Stempember W3

Awais Choudhry

15/09/2017

library(rtracklayer)

## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##   
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':  
##   
## clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,  
## clusterExport, clusterMap, parApply, parCapply, parLapply,  
## parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, append, as.data.frame, cbind, colnames,  
## do.call, duplicated, eval, evalq, Filter, Find, get, grep,  
## grepl, intersect, is.unsorted, lapply, lengths, Map, mapply,  
## match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,  
## Position, rank, rbind, Reduce, rownames, sapply, setdiff,  
## sort, table, tapply, union, unique, unsplit, which, which.max,  
## which.min

## Loading required package: S4Vectors

##   
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:base':  
##   
## colMeans, colSums, expand.grid, rowMeans, rowSums

## Loading required package: IRanges

## Loading required package: GenomeInfoDb

library(Biostrings)

## Loading required package: XVector

library(parallel)  
library(BSgenome.Hsapiens.UCSC.hg19)

## Loading required package: BSgenome

library(BSgenome.Mmusculus.UCSC.mm9)  
library(ggplot2)  
library(magrittr)  
library(pander)  
library(tibble)  
library(reshape2)  
library(AnnotationHub)  
library(tidyr)

##   
## Attaching package: 'tidyr'

## The following object is masked from 'package:reshape2':  
##   
## smiths

## The following object is masked from 'package:magrittr':  
##   
## extract

## The following object is masked from 'package:S4Vectors':  
##   
## expand

library(scales)  
library(MotifDb)

## See system.file("LICENSE", package="MotifDb") for use restrictions.

##   
## Attaching package: 'MotifDb'

## The following object is masked from 'package:AnnotationHub':  
##   
## query

library(GenomicInteractions)

## Loading required package: InteractionSet

## Loading required package: SummarizedExperiment

## Loading required package: Biobase

## Welcome to Bioconductor  
##   
## Vignettes contain introductory material; view with  
## 'browseVignettes()'. To cite Bioconductor, see  
## 'citation("Biobase")', and for packages 'citation("pkgname")'.

##   
## Attaching package: 'Biobase'

## The following object is masked from 'package:AnnotationHub':  
##   
## cache

## Warning: replacing previous import 'BiocGenerics::Position' by  
## 'ggplot2::Position' when loading 'GenomicInteractions'

##   
## Attaching package: 'GenomicInteractions'

## The following object is masked from 'package:scales':  
##   
## is.trans

library(knitr)  
library(ggbio)

## No methods found in "RSQLite" for requests: dbGetQuery

## Need specific help about ggbio? try mailing   
## the maintainer or visit http://tengfei.github.com/ggbio/

##   
## Attaching package: 'ggbio'

## The following object is masked from 'package:scales':  
##   
## rescale

## The following objects are masked from 'package:ggplot2':  
##   
## geom\_bar, geom\_rect, geom\_segment, ggsave, stat\_bin,  
## stat\_identity, xlim

library(readr)

##   
## Attaching package: 'readr'

## The following object is masked from 'package:scales':  
##   
## col\_factor

#### Generating a heatmap of the distribution of motifs across the genome for human and mouse!!

Mouse inputs: This script imports enhancer, genes and promoter datasets as well as Arx TFBS in the mouse genome. Whereby it then intersects this with the 60way phyloP60 selecting for motifs with 0.5 conservation score or more. Additionally it intersects it with the phastCon score which is 0.85 (because phyloP ranges from -14 to 3 we know that the score ranges from 0 to 17 essentially therefore 15.5 is 0.85 hence we used a 0.85 cut off).

enhancers<-import("~/DataFiles/Enhancer Tracks/Mouse/mouse\_permissive\_enhancers\_phase\_1\_and\_2.bed")  
genes<-import("~/DataFiles/Gene Tracks/Mouse/mm9.bed")  
promotersGenes<-promoters(genes)  
  
arx6merTFBS<-readRDS("~/DataFiles/ChIPseq/Mouse/ARX6mermm9Sites")  
ARXTandem2SpacedTFBS<-readRDS("~/DataFiles/ChIPseq/Mouse/ARXTande2SpacedSites")  
Plaindromic4SpacedTFBS<-readRDS("~/DataFiles/ChIPseq/Mouse/Plaindromic4SpacedTFBS")  
JolmaTFBS<-readRDS("~/DataFiles/ChIPseq/Mouse/JolmaTFBS")  
  
  
ARXMotifModelList<-c("6 Mer"= arx6merTFBS,  
 "Tandem 2 Spaced" = ARXTandem2SpacedTFBS,  
 "Palindromic 4 Spaced" = Plaindromic4SpacedTFBS,  
 "Jolma Model" = JolmaTFBS)  
  
  
ah<-AnnotationHub()

## snapshotDate(): 2016-10-11

mm9tomm10Query<-AnnotationHub::query(ah, c("mm9", "mm10"))  
  
mm9tomm10Chain<-mm9tomm10Query[["AH14596"]]

## loading from cache '/home/a1649239//.AnnotationHub/18691'

mm10tomm9Chain<-mm9tomm10Query[["AH14535"]]

## loading from cache '/home/a1649239//.AnnotationHub/18630'

##Phast Con Scores Same as above  
  
  
phyloPScores<-lapply(ARXMotifModelList, function(x){liftOver(import("~/DataFiles/Conservation/Mouse/mm10.60way.phyloP60way.bw",   
 which= liftOver(x, mm9tomm10Chain)%>%unlist()),   
 mm10tomm9Chain)%>%unlist})

## Discarding unchained sequences: chr1\_random, chr3\_random, chr4\_random, chr5\_random, chr7\_random, chr8\_random, chr9\_random, chr13\_random, chr16\_random, chr17\_random, chrX\_random, chrY\_random, chrUn\_random

## Discarding unchained sequences: chr1\_random, chr8\_random, chr9\_random, chrX\_random, chrY\_random, chrUn\_random

## Discarding unchained sequences: chr1\_random, chr4\_random, chr7\_random, chr8\_random, chr9\_random, chrX\_random, chrY\_random, chrUn\_random

## Discarding unchained sequences: chr1\_random, chr4\_random, chr7\_random, chr8\_random, chr9\_random, chr17\_random, chrX\_random, chrY\_random, chrUn\_random

phyloPAbove0.5<-lapply( phyloPScores,function(x){subset(x, score>=0.5)})  
  
# for(i in 1:4){  
# seqlevels(arxPromotersAndEnhancers[[i]])->seqlevels(phyloPAbove0.5[[i]], force=TRUE)  
# seqlengths(arxPromotersAndEnhancers[[i]])<-seqlengths(phyloPAbove0.5[[i]])  
# }  
  
conservedARXPromotersAndEnhancers<-list()  
for(i in 1:4){  
 conservedARXPromotersAndEnhancers[i]<-subset(ARXMotifModelList[[i]], countOverlaps(ARXMotifModelList[[i]], phyloPAbove0.5[[i]])>=6)  
 }

## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated

PhastConsScores<-lapply(ARXMotifModelList,function(x){liftOver(import("~/DataFiles/Conservation/Mouse/mm10.60way.phastCons.bw", which= liftOver(x, mm9tomm10Chain)%>%unlist()),   
 mm10tomm9Chain)%>%unlist})

## Discarding unchained sequences: chr1\_random, chr3\_random, chr4\_random, chr5\_random, chr7\_random, chr8\_random, chr9\_random, chr13\_random, chr16\_random, chr17\_random, chrX\_random, chrY\_random, chrUn\_random

## Discarding unchained sequences: chr1\_random, chr8\_random, chr9\_random, chrX\_random, chrY\_random, chrUn\_random

## Discarding unchained sequences: chr1\_random, chr4\_random, chr7\_random, chr8\_random, chr9\_random, chrX\_random, chrY\_random, chrUn\_random

## Discarding unchained sequences: chr1\_random, chr4\_random, chr7\_random, chr8\_random, chr9\_random, chr17\_random, chrX\_random, chrY\_random, chrUn\_random

PhastCon0.85<-lapply( PhastConsScores,function(x){subset(x, score>=0.85)})  
  
# for(i in 1:4){  
# seqlevels(arxPromotersAndEnhancers[[i]])->seqlevels(PhastCon0.5[[i]], force=TRUE)  
# seqlengths(arxPromotersAndEnhancers[[i]])<-seqlengths(PhastCon0.5[[i]])  
# }  
  
conservedARXPromotersAndEnhancersPhastCon<-list()  
for(i in 1:4){  
 conservedARXPromotersAndEnhancersPhastCon[i]<-subset(ARXMotifModelList[[i]], countOverlaps(ARXMotifModelList[[i]], PhastCon0.85[[i]])>=6)  
}

## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated

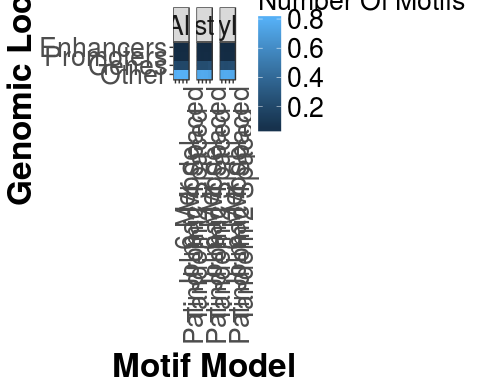
# Human inputs

enhancers<-import("~/DataFiles/Enhancer Tracks/Human/human\_permissive\_enhancers\_phase\_1\_and\_2.bed")  
genes<-import("~/DataFiles/Gene Tracks/Human/hg.bed")  
promotersGenes<-promoters(genes)  
  
arx6merTFBS<-readRDS("~/DataFiles/ChIPseq/Human/ARX6merHg19Sites")  
ARXTandem2SpacedTFBS<-readRDS("~/DataFiles/ChIPseq/Human/ARXTande2SpacedSites")  
Plaindromic4SpacedTFBS<-readRDS("~/DataFiles/ChIPseq/Human/Plaindromic4SpacedTFBS")  
JolmaTFBS<-readRDS("~/DataFiles/ChIPseq/Human/JolmaTFBS")  
  
ARXMotifModelList<-c("6 Mer"= arx6merTFBS,  
 "Tandem 2 Spaced" = ARXTandem2SpacedTFBS,  
 "Palindromic 4 Spaced" = Plaindromic4SpacedTFBS,  
 "Jolma Model" = JolmaTFBS)  
  
  
phyloPScores<-lapply(ARXMotifModelList, function(x){import("~/DataFiles/Conservation/Human/hg19.100way.phyloP100way.bw",  
 which= x)})  
phyloPAbove0.5<-lapply(phyloPScores, function(x){subset(x, score>=0.5)})  
  
conservedARXPromotersAndEnhancers<-list()  
for(i in 1:4){  
 conservedARXPromotersAndEnhancers[i]<-subset(ARXMotifModelList[[i]], countOverlaps(ARXMotifModelList[[i]], phyloPAbove0.5[[i]])>=6)  
}  
  
  
  
  
## PhastCon Motifs  
PhastConScores<-lapply(ARXMotifModelList, function(x){import("~/DataFiles/Conservation/Human/hg19.100way.phastCons.bw",  
 which= x)})  
PhastConAbove0.85<-lapply(phyloPScores, function(x){subset(x, score>=0.85)})  
conservedARXPromotersPhastCon<-list()  
for(i in 1:4){  
 conservedARXPromotersAndEnhancersPhastCon[i]<-subset(ARXMotifModelList[[i]],  
 countOverlaps(ARXMotifModelList[[i]], PhastConAbove0.85[[i]])>=6)  
}

We only overlapped by 6 bp despite the motifs being of variable size as conducting a full overlap of 12 etc resulted in only 0-2 homodimer motifs being present hence we wanted to be more inclusive.

## The script to generate proportions of motifs for genomic location based on motif model and conservation method!

MotifModelGenomicLocation<-lapply(ARXMotifModelList, function(x){  
 dataFrameGenomicLocation<-rbind.data.frame("Enhancers"=subsetByOverlaps(x, enhancers)%>%length()/length(x),  
 "Genes"=subsetByOverlaps(x, genes)%>%length() /length(x) ,  
 "Promoters"=subsetByOverlaps(x, promotersGenes)%>%length()/length(x),  
 "Other"=(length(x)-sum(subsetByOverlaps(x, enhancers)%>%length()/length(x),  
 subsetByOverlaps(x, genes)%>%length(),  
 subsetByOverlaps(x, promotersGenes)%>%length()))/length(x))  
 set\_colnames(dataFrameGenomicLocation,value = "Number Of Motifs")  
})  
  
  
  
PhyloPConservedGenomicLocation<-lapply(conservedARXPromotersAndEnhancers, function(x){  
 dataFrameGenomicLocation<-rbind.data.frame("Enhancers"=subsetByOverlaps(x, enhancers)%>%length()/length(x),  
 "Genes"=subsetByOverlaps(x, genes)%>%length() /length(x) ,  
 "Promoters"=subsetByOverlaps(x, promotersGenes)%>%length()/length(x),  
 "Other"=(length(x)-sum(subsetByOverlaps(x, enhancers)%>%length()/length(x),  
 subsetByOverlaps(x, genes)%>%length(),  
 subsetByOverlaps(x, promotersGenes)%>%length()))/length(x))  
 set\_colnames(dataFrameGenomicLocation,value = "Number Of Motifs")  
})  
  
  
  
  
  
PhastConservedGenomicLocation<-lapply(conservedARXPromotersAndEnhancersPhastCon, function(x){  
 dataFrameGenomicLocation<-rbind.data.frame("Enhancers"=subsetByOverlaps(x, enhancers)%>%length()/length(x),  
 "Genes"=subsetByOverlaps(x, genes)%>%length() /length(x) ,  
 "Promoters"=subsetByOverlaps(x, promotersGenes)%>%length()/length(x),  
 "Other"=(length(x)-sum(subsetByOverlaps(x, enhancers)%>%length()/length(x),  
 subsetByOverlaps(x, genes)%>%length(),  
 subsetByOverlaps(x, promotersGenes)%>%length()))/length(x))  
 set\_colnames(dataFrameGenomicLocation,value = "Number Of Motifs")  
})  
  
  
  
names(PhyloPConservedGenomicLocation)<-names(ARXMotifModelList)  
names(PhastConservedGenomicLocation)<-names(ARXMotifModelList)  
names(MotifModelGenomicLocation)<-names(ARXMotifModelList)  
  
AllMotifModels<-do.call(rbind.data.frame,c("PhyloP"=PhyloPConservedGenomicLocation,  
 "PhastCon"=PhastConservedGenomicLocation,  
 "All"=MotifModelGenomicLocation))%>%rownames\_to\_column(var= "Names")  
  
AllMotifModels<-separate(data = AllMotifModels,col = "Names",into = c("Conservation Method","Motif Model", "Genomic Location"), sep = '\\.' )  
AllMotifModels<-separate(data = AllMotifModels,col = "Genomic Location",into = c("Genomic Location", "Numbers" ),sep = "1")  
AllMotifModels<-separate(data = AllMotifModels,col = "Genomic Location",into = c("Genomic Location", "Numbers" ),sep = "2")  
  
AllMotifModels$`Genomic Location`<-factor(AllMotifModels$`Genomic Location`, levels = c("Other", "Genes", "Promoters", "Enhancers"))  
AllMotifModels$`Number Of Motifs`<-AllMotifModels$`Number Of Motifs`%>%as.numeric()  
AllMotifModels$`Motif Model`<-AllMotifModels$`Motif Model`%>%as.character()  
AllMotifModels$`Conservation Method`<-AllMotifModels$`Conservation Method`%>%as.character()  
  
  
ggplot(AllMotifModels)+  
 geom\_tile(aes(x=`Motif Model`, y= `Genomic Location`, fill= `Number Of Motifs` ))+  
 facet\_wrap(~`Conservation Method`)+  
 xlab(label = "Motif Model")+  
 ylab(label = "Genomic Location")+  
 theme\_bw()+  
 theme(axis.text.x = element\_text(size= 20,angle = 90),  
 axis.text.y= element\_text(size= 20),  
 axis.title = element\_text(size=25,face = "bold"),  
 legend.text = element\_text(size=20),   
 legend.title = element\_text(size=20),   
 strip.text = element\_text(size = 20))#+



# scale\_fill\_gradientn(colours=c("#69D2E7", "#A7DBD8",   
 # "#F38630", "#FA6900"),  
 # values=rescale(c( -2, -1,  
 # 1, 2)),  
 # guide="colorbar")

arx6mer<-(query(MotifDb, "Arx")[[6]])[,7:12]  
  
  
## Checking to see if the numbers are robust  
  
library(magrittr)  
library(GenomicRanges)  
library(ggplot2)  
library(magrittr)  
library(tibble)  
library(pander)  
library(reshape2)  
library(plyr)

##   
## Attaching package: 'plyr'

## The following object is masked from 'package:XVector':  
##   
## compact

## The following object is masked from 'package:IRanges':  
##   
## desc

## The following object is masked from 'package:S4Vectors':  
##   
## rename

library(MotifDb)  
  
library(magrittr)  
library(reshape2)  
library(BSgenome.Hsapiens.UCSC.hg19)  
  
# enhancerGrange <-  
# import(con = "~/DataFiles/Enhancer Tracks/Mouse/mouse\_permissive\_enhancers\_phase\_1\_and\_2.bed")  
# UCSCgenes <- import("~/DataFiles/Gene Tracks/Mouse/mm9.bed")  
# startSites<-subset(import("~/DataFiles/Gene Tracks/Mouse/FullMm9genome.GTF"), type== "start\_codon")  
# promoters <- promoters(UCSCgenes)  
# genome<-BSgenome.Mmusculus.UCSC.mm9  
  
enhancerGrange <-  
 import(con = "~/DataFiles/Enhancer Tracks/Human/human\_permissive\_enhancers\_phase\_1\_and\_2.bed")  
UCSCgenes <- import("~/DataFiles/Gene Tracks/Human/hg.bed")  
startSites<-subset(import("~/DataFiles/Gene Tracks/Human/hg19.gtf"), type== "start\_codon")  
promoters <- promoters(UCSCgenes)  
genome<-BSgenome.Hsapiens.UCSC.hg19  
ArxPlaindrmicMinus1<-cbind(arx6mer[,1:5], arx6mer[,5:1])  
  
arx6MerPWMNospace<-cbind(arx6mer, arx6mer[,6:1])  
  
arx6MerPWM1space<-cbind(arx6mer, 0.25, arx6mer[,6:1])  
  
arx6MerPWM2space<-cbind(arx6mer, 0.25, 0.25, arx6mer[,6:1])  
  
arx6MerPWM3space<- cbind(arx6mer, 0.25, 0.25, 0.25, arx6mer[,6:1])  
  
  
arx6MerPWM4space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25, arx6mer[,6:1])  
  
arx6MerPWM5space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25,0.25, arx6mer[,6:1])  
   
arx6MerPWM6space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25,0.25, 0.25, arx6mer[,6:1])  
  
arx6MerPWM7space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25,0.25, 0.25, 0.25, arx6mer[,6:1])  
  
### Tandeom Sites  
arxtandemMinus1<-cbind(arx6mer[,1:5], arx6mer[,1:5])  
   
arxJolma<-query(MotifDb, "Hsapiens-jolma2013-ARX")[[1]][,2:12]  
  
arxTandemNoSpace<-cbind(arx6mer, arx6mer)  
  
  
arxTandem1Space<-cbind(arx6mer, 0.25, arx6mer)  
  
arxTandem2Space<-cbind(arx6mer, 0.25, 0.25, arx6mer)  
  
arxTandem3Space<-cbind(arx6mer, 0.25, 0.25, 0.25, arx6mer)  
  
arxTandem4Space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25, arx6mer)  
  
arxTandem5Space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25, 0.25,arx6mer)  
  
arxTandem6Space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,arx6mer)  
  
arxTandem7Space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,arx6mer)  
  
arxTandem8Space<-cbind(arx6mer, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, arx6mer)  
  
  
##requires code from the 16-4-2017 to run  
# grangeJolmaMinus<-  
# matchPWM(arxJolma, genome, "100%")  
# grangeplaindromicMinus1 <-  
# matchPWM(ArxPlaindrmicMinus1, genome, "100%")  
# grangeplaindromicNospace <-  
# matchPWM(arx6MerPWMNospace, genome, "100%")  
# grangeplaindromic1space <-  
# matchPWM(arx6MerPWM1space, genome, "100%")  
# grangeplaindromic2space <-  
# matchPWM(arx6MerPWM2space, genome, "100%")  
# grangeplaindromic3space <-  
# matchPWM(arx6MerPWM3space, genome, "100%")  
# grangeplaindromic4space <-  
# matchPWM(arx6MerPWM4space, genome, "100%")  
# grangeplaindromic5space <-  
# matchPWM(arx6MerPWM5space, genome, "100%")  
# grangeplaindromic6space <-  
# matchPWM(arx6MerPWM6space, genome, "100%")  
# grangeplaindromic7space <-  
# matchPWM(arx6MerPWM7space, genome, "100%")  
  
# grangeTandemMinusOne <-  
# matchPWM(arxtandemMinus1, genome, "100%")  
# grangeTandemNoSpace<-  
# matchPWM(arxTandemNoSpace, genome, "100%")  
# grangeTandem1space <-  
# matchPWM(arxTandem1Space, genome, "100%")  
# grangeTandem2space <-  
# matchPWM(arxTandem2Space, genome, "100%")  
# grangeTandem3space <-  
# matchPWM(arxTandem3Space, genome, "100%")  
# grangeTandem4space <-  
# matchPWM(arxTandem4Space, genome, "100%")  
# grangeTandem5space <-  
# matchPWM(arxTandem5Space, genome, "100%")  
# grangeTandem6space <-  
# matchPWM(arxTandem6Space, genome, "100%")  
# grangeTandem7space <-  
# matchPWM(arxTandem7Space, genome, "100%")  
  
  
grangeJolmaMinus<-matchPWM(arxJolma, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromicMinus1 <-matchPWM(ArxPlaindrmicMinus1, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandemNoSpace<-matchPWM(arxTandemNoSpace, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic1space<-matchPWM(arx6MerPWM1space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic2space<-matchPWM(arx6MerPWM2space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic3space<-matchPWM(arx6MerPWM3space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic4space<-matchPWM(arx6MerPWM4space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic5space<-matchPWM(arx6MerPWM5space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic6space<-matchPWM(arx6MerPWM6space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromic7space<-matchPWM(arx6MerPWM7space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandemMinusOne <-matchPWM(arxtandemMinus1, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeplaindromicNospace <-matchPWM(arx6MerPWMNospace, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem1space<-matchPWM(arxTandem1Space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem2space<-matchPWM(arxTandem2Space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem3space<-matchPWM(arxTandem3Space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem4space<-matchPWM(arxTandem4Space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem5space<-matchPWM(arxTandem5Space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem6space<-matchPWM(arxTandem6Space, genome, "90%")

## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

grangeTandem7space<-matchPWM(arxTandem7Space, genome, "90%")

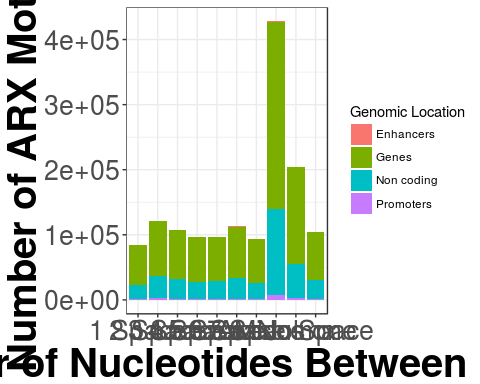
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them  
  
## Warning in .Call2("XString\_match\_PWM", pwm, subject, min.score,  
## count.only, : 'subject' contains letters not in [ACGT] ==> assigned weight  
## 0 to them

tandemDataTable <- rbind(  
 cbind(  
 length(grangeJolmaMinus),  
 sum(countOverlaps(grangeJolmaMinus, UCSCgenes)),  
 sum(countOverlaps(grangeJolmaMinus, promoters)),  
 sum(countOverlaps(grangeJolmaMinus, enhancerGrange)),  
 (length(grangeJolmaMinus)-sum(countOverlaps(grangeJolmaMinus, enhancerGrange))-  
 sum(countOverlaps(grangeJolmaMinus, promoters))- sum(countOverlaps(grangeJolmaMinus, UCSCgenes)))  
),  
 cbind(  
 numberofTandem <- length(grangeTandemMinusOne),  
 dataTableNoGenesminus1 <-  
 sum(countOverlaps(grangeTandemMinusOne, UCSCgenes)),  
 dataTableMinus1 <-  
 sum(countOverlaps(grangeTandemMinusOne, promoters)),  
 dataTableMinus1r <-  
 sum(countOverlaps(grangeTandemMinusOne, enhancerGrange)),  
 (length(grangeTandemMinusOne)-sum(countOverlaps(grangeTandemMinusOne, enhancerGrange))-  
 sum(countOverlaps(grangeTandemMinusOne, promoters))- sum(countOverlaps(grangeTandemMinusOne, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandemNoSpaceSites <- length(grangeTandemNoSpace),  
 dataTableNoGenes <-  
 sum(countOverlaps(grangeTandemNoSpace, UCSCgenes)),  
 dataTableNoSpacePromoters <-  
 sum(countOverlaps(grangeTandemNoSpace, promoters)),  
 dataTableNoSpaceEnhancer <-  
 sum(countOverlaps(grangeTandemNoSpace, enhancerGrange)),  
 (length(grangeTandemNoSpace)-sum(countOverlaps(grangeTandemNoSpace, enhancerGrange))-  
 sum(countOverlaps(grangeTandemNoSpace, promoters))- sum(countOverlaps(grangeTandemNoSpace, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandem1spaceSites <- length(grangeTandem1space),  
 dataTable1SpaceGenes <-  
 sum(countOverlaps(grangeTandem1space, UCSCgenes)),  
 dataTable1SpacePromoters <-  
 sum(countOverlaps(grangeTandem1space, promoters)),  
 dataTable1SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem1space, enhancerGrange)),  
 (length(grangeTandem1space)-sum(countOverlaps(grangeTandem1space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem1space, promoters))- sum(countOverlaps(grangeTandem1space, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandem2spaceSites <- length(grangeTandem2space),  
 dataTable2SpaceGenes <-  
 sum(countOverlaps(grangeTandem2space, UCSCgenes)),  
 dataTable2SpacePromoters <-  
 sum(countOverlaps(grangeTandem2space, promoters)),  
 dataTable2SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem2space, enhancerGrange)),  
 (length(grangeTandem2space)-sum(countOverlaps(grangeTandem2space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem2space, promoters))- sum(countOverlaps(grangeTandem2space, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandem3spaceSites <- length(grangeTandem3space),  
 dataTable3SpaceGenes <-  
 sum(countOverlaps(grangeTandem3space, UCSCgenes)),  
 dataTable3SpacePromoters <-  
 sum(countOverlaps(grangeTandem3space, promoters)),  
 dataTable3SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem3space, enhancerGrange)),  
 (length(grangeTandem3space)-sum(countOverlaps(grangeTandem3space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem3space, promoters))- sum(countOverlaps(grangeTandem3space, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandem4spaceSites <- length(grangeTandem4space),  
 dataTable4SpaceGenes <-  
 sum(countOverlaps(grangeTandem4space, UCSCgenes)),  
 dataTable4SpacePromoters <-  
 sum(countOverlaps(grangeTandem4space, promoters)),  
 dataTable4SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem4space, enhancerGrange)),  
 (length(grangeTandem4space)-sum(countOverlaps(grangeTandem4space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem4space, promoters))- sum(countOverlaps(grangeTandem4space, UCSCgenes)))  
 ),   
cbind(  
 numberofTandem5spaceSites <- length(grangeTandem5space),  
 dataTable5SpaceGenes <-  
 sum(countOverlaps(grangeTandem5space, UCSCgenes)),  
 dataTable5SpacePromoters <-  
 sum(countOverlaps(grangeTandem5space, promoters)),  
 dataTable5SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem5space, enhancerGrange)),  
 (length(grangeTandem5space)-sum(countOverlaps(grangeTandem5space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem5space, promoters))- sum(countOverlaps(grangeTandem5space, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandem6spaceSites <- length(grangeTandem6space),  
 dataTable6SpaceGenes <-  
 sum(countOverlaps(grangeTandem6space, UCSCgenes)),  
 dataTable6SpacePromoters <-  
 sum(countOverlaps(grangeTandem6space, promoters)),  
 dataTable6SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem6space, enhancerGrange)),  
 (length(grangeTandem6space)-sum(countOverlaps(grangeTandem6space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem6space, promoters))- sum(countOverlaps(grangeTandem6space, UCSCgenes)))  
 ),  
 cbind(  
 numberofTandem7spaceSites <- length(grangeTandem7space),  
 dataTable7SpaceGenes <-  
 sum(countOverlaps(grangeTandem7space, UCSCgenes)),  
 dataTable7SpacePromoters <-  
 sum(countOverlaps(grangeTandem7space, promoters)),  
 dataTable7SpaceEnhancer <-  
 sum(countOverlaps(grangeTandem7space, enhancerGrange)),  
 (length(grangeTandem7space)-sum(countOverlaps(grangeTandem7space, enhancerGrange))-  
 sum(countOverlaps(grangeTandem7space, promoters))- sum(countOverlaps(grangeTandem7space, UCSCgenes)))  
 )  
) %>% as.data.frame  
  
colnames(tandemDataTable) <- c("Total",  
 "Motifs in genes",  
 "Motifs in promoters",  
 "Motifs in enhancers",  
 "Non Coding")  
  
  
rownames(tandemDataTable) <- c("Arx Jolma",  
 "Minus one",  
 "No Space",  
 "1 Space",  
 "2 Space",  
 "3 Space",  
 "4 Space",  
 "5 Space",  
 "6 Space",  
 "7 Space")  
tandemDataTable %>% pander()

Table continues below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Total | | Motifs in genes | Motifs in promoters | |
| **Arx Jolma** | 428117 | | 287907 | 6919 | |
| **Minus one** | 204757 | | 148594 | 3462 | |
| **No Space** | 104926 | | 74441 | 1892 | |
| **1 Space** | 84588 | | 61231 | 1402 | |
| **2 Space** | 121418 | | 84710 | 2214 | |
| **3 Space** | 107950 | | 74946 | 1892 | |
| **4 Space** | 96302 | | 68464 | 1704 | |
| **5 Space** | 96574 | | 67820 | 1688 | |
| **6 Space** | 112956 | | 79105 | 1966 | |
| **7 Space** | 93932 | | 67293 | 1638 | |
|  | | Motifs in enhancers | | | Non Coding |
| **Arx Jolma** | | 1110 | | | 132181 |
| **Minus one** | | 575 | | | 52126 |
| **No Space** | | 266 | | | 28327 |
| **1 Space** | | 208 | | | 21747 |
| **2 Space** | | 244 | | | 34250 |
| **3 Space** | | 246 | | | 30866 |
| **4 Space** | | 220 | | | 25914 |
| **5 Space** | | 356 | | | 26710 |
| **6 Space** | | 252 | | | 31633 |
| **7 Space** | | 232 | | | 24769 |

tandemDataTable <- rownames\_to\_column(tandemDataTable)  
reshapedTandemDataTable<-reshape(tandemDataTable,  
 varying = c( "Motifs in promoters", "Motifs in enhancers", "Non Coding", "Motifs in genes"),  
 v.names = "Numbers of Motif",  
 timevar = "Location",  
 times = c( "Promoters", "Enhancers", "Non coding","Genes" ),  
 direction = "long")  
ggplot(reshapedTandemDataTable, aes(x = rowname, y = `Numbers of Motif`, fill = `Location`)) +  
 geom\_bar(stat = "identity") +  
 xlab(label= "Number of Nucleotides Between Motifs")+  
 ylab(label= "Number of ARX Motifs")+  
 guides(fill=guide\_legend(title="Genomic Location"))+  
 theme\_bw()+  
 theme(axis.title = element\_text(size=30, face = "bold"),  
 axis.text =element\_text(size=20))

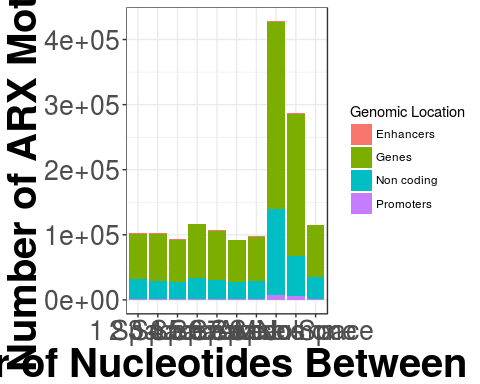


planindromicDataTable <- rbind(  
 cbind(  
 length(grangeJolmaMinus),  
 sum(countOverlaps(grangeJolmaMinus, UCSCgenes)),  
 sum(countOverlaps(grangeJolmaMinus, promoters)),  
 sum(countOverlaps(grangeJolmaMinus, enhancerGrange)),  
 (length(grangeJolmaMinus)-sum(countOverlaps(grangeJolmaMinus, enhancerGrange))-  
 sum(countOverlaps(grangeJolmaMinus, promoters))- sum(countOverlaps(grangeJolmaMinus, UCSCgenes)))  
 ),  
 cbind(  
 length(grangeplaindromicMinus1),  
 sum(countOverlaps(grangeplaindromicMinus1, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromicMinus1, promoters)),  
 sum(countOverlaps(grangeplaindromicMinus1, enhancerGrange)),  
 (length(grangeplaindromicMinus1)-sum(countOverlaps(grangeplaindromicMinus1, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromicMinus1, promoters))- sum(countOverlaps(grangeplaindromicMinus1, UCSCgenes)))  
 ),  
 cbind(  
 length(grangeplaindromicNospace),  
 sum(countOverlaps(grangeplaindromicNospace, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromicNospace, promoters)),  
 sum(countOverlaps(grangeplaindromicNospace, enhancerGrange)),  
 (length(grangeplaindromicNospace)-sum(countOverlaps(grangeplaindromicNospace, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromicNospace, promoters))- sum(countOverlaps(grangeplaindromicNospace, UCSCgenes)))  
 ),  
 cbind(  
 length(grangeplaindromic1space),  
 Arx6mer <- sum(countOverlaps(grangeplaindromic1space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic1space, promoters)),  
 sum(countOverlaps(grangeplaindromic1space, enhancerGrange)),  
 (length(grangeplaindromic1space)-sum(countOverlaps(grangeplaindromic1space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic1space, promoters))- sum(countOverlaps(grangeplaindromic1space, UCSCgenes)))  
 ),  
 cbind(  
 length(grangeplaindromic2space),  
 sum(countOverlaps(grangeplaindromic2space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic2space, promoters)),  
 sum(countOverlaps(grangeplaindromic2space, enhancerGrange)),  
 (length(grangeplaindromic2space)-sum(countOverlaps(grangeplaindromic2space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic2space, promoters))- sum(countOverlaps(grangeplaindromic2space, UCSCgenes)))  
 )  
 ,  
 cbind(  
 numberOfArxSitesPlaindromic3Space <- length(grangeplaindromic3space),  
 sum(countOverlaps(grangeplaindromic3space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic3space, promoters)),  
 sum(countOverlaps(grangeplaindromic4space, enhancerGrange)),  
 (length(grangeplaindromic3space)-sum(countOverlaps(grangeplaindromic3space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic3space, promoters))- sum(countOverlaps(grangeplaindromic3space, UCSCgenes)))  
 ),  
 cbind(  
 numberOfArxSitesPlaindromic4Space <- length(grangeplaindromic4space),  
 sum(countOverlaps(grangeplaindromic4space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic4space, promoters)),  
 sum(countOverlaps(grangeplaindromic4space, enhancerGrange)),  
 (length(grangeplaindromic4space)-sum(countOverlaps(grangeplaindromic4space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic4space, promoters))- sum(countOverlaps(grangeplaindromic4space, UCSCgenes)))  
 ),  
 cbind(  
 numberOfArxSitesPlaindromic5Space <- length(grangeplaindromic5space),  
 sum(countOverlaps(grangeplaindromic5space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic5space, promoters)),  
 sum(countOverlaps(grangeplaindromic5space, enhancerGrange)),  
 (length(grangeplaindromic5space)-sum(countOverlaps(grangeplaindromic5space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic5space, promoters))- sum(countOverlaps(grangeplaindromic5space, UCSCgenes)))  
 ),  
 cbind(  
 numberOfArxSitesPlaindromic6Space <- length(grangeplaindromic6space),  
 sum(countOverlaps(grangeplaindromic6space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic6space, promoters)),  
 sum(countOverlaps(grangeplaindromic6space, enhancerGrange)),  
 (length(grangeplaindromic6space)-sum(countOverlaps(grangeplaindromic6space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic6space, promoters))- sum(countOverlaps(grangeplaindromic6space, UCSCgenes)))  
 ),  
 cbind(  
 numberOfArxSitesPlaindromic7Space <- length(grangeplaindromic7space),  
 sum(countOverlaps(grangeplaindromic7space, UCSCgenes)),  
 sum(countOverlaps(grangeplaindromic7space, promoters)),  
 sum(countOverlaps(grangeplaindromic7space, enhancerGrange)),  
 (length(grangeplaindromic7space)-sum(countOverlaps(grangeplaindromic7space, enhancerGrange))-  
 sum(countOverlaps(grangeplaindromic7space, promoters))- sum(countOverlaps(grangeplaindromic7space, UCSCgenes)))  
 )  
) %>% as.data.frame()  
colnames(planindromicDataTable) <- c("Total",  
 "Motifs in genes",  
 "Motifs in Promoters",  
 "Motifs in Enhancers",  
 "Non Coding")  
rownames(planindromicDataTable) <-c("Arx Jolma",  
 "Minus one",  
 "No Space",  
 "1 Space",  
 "2 Space",  
 "3 Space",  
 "4 Space",  
 "5 Space",  
 "6 Space",  
 "7 Space")  
  
  
planindromicDataTable %>% pander()

Table continues below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Total | | Motifs in genes | Motifs in Promoters | |
| **Arx Jolma** | 428117 | | 287907 | 6919 | |
| **Minus one** | 286440 | | 217740 | 5503 | |
| **No Space** | 114612 | | 79677 | 2031 | |
| **1 Space** | 102177 | | 69972 | 1702 | |
| **2 Space** | 102033 | | 72384 | 2075 | |
| **3 Space** | 92852 | | 65199 | 1629 | |
| **4 Space** | 116418 | | 83044 | 2042 | |
| **5 Space** | 106736 | | 75575 | 1996 | |
| **6 Space** | 92466 | | 64711 | 1589 | |
| **7 Space** | 97637 | | 68266 | 1635 | |
|  | | Motifs in Enhancers | | | Non Coding |
| **Arx Jolma** | | 1110 | | | 132181 |
| **Minus one** | | 862 | | | 62335 |
| **No Space** | | 267 | | | 32637 |
| **1 Space** | | 211 | | | 30292 |
| **2 Space** | | 268 | | | 27306 |
| **3 Space** | | 246 | | | 25807 |
| **4 Space** | | 246 | | | 31086 |
| **5 Space** | | 245 | | | 28920 |
| **6 Space** | | 242 | | | 25924 |
| **7 Space** | | 281 | | | 27455 |

planindromicDataTable<- rownames\_to\_column(planindromicDataTable)  
  
reshapedPlaindromicDataTable<-reshape(planindromicDataTable,  
 varying = c( "Motifs in Promoters", "Motifs in Enhancers", "Non Coding", "Motifs in genes"),  
 v.names = "Numbers of Motif",  
 timevar = "Location",  
 times = c( "Promoters", "Enhancers", "Non coding","Genes" ),  
 direction = "long")  
ggplot(reshapedPlaindromicDataTable, aes(x = rowname, y = `Numbers of Motif`, fill = `Location`)) +  
 geom\_bar(stat = "identity") +  
 xlab(label= "Number of Nucleotides Between Motifs")+  
 ylab(label= "Number of ARX Motifs")+  
 guides(fill=guide\_legend(title="Genomic Location"))+  
 theme\_bw()+  
 theme(axis.title = element\_text(size=30, face = "bold"),  
 axis.text =element\_text(size=20) )+  
 scale\_color\_manual(values=c(`Enhancer`="#999999", `Genes`="#E69F00", `Non-coding`="#56B4E9", `Promoters`= "#56B4E9"))



### making histograms of distance of Tandem the Arx Start sites  
  
dataFrameDistance1SpacePromoter <-  
 distanceToNearest(grangeTandem1space, startSites) %>%   
 as.data.frame()  
dataFrameDistance2SpacePromoter <-  
 distanceToNearest(grangeTandem2space, startSites) %>%  
 as.data.frame()  
dataFrameDistance3SpacePromoter <-  
 distanceToNearest(grangeTandem3space, startSites) %>%  
 as.data.frame()  
dataFrameDistance6SpacePromoter <-  
 distanceToNearest(grangeTandem6space, startSites) %>%  
 as.data.frame()  
  
  
dataFrameMerger<-function(z,x,c,v){  
   
 test<-merge(z[3],x[3],by=0, all=TRUE, row.names=NULL)  
 test2<-merge(test, c[3], by=0, all=TRUE, row.names=NULL)  
 test3<- merge(test2, v[3], by=0,all=TRUE, row.names=NULL)  
   
   
 return(test3)  
}  
  
dataFrameDistanceofTandemMotifsFromPromoter<-dataFrameMerger(dataFrameDistance1SpacePromoter,  
 dataFrameDistance2SpacePromoter,   
 dataFrameDistance3SpacePromoter,   
 dataFrameDistance6SpacePromoter)

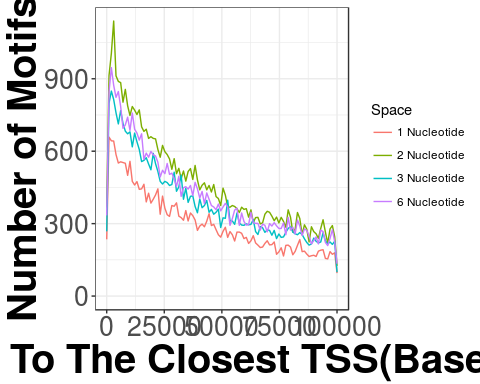
## Warning in merge.data.frame(test, c[3], by = 0, all = TRUE, row.names =  
## NULL): column name 'Row.names' is duplicated in the result

## Warning in merge.data.frame(test2, v[3], by = 0, all = TRUE, row.names =  
## NULL): column names 'Row.names', 'Row.names', 'distance.x', 'distance.y'  
## are duplicated in the result

dataFrameDistanceofTandemMotifsFromPromoter<- dataFrameDistanceofTandemMotifsFromPromoter[4:7]  
colnames(dataFrameDistanceofTandemMotifsFromPromoter)<- c("1 Space",  
 "2 Space",  
 "3 Space",  
 "6 Space")  
ggplotdataFrameDistanceofTandemicMotifsFromPromoter<-reshape(dataFrameDistanceofTandemMotifsFromPromoter,  
 varying = c("1 Space", "2 Space", "3 Space", "6 Space"),  
 v.names = "Distance",  
 timevar = "Space",  
 times = c("1 Nucleotide", "2 Nucleotide", "3 Nucleotide", "6 Nucleotide"),  
 direction = "long")  
  
ggplot(ggplotdataFrameDistanceofTandemicMotifsFromPromoter, aes(x=Distance, group=Space, color=Space))+  
 geom\_freqpoly(bins = 100)+  
 xlab(label = "Distance To The Closest TSS(Base Pairs)")+  
 ylab(label= "Number of Motifs")+  
 theme\_bw()+  
 theme(axis.title = element\_text(size=30, face = "bold"),  
 axis.text =element\_text(size=20) )+  
 scale\_x\_continuous(limits = c(0, 100000))

## Warning: Removed 327302 rows containing non-finite values (stat\_bin).

## Warning: Removed 8 rows containing missing values (geom\_path).



##histogram of distances of Plaindromic Motifs  
  
  
dataFrameDistancePlandromic1SpacePromoter <-  
 distanceToNearest(grangeplaindromic1space, startSites) %>% as.data.frame  
dataFrameDistancePlandromic2SpacePromoter <-  
 distanceToNearest(grangeplaindromic2space, startSites) %>% as.data.frame  
dataFrameDistancePlandromic3SpacePromoter <-  
 distanceToNearest(grangeplaindromic3space, startSites) %>% as.data.frame  
dataFrameDistancePlandromic4SpacePromoter <-  
 distanceToNearest(grangeplaindromic4space, startSites) %>% as.data.frame  
  
  
dataFrameDistanceofPlandromicMotifsFromPromoter <- dataFrameMerger(dataFrameDistancePlandromic1SpacePromoter,  
 dataFrameDistancePlandromic2SpacePromoter,  
 dataFrameDistancePlandromic3SpacePromoter,  
 dataFrameDistancePlandromic4SpacePromoter)

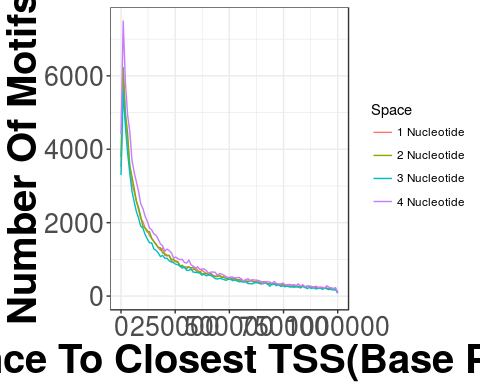
## Warning in merge.data.frame(test, c[3], by = 0, all = TRUE, row.names =  
## NULL): column name 'Row.names' is duplicated in the result

## Warning in merge.data.frame(test2, v[3], by = 0, all = TRUE, row.names =  
## NULL): column names 'Row.names', 'Row.names', 'distance.x', 'distance.y'  
## are duplicated in the result

dataFrameDistanceofPlandromicMotifsFromPromoter<-dataFrameDistanceofPlandromicMotifsFromPromoter[4:7]  
colnames(dataFrameDistanceofPlandromicMotifsFromPromoter)<- c("1 Space",  
 "2 Space",  
 "3 Space",  
 "4 Space")  
  
  
ggplotdataFrameDistanceofPlaindromicMotifsFromPromoter<-reshape(dataFrameDistanceofPlandromicMotifsFromPromoter,  
 varying = c("1 Space", "2 Space", "3 Space", "4 Space"),  
 v.names = "Distance",  
 timevar = "Space",  
 times = c("1 Nucleotide", "2 Nucleotide", "3 Nucleotide", "4 Nucleotide"),  
 direction = "long")  
ggplotdataFrameDistanceofPlaindromicMotifsFromPromoter$Distance<-ggplotdataFrameDistanceofPlaindromicMotifsFromPromoter$Distance%>%as.character%>%as.numeric()  
ggplot(ggplotdataFrameDistanceofPlaindromicMotifsFromPromoter, aes(x=Distance, group=Space, color=Space))+  
 geom\_freqpoly(bins = 100)+  
 xlab(label = "Distance To Closest TSS(Base Pairs)")+  
 ylab(label= "Number Of Motifs")+  
 scale\_x\_continuous(limits = c(0, 1000000))+  
 theme\_bw()+  
 theme(axis.title = element\_text(size=30, face = "bold"),  
 axis.text =element\_text(size=20))

## Warning: Removed 103986 rows containing non-finite values (stat\_bin).

## Warning: Removed 8 rows containing missing values (geom\_path).



## Average distances  
Numeric<-apply(dataFrameDistanceofTandemMotifsFromPromoter, 2, as.numeric)  
Numeric<-apply(dataFrameDistanceofPlandromicMotifsFromPromoter, 2, as.numeric)  
Space1Av<-sum(na.omit(Numeric[,1]))/length(na.omit(Numeric[,1]))  
Space2Av<-sum(na.omit(Numeric[,2]))/length(na.omit(Numeric[,2]))  
Space3Av<-sum(na.omit(Numeric[,3]))/length(na.omit(Numeric[,3]))  
Space4Av<-sum(na.omit(Numeric[,4]))/length(na.omit(Numeric[,4]))

## Thesis table subsetting for unique genes

library(readr)  
MouseEnhancerPromoterInteractions <- read\_delim("~/Thesis/MouseEnhancerPromoterInteractions",   
 "\t", escape\_double = FALSE, trim\_ws = TRUE)  
  
UNiquePromoterEnhancers<-subset(MouseEnhancerPromoterInteractions, isUnique(MouseEnhancerPromoterInteractions$`Gene Symbol`))  
  
  
write.table(UNiquePromoterEnhancers,file = "~/Thesis/MouseEnhancerPromoterInteractionsUnique",  
 sep= "\t",  
 row.names=FALSE,  
 append=FALSE,  
 quote=FALSE)  
EnhancerPromoterInteractionsHuman <- read\_delim("~/Thesis/EnhancerPromoterInteractionsHuman",   
 "\t", escape\_double = FALSE, trim\_ws = TRUE)  
uniquePromoterEnhancers<-subset(EnhancerPromoterInteractionsHuman, isUnique(EnhancerPromoterInteractionsHuman$`Gene Symbol`))  
write.table(uniquePromoterEnhancers,file = "~/Thesis/uniquePromoterEnhancersHuman",  
 sep= "\t",  
 row.names=FALSE,  
 append=FALSE,  
 quote=FALSE)

So this code chunk I have just de duplicated the enhancer promoter intearctions I made ealier based on the Gene symbol. So that was cool.

### Dr. Jimmy Breens method of identifying ARX Targets

Simply put: 1. Identify active motifs in enhancer and promoter regions in the Cell type 2. Map reads\_1 over Arx motifs 3. Map reads\_2 Over promoters Set cutoff for minimum number of reads? Overlapping the promoter & Motif? Edit: not setting minimum.

enhancers<-import("~/DataFiles/Enhancer Tracks/Human/human\_permissive\_enhancers\_phase\_1\_and\_2.bed")  
genes<-import("~/DataFiles/Gene Tracks/Human/hg.bed")  
promotersGenes<-promoters(genes)  
  
arx6merTFBS<-readRDS("~/DataFiles/ChIPseq/Human/ARX6merHg19Sites")  
ARXTandem2SpacedTFBS<-readRDS("~/DataFiles/ChIPseq/Human/ARXTande2SpacedSites")  
Plaindromic4SpacedTFBS<-readRDS("~/DataFiles/ChIPseq/Human/Plaindromic4SpacedTFBS")  
JolmaTFBS<-readRDS("~/DataFiles/ChIPseq/Human/JolmaTFBS")  
  
ARXMotifModelList<-c("6 Mer"= arx6merTFBS,  
 "Tandem 2 Spaced" = ARXTandem2SpacedTFBS,  
 "Palindromic 4 Spaced" = Plaindromic4SpacedTFBS,  
 "Jolma Model" = JolmaTFBS)  
  
arxPromoters<-lapply(ARXMotifModelList, function(x)c(subsetByOverlaps(x, promotersGenes))%>%unlist)  
arxEnhancers<-lapply(ARXMotifModelList, function(x)c(subsetByOverlaps(x, enhancers))%>%unlist)  
phyloPScores<-lapply(arxPromoters, function(x){import("~/DataFiles/Conservation/Human/hg19.100way.phyloP100way.bw", which= x)})

## Warning in .local(con, format, text, ...): 'which' contains seqlevels not  
## known to BigWig file: chrUn\_gl000226  
  
## Warning in .local(con, format, text, ...): 'which' contains seqlevels not  
## known to BigWig file: chrUn\_gl000226  
  
## Warning in .local(con, format, text, ...): 'which' contains seqlevels not  
## known to BigWig file: chrUn\_gl000226  
  
## Warning in .local(con, format, text, ...): 'which' contains seqlevels not  
## known to BigWig file: chrUn\_gl000226

phyloPAbove0.5<-lapply(phyloPScores, function(x){subset(x, score>=0.5)})  
conservedARXPromoters<-list()  
for(i in 1:4){  
 conservedARXPromoters[i]<-subset(arxPromoters[[i]], countOverlaps(arxPromoters[[i]], phyloPAbove0.5[[i]])>=6)  
}

## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated

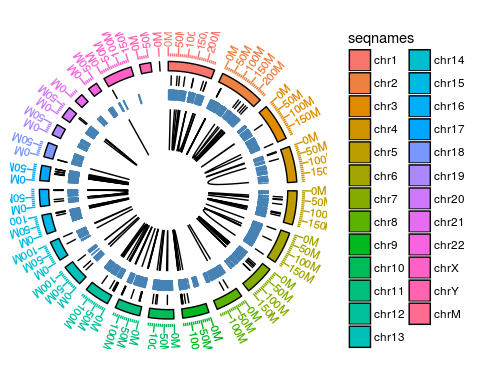
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated  
  
## Warning in `[<-`(`\*tmp\*`, i, value = <S4 object of class  
## structure("GRanges", package = "GenomicRanges")>): implicit list embedding  
## of S4 objects is deprecated

names(conservedARXPromoters)<-names(ARXMotifModelList)  
conservedARXPromotersAndEnhancers<-list(c(conservedARXPromoters$`6 Mer`, arxEnhancers$`6 Mer`)%>%unlist(),  
 c(conservedARXPromoters$`Tandem 2 Spaced`, arxEnhancers$`Tandem 2 Spaced`)%>%unlist(),  
 c(conservedARXPromoters$`Palindromic 4 Spaced`, arxEnhancers$`Palindromic 4 Spaced`)%>%unlist(),  
 c(conservedARXPromoters$`Jolma Model`, arxEnhancers$`Jolma Model`)%>%unlist()  
 )  
  
names(conservedARXPromotersAndEnhancers)<-names(ARXMotifModelList)  
  
  
PotenitalARXMotifs<-c(arxEnhancers[[1]],  
 arxEnhancers[[2]],  
 arxEnhancers[[3]],  
 arxEnhancers[[4]])%>%unlist  
  
# Epigenomic Track  
fetalBrainMaleHMM<-import("~/DataFiles/ChromHMM/human/E081\_15\_coreMarks\_dense.bed.gz")  
  
  
activeMotifs<-subset(fetalBrainMaleHMM, name %in% c( "5\_TxWk",   
 "4\_Tx",  
 "1\_TssA",   
 "3\_TxFlnk",  
 "2\_TssAFlnk",   
 "7\_Enh",  
 "6\_EnhG",  
 "12\_EnhBiv",   
 "11\_BivFlnk",   
 "10\_TssBiv" ))%>%subsetByOverlaps(PotenitalARXMotifs, .)  
  
  
  
  
# RawReads<-readRDS("~/DataFiles/HiC/Human/HumanRawReads")  
  
### THis wont work memory issues Use GRANGES list  
#   
#RawReads<-RawReads%>%as.data.frame()  
#  
# cbind.data.frame(RawReads$seqnames,   
# RawReads$start,  
# RawReads$end,  
# RawReads$X.seqnames,  
# RawReads$X.start,  
# RawReads$X.end,  
# "Interaction",  
# 1,  
# 1)%>%write.table(file = "~/DataFiles/HiC/Human/rawReads.bedpe",append = FALSE,   
# quote=FALSE,   
# col.names=FALSE,  
# row.names= FALSE,   
# sep= "\t")  
#   
#   
# #Import BedBPe file  
# RawReads<-makeGenomicInteractionsFromFile(fn = "~/DataFiles/HiC/Human/rawReads.bedpe",  
# type= "bedpe", experiment\_name = "NL CL Repilicate 2")  
# mcols(SignificantInteractions)<-cbind.data.frame("counts"=SignificantInteractions$counts,  
# "strand1",RawReads$strand,  
# "strand2",RawReads$X.strand)  
  
  
  
  
# ##Overlap GRANGE 1 with Motifs  
# Grange1OverlapsWithMotif<-subsetByOverlaps(RawReads,activeMotifs)  
#   
# ## Grange 2 With Active promoters! Usng a vector to subset OuterGRANGE  
# Grange2OverlapsWithPromoter<-subset(Grange1OverlapsWithMotif ,countOverlaps(mcols(Grange1OverlapsWithMotif)$X,  
# promotersGenes)>0)  
#   
#   
#   
# ## The same as above but in the other order therefore Grange 2 overlaps with motifs and Grange 1 overlaps with motifs  
# Grange2OverlapsWithMotif<-subset(RawReads ,countOverlaps(mcols(RawReads)$X,  
# activeMotifs)>0)  
# Grange1OverlapsWithPromoter<-subsetByOverlaps(Grange2OverlapsWithMotif, promotersGenes)  
  
  
  
  
  
#InteractionsBetweenActiveARXMotifsAndPromoters<-c(Grange1OverlapsWithPromoter, Grange2OverlapsWithPromoter)%>%unlist()  
  
#saveRDS(InteractionsBetweenActiveARXMotifsAndPromoters, "~/DataFiles/HiC/Human/RawInteractionsPromoterArxMotif")  
InteractionsBetweenActiveARXMotifsAndPromoters<-readRDS( "~/DataFiles/HiC/Human/RawInteractionsPromoterArxMotif")  
  
### Equalize Seq Levels between levels  
library(TxDb.Hsapiens.UCSC.hg19.knownGene)

## Loading required package: GenomicFeatures

## Loading required package: AnnotationDbi

txdb<-TxDb.Hsapiens.UCSC.hg19.knownGene  
seqlevels(txdb, force=TRUE)<-seqlevels(InteractionsBetweenActiveARXMotifsAndPromoters)  
seqinfo(InteractionsBetweenActiveARXMotifsAndPromoters)<-seqinfo(txdb)  
  
seqinfo(InteractionsBetweenActiveARXMotifsAndPromoters$X)<-seqinfo(txdb)  
  
seqlevels(activeMotifs)<-seqlevels(txdb)  
seqinfo(activeMotifs)<-seqinfo(txdb)  
  
  
PromotersCircosPlot<-subsetByOverlaps(promotersGenes, InteractionsBetweenActiveARXMotifsAndPromoters)  
PromotersCircosPlot<-c(PromotersCircosPlot,subsetByOverlaps(promotersGenes, InteractionsBetweenActiveARXMotifsAndPromoters$X))%>%unlist()  
  
  
seqlevels(PromotersCircosPlot)<-seqlevels(txdb)  
seqinfo(PromotersCircosPlot)<-seqinfo(txdb)  
  
## Make sick af plot  
library(ggbio)  
ggbio()+  
 circle(InteractionsBetweenActiveARXMotifsAndPromoters, geom= "link", linked.to="X")+  
 circle(reduce(activeMotifs), geom= "rect", color= "steelblue")+  
 circle(reduce(PromotersCircosPlot), geom= "rect")+  
 circle(reduce(PromotersCircosPlot), geom="ideogram", aes(fill=seqnames))+  
 circle(reduce(PromotersCircosPlot), geom="scale",aes(color=seqnames))



Here is a circos plot of the raw interactions using just the read Data!!

So its pretty lit tbh, the plot goes as follows: Circos plot along the exteriror to interiro:

1. Scale bar

2. Ideogram

3. Promoters Interacting With Arx motifs

4. Every active Predicted Arx TFBS/Motif

5. The itneraction occuring

The same of the Raw Reads Mapping between the motifs

InteractionsBetweenActiveARXMotifsAndPromoters%>%as.data.frame()%>%set\_colnames(c("chr1", "start1", "end1", "width1", "strand1","chr2", "start2", "end2", "width2", "strand2"))

## chr1 start1 end1 width1 strand1 chr2 start2 end2  
## 1 chr1 52869886 52869986 101 + chr1 85038907 85039007  
## 2 chr1 149815800 149815900 101 - chr1 149816854 149816954  
## 3 chr1 149815822 149815922 101 - chr1 149816553 149816653  
## 4 chr1 150457827 150457927 101 + chr1 212576390 212576490  
## 5 chr1 211431201 211431301 101 + chr1 212576304 212576404  
## 6 chr1 228869969 228870069 101 + chr1 229341170 229341270  
## 7 chr1 229052751 229052851 101 - chr1 229341249 229341349  
## 8 chr2 152590392 152590492 101 - chr2 200946113 200946213  
## 9 chr2 191783291 191783391 101 + chr2 208004437 208004537  
## 10 chr2 206952021 206952121 101 - chr2 225307496 225307596  
## 11 chr2 220315457 220315557 101 + chr2 231965854 231965954  
## 12 chr4 7382396 7382496 101 + chr4 7833161 7833261  
## 13 chr4 46738618 46738718 101 - chr4 170026571 170026671  
## 14 chr5 37177797 37177897 101 - chr5 38514646 38514746  
## 15 chr5 50676931 50677031 101 + chr5 52155133 52155233  
## 16 chr5 131992600 131992700 101 - chr5 159837762 159837862  
## 17 chr5 150156659 150156759 101 + chr5 150163143 150163243  
## 18 chr6 7388463 7388563 101 + chr6 7392758 7392858  
## 19 chr6 75954932 75955032 101 - chr6 79779416 79779516  
## 20 chr6 75954940 75955040 101 - chr6 79779416 79779516  
## 21 chr7 28725623 28725723 101 + chr7 28733103 28733203  
## 22 chr7 45037585 45037685 101 + chr7 55131810 55131910  
## 23 chr7 77046913 77047013 101 - chr7 77820363 77820463  
## 24 chr8 38645524 38645624 101 + chr8 38857116 38857216  
## 25 chr8 67091784 67091884 101 - chr8 67132182 67132282  
## 26 chr9 95187871 95187971 101 - chr9 134603082 134603182  
## 27 chr9 114522244 114522344 101 - chr9 114812325 114812425  
## 28 chr9 130342841 130342941 101 - chr9 130835009 130835109  
## 29 chr9 133970238 133970338 101 + chr9 134603081 134603181  
## 30 chr10 35363674 35363774 101 - chr10 43714228 43714328  
## 31 chr10 97824166 97824266 101 - chr10 102087085 102087185  
## 32 chr11 18287344 18287444 101 + chr11 18546369 18546469  
## 33 chr12 567896 567996 101 + chr12 1769811 1769911  
## 34 chr12 3589332 3589432 101 + chr12 76414338 76414438  
## 35 chr12 10325625 10325725 101 - chr12 64855548 64855648  
## 36 chr12 29376095 29376195 101 + chr12 76271831 76271931  
## 37 chr12 85308519 85308619 101 - chr12 89978381 89978481  
## 38 chr13 28922112 28922212 101 - chr13 29393400 29393500  
## 39 chr13 49821920 49822020 101 + chr13 52219120 52219220  
## 40 chr14 58908761 58908861 101 + chr14 58912013 58912113  
## 41 chr15 83654352 83654452 101 + chr15 83869296 83869396  
## 42 chr15 101835992 101836092 101 - chr15 101995907 101996007  
## 43 chr15 101836012 101836112 101 - chr15 101995924 101996024  
## 44 chr16 30771042 30771142 101 + chr16 31083274 31083374  
## 45 chr16 84695998 84696098 101 + chr16 85586436 85586536  
## 46 chr17 43883318 43883418 101 + chr17 44234370 44234470  
## 47 chr17 45926336 45926436 101 - chr17 46083796 46083896  
## 48 chr17 46047804 46047904 101 + chr17 46083796 46083896  
## 49 chr17 47325387 47325487 101 + chr17 59018558 59018658  
## 50 chr17 73662044 73662144 101 + chr17 80796152 80796252  
## 51 chr17 74100710 74100810 101 - chr17 74237450 74237550  
## 52 chr20 49411097 49411197 101 + chr20 52481895 52481995  
## 53 chrX 9934954 9935054 101 + chrX 9978832 9978932  
## 54 chr1 17387330 17387430 101 + chr1 17907236 17907336  
## 55 chr1 66854729 66854829 101 - chr1 67144367 67144467  
## 56 chr1 85038898 85038998 101 - chr1 98512854 98512954  
## 57 chr2 27304263 27304363 101 - chr2 28207708 28207808  
## 58 chr2 27304290 27304390 101 - chr2 27333881 27333981  
## 59 chr2 27304315 27304415 101 - chr2 27496639 27496739  
## 60 chr2 27304352 27304452 101 - chr2 27496593 27496693  
## 61 chr2 120040635 120040735 101 + chr2 167345426 167345526  
## 62 chr3 63946457 63946557 101 - chr3 66458467 66458567  
## 63 chr3 170187118 170187218 101 - chr3 196015894 196015994  
## 64 chr4 1874956 1875056 101 - chr4 1939310 1939410  
## 65 chr5 52155153 52155253 101 + chr5 54396778 54396878  
## 66 chr5 81074729 81074829 101 - chr5 81281290 81281390  
## 67 chr5 137001392 137001492 101 - chr5 137041818 137041918  
## 68 chr5 137001447 137001547 101 - chr5 137056848 137056948  
## 69 chr5 137001470 137001570 101 - chr5 137009744 137009844  
## 70 chr6 57052322 57052422 101 - chr6 170864412 170864512  
## 71 chr7 28733149 28733249 101 + chr7 38404618 38404718  
## 72 chr7 30220564 30220664 101 - chr7 54608048 54608148  
## 73 chr7 77754089 77754189 101 - chr7 78402001 78402101  
## 74 chr7 77754094 77754194 101 - chr7 78401991 78402091  
## 75 chr9 130320478 130320578 101 + chr9 130714083 130714183  
## 76 chr9 130835035 130835135 101 - chr9 131304116 131304216  
## 77 chr9 134603101 134603201 101 + chr9 137995655 137995755  
## 78 chr9 134603122 134603222 101 + chr9 134612916 134613016  
## 79 chr10 102087069 102087169 101 + chr10 105879718 105879818  
## 80 chr10 122963409 122963509 101 - chr10 123359724 123359824  
## 81 chr11 18546294 18546394 101 + chr11 18624729 18624829  
## 82 chr11 18546315 18546415 101 + chr11 18550472 18550572  
## 83 chr11 22362147 22362247 101 + chr11 22851618 22851718  
## 84 chr11 71704832 71704932 101 + chr11 72464110 72464210  
## 85 chr11 128392905 128393005 101 + chr11 128393157 128393257  
## 86 chr11 131942270 131942370 101 + chr11 133808828 133808928  
## 87 chr11 131942327 131942427 101 + chr11 132394970 132395070  
## 88 chr12 1769819 1769919 101 + chr12 2983611 2983711  
## 89 chr13 99128295 99128395 101 + chr13 101241495 101241595  
## 90 chr14 100241225 100241325 101 + chr14 105648580 105648680  
## 91 chr15 83869323 83869423 101 - chr15 97315567 97315667  
## 92 chr15 96886347 96886447 101 + chr15 97320466 97320566  
## 93 chr16 17458096 17458196 101 + chr16 17566038 17566138  
## 94 chr16 31083240 31083340 101 + chr16 87424397 87424497  
## 95 chr16 31083310 31083410 101 + chr16 86380071 86380171  
## 96 chr16 85368626 85368726 101 - chr16 85643081 85643181  
## 97 chr16 85586397 85586497 101 - chr16 85643112 85643212  
## 98 chr16 85586509 85586609 101 - chr16 85643105 85643205  
## 99 chr17 46083783 46083883 101 - chr17 61270033 61270133  
## 100 chr17 59018489 59018589 101 + chr17 59066788 59066888  
## 101 chr17 73333295 73333395 101 - chr17 73661940 73662040  
## 102 chr18 3251012 3251112 101 + chr18 3601996 3602096  
## 103 chr20 34463629 34463729 101 + chr20 57619192 57619292  
## width2 strand2  
## 1 101 -  
## 2 101 +  
## 3 101 -  
## 4 101 +  
## 5 101 +  
## 6 101 +  
## 7 101 +  
## 8 101 -  
## 9 101 -  
## 10 101 -  
## 11 101 +  
## 12 101 +  
## 13 101 +  
## 14 101 -  
## 15 101 -  
## 16 101 -  
## 17 101 -  
## 18 101 -  
## 19 101 +  
## 20 101 +  
## 21 101 -  
## 22 101 +  
## 23 101 +  
## 24 101 -  
## 25 101 -  
## 26 101 -  
## 27 101 -  
## 28 101 -  
## 29 101 -  
## 30 101 -  
## 31 101 +  
## 32 101 -  
## 33 101 -  
## 34 101 +  
## 35 101 +  
## 36 101 -  
## 37 101 -  
## 38 101 +  
## 39 101 +  
## 40 101 -  
## 41 101 -  
## 42 101 -  
## 43 101 -  
## 44 101 +  
## 45 101 -  
## 46 101 +  
## 47 101 -  
## 48 101 -  
## 49 101 +  
## 50 101 -  
## 51 101 -  
## 52 101 -  
## 53 101 +  
## 54 101 +  
## 55 101 -  
## 56 101 -  
## 57 101 -  
## 58 101 -  
## 59 101 +  
## 60 101 +  
## 61 101 -  
## 62 101 -  
## 63 101 -  
## 64 101 +  
## 65 101 +  
## 66 101 +  
## 67 101 -  
## 68 101 -  
## 69 101 -  
## 70 101 +  
## 71 101 -  
## 72 101 +  
## 73 101 -  
## 74 101 -  
## 75 101 -  
## 76 101 +  
## 77 101 +  
## 78 101 -  
## 79 101 +  
## 80 101 -  
## 81 101 +  
## 82 101 -  
## 83 101 -  
## 84 101 -  
## 85 101 -  
## 86 101 -  
## 87 101 +  
## 88 101 +  
## 89 101 -  
## 90 101 -  
## 91 101 -  
## 92 101 -  
## 93 101 -  
## 94 101 +  
## 95 101 -  
## 96 101 +  
## 97 101 +  
## 98 101 +  
## 99 101 +  
## 100 101 +  
## 101 101 +  
## 102 101 +  
## 103 101 -

promotersGenes <- read\_delim("~/DataFiles/Gene Tracks/Human/hg19WithNames.bed",   
 "\t", escape\_double = FALSE, trim\_ws = TRUE)%>%makeGRangesFromDataFrame(  
 keep.extra.columns=TRUE,  
 ignore.strand=FALSE,  
 seqinfo=NULL,  
 seqnames.field=c("hg19.knownGene.chrom"),  
 start.field="hg19.knownGene.txStart",  
 end.field=c("hg19.knownGene.txEnd"),  
 strand.field="hg19.knownGene.strand",  
 starts.in.df.are.0based=FALSE)%>%promoters()

## Parsed with column specification:  
## cols(  
## .default = col\_character(),  
## hg19.knownGene.txStart = col\_integer(),  
## hg19.knownGene.txEnd = col\_integer(),  
## hg19.knownGene.cdsStart = col\_integer(),  
## hg19.knownGene.cdsEnd = col\_integer(),  
## hg19.knownGene.exonCount = col\_integer(),  
## hg19.knownGene.exonStarts = col\_number(),  
## hg19.knownGene.exonEnds = col\_number()  
## )

## See spec(...) for full column specifications.

RawReads<-readRDS("~/DataFiles/HiC/Human/HumanRawReads")  
  
##Overlap GRANGE 1 with Motifs  
Grange1OverlapsWithMotif<-subsetByOverlaps(RawReads,activeMotifs)  
  
## Grange 2 With Active promoters! Usng a vector to subset OuterGRANGE  
Grange2OverlapsWithPromoter<-subset(Grange1OverlapsWithMotif ,countOverlaps(mcols(Grange1OverlapsWithMotif)$X,  
 promotersGenes)>0)  
  
  
  
## The same as above but in the other order therefore Grange 2 overlaps with motifs and Grange 1 overlaps with motifs  
Grange2OverlapsWithMotif<-subset(RawReads ,countOverlaps(mcols(RawReads)$X,  
 activeMotifs)>0)  
Grange1OverlapsWithPromoter<-subsetByOverlaps(Grange2OverlapsWithMotif, promotersGenes)  
  
  
  
  
  
# Get the promoters Who are interacting with our active Motifs?  
PromotersInteractingWithArxMotifs<-c(subsetByOverlaps(promotersGenes,Grange2OverlapsWithPromoter$X)[isUnique(subsetByOverlaps(promotersGenes,Grange2OverlapsWithPromoter$X)$hg19.kgXref.geneSymbol)],  
subsetByOverlaps(promotersGenes,Grange1OverlapsWithPromoter)[isUnique(subsetByOverlaps(promotersGenes,Grange1OverlapsWithPromoter)$hg19.kgXref.geneSymbol)])%>%unlist()%>%as.data.frame()  
  
cbind.data.frame(  
PromotersInteractingWithArxMotifs$hg19.kgXref.geneSymbol,  
PromotersInteractingWithArxMotifs$start,  
PromotersInteractingWithArxMotifs$end,  
PromotersInteractingWithArxMotifs$hg19.kgXref.description)%>%kable()

|  |  |  |  |
| --- | --- | --- | --- |
| PromotersInteractingWithArxMotifs$hg19.kgXref.geneSymbol | PromotersInteractingWithArxMotifs$start | PromotersInteractingWithArxMotifs$end | PromotersInteractingWithArxMotifs$hg19.kgXref.description |
| AK298300 | 67142511 | 67144710 | Homo sapiens cDNA FLJ59040 complete cds, highly similar to LINE-1 reverse transcriptase homolog. |
| MIR2682 | 98510708 | 98512907 | Homo sapiens microRNA 2682 (MIR2682), microRNA. |
| MIR137 | 98511528 | 98513727 | Homo sapiens microRNA 137 (MIR137), microRNA. |
| MIR137HG | 98511753 | 98513952 | Homo sapiens MIR137 host gene (non-protein coding) (MIR137HG), non-coding RNA. |
| CGREF1 | 27333771 | 27335970 | Homo sapiens cell growth regulator with EF-hand domain 1 (CGREF1), transcript variant 4, mRNA. |
| BC048132 | 28207694 | 28209893 | Homo sapiens cDNA clone IMAGE:5303859. |
| LRIG1 | 66456438 | 66458637 | Homo sapiens leucine-rich repeats and immunoglobulin-like domains 1 (LRIG1), mRNA. |
| WHSC1 | 1939007 | 1941206 | Homo sapiens Wolf-Hirschhorn syndrome candidate 1 (WHSC1), transcript variant 3, mRNA. |
| GZMA | 54396473 | 54398672 | Homo sapiens granzyme A (granzyme 1, cytotoxic T-lymphocyte-associated serine esterase 3) (GZMA), mRNA. |
| ATG10 | 81281377 | 81283576 | SubName: Full=Ubiquitin-like-conjugating enzyme ATG10; Flags: Fragment; |
| TBP | 170863969 | 170866168 | Homo sapiens TATA box binding protein (TBP), transcript variant 1, mRNA. |
| TCRGV | 38402920 | 38405119 | Homo sapiens cDNA clone IMAGE:5227869, \*\*\*\* WARNING: chimeric clone \*\*\*\*. |
| FAM102A | 130712794 | 130714993 | Homo sapiens family with sequence similarity 102, member A (FAM102A), transcript variant 2, mRNA. |
| U7 | 131304194 | 131306393 | Rfam model RF00066 hit found at contig region AL356481.16/24552-24613 |
| RAPGEF1 | 134612726 | 134614925 | Homo sapiens Rap guanine nucleotide exchange factor (GEF) 1 (RAPGEF1), transcript variant 1, mRNA. |
| OLFM1 | 137995485 | 137997684 | Homo sapiens olfactomedin 1 (OLFM1), transcript variant 1, mRNA. |
| SPTY2D1-AS1 | 18622533 | 18624732 | Homo sapiens SPTY2D1 antisense RNA 1 (SPTY2D1-AS1), non-coding RNA. |
| SVIP | 22851183 | 22853382 | Homo sapiens small VCP/p97-interacting protein (SVIP), mRNA. |
| ARAP1 | 72463235 | 72465434 | Homo sapiens ArfGAP with RhoGAP domain, ankyrin repeat and PH domain 1 (ARAP1), transcript variant 3, mRNA. |
| U6 | 132393702 | 132395901 | Rfam model RF00026 hit found at contig region AP000843.5/185541-185619 |
| IGSF9B | 133807796 | 133809995 | Homo sapiens immunoglobulin superfamily, member 9B (IGSF9B), mRNA. |
| RHNO1 | 2983423 | 2985622 | Homo sapiens RAD9-HUS1-RAD1 interacting nuclear orphan 1 (RHNO1), transcript variant 7, non-coding RNA. |
| GGACT | 101240847 | 101243046 | Homo sapiens gamma-glutamylamine cyclotransferase (GGACT), transcript variant 2, mRNA. |
| DQ571799 | 97314170 | 97316369 | Homo sapiens piRNA piR-31911, complete sequence. |
| DQ595265 | 97314233 | 97316432 | Homo sapiens piRNA piR-61377, complete sequence. |
| DQ590811 | 97314798 | 97316997 | Homo sapiens piRNA piR-57923, complete sequence. |
| DQ582375 | 97315017 | 97317216 | Homo sapiens piRNA piR-32487, complete sequence. |
| DQ595383 | 97315364 | 97317563 | Homo sapiens piRNA piR-44114, complete sequence. |
| DQ590792 | 97318600 | 97320799 | Homo sapiens piRNA piR-57904, complete sequence. |
| DQ576810 | 97319712 | 97321911 | Homo sapiens piRNA piR-44922, complete sequence. |
| XYLT1 | 17564539 | 17566738 | Homo sapiens xylosyltransferase I (XYLT1), mRNA. |
| GSE1 | 85643028 | 85645227 | Homo sapiens Gse1 coiled-coil protein (GSE1), transcript variant 2, mRNA. |
| TANC2 | 61269351 | 61271550 | Homo sapiens tetratricopeptide repeat, ankyrin repeat and coiled-coil containing 2 (TANC2), mRNA. |
| BC094703 | 3601735 | 3603934 | Homo sapiens cDNA clone IMAGE:30412101. |
| HIST2H2AA4 | 149814119 | 149816318 | Homo sapiens histone cluster 2, H2aa4 (HIST2H2AA4), mRNA. |
| RHOU | 228868823 | 228871022 | Homo sapiens ras homolog family member U (RHOU), transcript variant 1, mRNA. |
| AX748369 | 229052755 | 229054954 | Homo sapiens cDNA FLJ36611 fis, clone TRACH2016080. |
| NEB | 152589500 | 152591699 | Homo sapiens nebulin (NEB), transcript variant 1, mRNA. |
| INO80D | 206950707 | 206952906 | Homo sapiens INO80 complex subunit D (INO80D), mRNA. |
| SPEG | 220313391 | 220315590 | Homo sapiens SPEG complex locus (SPEG), transcript variant 4, mRNA. |
| SORCS2 | 7381624 | 7383823 | Homo sapiens sortilin-related VPS10 domain containing receptor 2 (SORCS2), mRNA. |
| COX7B2 | 46737010 | 46739209 | Homo sapiens cytochrome c oxidase subunit VIIb2 (COX7B2), nuclear gene encoding mitochondrial protein, mRNA. |
| C5orf42 | 37176002 | 37178201 | Homo sapiens chromosome 5 open reading frame 42 (C5orf42), mRNA. |
| ISL1 | 50676957 | 50679156 | Homo sapiens ISL LIM homeobox 1 (ISL1), mRNA. |
| BC042122 | 131991647 | 131993846 | Homo sapiens cDNA clone IMAGE:4836780. |
| SMIM3 | 150155507 | 150157706 | Homo sapiens small integral membrane protein 3 (SMIM3), mRNA. |
| CCM2 | 45037344 | 45039543 | Homo sapiens cerebral cavernous malformation 2 (CCM2), transcript variant 5, non-coding RNA. |
| GSAP | 77045518 | 77047717 | Homo sapiens gamma-secretase activating protein (GSAP), mRNA. |
| TACC1 | 38643623 | 38645822 | Homo sapiens transforming, acidic coiled-coil containing protein 1 (TACC1), transcript variant 3, mRNA. |
| CRH | 67090647 | 67092846 | Homo sapiens corticotropin releasing hormone (CRH), mRNA. |
| OMD | 95186637 | 95188836 | Homo sapiens osteomodulin (OMD), mRNA. |
| FAM129B | 130341069 | 130343268 | Homo sapiens family with sequence similarity 129, member B (FAM129B), transcript variant 2, mRNA. |
| LOC100288814 | 9933397 | 9935596 | Homo sapiens uncharacterized LOC100288814 (LOC100288814), mRNA. |
| MIR3157 | 97823957 | 97826156 | Homo sapiens microRNA 3157 (MIR3157), microRNA. |
| B4GALNT3 | 567542 | 569741 | Homo sapiens beta-1,4-N-acetyl-galactosaminyl transferase 3 (B4GALNT3), mRNA. |
| DQ588300 | 3588068 | 3590267 | Homo sapiens piRNA piR-55411, complete sequence. |
| DQ596092 | 3588833 | 3591032 | Homo sapiens piRNA piR-47048, complete sequence. |
| BC055085 | 10324532 | 10326731 | Homo sapiens cDNA clone IMAGE:4619390, partial cds. |
| FLT1 | 28920227 | 28922426 | Homo sapiens fms-related tyrosine kinase 1 (FLT1), transcript variant 1, mRNA. |
| KIAA0586 | 58907423 | 58909622 | Homo sapiens KIAA0586 (KIAA0586), transcript variant 6, mRNA. |
| FAM103A1 | 83652954 | 83655153 | Homo sapiens family with sequence similarity 103, member A1 (FAM103A1), mRNA. |
| AK057887 | 84694933 | 84697132 | Homo sapiens cDNA FLJ31635 fis, clone NT2RI2003420. |
| CRHR1 | 43882375 | 43884574 | Homo sapiens corticotropin releasing hormone receptor 1 (CRHR1), transcript variant 4, mRNA. |
| SP6 | 45925596 | 45927795 | Homo sapiens Sp6 transcription factor (SP6), mRNA. |

The gene list of Interacting genes with promoter Motifs using just the raw reads.

So the promoters of genes where a raw read overlaps the enhancer Arx motif and a promoter Arx

You do get more reads but MEHHH – I don’t know how valid this is.

sessionInfo(  
)

## R version 3.3.2 (2016-10-31)  
## Platform: x86\_64-pc-linux-gnu (64-bit)  
## Running under: Ubuntu 16.04.2 LTS  
##   
## locale:  
## [1] LC\_CTYPE=en\_US.UTF-8 LC\_NUMERIC=C   
## [3] LC\_TIME=en\_US.UTF-8 LC\_COLLATE=en\_US.UTF-8   
## [5] LC\_MONETARY=en\_US.UTF-8 LC\_MESSAGES=en\_US.UTF-8   
## [7] LC\_PAPER=en\_US.UTF-8 LC\_NAME=C   
## [9] LC\_ADDRESS=C LC\_TELEPHONE=C   
## [11] LC\_MEASUREMENT=en\_US.UTF-8 LC\_IDENTIFICATION=C   
##   
## attached base packages:  
## [1] parallel stats4 stats graphics grDevices utils datasets   
## [8] methods base   
##   
## other attached packages:  
## [1] readr\_1.1.1 ggbio\_1.22.4   
## [3] knitr\_1.17 GenomicInteractions\_1.8.1   
## [5] InteractionSet\_1.2.1 SummarizedExperiment\_1.4.0   
## [7] Biobase\_2.34.0 MotifDb\_1.16.1   
## [9] scales\_0.5.0 tidyr\_0.7.1   
## [11] AnnotationHub\_2.6.5 reshape2\_1.4.2   
## [13] tibble\_1.3.4 pander\_0.6.1   
## [15] magrittr\_1.5 ggplot2\_2.2.1   
## [17] BSgenome.Mmusculus.UCSC.mm9\_1.4.0 BSgenome.Hsapiens.UCSC.hg19\_1.4.0  
## [19] BSgenome\_1.42.0 Biostrings\_2.42.1   
## [21] XVector\_0.14.1 rtracklayer\_1.34.2   
## [23] GenomicRanges\_1.26.4 GenomeInfoDb\_1.10.3   
## [25] IRanges\_2.8.2 S4Vectors\_0.12.2   
## [27] BiocGenerics\_0.20.0   
##   
## loaded via a namespace (and not attached):  
## [1] bitops\_1.0-6 matrixStats\_0.52.2   
## [3] bit64\_0.9-7 RColorBrewer\_1.1-2   
## [5] httr\_1.3.1 rprojroot\_1.2   
## [7] tools\_3.3.2 backports\_1.1.0   
## [9] R6\_2.2.2 rpart\_4.1-11   
## [11] Hmisc\_4.0-3 DBI\_0.7   
## [13] lazyeval\_0.2.0 Gviz\_1.18.2   
## [15] colorspace\_1.3-2 nnet\_7.3-12   
## [17] GGally\_1.3.2 gridExtra\_2.3   
## [19] bit\_1.1-12 graph\_1.52.0   
## [21] htmlTable\_1.9 checkmate\_1.8.3   
## [23] RBGL\_1.50.0 stringr\_1.2.0   
## [25] digest\_0.6.12 Rsamtools\_1.26.2   
## [27] foreign\_0.8-69 rmarkdown\_1.6   
## [29] pkgconfig\_2.0.1 base64enc\_0.1-3   
## [31] dichromat\_2.0-0 htmltools\_0.3.6   
## [33] highr\_0.6 ensembldb\_1.6.2   
## [35] htmlwidgets\_0.9 rlang\_0.1.2   
## [37] RSQLite\_2.0 BiocInstaller\_1.24.0   
## [39] shiny\_1.0.5 bindr\_0.1   
## [41] BiocParallel\_1.8.2 dplyr\_0.7.3   
## [43] acepack\_1.4.1 VariantAnnotation\_1.20.3   
## [45] RCurl\_1.95-4.8 Formula\_1.2-2   
## [47] Matrix\_1.2-11 Rcpp\_0.12.12   
## [49] munsell\_0.4.3 stringi\_1.1.5   
## [51] yaml\_2.1.14 zlibbioc\_1.20.0   
## [53] plyr\_1.8.4 grid\_3.3.2   
## [55] blob\_1.1.0 lattice\_0.20-35   
## [57] splines\_3.3.2 GenomicFeatures\_1.26.4   
## [59] hms\_0.3 igraph\_1.1.2   
## [61] biomaRt\_2.30.0 XML\_3.98-1.9   
## [63] glue\_1.1.1 evaluate\_0.10.1   
## [65] biovizBase\_1.22.0 latticeExtra\_0.6-28   
## [67] data.table\_1.10.4 httpuv\_1.3.5   
## [69] gtable\_0.2.0 purrr\_0.2.3   
## [71] reshape\_0.8.7 assertthat\_0.2.0   
## [73] mime\_0.5 xtable\_1.8-2   
## [75] survival\_2.41-3 OrganismDbi\_1.16.0   
## [77] GenomicAlignments\_1.10.1 AnnotationDbi\_1.36.2   
## [79] memoise\_1.1.0 bindrcpp\_0.2   
## [81] cluster\_2.0.6 interactiveDisplayBase\_1.12.0