Midterm 2 - BIFX553

2023-04-19

Name: J. Jedediah Smith

## Exercise 1: Pattern matching

Let there be a DNA sequence (stored in R variable dna). You are asked to find all occurrences of a certain sequence motif (TAGCTG, stored in R variable motif).

dna = "GCAGCTAATCGAGTAGCTAGCTGACCTTAGCGTAGCGTGGATCCCGGTTACCGCGTAGCTGACCTTAGCGTAGCGTAGCTGACCTTAGCGTAGCGTAGCTGACCTTAGC"  
motif = "TAGCTG"

1. Use R code and a text based approach from R package stringr that locates all (that was a hint) occurrences of the motif in the longer sequence. Write the code that prints the result to the document.

library(stringr)  
str\_locate\_all(dna, motif)

## [[1]]  
## start end  
## [1,] 18 23  
## [2,] 56 61  
## [3,] 76 81  
## [4,] 96 101

1. Use the function matchPattern from R/Bioconductor package Biostrings to find all patterns.

library(Biostrings)  
matchPattern(motif,dna)

## Views on a 109-letter BString subject  
## subject: GCAGCTAATCGAGTAGCTAGCTGACCTTAGCGTA...TAGCTGACCTTAGCGTAGCGTAGCTGACCTTAGC  
## views:  
## start end width  
## [1] 18 23 6 [TAGCTG]  
## [2] 56 61 6 [TAGCTG]  
## [3] 76 81 6 [TAGCTG]  
## [4] 96 101 6 [TAGCTG]

1. Use the function matchPDict from the Biostrings package to achieve the same result of all matches.

# Properly define our variables, otherwise PDict gets angry.  
motif\_pd <- PDict(motif)  
dna\_str <- DNAString(dna)  
  
# Find matches with PDict.  
unlist(matchPDict(motif\_pd,dna\_str))

## IRanges object with 4 ranges and 0 metadata columns:  
## start end width  
## <integer> <integer> <integer>  
## [1] 18 23 6  
## [2] 56 61 6  
## [3] 76 81 6  
## [4] 96 101 6

You implemented 3 different ways to find pattern matches in a DNA sequence. Shockingly, depending on the biological question, all 3 implemented approaches are not entirely correct! Why? Because in many situations, DNA sequence fragments such as reads from a high-throughput experiment are derived either the forward or reverse strand of the source DNA. The genomic reference sequences store only the forward strand information - because the reverse strand can be easily derived from the forward strand by computing the reverse-complement. An added challenge is that none of the 3 approaches used above has an easy option to find both forward and reverse matches. Instead we have to explicitly program that. Fortunately, Biostrings has the function reverseComplement with which we can compute the reverse complement of sequences and they launch an additional search with them.

1. Use the function reverseComplement, to compute the reverse complement sequence from out short DNA motif (TAGCTG). Hint: that function may not work directly with R character strings, the motif may have to be converted to a proper data structure before using the function reverseComplement.

# Already made dna a DNAString for question 1c.  
dnarc\_str <- reverseComplement(dna\_str)  
dnarc\_str

## 109-letter DNAString object  
## seq: GCTAAGGTCAGCTACGCTACGCTAAGGTCAGCTACG...GCTACGCTAAGGTCAGCTAGCTACTCGATTAGCTGC

1. Apply functions PDict and matchPDict to find all matches of the reverse complement of that initial motif.

# Already made motif a PDict for question 1c.  
unlist(matchPDict(motif\_pd,dnarc\_str))

## IRanges object with 1 range and 0 metadata columns:  
## start end width  
## <integer> <integer> <integer>  
## [1] 103 108 6

1. Discuss if this needed extra step for finding reverse complements matches is common in bioinformatics (list at least 2 experimental techniques where it occurs) and suggest ways that help you remember that in future projects.

* Reverse complements can be used in two important places:
  + The creation of libraries for Next Generation Sequencing
  + The translation of DNA into proteins
* Perhaps a way to remember this would be to always generate the reverse complement, even when it might not be necessary. That way, one would always be reminded to check whether it is needed.

## Exercise 2: Count Matrices

Many types of high-throughput experiments can be reduced to the analysis of different types of counts of reads matching a certain genomic DNA or RNA transcript regions. When combining this with different experimental conditions, one obtains a matrix where rows correspond to genes or transcripts and columns correspond to the different experiments that have been performed.

1. Discuss under what circumstances the Poisson distribution is a good model for counts.

* For examples or small sets of data that lack large amounts of variation, Poisson works fine. However, it makes a lot of assumptions, such as independence and mean = variance.

1. Discuss under what circumstances the negative binomial distribution is a better model for count data compared to the Poisson distribution.

* With larger more varied sets of data, negative binomial is better. It is similar to Poisson, but there are more settings to play with and is not as constrained by mean and variance assumptions.

## Exercise 3: Differential Gene Expression

A Biotech company was founded on the premise that its founders previously identified 3 different genes whose expression on the RNA and protein level are associated with kidney cancer progression. They also claim that a novel drug treatment will change the expression of these genes.

To verify those claims, RNA-Seq experiments were performed with and without the drug treatment. You obtain results for 4 repeats of RNA-Seq corresponding to the drug treatment, and 3 repeated results corresponding to no treatment (negative control).

Assume you have the following count data matrix, where the first 4 columns corresponding to the drug treatment, and the last 3 columns correspond to the negative control. The 3 genes in question are called here X, Y and Z.

library(edgeR)  
# library(dplyr)  
options(warn=-1)  
countm = matrix(0, nrow=3, ncol=7) # 0 is just placeholder, will be replaced  
rownames(countm) = c("X", "Y", "Z")  
colnames(countm) = c("Treatment1", "Treatment2", "Treatment3", "Treatment4",  
 "Control1", "Control2", "Control3")  
countm["X",] = c(22, 9, 17, 2, 90, 59, 96) # counts for Gene1  
countm["Y",] = c(77, 61, 81, 104, 28, 36, 32) # counts for Gene2  
countm["Z",] = c(56, 61, 59, 67, 58, 57, 64) # counts for Gene3  
knitr::kable(countm)

|  | Treatment1 | Treatment2 | Treatment3 | Treatment4 | Control1 | Control2 | Control3 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| X | 22 | 9 | 17 | 2 | 90 | 59 | 96 |
| Y | 77 | 61 | 81 | 104 | 28 | 36 | 32 |
| Z | 56 | 61 | 59 | 67 | 58 | 57 | 64 |

Counts of reads detected in an RNA-Seq experiment matching its transcript sequences are given the corresponding row in above matrix stored in a variable with name countm.

1. For each gene X, Y, Z, Use a 2-sample t-test comparing the 4 values corresponding to the treatment to the 3 values corresponding to the control. Write code that prints the P-value of the t-test for each row of the matrix.

x\_test <- t.test(countm[1,1:4], countm[1,5:7])  
x\_test$p.value

## [1] 0.01614157

y\_test <- t.test(countm[2,1:4], countm[2,5:7])  
y\_test$p.value

## [1] 0.009407498

z\_test <- t.test(countm[3,1:4], countm[3,5:7])  
z\_test$p.value

## [1] 0.7481943

1. This row-wise t-test is indeed used for same types experiments. But for assessing differential gene expression via RNA-Seq data the 2-sample t-test may not be ideal because some assumptions about its data are not well fulfilled. Discuss why the t-test may be an approximation but may not be ideal for RNA-Seq experiments.

* The biggest problem I see is that t-tests assume individual outcomes are statistically independent. But that couldn’t be further from the truth with something like genetics, where the values of certain genes are often dependent on those of another. It is still a useful tool to quickly gauge the effectiveness of something, but further analysis should be conducted.

1. Using the R package edgeR, use the functions estimateDisp, glmQLFit and glmQLFTest as shown in the prior homework to compute the estimated differential expression of genes X,Y and Z. Use the function topTags to print the results.

library(edgeR)  
  
# Get the count table set up.  
counts = matrix(0, nrow=3, ncol=7)  
rownames(counts) = c("GeneX", "GeneY", "GeneZ")  
colnames(counts) = c("Case1", "Case2", "Case3", "Case4",  
 "Control1", "Control2", "Control3")  
counts["GeneX",] = countm[1,1:7]  
counts["GeneY",] = countm[2,1:7]  
counts["GeneZ",] = countm[3,1:7]  
  
knitr::kable(counts)

|  | Case1 | Case2 | Case3 | Case4 | Control1 | Control2 | Control3 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| GeneX | 22 | 9 | 17 | 2 | 90 | 59 | 96 |
| GeneY | 77 | 61 | 81 | 104 | 28 | 36 | 32 |
| GeneZ | 56 | 61 | 59 | 67 | 58 | 57 | 64 |

# Get the data frame set up.  
df = tibble::tibble(  
 ID=c("Case1","Case2","Case3","Case4","Control1","Control2","Control3"),  
 Group=c("Case","Case","Case","Case","Control","Control","Control")  
)  
  
knitr::kable(df)

| ID | Group |
| --- | --- |
| Case1 | Case |
| Case2 | Case |
| Case3 | Case |
| Case4 | Case |
| Control1 | Control |
| Control2 | Control |
| Control3 | Control |

# Set up factor levels in the right order.  
grouping <- factor(df[["Group"]], levels = c("Control","Case"))  
  
# Get the inputs formatted for edgeR.  
count\_dge <- edgeR::DGEList(counts=counts,group=grouping)  
design <- model.matrix(~ grouping)  
  
# Run the edgeR stuff.  
eset\_dge <- edgeR::estimateDisp(count\_dge, design)  
fit <- edgeR::glmQLFit(eset\_dge, design)  
result <- edgeR::glmQLFTest(fit, coef=2)  
  
knitr::kable(topTags(result))

|  | logFC | logCPM | F | PValue | FDR |
| --- | --- | --- | --- | --- | --- |
| GeneY | 1.5019583 | 18.51625 | 92.820352 | 0.0000001 | 0.0000002 |
| GeneX | -2.4908857 | 18.00043 | 9.397112 | 0.0078517 | 0.0117775 |
| GeneZ | 0.1961857 | 18.51959 | 1.898225 | 0.1884861 | 0.1884861 |

| x |
| --- |
| BH |

| x |
| --- |
| groupingCase |

| x |
| --- |
| glm |

1. What are the estimated fold-changes of the 3 genes? (not the logarithm of the fold changes)

# Fold Change in X, Y, Z order.  
2^result$table$logFC

## [1] 0.177897 2.832269 1.145665

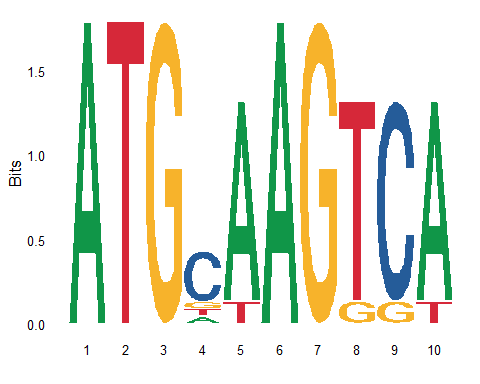
## Exercise 4: Information Theory and Sequence Logos

You obtain evidence about an important DNA motif associated with binding of a novel transcription factor.

logoseqs <- c("ATGCAAGTCA",  
 "ATGGAAGTCT",  
 "ATGCAAGTCA",  
 "ATGTAAGTCA",  
 "ATGCAAGGCA",  
 "ATGCAAGTGA",  
 "ATGAAAGTCA",  
 "ATGCAAGTCA",  
 "ATGCTAGTCA",  
 "ATGCAAGTCA")

1. Use R package ggseqlogo to display the sequence logo of these DNA fragments.

library(ggseqlogo)  
ggseqlogo(logoseqs)



1. Which position is least likely to be important for protein binding?

* Position 4, on account of the fact that it as not as conserved as the other positions.

The R package ggseqlogo is not so helpful when it comes to computing non-graphical numeric values such as information content. Let us try an alternative Bioconductor package called seqLogo. We are using Biostrings function consensusMatrix to count the occurrences of each nucleotide at each position.

# BiocManager::install("seqLogo")  
library(seqLogo)  
dna\_seqs <- DNAStringSet(logoseqs)  
nucleotide\_counts\_raw <- consensusMatrix(dna\_seqs, baseOnly = TRUE)  
nucleotide\_counts=nucleotide\_counts\_raw[rownames(nucleotide\_counts\_raw) != "other",] # not needed here  
nucleotide\_counts

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## A 10 0 0 1 9 10 0 0 0 9  
## C 0 0 0 7 0 0 0 0 9 0  
## G 0 0 10 1 0 0 10 1 1 0  
## T 0 10 0 1 1 0 0 9 0 1

This is the position specific count matrix, that shows how many times a certain nucleotide was observed at a certain position for the provided sequences.

1. Write code that converts the raw counts in variable nucleotide\_counts into probabilities (the probability that a randomly chosen sequence at a certain position is a certain nucleotide). Store the result in variable pwm0 and print.

library(dplyr)  
library(tidyr)  
  
nucleotide\_counts <- as.data.frame(nucleotide\_counts)  
  
pwm0 <- nucleotide\_counts %>%  
 mutate(across(everything(), ~ . / 10))  
pwm0

## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10  
## A 1 0 0 0.1 0.9 1 0 0.0 0.0 0.9  
## C 0 0 0 0.7 0.0 0 0 0.0 0.9 0.0  
## G 0 0 1 0.1 0.0 0 1 0.1 0.1 0.0  
## T 0 1 0 0.1 0.1 0 0 0.9 0.0 0.1

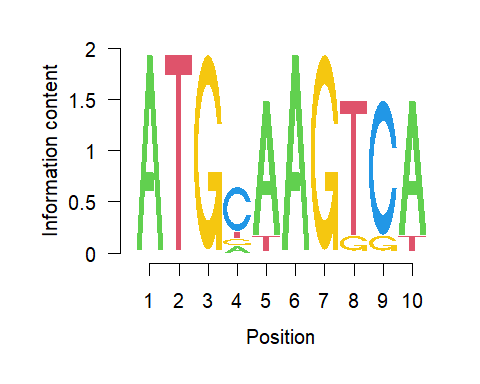
1. Write code that converts the raw counts in variable nucleotide\_counts into probabilities like before but with the difference that a pseudocount of value 1 was used. Store the result in variable pwm1 and print.

pwm1 <- nucleotide\_counts %>%  
 mutate(across(everything(), ~ . +1)) %>%  
 mutate(across(everything(), ~ . / 11))  
pwm1

## V1 V2 V3 V4 V5 V6 V7  
## A 1.00000000 0.09090909 0.09090909 0.1818182 0.90909091 1.00000000 0.09090909  
## C 0.09090909 0.09090909 0.09090909 0.7272727 0.09090909 0.09090909 0.09090909  
## G 0.09090909 0.09090909 1.00000000 0.1818182 0.09090909 0.09090909 1.00000000  
## T 0.09090909 1.00000000 0.09090909 0.1818182 0.18181818 0.09090909 0.09090909  
## V8 V9 V10  
## A 0.09090909 0.09090909 0.90909091  
## C 0.09090909 0.90909091 0.09090909  
## G 0.18181818 0.18181818 0.09090909  
## T 0.90909091 0.09090909 0.18181818

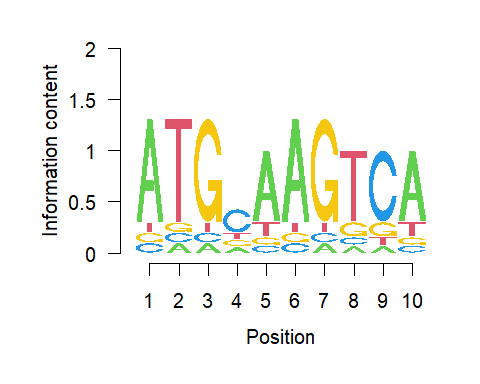
1. Display the sequence logo of the first case without pseudo-count (variable pwm0) using the function seqLogo from the R package seqLogo. Hint: before called the function seqLogo, our matrix of probabilities per position (a position-specific weight matrix of PWM) has to be converted into a format specific to the seqLogo package using its function seqLogo::makePWM(...)

pwm0 <- makePWM(pwm0, alphabet = "DNA")  
seqLogo(pwm0)



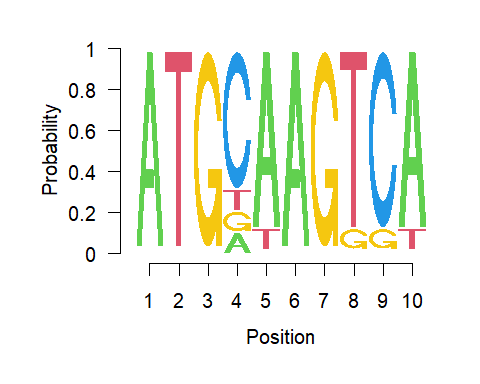
1. Display the sequence logo of the second case with pseudo-count (variable pwm1) using the function seqLogo from the R package seqLogo.

pwm1 <- makePWM(pwm1, alphabet = "DNA")  
seqLogo(pwm1)



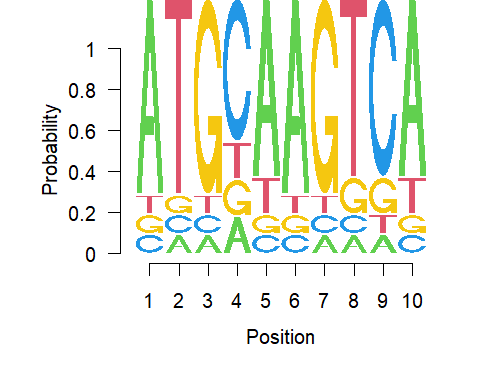
1. Using the option ic.scale=FALSE, plot the sequence logo of pwm0 as probability instead of information content:

seqLogo(pwm0, ic.scale=FALSE)



1. Using the option ic.scale=FALSE, plot the sequence logo of pwm1 as probability instead of information content:

seqLogo(pwm1, ic.scale=FALSE)



1. Using the function seqLogo::ic() one can compute the information content of each position of the position specific weight matrix. Compute this vector of numbers using the ic function applied to the pwm0b variable. Compute and print the sum of these information content values.

pwm0b <- ic(pwm0)  
pwm0b

## [1] 2.0000000 2.0000000 2.0000000 0.6432204 1.5310044 2.0000000 2.0000000  
## [8] 1.5310044 1.5310044 1.5310044

1. Using the function seqLogo::ic() one can compute the information content of each position of the position specific weight matrix. Compute this vector of numbers using the ic function applied to the pwm1b variable. Compute and print the sum of these information content values.

pwm1b <- ic(pwm1)  
pwm1b

## [1] 1.0565186 1.0565186 1.0565186 0.3243598 0.7988398 1.0565186 1.0565186  
## [8] 0.7988398 0.7988398 0.7988398

1. If IC is the total information content of a sequence logo, it is expected to occur every positions by chance. Compute the probabilities that our sequence motif occurs by chance at a certain position using variable pwm0b as well as pwm1b.

1/(2^pwm0b)

## [1] 0.2500000 0.2500000 0.2500000 0.6402821 0.3460364 0.2500000 0.2500000  
## [8] 0.3460364 0.3460364 0.3460364

1/(2^pwm1b)

## [1] 0.4807909 0.4807909 0.4807909 0.7986527 0.5748112 0.4807909 0.4807909  
## [8] 0.5748112 0.5748112 0.5748112

1. Discuss the differences in sequence logo and total information content between our 2 different versions - with and without pseudo-counts as reflected in variables pwm0 and pwm1. Which version is more “conservative” in the sense that it makes less strong claims about an identified logo?

* The more conservative option is pwm1 because it claims to have a higher chance of getting the indicated results by coincidence. It is attributing less confidence to the results.

1. Discuss the advantages and disadvantages of displaying the sequence logo in terms of information content or alternatively in terms of probability.

* Using probability makes it easier for the average person to wrap their head around and thus improves the readability of a given visualization. However, information content is more technically correct, and may be better suited for a knowledgeable audience, or when the output will be further processed.