

Elastic Net Poisson GLM Spatial Profile Vignette

This script implements the elastic net Poisson Generalized Linear Model used to generate the spatial profiles in Figure 3. It uses fragment overlap with 50bp genomic sequence bins to predict expression in a multiplicative manner. For elastic net regularization, we used the R package `glmnet`. We also make extensive use of the `GenomicRanges` package for generating the model matrix and storing the model results.

Libraries

```
library(GenomicRanges)
library(glmnet)
library(rtracklayer)
```

Fitting a single promoter spatial profile

First, we generate a single spatial profile for the 2kb region around the NUP214 TSS.

```
chr <- "chr9"
tss <- 134000976 #model will be centered here
antitss <- 134000822 #based on GRO-cap
window <- 2000 #size of the region to model
binsize <- 50 #size of the disjoint regions corresponding to columns of the model matrix

range <- GRanges(seqnames = chr,
                 ranges = IRanges(start = tss,
                                   width = 1,
                                   names = "NUP214"),
                 strand = "*")

range <- resize(range, width = window + binsize, fix = "center")
range
```

```
## GRanges object with 1 range and 0 metadata columns:
##           seqnames           ranges strand
##           <Rle>             <IRanges> <Rle>
##    NUP214      chr9 [133999951, 134002000]      *
##    -----
##    seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Previously, we generated a `GRanges` object containing the position and SuRE counts for all fragments within a 5kb window of an annotated TSS. The counts used for the model are the sum of the three K562 replicates.

```
sure <- readRDS("sureOverReducedTSS5K.rds")
minL <- 1500 #maximum fragment size
sure <- sure[width(sure)<minL]
mcols(sure) <- DataFrame(score = rowSums(as.data.frame(mcols(sure))[,3:5]))
sure
```

```
## GRanges object with 3863250 ranges and 1 metadata column:
##           seqnames           ranges strand |      score
##           <Rle>           <IRanges> <Rle> | <numeric>
##      [1]    chr1    [759009, 760415]    + |         0
##      [2]    chr1    [759110, 760418]    - |         0
##      [3]    chr1    [759264, 760711]    + |         0
##      [4]    chr1    [759265, 760639]    + |         0
##      [5]    chr1    [759296, 760605]    + |         0
##      ...      ...      ...      ...      ...
## [3863246]   chrX [155112956, 155113853]    + |         0
## [3863247]   chrX [155113047, 155114505]    - |         0
## [3863248]   chrX [155113214, 155114105]    - |         2
## [3863249]   chrX [155113307, 155114164]    + |         0
## [3863250]   chrX [155113478, 155113654]    + |         1
## -----
## seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

```
ranges <- subsetByOverlaps(sure, range, ignore.strand=T) #ranges overlapping NUP214 TSS region
```

The following function will generate a series of disjoint bins starting at the upstream end of the region (offset by a provided shift). These will repeat on each strand, but will be omitted if they do not overlap any fragments on a given strand.

```
mkBins <- function(ranges, binsize, shift, reduced_ranges = NULL) {
  if(is.null(reduced_ranges)) {
    reduced_ranges <- reduce(ranges, ignore.strand = T)
  }
  start(reduced_ranges) <- start(reduced_ranges) + shift
  bin_starts <- seq(start(reduced_ranges),
                    end(reduced_ranges),
                    binsize)
  bin_chr <- rep(seqnames(reduced_ranges),
                 times = length(bin_starts))
  bin_ranges <- IRanges(start = bin_starts,
                       width = binsize)
  bins <- GRanges(c(bin_chr, bin_chr),
                  ranges = c(bin_ranges, bin_ranges),
                  strand = rep(c("+", "-"), each = length(bin_ranges)))
  names(bins) <- paste(seqnames(bins),
                      start(bins),
                      strand(bins),
                      sep = "_")
  bins <- subsetByOverlaps(bins, ranges)
  return(bins)
}
```

Here's what the output looks like:

```
bins <- mkBins(ranges = ranges,
               binsize = binsize,
               shift = 0,
               reduced_ranges = range)

bins
```

```
## GRanges object with 82 ranges and 0 metadata columns:
##           seqnames           ranges strand
##           <Rle>             <IRanges> <Rle>
## chr9_133999951_+ chr9 [133999951, 134000000] +
## chr9_134000001_+ chr9 [134000001, 134000050] +
## chr9_134000051_+ chr9 [134000051, 134000100] +
## chr9_134000101_+ chr9 [134000101, 134000150] +
## chr9_134000151_+ chr9 [134000151, 134000200] +
## ...           ...           ...
## chr9_134001751_- chr9 [134001751, 134001800] -
## chr9_134001801_- chr9 [134001801, 134001850] -
## chr9_134001851_- chr9 [134001851, 134001900] -
## chr9_134001901_- chr9 [134001901, 134001950] -
## chr9_134001951_- chr9 [134001951, 134002000] -
## -----
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

These bins can be used to generate the design matrix. Each entry in the matrix will contain a 0 if a fragment (row) does not overlap a given 50bp bin (column), a 1 if they overlap completely, and a number between 0 and 1 if a fragment only partially overlaps a bin, representing the fraction of the bin that is overlapped. Because the matrix is sparse, we use a `dgCMatrix` object from the `Matrix` package.

```
mkBinMatrix <- function(bins, ranges, binsize){
  fo <- findOverlaps(ranges, bins)
  i <- queryHits(fo)
  j <- subjectHits(fo)

  intersect_ranges <- pintersect(ranges[i], bins[j])
  intersect_width <- width(intersect_ranges)/binsize

  mat <- sparseMatrix(i = i,
                      j = j,
                      x = intersect_width,
                      dimnames = list(NULL, names(bins)),
                      dims = c(length(ranges), length(bins)))

  return(mat)
}
```

Here is what the model matrix looks like:

```
X <- mkBinMatrix(bins, ranges, binsize)
head(X)
```

```
## 6 x 82 sparse Matrix of class "dgCMatrix"
```

```
##      [[ suppressing 82 column names 'chr9_133999951_+', 'chr9_134000001_+', 'chr9_134000051_+' ... ]]

##
## [1,] 1 1 1.0 1 1 1.00 0.38 . . . . .
## [2,] 1 1 1.0 1 1 1.00 1.00 0.94 . . . . .
## [3,] 1 1 1.0 1 1 0.48 . . . . .
```

```
## [4,] 1 1 1.0 1 1 1.00 1.00 1.00 1 1 0.82 . . . . .
## [5,] 1 1 1.0 1 1 0.36 . . . . .
## [6,] 1 1 0.5 . . . . .
##
## [1,] . . . . .
## [2,] . . . . .
## [3,] . . . . .
## [4,] . . . . .
## [5,] . . . . .
## [6,] . . . . .
##
## [1,] . . . . .
## [2,] . . . . .
## [3,] . . . . .
## [4,] . . . . .
## [5,] . . . . .
## [6,] . . . . .
```

Finally, this matrix can be combined with the SuRE counts to predict the contribution of each 50bp bin to expression.

```
#Elastic net parameters
alpha <- 0.5
lambda <- exp(0)
#LARS algorithm used by glmnet works faster when provided with
#a path of lambda values from largest to smallest.
lambda_sequence <- rev(exp(seq(log(lambda), 5, 0.5)))

counts <- score(ranges)

fit <- glmnet(X,
              y = counts,
              family = "poisson",
              alpha = alpha,
              standardize = F,
              lambda = lambda_sequence)

bins$betas <- as.numeric(coefficients(fit, s = lambda))[-1]
```

Here is a plot of the single fit exponentiated coefficients, with blue indicating the forward strand, red the reverse. These exponentiated coefficients represent the predicted multiplicative contribution of each 50bp region to the expression level of each fragment. We also indicate the sense and antisense TSS, according to GRO-cap.

```
is_plus <- as.logical(strand(bins) == "+")

plot(x = (start(bins) + end(bins))[is_plus]/2,
     y = exp(bins$betas[is_plus]),
     type = "n",
     main = "NUP214 single fit",
     xlab = "chr9",
     ylab = "contribution to expression (fold-change)",
     log = "y",
```

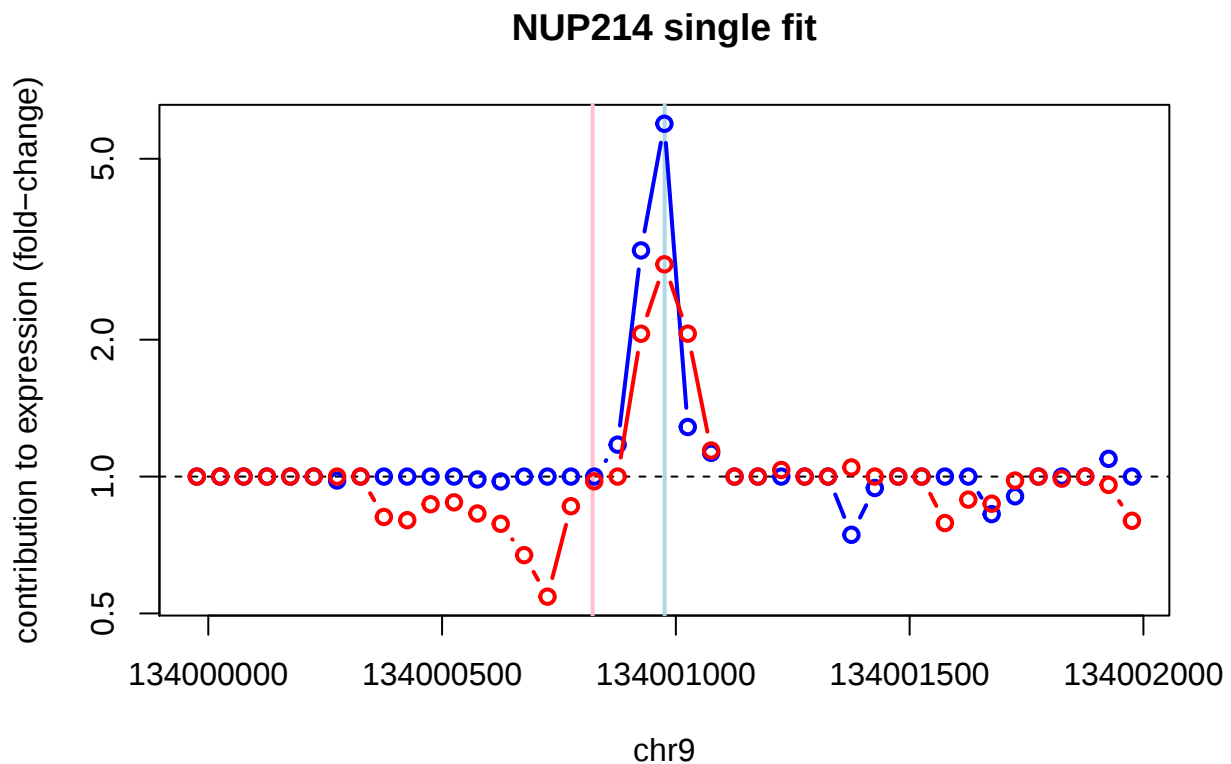
```

ylim = range(exp(bins$betas))

abline(h = 1, lwd = 1, lty = 2)
abline(v = c(tss, antitss), lwd = 2, col = c("lightblue", "pink"))

points(x = (start(bins) + end(bins))[is_plus]/2,
       y = exp(bins$betas[is_plus]),
       type = "b",
       col = "blue",
       lwd = 2)
points(x = (start(bins) + end(bins))[!is_plus]/2,
       y = exp(bins$betas[!is_plus]),
       type = "b",
       col = "red",
       lwd = 2)

```



Fitting a single promoter shifted spatial profile

To avoid any bias that may result from the choice of bin start positions relative to the TSS, we can repeat the procedure above using shifted versions of the same bins, then finding the average bin coefficient at each position. By dividing these average coefficients by the binsize, the coefficients can be interpreted as the predicted per-basepair contribution to expression.

```

# Wrapper for the commands above
spatial.fit <- function(ranges, counts, lambda, alpha,
                       binsize = 50, shift = 0, reduced_ranges = NULL){

  lambda_sequence <- rev(exp(seq(log(lambda), 5, 0.5)))

  bins <- mkBins(ranges, binsize, shift, reduced_ranges)
  X <- mkBinMatrix(bins, ranges, binsize)

  fit <- glmnet(X,
               y = counts,
               family = "poisson",
               alpha = alpha,
               standardize = F,
               lambda = lambda_sequence)

  bins$betas <- as.numeric(coefficients(fit, s = lambda))[-1]

  return(bins)
}

# Function to iterate over multiple shifts and output per-basepair average coefficient
shifted.spatial.fits <- function(shifts = NULL, ranges, counts, lambda, alpha,
                                binsize = 50, reduced_ranges = NULL) {

  if(is.null(shifts)){shifts <- (1 - binsize):0}

  if(is.null(reduced_ranges)) {
    reduced_ranges <- reduce(ranges, ignore.strand = T)
  }

  fit_bins <- lapply(shifts, function(shift){
    spatial.fit(ranges,
               counts,
               lambda,
               alpha,
               binsize,
               shift,
               reduced_ranges)
  })

  all_bins <- do.call("c",fit_bins)

  nz.disjoint <- disjoint(all_bins)
  nz.disjoint <- subsetByOverlaps(nz.disjoint, reduced_ranges)

  fo <- findOverlaps(nz.disjoint, all_bins)

```

```

beta_per_bp <- tapply(all_bins$betas[subjectHits(fo)]/(binsize*length(shifts)),
                      queryHits(fo),
                      sum)
score(nz.disjoint) <- beta_per_bp
return(nz.disjoint)
}

```

Now the shifted fit:

```

nup214_shifted <- shifted.spatial.fits(shifts = (-25:24),
                                       ranges = ranges,
                                       counts = counts,
                                       lambda = lambda,
                                       alpha = alpha,
                                       binsize = binsize,
                                       reduced_ranges = range)

is_plus <- as.logical(strand(nup214_shifted)=="+")
plot(x = start(nup214_shifted),
     y = exp(score(nup214_shifted)),
     type = "n",
     main = "NUP214 shifted fit",
     xlab = "chr9",
     ylab = "contribution to expression (fold-change)",
     log = "y",
     ylim = range(exp(score(nup214_shifted))))

abline(h = 1, lwd = 1, lty = 2)
abline(v = c(tss, antitss), lwd = 2, col = c("lightblue", "pink"))
points(x = start(nup214_shifted)[is_plus],
       y = exp(score(nup214_shifted)[is_plus]),
       type = "l",
       col = "blue",
       lwd = 3)
points(x = start(nup214_shifted)[!is_plus],
       y = exp(score(nup214_shifted)[!is_plus]),
       type = "l",
       col = "red",
       lwd = 3)

```

NUP214 shifted fit

