*Bioinformatics*, 2017, 12–12

doi: 10.1093/bioinformatics/xxxxx

|  |
| --- |
| CSE 549.01 Computational Biology  Adding Salmon-based Modules to MultiQC  Yash Shah1, Sharad Gupta1, Siddhartha Chhabra1, Dinesh Balani1  1Department of Computer Science, Stony Brook University, Stony Brook, 11794, United States  \*To whom correspondence should be addressed.  Received on 2017/12/12; revised on 2017/12/12  Abstract  **Motivation:** The motivation for this project is to provide support for Salmon modules on MultiQC tool.  **Results:** Plotting line graphs for GC bias and Sequence bias, Heatmap to show correlation between these biases.  **Availability:** Github URL: https://github.com/yash199/MultiQC-Salmon  **Contact:** ysshah@cs.stonybrook.edu |

# Introduction

MultiQC searches a directory for analysis logs and compiles a HTML report. It's a general use tool, perfect for summarizing the output from numerous bioinformatics tools. It Aggregate results from bioinformatics analyses across many samples into a single report. The objective of this project is to integrate Salmon, a tool for wicked-fast transcript quantification from RNA-sequence data which enables us to better quantify and visualize the Salmon outputs. In this project, we have developed the following modules.

1. Representation of the GC-bias model
2. Representation of the Sequencing-bias model (3’ and 5’)
3. Heatmap denoting correlation between average GC-bias and Sequencing-bias.

These representations help us in assessing the analysis results across all samples.

# Methods

2.1 GC Bias

In this method, we aim to represent information about GC bias (the percentage of G and C nucleotides) in the fragments, compared to expected values. The module parses the logs and checks if sample has  ‘gc\_bias\_correct’  flag set true in meta\_info.json file and plots the corresponding graph in MultiQC report.

* Get the sample path from command line argument.
* For each subdirectory in the path location, program will navigate through “/bias/aux\_info/” and read “gc\_bias\_correct” property in “meta\_info.json” file
* If this flag is set to true, program parses  ‘observed\_bias.gz’  and ‘expected\_bias.gz’  using parser helper function provided in GCModel.py to get observed and expected gc-bias values and corresponding weights.
* We now calculate weighted sum of 3 observed weights and weighted sum of 3 expected weights for each of low, medium and high bias type.
* We have taken ratio of the corresponding weighted values for each sample.
* We have taken the average of all 3 bias type of a sample. The result obtained for all bias types and the average is plotted on line graphs corresponding to each index value. X- axis has been scaled up to 100.

We can infer that the length of the coding sequence is directly proportional to higher G+C content.

2.2 Sequencing Bias

In this method, we read following 4 input.gz files namely obs3\_seq.gz, obs5\_seq.gz, exp3\_seq.gz, exp5\_seq.gz respectively. each corresponding to 3 prime and 5 prime observed and expected data.

We give 4 lists (one for each nucleotide) for each of observed 3’, observed 5’, expected 3’, expected 5’ as input

The input files are parsed to get 2-D matrix of size 4 X context length. Each of the row in this matrix contain probability of each nucleotide at each position in the context known as marginalized probabilities.

For each nucleotide (A, C, T, G) we calculate the ration of observed and expected marginal probability for both 3’ and 5’ respectively.

To show the result we plotted 2-line graphs – 3’ and 5’ for each nucleotide showing comparison between observed and expected ration for each of the nucleotides

2.3 Heatmap

In this method, we aim to show the correlation of the average GC and sequencing bias between the samples by plotting heatmap. We will be creating 3 heat maps (GC, 5', 3') of size nxn, where n is the number of total samples.

Basically, we want to show a graphical representation of data where

the individual values contained in a matrix are represented as colors. This can be achieved using a heatmap. We represent the degree of similarity by comparing GC bias and Sequence bias (3’ and 5’) between any two samples. If the score is close to zero (blue color area in heatmap), then two samples are similar. In cases of dissimilar samples, the score will be higher (represented by red color in heatmap).

To determine similarity between any two samples, we used Jensen Shannon Divergence as a metric.

[Note:The Jensen–Shannon divergence is a method of measuring the similarity between two probability distributions.]

**< Modify this: explain that we actually see what is present in 10 samples**

**In case, if any feature is missing in any sample, we do not consider it**

**for comparison. We just output it in missing feature table as in Figure 11.**

**command for running tool >**

# Results

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

## Data Structure This is Heading 2 style this is heading 2 style

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

### 3.1.1 This is heading 3 style

The quick brown fox jumps over the lazy dog. Unnumbered list style

The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

 (1)

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

**Fig. 1. Relation between τ and *t*.** This example has only two continuous Steppers, S1 and S2.

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

The quick brown fox jumps over the lazy dog.

**Table 1.**Benchmark results of the cascade oscillators model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |S| | Predicted cost | Timing | Predicted speed | Speed |
| 1 | S219.20(100%) | 68m43s | 1.00 | 1.00 |
| 2 | 29.10+219.10(~50%) | 35m13s | 2.00 | 1.95 |
| 4 | 219.20(100%) | 68m43s | 1.00 | 1.00 |
| 10 | 29.10+219.10(~50%) | 35m13s | 2.00 | 1.95 |
| 20 | 219.20(100%) | 68m43s | 1.00 | 9.5 |

This is table foot note sample text This is table foot note sample text This is table foot note sample text

The quick brown fox jumps over the lazy dog.

Acknowledgements

We would like to thank Professor Robert Patro (Department of Computer Science), Fatemeh Almodaresi and Laraib Malik for their continuous support and guidance.

References

Alexandrescu,A. (2001) Modern C++ Design: Generic Programming and Design Patterens Applied. Addision Wesley Professional, Boston.

Dormand,J.R. and Prince,P.J. (1980) A family of embedded Runge–Kutta formulae. *J. Comp. Appl. Math.*, **6**, 19–26.

Alexandrescu,A. (2001) *Modern C++ Design: Generic Programming and Design Patterens Applied.* Addision Wesley Professional, Boston.

Dormand,J.R. and Prince,P.J. (1980) A family of embedded Runge–Kutta formulae. *J. Comp. Appl. Math.*, **6**, 19–26.