Example BP scoring and classification

This Rmarkdown document walks through the relevant R code that does the bulk of the work for calling branchpoints from CoLa-seq data as described in the README for this repository. You should be able to follow along this document if you have already ran the snakemake in the repo on the included test data. That snakemake should do some preprocessing, and in this R, we will mostly just be summing log likelihoods to come up with a composite BP likelihood score. In my code I have been using words like probability and likelihood loosely to name variables. But I hope you get the gist of what I was trying to do. If anyone needs to better understand the BP calling process I used or modify it, I hope being able to see this code could be helpful?

First let's load some libraries necessary for calling BPs on this test data:

```r
library(tidyverse)
library(data.table)
library(stringr)
library(knitr)
library(readr)
```

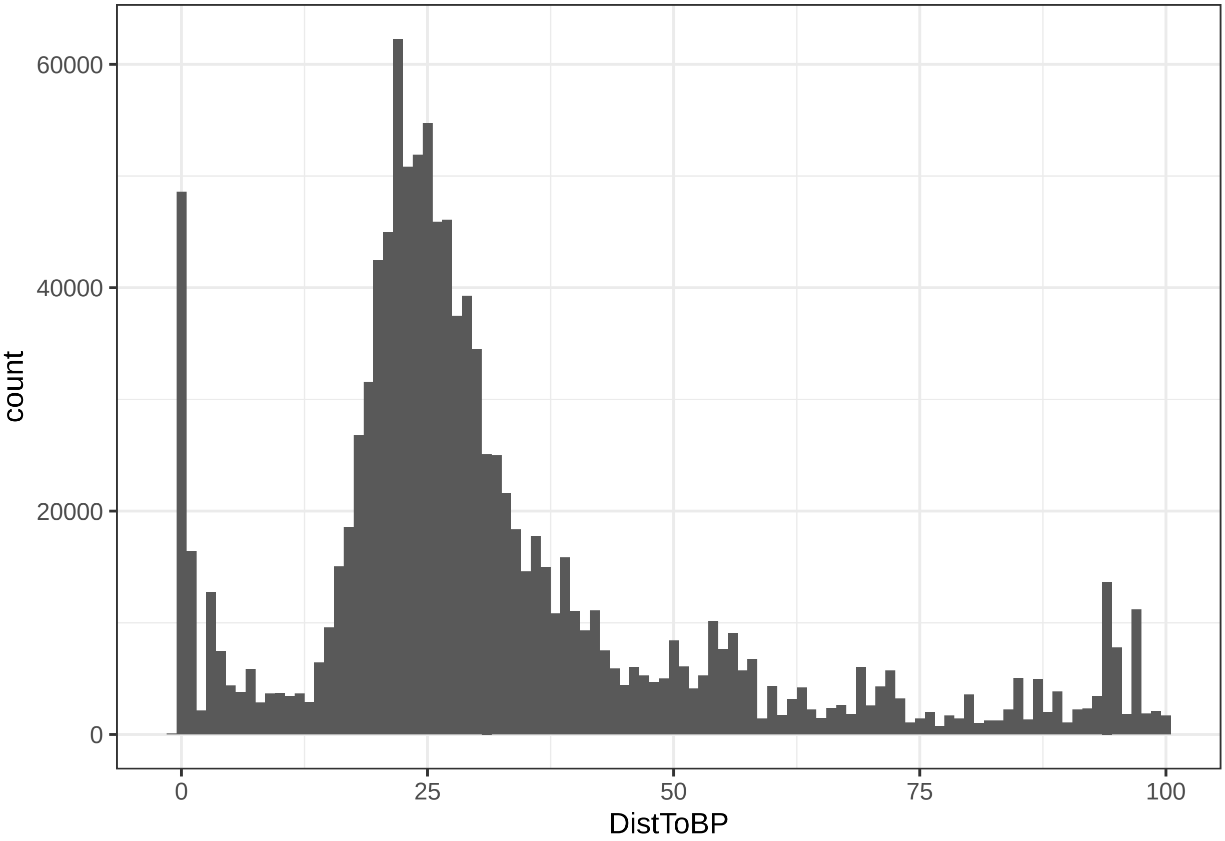
Now let's define some parameters for branchpoint calling.

```r
##Params coded as global variables
BpToDistanceTrainingData = "../code/Misc/BradleyDistTo3ss.txt.gz"
MotifProbabilityFunction = "../code/BP_calling/MergedGroup/MotifPerBaseScores.txt.gz"
ColaPutativeBranchStarts = "../code/BP_calling/MergedGroup/ColaPutativeBranchFragmentStarts.tab"

# The pseudocount of 5' ends added to each base in a region considered for BP calling
pseudocount=0.1

# The bandwidth used in the `density` function to smooth out the cola_seq 5' end
# coverage into a probability density. Bigger bandwidth means more smoothing.
# Too much smoothing obviously biases signal. But I figured a little smoothing
# might be good since we know RT doesn't always stop right at a BP, but
# sometimes is a base off.
cola_data_bandwidth=0.5

#Minimum cola-seq reads per window filter to consider the region for BP calling
MinReadsPerWindow = 10

#Distance from 3'ss to downstream edge of the window
Window_downstream_dist_to_3ss=5
Window_upstream_dist_to_3ss=100

# Composite score theshold. This threshold was used in the manuscript and was
# chosen based on analysis of precision and recall of a high confidence set of
# "gold standard" branchpoints.
Threshold=-8.881075
```

Ok. First let's create a likelihood function based on distance to 3'ss. For this, I am just reading in empirical BP-to-3'ss distances from Pineda et al, and applying what I believe is a reasonable amount of smoothing to reflect realy biology (the raw data was a bit too peaky/noisy). Note that the BP-to-3'ss distances have been weighted to the number of lariat junction read counts Pineda observed. In other words, BPs with more read counts get a porportionately stronger weight.

First let's plot the actual histogram, before creating a smoothed likelihood function

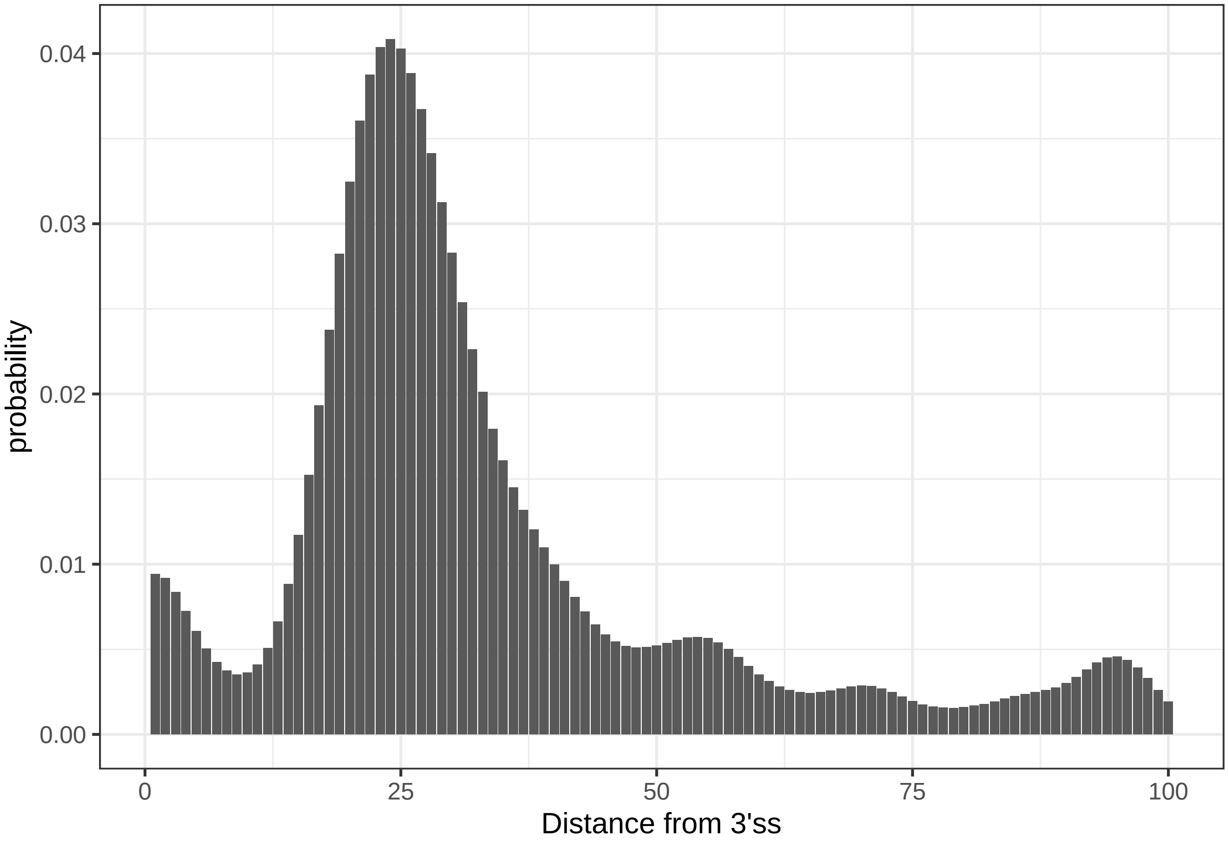
```
## Read in BP-to-3'ss distances from Pineda. This file has one line per lariat
## junction read from Pineda
lengths <- read.delim(BpToDistanceTrainingData,
                      col.names = c( "IntronType", "DistToBP" ), sep=" ") %>%
      filter(DistToBP<=Window_upstream_dist_to_3ss)

ggplot(lengths, aes(x=DistToBP)) +
  geom_histogram(binwidth=1) +
  theme_bw()
```



Ok, that is a little bit peaky which probably reflects noise and not real biology. So now let's plot a smoothed likelihood function:

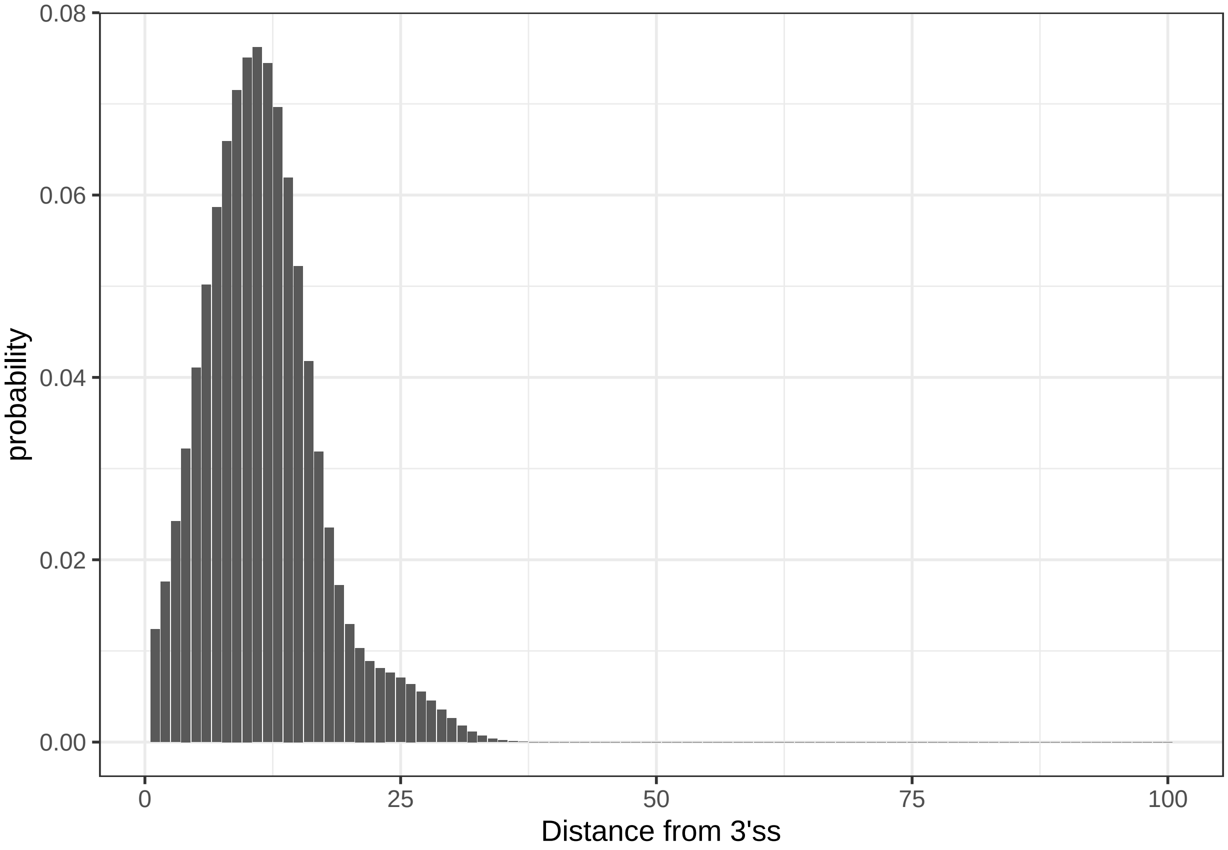
```
dx<-density(lengths %>% filter(IntronType=="u2") %>% pull(DistToBP),
            bw=3)

pmf.length <- as.data.frame(approx(dx$x,dx$y,xout=1:Window_upstream_dist_to_3ss)) %>%
  mutate(PositionScore=case_when(
    is.na(y) ~ log2(min(y, na.rm=T)),
    TRUE~ log2(y)
  )) %>%
  mutate(IntronType="u2") %>%
  dplyr::rename(RelPos=x) %>%
  dplyr::select(IntronType,RelPos,PositionScore)
```

```
PositionProbabilityFunction <-
  ggplot(pmf.length, aes(x=RelPos, y=2**PositionScore)) +
  geom_col() +
  xlab("Distance from 3'ss") +
  ylab("probability") +
  theme_bw()

PositionProbabilityFunction
```



Ok let's do the same for u12 introns which Pineda et al showed have a different positional constraint on BPs.

```
dx.u12<-density(lengths %>% filter(IntronType=="u12") %>% pull(DistToBP),
            bw=3)
pmf.length.u12 <- as.data.frame(
  approx(dx.u12$x,dx.u12$y,xout=1:Window_upstream_dist_to_3ss)) %>%
  mutate(PositionScore=case_when(
    is.na(y) ~ log2(min(y, na.rm=T)),
    TRUE~ log2(y)
  )) %>%
  mutate(IntronType="u12") %>%
  dplyr::rename(RelPos=x) %>%
  dplyr::select(IntronType,RelPos,PositionScore)

PositionProbabilityFunction.u12 <-
  ggplot(pmf.length.u12, aes(x=RelPos, y=2**PositionScore)) +
  geom_col() +
  xlab("Distance from 3'ss") +
  ylab("probability") +
  theme_bw()
PositionProbabilityFunction.u12
```



let's keep that same position-based likelihood information, for both u2 and u12 introns, in a table format, that we will use later… Keep in mind that in this table, I will deal with log (base2) probabilites, since log probabilities are easier to work with and we can just sum it with different log-likelihood scores later…

```
PositionProbabilities <- bind_rows(pmf.length.u12, pmf.length)

head(PositionProbabilities)
```

```
##   IntronType RelPos PositionScore
## 1        u12      1     -6.336468
## 2        u12      2     -5.828930
## 3        u12      3     -5.367032
## 4        u12      4     -4.957043
## 5        u12      5     -4.605851
## 6        u12      6     -4.317635
```

Ok next, we can integrate motif information for BP-likelihoods. I have used this script which is called in one of the snakemake rules, to read in the sequences under Pineda's BPs, create a position weight matrix, and score each position for it's BP motif match in terms of a log-base2 score and write it to a file. Here I will just read in that file:

```
## get motif pmf
MotifScores <- fread(MotifProbabilityFunction, check.names = F, header = T) %>%
  mutate(Intron=str_remove(Pos, "\\([+-]\\)")) %>%
  dplyr::select(-Pos) %>%
  dplyr::select(Intron, everything()) %>%
  gather(-Intron, key="RelPos", value="MotifScore") %>%
  separate(Intron,
           into=c("gene", "chrom", "windowStart", "windowStop", "strand", "IntronType"),
           sep="_", remove=F, convert=T) %>%
  mutate(RelPos=case_when(
    strand=="+" ~ as.numeric(RelPos),
    strand=="-" ~ as.numeric(RelPos) -1
  )) %>%
  dplyr::select(region=Intron, RelPos, MotifScore)


#Print some random rows so you can see what the table looks like.
sample_n(MotifScores, 20)
```

```
##                               region RelPos MotifScore
## 1  URB1_chr21_32354108_32354203_-_u2    106     -0.749
## 2  URB1_chr21_32378546_32378641_-_u2     88     -0.745
## 3  URB1_chr21_32384469_32384564_-_u2    106     -1.287
## 4  URB1_chr21_32361128_32361223_-_u2     15     -0.788
## 5  URB1_chr21_32325394_32325489_-_u2     34     -2.183
## 6  URB1_chr21_32359913_32360008_-_u2      8     -2.130
## 7  URB1_chr21_32337519_32337614_-_u2     47     -0.878
## 8  URB1_chr21_32384469_32384564_-_u2    102     -1.167
## 9  URB1_chr21_32315104_32315199_-_u2     46      0.982
## 10 URB1_chr21_32384469_32384564_-_u2      0         NA
## 11 URB1_chr21_32361128_32361223_-_u2     29      0.571
## 12 URB1_chr21_32322589_32322684_-_u2     19     -1.016
## 13 URB1_chr21_32319419_32319514_-_u2     23     -1.501
## 14 URB1_chr21_32363334_32363429_-_u2     12      0.571
## 15 URB1_chr21_32354108_32354203_-_u2    105      0.015
## 16 URB1_chr21_32384469_32384564_-_u2     74      2.875
## 17 URB1_chr21_32337162_32337257_-_u2     76     -1.889
## 18 URB1_chr21_32354108_32354203_-_u2     62     -0.959
## 19 URB1_chr21_32317070_32317165_-_u2     38     -1.683
## 20 URB1_chr21_32368603_32368698_-_u2     56     -2.753
```

Now, let's make a likelihood fuction based on cola-seq data. In the snakemake, there are rules that have already done some of the preprocessing… In particular, I have identified all the "putative" lariat derived reads that havea 5' end within 100bp upstream of a 3'ss and cross or terminate at a 3'ss. Let's read in that pre-processed data

```
## Make a smoothed prbability density function from cola-seq read ends
FragStarts <- fread(ColaPutativeBranchStarts,
                    col.names = c("feature", "Pos", "ReadName", "Extra")) %>%
  separate(feature,
           into=c("gene", "chrom", "windowStart", "windowStop", "strand", "IntronType"),
           sep="_", remove=F, convert=T) %>%
  separate(Extra, into=c("MismatchAtEnd", "ThreePrimeFragPos"), sep=";", convert=T) %>%
  mutate(ThreePrimeFragPos=ThreePrimeFragPos-1)

head(FragStarts)
```

```
##                              feature gene chrom windowStart windowStop strand
## 1: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 2: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 3: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 4: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 5: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 6: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
##    IntronType      Pos
## 1:         u2 32315116
## 2:         u2 32315123
## 3:         u2 32315123
## 4:         u2 32315123
## 5:         u2 32315123
## 6:         u2 32315123
##                                                      ReadName MismatchAtEnd
## 1:      K00242:553:H5757BBXY:8:2207:23419:46117:AGTTGC:YZ111         FALSE
## 2:      K00242:553:H5757BBXY:8:2205:28463:35936:TTCAGC:YZ111         FALSE
## 3:      K00242:592:H7MHTBBXY:5:1103:30421:30679:TCCATG:YZ111          TRUE
## 4:       K00242:592:H7MHTBBXY:5:2112:9851:44201:AAAAGC:YZ111          TRUE
## 5:       K00242:592:H7MHTBBXY:6:1127:6745:37589:CTGGAG:YZ111          TRUE
## 6: K00242:594:H7N3HBBXY:8:1206:14123:40913:GGCGGC:DMSO.YZ131         FALSE
##    ThreePrimeFragPos
## 1:          32314716
## 2:          32315002
## 3:          32315099
## 4:          32315099
## 5:          32314869
## 6:          32244278
```

Now let's Filter for 3'ss regions with at least 10 reads. Also convert chromosomal coordinate to position relative to 3'ss

```
FilteredAndAdjusted <- FragStarts %>%
  add_count(feature) %>%
  filter(n>=MinReadsPerWindow) %>%
  mutate(RelativePos=case_when(
    strand=="-" ~ Pos- windowStart + Window_downstream_dist_to_3ss,
    strand=="+" ~ windowStop - Pos + Window_downstream_dist_to_3ss
  )) %>%
  mutate(ThreeSS_Pos=case_when(
    strand=="-" ~ windowStart - Window_downstream_dist_to_3ss,
    strand=="+" ~ windowStop + Window_downstream_dist_to_3ss
  )) %>%
  mutate(Dist3PrimeEndTo3ss=case_when(
    strand=="-" ~ ThreeSS_Pos - ThreePrimeFragPos,
    strand=="+" ~ ThreePrimeFragPos - ThreeSS_Pos +1
  )) %>%
  mutate(
    LariatType=case_when(
      Dist3PrimeEndTo3ss == 0 ~ "ELI",
      TRUE ~ "Putative LI"
    ))

head(FilteredAndAdjusted)
```

```
##                                feature gene chrom windowStart windowStop strand
## 1: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 2: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 3: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 4: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 5: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
## 6: URB1_chr21_32315104_32315199_-_u2 URB1 chr21    32315104   32315199      -
##    IntronType      Pos
## 1:         u2 32315116
## 2:         u2 32315123
## 3:         u2 32315123
## 4:         u2 32315123
## 5:         u2 32315123
## 6:         u2 32315123
##                                                      ReadName MismatchAtEnd
## 1:      K00242:553:H5757BBXY:8:2207:23419:46117:AGTTGC:YZ111         FALSE
## 2:      K00242:553:H5757BBXY:8:2205:28463:35936:TTCAGC:YZ111         FALSE
## 3:      K00242:592:H7MHTBBXY:5:1103:30421:30679:TCCATG:YZ111          TRUE
## 4:       K00242:592:H7MHTBBXY:5:2112:9851:44201:AAAAGC:YZ111          TRUE
## 5:       K00242:592:H7MHTBBXY:6:1127:6745:37589:CTGGAG:YZ111          TRUE
## 6: K00242:594:H7N3HBBXY:8:1206:14123:40913:GGCGGC:DMSO.YZ131         FALSE
##    ThreePrimeFragPos  n RelativePos ThreeSS_Pos Dist3PrimeEndTo3ss  LariatType
## 1:          32314716 25          17    32315099                383 Putative LI
## 2:          32315002 25          24    32315099                 97 Putative LI
## 3:          32315099 25          24    32315099                  0         ELI
## 4:          32315099 25          24    32315099                  0         ELI
## 5:          32314869 25          24    32315099                230 Putative LI
## 6:          32244278 25          24    32315099              70821 Putative LI
```

Let's understand this data a bit more…

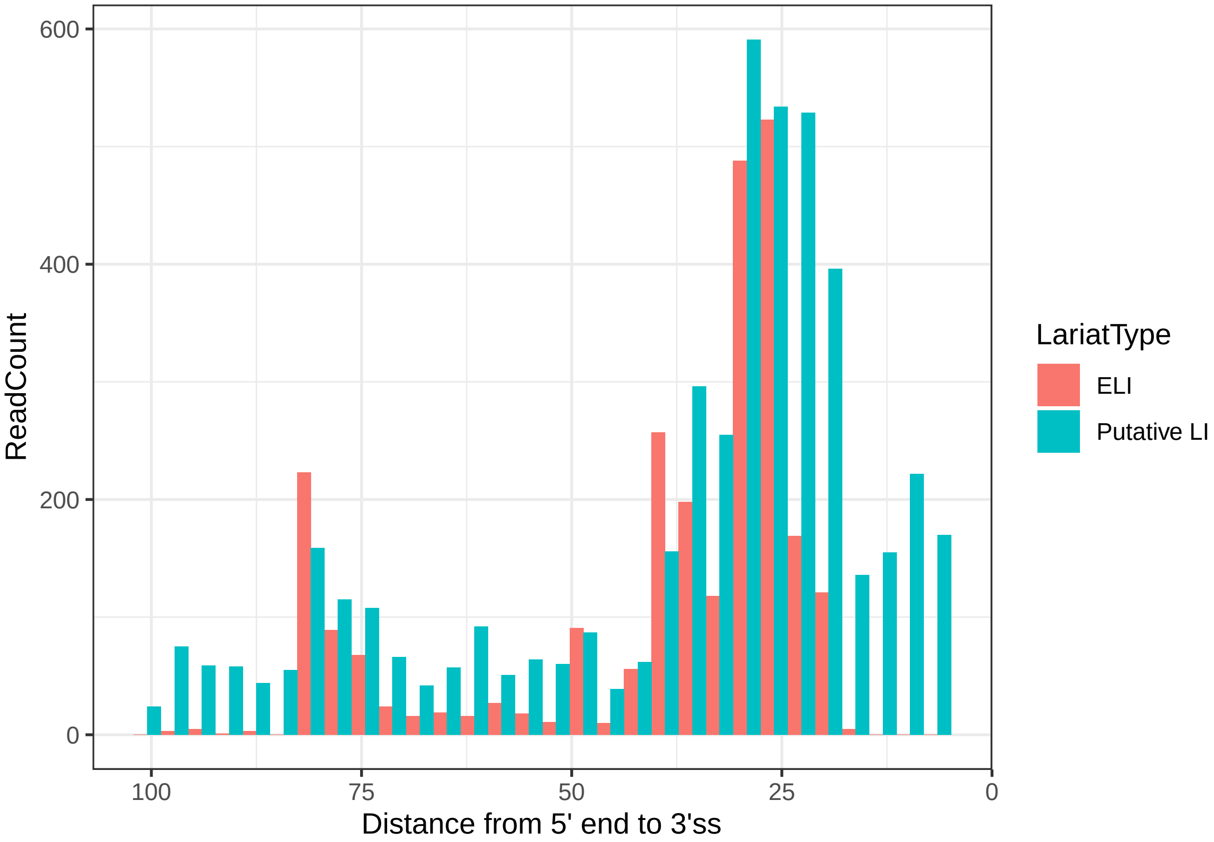
```r
#Num fragments pass filters that can be used to call branchpoints
nrow(FilteredAndAdjusted)
```

```
## [1] 7316
```

```r
#Num LI and ELI in unidows
table(FilteredAndAdjusted$LariatType)
```

```
##
##         ELI Putative LI
##        2559        4757
```

```r
#Histogram of relative 5' positions of putative NLI and ELI reads
LI_ELI.dist.hist <- FilteredAndAdjusted %>%
  ggplot(aes(x=RelativePos, fill=LariatType)) +
  geom_histogram(position="dodge") +
  scale_x_continuous(limits=c(0,100)) +
  scale_x_reverse() +
  xlab("Distance from 5' end to 3'ss") +
  ylab("ReadCount") +
  theme_bw()
LI_ELI.dist.hist
```



It looks like both putative ELI and putative LI reads have 5’ end peaks around 25-30bp upstream of 3’ss, which is to be expected for lariat derived fragments. The set of putative ELI reads may have less non-lariat derived reads than LI since terminating exactly at a 3’ss (the defining feature of putative ELI reads) probably strongly enriches for lariat derived fragments. Yi’s manuscript has some supplemental figures supporting this notion. Because of this, when I was exploring different methods to call bps, I considered considering 5’ end coverage from just putative ELI reads, or from just putative NLI reads, or from just reads with 5’ end mismatches (which are preferentially incorporated by RT at the BP structure), but all of these methods yielded very similar results, so in the end I stuck to the simplest: combinging all putative NLI and ELI 5’ ends.

Now, from all putative NLI and ELI read 5’ ends, let’s convert that to a smoothed relative coverage function and shift it up one base to get a probability function.

Here I wrote a long function to do that, which takes as input a data frame in the same format as `FilteredAndAdjusted` and will output a dataframe with log2 probabilities at each base. The function also can take an optional pseudocount parameter (to add a small amount of pseuodcoverage uniformly at each base) and optional pseudoprobability parameter (which defines the minimum probability to output). Adding pseudocounts or pseudoprobabilities is common in naive bayes classifiers, since sometimes the training data doesn’t have an observation like what the real data has, but that doesn’t necessarily mean we want to assign 0 probability to the real data.

```
## define a function to  Get probability function from colaseq data
GetColaseqPrbFunction <- function(FilteredAndAdjusted.df,
                                  pseudocount=0, bw=1,
                                  shift=1, pseudoprobability=2**-70,
                                  from=0, to=100){

  #Add a uniform distribution of counts, with relative weight determined by
```

```
#pseudocount parameter. pseudocount of 1 is equivalent to a literal
#pseudocount of 1 added at all positions
FilteredAndAdjusted.pseudocount.added <-
bind_rows(
  #Uniform distirbution
    (FilteredAndAdjusted.df %>%
    dplyr::select(feature, RelativePos) %>%
    expand(feature, RelativePos) %>%
    mutate(weight=pseudocount)),
  #Actual reads
    (FilteredAndAdjusted.df %>%
    dplyr::select(feature, RelativePos) %>%
    mutate(weight=1))
  ) %>%
#Normalize weight to sum to 1, as required by density function
group_by(feature) %>%
mutate(weightSum = sum(weight)) %>%
ungroup() %>%
mutate(RelativeWeights=weight/weightSum) %>%
dplyr::select(feature, RelativePos, RelativeWeights)

#Reformat data to make it easier to iterate through regions and calculate
#smoothed densities
group_names <- FilteredAndAdjusted.pseudocount.added %>%
  group_keys(feature) %>% pull(1)
ListOfDf <- FilteredAndAdjusted.pseudocount.added %>%
  group_split(feature) %>%
  set_names(group_names)

#I want the probability density function to be shifted from the cola-seq read
#ends by 1 to account for that cola-seq reads terminate 1 base downstream of
#branch

#Calculate points along smoothed kernel density for each region.
densities <- lapply(ListOfDf,function(i) {
  density(i$RelativePos+shift, weights = i$RelativeWeights, bw=bw)
  })

#Interpolate to get kernel density function points at discrete bases.
density.interpolations <- lapply(densities, function(i) {
  approx(i$x, i$y, xout=from:to)
  })

density.interpolations.df <- bind_rows(density.interpolations, .id = 'region')

#Add a very small (insignificant) amount of probability mass to all points to
#guarantee there are no NA or 0 values, even if there are no cola-seq reads
#and the density function sums is rounded to 0.

#Add the small proabability, and pivot the interpolated points to a more
#readable table format
output.df <- density.interpolations.df %>%
  mutate(Probability=case_when(
      is.na(y) ~ pseudoprobability,
      TRUE ~ y + pseudoprobability
    )) %>%
    dplyr::select(region, RelPos=x, Probability) %>%
    mutate(ColaSeqScore=log2(Probability)) %>%
    dplyr::select(-Probability)

  return(output.df)
}
```

Now let's use that function to get a probability function from the `FilteredAndAdjusted` data frame.

```
#Use function to get probability function
cola.seq.pmf.PerRegion <- GetColaseqPrbFunction(FilteredAndAdjusted,
                                                pseudocount=pseudocount,
                                                pseudoprobability = 2**-50,
                                                bw=cola_data_bandwidth)

head(cola.seq.pmf.PerRegion)
```

```
## # A tibble: 6 x 3
##   region                            RelPos ColaSeqScore
##   <chr>                              <int>        <dbl>
## 1 URB1_chr21_32315104_32315199_-_u2      0        -50
## 2 URB1_chr21_32315104_32315199_-_u2      1        -50
## 3 URB1_chr21_32315104_32315199_-_u2      2        -50
## 4 URB1_chr21_32315104_32315199_-_u2      3        -50
## 5 URB1_chr21_32315104_32315199_-_u2      4        -50
## 6 URB1_chr21_32315104_32315199_-_u2      5        -11.5
```

Ok, so now we have dataframes with likelihood scores for every base in every region based on distance-to-3'ss, motif, and CoLa-seq 5' end read coverage. Now we can just sum the log likelihoods to get a composite score.

But first, let's filter out the regions which may not have a score for each of those three likelihood scores.

```
#Only consider introns with motif info and that passed cola-seq filter.
IntronsToAnalyze <- intersect((cola.seq.pmf.PerRegion$region %>% unique()),
                              (MotifScores$region %>% unique()))

#Number of introns
N<-length(IntronsToAnalyze)
N
```

```
## [1] 38
```

I need to do some tidying up of the positional scores dataframes that I created above. Specifically, to make summing scores from the other data frames easier, I want to make a region column that can be easily matched to the other dataframes, and I also want to just consider positions -5 to -100 relative to 3'ss.

```
#Construct df of positional scores (log2 probabilities)
PositionalScores <- bind_rows(data.frame(region=IntronsToAnalyze),
          data.frame(RelPos=5:100)) %>%
  expand(region, RelPos ) %>%
  drop_na() %>%
  separate(region,
           into=c("gene", "chrom", "windowStart", "windowStop", "strand", "IntronType"),
           sep="_", remove=F, convert=T) %>%
  dplyr::select(region, RelPos, IntronType) %>%
  left_join(PositionProbabilities, by=c("IntronType", "RelPos")) %>%
  dplyr::select(-IntronType) %>%
  mutate(RelPos=as.numeric(RelPos))
head(PositionalScores)
```

```
## # A tibble: 6 x 3
##   region                           RelPos PositionScore
##   <chr>                             <dbl>         <dbl>
## 1 URB1_chr21_32315104_32315199_-_u2      5         -7.36
## 2 URB1_chr21_32315104_32315199_-_u2      6         -7.63
## 3 URB1_chr21_32315104_32315199_-_u2      7         -7.88
## 4 URB1_chr21_32315104_32315199_-_u2      8         -8.06
## 5 URB1_chr21_32315104_32315199_-_u2      9         -8.15
## 6 URB1_chr21_32315104_32315199_-_u2     10         -8.11
```

```
RelPosRange <- (Window_downstream_dist_to_3ss+1):Window_upstream_dist_to_3ss
```

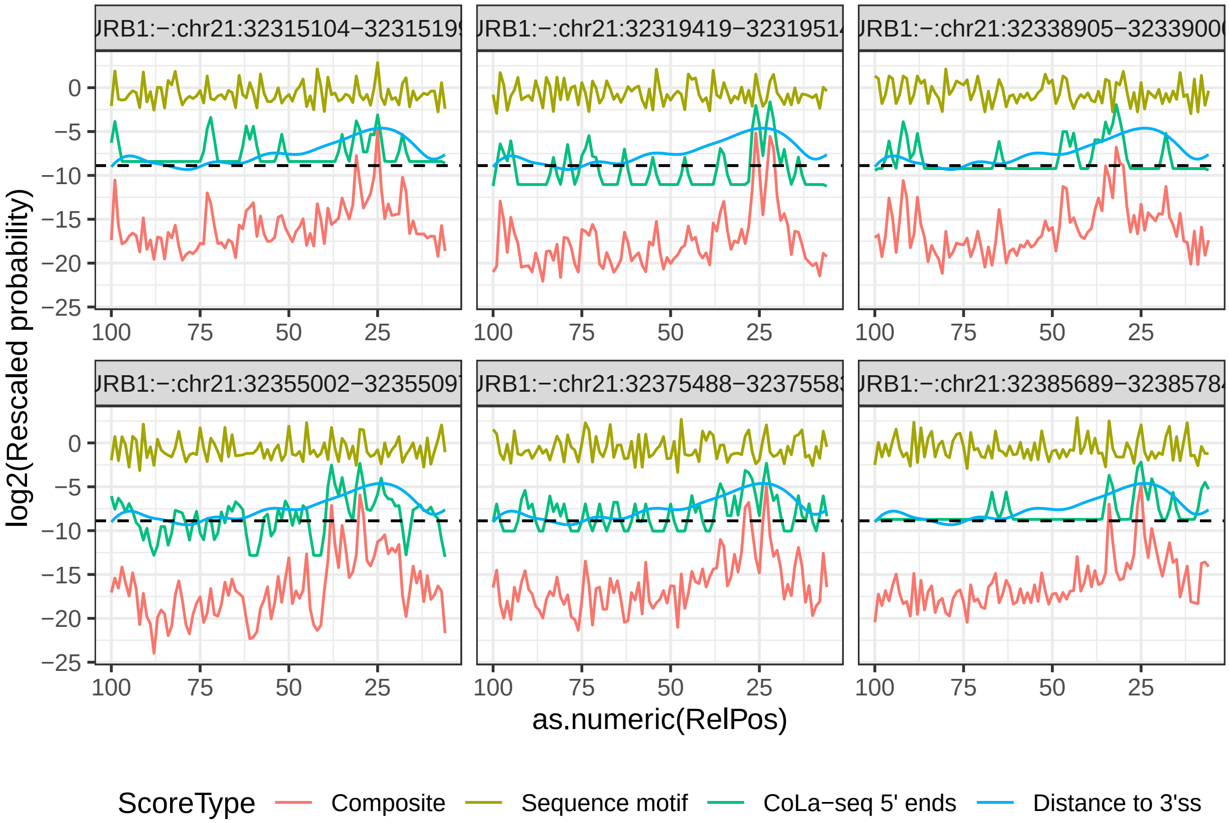
Now I think we are ready to just combine the data frames and sum the log2 probabilties to get a composite score:

```
# Add log2 probabilities
PerPosScores.df <-
  left_join(
    PositionalScores, cola.seq.pmf.PerRegion, by=c("region", "RelPos")) %>%
  left_join(
    MotifScores,by=c("region", "RelPos")) %>%
  mutate(CompositeScore=PositionScore+MotifScore+ColaSeqScore,
         NoMotifScore=PositionScore+ColaSeqScore,
         NoPositionScore=MotifScore+ColaSeqScore,
         NoColaScore=PositionScore+MotifScore) %>%
  filter(RelPos %in% RelPosRange) %>%
  separate(region,
           into=c("gene", "chrom", "windowStart", "windowStop", "strand"),
           sep = '_', remove=F, convert=T) %>%
  mutate(RelPos = as.numeric(RelPos)) %>%
  mutate(start=case_when(
    strand=="+" ~ windowStop + Window_downstream_dist_to_3ss - RelPos,
    strand=="-" ~ windowStart - Window_downstream_dist_to_3ss + RelPos
  )) %>%
  mutate(stop=start+1) %>%
  mutate(start=as.integer(start), stop=as.integer(stop)) %>%
  left_join((
            FilteredAndAdjusted %>%
                dplyr::select(region=feature, RelPos=RelativePos) %>%
                count(region, RelPos, name="ColaReadEndCount")
         ), by=c("region", "RelPos")
  )

head(PerPosScores.df)
```

```
## # A tibble: 6 x 17
##   region          gene  chrom windowStart windowStop strand RelPos PositionScore
##   <chr>           <chr> <chr>       <int>      <int> <chr>   <dbl>         <dbl>
## 1 URB1_chr21_323~ URB1  chr21    32315104   32315199 -           6         -7.63
## 2 URB1_chr21_323~ URB1  chr21    32315104   32315199 -           7         -7.88
## 3 URB1_chr21_323~ URB1  chr21    32315104   32315199 -           8         -8.06
## 4 URB1_chr21_323~ URB1  chr21    32315104   32315199 -           9         -8.15
## 5 URB1_chr21_323~ URB1  chr21    32315104   32315199 -          10         -8.11
## 6 URB1_chr21_323~ URB1  chr21    32315104   32315199 -          11         -7.93
## # ... with 9 more variables: ColaSeqScore <dbl>, MotifScore <dbl>,
## #   CompositeScore <dbl>, NoMotifScore <dbl>, NoPositionScore <dbl>,
## #   NoColaScore <dbl>, start <int>, stop <int>, ColaReadEndCount <int>
```

Ok, now let's plot some examples to gain more intuition on how these log probabilities look.

```
## Make some plots of random example introns with a true branchpoint
set.seed(0)
IntronsToPlot <- PerPosScores.df %>% distinct(region) %>%
  dplyr::select(region) %>%
  sample_n(6) %>% pull(region)

ScoresPlot <- PerPosScores.df %>%
  filter(region %in% IntronsToPlot) %>%
  mutate(
    Label=paste0(gene, ":", strand, ":", chrom, ":", windowStart, "-", windowStop)) %>%
  gather(key="ScoreType",
         value="log2ProbabilityScore",
         MotifScore, PositionScore, ColaSeqScore, CompositeScore) %>%
  mutate(
    ScoreType=recode(ScoreType,
                     MotifScore="Sequence motif",
                     PositionScore="Distance to 3'ss",
                     ColaSeqScore="CoLa-seq 5' ends",
                     CompositeScore="Composite")) %>%
  ggplot(
    aes(x=as.numeric(RelPos), y=as.numeric(log2ProbabilityScore), color=ScoreType)) +
    geom_line() +
    geom_hline(yintercept=Threshold, linetype="dashed") +
    scale_color_manual(
      values = c("Composite" = "#F8766D","Sequence motif" = "#A3A500", "CoLa-seq 5' ends" = "#00BF7D",
    scale_x_reverse() +
    ylab("log2(Rescaled probability)") +
    facet_wrap(~Label, scales="free_x", ncol=3) +
    theme_bw() +
    theme(legend.position="bottom")
ScoresPlot
```



Ok. that looks good. The threshold (dotted-line) I added corresponds to the threshold used in the manuscript for calling BPs. This threshold was chosen based on precision and recall of high confidence branchpoints determined experimentally (see ROC curve in Fig1D in manuscript).

Branchpoints are considered all regions that are above the threshold. Well not quite actually. What I do next (code isn't shown here, it is in the snakemake) is write out the bases that exceed the threshold as a bedfile (each entry is one base), then I merge bed entries within 2 bp of each other, then consider the max scoring base in each merged region as the final set of BPs.

Conclusion.

This method identified 164,532 BPs amongst 109,256 U2 introns and 750 BPs amongst 583 U12 introns from Yi's CoLa-seq data in K562. The quality of BPs are similar to Pineda et al based on the similar total number BPs discovered as Pineda, and a similar overlap rate with other experimental and computationally predicted BPs as Pineda et al.