UI for RNA-seq quantification *

Yigong Wang
SBU ID: 109973706
Department of Computer Science
Stony Brook University
yigwang@cs.stonybrook.edu

Hongjin Zhu
SBU ID: 109748818
Department of Computer Science
Stony Brook University
hozzhu@cs.stonybrook.edu

Jianglin Wu
SBU ID: 108075432
Department of Computer Science
Stony Brook University
jiangwu@cs.stonybrook.edu

Jian Yang
SBU ID: 110168771
Department of Computer Science
Stony Brook University
yang16@cs.stonybrook.edu

Abstract

RNA sequencing, is a technology that uses the capabilities of next-generation sequencing to reveal a snapshot of RNA presence and quantity from a genome at a given moment in time. There are many modern algorithms for RNA-seq but there is few visualization about the algorithms' results. Based on this status, we implemented a UI for RNA-Seq quality comparison between various of algorithms. The UI could upload results from algorithms, generate plots and tables for quality metrics and a user could interactively view any data on the plots.

1 Introduction

Software developed by programmer can process high throughput RNA-seq data and create detail results for Biologist and Bioinformatician for further analyzing. However, these tools cannot replace the role of a professional user who knows how to deal with the data and the meaning behind the number and text. A simple graph contains more information than paragraphs of text description. So, to provide an easy way for Biologist to research on the output data, we have our User Interface which consists of following functionalists: Bar chart for basic statistical information about the data; Scatter Plots for algorithm output data visualization, comparison and analysis; and Metrics for performance evaluation.
To implement such an application, we used Python as the backend, we build a web service based on Tool Django and using JavaScript at frontend and using Python at the backend.
As an application, a user could view a bar chart to compare the statistics among all the statistics and also could choose any two attributes for a scatter plot. Any also a node's information could be showed as interaction. At last, a performance evaluation could be listed with the ground truth file.

2 UI Architecture

As a web application, this application includes the following parts as:

1. Webpage displayed at user's browser.
2. Server to accept request from a user and send the webpage and contents to user.

*Course Project Report of CSE 549.

3. Request Handler to handle the request and query the related data from Data Layer. The result would also be packed to JSON format and send back to Server side.
4. Data Layer to load and process data.

The Architecture is showed in figure 1.

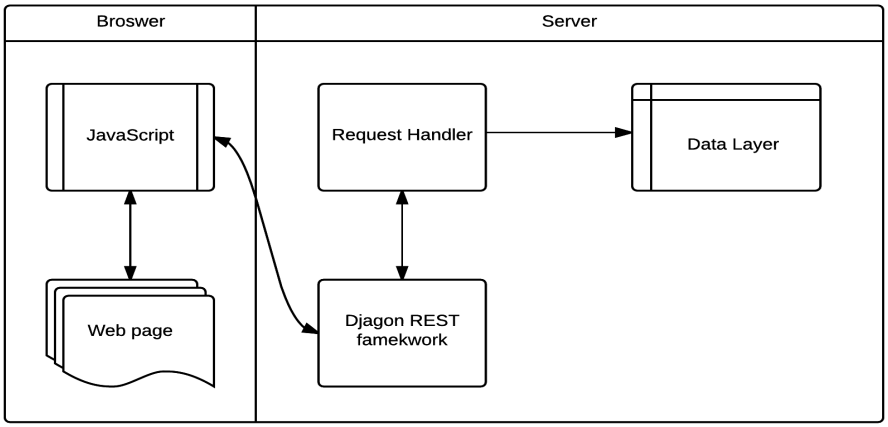


Figure 1: Architecture

As one example, here is one example story when a user wants to view the scatter plot for algorithm *Sailfish* with the attributes GC content versus abundant estimate. The internal steps are:

1. User clicks the page for single algorithm plot.
2. A request for an algorithm list would be sent by JavaScript on webpage to Django.
3. Django passes the request to a Request Handler, and the handler would get the algorithm list and packaged the list o JSON package and send the package to Django.
4. Django passes the result to JavaScript, and JavaScript lists the algorithms in a drop down list.
5. User chooses one algorithm to display. Let's say the user clicks sailfish.
6. Same steps as 1. to 4., a request for all the attributes of algorithm sailfish is sent to backend and finally the list would be displayed on webpage.
7. User chooses two attributes from the list on webpage.
8. Same steps as 1. to 4. a request for the two attribute data would be sent and the data is sent back finally.
9. JavaScript plots the data on webpage.
10. When the user moves the mouse to any point on the plot and the information would be displayed on webpage by JavaScript.

Here is the visualized flow for this example in figure 2.

3 UI Components

3.1 Data Layer

At data layer scripts, we did the following jobs.

1. Calculate the statistics from the transcript.
   We calculate the data such as GC content, ration of each nucleotides.
2. Load data from result files for algorithms and also the ground truth from profile files according to given parser.

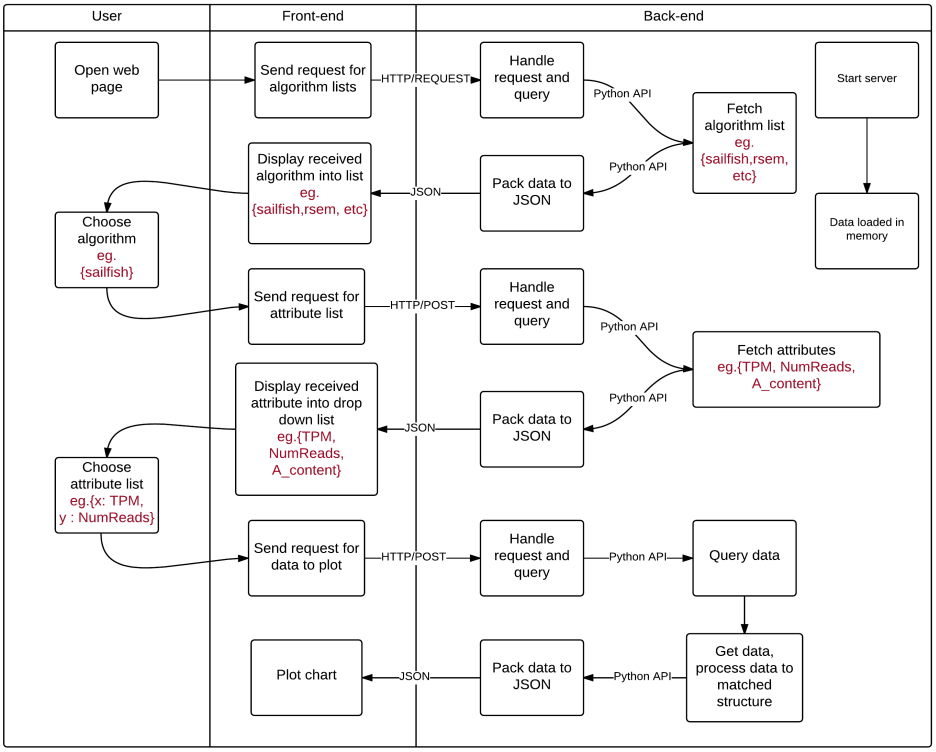


Figure 2: Example of one request

3. Concat the data of results, the ground truth and the statistics based on transcript name as index to a large table.
4. Provide the API to query the algorithm list, the attributes and the data for UI layers.
5. Calculate the performance evaluation matrix.
6. Sample data if the data has too many items.
   If there are too many data, for example, in the given data there are more than 80 thousands of points to be plotted at frontend. To reduce the size of the data and keep all the outliers, I used the following algorithm:
   (a) Calculate the centroid point of the data.
   (b) Calculate the distances for all points to center point and pick a distance to include 99% of the poinrts.
   (c) For any points which have longer distance than the picked 99% one, we would keep them.
   (d) For all points which have smaller distance than the picked 99% boundary, we would sample them by a ratio. By using this sample, we could get 0.5% to 1% size of the original data size.

3.2 Request Handler

When the project runs, a simple server script listen to the specified localhost port. The Django preprocess all received requests of that port. In *server/server/urls.py*, we defined the url scheme. Our scripts handling all requests according to this URL scheme, then wrap results to desired format and send to user browsers.
Because we have different types of interactive charts, bar charts of algorithm comparison and a performance evaluation table, the URL scheme assigned three corresponding URLs to them. Every URL has scripts to process requests to it.
When user access the root URL, the server load all data to the memory and calculate all algorithm compari-

son charts. Instead of accessing file system, later data access will go to the memory directly, which is more faster and flexible. After that, the page redirect user to the main website page.
Here are the request types and how we would handle them.

1. For requests of the interactive charts, the URL is */get*. Because we have three types of chart plots. scripts of processing these requests defined three request types. Corresponding to single algorithm scatter chart, two algorithms scatter chart and two algorithms comparison chart.
   (a) Firstly, JavaScripts in the client request algorithm types, server scripts give algorithm names and their attributes to the client. According to different picture types, the script gives different attributes. For example, in two algorithms comparison chart, we only give common attributes which make sense for comparison.
   (b) Later on, this URL also response to the client requests for get attributes data, scripts return required data in JSON format to the client.
2. For requests of the bar charts, the URL is */get_common*. This URL provides all algorithm shared attributes in JSON format to the client. The client can use a predefined image URL scheme to load calculated images. For example, if an attribute is ?TPM?, predefined URL might be *static/img/TPM_bar.png*
3. For requests of the performance evaluation table, the URL is */matrix*. It provides the algorithms? performance data to the client.

3.3 Frontend

We used and modified open source bootstrap template SB Admin 2 [1] as our webpage, then integrated and implemented following:

1. Javascript public library:
   - Jquery: jquery-1.8.3.min.js
   - D3js: d3.v3.min.js [2]
2. Javascript library which we implemented:
   - chart.js: plot all the scatter charts.
   - data.js: using Ajax call to fetch data from server.
   - dropDownList.js: using Ajax call to fetch algorithm and attribute information from server. Change drop down list dynamically based on user selection.

Design details:

1. Dynamic drop list.
   (a) Using Ajax call to fetch JSON data from user, which contains all feasible algorithms and their relevant attribute.
   (b) Listening to user selection of algorithm, and dynamic change the attributes drop down list.
   (c) Get user's selections from drop down list and send request to server. Then get data to plot scatter charts
2. There are three type of scatter charts.
   (a) Scatter Plot for a Single Algorithm.
   Get user input from front end, send requests through Ajax call to get matched data from server. Get each node and plot them on svg using D3js.
   Requirement input value from user: algorithm name, x axis, y axis
   (b) Scatter Plot for two Algorithms on the same metrics.
   Get user input from front end, send requests through Ajax call to get matched data from server. Get each node and plot them on svg using D3js.
   Requirement input value from user: algorithm 1, algorithm 2, attribute
   (c) Scatter Plot for two Algorithms on the same feature.
   Get user input from front end, send requests through Ajax call to get matched data from server. Get each node and plot them on svg using D3js.

Requirement input value from user: algorithm 1, algorithm 2, attribute

3. interactive with user.
   When user mouse put on the spot from any scatter charts. It will display the name of the node, the x attribute value and y attribute value, and other information that user designed to display.
4. Input check.
   Using Javascript to check user selections' validation. For example, user are not allowed to choose empty content and click the submit button. There will be some alert message.

4 Installation and Setup

This project provides a web-based user interface, which allows users access it locally or remotely. In this section, we give a brief introduction of installation.
The project require following package/software:

4.1 Dependencies

1. Modern web browser
   A modern web browser is required for the user interface. Because the interactive and data visualisation are highly depend on JavaScripts libraries, the latest version browser is highly recommended for better efficiency and experience.
2. Python 2.7.9 or later
   The website framework and data processing are based on Python programming language.
3. Django
   Django provides a basic web server, as well as a web framework. It lays between browser and data. To install Django, using the command (Please note that you might need the administrator privilege, e.g. sudo before pip command):

   ```
   pip install django
   ```

4. django-cors-header
   django-cors-headers library enables cross sites recourse sharing of the server.

   ```
   pip install django-cors-headers
   ```

5. pandas, scipy, matplotlib
   These libraries are used for data processing and bar charts plotting.

   ```
   pip install pandas scipy matplotlib
   ```

4.2 Start Web Server

To run the project, go to server folder of the project, use following command.

```
python manage.py runserver 8000
```

This command will run the server and listen to localhost port 8000. Next, open a browser then navigate to http://127.0.0.1:8000/.

5 UI Features

We implemented the following features for the quality metrics with interactive modes to compare and view quality about algorithms. Currently we support 3 algorithms and a ground truth is added to compare the precision and the recall ratios. Figure 3 shows the sidebar of the application and a user could choose the feature he wants to review.

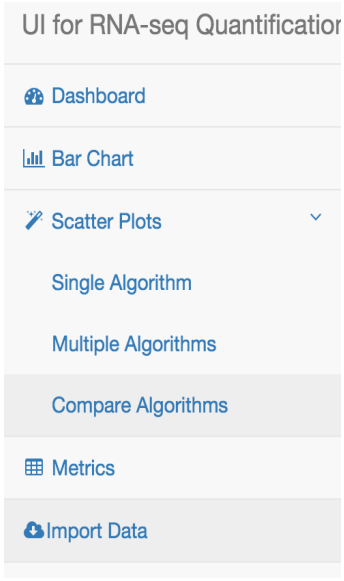


Figure 3: Web Application sidebar

5.1 Bar Chart for algorithms

We support the bar chart for all the algorithms to calculate the number of reads for each algorithm mapped in the experiment. One example is showed in figure 4.

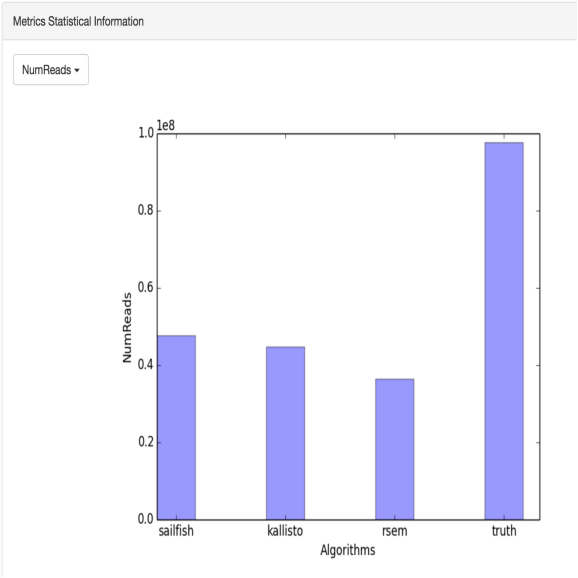
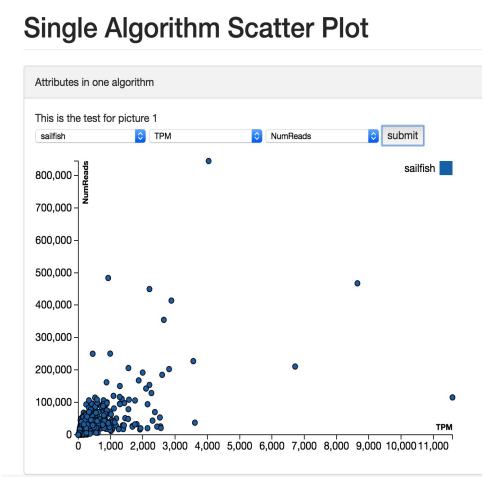


Figure 4: Bar Chart - NumReads

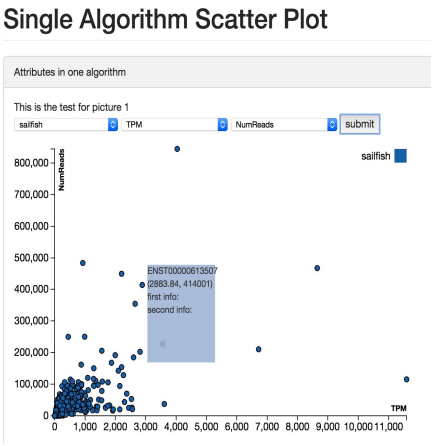
5.2 Scatter Plot for a Single Algorithm

For a single algorithm, we support the scatter plot about any two values such as transcript GC content versus abundance estimates. To plot a scatter plot, the user could go to the scatter plot for single algorithm page, choose an algorithm and the x-axis feature and the y-axis feature. With such a plot, every node on the scatter plot is one transcript and the coordinate is the chosen value pairs and as one example, the scatter plot for

sailfish is showed in figure 5a.
For an interactive mode, you could move the mouse to a node you want to view and the information would automatically showed as in figure 5b.



(a) Sailfish - NumReads vs Abundant Estimate



(b) Sailfish - NumReads vs Abundant Estimate (Interactive)

Figure 5: Pictures of Clustering with Random Sampling

5.3 Scatter Plot for two Algorithms on the same metrics

We could also compare two algorithms based on the same scatter plots. For example, we could compare the distribution between GC Content and Number of Reads for both Sailfish and Kallisto, as showed in figure 6.

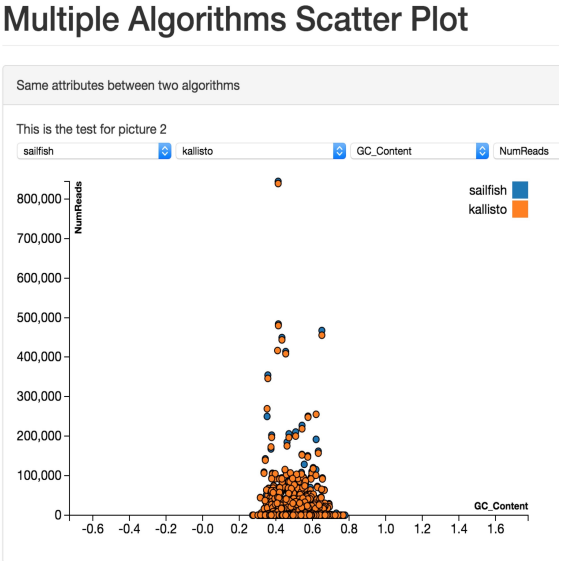
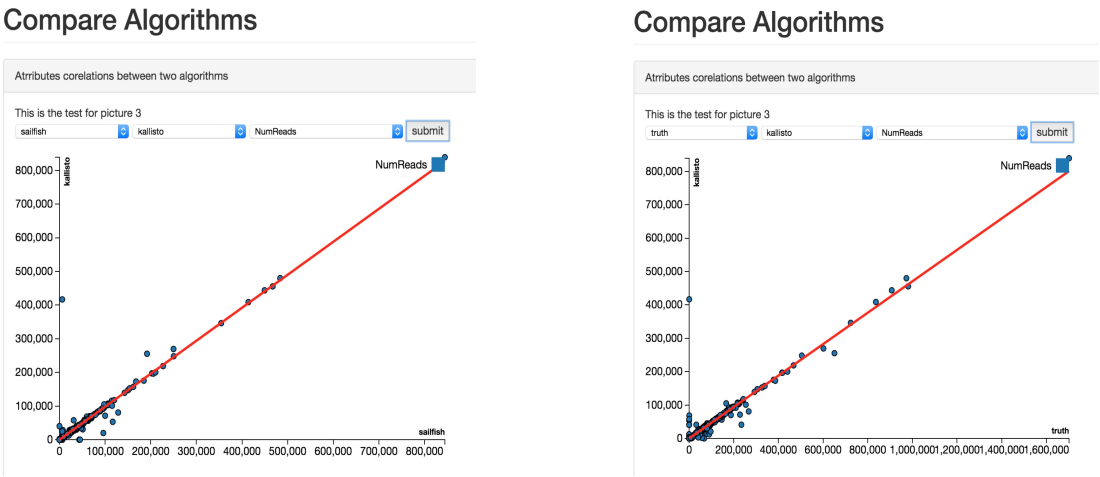


Figure 6: Sailfish and Kallisto: GC Content vs NumReads

5.4 Scatter Plot for two Algorithms on the same feature

The same features from two algorithms are also comparable. For example, Number of Reads for both Sailfish and Kallisto. We expected they are similar if both of them are good algorithms (figure 7a). And we

could also compare it with the ground truth (figure 7b).

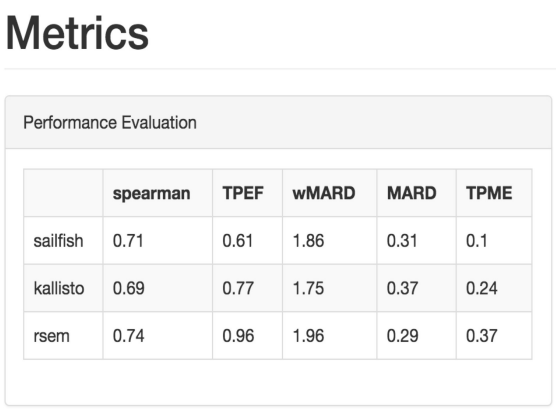


(a) Compare NumReads between Sailfish and Kallisto (b) Compare NumReads between ground truth and Kallisto

Figure 7: Pictures of Clustering with Random Sampling

5.5 Performance Evaluation

We could also calculate the performance evaluation based on ground truth. The ground truth could be uploaded at upload page showed in figure 8



	spearman	TPEF	wMARD	MARD	TPME
sailfish	0.71	0.61	1.86	0.31	0.1
kallisto	0.69	0.77	1.75	0.37	0.24
rsem	0.74	0.96	1.96	0.29	0.37

Figure 8: Performance Evaluation

By given the ground truth, we now could calculate the following metrics: Spearman corr., TPEF, TPME, MARD, wMARD [3].

6 Extensibility

6.1 Upload new data

Currently the uploading of ground truth is added to re-calculate the performance evaluation results.
We also considered the extensibility in our implementation. Currently the result files are directly uploaded to the server side and by upload and rename to the targeted name, call *DataAnalyzor.load_data*(), the server

could reload all data to replace the old metrics.
Currently the filename for each algorithm is coded in *dataAnalyzor*'s constructor.

6.2 Add new algorithm

To add a new algorithm, one just need to add a related method in *AnalyzorBuilder* with the corresponding result file, parsing method same as in file *ParsingUtility.py* and the new algorithm could automatically be supported by the whole framework.
At both the front-end and back-end, the algorithms are queried from the *DataAnalyzor*. Once the algorithms and the corresponding methods for reading the data in, the new algorithm and the data would be showed as same as other old features.

7 Improvement and Future Works

We still have several unfinished features such as:

1. Add upload file feature at the website.
2. Automatically add new algorithms. Currently the algorithm parser has predefined columns. If the result files are using unique names, such as *NumReads*, we could automatically add new algorithms just by upload the result file.

8 Acknowledge

Thanks and regards to Professor Rob Patro's lectures. We were inspired by the algorithms and the vivid examples.
Thanks and regards to the TA Hirak. He helped us a lot on the project.

References

[1] "SB Admin 2 - Free Bootstrap Admin Theme." Start Bootstrap. N.p., n.d. Web. 15 Dec. 2015. $< http://startbootstrap.com/template-overviews/sb-admin-2/ >$.

[2] "D3 Scatterplot Example." D3 Scatterplot Example. N.p., n.d. Web. 15 Dec. 2015. $< http://bl.ocks.org/weiglemc/6185069 >$.

[3] Srivastava, Avi, Hirak Sarkar, and Rob Patro. "RapMap: A Rapid, Sensitive and Accurate Tool for Mapping RNA-seq Reads to Transcriptomes." *bioRxiv* (2015): 029652.