5_sepcat) = c('P0', 'A', 'C', 'G', 'T')
#Write transfac file for input into weblogo
writeLines(c("ID Tn5_sepcat_bias",
             "BF Tn5_sepcat_bias"), 'Tn5_sepcat_bias.transfac')
write.table(transfac.Tn5_sepcat, file = "Tn5_sepcat_bias.transfac",
            append = TRUE, quote=FALSE, row.names =FALSE, col.names = TRUE, sep = '\t')
##############################################################################
```

2.5 Modifying the Weblogo python CLI to output information content file

In order to quantify information content calculated using Weblogo (as seen in figure 1 insets), modifying the Weblogo CLI source code is necessary. This modification was done on the Weblogo 3.7 distribution. The source code can be found here (accessed on Sep 16, 2022): https://github.com/WebLogo/weblogo/newline Open the file 'logo_formatter.py' in the Weblogo folder.
Enter the following code at the specified lines in parentheses (screenshots included for reference, as line numbers will change with modifications):

(line 7)

```
import sys
```

```
1  """
2  Logo formatting. Each formatter is a function f(data, format) that draws
3  a representation of the logo. The main graphical formatter is eps_formatter. A mapping
4  'formatters' containing all available formatters . Each formatter returns binary data. The eps
5  and data formats can decoded to strings, e.g. eps_as_string = eps_data.decode()
6  """
7  import sys
8  import os
9  import shutil
10 from math import log
11 from string import Template
12 from subprocess import PIPE, Popen
13 from typing import Optional
```

(line 122)

```
extra_lines = []
```

```
  def eps_formatter(logodata: LogoData, logoformat: LogoFormat) -> bytes:
      """Generate a logo in Encapsulated Postscript (EPS)"""
      substitutions = {}
      extra_lines = []
      from_format = [
```

(line 223)

```
extra_lines.append(stack_height)
```

```
          data.append("(%s) StartStack" % logoformat.annotate[seq_index])
          if conv_factor:
              assert logodata.entropy is not None
              assert logoformat.unit_name is not None
              stack_height = logodata.entropy[seq_index] * std_units[logoformat.unit_name]
              extra_lines.append(stack_height)
          else:
              stack_height = 1.0  # probability   # pragma: no cover
```

(line 292)

```
print(extra_lines)

extra_lines_save = [sys.argv[6][:-4], "_bitvals.txt"]
extra_lines_save = "".join(extra_lines_save)
print(extra_lines_save)
with open(extra_lines_save, 'w') as f:
       for s in extra_lines:
           f.write(str(s) + '\n')
```

```
      # Create and output logo
      template = resource_string(__name__, "template.eps", __file__).decode()

      logo = Template(template).substitute(substitutions)
      print(extra_lines)

      extra_lines_save = [sys.argv[6][:-4], "_bitvals.txt"]
      extra_lines_save = "".join(extra_lines_save)
      print(extra_lines_save)
      with open(extra_lines_save, 'w') as f:
          for s in extra_lines:
              f.write(str(s) + '\n')


      return logo.encode()
```

2.6 Determining background nucleotide frequency in mm39 and hg38 reference genomes

To determine background nucleotide frequency in the hg38 and mm39 reference genomes, a simple grep command may be used. To ensure that chromosome names aren't counted, a grep command using the chrom.sizes files for either genome may be used to

correct these counts.

```
#Find number of C in hg38:
grep -o "c\|C" ../data/hg38.fa | wc -l
#623727797
grep -o "c\|C" ../data/hg38.fa.chrom.sizes | wc -l
#455
#623727797-455 = 623727342

#Find number of G in hg38:
grep -o "g\|G" ../data/hg38.fa | wc -l
#626335226
grep -o "g\|G" ../data/hg38.fa.chrom.sizes | wc -l
#89
#626335226-89 = 626335137

#Find number of A in hg38:
grep -o "a\|A" ../data/hg38.fa | wc -l
#898285722
grep -o "a\|A" ../data/hg38.fa.chrom.sizes | wc -l
#303
#898285722-303 = 898285419

#Find number of T in hg38:
grep -o "t\|T" ../data/hg38.fa | wc -l
#900968146
grep -o "t\|T" ../data/hg38.fa.chrom.sizes | wc -l
#261
#900968146-261 = 900967885

#hg38
#623727342+626335137+898285419+900967885=3049315783
#C = 623727342/3049315783 = 0.2045467
#G = 626335137/3049315783 = 0.2054019
#A = 898285419/3049315783 = 0.2945859
#T = 900967885/3049315783 = 0.2954656
#0.2045467 + 0.2054019 + 0.2945859 + 0.2954656 = 1


#Find number of C in mm39:
grep -o "c\|C" ../data/mm39.fa | wc -l
#553008726
grep -o "c\|C" ../data/mm39.fa.chrom.sizes | wc -l
#61
#553008726-61 = 553008665

#Find number of G in mm39:
grep -o "g\|G" ../data/mm39.fa | wc -l
#553055984
grep -o "g\|G" ../data/mm39.fa.chrom.sizes | wc -l
#27
#553055984-27 = 553055957

#Find number of A in mm39:
grep -o "a\|A" ../data/mm39.fa | wc -l
#773810667
grep -o "a\|A" ../data/mm39.fa.chrom.sizes | wc -l
#18
#773810667-18 = 773810649

#Find number of T in mm39:
grep -o "t\|T" ../data/mm39.fa | wc -l
#774746512
grep -o "t\|T" ../data/mm39.fa.chrom.sizes | wc -l
```

```
#0
#774746512-0 = 774746512

#mm39
#553008665 + 553055957 + 773810649 + 774746512 = 2654621783
#C = 553008665/2654621783 = 0.2083192
#G = 553055957/2654621783 = 0.208337
#A = 773810649/2654621783 = 0.2914956
#T = 774746512/2654621783 = 0.2918482
#0.2083192 + 0.208337 + 0.2914956 + 0.2918482 = 1
```

2.7 Plotting enzymatic sequence bias sequence logos for genomic background frequency and equiprobable background frequency (figure S1) using weblogo

This will plot sequence logos for each enzyme's enzymatic sequence bias as seen in figure 1 and S1. These plots will also output information content files labeled with "_bitvals.txt" if the modifications to weblogo as previously outlined are implemented.

```
weblogo -f ../output/Figure1A_DNase_sepcat_bias.transfac -D transfac -o Figure1A_DNase_sepcat_bias.eps \
        --color-scheme classic -F eps -S 0.35 -Y YES -s large \
        --composition "{'A':29.5, 'C':20.5, 'G':20.5, 'T':29.5}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Tn5_sepcat_bias.transfac -D transfac -o Figure1A_Tn5_sepcat_bias.eps \
        --color-scheme classic -F eps -S 0.35 -Y YES -s large \
        --composition "{'A':29.5, 'C':20.5, 'G':20.5, 'T':29.5}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Benzonase_sepcat_bias.transfac -D transfac -o Figure1A_Benzonase_sepcat_bias.eps \
        --color-scheme classic -F eps -S 0.35 -Y YES -s large \
        --composition "{'A':29.2, 'C':20.8, 'G':20.8, 'T':29.2}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Cyanase_sepcat_bias.transfac -D transfac -o Figure1A_Cyanase_sepcat_bias.eps \
        --color-scheme classic -F eps -S 0.35 -Y YES -s large \
        --composition "{'A':29.2, 'C':20.8, 'G':20.8, 'T':29.2}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_MNase_sepcat_bias.transfac -D transfac -o Figure1A_MNase_sepcat_bias.eps \
        --color-scheme classic -F eps -S 0.35 -Y YES -s large \
        --composition "{'A':29.2, 'C':20.8, 'G':20.8, 'T':29.2}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO
#########################################################################################################################
weblogo -f ../output/Figure1A_Tn5_sepcat_bias.transfac -D transfac -o Figure1A_Tn5_sepcat_bias_greatestval.eps \
        --color-scheme classic -F eps -S 0.31 -Y YES -s large \
        --composition "{'A':29.5, 'C':20.5, 'G':20.5, 'T':29.5}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Benzonase_sepcat_bias.transfac -D transfac \
        -o Figure1A_Benzonase_sepcat_bias_greatestval.eps \
        --color-scheme classic -F eps -S 0.16 -Y YES -s large \
        --composition "{'A':29.2, 'C':20.8, 'G':20.8, 'T':29.2}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Cyanase_sepcat_bias.transfac -D transfac \
        -o Figure1A_Cyanase_sepcat_bias_greatestval.eps --color-scheme classic \
```

```
        -F eps -S 0.13 -Y YES -s large \
        --composition "{'A':29.2, 'C':20.8, 'G':20.8, 'T':29.2}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_MNase_sepcat_bias.transfac -D transfac \
        -o Figure1A_MNase_sepcat_bias_greatestval.eps --color-scheme classic \
        -F eps -S 0.31 -Y YES -s large  \
        --composition "{'A':29.2, 'C':20.8, 'G':20.8, 'T':29.2}" --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

#############################################################################################################
weblogo -f ../output/Figure1A_DNase_sepcat_bias.transfac -D transfac \
        -o Figure1A_DNase_sepcat_bias_EQUIPROBABLE_BACKGROUND.eps --color-scheme classic \
        -F eps -S 0.31 -Y YES -s large --composition equiprobable --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Tn5_sepcat_bias.transfac -D transfac \
        -o Figure1A_Tn5_sepcat_bias_EQUIPROBABLE_BACKGROUND.eps --color-scheme classic \
        -F eps -S 0.18 -Y YES -s large --composition equiprobable --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Benzonase_sepcat_bias.transfac -D transfac \
        -o Figure1A_Benzonase_sepcat_bias_EQUIPROBABLE_BACKGROUND.eps --color-scheme classic \
        -F eps -S 0.1 -Y YES -s large --composition equiprobable --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_Cyanase_sepcat_bias.transfac -D transfac \
        -o Figure1A_Cyanase_sepcat_bias_EQUIPROBABLE_BACKGROUND.eps --color-scheme classic \
        -F eps -S 0.1 -Y YES -s large --composition equiprobable --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO

weblogo -f ../output/Figure1A_MNase_sepcat_bias.transfac -D transfac \
        -o Figure1A_MNase_sepcat_bias_EQUIPROBABLE_BACKGROUND.eps --color-scheme classic \
        -F eps -S 0.42 -Y YES -s large --composition equiprobable --logo-font Arial-BoldMT \
        --annotate '-15,-14,-13,-12,-11,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15' \
        --number-fontsize 14 --rotate-numbers YES --fineprint '' --ylabel '' --errorbars NO
```

2.8 Weblogo background correction calculations

Using the output positional information content from weblogo as confirmation, the formula used for Weblogo background correction were determined.
First, classic uncorrected Shannon entropy of the nucleotide frequencies is determined:

$$H(l) = -\sum_{b=a}^{t} = f(b,l) log_2 f(b,l)$$

Where H(l) is the entropy at any given position and f(b,l) is the frequency of a base (b) at this position (l). Subtracting this value from 2 is the classic calculation for information content.

Next, background entropy (deviation from equiprobable) is calculated:

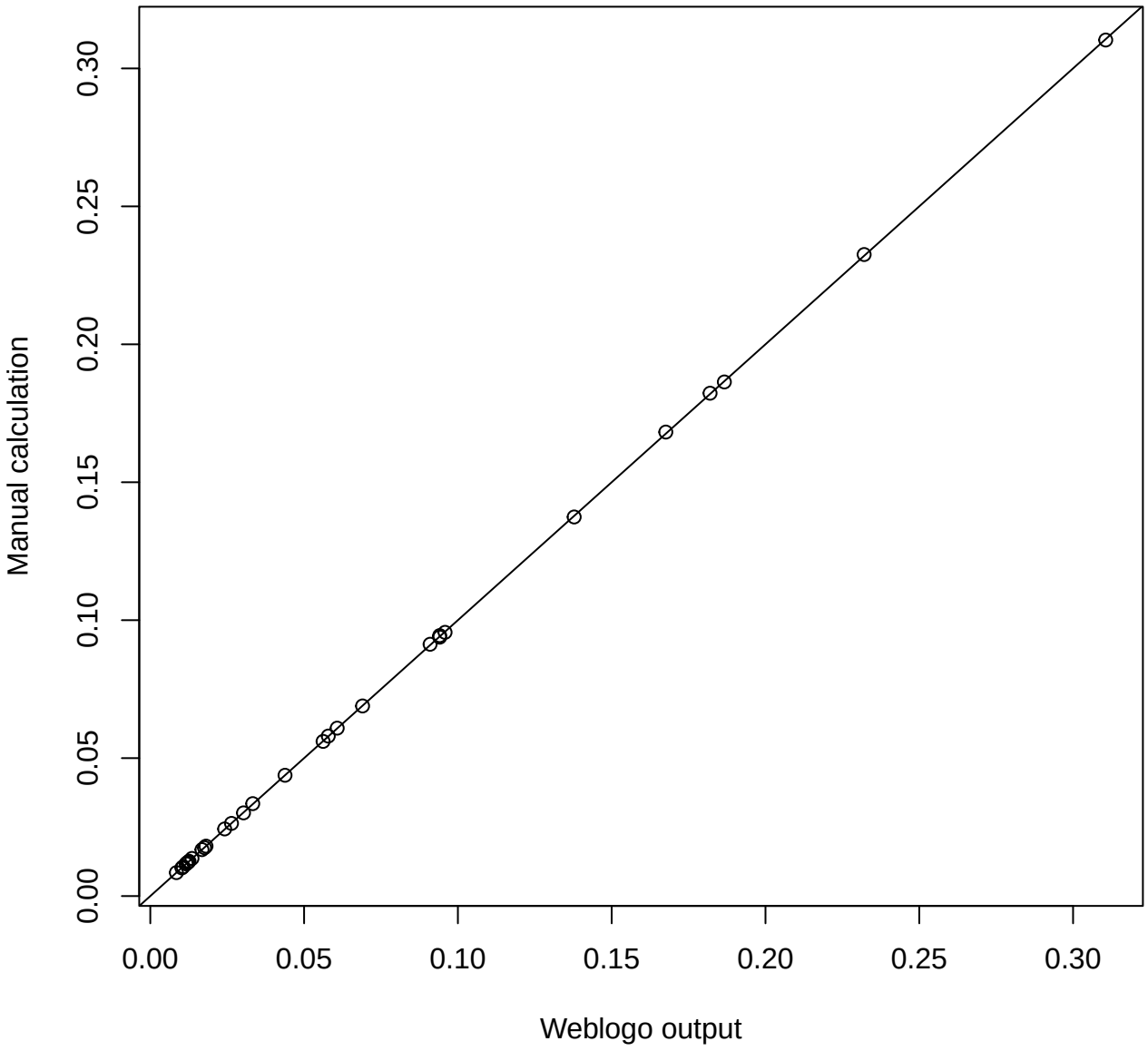
$$H(background) = -\sum_{b=a}^{t} = f(b,l) log_2 (\frac{0.25}{f(background,b)})$$

Where H(background) is the correction for background nucleotide frequency. And f(background,b) is the background frequency (background) for a base (b).

Thus, for any position, the background corrected information content is:

$$Information\ Content = 2 - (H(l) + H(background))$$

As a coherence check, we plotted manually calculated information content for background corrected Tn5 sequence bias motif against Weblogo output:



2.9 Converting transfac files into MEME file format for FIMO

```
#Add 'XX' to end of each transfac document
endline='XX'
for i in *.transfac
do
    echo $endline >> $i
done
```

```
#Convert transfacs to memes
for i in *.transfac
do
    name=$(echo $i | awk -F".transfac" '{print $1}')
    echo $name
    transfac2meme $i > $name.meme
done
```

2.10 Using the FIMO algorithm to find instances of highest conformation to enzyme bias

This allows us to find the composite signal at sites for each enzyme corresponding to highest bias.

```
fimo --thresh 0.00005 --text ../output/Figure1A_Tn5_sepcat_bias.meme \
                             ../hg38_data/hg38.fa > Tn5_sepcat_bias_transfac_00005_fimo.txt

fimo --thresh 0.0001 --text ../output/Figure1A_DNase_sepcat_bias.meme \
                            ../hg38_data/hg38.fa > DNase_sepcat_bias_transfac_0001_fimo.txt

fimo --thresh 0.00007 --text ../output/Figure1A_Cyanase_sepcat_bias.meme \
                             ../mm39_data/mm39.fa > Cyanase_sepcat_bias_transfac_00007_fimo.txt

fimo --thresh 0.00005 --text ../output/Figure1A_Benzonase_sepcat_bias.meme \
                             ../mm39_data/mm39.fa > Benzonase_sepcat_bias_transfac_00005_fimo.txt

fimo --thresh 0.0001 --text ../output/Figure1A_MNase_sepcat_bias.meme \
                            ../mm39_data/mm39.fa > MNase_sepcat_bias_transfac_0001_fimo.txt
```

2.11 Plotting composite profiles of each enzyme's bias

Here, we use the FIMO results to plot the regions with highest conformation to each enzyme's bias.

```
library(bigWig)
library(zoo)
library(lattice)
source('Figure1_functions.R')
setwd('output')

############################Benzonase unsep/minus/plus
Motifs <- c('Benzonase_sepcat_bias_transfac_00005_fimo.txt')
Motiflist <- vector('list', length(Motifs))

Motiflist[[1]] = FIMO.to.BED(Motifs)
names(Motiflist) <- Motifs

BWs <- c('mm39_liver_Benzonase_plus.bigWig', 'mm39_liver_Benzonase_minus.bigWig')

merged_benz <- BED.query.bigWig(Motiflist[[1]], BWs[1], BWs[2],
                    upstream = 50, downstream = 50, factor = 'Benzonase strand-specific',
                    group = 'Unscaled', ATAC = FALSE)

unscaled_composite_agg <- rbind(merged_benz)

############################Cyanase unsep/minus/plus
Motifs <- c('Cyanase_sepcat_bias_transfac_00005_fimo.txt')
Motiflist <- vector('list', length(Motifs))

Motiflist[[1]] = FIMO.to.BED(Motifs)
names(Motiflist) <- Motifs

BWs <- c('mm39_liver_Cyanase_plus.bigWig', 'mm39_liver_Cyanase_minus.bigWig')
```

```
merged_cyan <- BED.query.bigWig(Motiflist[[1]], BWs[1], BWs[2],
                                upstream = 50, downstream = 50, factor = 'Cyanase strand-specific',
                                group = 'Unscaled', ATAC = FALSE)


unscaled_composite_agg <- rbind(unscaled_composite_agg, merged_cyan)

##########################DNase unsep/minus/plus
Motifs <- c('DNase_sepcat_bias_transfac_00005_fimo.txt')
Motiflist <- vector('list', length(Motifs))

Motiflist[[1]] = FIMO.to.BED(Motifs)
names(Motiflist) <- Motifs

BWs <- c('mm39_liver_DNase_plus.bigWig', 'mm39_liver_DNase_minus.bigWig')

merged_DNase <- BED.query.bigWig(Motiflist[[1]], BWs[1], BWs[2],
                                upstream = 50, downstream = 50, factor = 'DNase strand-specific',
                                group = 'Unscaled', ATAC = FALSE)


unscaled_composite_agg <- rbind(unscaled_composite_agg, merged_DNase)

##########################MNase unsep/minus/plus
Motifs <- c('MNase_sepcat_bias_transfac_00005_fimo.txt')
Motiflist <- vector('list', length(Motifs))

Motiflist[[1]] = FIMO.to.BED(Motifs)
names(Motiflist) <- Motifs

BWs <- c('mm39_liver_MNase_plus.bigWig', 'mm39_liver_MNase_minus.bigWig')

merged_MNase <- BED.query.bigWig(Motiflist[[1]], BWs[1], BWs[2],
                                 upstream = 50, downstream = 50, factor = 'MNase strand-specific',
                                 group = 'Unscaled', ATAC = FALSE)


unscaled_composite_agg <- rbind(unscaled_composite_agg, merged_MNase)

##########################Tn5 unsep/minus/plus
Motifs <- c('Tn5_sepcat_bias_transfac_00005_fimo.txt')
Motiflist <- vector('list', length(Motifs))

Motiflist[[1]] = FIMO.to.BED(Motifs)
names(Motiflist) <- Motifs

BWs <- c('Tn5_C1_gDNA_rep1_plus.bigWig', 'Tn5_C1_gDNA_rep1_minus.bigWig')

merged_Tn5 <- BED.query.bigWig(Motiflist[[1]], BWs[1], BWs[2],
                                upstream = 50, downstream = 50, factor = 'Tn5 strand-specific',
                                group = 'Unscaled', ATAC = TRUE)

unscaled_composite_agg <- rbind(unscaled_composite_agg, merged_Tn5)

###Plot these all together
ssplot <- plot.composites(unscaled_composite_agg, legend = FALSE,
                          ylabel = 'Cut Frequency',
                          xlabel = 'Distance from Motif Center',
                          figwidth = 4, figheight = 12,
                          motifline = FALSE,
                          Motiflen = mlen,
                          striplabel = TRUE,
                          pdf_name = 'unscaled_striplabel_ss_composites',
```

```
                        indexlist = list(c(3,2,1,4,5)),
                        layoutgrid = c(1,5)
)
```

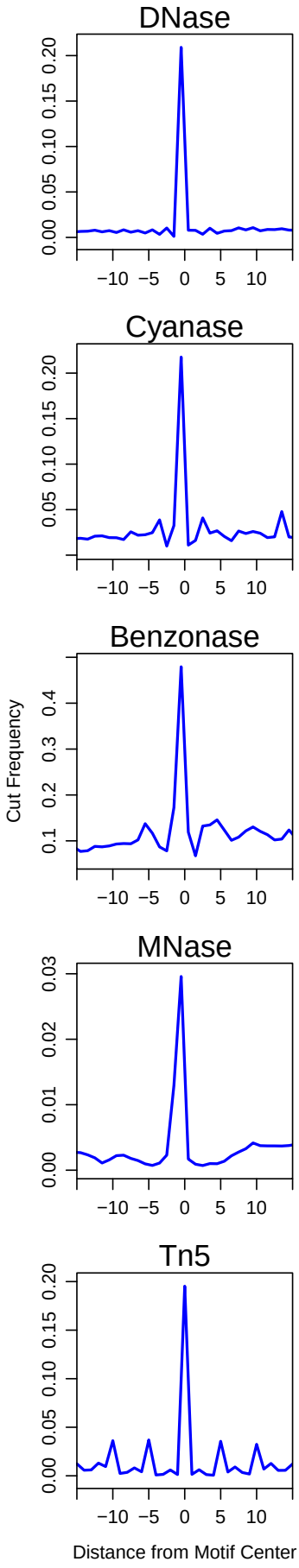


Figure 2: Unscaled enzyme bias composites

2.12 Generating iterative seqtables with seqOutBias

Using a for loop, and parallel computing, we run many instances of seqOutBias in order to generate seqtable files, so we may calculate scale factors for each 5-mer position within a 15bp range of the cutsite.

```
library(parallel)
library(bigWig)
source("https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_hcsearch.R")

system('mkdir EnzymeBias_masks')
setwd('EnzymeBias_masks')
neighbors <- function(vecmask) {
  # generate list of neighboring masks that differ
  # by the addition of a single unmasked position
  result <- vector(mode="list", length=length(vecmask))
  n <- 0
  for (k in 1:length(vecmask)) {
    if (vecmask[k] == -1) { # X
      n <- n + 1
      vi = vecmask
      vi[k] = 1 # N
      vi[k+1] = 1 # 2nd N
      vi[k+2] = 1 # 3rd N
      vi[k+3] = 1 # 4th N
      vi[k+4] = 1 # 5th N
      result[[n]] <- vec.to.mask(vi)
    }
  }
  if (n == 0) return(NULL)
  #Remove last line so you don't get an extra N space (mappable bases = mer-1)
  zz <- (n-4)
  result[1:zz]
}

###Make Tn5 masks-we need a wider range because we will need to shift these by 4 (this will take hours)
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../hg38_data/hg38.fa ../hg38_data/C1_gDNA_rep1.bam --read-size=76 --strand-specific --custom-shift=4,-
sqcmd = "seqOutBias"
prefix = "Tn5_"

masks = mclapply(nextlst, function(cutmask) {
  bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 3)

###Make DNase masks (this will take hours)
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../hg38_data/hg38.fa ../hg38_data/DNase_Naked.bam --read-size=76 --strand-specific"
sqcmd = "seqOutBias"
prefix = "DNase_"

masks = mclapply(nextlst, function(cutmask) {
  bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 3)

###Make Benzonase masks (this will take hours)
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../mm39_data/mm39.fa ../mm39_data/mm39_liver_Benzonase.bam --read-size=76 --strand-specific"
sqcmd = "seqOutBias"
prefix = "Benzonase_"

masks = mclapply(nextlst, function(cutmask) {
```

```
  bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 3)

###Make Cyanase masks (this will take hours)
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../mm39_data/mm39.fa ../mm39_data/mm39_liver_Cyanase.bam --read-size=76 --strand-specific"
sqcmd = "seqOutBias"
prefix = "Cyanase_"

masks = mclapply(nextlst, function(cutmask) {
  bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 3)

###Make MNase masks (this will take hours)
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../mm39_data/mm39.fa ../mm39_data/mm39_liver_MNase.bam --read-size=76 --strand-specific"
sqcmd = "seqOutBias"
prefix = "MNase_"

masks = mclapply(nextlst, function(cutmask) {
  bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 3)
```

2.13 Output k-mer counts tables for each 5-mer position

We now generate all k-mer counts for each 5-mer position using the .tbl files, data files and 'seqOutBias table' command

```
setwd('EnzymeBias_masks')
#Determine all Tn5 scale factors
for i in Tn5*.tbl
do
    name=$(echo $i | awk -F".tbl" '{print $1}')
    echo $name
    seqOutBias table $i ../hg38_data/C1_gDNA_rep1.bam > ${name}_scale_factors.txt
done

#Determine all DNase scale factors
for i in DNase*.tbl
do
    name=$(echo $i | awk -F".tbl" '{print $1}')
    echo $name
    seqOutBias table $i ../hg38_data/DNase_Naked.bam > ${name}_scale_factors.txt
done

#Determine all Benzonase scale factors
for i in Benzonase*.tbl
do
    name=$(echo $i | awk -F".tbl" '{print $1}')
    echo $name
    seqOutBias table $i ../mm39_data/mm39_liver_Benzonase.bam > ${name}_scale_factors.txt
done

#Determine all Cyanase scale factors
for i in Cyanase*.tbl
do
    name=$(echo $i | awk -F".tbl" '{print $1}')
    echo $name
    seqOutBias table $i ../mm39_data/mm39_liver_Cyanase.bam > ${name}_scale_factors.txt
done
```

```
#Determine all MNase scale factors
for i in MNase*.tbl
do
    name=$(echo $i | awk -F".tbl" '{print $1}')
    echo $name
    seqOutBias table $i ../mm39_data/mm39_liver_MNase.bam > ${name}_scale_factors.txt
done
```

2.14 Plotting scale factors for Enzyme masks

The k-mer counts tabulated with every mask are now plotted in box and whisker format in relation to the cut site. Scale factors are calculated using the scalefactor.func function.

```
library(lattice)
library(reshape2)
library(data.table)
source('Figure1_Functions.R')
setwd('../output')
#Load in scale factor files
Benzonase_factor_files = list.files('PATH/TO/mm39/scalefactors')
#Subset only the Benzonase files
Benzonase_factor_files = Benzonase_factor_files[grep('Benzonase', Benzonase_factor_files)]
Benzonase_factor_files = Benzonase_factor_files[grep('scale_factors', Benzonase_factor_files)]
Benzonase_scale_factors = vector(mode = 'list', length = length(Benzonase_factor_files))
#Determine scale factors for each k-mer position
for (i in 1:length(Benzonase_factor_files)) {
  Benzonase_scale_factors[[i]] = scalefactor.func(paste('PATH/TO/mm39/scalefactors/',
                                                        Benzonase_factor_files[i], sep = ''))
}
names(Benzonase_scale_factors) = gsub( '_scale_factors.txt', '',Benzonase_factor_files)
names(Benzonase_scale_factors) = gsub( 'Benzonase_', '',names(Benzonase_scale_factors))
#Turn these scale factors into a data.table with an identifier column
Benzonase_scale_factors_df = rbindlist(Benzonase_scale_factors, idcol = TRUE)
colnames(Benzonase_scale_factors_df)[1] = 'ID'
factor_key = data.frame(levels(factor(Benzonase_scale_factors_df$ID)))
factor_key[,2] = 1:nrow(factor_key)
colnames(factor_key) = c('mask', 'index')
Benzonase_scale_factors_df[, xaxis := factor_key[match(Benzonase_scale_factors_df$ID, factor_key$mask),2]-16.5]
#############################################################################################################
#Add benzonase to the list of enzyme scale factors for plotting
Figure1B_list = vector(mode = 'list', length = 1L)
Figure1B_list[[1]] = Benzonase_scale_factors_df[,c(10,14)]
colnames(Figure1B_list[[1]])[1] = 'minusscalefact'
Figure1B_list[[1]] = rbind(Figure1B_list[[1]], Benzonase_scale_factors_df[,c(13,14)])
names(Figure1B_list)[1] = 'Benzonase_scale_factors_df'
#############################################################################################################
#Load in scale factor files
Cyanase_factor_files = list.files('PATH/TO/mm39/scalefactors')
#Subset only the Cyanase files
Cyanase_factor_files = Cyanase_factor_files[grep('Cyanase', Cyanase_factor_files)]
Cyanase_factor_files = Cyanase_factor_files[grep('scale_factors', Cyanase_factor_files)]
Cyanase_scale_factors = vector(mode = 'list', length = length(Cyanase_factor_files))
#Determine scale factors for each k-mer position
for (i in 1:length(Cyanase_factor_files)) {
  Cyanase_scale_factors[[i]] = scalefactor.func(paste('PATH/TO/mm39/scalefactors/',
                                                      Cyanase_factor_files[i], sep = ''))
}
names(Cyanase_scale_factors) = gsub( '_scale_factors.txt', '',Cyanase_factor_files)
names(Cyanase_scale_factors) = gsub( 'Cyanase_', '',names(Cyanase_scale_factors))
#Turn these scale factors into a data.table with an identifier column
Cyanase_scale_factors_df = rbindlist(Cyanase_scale_factors, idcol = TRUE)
colnames(Cyanase_scale_factors_df)[1] = 'ID'
factor_key = data.frame(levels(factor(Cyanase_scale_factors_df$ID)))
```

```
factor_key[,2] = 1:nrow(factor_key)
colnames(factor_key) = c('mask', 'index')
Cyanase_scale_factors_df[, xaxis := factor_key[match(Cyanase_scale_factors_df$ID, factor_key$mask),2]-16.5]
#########################################################################################################
#Add Cyanase to the list of enzyme scale factors for plotting
Figure1B_list[[2]] = Cyanase_scale_factors_df[,c(10,14)]
colnames(Figure1B_list[[2]])[1] = 'minusscalefact'
Figure1B_list[[2]] = rbind(Figure1B_list[[2]], Cyanase_scale_factors_df[,c(13,14)])
names(Figure1B_list)[2] = 'Cyanase_scale_factors_df'
#########################################################################################################
#########################################################################################################
#Load in scale factor files
MNase_factor_files = list.files('PATH/TO/mm39/scalefactors')
#Subset only the MNase files
MNase_factor_files = MNase_factor_files[grep('MNase', MNase_factor_files)]
MNase_factor_files = MNase_factor_files[grep('scale_factors', MNase_factor_files)]
MNase_scale_factors = vector(mode = 'list', length = length(MNase_factor_files))
#Determine scale factors for each k-mer position
for (i in 1:length(MNase_factor_files)) {
  MNase_scale_factors[[i]] = scalefactor.func(paste('PATH/TO/mm39/scalefactors/',
                                                    MNase_factor_files[i], sep = ''))
}
names(MNase_scale_factors) = gsub( '_scale_factors.txt', '',MNase_factor_files)
names(MNase_scale_factors) = gsub( 'MNase_', '',names(MNase_scale_factors))
#Turn these scale factors into a data.table with an identifier column
MNase_scale_factors_df = rbindlist(MNase_scale_factors, idcol = TRUE)
colnames(MNase_scale_factors_df)[1] = 'ID'
factor_key = data.frame(levels(factor(MNase_scale_factors_df$ID)))
factor_key[,2] = 1:nrow(factor_key)
colnames(factor_key) = c('mask', 'index')
MNase_scale_factors_df[, xaxis := factor_key[match(MNase_scale_factors_df$ID, factor_key$mask),2]-16.5]
#########################################################################################################
#Add MNase to the list of enzyme scale factors for plotting
Figure1B_list[[3]] = MNase_scale_factors_df[,c(10,14)]
colnames(Figure1B_list[[3]])[1] = 'minusscalefact'
Figure1B_list[[3]] = rbind(Figure1B_list[[3]], MNase_scale_factors_df[,c(13,14)])
names(Figure1B_list)[3] = 'MNase_scale_factors_df'
#########################################################################################################
#########################################################################################################
#Load in scale factor files
DNase_factor_files = list.files('/PATH/TO/DNase_scale_factors')
#Subset only the DNase files
DNase_factor_files = DNase_factor_files[grep('DNase', DNase_factor_files)]
DNase_factor_files = DNase_factor_files[grep('scale_factors', DNase_factor_files)]
DNase_scale_factors = vector(mode = 'list', length = length(DNase_factor_files))
#Determine scale factors for each k-mer position
for (i in 1:length(DNase_factor_files)) {
  DNase_scale_factors[[i]] = scalefactor.func(paste('/PATH/TO/DNase_scale_factors/',
                                                    DNase_factor_files[i], sep = ''))
}
names(DNase_scale_factors) = gsub( '_scale_factors.txt', '',DNase_factor_files)
names(DNase_scale_factors) = gsub( 'DNase_', '',names(DNase_scale_factors))
#Turn these scale factors into a data.table with an identifier column
DNase_scale_factors_df = rbindlist(DNase_scale_factors, idcol = TRUE)
colnames(DNase_scale_factors_df)[1] = 'ID'
factor_key = data.frame(levels(factor(DNase_scale_factors_df$ID)))
factor_key[,2] = 1:nrow(factor_key)
colnames(factor_key) = c('mask', 'index')
DNase_scale_factors_df[, xaxis := factor_key[match(DNase_scale_factors_df$ID, factor_key$mask),2]-16.5]
#########################################################################################################
#Add DNase to the list of enzyme scale factors for plotting
Figure1B_list[[4]] = DNase_scale_factors_df[,c(10,14)]
colnames(Figure1B_list[[4]])[1] = 'minusscalefact'
Figure1B_list[[4]] = rbind(Figure1B_list[[4]], DNase_scale_factors_df[,c(13,14)])
```

```
names(Figure1B_list)[4] = 'DNase_scale_factors_df'
#####################################################################################################
#####################################################################################################
#Load in scale factor files
Tn5_factor_files = list.files('/PATH/TO/Scale_Factors/Tn5')
#Subset only the Tn5 files
Tn5_factor_files = Tn5_factor_files[grep('Tn5', Tn5_factor_files)]
Tn5_factor_files = Tn5_factor_files[grep('scale_factors', Tn5_factor_files)]
Tn5_factor_files = Tn5_factor_files[seq(3, length(Tn5_factor_files), 3)]
Tn5_factor_files = Tn5_factor_files[(grep('XXXXXXXXXXNNNNNXXXXXXXXCXXXXXXXXXXXXXXXXXXXXXXX', Tn5_factor_files)):
                            (grep('XXXXXXXXXXNNNNNXXXXXXXXCXXXXXXXXXXXXXXXXXXXXXXX', Tn5_factor_files)+30)]
Tn5_scale_factors = vector(mode = 'list', length = length(Tn5_factor_files))
#Determine scale factors for each k-mer position
for (i in 1:length(Tn5_factor_files)) {
  Tn5_scale_factors[[i]] = scalefactor.func(paste('/PATH/TO/Scale_Factors/Tn5/',
                                                  Tn5_factor_files[i], sep = ''))
}
names(Tn5_scale_factors) = gsub( '_scale_factors.txt', '',Tn5_factor_files)
names(Tn5_scale_factors) = gsub( 'Tn5_', '',names(Tn5_scale_factors))
#Turn these scale factors into a data.table with an identifier column
Tn5_scale_factors_df = rbindlist(Tn5_scale_factors, idcol = TRUE)
colnames(Tn5_scale_factors_df)[1] = 'ID'
factor_key = data.frame(levels(factor(Tn5_scale_factors_df$ID)))
factor_key[,2] = 1:nrow(factor_key)
colnames(factor_key) = c('mask', 'index')
Tn5_scale_factors_df[, xaxis := factor_key[match(Tn5_scale_factors_df$ID, factor_key$mask),2]-16]
#####################################################################################################
#Add Tn5 to the list of enzyme scale factors for plotting
Figure1B_list[[5]] = Tn5_scale_factors_df[,c(10,14)]
colnames(Figure1B_list[[5]])[1] = 'minusscalefact'
Figure1B_list[[5]] = rbind(Figure1B_list[[5]], Tn5_scale_factors_df[,c(13,14)])
names(Figure1B_list)[5] = 'Tn5_scale_factors_df'
#####################################################################################################
#Plot comparative nucleases
for (i in 1:(length(Figure1B_list)-1)) {
  pdf(paste(names(Figure1B_list)[i], '_maskpositions.pdf', sep = ''), width=23, height=7)
  par(mar=c(10,4,4,4))
  boxplot(log2((1/minusscalefact))~xaxis,data=Figure1B_list[[i]],
          las=1, pch = 16, outcex=0.33, pars  =  list(xaxt = "n"),
          ylab = '',
          xlab = '',
          whisklty = 1,
          whisklwd = 4,
          staplelwd = 4,
          ylim = c(-6,4),
          boxcol = 'light blue',
          col = 'light blue',
          frame.plot = FALSE,
          par(cex.axis=3))
  axis(1, at=seq(1, 32, by=1), labels = FALSE, tck = -0.05, lwd.ticks = 2)
  axis(2, lwd.ticks = 2, labels = FALSE)
  text(x = seq(1, 32, by=1)+0.25, par("usr")[3] - 0.75,
       labels = seq(-15.5, 15.5, by=1), srt = 90, pos = 2, xpd = TRUE, cex = 3)
  box(bty="l")
  dev.off()
}
#Plot Tn5 transposase
pdf('Tn5_scalefactors_maskpositions.pdf', width=23, height=7)
par(mar=c(10,4,4,4))
boxplot(log2((1/minusscalefact))~xaxis,data=Figure1B_list[[5]],
        las=1, pch = 16, outcex=0.33, pars  =  list(xaxt = "n"),
        ylab = '',
        xlab = '',
        whisklty = 1,
```

```
        whisklwd = 4,
        staplelwd = 4,
        ylim = c(-6,4),
        boxcol = 'light blue',
        col = 'light blue',
        frame.plot = FALSE,
        par(cex.axis=3))
axis(1, at=seq(1, 31, by=1), labels = FALSE, tck = -0.05, lwd.ticks = 2)
axis(2, lwd.ticks = 2, labels = FALSE)
text(x = seq(1, 31, by=1)+0.25, par("usr")[3] - 0.75,
     labels = seq(-15, 15, by=1), srt = 90, pos = 2, xpd = TRUE, cex = 3)
box(bty="l")
dev.off()
```

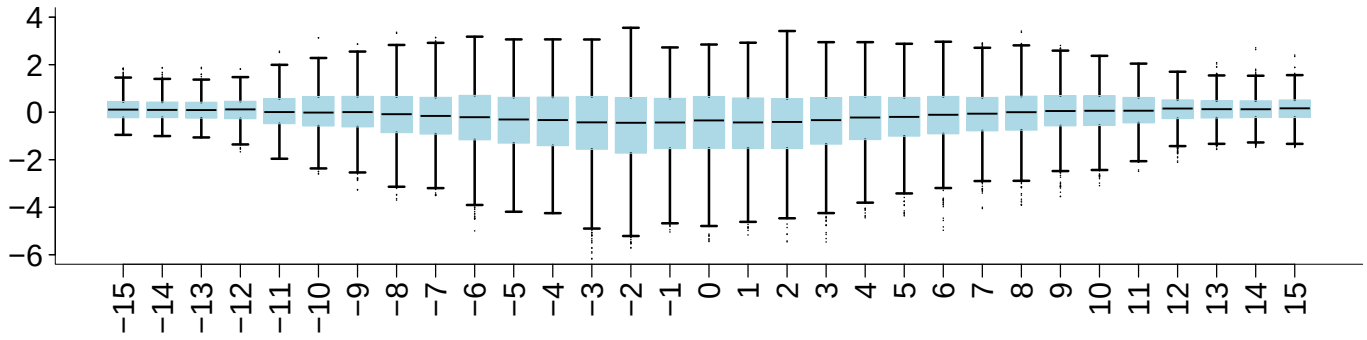


Figure 3: Transposase Scaling Factor Distributions Output

3 Figure 2. Use of scaling factors to correct Tn5 sequence bias is hindered by complex enzyme-DNA interactions.

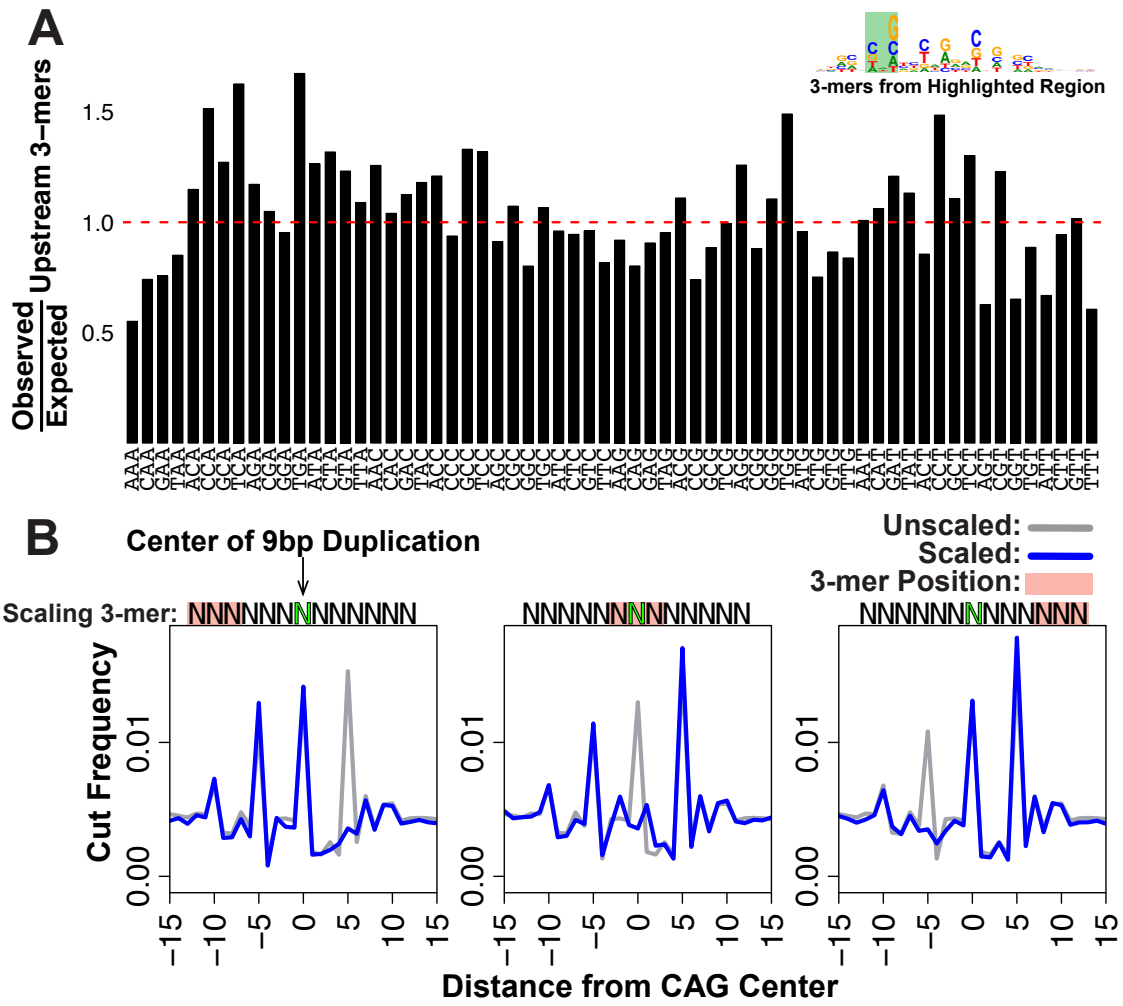


Figure 4: Use of direct k-mer scaling to correct Tn5 sequence bias is hindered by complex enzyme-DNA interactions. A) Barchart of observed divided by expected (from k-mer prevalence in the bias position probability matrix) k-mer frequency for specified 3-mers at a position -4 to -6bp upstream from the centrally recognized base. These values are corrected for background Tn5 3-mer frequency. B) Portions of the composite profile for the 3mer 'CAG' can be corrected by rationally designed masks, based on their position relative to the cutsite. Application of these corrections exacerbates the bias of nearby peaks within the composite.

3.1 Tn5 upstream observed 3-mers divided by sequence logo frequency expected 3-mers

This experiment illustrates that Tn5 sequence bias violates the assumption of positional independence in PSWM models. This is done by counting the number of each k-mer at positions -4 to -6 in the sequence data and dividing by the expected number of k-mers based on the PSWM frequencies. Before this, we determine background 3-mer observed divided by expected frequency and correct the observed values by multiplying them by the inverse of the background frequencies.

3.1.1 Background k-mer frequency in Tn5 data

First, we retrieve fasta sequence files for all of the sequences 50bp and 100bp upstream of our reads

```
#Make a fasta file of sequences 50bp upstream of all plus strand cutsites
awk '{$2 = $2 - 50; print}' ../data/C1_gDNA_rep1_plus.oneentry.bed | \
   awk '{OFS="\t";} {$3 = $2 + 3; print}' | grep -v - | \
```

```
   fastaFromBed -fi ../data/hg38.fa -s -bed stdin -fo C1_gDNA_rep1_50bp_plus.fasta

#Make a fasta file of sequences 50bp upstream of all minus strand cutsites
awk '{$2 = $2 - 50; print}' ../data/C1_gDNA_rep1_minus.oneentry.bed | \
   awk '{OFS="\t";} {$3 = $2 + 3; print}' |  grep -v - | \
   fastaFromBed -fi ../data/hg38.fa -s -bed stdin -fo C1_gDNA_rep1_50bp_minus.fasta

#Reverse complement all minus strand sequences
awk 'BEGIN{FS=" "}{if(!/>/){print toupper($0)}else{print $1}}' C1_gDNA_rep1_50bp_minus.fasta | \
  fastx_reverse_complement -o C1_gDNA_rep1_50bp_minus_RC.fasta

#Concatenate the plus and minus strand fasta files together
cat C1_gDNA_rep1_50bp_minus_RC.fasta C1_gDNA_rep1_50bp_plus.fasta > C1_gDNA_rep1_50bp_sepcat.fasta



#############################################################################################################
#Make a fasta file of sequences 100bp upstream of all plus strand cutsites
awk '{$2 = $2 - 100; print}' ../data/C1_gDNA_rep1_plus.oneentry.bed | \
   awk '{OFS="\t";} {$3 = $2 + 3; print}' | grep -v - | \
   fastaFromBed -fi ../data/hg38.fa -s -bed stdin -fo C1_gDNA_rep1_100bp_plus.fasta

#Make a fasta file of sequences 100bp upstream of all minus strand cutsites
awk '{$2 = $2 - 100; print}' ../data/C1_gDNA_rep1_minus.oneentry.bed | \
   awk '{OFS="\t";} {$3 = $2 + 3; print}' |  grep -v - | \
   fastaFromBed -fi ../data/hg38.fa -s -bed stdin -fo C1_gDNA_rep1_100bp_minus.fasta

#Reverse complement all minus strand sequences
awk 'BEGIN{FS=" "}{if(!/>/){print toupper($0)}else{print $1}}' C1_gDNA_rep1_100bp_minus.fasta | \
  fastx_reverse_complement -o C1_gDNA_rep1_100bp_minus_RC.fasta

#Concatenate the plus and minus strand fasta files together
cat C1_gDNA_rep1_100bp_minus_RC.fasta C1_gDNA_rep1_100bp_plus.fasta > C1_gDNA_rep1_100bp_sepcat.fasta
```

Next, we determine if background 50bp and 100bp k-mer frequencies are similar. First, we must determine nucleotide frequency for both in order to calculate the expected frequency, then we count k-mers in the desired positions and divide one by the other.

```
system('mkdir kmercount_coherence')
setwd('kmercount_coherence')
source('https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_functions.R')
source('../Tn5_Bias_Functions.R')
library(ggseqlogo)
options(scipen = 100)
#Convert all sequences to uppercase
uppercasenames <- list('C1_gDNA_rep1_100bp_sepcat_44cs.fasta', 'C1_gDNA_rep1_50bp_sepcat_44cs.fasta')
uplist <- lapply(uppercasenames, uppercase)
Tn5_100bp_sepcat = as.data.frame(uplist[1])
Tn5_50bp_sepcat = as.data.frame(uplist[2])
#Determine pswm of each:
Tn5_100bp_sepcat.pswm = pswm.func.2(Tn5_100bp_sepcat[,1], 'Tn5_100bp_bias', 3, FALSE)
Tn5_50bp_sepcat.pswm = pswm.func.2(Tn5_50bp_sepcat[,1], 'Tn5_50bp_bias', 3, FALSE)
#Make df of positions 50-47bp upstream (-50:-47 from cutsite)
Tn5_pos100_98 = data.table(substr(uplist[[1]][,1], 1, 3))
Tn5_pos100_98 = Tn5_pos100_98$V1
Tn5_pos50_48 = data.table(substr(uplist[[2]][,1], 1, 3))
Tn5_pos50_48 = Tn5_pos50_48$V1
#Make all possible 3mers:
mertable = expand.grid(rep(list(c('A','C','G','T')), 3))
mertable = data.frame(apply(mertable, 1 , paste, collapse = ""))
#Count number of each 3mer in each position:
for (i in 1:length(mertable[,1])) {
  mertable[i,2] = length(grep(mertable[i,1], Tn5_pos100_98))
  mertable[i,3] = length(grep(mertable[i,1], Tn5_pos50_48))
```

```
}
#Calculate percent frequency of each k-mer for each position
mertable[,4] = mertable[,2]/sum(mertable[,2])
mertable[,5] = mertable[,3]/sum(mertable[,3])
#Change column names
colnames(mertable) = c('kmer', 'count100bp', 'count50bp', 'percent100bp', 'percent50bp')
#Calculate expected k-mer frequency for 100bp, based on the PSWM created above
pswm_100bp_Tn5 = read.table('Tn5_100bp_bias.txt', skip = 9)
colnames(pswm_100bp_Tn5) = c('A', 'C', 'G', 'T')
###Calculate expected upstream values based on nucleotide prevalence:
expected_100bp_nuc_prevalence = expand.grid(rep(list(c('A','C','G','T')), 3))
for (i in 1:nrow(expected_100bp_nuc_prevalence)) {
  expected_100bp_nuc_prevalence[i,4] = pswm_100bp_Tn5[1 ,which(colnames(pswm_100bp_Tn5) == expected_100bp_nuc_prevalence[
  expected_100bp_nuc_prevalence[i,5] = pswm_100bp_Tn5[2 ,which(colnames(pswm_100bp_Tn5) == expected_100bp_nuc_prevalence[
  expected_100bp_nuc_prevalence[i,6] = pswm_100bp_Tn5[3 ,which(colnames(pswm_100bp_Tn5) == expected_100bp_nuc_prevalence[
}
expected_100bp_nuc_prevalence[,7] = expected_100bp_nuc_prevalence[,4] * expected_100bp_nuc_prevalence[,5] * expected_100b
#calculate 100bp observed / expected:
mertable$OE100bp = mertable$percent100bp / expected_100bp_nuc_prevalence$V7
#Repeat for 50bp upstream
#Calculate expected k-mer frequency for 50bp, based on the PSWM created above
pswm_50bp_Tn5 = read.table('Tn5_50bp_bias.txt', skip = 9)
colnames(pswm_50bp_Tn5) = c('A', 'C', 'G', 'T')
###Calculate expected upstream values based on nucleotide prevalence:
expected_50bp_nuc_prevalence = expand.grid(rep(list(c('A','C','G','T')), 3))
for (i in 1:nrow(expected_50bp_nuc_prevalence)) {
  expected_50bp_nuc_prevalence[i,4] = pswm_50bp_Tn5[1 ,which(colnames(pswm_50bp_Tn5) == expected_50bp_nuc_prevalence[i,1]
  expected_50bp_nuc_prevalence[i,5] = pswm_50bp_Tn5[2 ,which(colnames(pswm_50bp_Tn5) == expected_50bp_nuc_prevalence[i,2]
  expected_50bp_nuc_prevalence[i,6] = pswm_50bp_Tn5[3 ,which(colnames(pswm_50bp_Tn5) == expected_50bp_nuc_prevalence[i,3]
}
expected_50bp_nuc_prevalence[,7] = expected_50bp_nuc_prevalence[,4] * expected_50bp_nuc_prevalence[,5] * expected_50bp_nu
#calculate 50bp observed / expected:
mertable$OE50bp = mertable$percent50bp / expected_50bp_nuc_prevalence$V7
#Calculate 1/50bp * 100bp Observed/Expected values
mertable$Inv50OE100bp = mertable$OE100bp*(1/mertable$OE50bp)
#Plot this comparison
pdf(file = 'Tn5_100bpCorrectedby50bp_kmers.pdf', width=12, height=9)
ggplot(data=mertable, aes(x=kmer, y=Inv50OE100bp)) +
  geom_bar(stat="identity", fill = 'Black') +
  ylab("Ratio observed / expected") + xlab("") + theme_classic() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, size = 16, colour = 'black'),
        axis.text.y = element_text(size = 16, color = 'black'),
        axis.title.y = element_text(size = 24)) +
  geom_hline(yintercept=1, linetype="dashed",
             color = "red", size=1)
dev.off()
#Save the mertable df to use as a correction for observed/expected k-mer frequencies
Tn5_kmer_count_correction = mertable
save(Tn5_kmer_count_correction, file='Tn5_observedexpected_kmercounts.Rdata')
```

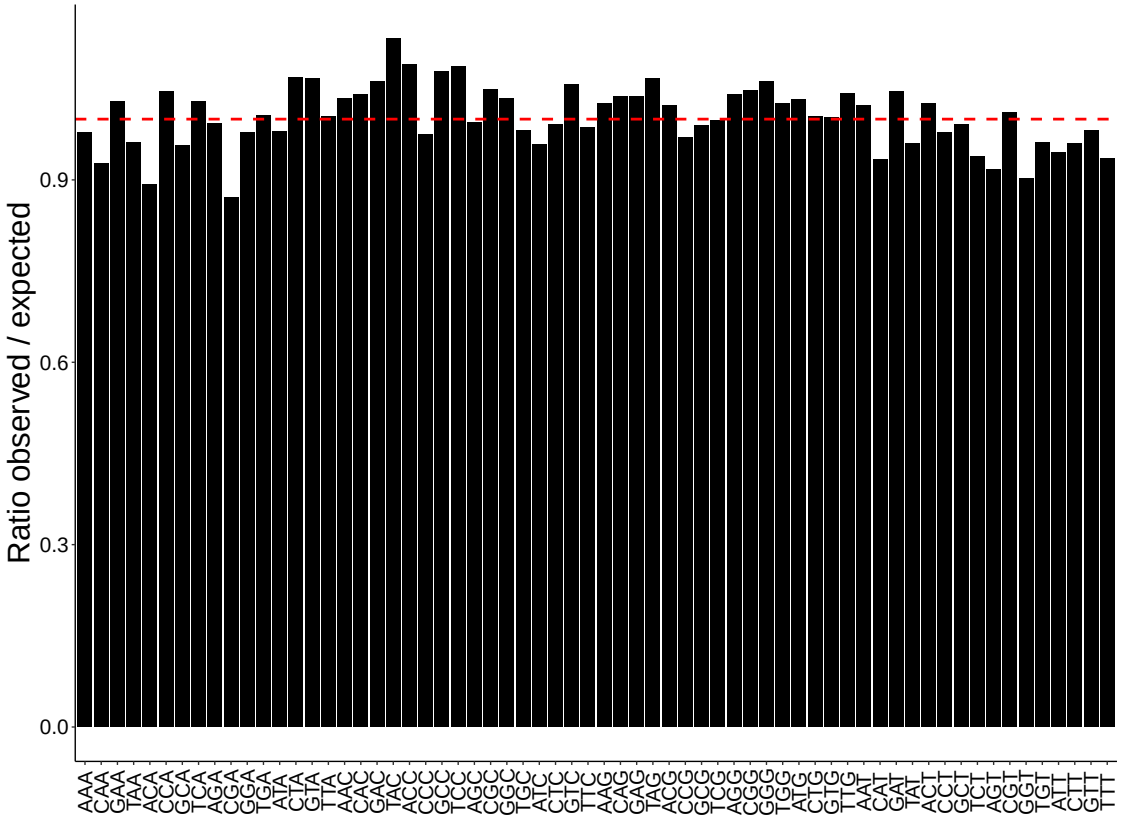


Figure 5: 50bp and 100bp upstream k-mer observed divided by expected frequencies are very similar

3.1.2 Upstream k-mer Observed divided by expected frequency

We can now calculate the upstream divided by expected k-mer frequency at positions -6:-4, and correct these values by the 50bp upstream k-mer observed/expected values

```
source('../Tn5_Bias_Functions.R')
source('https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_functions.R')
library(stringr)
library(data.table)
library(lattice)
options(scipen=10000)

setwd('../output')

#
#Input filenames (from wd) into uppercasenames in this order: pe1.minus, pe1.plus, pe2.minus, pe2.plus
uppercasenames <- list('../data/C1_gDNA_rep1_plus.fasta', '../data/C1_gDNA_rep1_minus_RC.fasta')
uplist <- lapply(uppercasenames, uppercase)
plus_tn5_44cs = as.data.frame(uplist[1])
save(plus_tn5_44cs, file = 'Tn5_plus_seqs.Rdata')
minus_tn5_44cs = as.data.frame(uplist[2])
save(minus_tn5_44cs, file = 'Tn5_minus_seqs.Rdata')
rm(uplist, uppercasenames)
#
#############################################################################################################
#Make df of positions 10-12 (-6:-4 from cutsite)
plus_Tn5_pos10_12 = data.table(str_split_fixed(plus_tn5_44cs[,1], '', 13))
plus_Tn5_pos10_12 = plus_Tn5_pos10_12[,10:12]
plus_Tn5_pos10_12[, kmer := paste0(V10, V11, V12)]
plus_Tn5_pos10_12 = plus_Tn5_pos10_12$kmer
rm(plus_tn5_44cs)
```

```
#Make df of positions 10-12 (-6:-4 from cutsite)
minus_Tn5_pos10_12 = data.table(str_split_fixed(minus_tn5_44cs[,1], '', 13))
minus_Tn5_pos10_12 = minus_Tn5_pos10_12[,10:12]
minus_Tn5_pos10_12[, kmer := paste0(V10, V11, V12)]
minus_Tn5_pos10_12 = minus_Tn5_pos10_12$kmer
rm(minus_tn5_44cs)

#Make all possible 3mers:
mertable = expand.grid(rep(list(c('A','C','G','T')), 3))
mertable = data.frame(apply(mertable, 1 , paste, collapse = ""))

#Count number of each 3mer in each position:
for (i in 1:length(mertable[,1])) {
  mertable[i,2] = length(grep(mertable[i,1], plus_Tn5_pos10_12))
  mertable[i,3] = length(grep(mertable[i,1], minus_Tn5_pos10_12))
}

colnames(mertable) = c('kmer', 'upPlus', 'upMinus')

save(mertable, file = 'mertable_kmer_counts.Rdata')
load('mertable_kmer_counts.Rdata')

mertable[,6] = mertable[,2] + mertable[,3]
mertable[,7] = mertable[,4] + mertable[,5]
colnames(mertable)[6:7] = c('upPercent', 'downPercent')
mertable = mertable[,-c(2:5)]
mertable[,2] = mertable[,2]/sum(mertable[,2])
mertable[,3] = mertable[,3]/sum(mertable[,3])
##Load in tn5 bias transfac
load('../ref/tn5_sepcat_bias_transfac.Rdata')
colnames(transfac.tn5) = c('A', 'C', 'G', 'T')

###Calculate expected upstream values based on nucleotide prevalence:
expected_upstream_nuc_prevalence = expand.grid(rep(list(c('A','C','G','T')), 3))

for (i in 1:nrow(expected_upstream_nuc_prevalence)) {
  expected_upstream_nuc_prevalence[i,4] =
    transfac.tn5[10 ,which(colnames(transfac.tn5) == expected_upstream_nuc_prevalence[i,1])]
  expected_upstream_nuc_prevalence[i,5] =
    transfac.tn5[11 ,which(colnames(transfac.tn5) == expected_upstream_nuc_prevalence[i,2])]
  expected_upstream_nuc_prevalence[i,6] =
    transfac.tn5[12 ,which(colnames(transfac.tn5) == expected_upstream_nuc_prevalence[i,3])]
}

#Multiply these together
expected_upstream_nuc_prevalence[,7] = expected_upstream_nuc_prevalence[,4] *
                                       expected_upstream_nuc_prevalence[,5] *
                                       expected_upstream_nuc_prevalence[,6]
#This should = 1:
sum(expected_upstream_nuc_prevalence[,7])
#calculate upstream observed / expected:
mertable$upOE = mertable$upPercent / expected_upstream_nuc_prevalence$V7
#Correct the observed divided by expected k-mer counts by the 50bp upstream counts
load('../kmercount_coherence/Tn5_observedexpected_kmercounts.Rdata')
mertable$CorrupOE = mertable$upOE*Tn5_kmer_count_correction$(1/mertable$OE50bp)
#Plot corrected upstream observed/expected k-mer counts
pdf(file = 'Figure2A_Tn5_upstream_OE_kmers.pdf', width=20, height=5)
barchart(CorrupOE~kmer, mertable,
         ylab = list(expression(bold(paste(frac(Observed, Expected), " 3-mers" ))), cex = 3),
         scales=list(x=list(cex=2.2, rot = 90, fontfamily = "mono"), y=list(cex=2.5)),
         ylim = c(0,1.8),
         col = 'black',
         par.settings = list(axis.line = list(col = 0)),
```

```
        panel=function(...) {
          panel.barchart(...)
          panel.abline(h=1, col = 'red', lwd = 4, lty = 2)
        })
dev.off()
```

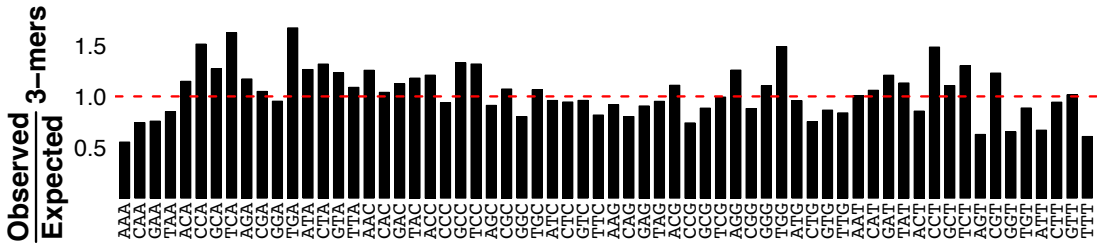


Figure 6: Figure 2A. Tn5 upstream observed divided by expected 3-mers

3.2 Generating 400,000 random CAG locations in the genome and plotting signal at these intervals

First use seqOutBias seqtable command to generate a .tbl file of all 3-mers in hg38. Next, use seqOutBias dump command to make list of all 3-mer locations in the hg38 genome. Then use the dump_to_kmer_bed.py script to output the locations of all CAGs in the genome, in bed format. Finally, we take random 400,000 instances of CAG for plotting.

```
mkdir seqdump
cd seqdump

#This will make a .tbl file of all 3-mers in the genome
seqOutBias seqtable ../hg38.fa --kmer-size=3 --plus-offset=3 --minus-offset=3 --read-size=76 --out=hg38.3.3.3.tbl
#This will dump all indexed 3-mer locations to a text file. This file will be LARGE
seqOutBias dump hg38.3.3.3.tbl > hg38.3.3.3.dump.txt
#This will take the plus and minus index for CAG and output a bed file for each
python dump_to_kmer_bed.py -i hg38.3.3.3.dump.txt -p 19 -m 31
#Combine the plus and minus bed files
cat hg38.3.3.3plus.19.bed hg38.3.3.3minus.31.bed > CAG_locations.bed
#Subset 400,000 random locations from combined bed files
grep -v '_' CAG_locations.bed | perl -MList::Util=shuffle -e 'print shuffle(<STDIN>);' |
                                head -400000 > CAG_locations_rand400k.bed

#Run seqOutBias with masks designed to correct each peak (mask is shifted 4 base pairs):
seqOutBias hg38.fa C1_gDNA_rep1_plus.bam --custom-shift=4,-4 --strand-specific --kmer-mask=NNCN \
                                         --bed=C1_gDNA_rep1_plus_NNNXXXC.bed \
                                         --bw=C1_gDNA_rep1_plus_NNNXXXC.bigWig --read-size=76
seqOutBias hg38.fa C1_gDNA_rep1_minus.bam --custom-shift=4,-4 --strand-specific --kmer-mask=NNCN \
                                          --bed=C1_gDNA_rep1_minus_NNNXXXC.bed \
                                          --bw=C1_gDNA_rep1_minus_NNNXXXC.bigWig --read-size=76

seqOutBias hg38.fa C1_gDNA_rep1_plus.bam --custom-shift=4,-4 --strand-specific --kmer-mask=CXXXNNN \
                                         --bed=C1_gDNA_rep1_plus_NCNN.bed \
                                         --bw=C1_gDNA_rep1_plus_NCNN.bigWig --read-size=76
seqOutBias hg38.fa C1_gDNA_rep1_minus.bam --custom-shift=4,-4 --strand-specific --kmer-mask=CXXXNNN \
                                          --bed=C1_gDNA_rep1_minus_NCNN.bed \
                                          --bw=C1_gDNA_rep1_minus_NCNN.bigWig --read-size=76

seqOutBias hg38.fa C1_gDNA_rep1_plus.bam --custom-shift=4,-4 --strand-specific --kmer-mask=CXXXXXXXXNNN \
                                         --bed=C1_gDNA_rep1_plus_CXXXXNNN.bed \
                                         --bw=C1_gDNA_rep1_plus_CXXXXNNN.bigWig --read-size=76
seqOutBias hg38.fa C1_gDNA_rep1_minus.bam --custom-shift=4,-4 --strand-specific --kmer-mask=CXXXXXXXXNNN \
                                          --bed=C1_gDNA_rep1_minus_CXXXXNNN.bed \
                                          --bw=C1_gDNA_rep1_minus_CXXXXNNN.bigWig --read-size=76
```

3.2.1 Plot CAG peak direction composites

```
library(bigWig)
library(lattice)
library(data.table)
source('../scripts/Figure1_Functions.R')
setwd('../seqdump')


####CAG peak direction
#Load in random 400,000 CAG instances
CAG_BED <- fread(input = '/PATH/TO/CAG_locations_rand400k.bed')
colnames(CAG_BED) <- c('chr', 'start', 'end', 'gene', 'location', 'strand')

#Plot the unscaled signal at the CAG instances
CAG_unscaled_composite <- BED.query.bigWig(CAG_BED,
                      bwPlus = '../data/tn5_C1_gDNA_rep1_plus.bigWig',
                      bwMinus = '../data/tn5_C1_gDNA_rep1_minus.bigWig',
                      upstream = 15, downstream = 15, group = 'Unscaled', factor = '')

#Plot the NNCN scaled signal at the CAG instances
CAG_NNCN_composite <- BED.query.bigWig(CAG_BED,
                      bwPlus =  '../data/C1_gDNA_rep1_plus_NNCN.bigWig',
                      bwMinus =  '../data/C1_gDNA_rep1_minus_NNCN.bigWig',
                      upstream = 15, downstream = 15, group = 'NNCN', factor = '')
CAG_NNCN_composite <- rbind(CAG_NNCN_composite, CAG_unscaled_composite)
CAG_NNCN_composite$factor <- c('NNCN')


#Plot the CXXXNNN scaled signal at the CAG instances
CAG_CXXXNNN_composite <- BED.query.bigWig(CAG_BED,
                       bwPlus =  '../data/C1_gDNA_rep1_plus_CXXXNNN.bigWig',
                       bwMinus =  '../data/C1_gDNA_rep1_minus_CXXXNNN.bigWig',
                       upstream = 15, downstream = 15, group = 'CXXXNNN', factor = '')
CAG_CXXXNNN_composite <- rbind(CAG_CXXXNNN_composite, CAG_unscaled_composite)
CAG_CXXXNNN_composite$factor <- c('CXXXNNN')

#Plot the CXXXXXXXXNNN scaled signal at the CAG instances
CAG_CXXXXXXXXNNN_composite <- BED.query.bigWig(CAG_BED,
                        bwPlus =  '../data/C1_gDNA_rep1_plus_CXXXXXXXXNNN.bigWig',
                        bwMinus =  '../data/C1_gDNA_rep1_minus_CXXXXXXXXNNN.bigWig',
                        upstream = 15, downstream = 15, group = 'CXXXXXXXXNNN', factor = '')
CAG_CXXXXXXXXNNN_composite <- rbind(CAG_CXXXXXXXXNNN_composite, CAG_unscaled_composite)
CAG_CXXXXXXXXNNN_composite$factor <- c('CXXXXXXXXNNN')
CAG_fig_composite <- rbind(CAG_NNCN_composite,
                           CAG_CXXXNNN_composite, CAG_CXXXXXXXXNNN_composite)
#Plot random CAG composites with mask corrections
plot.composites <- function(dat, ylabel = '', pdf_name = 'PLEASE_SET_FILE_NAME',
                            xlabel = '', striplabel = TRUE, legend = TRUE,
                            motifline = FALSE, Motiflen = 10,
                            figwidth = 2.5, figheight=3,
                            indexlist = NULL, layoutgrid = NULL,
                            col.lines = c("#0000FF", "#0000FF", "#0000FF", "#a1a3ab"),
                            fill.poly = c(rgb(0,0,1,1/4),
                                          rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),
                                          rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4))) {
  require(lattice)
  pdf(paste(pdf_name, '.pdf', sep = ''), width= figwidth, height= figheight)
  print(xyplot(est ~ x|factor, group = group, data = dat, strip = striplabel,
               type = 'l', as.table = TRUE,
               scales=list(x=list(at=seq(-15,15,5), cex=1.4, relation = "free", axs ="i", rot = 90),
                           y =list(cex=1.4, relation="free", tick.number=2)),
               col = col.lines,
```

```
                auto.key = if (legend == TRUE)
                {list(points=F, lines=T, cex=0.8)} else{},
                par.settings = list(strip.background=list(col="#00000000"),
                                    strip.border = list(col = 'transparent'),
                                    superpose.symbol = list(pch = c(16),
                                                            col=col.lines,
                                                            cex =0.5),
                                    superpose.line = list(col = col.lines,
                                                          lwd=c(2),
                                                          lty = c(1))),
                cex.axis=1.4,
                par.strip.text=list(cex=1.2, font=1, col='black'),
                aspect=1.0,
                between=list(y=0.5, x=0.5),
                lwd=3,
                ylab = list(label = paste(ylabel), cex =1.4),
                xlab = list(label = paste(xlabel), cex =1.4),
                xlim = c(-15,15),
                ylim = c(0,0.0175),
                index.cond = indexlist,
                layout = layoutgrid,
                panel = function(x, y, ...) {
                  panel.xyplot(x, y, ...)
                  #panel.abline(h = 0, lty =1, lwd = 1.0, col = '#A9A9A932')
                  level = dimnames(trellis.last.object())[["factor"]][packet.number()]
                  if (motifline == TRUE)
                  {panel.abline(v = Motiflen[rownames(Motiflen)==level,]/2, lty = 2, col = "red")} else{}
                  if (motifline == TRUE)
                  {panel.abline(v = -Motiflen[rownames(Motiflen)==level,]/2, lty = 2, col = "red")} else{}
                }
  ))
  dev.off()
}

plot.composites(CAG_fig_composite, legend = FALSE,
                pdf_name = 'Figure2B_CAG_direction_maskcompare',
                ylabel = 'Insertion Frequency',
                xlabel = 'Distance from CAG Center',
                indexlist = list(c(3,2,1)),
                layoutgrid = c(3,1),
                figwidth = 8, figheight=5,
                motifline = FALSE)
```

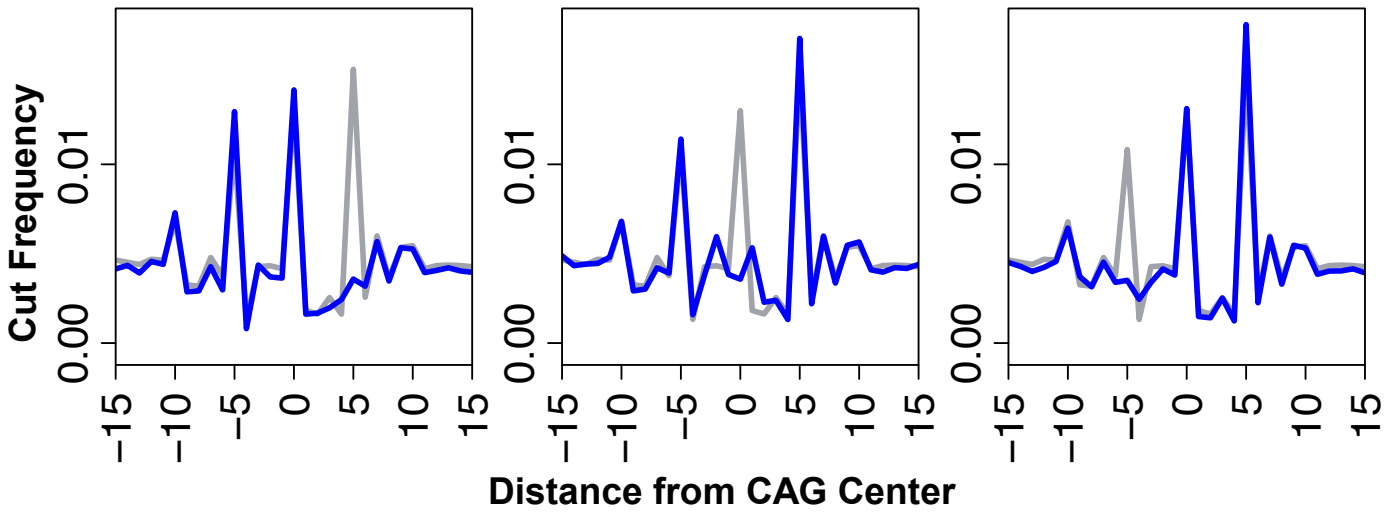


Figure 7: Figure2B CAG direction

4 Figure 4. Hierarchical clustering of transcription factor motifs based on information content and GC%

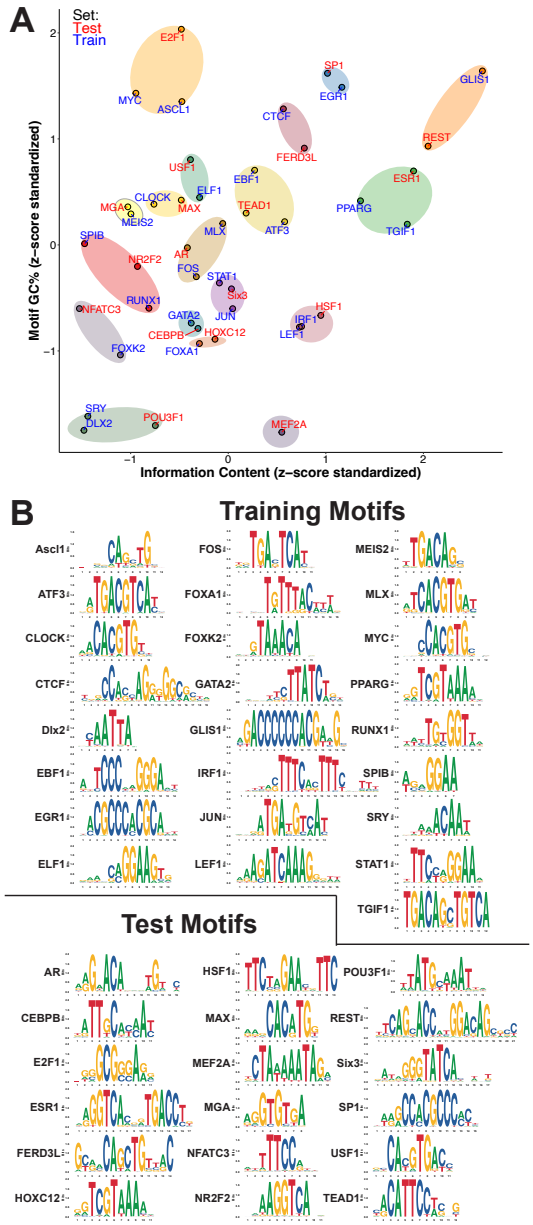


Figure 8: Figure 4. Unbiased hierarchical clustering of transcription factor motifs creates diverse test and training sets. A) transcription factor motifs are clustered into test and training sets based on motif information content and GC%. Points are colored according to cluster and set assignment is indicated by label name color. Cluster number and grouping is indicated by same-colored ovals. B) Training and test transcription factor motifs used for the rule ensemble.

4.1 Download transcription factor motifs from JASPAR

```
mkdir ref_FIMO
cd ref_FIMO
#wget transfac from JASPAR
#Chose TFs with ENCODE listed under 'source' field when multiple versions listed
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0007.2.meme > AR.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA1100.1.meme > ASCL1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0605.2.meme > ATF3.meme
```

```
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0466.1.meme > CEBPB.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0819.1.meme > CLOCK.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0139.1.meme > CTCF.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0885.1.meme > DLX2.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0468.1.meme > DUX4.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0024.2.meme > E2F1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0154.3.meme > EBF1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0162.3.meme > EGR1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0473.1.meme > ELF1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0112.3.meme > ESR1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA1485.1.meme > FERD3L.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0476.1.meme > FOS.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0148.3.meme > FOXA1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA1103.1.meme > FOXK2.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0036.2.meme > GATA2.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0735.1.meme > GLIS1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0906.1.meme > HOXC12.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0486.2.meme > HSF1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0050.2.meme > IRF1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0488.1.meme > JUN.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0768.1.meme > LEF1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0058.2.meme > MAX.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0052.3.meme > MEF2A.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0774.1.meme > MEIS2.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0801.1.meme > MGA.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0663.1.meme > MLX.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0147.3.meme > MYC.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0625.1.meme > NFATC3.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA1111.1.meme > NR2F2.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0786.1.meme > POU3F1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0906.1.meme > PPARG.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0138.2.meme > REST.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0002.1.meme > RUNX1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0631.1.meme > Six3.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0079.4.meme > SP1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0081.1.meme > SPIB.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0084.1.meme > SRY.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0137.3.meme > STAT1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0090.1.meme > TEAD1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0796.1.meme > TGIF1.meme
wget  -cO - http://jaspar.genereg.net/api/v1/matrix/MA0093.2.meme > USF1.meme
```

4.2 Implement size-constrained, hierchical clustering to make groups of TFs based on IC and GC content then plot TFs along with cluster/group designation

This clustering was done including the transcription factor DUX4. We later excluded DUX4 due to overlap with repetitive regions and odd baseline k-mer frequencies. This is why the variable for DUX4 is included in the clustering approach, then immediately removed from the data set. We manually made test/training determinations for each cluster in order to preserve an even distribution in both groups. Finally, the circles colored according to cluster number were added in a post-processing step.

```
library(data.table)
library(ggplot2)
library(seqLogo)
library(scclust)
library(RColorBrewer)
library(ggrepel)
library(ggnewscale)
################################################################################
setwd('../ref_FIMO')
train = c( 'ASCL1', 'ATF3', 'CLOCK', 'CTCF',
           'DLX2', 'EBF1', 'EGR1', 'ELF1',
           'FOS', 'FOXA1', 'FOXK2', 'GATA2',
```

```
           'GLIS1', 'IRF1', 'JUN', 'LEF1',
           'MEIS2', 'MLX', 'MYC', 'PPARG',
           'RUNX1', 'SPIB', 'SRY', 'STAT1', 'TGIF1')
test = c('AR', 'CEBPB', 'E2F1', 'ESR1',
         'FERD3L', 'HOXC12', 'HSF1', 'MAX',
         'MEF2A', 'MGA', 'NFATC3', 'NR2F2',
         'POU3F1', 'REST', 'Six3', 'SP1',
         'USF1', 'TEAD1')


#Determine GC and IC of each transcription factor
Memes <- list.files('./')
Memes = Memes[grep('.meme', Memes)]

GCvec = NULL
ICvec = NULL
for(i in 1:length(Memes)) {
  memefile <- paste0("./",Memes[i])
  minmeme <- read.csv(memefile, skip = 11, header = FALSE, sep = '')
  minmeme <- minmeme[1:nrow(minmeme)-1,]
  minmeme[,2] <- as.numeric(minmeme[,2])
  minmeme[,1] <- as.numeric(minmeme[,1])
  GCcon <- sum(minmeme[,c(2,3)])/sum(minmeme)
  GCvec[i] <- GCcon
  minmeme = t(minmeme)
  rownames(minmeme) = c('A', 'C', 'G', 'T')
  pwmMeme = makePWM(minmeme)
  ICcon = sum(ic(pwmMeme))
  ICvec[i] = ICcon
}

#Combine IC and GC vectors and add TF name column
GCbyIC = as.data.frame(cbind(GCvec, ICvec))
GCbyIC[,3] = substr(Memes, 1, nchar(Memes)- 5)
#Add column names
colnames(GCbyIC) = c('GC', 'IC', 'TF')
#z-standardize GC values
GCmean = mean(GCbyIC$GC)
GCstddv = sd(GCbyIC$GC)
GCbyICstd = as.data.frame((GCbyIC$GC-GCmean)/GCstddv)
#z-standardize IC values
ICmean = mean(GCbyIC$IC)
ICstddv = sd(GCbyIC$IC)
GCbyICstd[,2] = (GCbyIC$IC-ICmean)/ICstddv
GCbyICstd[,3] = GCbyIC$TF
#Add column names
colnames(GCbyICstd) = c('zstdGC', 'zstdIC', 'TF')

#Comput euclidean distances between points
GCbyIC_dist = distances(GCbyICstd,
                        id_variable = "TF",
                        dist_variables = c("zstdGC", "zstdIC"))
#Implement size-constrained clustering
set.seed(42)
GCbyIC_scclust = sc_clustering(GCbyIC_dist, 2)
#Refine clustering using hierarchical clustering
GCbyIC_clust = hierarchical_clustering(GCbyIC_dist, 2, batch_assign = TRUE,
                          existing_clustering = GCbyIC_scclust)
#Add clusters to dataframe
GCbyICstd[,4] = GCbyIC_clust
colnames(GCbyICstd)[4] = 'Cluster'
GCbyICstd[,4] = as.character(GCbyICstd[,4])
#Remove DUX4
GCbyICstd = GCbyICstd[-c(8),]
```

```
#Assign test and training labels to each TF
GCbyICstd$group = ''
for (i in 1:nrow(GCbyICstd)) {
  if (GCbyICstd[i,3] %in% train) {
    GCbyICstd[i,5] = 'Train'
  } else {
    GCbyICstd[i,5] = 'Test'
  }
}
#Assign colors to test and training sets and make a color pallette for clusters
GCbyICstd[,6] = 'red'
GCbyICstd[which(GCbyICstd[,5] == 'Train'),6] = 'blue'
colnames(GCbyICstd)[6] = 'TestTrain'
colnames(GCbyICstd)[5] = 'Set'
mycolors = colorRampPalette(brewer.pal(8, "Set1"))(22)

#Plot clustered transcription factor motifs
GCbyICstd_kmeans_plot = ggplot(data = GCbyICstd,
                               mapping = aes(x = zstdIC,
                                             y = zstdGC,
                                             color = Cluster, label = TF)) + geom_point(size = 4) +
  theme_classic() +
  xlab('Information Content (z-score standardized)') +
  ylab('Motif GC% (z-score standardized)') +
  theme(axis.title.x = element_text(family = 'sans', face = 'bold', size = 22)) +
  theme(axis.title.y = element_text(family = 'sans', face = 'bold', size = 22)) +
  theme(axis.text=element_text(size=18, color = 'black')) +
  theme(legend.text=element_text(size=18, color = 'black'), legend.title=element_text(size=18)) +
  geom_point(shape = 1,size = 4,colour = "black") +
  scale_color_manual(values = mycolors) +
  new_scale_color() + geom_point(aes(x = zstdIC, y = zstdGC, color = Set), alpha = 0) +
  scale_color_manual(values = c('red', 'blue')) +
  geom_text_repel(aes(label = TF),
                  size = 6, max.overlaps = 11, color = GCbyICstd$TestTrain,
                  force = 10, force_pull = 0.75, seed = 43L, min.segment.length = 0) +
  guides(colour = guide_legend(override.aes = list(alpha = 1))) + coord_fixed(ratio = 1.086468)

pdf("ICbyMotifGC_TestTrain_clust.pdf", width=12, height=12)
GCbyICstd_kmeans_plot
dev.off()
```

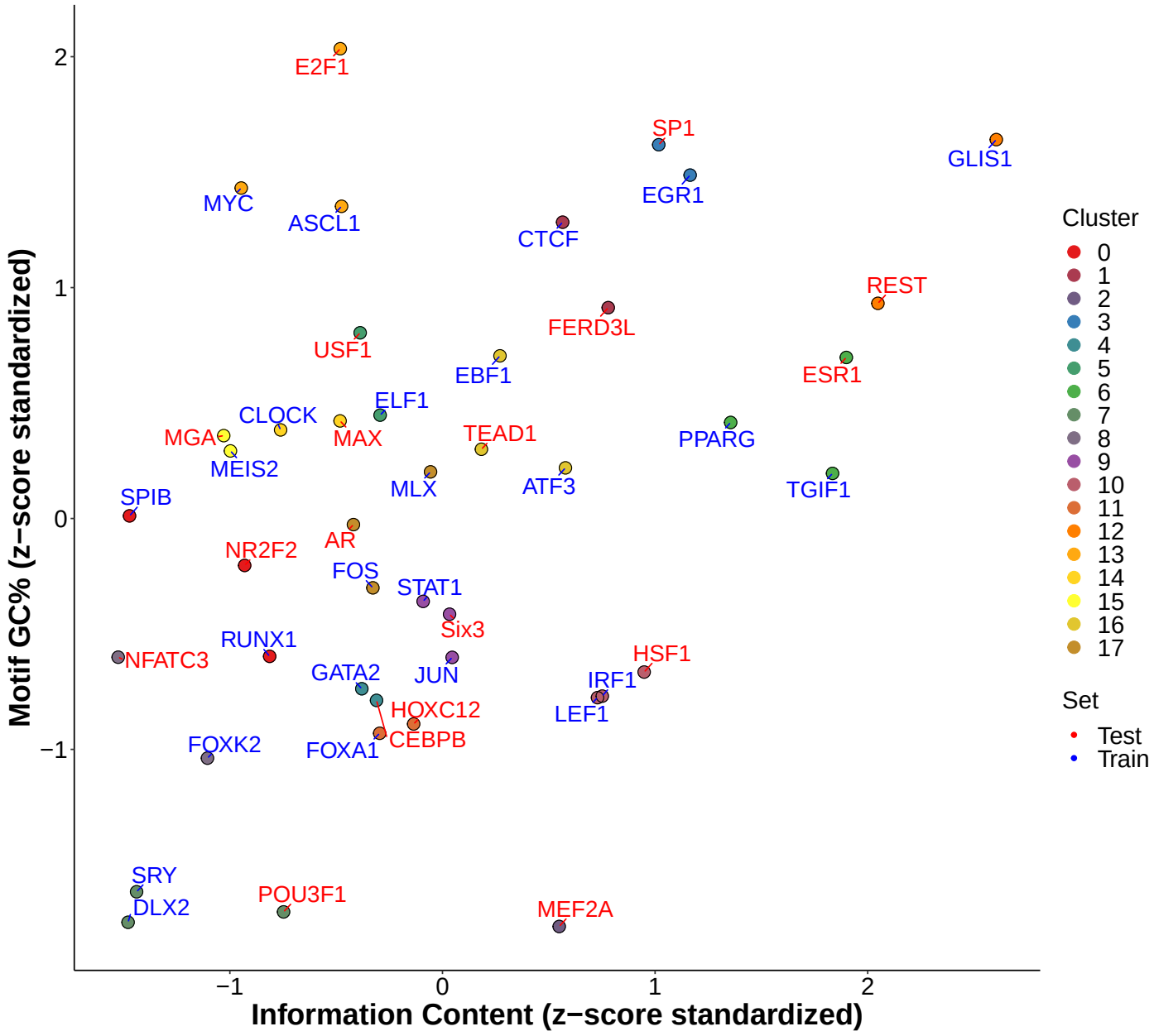


Figure 9: Figure 4A Plot

5 Figure 5. Application of a prediction rules ensemble to DNase data can slightly enhance k-mer scaling

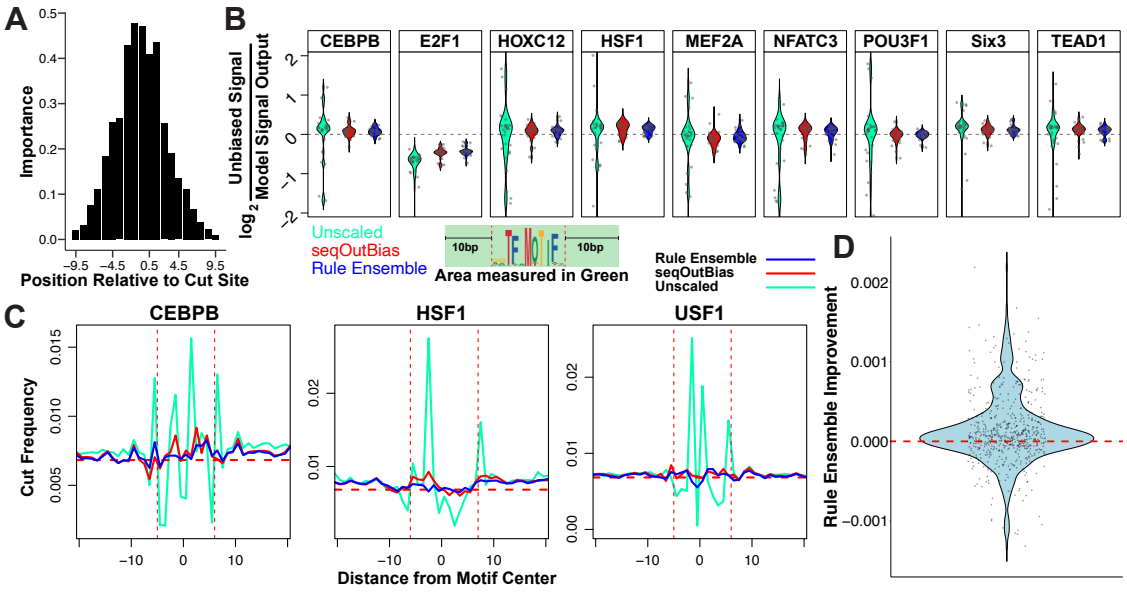


Figure 10: Figure 3. Application of a prediction rules ensemble to DNase data can slightly enhance kmer scaling.

5.1 Generate scaling factors for DNase

These scaling factors based on mask positions within 30bp of the cutsite are generated in Figure1.

5.2 Use the FIMO algorithm to find 400,000 locations of the test and training motifs in the genome.

```
cd ../Figure4/ref_FIMO

####FIMO transcription factor motifs using hg38
for meme in *.meme
do
    name=$(echo $meme | awk -F".meme" '{print $1}')
    echo $name
    fimo --thresh 0.001 --text ${name}.meme ../data/hg38.fa > ${name}_fimo.txt

done


##wget repeatmasker coordinates
wget  https://www.repeatmasker.org/genomes/hg38/RepeatMasker-rm405-db20140131/hg38.fa.out.gz
gunzip -c hg38.fa.out.gz > hg38_repeatmasker.fa
awk  '{ OFS="\t" } {print $5, $6, $7, $11}' hg38_repeatmasker.fa | tail +4 > hg38_repeatmasker.bed
rm hg38_repeatmasker.fa

###Remove repeatmasker regions from all FIMO files
for file in ../Figure4/ref_FIMO/*_fimo.txt
do
  TF=$(echo $file | awk -F"_fimo.txt" '{print $1}' | awk -F"../ref_FIMO/" '{print $2}')
  echo ${TF}
  awk  '{ OFS="\t" } {print $3, $4, $5, $2, $1, $6, $7, $8, $9}' $file | tail +2 > ${TF}_FIMO.bed
```

```
  bedtools intersect -wa -v \
  -a ${TF}_FIMO.bed \
  -b ../data/hg38_repeatmasker.bed \
   > ${TF}.removed.bed
  echo ${TF}.removed.bed
  awk  '{ OFS="\t" } {print $5, $4, $1, $2, $3, $6, $7, $8, "", $9}' ${TF}.removed.bed > ${TF}_nohead_FIMO.bed
  head -n 1 ../ref_FIMO/AR_fimo.txt | cat - ${TF}_nohead_FIMO.bed > ${TF}_rm_FIMO.txt

  rm ${TF}_FIMO.bed ${TF}.removed.bed ${TF}_nohead_FIMO.bed
done
```

5.3 Separate 400,000 plus/minus motifs from the original FIMO file to use for rule ensemble modeling

```
library(data.table)
system('mkdir ref_FIMO')
setwd('../Figure4/ref_FIMO')

FIMO_files = list.files('../Figure4/ref_FIMO/')
FIMO_files = FIMO_files[grep('rm_FIMO.txt')]
#Make 400k plus/minus strand reads
for (z in 1:length(FIMO_files)) {

#Load in FIMO file
FIMO = fread(paste('../ref_FIMO/', FIMO_files[z], sep = ''),
             sep = 'auto', skip = 0, header = TRUE, fill = FALSE)

#Remove alternative and mitochondrial chromosomes
FIMO = FIMO[-c(grep('_', FIMO$sequence_name))]
FIMO = FIMO[-c(grep('chrM', FIMO$sequence_name))]
print(unique(FIMO$sequence_name))

FIMO_minus = FIMO[FIMO$strand=='-']
FIMO_minus = FIMO_minus[order(-score)][1:400000]
write.table(FIMO_minus, file = paste(substr(FIMO_files[z], 1, nchar(FIMO_files[z])-9),
                                     '_minus_400k_fimo.txt', sep = ''),
                                     quote=FALSE, sep = '\t', row.names=FALSE, na = "")

FIMO_plus = FIMO[FIMO$strand=='+']
FIMO_plus = FIMO_plus[order(-score)][1:400000]
write.table(FIMO_plus, file = paste(substr(FIMO_files[z], 1, nchar(FIMO_files[z])-9),
                                     '_plus_400k_fimo.txt', sep = ''),
                                     quote=FALSE, sep = '\t', row.names=FALSE, na = "")
}

system('cp *_plus_400k_fimo.txt ../../Figure5/ref_FIMO')
system('cp *_minus_400k_fimo.txt ../../Figure5/ref_FIMO')
```

5.4 Plot unscaled DNase composite values for FIMO motifs (values which rule ensemble predicts)

```
library(bigWig)
library(zoo)
library(lattice)
library(data.table)
library(matrixStats)
library(gridExtra)
source('../Tn5_Bias_Functions.R')
```

```
system('mkdir DNase_unscaled_composites')
setwd('DNase_unscaled_composites')
########################### For only plus unscaled composites
#Load in plus coordinates for each TF Motif
Motifs <- list.files('../ref_FIMO')
Motifs = Motifs[grep('plus', Motifs)]
Motiflist <- vector('list', length(Motifs))
for (i in 1:length(Motifs)) {
  Motiflist[[i]] <- FIMO.to.BED(paste('../ref_FIMO/',Motifs[i], sep = ''))
}
names(Motiflist) = substr(Motifs, 1, nchar(Motifs)-14)
Motifs = names(Motiflist)
#Determine signal at each plus motif
BWs <- c('/PATH/TO/DNase_Naked_plus_unscaled.bigWig')
DNase_plus_unscaled_compositelist <- vector('list', length(Motifs))
for (i in 1:length(Motiflist)) {
  DNase_plus_unscaled_compositelist[[i]] = BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                      upstream = 100, downstream = 100, factor = Motifs[i],
                                      group = 'Unscaled', ATAC = FALSE)
}
names(DNase_plus_unscaled_compositelist) <- Motifs

#plot uncorrected values
for (i in 1:length(DNase_plus_unscaled_compositelist)) {
  plot.composites(DNase_plus_unscaled_compositelist[[i]], legend = FALSE,
                  pdf_name = paste('DNase_plus_unscaled_400k_',
                                   paste(names(DNase_plus_unscaled_compositelist[i])), '_composite', sep = ''),
                  ylabel = 'Cut Frequency',
                  xlabel = 'Distance from Motif Center',
                  motifline = FALSE, Motiflen = Motiflen[i]
  )}
#Save values
save(DNase_plus_unscaled_compositelist, file = 'DNase_plus_unscaled_compositelist.Rdata')

########################### For only minus unscaled composites
#Load in minus coordinates for each TF Motif
Motifs <- list.files('../ref_FIMO')
Motifs = Motifs[grep('minus', Motifs)]
Motiflist <- vector('list', length(Motifs))
for (i in 1:length(Motifs)) {
  Motiflist[[i]] <- FIMO.to.BED(paste('../ref_FIMO/',Motifs[i], sep = ''))
}
names(Motiflist) = substr(Motifs, 1, nchar(Motifs)-14)
Motifs = names(Motiflist)
#Determine signal at each minus motif
BWs <- c('/PATH/TO/DNase_Naked_minus_unscaled.bigWig')
DNase_minus_unscaled_compositelist <- vector('list', length(Motifs))
for (i in 1:length(Motiflist)) {
  DNase_minus_unscaled_compositelist[[i]] = BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                                             upstream = 100, downstream = 100, factor = Motifs[i],
                                                             group = 'Unscaled', ATAC = FALSE)
}
names(DNase_minus_unscaled_compositelist) <- Motifs

#plot uncorrected values
for (i in 1:length(DNase_minus_unscaled_compositelist)) {
  plot.composites(DNase_minus_unscaled_compositelist[[i]], legend = FALSE,
                  pdf_name = paste('DNase_minus_unscaled_400k_',
                                   paste(names(DNase_minus_unscaled_compositelist[i])), '_composite', sep = ''),
                  ylabel = 'Cut Frequency',
                  xlabel = 'Distance from Motif Center',
                  motifline = FALSE, Motiflen = Motiflen[i]
  )}
#Save values
```

```
save(DNase_minus_unscaled_compositelist, file = 'DNase_minus_unscaled_compositelist.Rdata')
```

5.5 Determine 5-mer frequency for rule ensemble modeling and multiply this by inverse scale factors for each mask

Before a rule ensemble model can be generated, k-mer frequency of the predicted motif must be calculated. In this section we take all of the sequences from our motifs of interest and digest them into 5bp chunks. Next, we tally the number of each 5-mer found at each position in our composites. Once we have this data, we then multiply each k-mer frequency at a position by its inverse scale factor as input for each unscaled composite trace. The output is then summed for each position and finally shifted to align with unscaled DNase traces as input for rule ensemble training.

```
library(data.table)
source('../Tn5_Bias_Functions.R')
library('bigWig')
system('mkdir DNase_RE_output')
setwd('DNase_RE_output')
options(scipen = 100)
train = c( 'ASCL1', 'ATF3', 'CLOCK', 'CTCF',
           'DLX2', 'EBF1', 'EGR1', 'ELF1',
           'FOS', 'FOXA1', 'FOXK2', 'GATA2',
           'GLIS1', 'IRF1', 'JUN', 'LEF1',
           'MEIS2', 'MLX', 'MYC', 'PPARG',
           'RUNX1', 'SPIB', 'SRY', 'STAT1', 'TGIF1')
#Import kmer counts and compute scale factors for each
factorfiles = list.files('../../Figure1/EnzymeBias_masks')
factorfiles = factorfiles[c(grep('DNase', factorfiles))]
factorfiles = factorfiles[c(grep('_scale_factors.txt', factorfiles))]
scalefactors = vector(mode = "list", length = length(factorfiles))
names(scalefactors) = factorfiles

for (i in 1:length(factorfiles)) {
  scalefactors[[i]] <- scalefactor.func(paste('../../Figure1/EnzymeBias_masks/', factorfiles[i], sep = ''))
}

#Import TF FIMO coordinates to extract the sequences from the FASTA
FIMOfiles = list.files('../ref_FIMO')
FIMOfiles = FIMOfiles[grep('plus', FIMOfiles)]
FIMOfiles = FIMOfiles[which(substr(FIMOfiles,1,nchar(FIMOfiles)-22) %in% train)]
TFmotifs = vector(mode = "list", length = length(FIMOfiles))
names(TFmotifs) = substr(FIMOfiles, 1, (nchar(FIMOfiles)-14))

#Import FIMO files for genomic coordinates and expand a 'window' around these sites (using a central base)
for (i in 1:length(TFmotifs)) {
  TFmotifs[[i]] <- FIMO.to.BED(paste('../ref_FIMO/', FIMOfiles[i], sep = ''), window = 100)
}

#Ensure that no start positions are off the chromosome
for (i in 1:length(TFmotifs)) {
  print(TFmotifs[[i]][which.min(TFmotifs[[i]]$start),])
}

#Write these coordinates into a bed file for use with 'getfasta' from bedtools
for (i in 1:length(TFmotifs)) {
  write.table(TFmotifs[[i]], file = paste(names(TFmotifs[i]), '_hg38coords.bed', sep = ''),
              row.names=FALSE, sep="\t", quote = FALSE, col.names = FALSE)
}

#Run getfasta on genomic coordinates to get sequences for window
for (i in 1:length(TFmotifs)) {
  system(paste('bedtools getfasta -bedOut -s -fi /PATH/TO/hg38.fa -bed ',
               names(TFmotifs[i]), '_hg38coords.bed > ',
               names(TFmotifs[i]), '_hg38seq.bed', sep = ''))
```

```
}
#load in bed files with sequences
TFseq_plus <- vector(mode = "list", length = length(TFmotifs))
names(TFseq_plus) <- names(TFmotifs)
for (i in 1:length(TFmotifs)) {
  TFseq_plus[[i]] <- fread(paste(names(TFmotifs[i]), '_hg38seq.bed', sep = ''))
}
#Make sure you still have 400k rows
for (i in 1:length(TFseq_plus)) {
  print(nrow(TFseq_plus[[i]]))
}
#Make all sequences upper case
TFseq_plus = lapply(TFseq_plus, function(x) {toupper(x$V7)})
#Break sequences into 5bp chunks (for 5mer)
TFseq_plus = lapply(TFseq_plus, mer.positions, mermask = "NNNNN")
#Find frequency of each possible 5mer for each position
TFseq_plus = lapply(TFseq_plus, position.frequencies, mermask = 'NNNNN')
#Multiply 1/scale_factor by its corresponding kmer frequency for each TF motif. This will produce an
# .rds file for each TF/mask combo (scale factor * k-mer frequency - sfkf)
TF_plus_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_plus))
names(TF_plus_scalefactors_kmerfreq) <- names(TFseq_plus)
for (i in 1:length(TF_plus_scalefactors_kmerfreq)) {
  TF_plus_scalefactors_kmerfreq[[i]] <- scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                        kmerfrequency = TFseq_plus[[i]],
                                        tfname = names(TF_plus_scalefactors_kmerfreq)[i])
}

#Next we load in each sfkf combo to the same list
TF_sfkf_5mer_plus <- vector(mode = "list", length = length(TFseq_plus))
names(TF_sfkf_5mer_plus) = names(TFseq_plus)


RDSlist = list.files('./')
RDSlist = RDSlist[grep('plus', RDSlist)]
RDSlist = RDSlist[grep('_.rds', RDSlist)]
for (i in 1:length(RDSlist)) {
  TF_sfkf_5mer_plus[[which(names(TF_sfkf_5mer_plus) ==
                    substr(RDSlist[i], 1, nchar(RDSlist[i]) - 49))]][[grep(substr(RDSlist[i],
                    nchar(RDSlist[i]) - 41, nchar(RDSlist[i])- 4),
                    factorfiles)]] = readRDS(paste( './', RDSlist[i], sep = ''))
  names(TF_sfkf_5mer_plus[[which(names(TF_sfkf_5mer_plus) ==
                    substr(RDSlist[i], 1, nchar(RDSlist[i]) - 49))]])[grep(substr(RDSlist[i],
                    nchar(RDSlist[i]) - 41, nchar(RDSlist[i])- 4),
                    factorfiles)] =  substr(RDSlist[i], 1, nchar(RDSlist[i]) - 4)
  TF_sfkf_5mer_plus[[which(names(TF_sfkf_5mer_plus) ==
                    substr(RDSlist[i], 1, nchar(RDSlist[i]) - 49))]][[grep(substr(RDSlist[i],
                    nchar(RDSlist[i]) - 41, nchar(RDSlist[i])- 4),
                    factorfiles)]] = scaledkmerfreq.sum.plus(TF_sfkf_5mer_plus[[which(names(TF_sfkf_5mer_plus)
                                                                                      == substr(RDSlist[i], 1,
                    nchar(RDSlist[i]) - 49))]][[grep(substr(RDSlist[i], nchar(RDSlist[i]) - 41,
                                                     nchar(RDSlist[i])- 4), factorfiles)]])
}
Plus_pre_input = TF_sfkf_5mer_plus

################################################################################
#Load in unscaled composites
load('../DNase_unscaled_composites/DNase_plus_unscaled_compositelist.Rdata')
DNase_unscaled_composites = do.call(rbind, DNase_plus_unscaled_compositelist)
#Trim unscaled composites to the size of predicted area
DNase_unscaled_composites = DNase_unscaled_composites[-c(which(DNase_unscaled_composites$x < -69.5 |
                                                               DNase_unscaled_composites$x > 82.5)),]
DNase_unscaled_composites$est = DNase_unscaled_composites$est/mean(DNase_unscaled_composites$correction)
#Shift pre predictor variables. Each masks' position shifts by the mask's offset
input_correction = data.frame(matrix(nrow = 32, ncol = 2))
```

```
input_correction[1:32,1] = 14:45
input_correction[1:32,2] = 166:197

Plus_pred_input = vector(mode = 'list', length = length(Plus_pre_input))
names(Plus_pred_input) <- names(Plus_pre_input)
for (i in 1:length(Plus_pre_input)) {
  for (p in 1:length(Plus_pre_input[[i]])) {
    Plus_pred_input[[i]][[p]] = Plus_pre_input[[i]][[p]][input_correction[p,1]:input_correction[p,2]]
  }
  Plus_pred_input[[i]] = data.frame(Plus_pred_input[[i]])
}

#Assign names
for (i in 1:length(Plus_pred_input)) {
  names(Plus_pred_input[[i]]) = names(Plus_pre_input[[i]])
}
#Turn plus_pred_input into DF
Plus_pred_input = cbind(seq(-69.5,82.5,1), DNase_unscaled_composites$est, do.call(rbind, Plus_pred_input))
colnames(Plus_pred_input)[1:2] = c('X', 'unscaled')
colnames(Plus_pred_input)[3:ncol(Plus_pred_input)] =
  substr(colnames(Plus_pred_input)[3:ncol(Plus_pred_input)], 21,
         nchar(colnames(Plus_pred_input)[3:ncol(Plus_pred_input)])-1)
#This is the plus input for the rule ensemble
save(Plus_pred_input, file = 'DNase_TF_plus_pre_input.Rdata')

################################################################################
#Repeat for Minus strand motifs
#Import TF FIMO coordinates to extract the sequences from the FASTA
FIMOfiles = list.files('../ref_FIMO')
FIMOfiles = FIMOfiles[grep('minus', FIMOfiles)]
FIMOfiles = FIMOfiles[which(substr(FIMOfiles,1,nchar(FIMOfiles)-23) %in% train)]
TFmotifs = vector(mode = "list", length = length(FIMOfiles))
names(TFmotifs) = substr(FIMOfiles, 1, (nchar(FIMOfiles)-14))

#Import FIMO files for genomic coordinates and expand a 'window' around these sites (using a central base)
for (i in 1:length(TFmotifs)) {
  TFmotifs[[i]] <- FIMO.to.BED(paste('../ref_FIMO/', FIMOfiles[i], sep = ''), window = 100)
}

#Ensure that no start positions are off the chromosome
for (i in 1:length(TFmotifs)) {
  print(TFmotifs[[i]][which.min(TFmotifs[[i]]$start),])
}

#Write these coordinates into a bed file for use with 'getfasta' from bedtools
for (i in 1:length(TFmotifs)) {
  write.table(TFmotifs[[i]], file = paste(names(TFmotifs[i]), '_hg38coords.bed', sep = ''),
              row.names=FALSE, sep="\t", quote = FALSE, col.names = FALSE)
}

#Run getfasta on genomic coordinates to get sequences for window
for (i in 1:length(TFmotifs)) {
  system(paste('bedtools getfasta -bedOut -s -fi /PATH/TO/hg38.fa -bed ',
               names(TFmotifs[i]), '_hg38coords.bed > ',
               names(TFmotifs[i]), '_hg38seq.bed', sep = ''))
}
#load in bed files with sequences
TFseq_minus <- vector(mode = "list", length = length(TFmotifs))
names(TFseq_minus) <- names(TFmotifs)
for (i in 1:length(TFmotifs)) {
  TFseq_minus[[i]] <- fread(paste(names(TFmotifs[i]), '_hg38seq.bed', sep = ''))
}
#Make sure you still have 400k rows
for (i in 1:length(TFseq_minus)) {
```

```
  print(nrow(TFseq_minus[[i]]))
}
#Make all sequences upper case
TFseq_minus = lapply(TFseq_minus, function(x) {toupper(x$V7)})
#Break sequences into 5bp chunks (for 5mer)
TFseq_minus = lapply(TFseq_minus, mer.positions, mermask = "NNNNN")
#Find frequency of each possible 5mer for each position
TFseq_minus = lapply(TFseq_minus, position.frequencies, mermask = 'NNNNN')
#Multiply 1/scale_factor by its corresponding kmer frequency for each TF motif. This will produce an
# .rds file for each TF/mask combo (scale factor * k-mer frequency - sfkf)
TF_minus_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_minus))
names(TF_minus_scalefactors_kmerfreq) <- names(TFseq_minus)
for (i in 1:length(TF_minus_scalefactors_kmerfreq)) {
  TF_minus_scalefactors_kmerfreq[[i]] <- scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                         kmerfrequency = TFseq_minus[[i]],
                                         tfname = names(TF_minus_scalefactors_kmerfreq)[i])
}

#Next we load in each sfkf combo to the same list
TF_sfkf_5mer_minus <- vector(mode = "list", length = length(TFseq_minus))
names(TF_sfkf_5mer_minus) = names(TFseq_minus)


RDSlist = list.files('./')
RDSlist = RDSlist[grep('minus', RDSlist)]
RDSlist = RDSlist[grep('_.rds', RDSlist)]
for (i in 1:length(RDSlist)) {
  TF_sfkf_5mer_minus[[which(names(TF_sfkf_5mer_minus) ==
                substr(RDSlist[i], 1, nchar(RDSlist[i]) - 49))]][[grep(substr(RDSlist[i],
                nchar(RDSlist[i]) - 41, nchar(RDSlist[i])- 5),
                factorfiles)]] = readRDS(paste( './', RDSlist[i], sep = ''))
  names(TF_sfkf_5mer_minus[[which(names(TF_sfkf_5mer_minus) ==
                substr(RDSlist[i], 1, nchar(RDSlist[i]) - 49))]])[grep(substr(RDSlist[i],
                nchar(RDSlist[i]) - 41, nchar(RDSlist[i])- 5),
                factorfiles)] =  substr(RDSlist[i], 1, nchar(RDSlist[i]) - 5)
  TF_sfkf_5mer_minus[[which(names(TF_sfkf_5mer_minus) ==
                substr(RDSlist[i], 1, nchar(RDSlist[i]) - 49))]][[grep(substr(RDSlist[i],
                nchar(RDSlist[i]) - 41, nchar(RDSlist[i])- 5),
                factorfiles)]] = scaledkmerfreq.sum.minus(TF_sfkf_5mer_minus[[which(names(TF_sfkf_5mer_minus)
                == substr(RDSlist[i], 1,
                nchar(RDSlist[i]) - 49))]][[grep(substr(RDSlist[i], nchar(RDSlist[i]) - 41,
                nchar(RDSlist[i])- 4), factorfiles)]])
}
minus_pre_input = TF_sfkf_5mer_minus

################################################################################
#Load in unscaled composites
load('../DNase_unscaled_composites/DNase_minus_unscaled_compositelist.Rdata')
DNase_unscaled_composites = do.call(rbind, DNase_minus_unscaled_compositelist)
#Trim unscaled composites to the size of predicted area
DNase_unscaled_composites = DNase_unscaled_composites[-c(which(DNase_unscaled_composites$x < -69.5 |
                                                               DNase_unscaled_composites$x > 82.5)),]
DNase_unscaled_composites$est = DNase_unscaled_composites$est/mean(DNase_unscaled_composites$correction)
#Shift pre predictor variables. Each masks' position shifts by the mask's offset
input_correction = data.frame(matrix(nrow = 32, ncol = 2))
input_correction[1:32,1] = 14:45
input_correction[1:32,2] = 166:197

minus_pred_input = vector(mode = 'list', length = length(minus_pre_input))
names(minus_pred_input) <- names(minus_pre_input)
for (i in 1:length(minus_pre_input)) {
  for (p in 1:length(minus_pre_input[[i]])) {
    minus_pred_input[[i]][[p]] = minus_pre_input[[i]][[p]][input_correction[p,1]:input_correction[p,2]]
  }
```

```
  minus_pred_input[[i]] = data.frame(minus_pred_input[[i]])
}

#Assign names
for (i in 1:length(minus_pred_input)) {
  names(minus_pred_input[[i]]) = names(minus_pre_input[[i]])
}
#Turn minus_pred_input into DF
minus_pred_input = cbind(seq(-69.5,82.5,1), DNase_unscaled_composites$est, do.call(rbind, minus_pred_input))
colnames(minus_pred_input)[1:2] = c('X', 'unscaled')
colnames(minus_pred_input)[3:ncol(minus_pred_input)] =
  substr(colnames(minus_pred_input)[3:ncol(minus_pred_input)], 22,
         nchar(colnames(minus_pred_input)[3:ncol(minus_pred_input)])-1)
#This is the minus input for the rule ensemble
save(minus_pred_input, file = 'DNase_TF_minus_pre_input.Rdata')
```

5.6 Train a rule ensemble model using the prepared DNase data from above.

```
library(data.table)
source('../scripts/Figure3_functions.R')
library('pre')
library("caret")
library('bigWig')
setwd('../DNase_RE_output')
options(scipen = 100)

#These are the training and test TF motif sets
train = c( 'ASCL1', 'ATF3', 'CLOCK', 'CTCF', 'DLX2', 'EBF1',
           'EGR1', 'ELF1', 'FOS', 'FOXA1', 'FOXK2', 'GATA2',
           'GLIS1', 'IRF1', 'JUN', 'LEF1', 'MEIS2', 'MLX',
           'MYC', 'PPARG','RUNX1', 'SPIB', 'SRY', 'STAT1', 'TGIF1')

test = c('AR', 'CEBPB', 'E2F1', 'ESR1', 'FERD3L', 'HOXC12',
         'HSF1', 'MAX', 'MEF2A', 'MGA', 'NFATC3', 'NR2F2',
         'POU3F1', 'REST', 'Six3', 'SP1', 'USF1', 'TEAD1')

load('/path/to/DNase_TF_pre_input.Rdata')
#load in unscaled composites and reduce the size to the predicted area
#(last 4 bases are unmappable because of 5mer)
load('/path/to/DNase_unscaled_compositelist.Rdata')

DNase_unscaled_composites = do.call(rbind, DNase_unscaled_compositelist)

#Subset test data
Plus_pred_test = Plus_pred_input[which(names(Plus_pred_input) %in% test)]
DNase_pre_test = do.call(rbind, Plus_pred_test)
Plus_pred_input = Plus_pred_input[which(names(Plus_pred_input) %in% train)]


DNase_unscaled_composites = DNase_unscaled_composites[-c(which(DNase_unscaled_composites$x < -77.5 |
                                                               DNase_unscaled_composites$x > 78.5)),]
DNase_unscaled_composites$factor = substr(DNase_unscaled_composites$factor, 1,
                                          nchar(DNase_unscaled_composites$factor)-8)
DNase_unscaled_composites_test = DNase_unscaled_composites[which(DNase_unscaled_composites$factor %in% test),]
DNase_unscaled_composites = DNase_unscaled_composites[which(DNase_unscaled_composites$factor %in% train),]

DNase_pre_data <- cbind(DNase_unscaled_composites, do.call(rbind, Plus_pred_input))

#Remove X, factor, group, correction columns
DNase_pre_data = data.frame(DNase_pre_data[,-c(1,3,4,5)])
```

```
################################################################################
#Train rule ensemble model using DNase input
set.seed(42)
DNase_pre_model <- pre(est ~  ., data = DNase_pre_data, type = 'both', sampfrac = 0.5,
                       verbose = TRUE, ntrees = 1000,
                       use.grad = TRUE, nfolds = 20, winsfrac = 0, maxdepth = 3L,
                       learnrate = 0.1, intercept = FALSE)
save(DNase_pre_model, file = 'DNase_PRE_model.Rdata')
################################################################################
#Write table for rule ensemble model coefficients/rules
print(DNase_pre_model, penalty.par.val = DNase_pre_model$glmnet.fit$lambda.min)
DNase_pre_model_coef <- coef(DNase_pre_model, penalty.par.val = DNase_pre_model$glmnet.fit$lambda.min)
DNase_pre_model_coef = DNase_pre_model_coef[which(DNase_pre_model_coef$coefficient != 0),]
write.table(DNase_pre_model_coef[,2:3], file = 'DNasePN_pre_coeff.txt', quote = FALSE, sep = ',',
            row.names = FALSE)
#


#Plot variable importances for PRE model:
imps = importance(DNase_pre_model, penalty.par.val = DNase_pre_model$glmnet.fit$lambda.min,
                  abbreviate = FALSE)
base_imps = imps$varimps
base_imps$varname = as.factor(base_imps$varname)
redundant_base_imps = redundantpos(base_imps, labelnum = 1, impnum = 2)
#Plot redundant_var_imps
pdf(file = 'Redundant_DNasePN_RE_variable_importances.pdf', width = 8, height = 10)
impbarplot <-ggplot(data=redundant_base_imps, aes(x=position, y=possum))
impbarplot +
  labs(x= 'Position', y= 'Importance') +
  geom_bar(stat="identity", fill="black") +
  theme_classic() +
  theme(axis.title.x = element_text(size=14, face="bold", colour = "black"),
        axis.title.y = element_text(size=16, face="bold", colour = "black"),
        axis.text.x = element_text(size=10, face="bold", colour = "black"),
        axis.text.y = element_text(size=12, face="bold", colour = "black"),
        axis.line = element_line(colour = 'black', size = 1),
        plot.title = element_text(hjust = 0.5)) +
  theme(axis.text.x=element_text(angle=75,hjust=1,vjust=1))
dev.off()
```

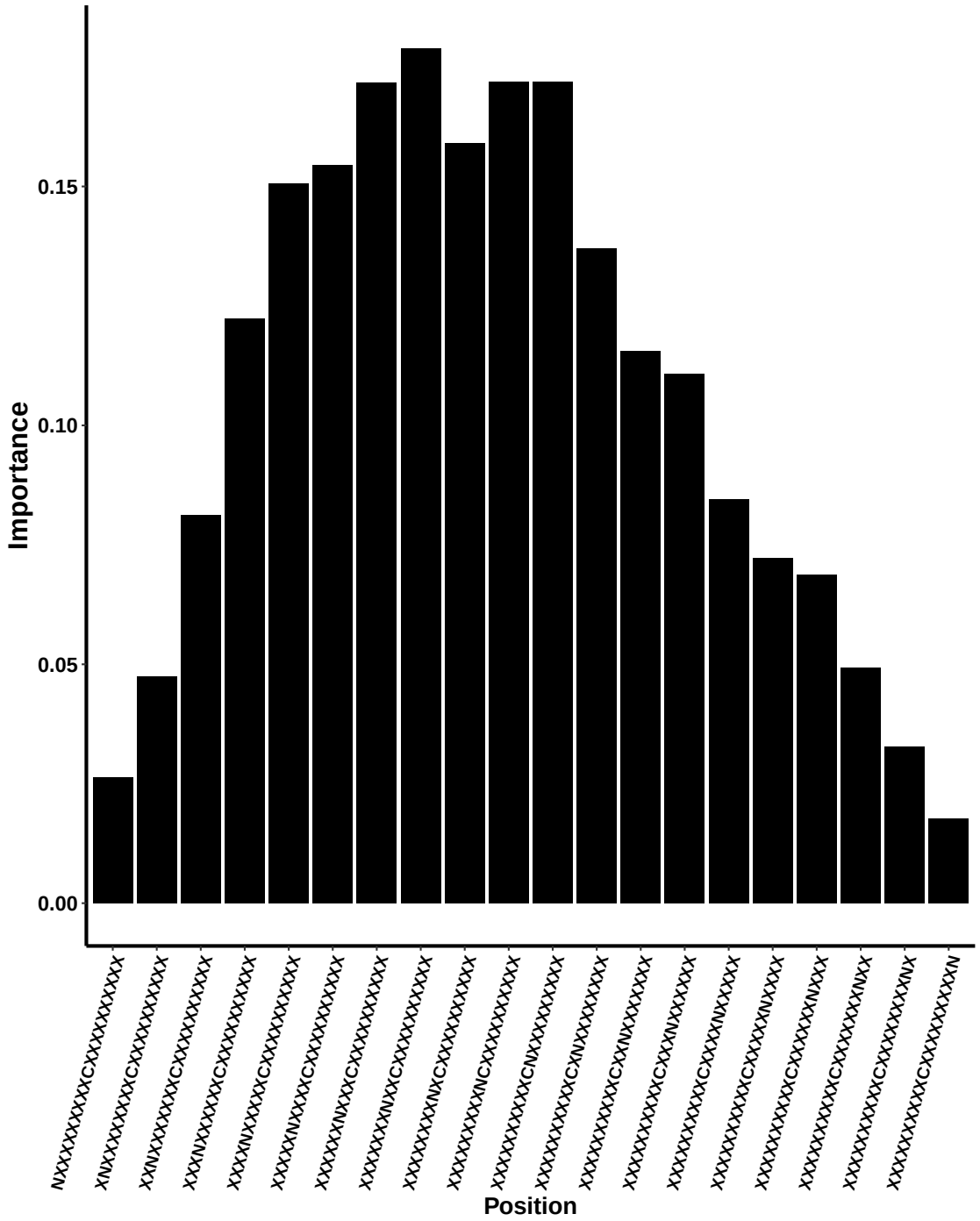


Figure 11: DNase rule ensemble variable importances

5.7 Combine all relevant bigwig files into a single bedGraph for application of the rule ensemble model

```
#convert bigWigs into bedGraphs
for bw in /path/to/bigwigs/*.bigWig
do
    name=$(echo $bw | awk -F".bigWig" '{print $1}' | awk -F"/path/to/bigwigs/" '{print $2}')
  echo $name
  bigWigToBedGraph '/path/to/bigwigs/'$name'.bigWig' $name'.bedGraph'
done
```

```
#Make a text file with all bedgraph names and order
for bw in *.bedGraph
do
    name=$(echo $bw | awk -F".bedGraph" '{print $1}')
    echo $name
    echo $name >> DNase_allmasks_names.txt
done

#Combine all bedGraphs into a single file
bedtools unionbedg -i *.bedGraph > DNase_5mer_union.bedGraph
```

5.8 Apply a rule ensemble model to a bedgraph then convert the bedgraph into a bigwig

```
library(data.table)
options(scipen = 100)
setwd('../multiscale')
################################################################################
import.rules = function(input) {
  rulesdf = data.table(read.table(input, sep = ',', header = TRUE))
  intcoeff = NULL
  lincoeff = NULL
  rulescoeff = NULL
  for (i in 1:nrow(rulesdf)) {
    intcoeff[i] = fifelse(rulesdf[i,description] == 1, rulesdf[i,coefficient], 0)
  }
  intcoeff = intcoeff[!(intcoeff==0)]
  for (i in 1:nrow(rulesdf)) {
    lincoeff[i] = fifelse(!grepl(">|<|=",rulesdf[i, description]),
                          paste('(',rulesdf[i, description], '*',
                                rulesdf[i, coefficient], ')', sep = ''),paste(''))}
  lincoeff = lincoeff[!(lincoeff=='')]
  linpaste = capture.output(cat(lincoeff, sep = ' + '))
  for (i in 1:nrow(rulesdf)) {
    rulescoeff[i] = fifelse(grepl(">|<|=",rulesdf[i, description]),
                            paste('fifelse(', rulesdf[i, description], ', ',
                                  rulesdf[i, coefficient], ', 0)', sep = ''), '')}
  rulescoeff = rulescoeff[!(rulescoeff=='')]
  rulespaste = capture.output(cat(rulescoeff, sep = ' + '))
  output = paste(intcoeff, linpaste, rulespaste, sep = ' + ')
  return(output)
}
################################################################################
x <- fread('/path/to/DNase_5mer_union.bedGraph')
masknames = fread('/path/to/DNase_5mer_names.txt', header = FALSE)
masknames = masknames$V1
masknames = substr(masknames, 6, nchar(masknames))
masknames = c('chr', 'start', 'stop', masknames)
colnames(x) = masknames

unscaled_read_depth = sum(x$unscaled)

prerules = import.rules('DNasePN_pre_coeff.txt')

#Take this output and find + replace ' + ' with ' + \n'
#then find & replace '\\n' to '\n' w/ REGEX enabled
prerules

#THIS IS NOT THE MODEL APPLIED IN THIS FIGURE
#BECAUSE MODELS ARE MANY LINES LONG, THIS IS AN EXAMPLE RE MODEL
x[, pre := ((XXXXXXXXXXXXXXXXXXXXNNNXCXXXXXXXXXXXXXNNNXXXXXXXXX*-0.00033835853362444) +
              (XXXXXXXXXXXXXXXXXXXXXXXCXXXXXNNNNNNNXXXXXXXXXXXX*0.000325170754782039) +
```

```
              (XXXXXXXXXXXXXXXXXXXXXXXCXXXXXXNNNXXXXXXXXXXNNNX*-0.00017781714699625) +
              (XXXXXXXXXXXXXXXXXXXXXXXCXXXXXXXXXXXNNNNNXXXXXX*0.000156335578871717) +
              (XXXXXNNNNNNXXXXXXXXXXXXCXXXXXXXXXXXXXXXXXXXXX*0.000120334948927112) +
              (XXXXXNNNNNNNXXXXXXXXXXXCXXXXXXXXXXXXXXXXXXXXX*0.0000996545475286086) +
              (XXXXXXXXXXXXXXXXXNNNNNCXXXXXXXXXXXXXXXXXXXXX*-0.0000554629144223403) +
              (XXXXXXXXXXXXNNNXXXXXXXCNNNXXXXXXXXXXXXXXXXXX*-0.0000228569691501714) +
              fifelse(XXXXXXXXXXXXXXXXXXNNNNNNCNXXXXXXXXXXXXXXXXXXXX > 1.31426302696887 &
                      XXXXXXXXXXXXXXXXXXXXNNNCNNNNXXXXXXXXXXXXXXXXXX > 2.3666460904978,
                      0.199133517904557, 0) +
              fifelse(XXXXXXXXXXXXXXXXXXXXXXCXXXXXXXXXXNNNNNNNXXXXXX > 1.32015962584499 &
                      XXXXXXXXXXXXXXXXXXXXXXCXXXXNNNNNNNNXXXXXXXXXXX > 1.84844446194714,
                      0.139895551532551, 0) +
              fifelse(XXXXXXXXXXXXXXXXXXXXXXCXXNNNNNNNNXXXXXXXXXXXX > 1.05395715443702 &
                      XXXXXXXXXXXXXXXXXXXNNNXCXXXXXXXXXXXXNNNXXXXXXXX > 1.88046112618925,
                      0.13034286851597, 0) +
              fifelse(XXXXXXXXXXXXXXXXXXXXXXCNNNNNNNXXXXXXXXXXXXXX > 1.2080714012571 &
                      XXXXXXXXXXXXXXXXXXXXXXCXXXXNNNNNNNNXXXXXXXXXXX > 1.91740303016198,
                      0.127269204338179, 0) +
              fifelse(XXXXXXXXXXXXXXXXXXXXXXNCNNXXXXXXXNNNXXXXXXXXXX > 1.18763877707487 &
                      XXXXXXXXXXXXXXXXXXXXXXCXXXNNNNXNNNXXXXXXXXXXXX > 1.60240612072104,
                      0.124824271879124, 0))]

pre_read_depth = sum(x$pre)

RDS = unscaled_read_depth/pre_read_depth

x[, pre_RDS := pre*RDS ]

colnames(x)[96]
write.table(x[,c(1:3,96)], file = 'DNase_RE.bedGraph', col.names = FALSE,
            row.names=FALSE, sep = '\t', quote=FALSE)

#Run the following line from terminal in pwd
#bedGraphToBigWig DNase_RE.bedGraph hg38.fa.chrom.sizes DNase_RE.bedGraph
```

5.9 Once we have the applied rule ensemble model in a bigwig format, we can analyze the bias correction

```
library(bigWig)
library(zoo)
library(lattice)
library(ggseqlogo)
library(data.table)
library(matrixStats)
library(gridExtra)
library(ggplot2)
source('path/to/Figure3_functions.R')
#Set test and training sets
train = c('AR', 'ASCL1', 'ATF3', 'CEBPB', 'CLOCK', 'CTCF', 'DLX2', 'DUX4', 'EGR1',
          'ELF1', 'ESR1', 'FOXA1', 'FOXK2', 'GLIS1', 'HSF1', 'JUN', 'LEF1',
          'MEIS2', 'MLX', 'MYC', 'NR2F2', 'SPIB', 'SRY', 'STAT1', 'TEAD1', 'TGIF1')
test = c('E2F1', 'EBF1', 'FERD3L', 'FOS', 'GATA2', 'HOXC12', 'IRF1', 'MAX', 'MEF2A',
         'MGA', 'NFATC3', 'POU3F1', 'PPARG', 'REST', 'RUNX1', 'Six3', 'SP1', 'USF1')

setwd('DNase_RE_analysis')
#Load in and format unscaled traces
load('DNase_unscaled_compositelist.Rdata')
names(unscaled_compositelist) = substr(names(unscaled_compositelist), 1,
                                       nchar(names(unscaled_compositelist) )- 8)
for (i in 1:length(unscaled_compositelist)) {
  unscaled_compositelist[[i]]$factor = substr(unscaled_compositelist[[i]]$factor, 1,
```

```
                                          nchar(unscaled_compositelist[[i]]$factor) - 8)
}

########################## Load in TF motif coordinates and each TF's length
Motifs <- list.files('/path/to/FIMO')

Motiflist <- vector('list', length(Motifs))
for (i in 1:length(Motifs)) {
  Motiflist[[i]] <- FIMO.to.BED(paste('/path/to/FIMO/', Motifs[i], sep = ''))
}
names(Motiflist) <- substr(Motifs, 1, nchar(Motifs)-22)
Motifs <- names(Motiflist)
Motiflen = NULL
for (i in 1:length(Motiflist)) {
  Motiflen[i] = Motiflist[[i]]$end - Motiflist[[i]]$start
}
#Plot RE model corrected TFs
BWs <- c('/path/to/DNase_RE.bigWig')

compositelist <- vector('list', length(Motifs))
for (i in 1:length(Motiflist)) {
  compositelist[[i]] <- BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                         upstream = 100, downstream = 100,
                                         factor = Motifs[i], group = 'Rule Ensemble', ATAC = FALSE)
}

names(compositelist) <- Motifs
DNase_RE_composite = compositelist
for (i in 1:length(DNase_RE_composite)) {
  DNase_RE_composite[[i]]$factor = substr(DNase_RE_composite[[i]]$factor,
                                          1, nchar(DNase_RE_composite[[i]]$factor) - 22)
}
save(DNase_RE_composite, file = 'DNase_RE_composite.Rdata')

#Plot masked TFs for comparison (this seqOutBias command was run for figure 1C)
BWs <- c('/path/to/DNase_XXXXXXXNNNCNNXXXXXXXX.bigWig')

compositelist <- vector('list', length(Motifs))
for (i in 1:length(Motiflist)) {
  compositelist[[i]] <- BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                         upstream = 100, downstream = 100,
                                         factor = Motifs[i], group = 'seqOutBias', ATAC = FALSE)
}

names(compositelist) <- Motifs
DNase_NNNCNN_compositelist = compositelist

save(DNase_NNNCNN_compositelist, file = 'DNase_NNNCNN_compositelist.Rdata')

##Combine rule ensemble, seqOutBias and unscaled compositelists for plotting
combined_compositelist <- vector(mode = 'list', length = 1)
for (i in 1:length(unscaled_compositelist)) {
  combined_compositelist[[i]] <- rbind(DNase_RE_composite[[i]],
                                       unscaled_compositelist[[i]],
                                       DNase_NNNCNN_compositelist[[i]])
}
names(combined_compositelist) <- names(compositelist)

#Separate out the TFs to be plotted (for 3C) and their motif lengths
figure3C_plot = do.call(rbind, combined_compositelist[c(13, 35, 44)])
mlen <- Motiflen[c(13, 35, 44)]
mlen <- as.data.frame(mlen)
rownames(mlen) <- levels(as.factor(figure3C_plot$factor))
```

```
plot.composites(figure3C_plot, legend = TRUE,
                pdf_name = 'Figure3C_DNasePN_PRE_NNNCNN_comparison',
                figwidth = 8, figheight = 4,
                ylabel = 'Cut Frequency',
                xlabel = 'Distance from Motif Center',
                motifline = TRUE, Motiflen = mlen, layoutgrid = c(3,1))

###Plot log fold change from calculated avg (0.003422133)
test_DNase_RE_compositelist = DNase_RE_composite[which(names(DNase_RE_composite) %in% test)]
test_unscaled_compositelist = unscaled_compositelist[which(names(unscaled_compositelist) %in% test)]
test_DNase_NNNCNN_compositelist =
  DNase_NNNCNN_compositelist[which(names(DNase_NNNCNN_compositelist) %in% test)]

#Plot difference b/t avg baseline and corrected...
test_DNase_RE = do.call(rbind, test_DNase_RE_compositelist)
test_unscaled = do.call(rbind, test_unscaled_compositelist)
test_DNase_NNNCNN = do.call(rbind, test_DNase_NNNCNN_compositelist)

#to get log2 diff- first change to percent diff from baseline
#Calculate distance between target and output
DNase_RE_log = NULL
DNase_NNNCNN_log = NULL
unscaled_log = NULL
for (i in 1:length(test_unscaled$factor)) {
  DNase_RE_log[i] = log2(test_DNase_RE$est[i] / 0.00309736)
  unscaled_log[i] = log2(test_unscaled$est[i] / 0.00309736)
  DNase_NNNCNN_log[i] = log2(test_DNase_NNNCNN$est[i] / 0.00309736)
}

unscaled_log = data.frame(unscaled_log)
unscaled_log[,2] = test_unscaled$factor
unscaled_log[,3] = 'Unscaled'
colnames(unscaled_log) = c('Difference', 'Factor', 'Treatment')

DNase_RE_log = data.frame(DNase_RE_log)
DNase_RE_log[,2] = test_unscaled$factor
DNase_RE_log[,3] = 'RE'
colnames(DNase_RE_log) = c('Difference', 'Factor', 'Treatment')

DNase_NNNCNN_log = data.frame(DNase_NNNCNN_log)
DNase_NNNCNN_log[,2] = test_unscaled$factor
DNase_NNNCNN_log[,3] = 'NNNCNN'
colnames(DNase_NNNCNN_log) = c('Difference', 'Factor', 'Treatment')

log_fold = rbind(DNase_NNNCNN_log, DNase_RE_log, unscaled_log)
log_fold$Treatment = factor(log_fold$Treatment, levels = c('Unscaled',
                                                           'seqOutBias',
                                                           'Rule Ensemble'))


x = DNase_PRE_NNNCNN_log_fold_change

panel.violin.hack <-
  function (x, y, box.ratio = 1, box.width = box.ratio/(1 + box.ratio),
            horizontal = TRUE, alpha = plot.polygon$alpha, border =
              plot.polygon$border,
            lty = plot.polygon$lty, lwd = plot.polygon$lwd, col = plot.polygon
            $col,
            varwidth = FALSE, bw = NULL, adjust = NULL, kernel = NULL,
            window = NULL, width = NULL, n = 50, from = NULL, to = NULL,
            cut = NULL, na.rm = TRUE, ...){
    if (all(is.na(x) | is.na(y)))
      return()
    x <- as.numeric(x)
```

```
    y <- as.numeric(y)
    plot.polygon <- trellis.par.get("plot.polygon")
    darg <- list()
    darg$bw <- bw
    darg$adjust <- adjust
    darg$kernel <- kernel
    darg$window <- window
    darg$width <- width
    darg$n <- n
    darg$from <- from
    darg$to <- to
    darg$cut <- cut
    darg$na.rm <- na.rm
    my.density <- function(x) {
      ans <- try(do.call("density", c(list(x = x), darg)),
                 silent = TRUE)
      if (inherits(ans, "try-error"))
        list(x = rep(x[1], 3), y = c(0, 1, 0))
      else ans
    }
    numeric.list <- if (horizontal)
      split(x, factor(y))
    else split(y, factor(x))
    levels.fos <- as.numeric(names(numeric.list))
    d.list <- lapply(numeric.list, my.density)
    dx.list <- lapply(d.list, "[[", "x")
    dy.list <- lapply(d.list, "[[", "y")
    max.d <- sapply(dy.list, max)
    if (varwidth)
      max.d[] <- max(max.d)
    xscale <- current.panel.limits()$xlim
    yscale <- current.panel.limits()$ylim
    height <- box.width
    if (horizontal) {
      for (i in seq_along(levels.fos)) {
        if (is.finite(max.d[i])) {
          pushViewport(viewport(y = unit(levels.fos[i],
                                         "native"), height = unit(height, "native"),
                                yscale = c(max.d[i] * c(-1, 1)), xscale = xscale))
          grid.polygon(x = c(dx.list[[i]], rev(dx.list[[i]])),
                       y = c(dy.list[[i]], -rev(dy.list[[i]])),
                       default.units = "native",
                       # this is the point at which the index is added
                       gp = gpar(fill = col[i], col = border, lty = lty,
                                 lwd = lwd, alpha = alpha))
          popViewport()
        }
      }
    }
    else {
      for (i in seq_along(levels.fos)) {
        if (is.finite(max.d[i])) {
          pushViewport(viewport(x = unit(levels.fos[i],
                                         "native"), width = unit(height, "native"),
                                xscale = c(max.d[i] * c(-1, 1)), yscale = yscale))
          grid.polygon(y = c(dx.list[[i]], rev(dx.list[[i]])),
                       x = c(dy.list[[i]], -rev(dy.list[[i]])),
                       default.units = "native",
                       # this is the point at which the index is added
                       gp = gpar(fill = col[i], col = border, lty = lty,
                                 lwd = lwd, alpha = alpha))
          popViewport()
        }
      }
```

```
    }
    invisible()
  }



x[,3] <- as.factor(x[,3])


pdf('DNase_log2_comparison.pdf', useDingbats = FALSE, width=10.83, height=6)

trellis.par.set(box.umbrella = list(lty = 1, col="#93939380", lwd=2),
                box.rectangle = list(col = '#93939380', lwd=1.6),
                plot.symbol = list(col='#93939380', lwd=1.6, pch ='.'))

###############################################################################
print(bwplot(Difference ~ Treatment | Factor , data = x,
             between=list(y=0.5, x = 0.5),
             scales=list(x=list(draw=TRUE),rot = 45,
                         alternating=c(1,1,1,1),cex=1,font=1),
             #xlab = '',
             ylim = c(-4.5, 3.5),
             #   main = "Pause Index (PI) Ratio",
             ylab =expression("log"[2]~"(deviation from random cleavage)"),
             horizontal =FALSE,  col= 'black',
             aspect = 2,
             par.settings=list(par.xlab.text=list(cex=1.2,font=1),
                               par.ylab.text=list(cex=1.2,font=1),
                               par.main.text=list(cex=1.2, font=1),
                               plot.symbol = list(col='black', lwd=0, pch =19, cex = 0.0)),
             strip = function(..., which.panel, bg) {
               bg.col = c("grey90")
               strip.default(..., which.panel = which.panel,
                             bg = rep(bg.col, length = which.panel)[which.panel])
             },
             panel = function(..., box.ratio, col) {
               panel.abline(h = 0, col = 'grey45', lty = 2)
               panel.violin.hack(..., col = c("#FF7E78", "#FF5FD2", "#50FFE6"),
                                 varwidth = FALSE, box.ratio = box.ratio, outer = FALSE)
               panel.stripplot(..., col='#54545380', do.out=FALSE, jitter.data=TRUE,
                               amount = 0.2, pch = 16)
               panel.bwplot(..., pch = '|', do.out = FALSE)

             }))
dev.off()
```

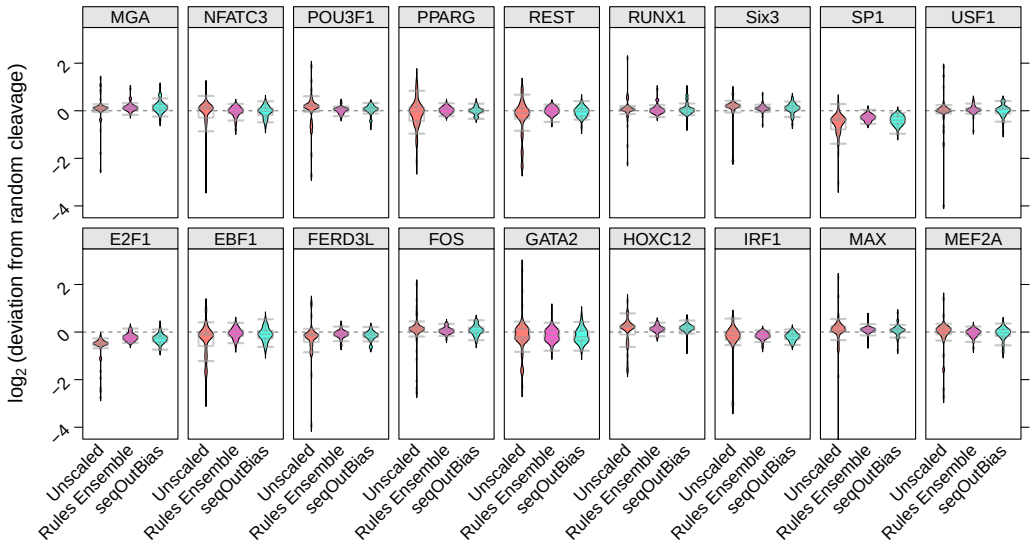


Figure 12: DNase single nucleotide log2 comparison

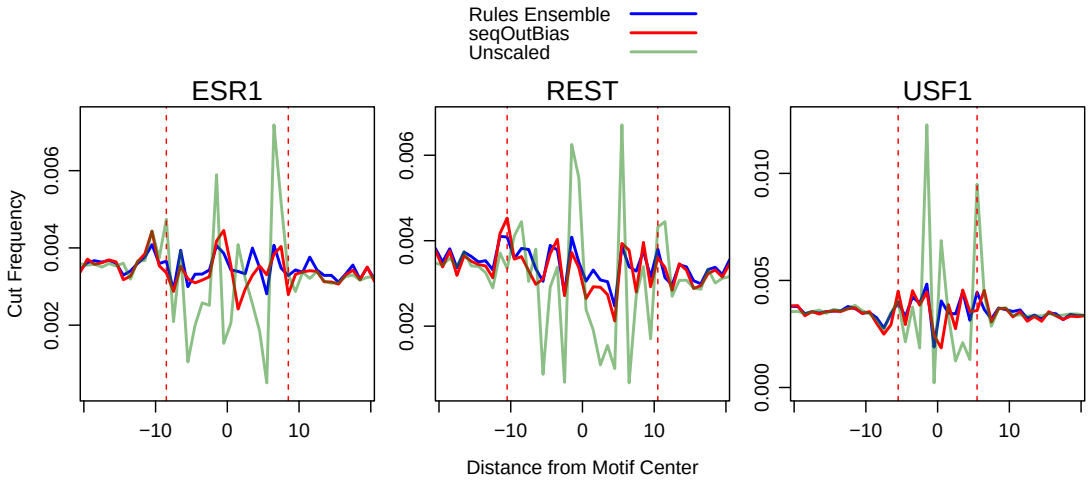


Figure 13: DNase single nucleotide composite comparison

5.10 Figure 3 Functions

```
#Comput scalefactors from seqOutBias seqtable output
scalefactor.func <- function(factortable){
    require(data.table)
    datatable = fread(factortable, skip = 1)
    datatable[, pluscountpercent := datatable[,3]/sum(datatable[,3])]
    datatable[, plusobspercent := datatable[,5]/sum(datatable[,5])]
    datatable[, plusscalefact := round(datatable[,pluscountpercent]/datatable[,plusobspercent], 6)]
    datatable[, minuscountpercent := datatable[,4]/sum(datatable[,4])]
    datatable[, minusobspercent := datatable[,6]/sum(datatable[,6])]
    datatable[, minusscalefact := round(datatable[,minuscountpercent]/datatable[,minusobspercent], 6)]
}
```

```
#FIMO.to.Bed takes input in FIMO format (1-based start and end) and converts
#it to BED format (0-based start, 1-based end), also increases window by amount specified.
FIMO.to.BED <- function(fimofile, window = 0) {
    require(data.table)
    #load in FIMO file
    FIMOdf <- fread(fimofile, sep = '\t', header = TRUE)
    #shift start position by -1 to make it 0-based
    FIMOdf[,4] = FIMOdf[,4] - 1
    #make location column
    FIMOdf[,location := paste(sequence_name, ':', start, '-', stop, sep = '')]
    #Rearrange columns for BED format
    beddf <- FIMOdf[, c(3,4,5,2,11,6)]
    colnames(beddf) <- c('chr', 'start', 'end', 'gene', 'location', 'strand')
    #CB (central base) determines the central base which the window is based on
    #also used as the 0 position on x-axis
    #even motifs will == 0, odd will == 1
    if ((beddf[1,3] - beddf[1,2])%%2 == 0) {
        CB = data.table(beddf)
        #convert start positions to 1-base for calculating CB
        CB[,start := start + 1]
        #Calculate median ceiling for positive motifs, and median floor value
        # for minus motifs then use this value to determine
        #start and end using upstream/downstream
        CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
        setkey(CB,strand)
        CB[c("-"),median := floor(median)]
        CB[c("+"),median := ceiling(median)]
        CB[,start := median - window]
        CB[,end := median + window]
        #Convert start position back to 0-base
        CB[,start := start - 1]
        CB[,median:=NULL]
    } else {
        CB = data.table(beddf)
        #convert start positions to 1-base for calculating median
        CB[,start := start + 1]
        #Calculate median for each row, then use this value to determine
        #start and end using upstream/downstream
        CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
        CB[,start := median - window]
        CB[,end := median + window]
        #Convert start position back to 0-base
        CB[,start := start - 1]
        CB[,median:=NULL]
    }

    return(CB)
}

#Split up a list of sequences into mers = mermask for each mappable position in the sequence
mer.positions <- function(input, mermask = 'NNNXXXXXNNN') {
    seqdf_mer <- vector(mode = "list", length = nchar(input[1]))
    if (grepl('X', mermask) == TRUE) {
    mermask_spaces = unlist(strsplit(mermask, 'X'))
    mer1 = nchar(mermask_spaces)[1]
    unmasked = length(which(mermask_spaces == '')) + 1
    mer2 = nchar(mermask_spaces)[length(mermask_spaces)]
    for (i in 1:nchar(input[1])) {
        seqdf_store = data.table()
        seqdf_store[, alltogether := paste(substr(input[1:length(input)], i, i + mer1 - 1),
                                           paste(rep('X', unmasked), collapse = ""),
                                           substr(input[1:length(input)],
                                           i + mer1 + unmasked, i + mer1 + unmasked + mer2 - 1), sep = '')]
        seqdf_mer[[i]] <- seqdf_store$alltogether
```

```
    }
    mersize = nchar(mermask)
    names(seqdf_mer) <- paste0("Position_", seq_along(1:nchar(input[1])))
    seqdf_mer <- seqdf_mer[1:(length(seqdf_mer)-mersize+1)]}
    else {
      seqdf_store <- NULL
      mersize = nchar(mermask)
      for (i in 1:nchar(input[1])) {
        seqdf_store <- substr(input[1:length(input)], i, i+ (mersize-1))
        seqdf_mer[[i]] <- seqdf_store
        names(seqdf_mer) <- paste0("Position_", seq_along(1:nchar(input[1])))
      }
      seqdf_mer <- seqdf_mer[1:(length(seqdf_mer)-mersize+1)]
    }
    return(seqdf_mer)
}


#Get mercounts for each possible mer from the output of mer.positions - only for 1 sequence
mer.counts = function(sequence_mer_positions, mermask = 'NNNN') {
  #Create all possible combinations of kmer
  mermask = unlist(strsplit(mermask, split = ''))
  mertable = expand.grid(rep(list(c('A','C','G','T')), length(which(mermask == 'N'))))
  if (length(which(mermask == 'X')) > 0) {
    Xrows = matrix(nrow = nrow(mertable), ncol = length(which(mermask == 'X')))
    Xrows[,c(1:length(which(mermask == 'X')))] = rep('X', nrow(mertable))
    mertable = cbind(mertable[,1:(which(mermask=='X')[1]-1)],
                     Xrows, mertable[,(which(mermask=='X')[1]:ncol(mertable))])
  } else {}
  mertable = data.frame(apply(mertable, 1 , paste, collapse = ""))
  mercount = as.character(sequence_mer_positions)
  merstore = NULL
  #
  for (i in 1:nrow(mertable)) {
    merstore = length(which(mercount %in% mertable[i,1]))
    if (length(merstore) == 0) {
      merstore = 0
    } else {}
    mertable[i,2] = merstore
  }
  colnames(mertable) = c('kmer', 'count')
  return(mertable)
}


#Get position frequencies for each possible mer from the output of mer.positions
position.frequencies = function(sequence_mer_positions, mermask = 'NNNNNNN') {
  #Create all possible combinations of kmer
  mermask = unlist(strsplit(mermask, split = ''))
  mertable = expand.grid(rep(list(c('A','C','G','T')), length(which(mermask == 'N'))))
  if (length(which(mermask == 'X')) > 0) {
  Xrows = matrix(nrow = nrow(mertable), ncol = length(which(mermask == 'X')))
  Xrows[,c(1:length(which(mermask == 'X')))] = rep('X', nrow(mertable))
  mertable = cbind(mertable[,1:(which(mermask=='X')[1]-1)],
                   Xrows, mertable[,(which(mermask=='X')[1]:ncol(mertable))])
  } else {}
  mertable = data.table(apply(mertable, 1 , paste, collapse = ""))
  posfreq_poslist = vector(mode = "list", length = length(sequence_mer_positions))
  #
  for (p in 1:length(sequence_mer_positions)) {
    posfreq_posstore = data.table(mertable$V1)
    postable = table(sequence_mer_positions[[p]])
    if (length(which(!names(postable) %in% mertable$V1)) > 0) {
    postable = postable[-c(which(!names(postable) %in% mertable$V1))]
```

```
    } else {}
    missing_mers = as.character(mertable$V1[which(!mertable$V1 %in% names(postable))])
    append_mers = rep(0, length(missing_mers))
    names(append_mers) = missing_mers
    postable = c(postable, append_mers)
    postable = postable/length(sequence_mer_positions[[1]])
    posfreq_poslist[[p]] <- postable

  }
  names(posfreq_poslist) <- paste0("Position_", seq_along(1:length(posfreq_poslist)))
  return(posfreq_poslist)
}
##Multiply 1/scale_factor by its corresponding kmer frequency for each position in
##the composite
scalefactor.by.kmerfrequency <- function(scalefactors, kmerfrequency, tfname) {
    all_scalefactors = vector(mode = "list", length = length(scalefactors))
    names(all_scalefactors) <- names(scalefactors)
   for (j in 1:length(scalefactors)) {
       scalefactor = scalefactors[[j]]
       scalefactor = scalefactor[order(V2)]
       scaledkfreq = vector(mode = "list", length = length(kmerfrequency))
       names(scaledkfreq) = names(kmerfrequency)
    for (i in 1:length(kmerfrequency)) {
        scaledkfreq[[i]] = data.table(stack(kmerfrequency[[i]]))
        colnames(scaledkfreq[[i]]) = c('kmerfreq', 'kmer')
        scaledkfreq[[i]]$kmer = as.character(scaledkfreq[[i]]$kmer)
        scaledkfreq[[i]] = scaledkfreq[[i]][order(kmer)]
        scaledkfreq[[i]][, mask := scalefactor[,2]]
        scaledkfreq[[i]][, plusscalefreq := (1/(scalefactor[,9])) * kmerfreq ]
        scaledkfreq[[i]][, minusscalefreq := (1/(scalefactor[,12])) * kmerfreq ]

        }
       saveRDS(scaledkfreq, file = paste(tfname, '_',
                       substr(names(scalefactors[j]), 1,
                              nchar(names(scalefactors[j]))-27), '.rds', sep = ''))
       all_scalefactors[[j]] = scaledkfreq
       }
      return(all_scalefactors)
}

##Take sum of each position's scaledkmerfreq (1/scale_factor * kmer frequency) and use to create
##data.table of inputs for each mask for each TF motif
scaledkmerfreq.sum.plus <- function(input) {
    output = data.table(matrix(nrow = length(input)))
    for (i in 1:length(input)) {

            output[i, scaledkmerfreqsumplus := sum(input[[i]][,4])]

    }
    output[,1] <- NULL
    rownames(output) = names(input)
    return(output)
}



##############################################################################
##############################################################################
#Combine values from mask positions to plot redudant importance values for each position
redundantpos = function(input, labelnum = 2, impnum = 3) {
  input[,labelnum] = as.character(input[,labelnum])
  input[,labelnum] = gsub('C', '', input[,labelnum])
  posimp = vector(mode = 'list', length = nchar(input[1,labelnum]))
  for (i in 1:nchar(input[1,labelnum])) {
```

```
    for (p in 1:nrow(input)) {
      posimp[[i]][p] = fifelse(substr(input[p,labelnum],i,i) == 'N', input[p,impnum],0)
    }
  }
  output = data.table(matrix(nrow = length(posimp), ncol = 1))
  for (i in 1:length(posimp)) {
    output[i, possum := sum(posimp[[i]])]
  }
  output = output[,-c(1)]
  output[, position := paste(rep('X',36,sep = ''), collapse = '')]

  for (i in 1:nrow(output)) {
    output[i,]$position = paste(substr(output[i, position],1,i-1),'N',
                                substr(output[i, position],i+1,nchar(output[i, position])), sep = '')
    output[i,]$position = paste(substr(output[i, position],1,(nchar(output[i, position])/2)), 'C',
                                paste(substr(output[i, position],
                                (nchar(output[i, position])/2+1),nchar(output[i, position]))), sep = '')
  }
  return(output)
}

#BED.query.bigWig uses the bed coordinates produced by FIMO.to.BED to
#query a supplied bigWig file for the read counts at the specified positions
#In order to create a window around the base of interest, a central base (CB)
#is determined based on whether or not the motif of interest is odd or even
#additionally, if ATAC=FALSE, minus-aligned motifs are shifted +1
#relative to the plus strand to account for DNase etc. cutting between bases
#finally, the mean of each position's read counts is calculated for plotting
BED.query.bigWig <- function(beddf, bwPlus, bwMinus, upstream = 10,
                             downstream = 10,
                             factor = '', group = '', ATAC = TRUE) {
    require(bigWig)
    require(data.table)
    require(matrixStats)
    step = 1
    #load bigWigs
    bw.plus = load.bigWig(bwPlus)
    bw.minus = load.bigWig(bwMinus)
    #CB (central base) determines the central base which the window is based on
    #also used as the 0 position on x-axis
    #even motifs will == 0, odd will == 1
    if ((beddf[1,3] - beddf[1,2])%%2 == 0) {
        CB = data.table(beddf)
        #convert start positions to 1-base for calculating CB
        CB[,start := start + 1]
        #Calculate median ceiling for positive motifs, and median floor value
        # for minus motifs then use this value to determine
        #start and end using upstream/downstream
        CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
        setkey(CB,strand)
        CB[c("-"),median := floor(median)]
        CB[c("+"),median := ceiling(median)]
        CB[,start := median - upstream]
        CB[,end := median + downstream]
        #Convert start position back to 0-base
        CB[,start := start - 1]
        CB[,median:=NULL]
    } else {
        CB = data.table(beddf)
        #convert start positions to 1-base for calculating median
        CB[,start := start + 1]
        #Calculate median for each row, then use this value to determine
        #start and end using upstream/downstream
        CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
```

```
        CB[,start := median - upstream]
        CB[,end := median + downstream]
        #Convert start position back to 0-base
        CB[,start := start - 1]
        CB[,median:=NULL]
    }
    #Shift minus-aligned motifs by +1 for DNase etc.
    if (ATAC == FALSE) {
        setkey(CB,strand)
        CB[c("-"),start := start + 1]
        CB[c("-"),end := end + 1]
    } else {}
    #extract read count information from bigWigs, using CB-derived
    #window as coordinates
    tss.matrix = bed6.step.bpQuery.bigWig(bw.plus, bw.minus , CB,
                                step = 1, as.matrix=TRUE, follow.strand=TRUE)
    tss_correction = sum(tss.matrix)/(nrow(tss.matrix)*ncol(tss.matrix))
    #Determine beginning and end of plot
    #if not ATAC, peaks are between bases, so offset by -1 and
    #add 0.5 to align 'cuts' on 0 at -0.5
    if (ATAC == FALSE) {
        coordin.start = (-upstream - 0.5 )
        coordin.end = (downstream - 0.5 )
    } else {
        #For tn5, add negative upstream value and downstream value
        coordin.start = (-upstream)
        coordin.end = (downstream)
    }
    #create coordinates for aligning read counts on composite profile
    #also take means of each column from tss.matrix and label each row with
    #factor column
    composite.lattice = data.table(seq(coordin.start, coordin.end, by = step),
                                   colMeans(tss.matrix),
                                   factor, group, tss_correction,
                                   stringsAsFactors = FALSE)
    colnames(composite.lattice) = c('x', 'est', 'factor', 'group', 'correction')
    composite.lattice$x = as.numeric(composite.lattice$x)
    unload.bigWig(bw.plus)
    unload.bigWig(bw.minus)
    return(composite.lattice)
}
#

##Plot.composites takes the composite.lattice object and creates a plot for
#this data, while also allowing a mapping of the original motif width onto the
#plot by specifying a TFlen value (the default is 10).
plot.composites <- function(dat, ylabel = '', pdf_name = 'PLEASE_SET_FILE_NAME',
                            xlabel = '', striplabel = TRUE, legend = TRUE,
                            motifline = FALSE, Motiflen = 10,
                            figwidth = 2.5, figheight=3,
                            indexlist = NULL, layoutgrid = NULL,
                    col.lines = c("#0000FF", "#FF0000", "#00000090",
                                  rgb(0.1,0.5,0.05,1/2), rgb(0,0,0,1/2),
                                  rgb(1/2,0,1/2,1/2), rgb(0,1/2,1/2,1/2),
                                  rgb(1/2,1/2,0,1/2)),
                                  fill.poly = c(rgb(0,0,1,1/4),
                                  rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),
                                  rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4))) {
    require(lattice)
    pdf(paste(pdf_name, '.pdf', sep = ''), width= figwidth, height= figheight)
    print(xyplot(est ~ x|factor, group = group, data = dat, strip = striplabel,
                 type = 'l', as.table = TRUE,
                 scales=list(x=list(cex=0.8,relation = "free", axs ="i"),
                             y =list(cex=0.8, relation="free", tick.number=4)),
```

```
                col = col.lines,
                auto.key = if (legend == TRUE)
                    {list(points=F, lines=T, cex=0.8)} else{},
                par.settings = list(strip.background=list(col="#00000000"),
                                    strip.border = list(col = 'transparent'),
                                    superpose.symbol = list(pch = c(16),
                                                            col=col.lines,
                                                            cex =0.5),
                                    superpose.line = list(col = col.lines,
                                                          lwd=c(2),
                                                          lty = c(1))),
                cex.axis=1.0,
                par.strip.text=list(cex=0.9, font=1, col='black'),
                aspect=1.0,
                between=list(y=0.5, x=0.5),
                lwd=2,
                ylab = list(label = paste(ylabel), cex =0.8),
                xlab = list(label = paste(xlabel), cex =0.8),
                index.cond = indexlist,
                layout = layoutgrid,
                panel = function(x, y, ...) {
                    panel.xyplot(x, y, ...)
                    #panel.abline(h = 0, lty =1, lwd = 1.0, col = '#A9A9A932')
                    if (motifline == TRUE)
                       {panel.abline(v = Motiflen/2, lty = 2, col = "red")} else{}
                    if (motifline == TRUE)
                       {panel.abline(v = -Motiflen/2, lty = 2, col = "red")} else{}
                }
    ))
    dev.off()
}
#
##Plots seqlogos
plot.seqlogo.func <- function(x, outfile = "PLEASE_SET_FILE_NAME.pdf") {
    require(ggseqlogo)
    w =  0.663 + (ncol(x) + 1)*0.018 + (ncol(x)+2)* .336
    pdf(outfile, useDingbats=FALSE, width=w, height=2.695)
    print(ggseqlogo(x,  facet = "wrap", font = 'helvetica_bold'))
    dev.off()
}
```

6 Figure 6. Rule ensemble modeling corrects Tn5 transposon bias in ATAC-seq data

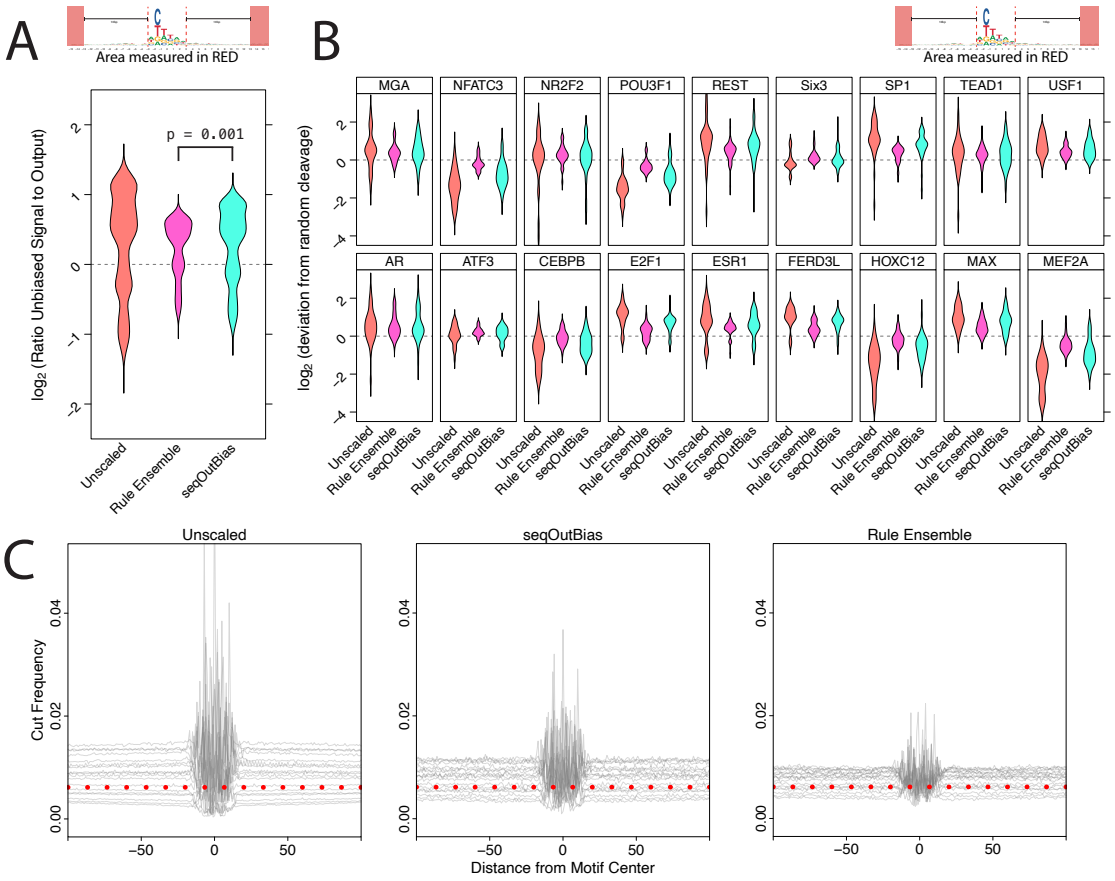


Figure 5. Rule ensemble modeling corrects Tn5 transposon bias in ATAC-seq data. (A) Comparison of baseline correction by rule ensemble modeling vs single mask seqOutBias. Baseline correction is measured by the numerical difference from random cleavage. Regions plotted are indicated by the PSWM below.(B) Comparison of the single nucleotide correction for each transcription factor's composite profile given the method of kmer scaling. All 18 test transcription factors examined were included in this figure. (C) Composite profile overlays for each method of bias correction. The red dashed line indicates the calculated random cleavage frequency.

Figure 14: Figure 5. Rule ensemble modeling corrects Tn5 transposon bias in ATAC-seq data

6.1 Generate scale factors for Tn5 using all 5/6/7/spaced 3-mers in a 46bp window

Using the previously downloaded and aligned deproteinized Tn5 data, we begin generating iterative masks. First, we run seqOutBias with no scale to generate all suffixerator and tallymer intermediate files. This allows us to then use these files in parallel runs of many masks.

```
#Make a directory for Tn5 masks
mkdir Tn5_data
cd Tn5_data

#Run seqOutBias with no scaling to generate intermediate files which will be used by parallel processors
seqOutBias ../hg38.fa ../C1_gDNA_rep1.bam --no-scale --strand-specific --custom-shift=4,-4 --bed=hg38_Tn5_C1gDNA_unscaled
```

Generate tbl files for all 5-mers within a 46bp window

```
library(parallel)
library(bigWig)
```

```
source("https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_hcsearch.R")
#This is the span of bp the 5bp mask will be iterated over
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
#A function which iterates across the start_mask, creating all possible 5-mers
neighbors <- function(vecmask) {
    # generate list of neighboring masks that differ
    # by the addition of a single unmasked position
    result <- vector(mode="list", length=length(vecmask))
    n <- 0
    for (k in 1:length(vecmask)) {
        if (vecmask[k] == -1) { # X
            n <- n + 1
            vi = vecmask
            vi[k] = 1 # N
            vi[k+1] = 1 # 2nd N
            vi[k+2] = 1 # 3rd N
            vi[k+3] = 1 # 4th N
            vi[k+4] = 1 # 5th N
            result[[n]] <- vec.to.mask(vi)
        }
    }
    if (n == 0) return(NULL)
    #Remove last line so you don't get an extra N space (mappable bases = mer-1)
    zz <- (n-4)
    result[1:zz]
}

#Create a list of all possible 5-mer positions
nextlst = neighbors(mask.to.vec(start_mask))
#seqOutBias arguments passed to each run
seqOutBias.args = "../hg38.fa ../C1_gDNA_rep1.bam --read-size=76 --strand-specific --custom-shift=4,-4"
#Command to call seqOutBias
sqcmd = "seqOutBias"
#Prefix added to each output file
prefix = "Tn5_"
#Apply run.cutmask to each possible 5-mer
scores = mclapply(nextlst, function(cutmask) {
        bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 20)
```

Generate tbl files for all 6-mers within a 46bp window, for comments see 5-mer script above

```
library(parallel)
library(bigWig)
source("https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_hcsearch.R")
setwd('../output')
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
neighbors <- function(vecmask) {
    # generate list of neighboring masks that differ
    # by the addition of a single unmasked position
    result <- vector(mode="list", length=length(vecmask))
    n <- 0
    for (k in 1:length(vecmask)) {
        if (vecmask[k] == -1) { # X
            n <- n + 1
            vi = vecmask
            vi[k] = 1 # N
            vi[k+1] = 1 # 2nd N
            vi[k+2] = 1 # 3rd N
            vi[k+3] = 1 # 4th N
            vi[k+4] = 1 # 5th N
            vi[k+5] = 1 # 6th N
            result[[n]] <- vec.to.mask(vi)
```

```
        }
    }
    if (n == 0) return(NULL)
    #Remove last line so you don't get an extra N space (mappable bases = mer-1)
    zz <- (n-5)
    result[1:zz]
}
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../hg38.fa ../C1_gDNA_rep1.bam --read-size=76 --strand-specific --custom-shift=4,-4"
sqcmd = "seqOutBias"
prefix = "Tn5_"
scores = mclapply(nextlst, function(cutmask) {
        bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 20)
```

Generate tbl files for all 7-mers within a 46bp window, for comments see 5-mer script above

```
library(parallel)
library(bigWig)
source("https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_hcsearch.R")
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
neighbors <- function(vecmask) {
    # generate list of neighboring masks that differ
    # by the addition of a single unmasked position
    result <- vector(mode="list", length=length(vecmask))
    n <- 0
    for (k in 1:length(vecmask)) {
        if (vecmask[k] == -1) { # X
            n <- n + 1
            vi = vecmask
            vi[k] = 1 # N
            vi[k+1] = 1 # 2nd N
            vi[k+2] = 1 # 3rd N
            vi[k+3] = 1 # 4th N
            vi[k+4] = 1 # 5th N
            vi[k+5] = 1 # 6th N
            vi[k+6] = 1 # 7th N
            result[[n]] <- vec.to.mask(vi)
        }
    }
    if (n == 0) return(NULL)
    #Remove last line so you don't get an extra N space (mappable bases = mer-1)
    zz <- (n-6)
    result[1:zz]
}
nextlst = neighbors(mask.to.vec(start_mask))
seqOutBias.args = "../hg38.fa ../C1_gDNA_rep1.bam --read-size=76 --strand-specific --custom-shift=4,-4"
sqcmd = "seqOutBias"
prefix = "Tn5_"
scores = mclapply(nextlst, function(cutmask) {
        bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 20)
```

Make a text file of all possible combinations of two 3-mers in a 46bp span

```
source("https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_hcsearch.R")
#Input the span (46bp here) of the iterations
start_mask = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
#A function which iterates across the start_mask, creating all possible 5-mers
neighbors <- function(vecmask) {
  # generate list of neighboring masks that differ
  # by the addition of a single unmasked position
  result <- vector(mode="list", length=length(vecmask))
```

```
  n <- 0
  for (k in 1:length(vecmask)) {
    if (vecmask[k] == -1) { # X
      n <- n + 1
      vi = vecmask
      vi[k] = 1 # N
      vi[k+1] = 1 # 2nd N
      vi[k+2] = 1 # 3rd N
      result[[n]] <- vec.to.mask(vi)
    }
  }
  if (n == 0) return(NULL)
  #Remove last line so you don't get an extra N space (mappable bases = mer-1)
  zz <- (n-2)
  result[1:zz]
}
#Take the output from the first iteration of all possible 3-mers
initial_masks = unlist(neighbors(mask.to.vec(start_mask)))
#Make second iteration of 3-mers by making a permutation for each possible spacing
#of the original iteration across the 46-bp span
mask_store = vector(mode = 'list', length = length(initial_masks))
for (i in 1:length(initial_masks)) {
  mask_store[[i]] = unlist(neighbors(mask.to.vec(initial_masks[i])))
}
#Take each list object and turn it into a column of a dataframe
all_masks = data.frame(do.call(rbind, mask_store))
mask_col = NULL
#Take each column of the data frame and concatenate it
for (i in 1:ncol(all_masks)) {
  mask_col = c(mask_col, all_masks[,i])
}
#remove overlapping positions
mask_col = mask_col[-grep('NNNN' ,mask_col)]
#Find unique masks
mask_col = unique(mask_col)
write.table(mask_col, file = 'all_possible_spaced_3mers_46bp.txt', quote = FALSE,
            sep = '\n', row.names = FALSE)
system('mv all_possible_spaced_3mers_46bp.txt ../')
```

Run seqOutBias with all possible spaced 3-mer masks to generate their .tbl files

```
library(parallel)
library(bigWig)
source("https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_hcsearch.R")
#Load in all spaced 3-mer masks
masks = read.table('../scripts/all_possible_spaced_3mers_46bp.txt', skip = 1)
#Assign each mask to a list object in nextlst
nextlst = vector(mode = 'list', length = masks)
for (i in 1:length(masks)) {
  nextlst[[i]] = masks[i,1]
}
#Arguments for seqOutBias
seqOutBias.args = "../hg38.fa ../C1_gDNA_rep1.bam --read-size=76 --strand-specific --custom-shift=4,-4"
#Command to call seqOutBias
sqcmd = "seqOutBias"
#Prefix for output
prefix = "Tn5_"
#Run seqOutBias for each mask and get their .tbl files
scores = mclapply(nextlst, function(cutmask) {
  bw.paths = run.cutmask(cutmask, seqOutBias.args, sqcmd=sqcmd, prefix=prefix, cleanup = FALSE)
}, mc.cores = 20)
```

Run seqOutBias table for each mask tbl file to determine scale factors for each mask.

```
library('doParallel')
masks = list.files('./')
masks = masks[grep('.tbl', masks)]
#
registerDoParallel(20)
foreach (i = 1:length(masks)) %dopar% {system(paste('seqOutBias table ',
         masks[i], ' ../C1_gDNA_rep1.bam > ', substr(masks[i], 1, 51),
         '_fulldat_scale_factors.txt', sep = ''))}
```

6.2 Determine k-mer frequency at each position for the plus and minus training motifs for 5/6/7 spaced 3-mers

Calculate all 5-mer frequencies for each position in motif sequences

```
library(data.table)
source('Tn5_Bias_Functions.R')
system('mkdir 5mer')
setwd('../5mer')
options(scipen = 100)
system('mkdir plus')
setwd('plus')
train = c('AR', 'ASCL1', 'ATF3', 'CEBPB', 'CLOCK', 'CTCF', 'DLX2', 'DUX4', 'EGR1',
          'ELF1', 'ESR1', 'FOXA1', 'FOXK2', 'GLIS1', 'HSF1', 'JUN', 'LEF1',
          'MEIS2', 'MLX', 'MYC', 'NR2F2', 'SPIB', 'SRY', 'STAT1', 'TEAD1', 'TGIF1')
#Import TF FIMO coordinates to extract the sequences from the FASTA...
FIMOfiles_plus = list.files('../../FIMO_plus')
FIMOfiles_plus = FIMOfiles_plus[grep(paste(train, collapse = '_|'), FIMOfiles_plus)]
TFmotifs_plus = vector(mode = "list", length = length(FIMOfiles_plus))
names(TFmotifs_plus) = substr(FIMOfiles_plus, 1, (nchar(FIMOfiles_plus)-14))
#Import FIMO files for genomic coordinates and
#expand a 'window' around these sites (using a central base)
for (i in 1:length(TFmotifs_plus)) {
  TFmotifs_plus[[i]] <- FIMO.to.BED(paste('../../FIMO_plus/',
                                          FIMOfiles_plus[i], sep = ''), window = 100)
}
#Make sure no positions are less than 0
for (i in 1:length(TFmotifs_plus)) {
  print(TFmotifs_plus[[i]][which.min(TFmotifs_plus[[i]]$start),])
}
#Write these coordinates into a bed file for use with 'getfasta' from bedtools
for (i in 1:length(TFmotifs_plus)) {
  write.table(TFmotifs_plus[[i]], file = paste(names(TFmotifs_plus[i]),
                                               '_hg38coords.bed', sep = ''),
              row.names=FALSE, sep="\t", quote = FALSE, col.names = FALSE)
}
#Run getfasta on genomic coordinates to get sequences for window
for (i in 1:length(TFmotifs_plus)) {
  system(paste('bedtools getfasta -bedOut -s -fi ../../data/hg38.fa -bed ',
               names(TFmotifs_plus[i]), '_hg38coords.bed > ',
               names(TFmotifs_plus[i]), '_hg38seq.bed', sep = ''))
}
#load in bed files with sequences
TFseq_plus <- vector(mode = "list", length = length(TFmotifs_plus))
names(TFseq_plus) <- names(TFmotifs_plus)
for (i in 1:length(TFmotifs_plus)) {
  TFseq_plus[[i]] <- fread(paste(names(TFmotifs_plus[i]),
                                 '_hg38seq.bed', sep = ''))
}

#Make sure no sequences have been lost
for (i in 1:length(TFseq_plus)) {
  print(nrow(TFseq_plus[[i]]))
```

```
}
#Make all sequences upper case
TFseq_plus = lapply(TFseq_plus, function(x) {toupper(x$V7)})
save(TFseq_plus, file = 'TFseq_plus.Rdata')

#Break sequences into 5bp chunks (for 5mer)
TFseq_plus = lapply(TFseq_plus, mer.positions, mermask = "NNNNN")
#Find frequency of each possible 5mer for each position
TFseq_plus = lapply(TFseq_plus, position.frequencies, mermask = "NNNNN")
save(TFseq_plus, file = 'Tn5_TF_position_frequencies_5mer_plus.Rdata')

#Determine minus strand 5-mer frequencies
setwd('../minus')
#Import TF FIMO coordinates to extract the sequences from the FASTA...
FIMOfiles_minus = list.files('../../FIMO_minus')
FIMOfiles_minus = FIMOfiles_minus[grep(paste(train, collapse = '_|'), FIMOfiles_minus)]
TFmotifs_minus = vector(mode = "list", length = length(FIMOfiles_minus))
names(TFmotifs_minus) = substr(FIMOfiles_minus, 1, (nchar(FIMOfiles_minus)-14))
#Import FIMO files for genomic coordinates and expand a 'window' around
#these sites (using a central base)
for (i in 1:length(TFmotifs_minus)) {
  TFmotifs_minus[[i]] <- FIMO.to.BED(paste('../../FIMO_minus/',
                                           FIMOfiles_minus[i], sep = ''), window = 100)
}
#CHECK TO SEE IF OFF CHROMOSOME -- SMALLEST START POSITION
for (i in 1:length(TFmotifs_minus)) {
  print(TFmotifs_minus[[i]][which.min(TFmotifs_minus[[i]]$start),])
}
#Write these coordinates into a bed file for use with 'getfasta' from bedtools
for (i in 1:length(TFmotifs_minus)) {
  write.table(TFmotifs_minus[[i]], file = paste(names(TFmotifs_minus[i]),
                                                '_hg38coords.bed', sep = ''),
              row.names=FALSE, sep="\t", quote = FALSE, col.names = FALSE)
}
#Run getfasta on genomic coordinates to get sequences for window
for (i in 1:length(TFmotifs_minus)) {
  system(paste('bedtools getfasta -bedOut -s -fi ../../data/hg38.fa -bed ',
               names(TFmotifs_minus[i]), '_hg38coords.bed > ',
               names(TFmotifs_minus[i]), '_hg38seq.bed', sep = ''))
}


#load in bed files with sequences
TFseq_minus <- vector(mode = "list", length = length(TFmotifs_minus))
names(TFseq_minus) <- names(TFmotifs_minus)
for (i in 1:length(TFmotifs_minus)) {
  TFseq_minus[[i]] <- fread(paste(names(TFmotifs_minus[i]), '_hg38seq.bed', sep = ''))
}

#Check to make sure no sequences have been lost
for (i in 1:length(TFseq_minus)) {
  print(nrow(TFseq_minus[[i]]))
}

#Make all sequences upper case
TFseq_minus = lapply(TFseq_minus, function(x) {toupper(x$V7)})
save(TFseq_minus, file = 'TFseq_minus.Rdata')
#Break sequences into 5bp chunks (for 5mer)
TFseq_minus = lapply(TFseq_minus, mer.positions, mermask = "NNNNN")
#Find frequency of each possible 5mer for each position
TFseq_minus = lapply(TFseq_minus, position.frequencies, mermask = "NNNNN")
save(TFseq_minus, file = 'Tn5_TF_position_frequencies_5mer_minus.Rdata')
```

Calculate all 6-mer frequencies for each position in motif sequences

```
library(data.table)
source('Tn5_Bias_Functions.R')
system('mkdir 6mer')
setwd('../6mer')
options(scipen = 100)
system('mkdir plus')
setwd('plus')
#Load in plus sequence data
load('TFseq_plus.Rdata')
#Break sequences into 5bp chunks (for 6mer)
TFseq_plus = lapply(TFseq_plus, mer.positions, mermask = "NNNNNN")
#Find frequency of each possible 6mer for each position
TFseq_plus = lapply(TFseq_plus, position.frequencies, mermask = "NNNNNN")
save(TFseq_plus, file = 'Tn5_TF_position_frequencies_6mer_plus.Rdata')

#Do this for minus strand sequences
system('mkdir ../minus')
load('TFseq_minus.Rdata')
#Make all sequences upper case
TFseq_minus = lapply(TFseq_minus, function(x) {toupper(x$V7)})
#Break sequences into 5bp chunks (for 6mer)
TFseq_minus = lapply(TFseq_minus, mer.positions, mermask = "NNNNNN")
#Find frequency of each possible 6mer for each position
TFseq_minus = lapply(TFseq_minus, position.frequencies, mermask = "NNNNNN")
save(TFseq_minus, file = 'Tn5_TF_position_frequencies_6mer_minus.Rdata')
```

Calculate all 7-mer frequencies for each position in motif sequences

```
library(data.table)
source('Tn5_Bias_Functions.R')
system('mkdir 7mer')
setwd('../7mer')
options(scipen = 100)
system('mkdir plus')
setwd('plus')
#
load('TFseq_plus.Rdata')
#Break sequences into 5bp chunks (for 7mer)
TFseq_plus = lapply(TFseq_plus, mer.positions, mermask = "NNNNNNN")
#Find frequency of each possible 7mer for each position
TFseq_plus = lapply(TFseq_plus, position.frequencies, mermask = "NNNNNNN")
save(TFseq_plus, file = 'Tn5_TF_position_frequencies_7mer_plus.Rdata')

#Do this for minus strand sequences
system('mkdir ../minus')
load('TFseq_minus.Rdata')
#Break sequences into 5bp chunks (for 7mer)
TFseq_minus = lapply(TFseq_minus, mer.positions, mermask = "NNNNNNN")
#Find frequency of each possible 7mer for each position
TFseq_minus = lapply(TFseq_minus, position.frequencies, mermask = "NNNNNNN")
save(TFseq_minus, file = 'Tn5_TF_position_frequencies_7mer_minus.Rdata')
```

Calculate all spaced 3-mer frequencies for each position in motif sequences

```
library(data.table)
source('Tn5_Bias_Functions.R')
system('mkdir spaced_3mer')
setwd('spaced_3mer')
system('mkdir plus')
setwd('plus')
options(scipen = 100)
#Determine the unique spacings of 3mers in 46bp
factorfiles <- read.table('../../all_possible_spaced_3mers_46bp.txt', skip = 1)
```

```
factorfiles = factorfiles[,1]
unique_factorfiles_spacing = factorfiles
unique_factorfiles_spacing = strsplit(unique_factorfiles_spacing, 'NNN')
unique_factorfiles = NULL
for (i in 1:length(unique_factorfiles_spacing)) {
  unique_factorfiles[i] = unique_factorfiles_spacing[[i]][2]
}
unique_factorfiles_spacing = gsub('^X', 'NNNX', unique_factorfiles)
unique_factorfiles_spacing = gsub('X$', 'XNNN', unique_factorfiles_spacing)
unique_factorfiles_spacing = unique(unique_factorfiles_spacing)
rm(unique_factorfiles)
################################################################
load('../../5mer/plus/TFseq_plus.Rdata')
for (i in 1:length(unique_factorfiles_spacing)) {
  TFseq_plus_s = lapply(TFseq_plus, mer.positions,
                        mermask = unique_factorfiles_spacing[i])
  TFseq_plus_s = lapply(TFseq_plus_s, position.frequencies,
                        mermask = unique_factorfiles_spacing[i])
  save(TFseq_plus_s, file = paste(unique_factorfiles_spacing[i],
                                  '_TFseq_plus_posfreqs.Rdata', sep = ''))
}
################################################################
system('mkdir ../minus')
setwd('../minus')
load('../../5mer/minus/TFseq_minus.Rdata')
for (i in 1:length(unique_factorfiles_spacing)) {
  TFseq_minus_s = lapply(TFseq_minus, mer.positions,
                         mermask = unique_factorfiles_spacing[i])
  TFseq_minus_s = lapply(TFseq_minus_s, position.frequencies,
                         mermask = unique_factorfiles_spacing[i])
  save(TFseq_minus_s, file = paste(unique_factorfiles_spacing[i],
                                   '_TFseq_minus_posfreqs.Rdata', sep = ''))
}
```

6.3 Multiply each k-mer frequency by the inverse of its scaling factors

Multiply 5-mer frequencies by their scale factors, for each TF

```
library(data.table)
source('Tn5_Bias_Functions.R')
library('parallel')
library('doParallel')
setwd('5mer')
options(scipen = 100)

#Import kmer counts and compute scale factors for each....
factorfiles <- list.files('../Tn5_data')
factorfiles = grep('_fulldat_scale_factors.txt', factorfiles)
gsub_factorfiles = gsub('NC', 'N', factorfiles)
gsub_factorfiles = gsub('CN', 'N', gsub_factorfiles)
factorfiles = factorfiles[grep('NNNNN', gsub_factorfiles)]
gsub_factorfiles = gsub_factorfiles[grep('NNNNN', gsub_factorfiles)]
factorfiles = factorfiles[-grep('NNNNNN', gsub_factorfiles)]
rm(gsub_factorfiles)
scalefactors <- vector(mode = "list", length = length(factorfiles))
names(scalefactors) <- factorfiles

for (i in 1:length(factorfiles)) {
  scalefactors[[i]] <- scalefactor.func(paste('../Tn5_data',
                                              factorfiles[i], sep = ''))
}
```

```
##5mer Plus scale factor by k-mer frequency multiplication
setwd('plus')
load('Tn5_TF_position_frequencies_5mer_plus.Rdata')

registerDoParallel(3)
TF_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_plus))
names(TF_scalefactors_kmerfreq) = names(TFseq_plus)

foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar%
  {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                                                kmerfrequency = TFseq_plus[[i]],
                                                                tfname = names(TFseq_plus[i]))}
################################################################################
##5mer minus scale factor by k-mer frequency multiplication
setwd('../minus')
load('Tn5_TF_position_frequencies_5mer_minus.Rdata')


registerDoParallel(3)
TF_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_minus))
names(TF_scalefactors_kmerfreq) = names(TFseq_minus)

foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar%
  {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                                                kmerfrequency = TFseq_minus[[i]],
                                                                tfname = names(TFseq_minus[i]))}
```

Multiply 6-mer frequencies by their scale factors, for each TF

```
library(data.table)
source('Tn5_Bias_Functions.R')
library('parallel')
library('doParallel')
setwd('6mer')
options(scipen = 100)

#Import kmer counts and compute scale factors for each....
factorfiles <- list.files('../Tn5_data')
factorfiles = grep('_fulldat_scale_factors.txt', factorfiles)
gsub_factorfiles = gsub('NC', 'N', factorfiles)
gsub_factorfiles = gsub('CN', 'N', gsub_factorfiles)
factorfiles = factorfiles[grep('NNNNNN', gsub_factorfiles)]
gsub_factorfiles = gsub_factorfiles[grep('NNNNNN', gsub_factorfiles)]
factorfiles = factorfiles[-grep('NNNNNNN', gsub_factorfiles)]
rm(gsub_factorfiles)


scalefactors <- vector(mode = "list", length = length(factorfiles))
names(scalefactors) <- factorfiles

for (i in 1:length(factorfiles)) {
  scalefactors[[i]] <- scalefactor.func(paste('../Tn5_data',
                                              factorfiles[i], sep = ''))
}

##6mer Plus scale factor by k-mer frequency multiplication
setwd('plus')
load('Tn5_TF_position_frequencies_6mer_plus.Rdata')

registerDoParallel(3)
TF_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_plus))
names(TF_scalefactors_kmerfreq) = names(TFseq_plus)
```

```
foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar%
  {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                                                kmerfrequency = TFseq_plus[[i]],
                                                                tfname = names(TFseq_plus[i]))}
################################################################################
##6mer minus scale factor by k-mer frequency multiplication
setwd('../minus')
load('Tn5_TF_position_frequencies_6mer_minus.Rdata')


registerDoParallel(3)
TF_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_minus))
names(TF_scalefactors_kmerfreq) = names(TFseq_minus)

foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar%
  {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                                                kmerfrequency = TFseq_minus[[i]],
                                                                tfname = names(TFseq_minus[i]))}
```

Multiply 7-mer frequencies by their scale factors, for each TF

```
library(data.table)
source('Tn5_Bias_Functions.R')
library('parallel')
library('doParallel')
setwd('7mer')
options(scipen = 100)

#Import kmer counts and compute scale factors for each....
factorfiles <- list.files('../Tn5_data')
factorfiles = grep('_fulldat_scale_factors.txt', factorfiles)
gsub_factorfiles = gsub('NC', 'N', factorfiles)
gsub_factorfiles = gsub('CN', 'N', gsub_factorfiles)
factorfiles = factorfiles[grep('NNNNNNN', gsub_factorfiles)]
rm(gsub_factorfiles)

scalefactors <- vector(mode = "list", length = length(factorfiles))
names(scalefactors) <- factorfiles

for (i in 1:length(factorfiles)) {
  scalefactors[[i]] <- scalefactor.func(paste('../Tn5_data',
                                              factorfiles[i], sep = ''))
}

##7mer Plus scale factor by k-mer frequency multiplication
setwd('plus')
load('Tn5_TF_position_frequencies_7mer_plus.Rdata')

registerDoParallel(3)
TF_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_plus))
names(TF_scalefactors_kmerfreq) = names(TFseq_plus)

foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar%
  {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                                                kmerfrequency = TFseq_plus[[i]],
                                                                tfname = names(TFseq_plus[i]))}
################################################################################
##7mer minus scale factor by k-mer frequency multiplication
setwd('../minus')
load('Tn5_TF_position_frequencies_7mer_minus.Rdata')

registerDoParallel(3)
TF_scalefactors_kmerfreq <- vector(mode = "list", length = length(TFseq_minus))
```

```
names(TF_scalefactors_kmerfreq) = names(TFseq_minus)

foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar%
  {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.kmerfrequency(scalefactors = scalefactors,
                                                                kmerfrequency = TFseq_minus[[i]],
                                                                tfname = names(TFseq_minus[i]))}
```

Multiply spaced 3-mer frequencies by their scale factors, for each TF. As this will require generating close to 1000 masks, this script writes a script for each possible spacing of both 3-mers within the 46bp span. When ran, these scripts will multiply 1/scale factor by the k-mer frequencies for a given spacing. This means that spacings of the two 3-mers which are wider will have proportionally less runs.

```
library(data.table)
source('Tn5_Bias_Functions.R')
library('bigWig')
options(scipen = 100)

#Import kmer counts and compute scale factors for each....
factorfiles <- list.files('/PATH/TO/spaced_3mer_scalefactors/')
unique_factorfiles_spacing = substr(factorfiles, 5, nchar(factorfiles) -26)
unique_factorfiles_spacing = gsub('C', '', unique_factorfiles_spacing)
unique_factorfiles_spacing = strsplit(unique_factorfiles_spacing, 'NNN')
unique_factorfiles = NULL
for (i in 1:length(unique_factorfiles_spacing)) {
  unique_factorfiles[i] = unique_factorfiles_spacing[[i]][2]
}
unique_factorfiles_spacing = gsub('^X', 'NNNX', unique_factorfiles)
unique_factorfiles_spacing = gsub('X$', 'XNNN', unique_factorfiles_spacing)
unique_factorfiles_spacing = unique(unique_factorfiles_spacing)
rm(unique_factorfiles)

scalefactors <- vector(mode = "list", length = length(factorfiles))
names(scalefactors) <- factorfiles

for (i in 1:length(factorfiles)) {
  scalefactors[[i]] <- scalefactor.func(paste('/PATH/TO/spaced_3mer_scalefactors/',
                                              factorfiles[i], sep = ''))
}

#Want scalefactor masks to have Xs instead of Ns
for (i in 1:length(scalefactors)) {
  scalefactors[[i]]$V2  = gsub('N', 'X', scalefactors[[i]]$V2)
}
#Want to match scalefactor with its kmerfreq..
scalefactor_names = substr(names(scalefactors), 5, 51)
scalefactor_names = gsub('C', '', scalefactor_names)

TFseq_files = list.files('/PATH/TO/spaced_3mer/plus')

#Here we make a 'key' for each spacing of 3mers. Each key lists the spacings of k-mer frequency to be
#multiplied by their respective scale factors
setwd('../sfkf_keys')
for (i in 1:length(TFseq_files)) {
  sfkf_key = names(scalefactors)[grep(substr(TFseq_files[i], 1,
                                             nchar(TFseq_files[i])-26), scalefactor_names)]
  save(sfkf_key, file = paste(substr(TFseq_files[i], 1,
                                     nchar(TFseq_files[i])-26), '_sfkf_key.Rdata', sep = ''))
}
#This section writes a script for each possible spacing of 3-mers within the window
setwd('../Rivanna_scripts')
for (i in 1:length(TFseq_files)) {
writeLines(c("library(data.table)",
             "source('../scripts/Tn5_Bias_Functions.R')",
```

```
            "library('parallel')",
            "library('foreach')",
            "library('doParallel')",
            paste("load('../posfreqs/plus/", TFseq_files[i], "')", sep = ''),
            paste("load('../sfkf_keys/", paste(substr(TFseq_files[i],
                                                      1, nchar(TFseq_files[i])-26),
                                               '_sfkf_key.Rdata', sep = ''), "')", sep = ''),
            "load('../Tn5_spaced3mer_scalefactors.Rdata')",
            "setwd('../plus_Rivanna_output')",
            "options(scipen = 100)",
            "registerDoParallel(20)",
            "TF_scalefactors_kmerfreq <- vector(mode = 'list', length = length(TFseq_plus_s))",
            "names(TF_scalefactors_kmerfreq) = names(TFseq_plus_s)",
            "foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar% {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.km
            kmerfrequency = TFseq_plus_s[[i]], tfname = names(TFseq_plus_s[i]))}",
            "rm(TFseq_plus_s)",
            "setwd('../minus_Rivanna_output')",
            paste("load('../posfreqs/minus/", gsub('plus_', 'minus_', TFseq_files[i]), "')", sep = ''),
            "TF_scalefactors_kmerfreq <- vector(mode = 'list', length = length(TFseq_minus_s))",
            "names(TF_scalefactors_kmerfreq) = names(TFseq_minus_s)",
            "foreach (i = 1:length(TF_scalefactors_kmerfreq)) %dopar% {TF_scalefactors_kmerfreq[[i]] = scalefactor.by.km
            kmerfrequency = TFseq_minus_s[[i]], tfname = paste(names(TFseq_minus_s[i]), sep = ''))}"
           ),
          paste(substr(TFseq_files[i], 1, nchar(TFseq_files[i])-26), "_sfkf_Rivanna.R", sep = ''))
}
#This section writes a slurm script to run each of the above scripts
for (i in 1:length(TFseq_files)) {
  writeLines(c("#!/bin/bash",
               "#SBATCH --time=3:00:00",
               "#SBATCH -A shefflab",
               "#SBATCH -n 20",
               "#SBATCH -p standard",
               "#SBATCH --mem=9000",
               paste("#SBATCH -o Tn5_", substr(TFseq_files[i],
               1, nchar(TFseq_files[i])-26), "_sfkf.out", sep = ''),
               "module load gcc/9.2.0",
               "module load intel/18.0",
               "module load intelmpi/18.0",
               "module load R/4.0.3",
               paste("Rscript ", substr(TFseq_files[i], 1,
                        nchar(TFseq_files[i])-26), "_sfkf_Rivanna.R", sep = "")),
             paste(substr(TFseq_files[i], 1,
                          nchar(TFseq_files[i])-26), "_sfkf_Rivanna.slurm", sep = ''))
}
```

Sum the plus spaced 3-mer scale factor * k-mer frequency output for each position, for each TF

```
library(data.table)
library('parallel')
library('foreach')
library('doParallel')
source('../scripts/Tn5_Bias_Functions.R')
load('../Tn5_spaced3mers_factorfiles.Rdata')
registerDoParallel(20)
setwd('../sfkf_sum')
options(scipen = 100)

RDSlist = list.files('../plus_Rivanna_output')
TF_plus_scalefactors_kmerfreq <- vector(mode = "list", length = length(RDSlist))
TF_plus_scalefactors_kmerfreq = foreach (i = 1:length(RDSlist)) %dopar% {
  RDSstore = readRDS(paste( '../plus_Rivanna_output/', RDSlist[i], sep = ''))
  RDSstore = scaledkmerfreq.sum.plus(RDSstore)
  return(RDSstore)}
```

```
names(TF_plus_scalefactors_kmerfreq) = RDSlist

save(TF_plus_scalefactors_kmerfreq,
     file = 'Tn5_TF_NEW_plus_pre_input_spaced3mers.Rdata')
```

Sum the minus spaced 3-mer scale factor * k-mer frequency output for each position, for each TF

```
library(data.table)
library('parallel')
library('foreach')
library('doParallel')
source('../scripts/Tn5_Bias_Functions.R')
load('../Tn5_spaced3mers_factorfiles.Rdata')
registerDoParallel(20)
setwd('../sfkf_sum')
options(scipen = 100)

RDSlist = list.files('../minus_Rivanna_output')
TF_minus_scalefactors_kmerfreq <- vector(mode = "list", length = length(RDSlist))
TF_minus_scalefactors_kmerfreq = foreach (i = 1:length(RDSlist)) %dopar% {
  RDSstore = readRDS(paste( '../minus_Rivanna_output/', RDSlist[i], sep = ''))
  RDSstore = scaledkmerfreq.sum.minus(RDSstore)
  return(RDSstore)}

names(TF_minus_scalefactors_kmerfreq) = RDSlist

save(TF_minus_scalefactors_kmerfreq,
     file = 'Tn5_TF_NEW_minus_pre_input_spaced3mers.Rdata')
```

6.4 Use 5/6/7/ spaced 3-mer inverse scale factor * k-mer frequency output to predict unbiased traces. Use this as input for a rule ensemble model

```
library(data.table)
source('Tn5_Bias_Functions.R')
library('pre')
setwd('../output')
options(scipen = 100)
Sys.setenv(PATH=paste(Sys.getenv("PATH"), "/Users/guertinlab/opt/anaconda3/bin", sep=":"))
################################################################################
train = c( 'ASCL1', 'CLOCK', 'CTCF', 'DLX2', 'EBF1', 'EGR1', 'ELF1', 'FOS',
           'FOXA1', 'FOXK2', 'GATA2', 'GLIS1', 'HSF1', 'IRF1', 'JUN', 'LEF1',
           'MEIS2', 'MLX', 'MYC', 'PPARG','RUNX1', 'SPIB', 'SRY', 'STAT1', 'TGIF1')
test = c('AR', 'ATF3', 'CEBPB', 'E2F1', 'ESR1', 'FERD3L', 'HOXC12', 'MAX', 'MEF2A',
         'MGA', 'NFATC3', 'NR2F2', 'POU3F1', 'REST', 'Six3', 'SP1', 'USF1', 'TEAD1')

#Load in 5mer Rules Ensemble input and add other inputs from here:
load('../RE_input/Tn5_TF_Plus_pre_input_5mers.Rdata')
Plus_pred_input = Plus_pred_input[order(names(Plus_pred_input))]
#Remove test data
Plus_pred_input = Plus_pred_input[which(substr(names(Plus_pred_input),
                                1, nchar(names(Plus_pred_input))-8) %in% train)]
#Trim 5mer RE input to match smaller widths:
Plus_pre_input = vector(mode = 'list', length = length(Plus_pred_input))
for (i in 1:length(Plus_pred_input)) {
 for (p in 1:length(Plus_pred_input[[i]])) {
   Plus_pre_input[[i]][[p]] = Plus_pred_input[[i]][[p]][c(which(Plus_pred_input[[i]][[p]]$x >= -73 &
                                                          Plus_pred_input[[i]][[p]]$x <= 57)),1]
   names(Plus_pre_input)[i] = names(Plus_pred_input[i])
   Plus_pre_input[[i]][[p]] = as.data.frame(Plus_pre_input[[i]][[p]])
```

```
 }
  Plus_pre_input[[i]] = as.data.frame(Plus_pre_input[[i]])
  names(Plus_pre_input[[i]]) = names(Plus_pred_input[[i]])
}
#load in unscaled composites and reduce the size to the predicted
#area (last 4 bases are unmappable because of 5mer)
load('../data/Tn5_plus_unscaled_RM_composites.Rdata')
Tn5_unscaled_composites =
  Tn5_unscaled_composites[which(substr(Tn5_unscaled_composites$factor,
                                       1, nchar(Tn5_unscaled_composites$factor)-8) %in% train),]
Tn5_unscaled_composites =
  Tn5_unscaled_composites[-c(which(Tn5_unscaled_composites$x < - 73 |  Tn5_unscaled_composites$x > 57)),]
Tn5_pre_data_plus = cbind(Tn5_unscaled_composites, do.call(rbind, Plus_pre_input))
rm(Plus_pre_input, Plus_pred_input)
#Next, load in 6mers...
load('../RE_input/Tn5_TF_Plus_pre_input_6mers.Rdata')
Plus_pred_input = Plus_pred_input[order(names(Plus_pred_input))]
Plus_pred_input =
  Plus_pred_input[which(substr(names(Plus_pred_input), 1, nchar(names(Plus_pred_input))-8) %in% train)]
#Trim 6mer RE input to match smaller widths:
Plus_pre_input = vector(mode = 'list', length = length(Plus_pred_input))
for (i in 1:length(Plus_pred_input)) {
  for (p in 1:length(Plus_pred_input[[i]])) {
    Plus_pre_input[[i]][[p]] = Plus_pred_input[[i]][[p]][c(which(Plus_pred_input[[i]][[p]]$x >= -73 &
                                                                 Plus_pred_input[[i]][[p]]$x <= 57)),1]
    names(Plus_pre_input)[i] = names(Plus_pred_input[i])
    Plus_pre_input[[i]][[p]] = as.data.frame(Plus_pre_input[[i]][[p]])
  }
  Plus_pre_input[[i]] = as.data.frame(Plus_pre_input[[i]])
  names(Plus_pre_input[[i]]) = names(Plus_pred_input[[i]])
}
#Append 6mers:
Tn5_pre_data_plus = cbind(Tn5_pre_data_plus, do.call(rbind, Plus_pre_input))
rm(Plus_pre_input, Plus_pred_input)
#Next, load in 7mers...
load('../RE_input/Tn5_TF_Plus_pre_input_7mers.Rdata')
Plus_pred_input = Plus_pred_input[order(names(Plus_pred_input))]
Plus_pred_input = Plus_pred_input[which(substr(names(Plus_pred_input),
                                               1, nchar(names(Plus_pred_input))-8) %in% train)]
#Trim 7mer RE input to match smaller widths:
Plus_pre_input = vector(mode = 'list', length = length(Plus_pred_input))
for (i in 1:length(Plus_pred_input)) {
  for (p in 1:length(Plus_pred_input[[i]])) {
    Plus_pre_input[[i]][[p]] = Plus_pred_input[[i]][[p]][c(which(Plus_pred_input[[i]][[p]]$x >= -73 &
                                                                 Plus_pred_input[[i]][[p]]$x <= 57)),1]
    names(Plus_pre_input)[i] = names(Plus_pred_input[i])
    Plus_pre_input[[i]][[p]] = as.data.frame(Plus_pre_input[[i]][[p]])
  }
  Plus_pre_input[[i]] = as.data.frame(Plus_pre_input[[i]])
  names(Plus_pre_input[[i]]) = names(Plus_pred_input[[i]])
}
#Append 7mers:
Tn5_pre_data_plus = cbind(Tn5_pre_data_plus, do.call(rbind, Plus_pre_input))
rm(Plus_pre_input, Plus_pred_input)
#Next, load in spaced 3mers...
load('../RE_input/Tn5_TF_Plus_pre_input_spaced3mers.Rdata')
Plus_pred_input = Plus_pred_input[order(names(Plus_pred_input))]
Plus_pred_input = Plus_pred_input[which(substr(names(Plus_pred_input),
                                               1, nchar(names(Plus_pred_input))-8) %in% train)]
#Trim 7mer RE input to match smaller widths:
Plus_pre_input = vector(mode = 'list', length = length(Plus_pred_input))
for (i in 1:length(Plus_pred_input)) {
  for (p in 1:length(Plus_pred_input[[i]])) {
    Plus_pre_input[[i]][[p]] = Plus_pred_input[[i]][[p]][c(which(Plus_pred_input[[i]][[p]]$x >= -73 &
```

```
                                                                    Plus_pred_input[[i]][[p]]$x <= 57)),1]
    names(Plus_pre_input)[i] = names(Plus_pred_input[i])
    Plus_pre_input[[i]][[p]] = as.data.frame(Plus_pre_input[[i]][[p]])

  }
  Plus_pre_input[[i]] = as.data.frame(Plus_pre_input[[i]])
  names(Plus_pre_input[[i]]) = names(Plus_pred_input[[i]])
}
#Append spaced 3mers:
Tn5_pre_data_plus = cbind(Tn5_pre_data_plus, do.call(rbind, Plus_pre_input))
rm(Plus_pre_input, Plus_pred_input, Tn5_unscaled_composites)
##Repeat this process for minus strand data
#Load in original 5mer Rules Ensemble input and add other inputs from here:
load('../RE_input/Tn5_TF_minus_pre_input_5mers.Rdata')
minus_pred_input = minus_pred_input[order(names(minus_pred_input))]
#Remove test data
minus_pred_input = minus_pred_input[which(substr(names(minus_pred_input),
                                                 1, nchar(names(minus_pred_input))-9) %in% train)]
#Trim 5mer RE input to match smaller widths:
minus_pre_input = vector(mode = 'list', length = length(minus_pred_input))
for (i in 1:length(minus_pred_input)) {
  for (p in 1:length(minus_pred_input[[i]])) {
    minus_pre_input[[i]][[p]] = minus_pred_input[[i]][[p]][c(which(minus_pred_input[[i]][[p]]$x >= -73 &
                                                                   minus_pred_input[[i]][[p]]$x <= 57)),1]
    names(minus_pre_input)[i] = names(minus_pred_input[i])
    minus_pre_input[[i]][[p]] = as.data.frame(minus_pre_input[[i]][[p]])
  }
  minus_pre_input[[i]] = as.data.frame(minus_pre_input[[i]])
  names(minus_pre_input[[i]]) = names(minus_pred_input[[i]])
}

#load in unscaled composites and reduce the size to the predicted
#area (last 4 bases are unmappable because of 5mer)
load('../data/Tn5minus_unscaled_compositelist.Rdata')
Tn5minus_unscaled_composites = do.call(rbind, Tn5minus_unscaled_compositelist)
Tn5minus_unscaled_composites$est = Tn5minus_unscaled_composites$est/0.00309736

Tn5minus_unscaled_composites = Tn5minus_unscaled_composites[which(substr(Tn5minus_unscaled_composites$factor,
                                                 1, nchar(Tn5minus_unscaled_composites$factor)-9) %in% train),]
Tn5minus_unscaled_composites = Tn5minus_unscaled_composites[-c(which(Tn5minus_unscaled_composites$x < - 73 |
                                                                     Tn5minus_unscaled_composites$x > 57)),]

Tn5_pre_data_minus = cbind(Tn5minus_unscaled_composites, do.call(rbind, minus_pre_input))
rm(minus_pre_input, minus_pred_input, Tn5minus_unscaled_compositelist, Tn5minus_unscaled_composites)
#Next, load in 6mers...
load('../RE_input/Tn5_TF_minus_pre_input_6mers.Rdata')
minus_pred_input = minus_pred_input[order(names(minus_pred_input))]
minus_pred_input = minus_pred_input[which(substr(names(minus_pred_input),
                                                 1, nchar(names(minus_pred_input))-9) %in% train)]
#Trim 6mer RE input to match smaller widths:
minus_pre_input = vector(mode = 'list', length = length(minus_pred_input))
for (i in 1:length(minus_pred_input)) {
  for (p in 1:length(minus_pred_input[[i]])) {
    minus_pre_input[[i]][[p]] = minus_pred_input[[i]][[p]][c(which(minus_pred_input[[i]][[p]]$x >= -73 &
                                                                   minus_pred_input[[i]][[p]]$x <= 57)),1]
    names(minus_pre_input)[i] = names(minus_pred_input[i])
    minus_pre_input[[i]][[p]] = as.data.frame(minus_pre_input[[i]][[p]])

  }
  minus_pre_input[[i]] = as.data.frame(minus_pre_input[[i]])
  names(minus_pre_input[[i]]) = names(minus_pred_input[[i]])
}
#Append 6mers:
Tn5_pre_data_minus = cbind(Tn5_pre_data_minus, do.call(rbind, minus_pre_input))
```

```
rm(minus_pre_input, minus_pred_input)
#Next, load in 7mers...
load('../RE_input/Tn5_TF_minus_pre_input_7mers.Rdata')
minus_pred_input = minus_pred_input[order(names(minus_pred_input))]
minus_pred_input = minus_pred_input[which(substr(names(minus_pred_input), 1,
                                                 nchar(names(minus_pred_input))-9) %in% train)]
#Trim 7mer RE input to match smaller widths:
minus_pre_input = vector(mode = 'list', length = length(minus_pred_input))
for (i in 1:length(minus_pred_input)) {
  for (p in 1:length(minus_pred_input[[i]])) {
    minus_pre_input[[i]][[p]] = minus_pred_input[[i]][[p]][c(which(minus_pred_input[[i]][[p]]$x >= -73 &
                                                                   minus_pred_input[[i]][[p]]$x <= 57)),1]
    names(minus_pre_input)[i] = names(minus_pred_input[i])
    minus_pre_input[[i]][[p]] = as.data.frame(minus_pre_input[[i]][[p]])

  }
  minus_pre_input[[i]] = as.data.frame(minus_pre_input[[i]])
  names(minus_pre_input[[i]]) = names(minus_pred_input[[i]])
}
#Append 7mers:
Tn5_pre_data_minus = cbind(Tn5_pre_data_minus, do.call(rbind, minus_pre_input))
rm(minus_pre_input, minus_pred_input)
#Next, load in spaced 3mers...
load('../RE_input/Tn5_TF_minus_pre_input_spaced3mers.Rdata')
minus_pred_input = minus_pred_input[order(names(minus_pred_input))]
minus_pred_input = minus_pred_input[which(substr(names(minus_pred_input),
                                                 1, nchar(names(minus_pred_input))-9) %in% train)]
#Trim spaced 3mer RE input to match smaller widths:
minus_pre_input = vector(mode = 'list', length = length(minus_pred_input))
for (i in 1:length(minus_pred_input)) {
  for (p in 1:length(minus_pred_input[[i]])) {
    minus_pre_input[[i]][[p]] = minus_pred_input[[i]][[p]][c(which(minus_pred_input[[i]][[p]]$x >= -73 &
                                                                   minus_pred_input[[i]][[p]]$x <= 57)),1]
    names(minus_pre_input)[i] = names(minus_pred_input[i])
    minus_pre_input[[i]][[p]] = as.data.frame(minus_pre_input[[i]][[p]])

  }
  minus_pre_input[[i]] = as.data.frame(minus_pre_input[[i]])
  names(minus_pre_input[[i]]) = names(minus_pred_input[[i]])
}
#Append spaced 3mers:
Tn5_pre_data_minus = cbind(Tn5_pre_data_minus, do.call(rbind, minus_pre_input))
rm(minus_pre_input, minus_pred_input)
###Combine minus and plus models together:
Tn5_RE_input = rbind(Tn5_pre_data_plus, Tn5_pre_data_minus)
#Remove columns 1,3,4,5
Tn5_RE_input = Tn5_RE_input[,-c(1,3,4,5)]
sapply(Tn5_RE_input, function(x) sum(is.na(x)))
##############################################################################
#Make linear model
set.seed(42)
Tn5_linear_model = pre(est ~  ., data = Tn5_RE_input, type = 'linear', sampfrac = 0.5,
                       verbose = TRUE, ntrees = 50000,
                       use.grad = FALSE, nfolds = 5, winsfrac = 0, maxdepth = 3L,
                       learnrate = 0.1, intercept = FALSE)
save(Tn5_linear_model, file = 'Tn5_Linear_model.Rdata')
load('Tn5_Linear_model.Rdata')
##############################################################################
#print(Tn5_pre_model, penalty.par.val = Tn5_pre_model$glmnet.fit$lambda.min)
#Plot variable importances for PRE model:
imps = importance(Tn5_linear_model, penalty.par.val =
                    Tn5_linear_model$glmnet.fit$lambda.min, abbreviate = FALSE)
base_imps = imps$baseimps
##Take top 10% of masks (94) for Rules Ensemble:
```

```
RE_masks = base_imps$rule[1:94]
write.table(RE_masks, file = 'RulesEnsemble_top10_masks.txt', row.names = FALSE,
            quote = FALSE, sep = '\n', col.names = FALSE)

#Rm all of importance dfs
rm(imps, base_imps, Tn5_pre_data_minus, Tn5_pre_data_plus)
###Create data frame with only top 10%:
Tn5_RE_input = as.data.frame(Tn5_RE_input)
Tn5_RE_train = Tn5_RE_input[,c(1, which(colnames(Tn5_RE_input) %in% RE_masks))]
save(Tn5_RE_train, file = 'Tn5_RE_input_top10masks.Rdata')
#Make Prediction using linear+rules PRE model
set.seed(42)
Tn5_RE_model = pre(est ~  ., data = Tn5_RE_train, type = 'both',
                   sampfrac = 0.5, verbose = TRUE, ntrees = 100000,
                       use.grad = FALSE, nfolds = 10, winsfrac = 0,
                   maxdepth = 3L, learnrate = 0.1, intercept = FALSE)

save(Tn5_RE_model, file = 'Tn5_RE_model.Rdata')
load('Tn5_RE_model.Rdata')
#Graph prediction
pre_train_predict <- predict(Tn5_RE_model, type = "link",
                         penalty.par.val = Tn5_RE_model$glmnet.fit$lambda.min)
pdf(file = 'Tn5_pre_predict_min_TRAIN.pdf', width = 18, height = 8)
plot(1:628, Tn5_RE_train$est[1:628], type = 'l', xlab = '', ylab = '')
  lines(1:628,pre_train_predict[1:628], col = 'red')
  legend('topleft', legend=c("Unscaled", "RE Prediction"),
         col=c("black", "red"), lty=1, cex=0.8)
dev.off()
#Graph FOXA1 Prediction
pdf(file = 'Tn5_pre_predict_EGR1.pdf', width = 12, height = 8)
plot(918:1045, Tn5_RE_train$est[918:1045], type = 'l', lwd=2.0, xlab = '', ylab = '')
  lines(918:1045,pre_train_predict[918:1045], col = 'red', lwd=2.0)
  legend('topleft', legend=c("Unscaled", "RE Prediction"),
         col=c("black", "red"), lty=1, cex=0.8)
dev.off()
#Write tables for each desired model coefficients/rules
print(Tn5_RE_model, penalty.par.val = Tn5_RE_model$glmnet.fit$lambda.min)
Tn5_RE_model_coef <- coef(Tn5_RE_model, penalty.par.val = Tn5_RE_model$glmnet.fit$lambda.min)
Tn5_RE_model_coef = Tn5_RE_model_coef[which(Tn5_RE_model_coef$coefficient != 0),]
write.table(Tn5_RE_model_coef[,2:3], file = 'Tn5_Rules_Ensemble_model.txt',
            quote = FALSE, sep = ',', row.names = FALSE)
```

6.5 Apply rule ensemble model to scale seqOutBias output

```
library(data.table)
options(scipen = 100)
system('mkdir ../multiscale')
setwd('../multiscale')
#
################################################################################
#Function to import rule ensemble model into R script
import.rules = function(input) {
  rulesdf = data.table(read.table(input, sep = ',', header = TRUE))
  intcoeff = NULL
  lincoeff = NULL
  rulescoeff = NULL
  for (i in 1:nrow(rulesdf)) {
    intcoeff[i] = fifelse(rulesdf[i,description] == 1, rulesdf[i,coefficient], 0)
  }
  intcoeff = intcoeff[!(intcoeff==0)]
  for (i in 1:nrow(rulesdf)) {
```

```
    lincoeff[i] = fifelse(!grepl(">|<|=",rulesdf[i, description]),
                          paste('(',rulesdf[i, description], '*',
                                rulesdf[i, coefficient], ')', sep = ''),paste(''))}
  lincoeff = lincoeff[!(lincoeff=='')]
  linpaste = capture.output(cat(lincoeff, sep = ' + '))
  for (i in 1:nrow(rulesdf)) {
    rulescoeff[i] = fifelse(grepl(">|<|=",rulesdf[i, description]),
                            paste('fifelse(', rulesdf[i, description],
                                  ', ', rulesdf[i, coefficient], ', 0)', sep = ''), '')}
  rulescoeff = rulescoeff[!(rulescoeff=='')]
  rulespaste = capture.output(cat(rulescoeff, sep = ' + '))
  output = paste(intcoeff, linpaste, rulespaste, sep = ' + ')
  return(output)
}
################################################################################
#Load in unscaled bedgraph in order to get read depth
unscaled_bed = fread('/PATH/TO/hg38_Tn5_C1gDNA_unscaled.bedGraph')
unscaled_read_depth = sum(unscaled_bed$V4)
rm(unscaled_bed)
#Load in bedgraph which contains all input mask bedgraphs
x <- fread('../top10_bw/top10_Tn5_RulesEnsemble_union.bedGraph')
#Load in column names for the above bedgraph
masknames = fread('../top10_bw/Tn5_allmasks_names.txt', header = FALSE)
masknames = masknames$V1
masknames = substr(masknames, 5, nchar(masknames))
masknames = c('chr', 'start', 'stop', masknames)
colnames(x) = masknames
#Import rule ensemble model
prerules = import.rules('../output/Tn5_Rules_Ensemble_model.txt')
prerules
x[, pre := (RULE ENSEMBLE MODEL HERE)]
pre_read_depth = sum(x$pre)
#Determine ratio of unscaled read depth to rule ensemble read depth
RDS = unscaled_read_depth/pre_read_depth
#Scale rule ensemble read depth to unscaled read depth
x[, pre_RDS := pre*RDS ]
colnames(x)[99]
write.table(x[,c(1:3,99)], file = 'Tn5_RE.bedGraph', col.names = FALSE, row.names=FALSE, sep = '\t', quote=FALSE)
#bedGraphToBigWig Tn5_RE.bedGraph hg38.fa.chrom.sizes Tn5_RE.bigWig
```

6.6 Analyze scaled rule ensemble output

```
library(bigWig)
library(lattice)
library(data.table)
library(ggplot2)
source('../scripts/composite_functions_mk2.R')
################################################################################
train = c( 'ASCL1', 'CLOCK', 'CTCF', 'DLX2', 'EBF1', 'EGR1', 'ELF1', 'FOS', 'FOXA1',
           'FOXK2', 'GATA2', 'GLIS1', 'HSF1', 'IRF1', 'JUN', 'LEF1', 'MEIS2', 'MLX',
           'MYC', 'PPARG','RUNX1', 'SPIB', 'SRY', 'STAT1', 'TGIF1')
test = c('AR', 'ATF3', 'CEBPB', 'E2F1', 'ESR1', 'FERD3L', 'HOXC12', 'MAX', 'MEF2A',
         'MGA', 'NFATC3', 'NR2F2', 'POU3F1', 'REST', 'Six3', 'SP1', 'USF1', 'TEAD1')
system('mkdir ../analysis')
setwd('../analysis')
################################################################################
#Load in genomic locations determined using FIMO
Motifs <- list.files('../FIMO')
Motiflist <- vector('list', length(Motifs))
for (i in 1:length(Motifs)) {
  Motiflist[[i]] <- FIMO.to.BED(paste('../FIMO/', Motifs[i], sep = ''))
```

```
}
names(Motiflist) = Motifs
#Combine the minus and plus FIMO motifs
minus_Motifs = Motiflist[seq(1, 88, 2)]
plus_Motifs = Motiflist[-c(seq(1, 88, 2))]
Motiflist = vector('list', length(plus_Motifs))
for (i in 1:length(plus_Motifs)) {
  Motiflist[[i]] = rbind(plus_Motifs[[i]], minus_Motifs[[i]])
}
names(Motiflist) = substr(names(plus_Motifs), 1 , nchar(names(plus_Motifs))-22)
rm(plus_Motifs, minus_Motifs)
#Make a vector of each motif's length (for graphing)
Motiflen = NULL
for (i in 1:length(Motiflist)) {
  Motiflen[i] = Motiflist[[i]]$end - Motiflist[[i]]$start
}
names(Motiflen) = names(Motiflist)
rm(Motifs)
save(Motiflen, file = 'Motif_lengths.Rdata')
################################################################################
#Make composite plots of unscaled values
BWs <- c('../data/hg38_Tn5_C1gDNA_unscaled.bigWig')
compositelist <- vector('list', length(Motiflist))
for (i in 1:length(Motiflist)) {
  compositelist[[i]] = BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                            upstream = 100, downstream = 100,
                            factor = paste(substr(Motifs[i], 1, nchar(Motifs[i])-3)),
                            group = 'unscaled', ATAC = TRUE)
}
names(compositelist) = names(Motiflist)
for (i in 1:length(compositelist)) {
  compositelist[[i]]$factor <- names(compositelist)[[i]]
}
Tn5_unscaled_compositelist = compositelist
save(Tn5_unscaled_compositelist, file = 'Tn5_unscaled_compositelist.Rdata')
################################################################################################
#Make composite plots of XXXXXXXXXXXXXXXXXXXXNNNCNNNNXXXXXXXXXXXXXXXXXXX
BWs <- c('../data/Tn5_XXXXXXXXXXXXXXXXXXXXNNNCNNNNXXXXXXXXXXXXXXXXXXX.bw')
#Reorder BWs to match Motifs:
compositelist <- vector('list', length(Motiflist))
for (i in 1:length(Motiflist)) {
  compositelist[[i]] = BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                        upstream = 100, downstream = 100,
                                        factor = names(Motiflist[i]),
                                        group = 'NNNCNNNN', ATAC = TRUE)
}
names(compositelist) = names(Motiflist)
for (i in 1:length(compositelist)) {
  compositelist[[i]]$factor <- names(compositelist)[[i]]
}
Tn5_NNNCNNNN_compositelist = compositelist
save(Tn5_NNNCNNNN_compositelist, file = 'Tn5_NNNCNNNN_compositelist.Rdata')
################################################################################################
#Make composite plots of rule ensemble output
BWs <- c('../multiscale/Tn5_RE.bigWig')
#Reorder BWs to match Motifs:
compositelist <- vector('list', length(Motiflist))
for (i in 1:length(Motiflist)) {
  compositelist[[i]] = BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                        upstream = 100, downstream = 100,
                                        factor = names(Motiflist[i]),
                                        group = 'Rule Ensemble', ATAC = TRUE)
}
names(compositelist) <- names(Motiflist)
```

```
for (i in 1:length(compositelist)) {
  compositelist[[i]]$factor <- names(compositelist)[[i]]
}
Tn5_RE_compositelist = compositelist
save(Tn5_RE_compositelist, file = 'Tn5_RE_compositelist.Rdata')
################################################################################################
#Remove train data from output and unscaled
test_Tn5_RE_RM_compositelist =
  Tn5_RE_compositelist[which(names(Tn5_RE_compositelist) %in% test)]
test_unscaled_compositelist =
  Tn5_unscaled_compositelist[which(names(Tn5_unscaled_compositelist) %in% test)]
test_Tn5_NNNCNNNN_compositelist =
  Tn5_NNNCNNNN_compositelist[which(names(Tn5_NNNCNNNN_compositelist) %in% test)]
#Plot difference b/t avg baseline and corrected...
#trim composites to 20bp around center
test_Tn5_RE = do.call(rbind, test_Tn5_RE_RM_compositelist)
test_Tn5_RE$group = 'Rule Ensemble'
test_unscaled = do.call(rbind, test_unscaled_compositelist)
test_unscaled$group = 'Unscaled'
test_NNNCNNNN = do.call(rbind, test_Tn5_NNNCNNNN_compositelist)
test_NNNCNNNN$group = 'seqOutBias'
################################################################################################
#Violin Hack
panel.violin.hack <-
  function (x, y, box.ratio = 1, box.width = box.ratio/(1 + box.ratio),
            horizontal = TRUE, alpha = plot.polygon$alpha, border =
              plot.polygon$border,
            lty = plot.polygon$lty, lwd = plot.polygon$lwd, col = plot.polygon
            $col,
            varwidth = FALSE, bw = NULL, adjust = NULL, kernel = NULL,
            window = NULL, width = NULL, n = 50, from = NULL, to = NULL,
            cut = NULL, na.rm = TRUE, ...){
    if (all(is.na(x) | is.na(y)))
      return()
    x <- as.numeric(x)
    y <- as.numeric(y)
    plot.polygon <- trellis.par.get("plot.polygon")
    darg <- list()
    darg$bw <- bw
    darg$adjust <- adjust
    darg$kernel <- kernel
    darg$window <- window
    darg$width <- width
    darg$n <- n
    darg$from <- from
    darg$to <- to
    darg$cut <- cut
    darg$na.rm <- na.rm
    my.density <- function(x) {
      ans <- try(do.call("density", c(list(x = x), darg)),
                 silent = TRUE)
      if (inherits(ans, "try-error"))
        list(x = rep(x[1], 3), y = c(0, 1, 0))
      else ans
    }
    numeric.list <- if (horizontal)
      split(x, factor(y))
    else split(y, factor(x))
    levels.fos <- as.numeric(names(numeric.list))
    d.list <- lapply(numeric.list, my.density)
    dx.list <- lapply(d.list, "[[", "x")
    dy.list <- lapply(d.list, "[[", "y")
    max.d <- sapply(dy.list, max)
    if (varwidth)
```

```
      max.d[] <- max(max.d)
    xscale <- current.panel.limits()$xlim
    yscale <- current.panel.limits()$ylim
    height <- box.width
    if (horizontal) {
      for (i in seq_along(levels.fos)) {
        if (is.finite(max.d[i])) {
          pushViewport(viewport(y = unit(levels.fos[i],
                                         "native"), height = unit(height, "native"),
                                yscale = c(max.d[i] * c(-1, 1)), xscale = xscale))
          grid.polygon(x = c(dx.list[[i]], rev(dx.list[[i]])),
                       y = c(dy.list[[i]], -rev(dy.list[[i]])),
                       default.units = "native",
                       # this is the point at which the index is added
                       gp = gpar(fill = col[i], col = border, lty = lty,
                                 lwd = lwd, alpha = alpha))
          popViewport()
        }
      }
    }
    else {
      for (i in seq_along(levels.fos)) {
        if (is.finite(max.d[i])) {
          pushViewport(viewport(x = unit(levels.fos[i],
                                         "native"), width = unit(height, "native"),
                                xscale = c(max.d[i] * c(-1, 1)), yscale = yscale))
          grid.polygon(y = c(dx.list[[i]], rev(dx.list[[i]])),
                       x = c(dy.list[[i]], -rev(dy.list[[i]])),
                       default.units = "native",
                       # this is the point at which the index is added
                       gp = gpar(fill = col[i], col = border, lty = lty,
                                 lwd = lwd, alpha = alpha))
          popViewport()
        }
      }
    }
    invisible()
  }

###Subset out baseline values (motif +/-10) and single nucleotide (motif)
all_test_composites = rbind(test_Tn5_RE, test_unscaled, test_NNNCNNNN)

baseline_frame = NULL
baseline_store = NULL
singlenuc_frame = NULL
singlenuc_store = NULL
for (i in 1:length(Motiflen)) {
  baseline_store = all_test_composites[which(all_test_composites$factor ==
           names(Motiflen)[i] & all_test_composites$x > ((Motiflen[i]/2)+10) |
           all_test_composites$factor == names(Motiflen)[i] &
             all_test_composites$x < -((Motiflen[i]/2)+10)),]
  baseline_frame = rbind(baseline_frame, baseline_store)
  singlenuc_store = all_test_composites[which(all_test_composites$factor ==
           names(Motiflen)[i] & all_test_composites$x < ((Motiflen[i]/2)+10) &
             all_test_composites$x > - ((Motiflen[i]/2)+10)),]
  singlenuc_frame = rbind(singlenuc_frame, singlenuc_store)
}
#Compute baseline and single nucleotide log2 ratios
baseline_log = NULL
singlenuc_log = NULL
for (i in 1:length(baseline_frame$x)) {
  baseline_log[i] = log2(baseline_frame$est[i] / 0.006140407)
}
for (i in 1:length(singlenuc_frame$factor)) {
```

```
  singlenuc_log[i] = log2(singlenuc_frame$est[i] / 0.006140407)
}

baseline_log = data.frame(baseline_log)
baseline_log[,2] = baseline_frame$group
colnames(baseline_log) = c('Difference', 'Treatment')
baseline_log$Treatment = factor(baseline_log$Treatment,
                                levels = c('Unscaled', 'Rule Ensemble', 'seqOutBias'))
singlenuc_log = data.frame(singlenuc_log)
singlenuc_log[,2] = singlenuc_frame$factor
singlenuc_log[,3] = singlenuc_frame$group
colnames(singlenuc_log) = c('Difference', 'Factor', 'Treatment')
singlenuc_log$Treatment = factor(singlenuc_log$Treatment,
                                 levels = c('Unscaled', 'Rule Ensemble', 'seqOutBias'))
#plot baselines:
pdf('Tn5_baseline_log2_comparison.pdf', useDingbats = FALSE, width=10.83, height=6)
trellis.par.set(box.umbrella = list(lty = 1, col="#93939380", lwd=2),
                box.rectangle = list(col = '#93939380', lwd=1.6),
                plot.symbol = list(col='#93939380', lwd=1.6, pch ='.'))
###############################################################################
print(bwplot(Difference ~ Treatment , data = baseline_log,
             between=list(y=0.5, x = 0.5),
             scales=list(x=list(draw=TRUE),rot = 45, alternating=c(1,1,1,1),cex=1,font=1),
             #xlab = '',
             ylim = c(-2.5, 2.5),
             #   main = "Pause Index (PI) Ratio",
             ylab =expression("log"[2]~"(deviation from random cleavage)"),
             horizontal =FALSE,  col= 'black',
             aspect = 2,
             par.settings=list(par.xlab.text=list(cex=1.2,font=1),
                               par.ylab.text=list(cex=1.2,font=1),
                               par.main.text=list(cex=1.2, font=1),
                               plot.symbol = list(col='black', lwd=0, pch =19, cex = 0.0)),
             strip = function(..., which.panel, bg) {
               bg.col = c("grey90")
               strip.default(..., which.panel = which.panel,
                             bg = rep(bg.col, length = which.panel)[which.panel])
             },
             panel = function(..., box.ratio, col) {
               panel.abline(h = 0, col = 'grey45', lty = 2)
               panel.violin.hack(..., col = c("#FF7E78", "#FF5FD2", "#50FFE6"),
                                 varwidth = FALSE, box.ratio = box.ratio, outer = FALSE)
               panel.stripplot(..., col='#54545380', do.out=FALSE,
                               jitter.data=TRUE, amount = 0.2, pch = 16)
               panel.bwplot(..., pch = '|', do.out = FALSE)

             }))
dev.off()
save(baseline_log, file = 'Figure5_Tn5_baseline_log2_comparison.Rdata')
#Calculate baseline means
baseline_means = NULL
for (i in 1:length(unique(baseline_log$Treatment))) {
  baseline_means[i] = mean(baseline_log[which(baseline_log$Treatment ==
                                                unique(baseline_log$Treatment)[i]),1])
}
names(baseline_means) = unique(baseline_log$Treatment)
baseline_means
#Calculate baseline ranges
baseline_range = data.frame()
for (i in 1:length(unique(baseline_log$Treatment))) {
  baseline_range[i,1] = max(baseline_log[which(baseline_log$Treatment ==
                                                 unique(baseline_log$Treatment)[i]),1])
  baseline_range[i,2] = min(baseline_log[which(baseline_log$Treatment ==
                                                unique(baseline_log$Treatment)[i]),1])
```

```
}
names(baseline_means) = unique(baseline_log$Treatment)
baseline_means
baseline_range[,3] = unique(baseline_log$Treatment)
colnames(baseline_range) = c('max', 'min', 'Treatment')
baseline_range
#Plot each TF (singlenuc)
pdf('Tn5_singlenuc_log2_comparison.pdf', useDingbats = FALSE, width=10.83, height=6)

trellis.par.set(box.umbrella = list(lty = 1, col="#93939380", lwd=2),
                box.rectangle = list(col = '#93939380', lwd=1.6),
                plot.symbol = list(col='#93939380', lwd=1.6, pch ='.'))
###############################################################################
print(bwplot(Difference ~ Treatment | Factor , data = singlenuc_log,
             between=list(y=0.5, x = 0.5),
             scales=list(x=list(draw=TRUE),rot = 45, alternating=c(1,1,1,1),cex=1,font=1),
             #xlab = '',
             ylim = c(-4.5, 3.5),
             #   main = "Pause Index (PI) Ratio",
             ylab =expression("log"[2]~"(deviation from random cleavage)"),
             horizontal =FALSE,  col= 'black',
             aspect = 2,
             par.settings=list(par.xlab.text=list(cex=1.2,font=1),
                               par.ylab.text=list(cex=1.2,font=1),
                               par.main.text=list(cex=1.2, font=1),
                               plot.symbol = list(col='black', lwd=0, pch =19, cex = 0.0)),
             strip = function(..., which.panel, bg) {
               bg.col = c("grey90")
               strip.default(..., which.panel = which.panel,
                             bg = rep(bg.col, length = which.panel)[which.panel])
             },
             panel = function(..., box.ratio, col) {
               panel.abline(h = 0, col = 'grey45', lty = 2)
               panel.violin.hack(..., col = c("#FF7E78", "#FF5FD2", "#50FFE6"),
                                 varwidth = FALSE, box.ratio = box.ratio, outer = FALSE)
               panel.stripplot(..., col='#54545380', do.out=FALSE,
                               jitter.data=TRUE, amount = 0.2, pch = 16)
               panel.bwplot(..., pch = '|', do.out = FALSE)
             }))
dev.off()

save(singlenuc_log, file = 'Figure5_Tn5_singlenuc_log2_comparison.Rdata')
#Calculate single nucleotide means
singlenuc_means = NULL
singlenuc_range = data.frame()
for (i in 1:length(unique(singlenuc_log$Treatment))) {
  singlenuc_means[i] = mean(singlenuc_log[which(singlenuc_log$Treatment ==
                                                  unique(singlenuc_log$Treatment)[i]),1])
  singlenuc_range[i,1] = max(singlenuc_log[which(singlenuc_log$Treatment ==
                                                   unique(singlenuc_log$Treatment)[i]),1])
  singlenuc_range[i,2] = min(singlenuc_log[which(singlenuc_log$Treatment ==
                                                   unique(singlenuc_log$Treatment)[i]),1])
}
names(singlenuc_means) = unique(singlenuc_log$Treatment)
singlenuc_means
singlenuc_range[,3] = unique(singlenuc_log$Treatment)
colnames(singlenuc_range) = c('max', 'min', 'Treatment')
#Plot overlays:
###############################################################################
plot.composites = function(dat, ylabel = '', x_axis_range = min(dat$x):max(dat$x),
                           pdf_name = 'PLEASE_SET_FILE_NAME',
                           xlabel = '', striplabel = TRUE, legend = TRUE,
                           motifline = FALSE, Motiflen = 10,
                           figwidth = 2.5, figheight=3,
```

```
                          indexlist = NULL, layoutgrid = NULL,
                          col.lines = c("#0000FF", "#FF0000", "#00000090",
                                        rgb(0.1,0.5,0.05,1/2), rgb(0,0,0,1/2),
                                        rgb(1/2,0,1/2,1/2), rgb(0,1/2,1/2,1/2),
                                        rgb(1/2,1/2,0,1/2)),
                          fill.poly = c(rgb(0,0,1,1/4),
                                        rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),
                                        rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4))) {
  require(lattice)
  dat = dat[which(dat$x %in% x_axis_range),]
  pdf(paste(pdf_name, '.pdf', sep = ''), width= figwidth, height= figheight)
  print(xyplot(est ~ x|factor, group = group, data = dat, strip = striplabel,
               type = 'l', as.table = TRUE,
               scales=list(x=list(cex=1.5,relation = "free", axs ="i"),
                           y =list(cex=1.5, relation="free", tick.number=4)),
               col = col.lines,
               auto.key = if (legend == TRUE)
               {list(points=F, lines=T, cex=0.8)} else{},
               par.settings = list(strip.background=list(col="#00000000"),
                                   strip.border = list(col = 'transparent'),
                                   superpose.symbol = list(pch = c(16),
                                                           col=col.lines,
                                                           cex =0.5),
                                   superpose.line = list(col = col.lines,
                                                         lwd=c(2),
                                                         lty = c(1))),
               cex.axis=1.0,
               par.strip.text=list(cex=2, font=1, col='black'),
               ylim = c(0, 0.05),
               aspect=1.0,
               between=list(y=0.5, x=0.5),
               lwd=2,
               ylab = list(label = paste(ylabel), cex =1.75),
               xlab = list(label = paste(xlabel), cex =1.75),
               index.cond = indexlist,
               layout = layoutgrid,
               panel = function(x, y, ...) {
                 panel.xyplot(x, y, ...)
                 panel.abline(h = 0.006140407, lty =3, lwd = 10.0, col = 'red')

               }
  ))
  dev.off()
}
################################################################################
#Plot seqOutBias overlay
OLD_Tn5_RE_overlay = Tn5_NNNCNNNN_compositelist[which(names(Tn5_NNNCNNNN_compositelist) %in% test)]
OLD_Tn5_RE_overlay = do.call(rbind, OLD_Tn5_RE_overlay)
OLD_Tn5_RE_overlay$correction = OLD_Tn5_RE_overlay$factor
OLD_Tn5_RE_overlay$factor = OLD_Tn5_RE_overlay$group
OLD_Tn5_RE_overlay$group = OLD_Tn5_RE_overlay$correction
OLD_Tn5_RE_overlay$factor = 'seqOutBias'
plot.composites(OLD_Tn5_RE_overlay, legend = FALSE,
                pdf_name = 'seqOutBias_NNNCNN_Tn5_RE_overlay',
                ylabel = 'Cut Frequency',
                xlabel = 'Distance from Motif Center',
                motifline = FALSE, Motiflen = 0, figwidth = 12, figheight = 8, x_axis_range = -100:100)
#Plot rule ensemble overlay
RE_Tn5_overlay = Tn5_RE_compositelist[which(names(Tn5_RE_compositelist) %in% test)]
RE_Tn5_overlay = do.call(rbind, RE_Tn5_overlay)
RE_Tn5_overlay$correction = RE_Tn5_overlay$factor
RE_Tn5_overlay$factor = RE_Tn5_overlay$group
RE_Tn5_overlay$group = RE_Tn5_overlay$correction
RE_Tn5_overlay$factor = 'Rule Ensemble'
```

```
plot.composites(RE_Tn5_overlay, legend = FALSE,
                pdf_name = '567mer_spaced3mer_Tn5_RE_overlay',
                ylabel = 'Cut Frequency',
                xlabel = 'Distance from Motif Center',
                motifline = FALSE, Motiflen = 0, figwidth = 12, figheight = 8,
                x_axis_range = -100:100)
#Plot unscaled overlay
unscaled_Tn5_overlay = Tn5_unscaled_compositelist[which(names(Tn5_unscaled_compositelist) %in% test)]
unscaled_Tn5_overlay = do.call(rbind, unscaled_Tn5_overlay)
unscaled_Tn5_overlay$correction = unscaled_Tn5_overlay$factor
unscaled_Tn5_overlay$factor = unscaled_Tn5_overlay$group
unscaled_Tn5_overlay$group = unscaled_Tn5_overlay$correction
unscaled_Tn5_overlay$factor = 'Unscaled'
plot.composites(unscaled_Tn5_overlay, legend = FALSE,
                pdf_name = 'unscaled_Tn5_overlay',
                ylabel = 'Cut Frequency',
                xlabel = 'Distance from Motif Center',
                motifline = FALSE, Motiflen = 0, figwidth = 12, figheight = 8,
                x_axis_range = -100:100)
```

7 Figure 7. Tn5 preference for high GC content regions creates a distinct bias from single nucleotide sequence

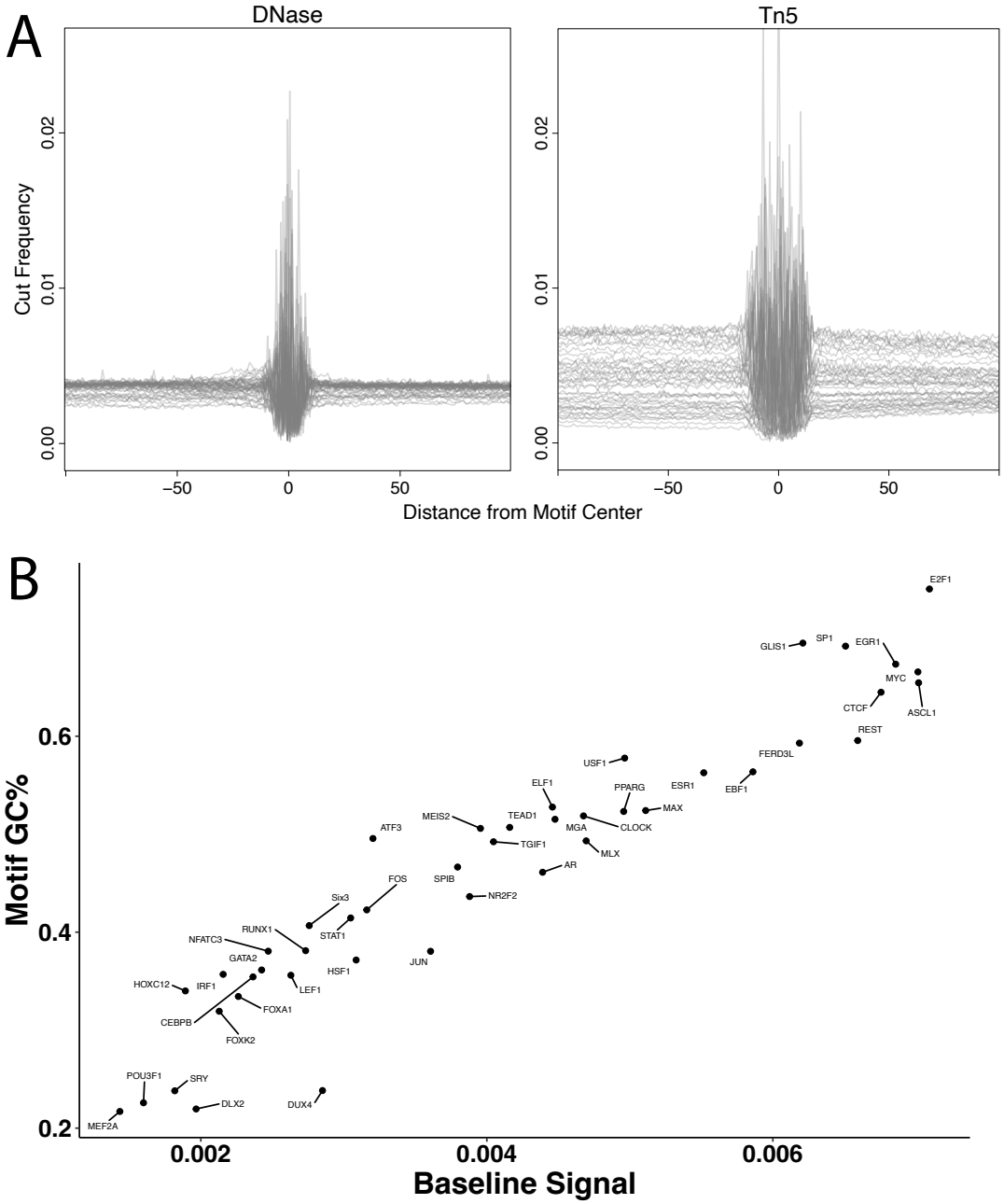


Figure 4. Tn5 preference for high GC content regions creates a distinct bias from single nucleotide sequence. (A) Unscaled overlays of DNase and Tn5 composite profiles for 44 transcription factors surveyed. While sequence bias affects the local signal of each plot, baseline signal far away from each motif is highly variable as well. (C) Plot of motif GC content vs. baseline signal. In all examined transcription factor motifs, a high degree of positive correlation was found between baseline signal and motif GC content.

Figure 15: Figure 4. Tn5 preference for high GC content regions creates a distinct bias from single nucleotide sequence

7.1 Plotting Tn5 unscaled composites of TF motifs

```
library(bigWig)
library(zoo)
library(lattice)
library(ggseqlogo)
library(data.table)
library(matrixStats)
library(gridExtra)
source('Figure3_functions.R')
setwd('../Tn5_unscaled_composites')
########################### For only plus unscaled composites
Motifs <- list.files('../FIMO')
Motiflist <- vector('list', length(Motifs))
for (i in 1:length(Motifs)) {
  Motiflist[[i]] <- FIMO.to.BED(paste('../FIMO/',Motifs[i], sep = ''))
}
names(Motiflist) = paste(substr(Motifs, 1, nchar(Motifs)-14), sep = '')
Motifs = names(Motiflist)


BWs <- c('path/to/Tn5_C1_gDNA_rep1_plus.bigWig')
compositelist <- vector('list', length(Motifs))
for (i in 1:length(Motiflist)) {
  compositelist[[i]] = BED.query.bigWig(Motiflist[[i]], paste(BWs), paste(BWs),
                                        upstream = 100, downstream = 100, factor = paste(substr(Motifs[i],
                                        1, nchar(Motifs[i])-3)), group = 'Unscaled', ATAC = FALSE)
}


names(compositelist) <- Motifs

for (i in 1:length(compositelist)) {
  compositelist[[i]]$factor <- names(compositelist)[[i]]
}


#plot uncorrected values
for (i in 1:length(compositelist)) {
  plot.composites(compositelist[[i]], legend = FALSE,
                  pdf_name = paste('Tn5_unscaled_400k_',
                                   paste(names(compositelist[i])), '_composite', sep = ''),
                  ylabel = 'Cut Frequency',
                  xlabel = 'Distance from Motif Center',
                  motifline = TRUE, Motiflen = Motiflen[i]
  )}


Tn5_unscaled_compositelist = compositelist
save(Tn5_unscaled_compositelist, file = 'Tn5_unscaled_compositelist.Rdata')
```

7.2 Plotting all unscaled DNase and Tn5 composites overlaid on the same plot

```
library(lattice)
################################################################################
################################################################################
################################################################################
################################################################################
plot.composites = function(dat, ylabel = '', pdf_name = 'PLEASE_SET_FILE_NAME',
                           xlabel = '', striplabel = TRUE, legend = TRUE,
                           motifline = FALSE, Motiflen = 10,
```

```
                          figwidth = 2.5, figheight=3,
                          indexlist = NULL, layoutgrid = NULL,
                          col.lines = c('#8080804D'),
                          fill.poly = c(rgb(0,0,1,1/4),
                                        rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),
                                        rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4))) {
  require(lattice)
  pdf(paste(pdf_name, '.pdf', sep = ''), width= figwidth, height= figheight)
  print(xyplot(est ~ x|factor, group = group, data = dat, strip = striplabel,
               type = 'l', as.table = TRUE,
               scales=list(x=list(cex=1.5,relation = "free", axs ="i"),
                           y =list(cex=1.5, relation="free", tick.number=4)),
               col = col.lines,
               auto.key = if (legend == TRUE)
               {list(points=F, lines=T, cex=0.8)} else{},
               par.settings = list(strip.background=list(col="#00000000"),
                                   strip.border = list(col = 'transparent'),
                                   superpose.symbol = list(pch = c(16),
                                                           col=col.lines,
                                                           cex =0.5),
                                   superpose.line = list(col = col.lines,
                                                         lwd=c(2),
                                                         lty = c(1))),
               cex.axis=1.0,
               par.strip.text=list(cex=2, font=1, col='black'),
               ylim = c(0, 0.025),
               aspect=1.0,
               between=list(y=0.5, x=0.5),
               lwd=2,
               ylab = list(label = paste(ylabel), cex =1.75),
               xlab = list(label = paste(xlabel), cex =1.75),
               index.cond = indexlist,
               layout = layoutgrid,
               panel = function(x, y, ...) {
                 panel.xyplot(x, y, ...)
                 #panel.abline(h = 0, lty =1, lwd = 1.0, col = '#A9A9A932')
                 if (motifline == TRUE)
                 {panel.abline(v = Motiflen/2, lty = 2, col = "red")} else{}
                 if (motifline == TRUE)
                 {panel.abline(v = -Motiflen/2, lty = 2, col = "red")} else{}
               }
  ))
  dev.off()
}

################################################################################
#Plot DNase unscaled composite overlay
#Change DNase group to a given factor, then change the factor column to 'DNase'
load('/path/to/DNase_unscaled_compositelist.Rdata')
DNase_unscaled_composites_overlay = do.call(rbind, DNase_unscaled_compositelist)
DNase_unscaled_composites_overlay$group = DNase_unscaled_composites_overlay$factor
DNase_unscaled_composites_overlay$factor = 'DNase'

plot.composites(DNase_unscaled_composites_overlay, legend = FALSE,
                pdf_name = 'Figure4A_DNase_unscaled_overlay',
                ylabel = 'Cut Frequency',
                xlabel = 'Distance from Motif Center',
                motifline = FALSE, Motiflen = 0, figwidth = 12, figheight = 8)

################################################################################
#Plot Tn5 unscaled composite overlay
#Change Tn5 group to a given factor, then change the factor column to 'Tn5'
load('/path/to/Tn5_unscaled_compositelist.Rdata')
Tn5_unscaled_composites_overlay = do.call(rbind, Tn5_unscaled_compositelist)
```

```
Tn5_unscaled_composites_overlay$group = Tn5_unscaled_composites_overlay$factor
Tn5_unscaled_composites_overlay$factor = 'Tn5'

plot.composites(Tn5_unscaled_composites_overlay, legend = FALSE,
                  pdf_name = 'Figure4A_Tn5_unscaled_overlay',
                  ylabel = 'Cut Frequency',
                  xlabel = 'Distance from Motif Center',
                  motifline = FALSE, Motiflen = 0, figwidth = 12, figheight = 8)
```

7.3 Plotting Each TFs baseline signal vs GC content

```
library(data.table)
library(ggplot2)
#################################################################################
load('path/to/Tn5_unscaled_compositelist.Rdata')
#Remove middle 40bp
unscaled_composites = do.call(rbind, Tn5_unscaled_compositelist)
unscaled_composites =
  unscaled_composites[-which(unscaled_composites$x >= -20 & unscaled_composites$x <= 20)]
unscaled_composites$factor = as.factor(unscaled_composites$factor)

#Plot motif GC content vs. baseline GC content
baseline_mean = aggregate(unscaled_composites[, 6], list(unscaled_composites$factor), mean)

#Plot unscaled and PRE output GC content
Motifs <- list.files('/path/to/meme_files/')

Motifs = substr(Motifs, 1, nchar(Motifs)-14)
GCvec <- NULL
for(i in 1:length(Motifs)) {
  memefile <- paste0("/path/to/meme_files/",Motifs[i], ".meme")
  minmeme <- read.csv(memefile, skip = 11, header = FALSE, sep = '')
  minmeme <- minmeme[1:nrow(minmeme)-1,]
  minmeme[,2] <- as.numeric(minmeme[,2])
  minmeme[,1] <- as.numeric(minmeme[,1])
  GCcon <- sum(minmeme[,c(2,3)])/sum(minmeme)
  GCvec[i] <- GCcon
}


baseline_mean[,3] = GCvec


pdf("Figure4_Tn5_unscaled_BaselineVsGCcon.pdf", width=9, height=6)
ggplot(baseline_mean, aes(x = V1, y = V3, label = V4)) +
  geom_point(stat = 'identity') +
  theme_classic() + xlab('Baseline Signal') + ylab('Motif GC%') +
  geom_text_repel(size = 2, stat = 'identity', force_pull = 0.75, force = 20,
                  max.overlaps = 20, seed = 42) +
  theme(axis.text=element_text(size=16, face = 'bold', color = 'black'),
        axis.title=element_text(size=20,face="bold", color = 'black'))
dev.off()
```

8 Functions

Here are all functions used in this investigation, generally called as a single script.

```
#
scalefactor.func <- function(factortable){
```

```
    require(data.table)
    datatable = fread(factortable, skip = 1)
    datatable[, pluscountpercent := datatable[,3]/sum(datatable[,3])]
    datatable[, plusobspercent := datatable[,5]/sum(datatable[,5])]
    datatable[, plusscalefact := round(datatable[,pluscountpercent]/datatable[,plusobspercent], 6)]
    datatable[, minuscountpercent := datatable[,4]/sum(datatable[,4])]
    datatable[, minusobspercent := datatable[,6]/sum(datatable[,6])]
    datatable[, minusscalefact := round(datatable[,minuscountpercent]/datatable[,minusobspercent], 6)]
}
#

#FIMO.to.Bed takes input in FIMO format (1-based start and end) and converts
#it to BED format (0-based start, 1-based end), also increases window by amount specified.
FIMO.to.BED <- function(fimofile, window = 0) {
    require(data.table)
    #load in FIMO file
    FIMOdf <- fread(fimofile, sep = '\t', header = TRUE)
    #shift start position by -1 to make it 0-based
    FIMOdf[,4] = FIMOdf[,4] - 1
    #make location column
    FIMOdf[,location := paste(sequence_name, ':', start, '-', stop, sep = '')]
    #Rearrange columns for BED format
    beddf <- FIMOdf[, c(3,4,5,2,11,6)]
    colnames(beddf) <- c('chr', 'start', 'end', 'gene', 'location', 'strand')
    #CB (central base) determines the central base which the window is based on
    #also used as the 0 position on x-axis
    #even motifs will == 0, odd will == 1
    if ((beddf[1,3] - beddf[1,2])%%2 == 0) {
        CB = data.table(beddf)
        #convert start positions to 1-base for calculating CB
        CB[,start := start + 1]
        #Calculate median ceiling for positive motifs, and median floor value
        # for minus motifs then use this value to determine
        #start and end using upstream/downstream
        CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
        setkey(CB,strand)
        CB[c("-"),median := floor(median)]
        CB[c("+"),median := ceiling(median)]
        CB[,start := median - window]
        CB[,end := median + window]
        #Convert start position back to 0-base
        CB[,start := start - 1]
        CB[,median:=NULL]
    } else {
        CB = data.table(beddf)
        #convert start positions to 1-base for calculating median
        CB[,start := start + 1]
        #Calculate median for each row, then use this value to determine
        #start and end using upstream/downstream
        CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
        CB[,start := median - window]
        CB[,end := median + window]
        #Convert start position back to 0-base
        CB[,start := start - 1]
        CB[,median:=NULL]
    }

    return(CB)
}

#Split up a list of sequences into mers = mermask for each mappable position in the sequence
mer.positions <- function(input, mermask = 'NNNXXXXXNNN') {
    seqdf_mer <- vector(mode = "list", length = nchar(input[1]))
    if (grepl('X', mermask) == TRUE) {
```

```
    mermask_spaces = unlist(strsplit(mermask, 'X'))
    mer1 = nchar(mermask_spaces)[1]
    unmasked = length(which(mermask_spaces == '')) + 1
    mer2 = nchar(mermask_spaces)[length(mermask_spaces)]
    for (i in 1:nchar(input[1])) {
        seqdf_store = data.table()
        seqdf_store[, alltogether := paste(substr(input[1:length(input)], i, i + mer1 - 1),
                                           paste(rep('X', unmasked), collapse = ""),
                                           substr(input[1:length(input)],
                                           i + mer1 + unmasked, i + mer1 + unmasked + mer2 - 1), sep = '')]
        seqdf_mer[[i]] <- seqdf_store$alltogether
    }
    mersize = nchar(mermask)
    names(seqdf_mer) <- paste0("Position_", seq_along(1:nchar(input[1])))
    seqdf_mer <- seqdf_mer[1:(length(seqdf_mer)-mersize+1)]}
    else {
      seqdf_store <- NULL
      mersize = nchar(mermask)
      for (i in 1:nchar(input[1])) {
        seqdf_store <- substr(input[1:length(input)], i, i+ (mersize-1))
        seqdf_mer[[i]] <- seqdf_store
        names(seqdf_mer) <- paste0("Position_", seq_along(1:nchar(input[1])))
      }
      seqdf_mer <- seqdf_mer[1:(length(seqdf_mer)-mersize+1)]
    }
    return(seqdf_mer)
}


#Get mercounts for each possible mer from the output of mer.positions - only for 1 sequence
mer.counts = function(sequence_mer_positions, mermask = 'NNNN') {
  #Create all possible combinations of kmer
  mermask = unlist(strsplit(mermask, split = ''))
  mertable = expand.grid(rep(list(c('A','C','G','T')), length(which(mermask == 'N'))))
  if (length(which(mermask == 'X')) > 0) {
    Xrows = matrix(nrow = nrow(mertable), ncol = length(which(mermask == 'X')))
    Xrows[,c(1:length(which(mermask == 'X')))] = rep('X', nrow(mertable))
    mertable = cbind(mertable[,1:(which(mermask=='X')[1]-1)],
                     Xrows, mertable[,(which(mermask=='X')[1]:ncol(mertable))])
  } else {}
  mertable = data.frame(apply(mertable, 1 , paste, collapse = ""))
  mercount = as.character(sequence_mer_positions)
  merstore = NULL
  #
  for (i in 1:nrow(mertable)) {
    merstore = length(which(mercount %in% mertable[i,1]))
    if (length(merstore) == 0) {
      merstore = 0
    } else {}
    mertable[i,2] = merstore
  }
  colnames(mertable) = c('kmer', 'count')
  return(mertable)
}


#Get position frequencies for each possible mer from the output of mer.positions
position.frequencies = function(sequence_mer_positions, mermask = 'NNNNNNN') {
  #Create all possible combinations of kmer
  mermask = unlist(strsplit(mermask, split = ''))
  mertable = expand.grid(rep(list(c('A','C','G','T')), length(which(mermask == 'N'))))
  if (length(which(mermask == 'X')) > 0) {
  Xrows = matrix(nrow = nrow(mertable), ncol = length(which(mermask == 'X')))
  Xrows[,c(1:length(which(mermask == 'X')))] = rep('X', nrow(mertable))
```

```
  mertable = cbind(mertable[,1:(which(mermask=='X')[1]-1)],
                   Xrows, mertable[,(which(mermask=='X')[1]:ncol(mertable))])
  } else {}
  mertable = data.table(apply(mertable, 1 , paste, collapse = ""))
  posfreq_poslist = vector(mode = "list", length = length(sequence_mer_positions))
  #
  for (p in 1:length(sequence_mer_positions)) {
    posfreq_posstore = data.table(mertable$V1)
    postable = table(sequence_mer_positions[[p]])
    if (length(which(!names(postable) %in% mertable$V1)) > 0) {
    postable = postable[-c(which(!names(postable) %in% mertable$V1))]
    } else {}
    missing_mers = as.character(mertable$V1[which(!mertable$V1 %in% names(postable))])
    append_mers = rep(0, length(missing_mers))
    names(append_mers) = missing_mers
    postable = c(postable, append_mers)
    postable = postable/length(sequence_mer_positions[[1]])
    posfreq_poslist[[p]] <- postable

  }
  names(posfreq_poslist) <- paste0("Position_", seq_along(1:length(posfreq_poslist)))
  return(posfreq_poslist)
}
##Multiply 1/scale_factor by its corresponding kmer frequency for each position in
##the composite
scalefactor.by.kmerfrequency <- function(scalefactors, kmerfrequency, tfname) {
    all_scalefactors = vector(mode = "list", length = length(scalefactors))
    names(all_scalefactors) <- names(scalefactors)
   for (j in 1:length(scalefactors)) {
       scalefactor = scalefactors[[j]]
       scalefactor = scalefactor[order(V2)]
       scaledkfreq = vector(mode = "list", length = length(kmerfrequency))
       names(scaledkfreq) = names(kmerfrequency)
    for (i in 1:length(kmerfrequency)) {
        scaledkfreq[[i]] = data.table(stack(kmerfrequency[[i]]))
        colnames(scaledkfreq[[i]]) = c('kmerfreq', 'kmer')
        scaledkfreq[[i]]$kmer = as.character(scaledkfreq[[i]]$kmer)
        scaledkfreq[[i]] = scaledkfreq[[i]][order(kmer)]
        scaledkfreq[[i]][, mask := scalefactor[,2]]
        scaledkfreq[[i]][, plusscalefreq := (1/(scalefactor[,9])) * kmerfreq ]
        scaledkfreq[[i]][, minusscalefreq := (1/(scalefactor[,12])) * kmerfreq ]

        }
       saveRDS(scaledkfreq, file = paste(tfname, '_',
                       substr(names(scalefactors[j]), 1,
                              nchar(names(scalefactors[j]))-17), '.rds', sep = ''))
       all_scalefactors[[j]] = scaledkfreq
       }
      return(all_scalefactors)
}

##Take sum of each position's scaledkmerfreq (1/scale_factor * kmer frequency) and use to create
##data.table of inputs for each mask for each TF motif
scaledkmerfreq.sum.plus <- function(input) {
  output = data.table(matrix(nrow = length(input)))
  for (i in 1:length(input)) {

    output[i, scaledkmerfreqsumplus := sum(input[[i]][,4])]

  }
  output[,1] <- NULL
  rownames(output) = names(input)
  return(output)
}
```

```
scaledkmerfreq.sum.minus <- function(input) {
  output = data.table(matrix(nrow = length(input)))
  for (i in 1:length(input)) {

    output[i, scaledkmerfreqsumminus := sum(input[[i]][,5])]

  }
  output[,1] <- NULL
  rownames(output) = names(input)
  return(output)
}




################################################################################
################################################################################
#Combine values from mask positions to plot redudant importance values for each position
redundantpos = function(input, labelnum = 2, impnum = 3) {
  input[,labelnum] = as.character(input[,labelnum])
  input[,labelnum] = gsub('C', '', input[,labelnum])
  posimp = vector(mode = 'list', length = nchar(input[1,labelnum]))
  for (i in 1:nchar(input[1,labelnum])) {
    for (p in 1:nrow(input)) {
      posimp[[i]][p] = fifelse(substr(input[p,labelnum],i,i) == 'N', input[p,impnum],0)
    }
  }
  output = data.table(matrix(nrow = length(posimp), ncol = 1))
  for (i in 1:length(posimp)) {
    output[i, possum := sum(posimp[[i]])]
  }
  output = output[,-c(1)]
  output[, position := paste(rep('X',36,sep = ''), collapse = '')]

  for (i in 1:nrow(output)) {
    output[i,]$position = paste(substr(output[i, position],1,i-1),'N',
                                substr(output[i, position],i+1,nchar(output[i, position])), sep = '')
    output[i,]$position = paste(substr(output[i, position],1,(nchar(output[i, position])/2)), 'C',
                                paste(substr(output[i, position],
                                (nchar(output[i, position])/2+1),nchar(output[i, position]))), sep = '')
  }
  return(output)
}


#
#BED.query.bigWig uses the bed coordinates produced by FIMO.to.BED to
#query a supplied bigWig file for the read counts at the specified positions
#In order to create a window around the base of interest, a central base (CB)
#is determined based on whether or not the motif of interest is odd or even
#additionally, if ATAC=FALSE, minus-aligned motifs are shifted +1
#relative to the plus strand to account for DNase etc. cutting between bases
#finally, the mean of each position's read counts is calculated for plotting
BED.query.bigWig <- function(beddf, bwPlus, bwMinus, upstream = 10,
                             downstream = 10,
                             factor = '', group = '', ATAC = TRUE) {
  require(bigWig)
  require(data.table)
  require(matrixStats)
  step = 1
  #load bigWigs
  bw.plus = load.bigWig(bwPlus)
```

```
bw.minus = load.bigWig(bwMinus)
#CB (central base) determines the central base which the window is based on
#also used as the 0 position on x-axis
#even motifs will == 0, odd will == 1
if ((beddf[1,3] - beddf[1,2])%%2 == 0) {
  CB = data.table(beddf)
  #convert start positions to 1-base for calculating CB
  CB[,start := start + 1]
  #Calculate median ceiling for positive motifs, and median floor value
  # for minus motifs then use this value to determine
  #start and end using upstream/downstream
  CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
  setkey(CB,strand)
  CB[c("-"),median := floor(median)]
  CB[c("+"),median := ceiling(median)]
  CB[,start := median - upstream]
  CB[,end := median + downstream]
  #Convert start position back to 0-base
  CB[,start := start - 1]
  CB[,median:=NULL]
} else {
  CB = data.table(beddf)
  #convert start positions to 1-base for calculating median
  CB[,start := start + 1]
  #Calculate median for each row, then use this value to determine
  #start and end using upstream/downstream
  CB[,median := apply(CB[,2:3],1, median, na.rm = TRUE)][]
  CB[,start := median - upstream]
  CB[,end := median + downstream]
  #Convert start position back to 0-base
  CB[,start := start - 1]
  CB[,median:=NULL]
}
#Shift minus-aligned motifs by +1 for DNase etc.
if (ATAC == FALSE) {
  setkey(CB,strand)
  CB[c("-"),start := start + 1]
  CB[c("-"),end := end + 1]
} else {}
#extract read count information from bigWigs, using CB-derived
#window as coordinates
tss.matrix = bed6.step.bpQuery.bigWig(bw.plus, bw.minus , CB,
                                      step = 1, as.matrix=TRUE, follow.strand=TRUE)
tss_correction = sum(tss.matrix)/(nrow(tss.matrix)*ncol(tss.matrix))
#Determine beginning and end of plot
#if not ATAC, peaks are between bases, so offset by -1 and
#add 0.5 to align 'cuts' on 0 at -0.5
if (ATAC == FALSE) {
  coordin.start = (-upstream - 0.5 )
  coordin.end = (downstream - 0.5 )
} else {
  #For tn5, add negative upstream value and downstream value
  coordin.start = (-upstream)
  coordin.end = (downstream)
}
#create coordinates for aligning read counts on composite profile
#also take means of each column from tss.matrix and label each row with
#factor column
composite.lattice = data.table(seq(coordin.start, coordin.end, by = step),
                               colMeans(tss.matrix),
                               factor, group, tss_correction,
                               stringsAsFactors = FALSE)
colnames(composite.lattice) = c('x', 'est', 'factor', 'group', 'correction')
composite.lattice$x = as.numeric(composite.lattice$x)
```

```
  unload.bigWig(bw.plus)
  unload.bigWig(bw.minus)
  return(composite.lattice)
}
#

##Plot.composites takes the composite.lattice object and creates a plot for
#this data, while also allowing a mapping of the original motif width onto the
#plot by specifying a TFlen value (the default is 10).
plot.composites <- function(dat, ylabel = '', pdf_name = 'PLEASE_SET_FILE_NAME',
                            xlabel = '', striplabel = TRUE, legend = TRUE,
                            motifline = FALSE, Motiflen = 10,
                            figwidth = 2.5, figheight=3,
                            indexlist = NULL, layoutgrid = NULL,
                            col.lines = c("#0000FF", "#FF0000", "#00000090",
                                          rgb(0.1,0.5,0.05,1/2), rgb(0,0,0,1/2),
                                          rgb(1/2,0,1/2,1/2), rgb(0,1/2,1/2,1/2),
                                          rgb(1/2,1/2,0,1/2)),
                            fill.poly = c(rgb(0,0,1,1/4),
                                          rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),
                                          rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4))) {
  require(lattice)
  pdf(paste(pdf_name, '.pdf', sep = ''), width= figwidth, height= figheight)
  print(xyplot(est ~ x|factor, group = group, data = dat, strip = striplabel,
               type = 'l', as.table = TRUE,
               scales=list(x=list(cex=0.8,relation = "free", axs ="i"),
                           y =list(cex=0.8, relation="free", tick.number=4)),
               col = col.lines,
               auto.key = if (legend == TRUE)
               {list(points=F, lines=T, cex=0.8)} else{},
               par.settings = list(strip.background=list(col="#00000000"),
                                   strip.border = list(col = 'transparent'),
                                   superpose.symbol = list(pch = c(16),
                                                           col=col.lines,
                                                           cex =0.5),
                                   superpose.line = list(col = col.lines,
                                                         lwd=c(2),
                                                         lty = c(1))),
               cex.axis=1.0,
               par.strip.text=list(cex=0.9, font=1, col='black'),
               aspect=1.0,
               between=list(y=0.5, x=0.5),
               lwd=2,
               ylab = list(label = paste(ylabel), cex =0.8),
               xlab = list(label = paste(xlabel), cex =0.8),
               index.cond = indexlist,
               layout = layoutgrid,
               panel = function(x, y, ...) {
                 panel.xyplot(x, y, ...)
                 #panel.abline(h = 0, lty =1, lwd = 1.0, col = '#A9A9A932')
                 if (motifline == TRUE)
                 {panel.abline(v = Motiflen/2, lty = 2, col = "red")} else{}
                 if (motifline == TRUE)
                 {panel.abline(v = -Motiflen/2, lty = 2, col = "red")} else{}
               }
  ))
  dev.off()
}
#
##Plots seqlogos from PSWMs
plot.seqlogo.func <- function(x, outfile = "PLEASE_SET_FILE_NAME.pdf") {
  require(ggseqlogo)
  w =  0.663 + (ncol(x) + 1)*0.018 + (ncol(x)+2)* .336
  pdf(outfile, useDingbats=FALSE, width=w, height=2.695)
```

```
  print(ggseqlogo(x,  facet = "wrap", font = 'helvetica_bold'))
  dev.off()
}


##Determines nucleotide frequency in a fasta file and writes a text file in meme format
pswm.func.2 <- function(x.ligation, out = 'outfilename', posnum) {
  col.matrix = matrix()
  for (g in 1:posnum){
    itnum = lapply(strsplit(as.character(x.ligation), ''), "[", g)
    if (g == 1) {
      col.matrix = itnum
    } else {
      col.matrix = cbind(col.matrix, itnum)
    }
  }

  a.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "A"))
  t.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "T"))
  c.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "C"))
  g.nuc = sapply(1:posnum, function(x) sum(col.matrix[,x] == "G"))

  pswm = cbind(a.nuc, c.nuc, g.nuc, t.nuc)
  print(pswm)
  outfile = file(paste0(out, '.txt'))
  on.exit(close(outfile))
  writeLines(c("MEME version 4", "ALPHABET= ACGT", "strands: + -", " ",
               "Background letter frequencies (from uniform background):",
               "A 0.29460 C 0.20450 G 0.20540 T 0.29550", paste("MOTIF", out), " ",
               paste("letter-probability matrix: alength= 4 w= ", posnum)), outfile)
  pswm = pswm/rowSums(pswm)
  write.table(pswm, file = paste0(out, '.txt'), append = TRUE, quote=FALSE, row.names =FALSE, col.names = FALSE)
  return(pswm)
}

##Make a DF which contains characters into all uppercase characters
uppercase <- function(tableinput){
  upperdf <- read.table(tableinput, comment.char = '>')
  upperdf[,1] = as.character(upperdf[,1])
  upperdf = data.frame(lapply(upperdf, function(v) {
    if (is.character(v)) return(toupper(v))
    else return(v)}))
}
```