Comparison of Computational Efficiency in RNA Sequencing Analysis Normalization Algorithms

INFO-B536

Pradeep Varathan

Racheal Benoy

December 14th,2020

Summary

RNA Sequencing Analysis uses next generation sequencing to examine the quantity and sequences of RNA in a sample.It has many application such as transcriptional profiling, SNP identification, RNA editing and differential gene expression analysis.Our concentration will be on the normalization process in the gene expression testing.This project will explore the various algorithms and packages that are available to do this analysis.The results of this project will allow bioinformaticians in the future to pick the optimal tools based on their dataset size , availability of these tools and proficiency in coding.

Introduction

The goal of this project is to compare various normalization techniques that are used today to find the optimal tool for each need. We will be comparing 4 different normalization algorithms to each other based on qualitative and quantitative parameters such as box plots and memory time analysis. The datasets we have chosen for this project are all human so that we can keep the genetic size considerably close to each other. We gathered multiple datasets to check the true efficiency of these algorithms and how accurately and fast they run when given datasets of various sample sizes.There are 3 datasets that we will be using, the first one contains gene expression biomarker candidates for neuro behavioral impairment from total sleep deprivation , it has 555 samples from the Gene Expression Omnibus(GEO) . The second dataset also from GEO contains the expression data from thalamus or parietal lobe of FFI patients and normal humans, this has 8 samples. The final dataset is our own RNA sequencing dataset created through simulation using the package PROPER in R. The 4 different algorithms we will be using are DESeq2,edgeR(TMM) and FPKM ; we will be utilizing these as packages in R . The results will tell us the efficiency of each algorithm tool based on computational running time ,memory usage and box plots will be utilized to display the ranking of these tools for each parameter.

Algorithms

For this project there are 4 main normalization algorithms that we will be utilizing Poisson Seq, DESeq2,TMM(edgeR) and RPKM.

Poisson PseudoCode :

Input: Count Matrix with n rows (genes/transcripts) and m columns (subjects),
A matrix of lengths in base pair,number of matched genome transcripts of all n transcripts
Output : Normalized Count Matrix with n rows and m columns

```
size = list of size m for each subject
multiplication_factor = list of size n for each transcript
temp_matrix = expression matrix of dimension of m*n
FOR subject_col in columns of temp_matrix:
    multiplication_factor = multiplication_factor*subject_col
normalization_list = empty list ()
Function sequence_depth(n,iter,ct_sum,ct_mean){
INPUT : n = temp_matrix
      tier = number of iterations used = 5
      ct_sum = total number of reads of a gene across all experiments = 5
      ct_mean = mean number of reads of a gene across all experiments = .5
         seq_depth = mean of n
         seq_depth = exp(log(seq_depth) - mean(log(seq_depth))
 RETURN seq depth
 normalization_list = sequence_depth(temp_matrix,5,5,0.5)
 FOR subject_col in columns of temp_matrix :
   temp_matrix[,subject_col] = temp[,subject_col]/normalization_list[subject_col]
 RETURN temp_matrix
```

Poisson Seq's process of normalization is implemented in the wolfram language , so that the sum of probabilities equals 1. It would modify the observed library size to an effective size which adjusts the modeled mean. The way the pseudo code was written , will have an input of a count matrix with gene n rows and subject m columns and utilizes a function that calculates sequence depth which will in turn be used to calculate the normalized count matrix by dividing the each subject with the sequence depth.

TMM Pseudocode:

```
(Trimmed Mean of M-values)
Input: Count Matrix with n rows (genes/transcripts) and m columns (subjects),A matrix of
lengths in base pair,number of matched genome transcripts of all n transcripts
Output : Normalized Count Matrix with n rows and m columns

Count_Matrix = expression matrix of dimensions n*m
mult_factor = list of size m for each subject
Temporary_Count_Matrix = Count_Matrix
FOR transcript in rows of Temporary_Count_Matrix:
        FOR subject in columns of Temporary_Count_Matrix:
                Temporary_Count_Matrix[transcript,subject] = ((log2(subject/mult_factor))
RETURN Temporary_Count_Matrix
```

TMM (trimmed mean of M values) used by edgeR also follows the similar hypothesis testing as DESeq, but instead of using the mean variance to weigh the model, it uses a maximum weighted likelihood for estimating the gene-wise dispersion parameters. TMM normalization factors (produced by calcNormFactors Function) do not take into account library sizes or gene length but only the relative scaling factors and it compares the amount of times the gene has been counted within a sample. It repeats the same process to all samples and normalizes with a common factor for each gene inside the sample. In edgeR, these normalization factors are used as offset parameters in the statistical model for differential gene expression analysis. TMM is a good choice in algorithms when you want to remove the batch effects while comparing the samples where the RNA population is different among the samples.

RPKM Pseudocode:

(Reads per kilobase per million mapped reads)
Input: Count Matrix with n rows (genes/transcripts) and m columns (subjects),
A matrix of lengths in base pair,number of matched genome transcripts of all n transcripts
Output : Normalized Count Matrix with n rows and m columns

```
Count_Matrix = expression matrix of dimensions n*m
length_matrix = list of size n*2 for each transcript length
total_length = SUM(all lengths in length_list)
Temporary_Count_Matrix = Count_Matrix
FOR transcript in rows of Temporary_Count_Matrix:
        FOR subject in columns of Temporary_Count_Matrix:
                Temporary_Count_Matrix[transcript,subject] = ((number of matched genome
        transcripts for transcript from length_matrix)*(10^3)*(10*6))/ ((total_length)*(length of
        gene in base pair for transcript from length_matrix))
RETURN Temporary_Count_Matrix
```

RPKM normalization method, when expanded, the reads per kilobase of exon per million reads mapped normalization method compares the amount of times the gene has been counted within a sample. It repeats the same process to all samples and normalizes with a common factor for each gene inside the sample. Due to its brute force algorithm that goes through each gene in each sample at a time, as described in the algorithm, it takes much less time for smaller datasets but when the amount of gene replicates increases, it makes the algorithm run much longer.

DESeq2 PseudoCode:

Input: Count Matrix with n rows (genes/transcripts) and m columns (subjects)
Output : Normalized Count Matrix with n rows and m columns

```
Count_Matrix = expression matrix of dimensions n*m
size_factors = list of size m for each subject
multiplicative_factor = list of size n for each transcript
FOR subject_column in columns of Count_Matrix:
        multiplicative_factor = multiplicative_factor*subject_column
multiplicative_factor = multiplicative_factor^(1/m)
Temporary_Count_Matrix = Count_Matrix/multiplicative_factor4
median_list = empty list of m values
FOR subject_column in columns of Temporary_Count_Matrix:
        median_list = c(median_list, MEDIAN(Temporary_Count_Matrix[,subject_column]))
        Temporary_Count_Matrix[,subject_column] =
Temporary_Count_Matrix[,subject_column]/median_list[subject_column]
RETURN Temporary_Count_Matrix
```

DESeq2 on the other hand, has a normalization method that uses the median of the normalization factors which is defined by the whole experiment, including all the samples for the gene counts taken all together and thus, has a more relative normalization factor than most other methods. This method resembles brute force in a way but since it takes all the samples together, the difference in complexity when the number of gene replicates increase is minimal, as can be seen in pseudo-code drafted above. The steps include calculating a pseudo-reference sample that includes taking the geometric mean of all samples of a particular gene read and then calculating the ratio factor for each sample among the references to provide a normalization factor. The median of all such normalization factors is taken into account for normalizing the whole matrix.

Datasets

The data that will be utilized for this project will be a mixture of simulated data and data from the GEO. They are all human samples.

1. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE40562 (Accession : GSE40562) This dataset contains 8 samples with expression data from thalamus or parietal lobe of FFI patients and normal human

2. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE98582 (Accession : GSE98582) This dataset contains 555 samples with Exploring gene expression biomarker candidates for neurobehavioral impairment from total sleep deprivation
3. Since GEO datasets are not available with the desired amount of genes and different counts and the amount of subjects, we decided to simulate our own RNA sequencing dataset for the following gene sample sizes and subjects, using the package PROPER in R, which generates a count data matrix designed based on negative binomial distribution for a two-group comparison design and sample sizes with replicates in each condition, then generate a matrix of counts.

DataSet	Number of Genes	Subjects	Controls
Data1	10000	4	4
Data2	20000	4	4
Data3	50000	4	4
Data4	10000	50	10
Data5	20000	50	10
Data6	50000	50	10
Data7	10000	450	100
Data8	20000	450	100
Data9	50000	450	100

Experimental Designs

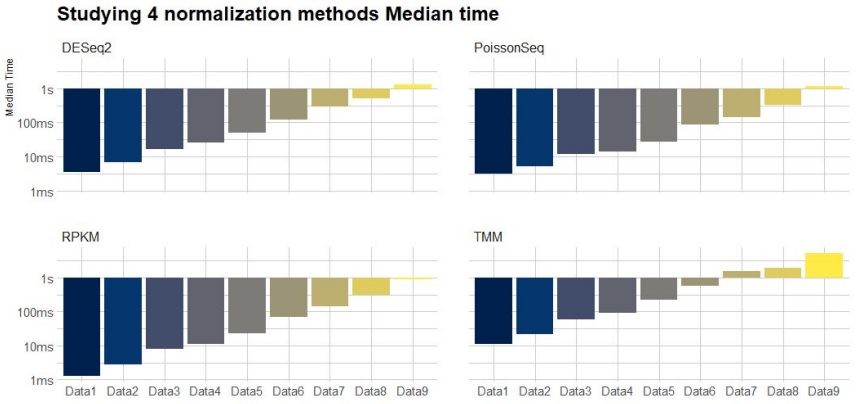
As for the design, we built the whole process in R using multiple packages such as GEOquery, by which we downloaded the GEO datasets from NCBI, NOISeq,DESeq2 and PoissonSeq packages which were utilized to call the normalization functions such as TMM,RPKM while DESeq2 and PoissonSeq normalization functions had to be coded separately

using the pseudo-code as reference since there wasn't a direct normalization command in these packages. The ggplot2, RColorBrewer and ggrepel were used for visualization of the results to understand the significance of each normalization method in both time and memory. The data management packages such as tidyverse and dplyr were used as dependencies in the normalization methods defined as functions and also in the simulation functions. The PROPER package, as suggested above, was used in the simulation process while the bench package was used to run multiple runs of the process to simulate median time and memory allocated for each of the methods in multiple datasets as described above. The code for the process is attached in the appendix and can also be found in the github repository Comparing Computation Efficiency RNA Seq Normalization Methods

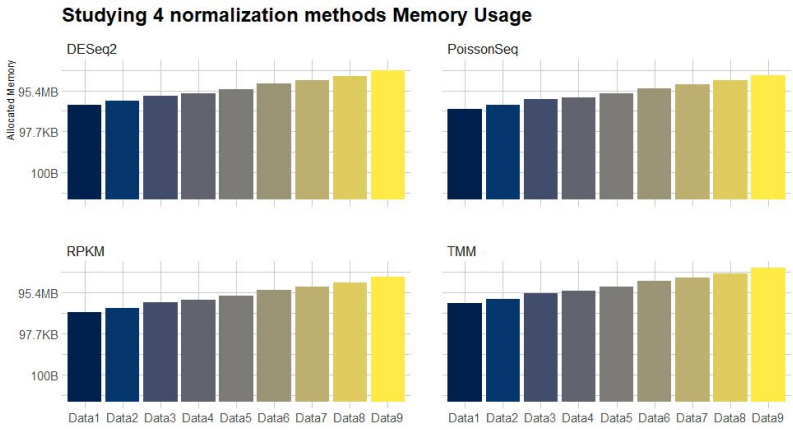
We first use the simulated datasets and use the benchmark pipeline, which allows us to calculate the memory allocated and time takes for each normalization function under each dataset with multiple iterations to get the median time taken and memory allocated for each dataset. This allows us to compare the method individually while also using ggplot2 visualization, we can clearly see in the results below how each dataset influences these variables. Furthermore, we used the GSE40562 and GSE98582 datasets for the same purpose. First using the GEOQuery, we were able to obtain the dataset directly from the NCBI as a raw datasource. We then converted it into the expressionset objects in R for easy analysis using the methods. Then, using the bench functions, we iterated each method in each dataset for 50 times and calculated the time and memory median and allocated for every iteration and tabulated them for further analysis. These results can be seen below and a detailed results file is present in the github repository.

Experimental Results

First after running the normalization functions for the simulated datasets, we found that for running small datasets, the trend is similar among all the normalization methods but as the dataset's count matrix increases in size, that is, increase in both the number of subjects and the number of genes per subject, we see that DESeq2 and PoissonSeq normalization methods fare good with RPKM being the fastest but due to the higher brute force method, TMM does not efficiently work in these stages. In terms of memory allocation, we see that TMM and DESeq2

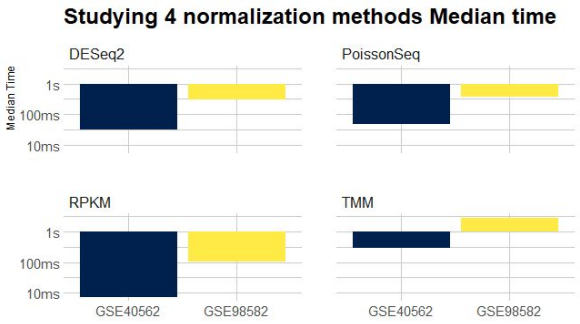


consume a large amount of memory in comparison to RPKM method and PoissonSeq method of normalization, while TMM can be considered as most computationally intensive since it consumes more memory and time while not providing the best normalization when the dataset is smaller. From the simulation

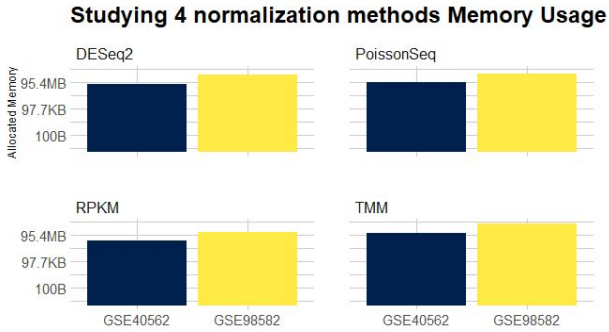


studies, we can see that the PoissonSeq normalization method is the best to use when the dataset has a large number of subjects and genes to limit the memory usage.

While running the non-simulated data that we obtained from NCBI GEO datasets, GSE40562 and GSE98582, with 8 subjects and 555 subjects respectively, we were able to plot their statistics with similar methodology as described above.



Similarly in memory allocation, we can see that DESeq2 and TMM utilize a lot more memory in their normalization method. The RPKM method though has used the minimal memory among all, its normalization when visualized in terms of bar plots was not completely normalized compared to PoissonSeq, which utilizes the second least memory for the process to compile.



The figures below display the memory time analysis for each normalization method against the dataset used :

GSE40562 (8 samples)	Median time	Memory_allocation
DESeq2	48.34ms	54.4MB
PoissonSeq	72.36ms	96.8MB
TMM	324.3ms	174.9MB
RPKM	6.04ms	20MB

GSE98582 (555 samples)	Median time	Memory_allocation
DESeq2	706.85ms	586.73MB
PoissonSeq	949.37ms	824.9MB
TMM	4610ms	2.04GB
RPKM	798.88ms	182.93MB

Findings

From our results , we can conclude that that TMM is the least efficient normalization method since it not only utilizes the most memory as sample size increases but also takes the most time to complete the process. DESeq2 also utilizes more memory than RPKM and Poisson but required less time and fared better than TMM, the time does however increase based on the size of the datacount matrix as there is an increase in the number of subjects and number of genes per subjects.Poisson Seq's performance was very neutral as it didn't have the best performance time wise or memory wise , it's time performance was better than TMM but was slower than the rest of the algorithms, its memory usage was far less than TMM but almost identical to RPKM. Poisson Seq does however have more normalized data in comparison to RPKM which not only had the best performance in time which can be attributed to the brute force algorithm it utilizes. Though it is less memory intensive than all of the other algorithms, it isn't as accurate with its normalization as is evident by the box plot.

Conclusion

We can conclude that from the algorithms we hoped to compare that RPKM and Poisson Seq fare better than the other two algorithms. While RPKM and Poisson seq both require less memory and time. Poisson Seq has a better normalization performance than RPKM. We did stay on target with the project and accomplished what we wanted, we succeeded in showing which algorithm performed better based on the metrics we decided from the beginning .If we had more time we would have liked to use a larger dataset and less simulation data, we also would like have furthered the whole RNA sequencing process and see if there was a noticeable difference in the final results based on these normalization techniques.

References

1.Tian, C. (2015, August 30). *Expression data from thalamus or parietal lobe of FFI patients and normal humans.* Gene Expression Omnibus (GEO) Accession Viewer. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE40562

2.Uyheljl, H. A. (2018, April 10). *Exploring gene expression biomarker candidates for neurobehavioral impairment from total sleep deprivation*. Gene Expression Omnibus (GEO) Accession Viewer. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE98582

3.Maza, E. (2016). *In Papyro Comparison of TMM (edgeR), RLE (DESeq2), and MRN Normalization Methods for a Simple Two-Conditions-Without-Replicates RNA-Seq Experimental Design*. Frontiers. https://www.frontiersin.org/articles/10.3389/fgene.2016.00164/full

4.Li, J. (2019, May 1). *estimate the sequencing depths*. Rddr.Io. https://rdrr.io/cran/PoissonSeq/man/PS.Est.Depth.html

5.Assefa, A. T. (2018, July 24). *Differential gene expression analysis tools exhibit substandard performance for long non-coding RNA-sequencing data*. Genome Biology. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-018-1466-5

6.Soneson, C. (2013, March 9). *A comparison of methods for differential expression analysis of RNA-seq data*. BMC Bioinformatics. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-91

7. Assefa, A. T. (2018, July 24). *Differential gene expression analysis tools exhibit substandard performance for long non-coding RNA-sequencing data*. Genome Biology. https://genomebiology.biomedcentral.com/articles/10.1186/s13059-018-1466-5

8.Robinson MD, McCarthy DJ, Smyth GK (2010). *edgeR: a Bioconductor package for differential expression analysis of digital gene expression data.* Bioinformatics, **26**(1), 139-140. doi: 10.1093/bioinformatics/btp616.

9.Bioconductor. (n.d.). *GEOquery*. Retrieved December 1, 2020, from https://www.bioconductor.org/packages//2.10/bioc/html/GEOquery.html

10.Bioconductor. (n.d.). *GEOquery*. Retrieved December 1, 2020, from https://bioconductor.org/packages/release/bioc/html/NOISeq.html

11.Bioconductor. (n.d.). *GEOquery*. Retrieved December 1, 2020, from http://bioconductor.org/packages/release/bioc/html/DESeq2.html

12.Li, J. (2019b, May 1). *PoissonSeq: Significance analysis of sequencing data based on a Poisson log linear model*. Rddr.Io. https://rdrr.io/cran/PoissonSeq/

13.Comprehensive R Archive Network (CRAN). (2020, June 19). *CRAN - Package ggplot2*. R-Project. https://cran.r-project.org/web/packages/ggplot2/index.html

14.Neuwirth, E. (2014, December 7). *CRAN - Package RColorBrewer*. CRAN R-Project. https://cran.r-project.org/web/packages/RColorBrewer/index.html

15.*ggrepel package | R Documentation*. (n.d.). R-Documentation. https://www.rdocumentation.org/packages/ggrepel/versions/0.8.2

16.*clusterProfiler*. (2018). Bioconductor. https://bioconductor.org/packages/release/bioc/html/clusterProfiler.html

17. DOSE.(2018).Bioconductor https://www.bioconductor.org/packages/release/bioc/html/DOSE.html

18.Tidyverse (2020,October 27).Bioconductor https://bioconductor.org/packages/release/bioc/vignettes/destiny/inst/doc/tidyverse.pdf

19.Comprehensive R Archive Network (CRAN). (2020b, August 18). *CRAN - Package dplyr*. https://cran.r-project.org/web/packages/dplyr/index.html

20. *PROPER*. (2018). Bioconductor. https://bioconductor.org/packages/release/bioc/html/PROPER.html

21.Hester, J. (2020, January 13). *CRAN - Package bench*. Comprehensive R Archive Network (CRAN). https://cran.r-project.org/web/packages/bench/index.html

22.*Poisson Distribution -- from Wolfram MathWorld*. (n.d.). Mathworld. https://mathworld.wolfram.com/PoissonDistribution.html

23.Bedre, R. (2017, October 5). *Gene expression units explained: RPM, RPKM, FPKM, TPM, DESeq, TMM, SCnorm, GeTMM, and ComBat-Seq*. Renesh Bedre. https://reneshbedre.github.io/blog/expression_units.html