Resolving ambiguous motifs with ChIP-seq

Michael Lawrence

January 21, 2009

Outline

Resolving motifs

- DNA binding motifs often have ambiguous consensus sequences

Example

`CANNTG`

- The islands (bound regions) can help resolve the consensus
- Three step process:
  1. Find regions matching consensus sequence
  2. Tabulate the matching sequences under a variety of filters: peaks, promoters, etc.
  3. Compare the counts, e.g. are some sequences over represented under the peaks and in promoters?

Outline

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome

Code

```
> library(BSgenome.Mmusculus.UCSC.mm9)
```

2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`

Code

```
> NN <- mkAllStrings(c("A","C","G","T"), 2)
> motifs <- DNAStringSet(paste("CA",NN,"TG",sep=""))
> pD <- PDict(motifs)
> motifs <- as.character(motifs)
```

3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function

Code

```
> findMotifs <- function(chr) {
+   mindex <- matchPDict(pD, chr)
+   seq <- rep(motifs, countIndex(mindex))
+   gd <- RangedData(unlist(mindex), seq)
+   gd[order(start(gd)),]
+ }
```

4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Finding CANNTG in the mouse genome

An application of bsapply()

Perform matching across autosomal chromosomes:

1. Load the mouse genome
2. Initialize `PDict` with variants of `CANNTG`
3. Define matching function
4. Invoke `bsapply()` and reduce result to *RangedData*

Code

```
> params <- new("BSParams", X = Mmusculus,
+                   FUN = findMotifs,
+                   exclude = "[_MXY]")
> motifLocs <- do.call("c", bsapply(params))
```

Outline

Tabulating the matching sequences

An application of rdapply()

- Count sequences over all chromosomes using `rdapply`
- Use filters to separately count sequences occurring:
  - Anywhere in the genome
  - Within peaks
  - Within promoters

Preparing the filters

Island filter Use the peaks with depth >= 8

Promoter filter Find the promoters

Preparing the filters

Island filter Use the peaks with depth >= 8

Code

```
> load("../data/alignedLocs.rda")
> library(chipseq)
> sample1 <- alignedLocs$sample1
> extended <- extendReads(as.list(sample1))
> callPeaks <- function(chr) {
+   cov <- coverage(chr, start = 1,
+                   end = max(end(chr)))
+   slice(cov, 8)
+ }
> peaks <- lapply(extended, callPeaks)
```

Promoter filter Find the promoters

Preparing the filters

Island filter Use the peaks with depth >= 8

Promoter filter Find the promoters

Preparing the filters

Island filter Use the peaks with depth >= 8

Promoter filter Find the promoters

Code

```
> data(geneMouse)
> regions <- genomic_regions(geneMouse)
> promRanges <- IRanges(regions$promoter.start,
+                       regions$promoter.end)
> promoters <- split(promRanges, regions$chrom)
```

Preparing to count

1. Define filter rules
2. Define counting function
3. Define reducing function to aggregate counts
4. Construct *RDApplyParams*

Preparing to count

1. Define filter rules

Code

```
> overlapFilter <- function(x) {
+   function(rd)
+     ranges(rd)[[1]] %in% x[[names(rd)]]
+ }
> promoterFilter <- overlapFilter(promoters)
> peakFilter <- overlapFilter(peaks)
> filters <- list(promoter = promoterFilter,
+                 peak = peakFilter)
> rules <- FilterRules(filters, active = FALSE)
```

2. Define counting function
3. Define reducing function to aggregate counts
4. Construct *RDApplyParams*

Preparing to count

1. Define filter rules
2. Define counting function
3. Define reducing function to aggregate counts
4. Construct *RDApplyParams*

Preparing to count

1. Define filter rules
2. Define counting function

Code

```
> count_motifs <- function(rd) {
+   nn <- substring(rd[["seq"]], 3, 4)
+   df <- as.data.frame(table(factor(nn, NN)))
+   colnames(df) <- c("seq", "count")
+   df
+ }
```

3. Define reducing function to aggregate counts
4. Construct *RDApplyParams*

Preparing to count

1. Define filter rules
2. Define counting function
3. Define reducing function to aggregate counts
4. Construct *RDApplyParams*

Preparing to count

1. Define filter rules
2. Define counting function
3. Define reducing function to aggregate counts

Code

```
> reduce_counts <- function(counts) {
+   counts <- do.call("rbind", counts)
+   counts <- aggregate(counts[,2,drop=FALSE],
+                       list(seq = counts$seq), sum)
+   counts$freq <- counts$count / sum(counts$count)
+   counts
+ }
```

4. Construct *RDApplyParams*

Preparing to count

1. Define filter rules
2. Define counting function
3. Define reducing function to aggregate counts
4. Construct *RDApplyParams*

Code

```
> rda <- RDApplyParams(motifLocs, count_motifs,
+                      filterRules = rules,
+                      reducerFun = reduce_counts)
```

Counting the variants of CANNTG

1. Over the entire genome
2. Within the peaks
3. Within the peaks and under the peaks
4. Compare the results

Counting the variants of CANNTG

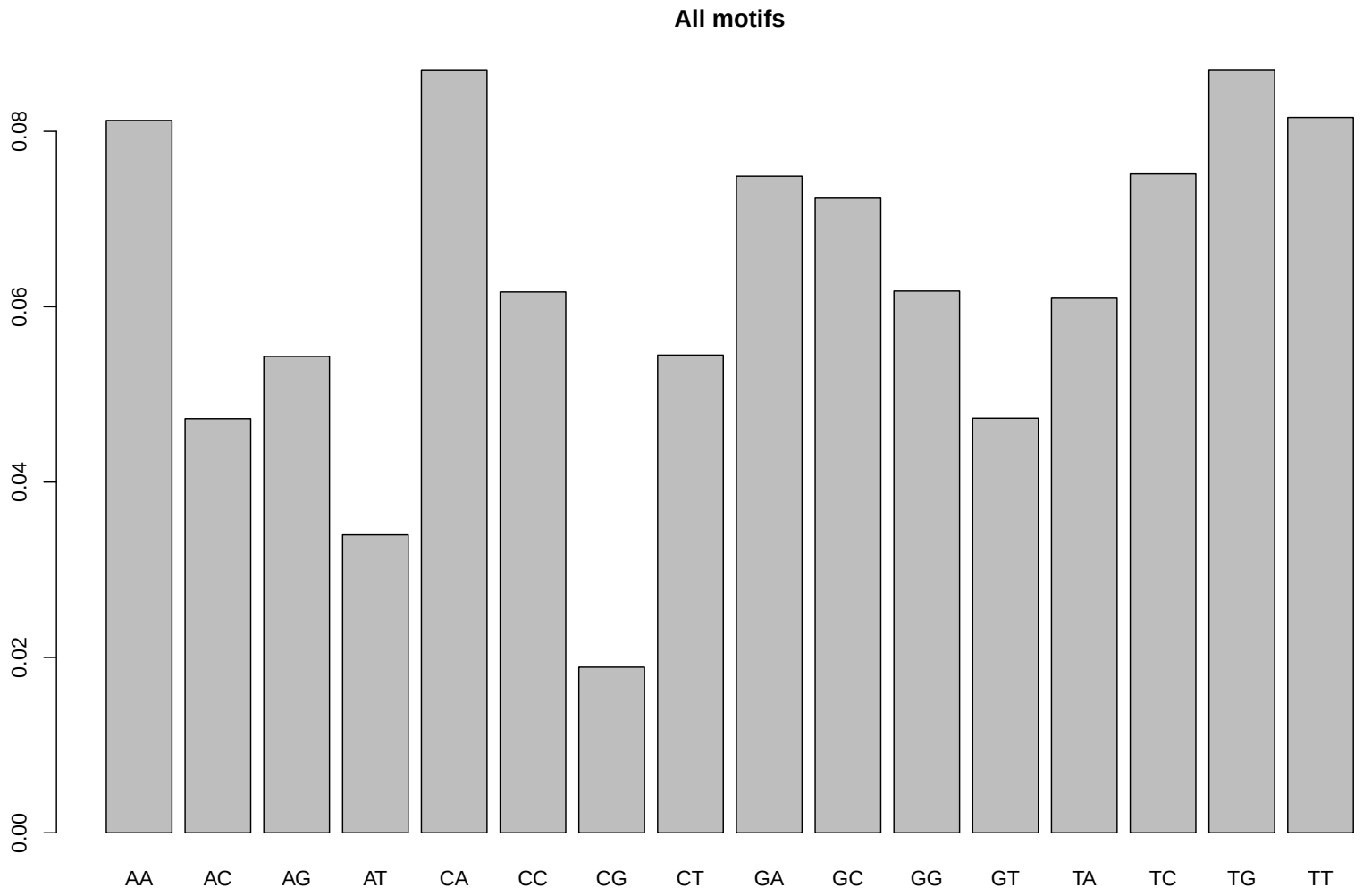
1. Over the entire genome

Code

```
> allCounts <- rdapply(rda)
```

2. Within the peaks
3. Within the peaks and under the peaks
4. Compare the results

Counting the variants of CANNTG



Counting the variants of CANNTG

1. Over the entire genome
2. Within the peaks
3. Within the peaks and under the peaks
4. Compare the results

Counting the variants of CANNTG

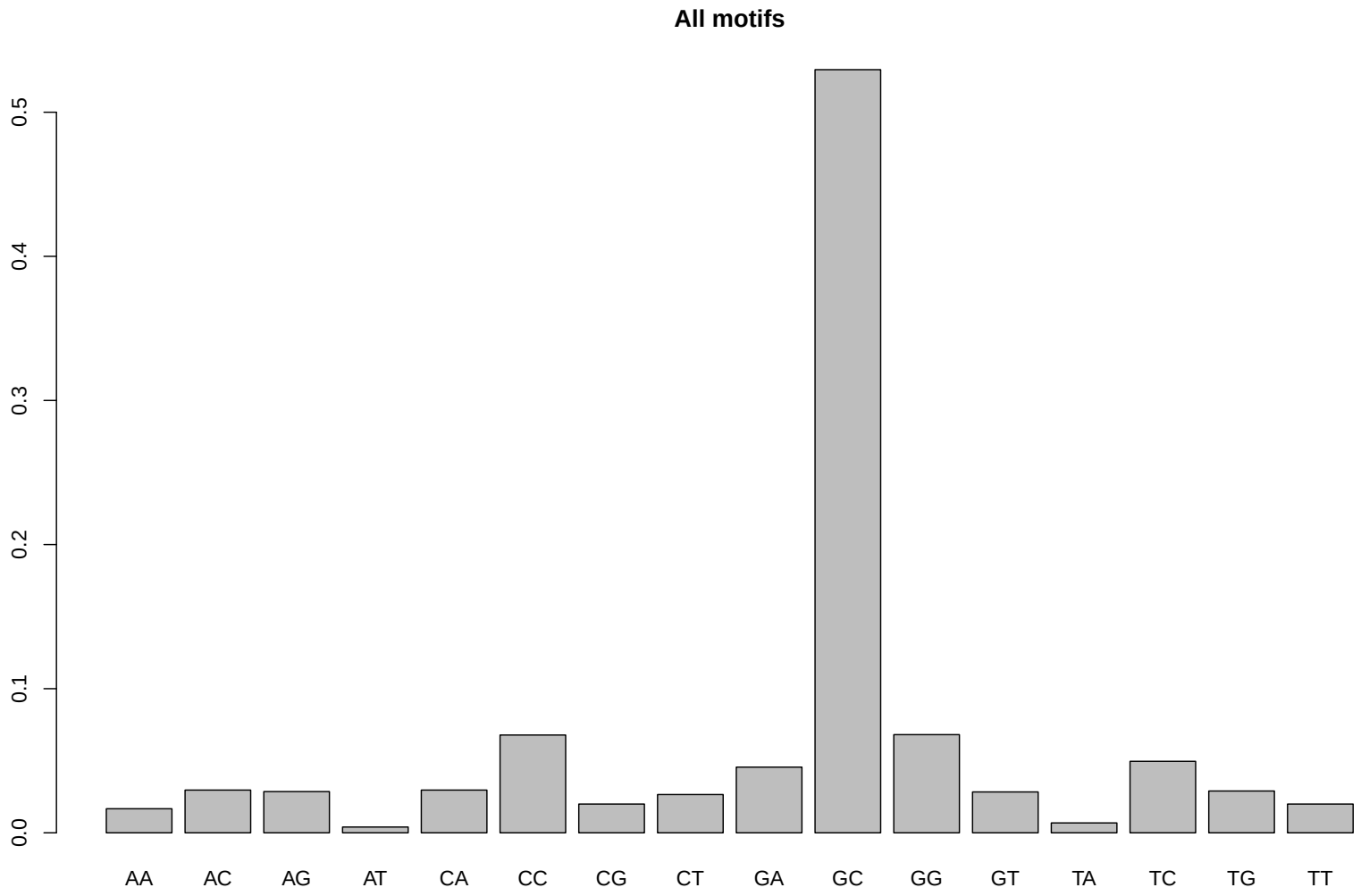
1. Over the entire genome
2. Within the peaks

Code

```
> active(filterRules(rda))["peak"] <- TRUE
> peakCounts <- rdapply(rda)
```

3. Within the peaks and under the peaks
4. Compare the results

Counting the variants of CANNTG



Counting the variants of CANNTG

1. Over the entire genome
2. Within the peaks
3. Within the peaks and under the peaks
4. Compare the results

Counting the variants of CANNTG

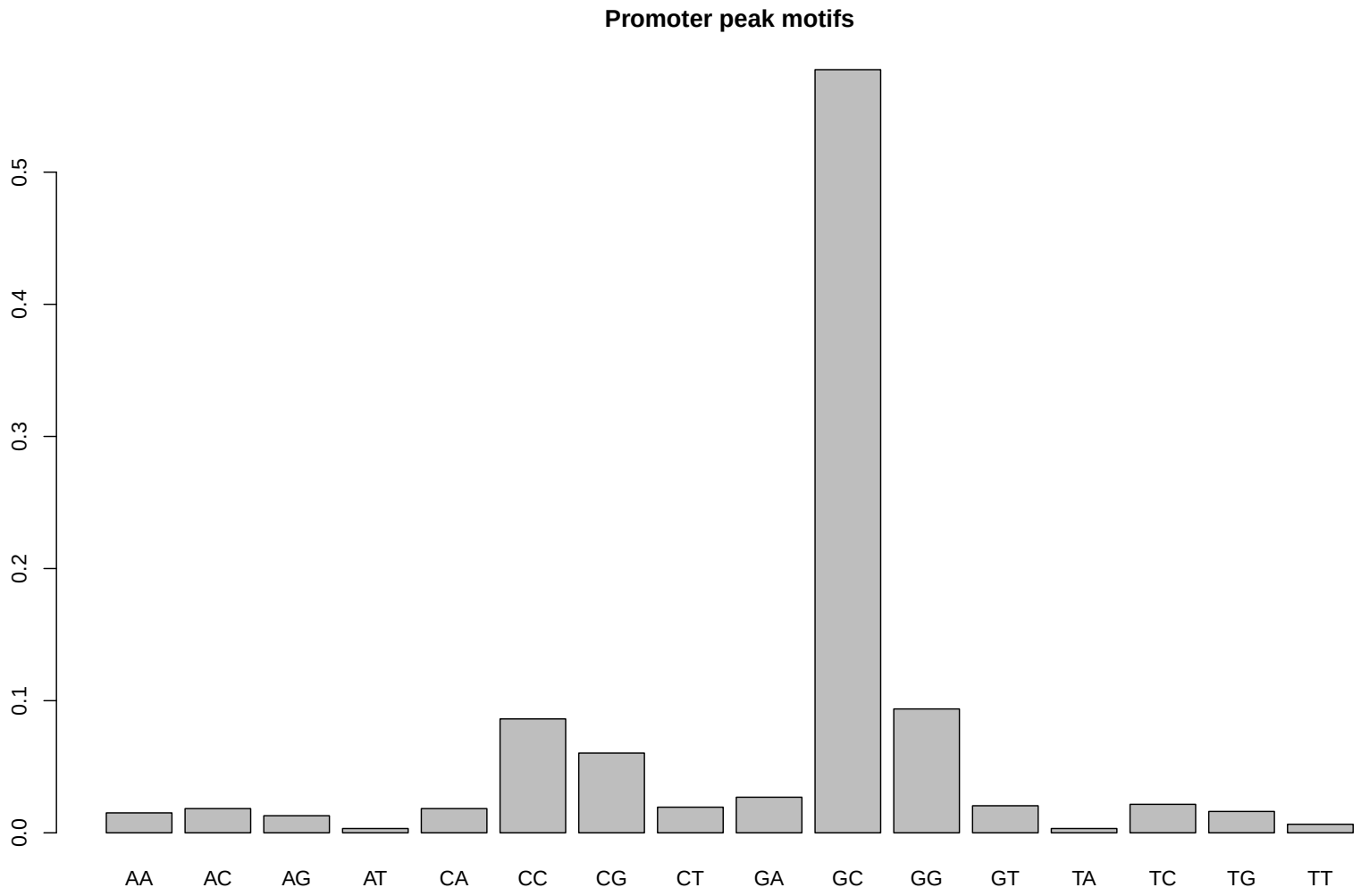
1. Over the entire genome
2. Within the peaks
3. Within the peaks and under the peaks

Code

```
> active(filterRules(rda))["promoter"] <- TRUE
> promoterCounts <- rdapply(rda)
```

4. Compare the results

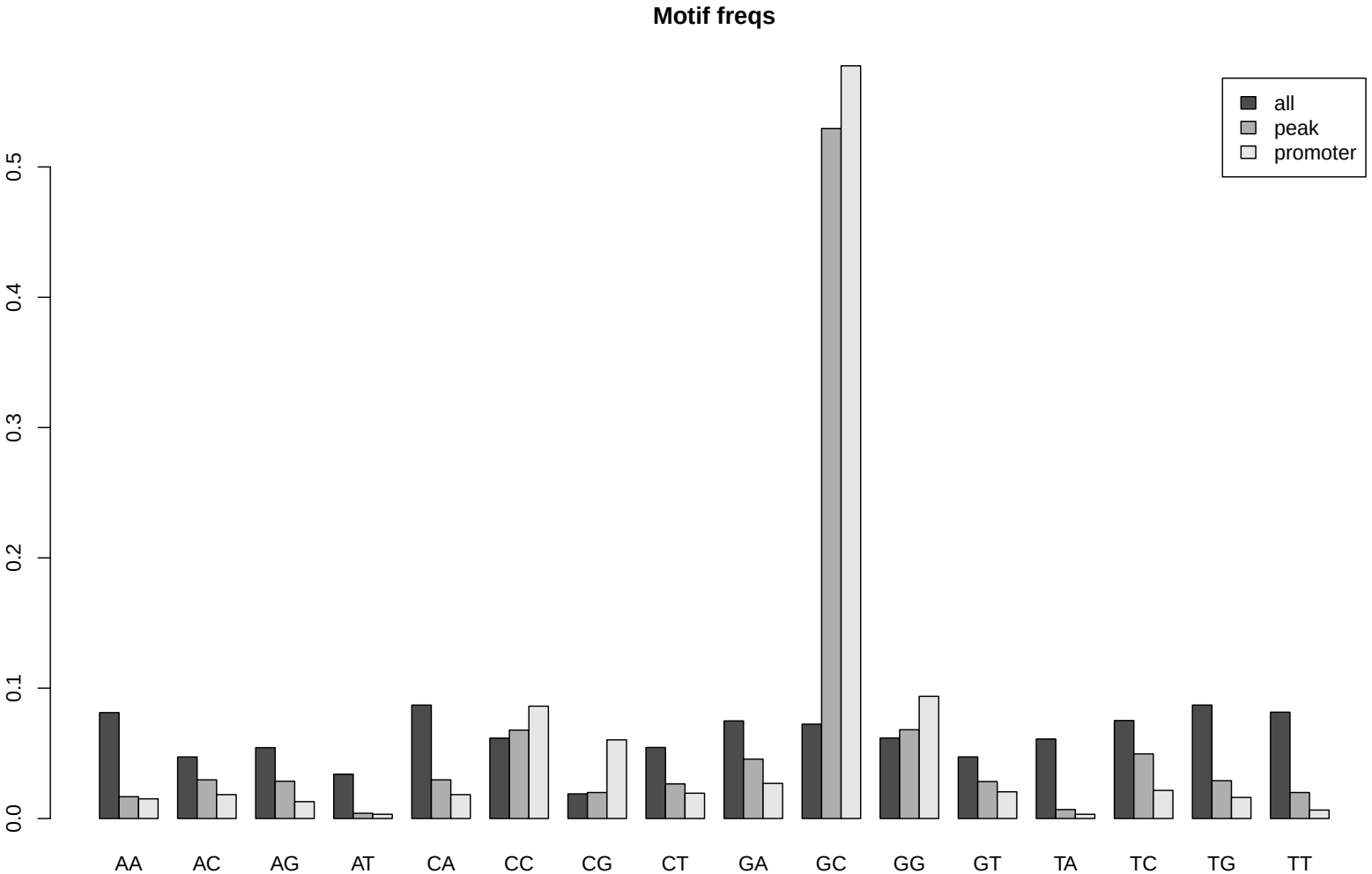
Counting the variants of CANNTG



Counting the variants of CANNTG

1. Over the entire genome
2. Within the peaks
3. Within the peaks and under the peaks
4. Compare the results

Counting the variants of CANNTG



Session info

```
> sessionInfo()

R version 2.9.0 Under development (unstable) (--)
i686-pc-linux-gnu

locale:
C

attached base packages:
[1] tools     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] chipseq_0.1.8                      ShortRead_1.1.26
[3] lattice_0.17-20                    Biobase_2.3.5
[5] BSgenome.Mmusculus.UCSC.mm9_1.3.11 BSgenome_1.11.9
[7] Biostrings_2.11.18                 IRanges_1.1.34

loaded via a namespace (and not attached):
[1] Matrix_0.999375-17 grid_2.9.0
```