

## ОГЛАВЛЕНИЕ

|  |    |
|--|----|
| ВВЕДЕНИЕ.....  | 6  |
| 1. Обзор предметной области .....                            | 7  |
| 1.1. Биоинформатика .....                                    | 7  |
| 1.1.1. Анализ экспрессии генов .....                         | 7  |
| 1.1.2. Используемые методы .....                             | 7  |
| 1.2. Существующие решения для анализа экспрессии генов ..... | 8  |
| 1.2.1. R/Bioconductor .....                                  | 8  |
| 1.2.2. GENE-E .....  | 10 |
| 1.2.3. morpheus.js .....                                     | 10 |
| 1.2.4. ProjectX .....  | 13 |
| 1.3. Использование R в веб-разработке .....                  | 14 |
| 1.3.1. Shiny .....   | 14 |
| 1.3.2. OpenCPU .....   | 14 |
| 1.4. Инфраструктура .....                                    | 16 |
| 1.4.1. Docker .....  | 16 |
| 1.4.2. Apache.....   | 17 |
| 1.5. Форматы сериализации данных .....                       | 17 |
| 1.5.1. JSON .....  | 17 |
| 1.5.2. Protocol Buffers .....                                | 18 |
| 1.6. Открытые источники данных экспрессии генов .....        | 18 |
| 1.6.1. Gene Expression Omnibus .....                         | 18 |
| 1.6.2. The Cancer Genome Atlas .....                         | 19 |
| 1.7. Постановка задачи.....                                  | 19 |
| 1.7.1. Цель работы .....                                     | 19 |
| 1.7.2. Основные задачи.....                                  | 19 |
| 1.7.3. Требования к веб-приложению phantusus.....            | 20 |
| Выводы по главе 1.....                                       | 20 |
| 2. Архитектура проекта phantusus.....                        | 21 |
| 2.1. Общая схема .....                                       | 21 |
| 2.2. Взаимодействие между клиентом и сервером.....           | 21 |
| 2.2.1. OpenCPU .....   | 22 |
| 2.2.2. Protocol Buffers .....                                | 23 |

|  |    |
|--|----|
| 2.3. Поддержка ExpressionSet .....                                       | 24 |
| 2.3.1. Поддержка ExpressionSet на стороне клиента.....                   | 24 |
| 2.3.2. Создание ExpressionSet из внешних данных на стороне сервера ..... | 25 |
| 2.4. Способы визуализации.....   | 26 |
| 2.4.1. Отрисовка графиков на стороне сервера .....                       | 26 |
| 2.4.2. Визуализация на стороне клиента .....                             | 26 |
| Выводы по главе 2.....   | 27 |
| 3. Реализация и использование .....                                      | 28 |
| 3.1. Реализованные методы анализа экспрессии.....                        | 28 |
| 3.1.1. Загрузка данных из GEO .....                                      | 28 |
| 3.1.2. Метод главных компонент и визуализация его результата .....       | 29 |
| 3.1.3. Кластеризация методом kmeans .....                                | 31 |
| 3.1.4. Анализ дифференциальной экспрессии.....                           | 33 |
| 3.2. Инфраструктура проекта phantasmus .....                             | 35 |
| 3.2.1. Структура git-репозитория .....                                   | 35 |
| 3.2.2. Запуск приложения .....   | 36 |
| 3.2.3. Кэш для данных из GEO.....  | 38 |
| 3.3. Настройка с помощью Apache .....                                    | 39 |
| 3.3.1. Переадресация OpenCPU-сервера.....                                | 39 |
| 3.3.2. Балансировщик для multi-user соединения .....                     | 39 |
| Выводы по главе 3.....   | 39 |
| ЗАКЛЮЧЕНИЕ.....  | 40 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....                                    | 41 |
| ПРИЛОЖЕНИЕ А. Протокол сериализации в ProtoBuf .....                     | 44 |
| ПРИЛОЖЕНИЕ Б. Dockerfile.....  | 45 |
| ПРИЛОЖЕНИЕ В. Конфигурационный файл Apache .....                         | 46 |

## **ВВЕДЕНИЕ**

Биоинформатика образована на стыке биологических направлений и информатики, как реакция на всё увеличивающийся объем данных, требующих сложных, быстрых и качественных алгоритмов для обработки и анализа. Разумеется, биоинформатикой занимаются как, непосредственно, информатики, которые обладают навыками программирования и могут реализовывать алгоритмы самостоятельно, так и биологи, которые отлично могут интерпретировать результаты работы алгоритмов и сами данные, но не имеют соответствующей подготовки для использования этих методов. Исследователям для более продуктивной работы нужны удобные и интуитивно понятные инструменты для анализа данных, которые бы хорошо покрывали все их потребности в реализованных методах и алгоритмах. На данный момент, таких инструментов достаточно мало, а в тех, что есть, неполноценный функционал. Таким образом, целью данной работы является разработка инструмента для полноценного анализа биологических данных.

## ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

В данной обзорной главе будет представлена предметная область биоинформатики, введено понятие экспрессии генов, рассмотрены существующие решения для анализа экспрессии генов, а также перечислен ряд технологий, которые используются или могут быть использованы для создания инструментов анализа экспрессии генов.

### 1.1. Биоинформатика

**Биоинформатика** — наука, объединяющая в себе методы прикладной математики, статистики, информатики для создания новых методов и алгоритмов для анализа разного рода биологических данных.

Биоинформатика занимается биохимией, биофизикой, экологией и многими другими областями биологии. Однако фокус в данной работе будет сосредоточен на конкретную задачу биоинформатики — анализ экспрессии генов.

#### 1.1.1. Анализ экспрессии генов

**Экспрессия генов** — процесс преобразования наследственной информации от гена (в виде последовательности нуклеотидов ДНК) в функциональный продукт (РНК или белок).

Анализ экспрессии генов позволяет выяснить, как ведет себя каждый отдельный ген в разных условиях, тканях или организмах.

Экспрессия гена в образце характеризуется вещественным числом, которое также можно назвать некоторой мерой активности гена в данных условиях.

#### 1.1.2. Используемые методы

Как было сказано ранее, биоинформатика использует в себе математику, информатику и статистику. Соответственно, задача анализа экспрессии генов сводится к исследованию путем статистических методов и алгоритмов числовой двумерной матрицы, где в виде вещественных чисел демонстрируется активность каждого гена в каждом образце. Пример такой матрицы можно увидеть в таблице 1.

Одним из первоочередных методов, применяемых для анализа экспрессии, является *визуальный анализ*. Числовая матрица представля-

Таблица 1 – Срез матрицы GSE14308. Строки матрицы соответствуют генам, столбцы — образцам.

|        | GSM357839 | GSM357841 | GSM357842 | GSM357843 | GSM357844 |
|--------|-----------|-----------|-----------|-----------|-----------|
| Rps29  | 16.32     | 16.30     | 16.25     | 16.32     | 16.30     |
| Rpl13a | 16.27     | 16.23     | 16.32     | 16.30     | 16.27     |
| Rps3a1 | 16.23     | 16.19     | 16.30     | 16.25     | 16.25     |
| Rpl38  | 16.21     | 16.25     | 16.27     | 16.27     | 16.21     |
| Tmsb4x | 16.30     | 16.32     | 16.23     | 16.21     | 16.32     |

ется в виде *тепловой карты*, где цветом показана активность каждого гена.

На рисунке 1 можно увидеть визуализацию матрицы экспрессии из таблицы 1 в виде тепловой карты.

Также к основным методам анализа относятся:

- кластеризация:
  - иерархическая: метод упорядования данных таким образом, чтобы их можно было визуализировать в виде дерева (дендрограммы);
  - вероятностная: метод разбиения данных на несколько групп (кластеров);
- дифференциальная экспрессия: метод, позволяющий сравнивать поведение генов в разных условиях и искать закономерности;
- метод главных компонент: метод для уменьшения размерности данных с наименьшей потерей информации.

## 1.2. Существующие решения для анализа экспрессии генов

### 1.2.1. R/Bioconductor

**R** — язык программирования для статистического анализа данных и работы с графикой [1].

**Bioconductor** — библиотека, содержащая в себе множество реализаций биоинформатических алгоритмов и методов обработки биологических данных на *R*. Она постоянно обновляется, пополняется новыми библиотеками, модерируется сообществом [2]. *R* и *Bioconductor* очень популярны в биоинформатической среде ввиду предоставляемых возможностей.

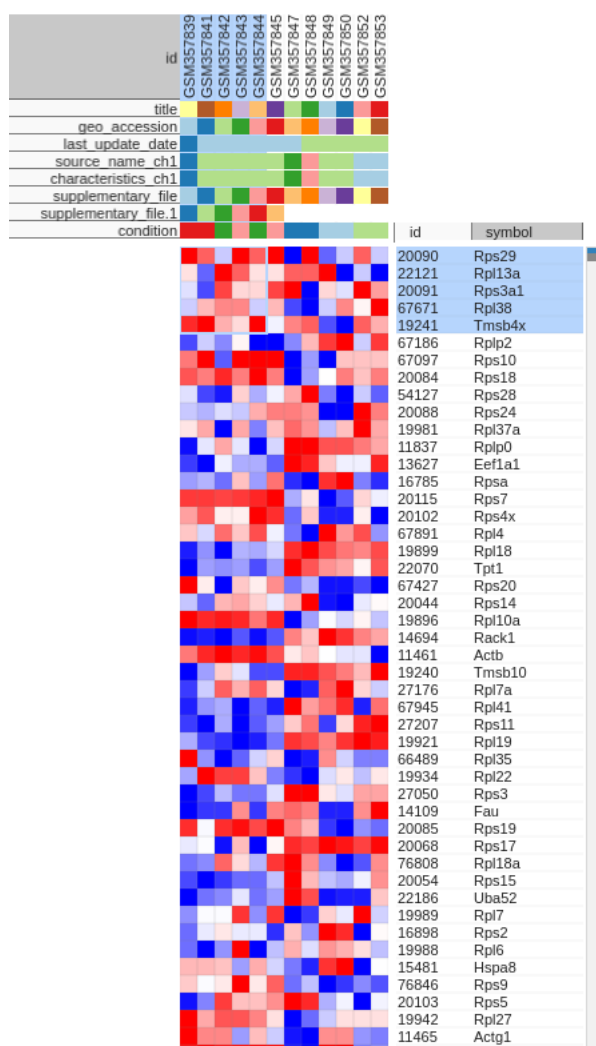


Рисунок 1 – Визуализация матрицы GSE14308 в виде тепловой карты в веб-приложении Morpheus

Однако для качественного и полноценного анализа с помощью этих инструментов, нужно иметь навыки программирования на *R*, что весьма неудобно для исследователей биологических специальностей.

#### 1.2.1.1. Biobase и ExpressionSet

Необходимый минимум функций для работы с геномными данными содержится в *R*-пакете *Biobase* [2].

Класс *ExpressionSet* [3] также содержится в *Biobase*. Он помогает представлять данные об экспрессии генов в удобном формате:

- *assayData* — описание матрицы:
  - *features* — количество генов;
  - *samples* — количество образцов;
  - *exprs* — числовая матрица экспрессии;

- `phenoData` — аннотация к образцам:
  - `sampleNames` — идентификаторы образцов;
  - `varLabels` — названия характеристик;
  - `varMetadata` — описание характеристик;
  - `pData` — матрица характеристик;
- `featureData` — аннотация к генам:
  - `featureNames` — идентификаторы генов;
  - `fvarLabels` — названия характеристик;
  - `fvarMetadata` — описание характеристик;
  - `fData` — матрица характеристик.

Для доступа к каждому из элементов есть одноименная функция, что позволяет удобно взаимодействовать с экземплярами класса. Также многие из функций обработки данных в *Bioconductor* и в *Biobase* в частности завязаны на использование `ExpressionSet`.

### 1.2.2. GENE-E

**GENE-E** — Платформа для анализа данных и визуального исследования данных, созданная на *Java* и *R* [4]. Содержит в себе множество полезных для исследования инструментов: тепловые карты, кластеризацию, фильтрацию, построение графиков и т.д. Позволяет исследовать любые данные в виде матрицы. К тому же, содержит дополнительные инструменты для данных генетической экспрессии.

Недостатки:

- чтобы использовать, необходимо устанавливать на свой компьютер;
- поддержка данного приложения прекратилась в связи с созданием *morpheus.js* [5];
- не имеет открытого исходного кода, а только *API* для взаимодействия и создания новых приложений на его основе.

### 1.2.3. morpheus.js

**Morpheus.js** — веб-приложение для визуализации и анализа матриц от создателя *GENE-E* [5]. В отличие от предшественника, создано на *JavaScript* и с открытым исходным кодом. Удобно для использования исследователями без навыков программирования и так же, как и *GENE-E*, применимо к любым матрицам.

Недостатки:

- ограниченный набор функций, которых недостаточно для полноценного анализа;
- для расширения биоинформатическими алгоритмами, не прибегая к дополнительным инструментам, требуется реализовывать их заново на *JavaScript*;

Далее будут описаны основные компоненты, классы и методы, реализованные в *morpheus.js*, с которыми необходимо работать в случае расширения данного приложения.

### 1.2.3.1. Чтение данных

В *morpheus.js* данные могут быть загружены из файла одним из следующих путей:

- Из компьютера;
- По *URL*-ссылке;
- Из *Dropbox*.

Допустимые форматы загружаемых файлов:

- *txt*-файл с *tab*-разделителями;
- *Excel*-таблица;
- *Mutation Annotation Format (MAF)* [6];
- *Gene Cluster Text (GCT)* [7];
- *Gene Matrix Transposed (GMT)* [8].

Для каждого формата файла в исходном коде *morpheus.js* присутствует соответствующий обработчик данных.

Также, *morpheus.js* предлагает набор предзагруженных данных из базы *The Cancer Genome Atlas (TCGA)* [9].

### 1.2.3.2. Класс Dataset

Одним из ключевых классов всего веб-приложения является класс *Dataset*. В каждом экземпляре этого класса хранится вся необходимая информация о данных, в которую входят:

- числовая матрица, характеризующая уровень экспрессии всех генов во всех образцах;
- количество строк и столбцов в матрице;
- аннотация к образцам, например:



- пол, возраст, контактную информацию испытуемых, если образцы были взяты с людей;
- есть или нет инфекция в данном образце;
- способ лечения;
- контакты ответственного за взятие данного образца и пр.;
- аннотация к генам, например:
  - идентификатор гена в том или ином стандарте;
  - числовые характеристики гена (средний уровень экспрессии по образцам, номер кластера) и пр.

Аннотация реализована в классе `MetadataModel`, который представляет собой не что иное, как набор именованных векторов с характеристиками. В каждом векторе хранятся:

- название;
- формат (строка, число);
- массив значений.

Для векторов так же предусмотрены утилиты для визуализации. Так, например, есть возможность показать аннотацию в виде текста и/или цветом, что удобно для категориальных характеристик.

#### 1.2.3.3. Класс `SlicedDatasetView`

Чаще всего во время работы программы экземпляры класса `Dataset` становятся обернуты в оболочку из `SlicedDatasetView`. Этот дополнительный класс дает возможность не пересоздавать каждый раз `Dataset`, а просто добавляет к данным информацию о том, какие индексы строк и столбцов выбраны и используются в данный момент и в каком порядке.

#### 1.2.3.4. Класс `HeatMap`

Данный класс предназначен для визуализации данных, обернутых в класс `Dataset` или `SlicedDatasetView`. Он дает возможность выбирать, какая аннотация будет представлена на экране, цветовой код, выбирать строки и столбцы, с которыми будут работать те или иные инструменты.

### 1.2.3.5. Класс DatasetUtil

Класс DatasetUtil содержит в себе утилиты для обработки и чтения данных в Dataset:

- обработка входных файлов с данными и отправка их на соответствующий класс чтения в зависимости от формата;
- поиск по данным;
- преобразование данных в JSON и обратно.

### 1.2.3.6. Реализованные методы

В *morpheus.js* имеются реализации следующих методов обработки и анализа данных:

- Adjust — инструмент для корректировки данных:
  - $\log_2$ ;
  - $\log_2^{-1}$ ;
  - Квантиль-нормализация;
  - Z-тест;
  - Устойчивый Z-тест;
- Collapse — инструмент, позволяющий агрегировать строки или столбцы с одинаковыми значениями с помощью функции: *min*, *max*, *mean*, *median*, *sum*, максимум 25-го и 75-го перцентилей;
- CalculatedAnnotation — добавление вычисленной аннотации для строк или столбцов;
- Similarity Matrix — построение матрицы соответствия строк или столбцов друг другу;
- Transpose — транспонирование матрицы;
- t-SNE — реализация алгоритма снижения размерности;
- ChartTool — построение графиков.

Также присутствуют фильтрация и сортировка.

### 1.2.4. ProjectX

**ProjectX** — веб-приложение, созданное в рамках выпускной квалификационной работы [10], которое соединяло в себе возможности, предоставляемые *API GENE-E* и методы из библиотеки *Bioconductor* с помощью *OpenCPU*. Реализовано было данное приложение во фреймворке *Django*. Преимуществом данного приложения перед *GENE-E* была во-

производительность исследований (на каждом этапе пользователь мог скачать исполняемый *R*-код, эквивалентный коду, выполненному в сервисе), также он содержал в себе большее число методов анализа и обработки данных.

Недостатки:

- проект использовал в своей базе приложение, которое больше не поддерживается разработчиками;
- работа над проектом была завершена до того, как у него появились пользователи, так что оно осталось невостребованным.

### 1.3. Использование *R* в веб-разработке

Как было сказано ранее, *R* удобно использовать для статистического анализа данных. Но чтобы использовать методы *R* в веб-приложениях, возникает необходимость в дополнительных технологиях для интеграции инструментов веб-разработки и *R*.

#### 1.3.1. Shiny

**Shiny** — фреймворк для создания веб-приложений, используя только язык *R* [11].

#### 1.3.2. OpenCPU

**HTTP** (*HyperText Transfer Protocol*) — протокол прикладного уровня передачи данных на основе технологии «клиент-сервер» [12].

**HTTP API** - набор процедур и функций, вызов которых и возвращение их результата осуществляется посредством протокола *HTTP*.

**RPC** (*Remote Procedure Call*) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве.

**Веб-сервер** — сервер, принимающий *HTTP*-запросы от клиентов, обычно веб-браузеров, и выдающий им *HTTP*-ответы.

**OpenCPU** — система для встроенных научных вычислений и воспроизводимых исследований, предоставляющая *HTTP API* для вызова *R*-функций и взаимодействия с *R*-объектами с помощью *POST* и *GET* запросов [13].

### 1.3.2.1. OpenCPU-сервер

*OpenCPU*-сервер можно запустить одним из следующих способов:

- однопользовательский сервер: сервер запускается из активной *R*-сессии и предназначен в основном для разработки и локального использования. Такой сервер не поддерживает параллельных запросов, так как *R*-сессии работают в одном потоке;
- облачный сервер: этот сервер можно запустить на *Ubuntu 16.04* и выше. Запуск и настройка облачного сервера осуществляется с помощью *Apache* или *Nginx*. В отличие от однопользовательского сервера, здесь поддерживаются параллельные запросы и настройка безопасности.

Однопользовательский сервер в большинстве случаев работает значительно быстрее, чем облачный, так как последний управляется с помощью *rApache* [14], что добавляет удобства в использовании, но замедляет доступ к серверу. Первым же можно пользоваться как многопользовательским, используя несколько экземпляров сервера, доступ к которым осуществляется с помощью *Apache* и балансировщика, который тот предоставляет.

### 1.3.2.2. OpenCPU API

Входной точкой *API* является `/оспу/`. GET-запрос используется для получения некоторого ресурса, а POST-запрос используется для *RPC*. На таблице 2 можно увидеть более подробное описание запросов.

Таблица 2 – Запросы к *OpenCPU*-серверу, их аргументы и действия

| Запрос | Действие          | Аргументы                    | Пример                                 |
|--------|-------------------|------------------------------|--|
| GET    | посмотреть объект | формат представления объекта | GET /оспу/library/MASS/R/cats/json     |
| POST   | вызвать функцию   | аргументы функции            | POST /оспу/library/stats/R/rnorm       |
| GET    | прочитать файл    | -                            | GET /оспу/library/MASS/NEWS            |
| POST   | запустить скрипт  | аргументы запуска скрипта    | POST /оспу/library/MASS?scripts/ch01.R |

На *OpenCPU*-сервере могут быть доступны:

- библиотеки и содержащиеся в них пакеты: `/оспу/library/pkgname/`;
- пакеты из *git*-репозитория: `/оспу/apps/gituser/reponame/`;

- пакеты, установленные в домашней директории *Linux*-пользователя: `/оспу/user/username/library/pkgname/`;
- временные сессии, содержащие вывод от запуска функции или скрипта: `/оспу/tmp/key/`.

### 1.3.2.3. Opencpu.js

Так как зачастую *OpenCPU* используется разработчиками для использования *R* в веб-приложениях для анализа и визуализации данных, для удобной интеграции *JavaScript* и *R* существует библиотека *opencpu.js*, которая реализует *RPC*-вызовы по принципу *Asynchronous JavaScript and XML (Ajax)* [15]. Таким образом запросы обрабатываются в фоновом режиме, тем самым не замедляя работу графического интерфейса и вычислений, осуществляемых на стороне клиента.

В данной библиотеке реализован класс *Session*, содержащий в себе ключ сессии, адреса на ссылки, файлы и переменные, существующие внутри сессии.

Для подключения к *R*-пакету на *OpenCPU*-сервере удобно использовать код, представленный на листинге 1. Для успешного подключения *R*-пакет должен быть предварительно установлен на *host*-машину, на которой располагается сервер.

```
1  ocpu.seturl("//hostname/ocpu/library/{pkgname}/R");
```

Листинг 1 – Подключение к *R*-пакету

После этого можно вызывать и запускать функции, содержащиеся в данном *R*-пакете, например, как в листинге 2.

```
1  var req = ocpu.rpc("function.name", arguments, callback(session) {
2    \\ Handling result
3  });
```

Листинг 2 – Шаблон вызова *R*-функции из *JavaScript*

## 1.4. Инфраструктура

### 1.4.1. Docker

**Docker** — программное обеспечение для автоматизации запуска и внедрения приложений внутри контейнеров [16].

Для дальнейшего описания данного инструмента введем несколько определений.

**Образ** — отдельный исполняемый пакет, включающий себя все необходимое для запуска единицы программного обеспечения, в том числе исходный код, библиотеки, переменные окружения, конфигурационные файлы. Зачастую образ построен на основе другого образа с дополнительной конфигурацией. Образ компилируется по *Dockerfile*, каждая команда в котором соответствует новому слою. При перекомпиляции обновляются только те слои, которые изменились.

**Контейнер** — запущенный экземпляр образа. Контейнер обычно выполняется изолированно от окружения, имея доступ к файлам или портам хост-системы только при наличии соответствующей конфигурации.

В отличие от виртуальных машин, которые запускают гостевую операционную систему в каждом экземпляре, контейнеры могут разделять общее ядро, и вся информация, которая должна быть в контейнере, это исполняемый процесс и его зависимости. Исполняемые процессы из контейнеров работают как нативные процессы, и могут управляться по отдельности.

Для контроля версий и хранения образов в открытом доступе используется *Docker Hub* [17]. В этом хранилище можно как добавлять репозитории, управляемые вручную, так и поддерживать автоматические сборки (*Automated Build*), которые привязаны к репозиториям на в популярных системах контроля версий: GitHub [18] и Bitbucket [19].

#### 1.4.2. Apache

**Apache HTTP Server Project** — устойчивый, полностью открытый *HTTP*-сервер [20].

*Apache* позволяет конфигурировать веб-сервисы, переадресацию, балансировку нескольких экземпляров серверов.

### 1.5. Форматы сериализации данных

#### 1.5.1. JSON

**JSON** — текстовый формат представления данных, основанный на *JavaScript*, который, среди прочих достоинств, легко читается человеком.

*JSON*-текст представляет собой одну из следующих структур:

- пара *ключ: значение*, где ключом может быть только регистрозависимая строка, а в качестве значения может выступать массив, число, строка, литералы или другой *JSON*-объект;
- набор значений.

### 1.5.2. Protocol Buffers

**Protocol Buffers (Protobuf)** — гибкий, универсальный и автоматизированный механизм для сериализации структурированных данных [21].

Структура информации задается с помощью \*.proto-файлов в форме сообщений (message).

*ProtoBuf*-формат не является человекочитаемым, данные хранятся в двоичном формате. Для десериализации и дальнейшего чтения необходим соответствующий \*.proto-файл. Файл с форматом компилируется соответствующим выбранному языку программирования компилятором, таким образом будет создан класс доступа к данным. С помощью этого класса уже можно сериализовать/десериализовать данные, получать данные с помощью get/set-методов и пр.

## 1.6. Открытые источники данных экспрессии генов

### 1.6.1. Gene Expression Omnibus

**Gene Expression Omnibus (GEO)** — международный публичный репозиторий, агрегирующий и распространяющий различные формы геномных данных от исследовательского сообщества [22].

*GEO* предоставляет устойчивую базу данных для эффективного хранения геномных данных, содержит их в качественно аннотированном формате и дает возможность удобно как добавлять новые данные для публикации, так и запрашивать интересующие данные для исследований.

В *GEO* содержатся следующие типы и форматы данных:

- информация о платформе, на которой производилось секвенирование генов (*Platform records*): GPLxxx;
- информация об образцах и условиях, в которых производились исследования этих образцов (*Sample records*): GSMxxx;

- информация о непосредственно исследованиях, полученные данные и выводы (*Series records*): GSExxx.
- обработанная и подготовленная для дальнейшего статистического анализа GEO-кураторами информация об исследованиях ((*DataSet records*)): GDSxxx.

В библиотеке *Bioconductor* есть *R*-пакет *GEOquery* для удобной загрузки данных из GEO [23].

### 1.6.2. The Cancer Genome Atlas

**The Cancer Genome Atlas (TCGA)** — проект, нацеленный на систематизацию данных о генетических мутациях, приводящих к возникновению рака. На данный момент участниками проекта отсеквенировано и проанализировано 33 вида рака [9].

## 1.7. Постановка задачи

Рассмотрев существующие решения для анализа экспрессии генов и инструментов, которые могли бы пригодиться для будущих решений, можно сформулировать цель и основные задачи данной работы

### 1.7.1. Цель работы

Целью работы является создание веб-приложения, интегрирующего существующие возможности веб-приложения *morpheus.js* и методы анализа, реализованные в *Bioconductor*.

### 1.7.2. Основные задачи

Для достижения поставленной цели были сформулированы следующие задачи:

- а) разработать способ взаимодействия между *js*-клиентом и *R* и встроить его в *morpheus.js*, чтобы избежать реализации с нуля уже существующих алгоритмов;
- б) реализовать графический интерфейс в *js*-клиенте и серверную реализацию в *R*-пакете;
- в) соединить все составляющие в одном веб-приложении *phantasus*;
- г) запустить веб-приложение в открытый доступ для исследователей.



### 1.7.3. Требования к веб-приложению *phantasus*

#### 1.7.3.1. Доступность

Необходимо, чтобы веб-приложение *phantasus* было доступно для исследователей независимо от их местоположения и времени суток. Варианты действий:

- а) сделать его доступным по определенному веб-адресу, и тогда пользователь сможет продолжать исследования из любой точки, где есть подключение к интернету;
- б) предоставить возможность запускать приложение локально, например, с помощью *Docker* или внутри *R*.

#### 1.7.3.2. Возможность дальнейшего расширения функционала

Как уже было сказано выше, библиотека *Bioconductor* постоянно обновляется и пополняется новыми алгоритмами, а исследователи находят новые методы для анализа экспрессии генов, так что необходимо не только реализовать дополнительные методы, но также отладить и описать алгоритм действий для добавления новых.

### Выводы по главе 1

В данной главе были введены понятия биоинформатики и анализа экспрессии генов, обозначена цель работы и задачи, выполнение которых необходимо для ее достижения, представлены существующие решения с их достоинствами и недостатками, а также перечислен ряд инструментов, которые могут быть использованы для решения поставленных задач работы.

## ГЛАВА 2. АРХИТЕКТУРА ПРОЕКТА PHANTASUS

В этой главе будет рассмотрена архитектура проекта *phantasus*, общая схема, ключевые компоненты и их взаимодействие между ними. Также будут описаны сопутствующие инструменты и их предназначение в системе, и ключевые для архитектуры выдержки из исходного кода.

### 2.1. Общая схема

Проект *phantasus* состоит из следующих компонент:

- клиентская сторона проекта: *phantasus.js* — модифицированный *morpheus.js*, дополненный интерфейсами для новых инструментов, поддержкой *ExpressionSet* и сериализации в *ProtoBuf*;
- серверная сторона проекта: *R*-пакет *phantasus* — пакет, включающий в себя вычисления и реализации новых инструментов.

Общую схему взаимодействия можно увидеть на диаграмме 2.

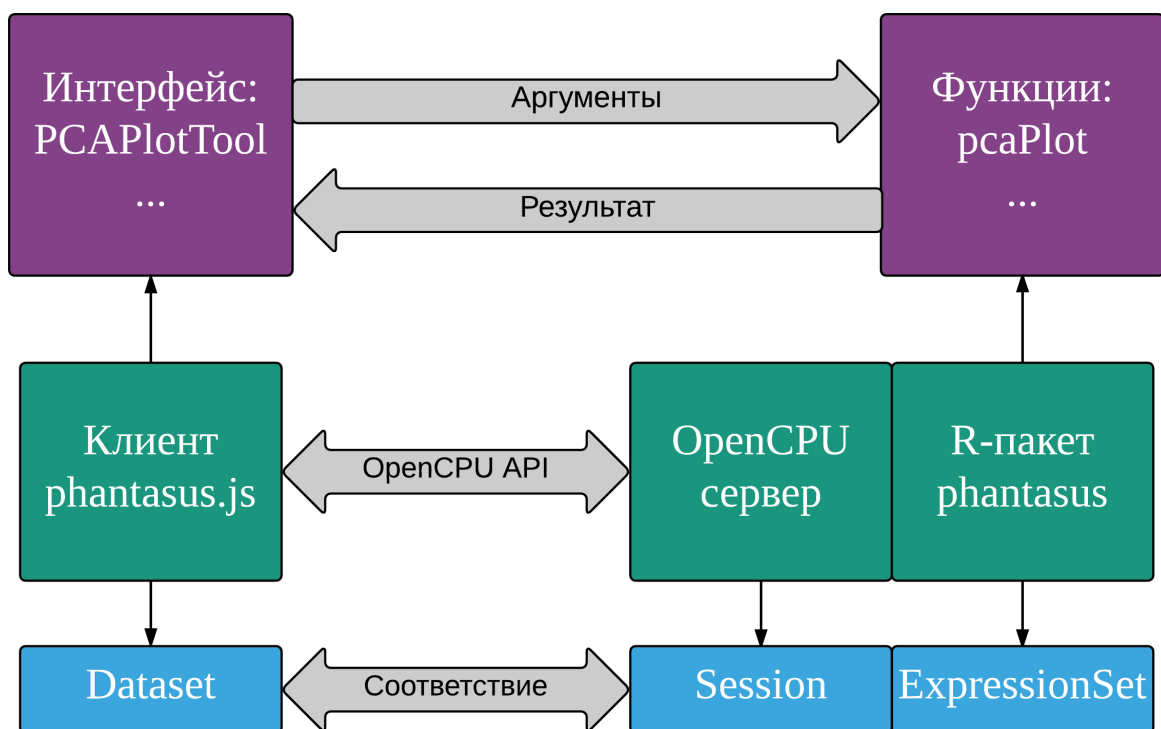


Рисунок 2 – Общая схема проекта *phantasus*

### 2.2. Взаимодействие между клиентом и сервером

Все инструменты, реализованные в проекте, имеют две компоненты:

- графический интерфейс в *phantasus.js*;
- вычислительную реализацию в *R*-пакете *phantasus*.

Соответственно, для связи этих компонент необходим своеобразный мост между клиентом, написанном на *JavaScript*, и функциями, написанными на *R*. В качестве такого моста используется описанный в обзоре *OpenCPU*.

Помимо наличия моста также необходимо, чтобы общение между клиентом и сервером происходило достаточно быстро, что приводит к потребности в дополнительной сериализации сообщений. В проекте для этого используется протокол *ProtoBuf*. В данном разделе будет подробно описано об использовании данных технологий при реализации взаимодействия между клиентом и сервером.

### 2.2.1. OpenCPU

Как было сказано выше, *OpenCPU* используется для связи между *JavaScript*-клиентом и *R*-функциями.

#### 2.2.1.1. Использование OpenCPU со стороны клиента

Как было сказано в обзоре, для удобной интеграции *JavaScript* и *R* существует *JavaScript*-библиотека *opencpu.js*, в которой реализованы *RPC*-вызовы. С помощью этой библиотеки в проекте реализована взаимосвязь между графическим интерфейсом инструментов и соответствующих функций из *R*-пакета.

Каждый из инструментов, реализованных в *phantasus.js* действует по следующему принципу:

- а) обработка аргументов, полученных в интерфейсе;
- б) подготовка данных и аргументов к отправке на сервер;
- в) *RPC*-вызов функции, как на листинге 2;
- г) обработка содержащегося в полученной сессии результата в виде глобальной переменной, ответа функции или сохраненного в сессии файла;
- д) передача ответа в интерфейс.

#### 2.2.1.2. Использование OpenCPU со стороны R-пакета

Со стороны *R*-функций никаким специальным образом не обозначается, что результат работы будет передан именно в *JavaScript*-клиент,

все реализованные функции достаточно универсальны. Функция возвращает результат, в некоторых случаях дополнительно сохраняя его в файл или глобальную переменную, что в дальнейшем использует клиент, который заранее знает, в каком формате получит ответ.

### 2.2.2. Protocol Buffers

*OpenCPU API* поддерживает передачу сообщений в *ProtoBuf*-формате. Однако, *JavaScript*-библиотека не предусматривает такой возможности, так что в процессе работы над проектом была добавлена сериализация данных в *ProtoBuf* на стороне клиента и поддержка сериализованных сообщений в *opencpu.js*.

#### 2.2.2.1. Сериализация данных со стороны клиента — *protobuf.js*

Чаще всего обрабатываемые матрицы содержат от 10000 до 40000 строк и от 12 до 40 столбцов. Соответственно, пересылать их между клиентом и сервером в *JSON*-формате слишком долго.

Как было сказано в обзоре, технология *Protocol Buffers* позволяет лучше сериализовать данные, чтобы уменьшить размер пересылаемого пакета.

К сожалению, *Google Developers* официально поддерживают только *Java*, *Python*, *C++*, *Go*, *Objective-C*, *Ruby*, *JavaNano* и *C#*. Для *JavaScript* сообщество создает поддержку самостоятельно. После анализа существующих решений, было решено выбрать библиотеку *ProtoBuf.js* [24].

Данная библиотека реализует класс *Builder*, который компилируется из \*.proto-файлов и позволяет получить доступ к сериализованным данным. С помощью экземпляра данного класса можно закодировать соответствующий *JSON*-объект в *Uint8Array*, чтобы после переслать его в сжатом виде на сервер.

Для преобразования экземпляров классов *Dataset* и *SlicedDatasetView*, которые были представлены в обзоре приложения *morpheus.js*, в сериализованный *Uint8Array* используется протокол, описанный в приложении А. Данный протокол используется также в *R*-пакете *protolite* [25], с помощью которого данные десериализуются на сервере.

Таким образом в класс *DatasetUtil* была добавлена утилита, сериализующая *Dataset*, которая используется перед отправкой данных

на сервер, и десериализующая результат работы *R*-функций, если те сохраняют результат в бинарном файле.

#### 2.2.2.2. Сериализация данных со стороны сервера — *protolite*

Внутри *R*-функций нет необходимости вручную явно десериализовать входные данные, так как *OpenCPU* автоматически разбирает аргументы до входа в функцию.

Однако, если функция возвращает матрицы больших размеров, используется *R*-пакет *protolite* [25] для сериализации результата работы, который после сериализованный записывается в бинарный файл. В дальнейшем этот файл считывается клиентом из директории возвращенной временной сессии.

### 2.3. Поддержка *ExpressionSet*

Как было сказано ранее, а также продемонстрировано на схеме 2, в ходе работы приложения постоянно поддерживается соответствие между экземпляром класса *Dataset* на клиенте и экземпляром класса *ExpressionSet* на сервере. В данном разделе будет подробнее разобрана реализация данного соответствие.

#### 2.3.1. Поддержка *ExpressionSet* на стороне клиента

В *phantasus.js* в *Dataset* добавлено дополнительное поле *esSession*, в котором находится объект класса *Promise* для асинхронного обновления ключа сессии в этом поле.

При загрузке или обновлении *Dataset* осуществляется следующий ряд действий:

- а) в поле `dataset.esSession` записывается экземпляр класса *Promise*, который позволяет продолжать загрузку данных в фоновом режиме, а также ждать, когда данные будут обработаны прежде чем запускать функции использующие *ExpressionSet* в качестве аргумента (*pcaPlot*, *kmeans*, *limma*). При создании *Promise* в аргументах указывается две функции: *reject* и *resolve*;
- б) актуальное содержимое экземпляра класса *Dataset* вместе с матрицей и аннотацией сериализуется в *ProtoBuf* по протоколу, описанному в приложении А;

- в) с помощью *opencpu.js* отправляется *RPC* за функцией `createES` с аргументом в виде сериализованных данных;
- г) данные поступают на сервер, десериализуются там автоматически и функция `createES` создает `ExpressionSet`, являющийся копией `Dataset` из клиента;
- д) функция `createES` объявляет данный `ExpressionSet` глобальной переменной, таким образом имеется доступ к этому объекту по *API-entrypoint* `/ocpu/tmp/key/R/es`;
- е) по завершении *RPC* получает ключ временной сессии, содержащий созданный `ExpressionSet` и завершает `Promise` с `resolve(session)`;
- ж) если во время одного из этапов произошла ошибка, то `Promise` завершается с `reject(error)` с текстом ошибки.

### 2.3.2. Создание `ExpressionSet` из внешних данных на стороне сервера

В начале работы с *phantasus* необходимо загрузить данные. Если данные загружены из файла, то они будут сначала обработаны на клиенте, а после пересланы на сервер для создания `ExpressionSet` из них с помощью кода на листинге 3.

Функция `createES` принимает следующие аргументы:

- `data` — непосредственно матрица экспрессии;
- `pData` — аннотация к образцам;
- `varLabels` — названия характеристик описания образцов;
- `fData` — аннотация к генам;
- `fvarLabels` — названия характеристик описания генов.

По завершении функция отправляет `es` в глобальные переменные, чтобы созданный `ExpressionSet` был доступен по адресу: `/ocpu/tmp/key/R/es`. Таким образом, получив ключ данной сессии, можно иметь доступ и к `ExpressionSet`, находящемуся в ней.

Ключ сессии обновляется каждый раз при изменении `Dataset` в *phantasus.js*. Чаще всего изменения происходят в результате работы одного из следующих инструментов: `Adjust`, `Collapse`, `new HeatMap`, `Transpose`. Изменные данные, точно так же, как и новые, пересылаются на сервер и ключ сессии обновляется в поле `esSession` в `Dataset`.

```

1 createES <- function(data, pData, varLabels, fData, fvarLabels) {
2   exprs <- t(data)
3   phenoData <- AnnotatedDataFrame(data.frame(pData))
4   varLabels(phenoData) <- varLabels
5
6   featureData <- AnnotatedDataFrame(data.frame(fData))
7   varLabels(featureData) <- fvarLabels
8
9   es <- ExpressionSet(assayData = exprs, phenoData=phenoData, featureData =
   featureData)
10  assign("es", es, envir = parent.frame())
11  es
12 }

```

Листинг 3 – Функция создания ExpressionSet из исходных данных

## 2.4. Способы визуализации

В данном разделе будут показаны достоинства и недостатки различных способов визуализации, а также применимость в различных ситуациях.

### 2.4.1. Отрисовка графиков на стороне сервера

Первый вариант визуализации графиков и схем состоит в отрисовке их там же, где и происходит вычисление всех необходимых для них данных, то есть на сервере внутри *R*-функции. Данный способ подразумевает отрисовку и сохранение изображений в виде *png* или *svg*-файла, который сохраняется внутри временной *OpenCPU*-сессии. Клиент после забирает из сессии изображение и показывает в графическом интерфейсе приложения.

Достоинства:

- возможность пользоваться проверенными *R*-пакетами для визуализации, например, *ggplot2* [26];
- файл можно переиспользовать при необходимости, нужно знать только ключ сессии, где он находится.

Однако таким образом можно использовать только статичные изображения.

### 2.4.2. Визуализация на стороне клиента

Для отображения интерактивных графиков удобно использовать библиотеку *plotly.js* [27], которая предоставляет *API*, где описание графика строится в *JSON*-формате. Таким образом можно строить интерак-

тивные изображения, переложить всю работу по визуализации с сервера на клиент.

Именно этот способ на данный момент используется в проекте, подробнее об инструменте будет рассказано в главе 3.

### Выводы по главе 2

В данной главе были рассмотрены основные составляющие проекта:

- *phantasus.js* — расширенный *morpheus.js*;
- *R*-пакет *phantasus* — *R*-пакет, содержащий в себе серверные реализации всех добавленных методов и инструментов.

Также были представлены следующие подробности архитектуры проекта:

- общая схема всего проекта, где показаны компоненты и связи между ними;
- способ взаимодействия между компонентами с помощью *OpenCPU*;
- сериализация данных в *ProtoBuf* на стороне клиента и на стороне сервера;
- поддержка *ExpressionSet*;
- способы визуализации.



### ГЛАВА 3. РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ

В этой главе будут подробно описаны реализованные методы анализа в проекте *phantasus*, их реализация на стороне сервера и на стороне клиента, будет рассказано о способах запуска приложения и другие инфраструктурные подробности.

#### 3.1. Реализованные методы анализа экспрессии

В ходе работы над проектом были реализованы следующие функции и методы анализа экспрессии:

- `loadGEO` — загрузка и визуализация данных из *Gene Expression Omnibus*;
- `rsaPlot` — реализация метода главных компонент и визуализация результата;
- `kmeans` — реализация кластеризации методом *kmeans*;
- `limmaAnalysis` — анализ дифференциальной экспрессии для сравнения образцов.

В этом разделе будут один за другим описаны все вышеперечисленные методы.

##### 3.1.1. Загрузка данных из GEO

В разделе 1.6.1 обзора были описаны форматы данных в репозитории *Gene Expression Omnibus*. В *phantasus* загрузка данных из *GEO* осуществляется следующим образом:

- а) функция `loadGEO` принимает на вход идентификатор *GEO*;
- б) в зависимости от его вида (*GSE* или *GDS*) запускаются дополнительные функции (`getGSE` на листинге 4 и `getGDS` на листинге 5);
- в) в каждой из двух функций с помощью `GEOquery::getGEO` загружаются данные с аннотацией (или подгружаются из кэша, если он указан или если их уже загружали);
- г) результат обрабатывается, создается `ExpressionSet` и отправляется в глобальные переменные;
- д) в файл записываются сериализованные в *ProtoBuf* данные, в том же формате, что и при создании `ExpressionSet` из внешних данных (смотри раздел 2.3.2), которые после считывает и обрабатывает клиент.

```

1 getGSE <- function(name, destdir = tempdir()) {
2   es <- getGEO(name, AnnotGPL = T, destdir = destdir)[[1]]
3   featureData(es) <- featureData(es)[,grepl("symbol", fvarLabels(es), ignore.case
4     = T)]
5   phenoData(es) <- phenoData(es)[,grepl("characteristics", varLabels(es), ignore.
6     case = T)
7     | (varLabels(es) %in% c("title", "id", "geo_
8       accession"))]
9   chr <- varLabels(es)[grepl("characteristics", varLabels(es), ignore.case = T)]
10  take <- function(x, n) {
11    supply(x, function(x) { x[[n]] })
12  }
13  rename <- function(prevName, x) {
14    splitted <- strsplit(x, ": ")
15    sumlength <- sum(sapply(as.vector(splitted), length))
16    if (sumlength != 2 * length(x)) {
17      return(list(name = prevName, x = x))
18    }
19    splittedFirst <- unique(take(splitted, 1))
20    if (length(splittedFirst) == 1) {
21      res = list(name = splittedFirst[1], x = take(splitted, 2))
22    }
23    else {
24      res = list(name = prevName, x = x)
25    }
26  }
27  renamed <- lapply(chr, function(x) { rename(x, as.vector(pData(es)[,x])) })
28  phenoData(es) <- phenoData(es)[, !(varLabels(es) %in% chr)]
29  pData(es)[,take(renamed,1)] <- take(renamed,2)
30  es
31 }

```

Листинг 4 – Загрузка данных типа GSE из Gene Expression Omnibus

Основной код загрузки и дополнительные утилиты к нему можно увидеть на листинге 6.

Такая реализация загрузки данных из *GEO* позволяет избежать проблемы *cross-origin request*, которая служила препятствием для загрузки из *GEO* в исходном приложении *Morpheus*.

### 3.1.2. Метод главных компонент и визуализация его результата

Данный инструмент предназначен для построения графиков в соответствии с методом главных компонент. В качестве аргументов на вход к инструменту подается:

- номера образцов для сравнения;
- категориальная аннотация для различения точек по цвету (если не указана, то стандартный цвет);

```

1 getGDS <- function(name, destdir = tempdir()) {
2   l <- getGEO(name, destdir = destdir)
3   table <- slot(l, 'dataTable') # extracting all useful information on dataset
4   data <- Table(table) # extracting table ID_REF | IDENTIFIER/SAMPLE | SAMPLE1 |
5   ...
6   columnsMeta <- Columns(table) # phenoData
7   sampleNames <- as.vector(columnsMeta[["sample"]])
8   rownames <- as.vector(data[["ID_REF"]])
9   symbol <- as.vector(data[["IDENTIFIER"]])
10  data <- data[, sampleNames] # expression data
11  exprs <- as.matrix(data)
12  row.names(exprs) <- rownames
13  row.names(columnsMeta) <- sampleNames
14  # columnsMeta <- columnsMeta[,!(colnames(columnsMeta) %in% c('sample'))]
15  pData <- AnnotatedDataFrame(data.frame(columnsMeta, check.names = F))
16  fData <- data.frame(matrix(symbol, nrow(exprs), 1));
17  colnames(fData) <- "symbol"
18  fData <- AnnotatedDataFrame(fData)
19  featureNames(fData) <- rownames
20  ExpressionSet(assayData = exprs, phenoData = pData, featureData = fData)
}

```

Листинг 5 – Загрузка данных типа GDS из Gene Expression Omnibus

- числовая аннотация для различения точек по размеру (если не указана, то стандартный размер);
- аннотация для подписей к точкам (если не указана, то без подписи);
- функция замены NA в данных при вычислении матрицы PCA (mean или median).

Далее по алгоритму, описанному в разделе 2.2.1.1, на *OpenCPU*-сервер отправляется *RPC*-вызов с аргументами: ключ сессии, содержащий актуальный *ExpressionSet*, и функция замены NA.

Данные аргументы приходят на вход к функции *rsaPlot*, реализованной в *R*-пакете *phantasus*, код которой можно увидеть на листинге 7. Предварительно все NA-значения заменяются в соответствии с переданной функцией, если этого не сделать, то дальнейшие вычисления будут невозможны. Матрица экспрессии из входного *ExpressionSet* передается в стандартную функцию *prcomp* из *R*-пакета *stats* [28], которая и вычисляет результирующую матрицу.

На клиент в *JSON*-формате приходит вычисленная матрица PCA.

После, по дополнительным аргументам и вычисленной матрице, строится интерактивный график с помощью *plotly.js*, пример которого можно увидеть на рисунке 3.

```

1 loadGEO <- function(name, type = NA) {
2   es <- getES(name, type, destdir = "/var/phantasus/cache")
3   assign("es", es, envir = parent.frame())
4   data <- as.matrix(exprs(es)); colnames(data) <- NULL; row.names(data) <- NULL
5
6   pdata <- as.matrix(pData(es)); colnames(pdata) <- NULL; row.names(pdata) <-
   NULL
7
8   participants <- colnames(es)
9   rownames <- rownames(es)
10
11  fdata <- as.matrix(fData(es))
12  colnames(fdata) <- NULL
13  row.names(fdata) <- NULL
14
15  res <- list(data = data, pdata = pdata,
16             fdata = fdata, rownames = rownames,
17             colMetaNames = varLabels(phenoData(es)),
18             rowMetaNames = varLabels(featureData(es)))
19
20  f <- tempfile(pattern = "gse", tmpdir = getwd(), fileext = ".bin")
21  writeBin(protolite::serialize_pb(res), f)
22  f
23 }
24 getES <- function(name, type = NA, destdir = tempdir()) {
25   if (is.na(type)) {
26     type = substr(name, 1, 3)
27   }
28   if (type == 'GSE') {
29     es <- getGSE(name, destdir)
30   }
31   else if (type == 'GDS') {
32     es <- getGDS(name, destdir)
33   }
34   else {
35     stop("Incorrect name or type of the dataset")
36   }
37   es
38 }

```

Листинг 6 – Загрузка данных из Gene Expression Omnibus

### 3.1.3. Кластеризация методом kmeans

Этот инструмент осуществляет разбиение генов на указанное пользователем число кластеров по алгоритму *kmeans*.

На клиенте в инструменте KmeansTool, который показан на рисунке 4, считываются следующие аргументы:

- количество кластеров, на которые нужно разбить данные;
- функция замены NA в данных.

Данные аргументы и ключ сессии актуального ExpressionSet отправляются на сервер в соответствующую функцию *kmeans*, код которой можно увидеть на листинге 8. Также как и в *rsaPlot*, здесь сначала

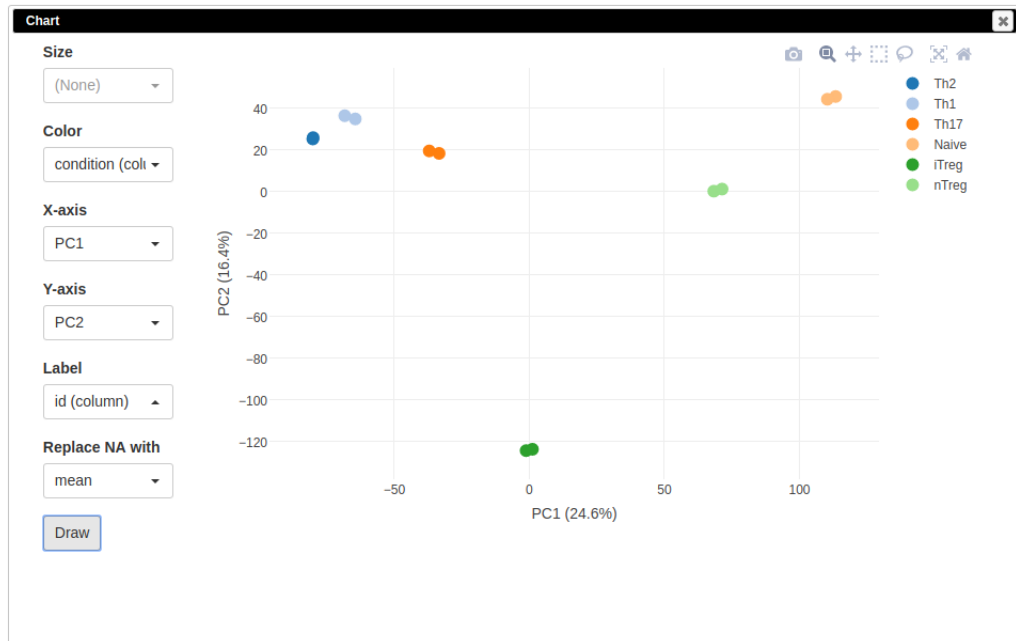


Рисунок 3 – Инструмент PcaPlotTool и отрисованный график по данным GSE14308

```

1  pcaPlot <- function(es, rows=c(), columns = c(), replacena = "mean") {
2  rows <- getIndicesVector(rows, nrow(exprs(es)))
3  columns <- getIndicesVector(columns, ncol(exprs(es)))
4  data <- data.frame(exprs(es))[rows, columns]
5
6  ind <- which(is.na(data), arr.ind = T)
7  if (nrow(ind) > 0) {
8    data[ind] <- apply(data, 1, replacena, na.rm = T)[ind[,1]]
9  }
10 ind1 <- which(!is.nan(as.matrix(data)), arr.ind = T)
11 left.rows <- unique(ind1[, "row"])
12 data <- data[left.rows,]
13 data <- t(data)
14
15 pca <- prcomp(data)
16 explained <- (pca$sdev)^2 / sum(pca$sdev^2)
17
18 xs <- sprintf("PC%s", seq_along(explained))
19 xlabs <- sprintf("%s (%.1f%%)", xs, explained * 100)
20
21 pca.res <- as.matrix(pca$x); colnames(pca.res) <- NULL; row.names(pca.res) <-
  NULL
22 return(jsonlite::toJSON(list(pca = t(pca.res), xlabs=xlabs)))
23 }

```

Листинг 7 – Вычисление матрицы главных компонент

ла заменяются NA на соответствующие данной функции значения, так как в этом случае NA-значения мешают вычислить среднее среди векторов. После данные отправляются в стандартную функцию kmeans па-

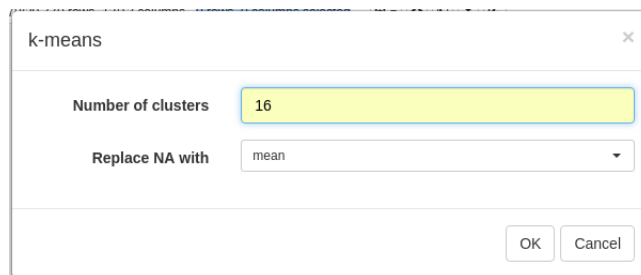


Рисунок 4 – Графический интерфейс инструмента KmeansTool

кета *stats* [28]. Результат возвращается как список соответствия каждого гена определенному кластеру, который приходит на клиент в *JSON*-формате и отрисовывается как новая цветовая аннотация к строкам. Пример такой аннотации можно увидеть на рисунке 5.

```

1 kmeans <- function(es, columns = c(), rows = c(), k, replacena = "mean") {
2   assertthat::assert_that(k > 0)
3
4   rows <- getIndicesVector(rows, nrow(exprs(es)))
5   columns <- getIndicesVector(columns, ncol(exprs(es)))
6   data <- replacenas(data.frame(exprs(es))[rows, columns], replacena)
7
8   data <- t(scale(t(data)))
9   while (sum(is.na(data)) > 0) {
10    data <- replacenas()
11    data <- t(scale(t(data)))
12  }
13
14  km <- stats::kmeans(data, k)
15  res <- data.frame(row.names = row.names(exprs(es)))
16  res[["cluster"]] <- NA
17  res[names(km$cluster), "cluster"] <- as.vector(km$cluster)
18  return(toJSON(as.vector(km$cluster)))
19 }

```

Листинг 8 – Кластеризация методом kmeans

### 3.1.4. Анализ дифференциальной экспрессии

Инструмент предназначен для анализа дифференциальной экспрессии: экспрессия генов сравнивается в двух группах образцов, и вычисляются несколько статистических характеристик, показывающих, насколько случайны различия этих групп.

На клиенте, в инструменте, показанном на рисунке 6, осуществляется получение следующих аргументов:

- какие аннотации образцов участвуют в сравнении;
- какая комбинация значений указанных выше аннотаций обозначает класс A;

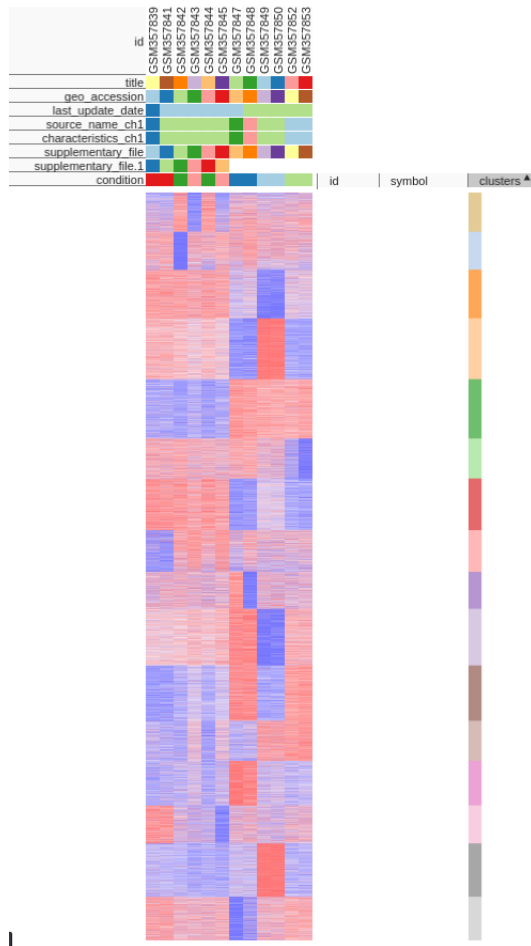


Рисунок 5 – Результат работы инструмента KmeansTool на данных GSE14308

— аналогично для класса В.

Далее происходит подготовка аргументов к отправке на сервер: образцы разбиваются по выбранным аннотациям на три группы: А, В и не участвующие в сравнении. Список соответствия образцов классам и ключ сессии, содержащей актуальный ExpressionSet, отправляются на сервер.

Аргументы приходят в функцию `limmaAnalysis`, код которой можно увидеть на листинге 9. Прежде чем использовать функцию `de` (*differential expression*), реализованную в пакете *limma* [29], которая помогает увидеть, насколько случайны различия между образцами, гены которых находятся в разных условиях, необходимо дополнить аннотацию образцов списком, содержащим идентификаторы классов сравнения.

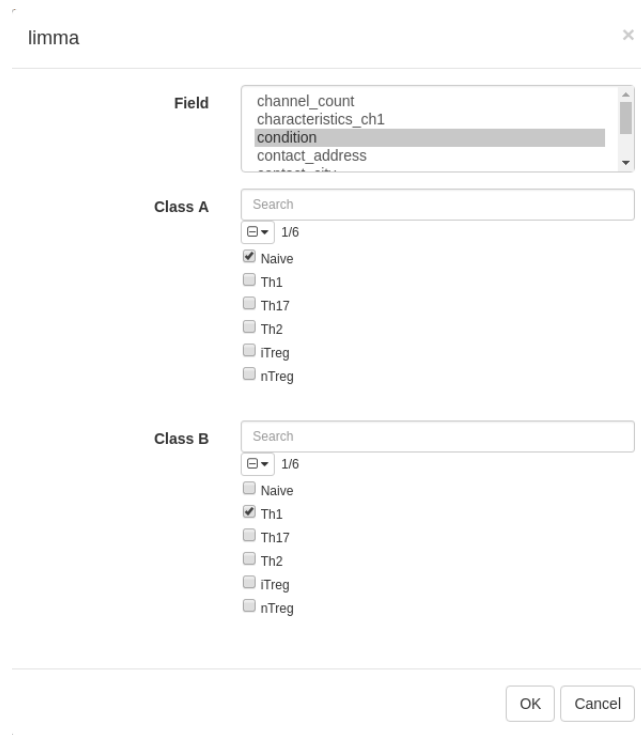


Рисунок 6 – Графический интерфейс инструмента LimmaTool

Построенный дизайн сравнения передается в функцию `de`, которая возвращает матрицу статистических характеристик к каждому гену. Эта матрица далее сериализуется в *ProtoBuf* и записывается в файл.

Клиент, получив ключ временной *OpenCPU*-сессии, читает файл с сериализованной в *ProtoBuf* матрицей результатов, которые с помощью *protobuf.js* разбираются и после отрисовываются в виде аннотации к строкам. Результат работы можно увидеть на рисунке 7.

## 3.2. Инфраструктура проекта phantasus

### 3.2.1. Структура git-репозитория

Как следует из главы об архитектуре проекта, проект состоит из двух составляющих:

- *phantasus.js* — *fork* репозитория *morpheus.js* [5];
- *phantasus* — репозиторий для *R*-пакета.

Внутри репозитория *phantasus* находится подмодуль для репозитория *phantasus.js*. Соответственно, чтобы загрузить целиком весь проект достаточно вызвать команду из листинга 10.



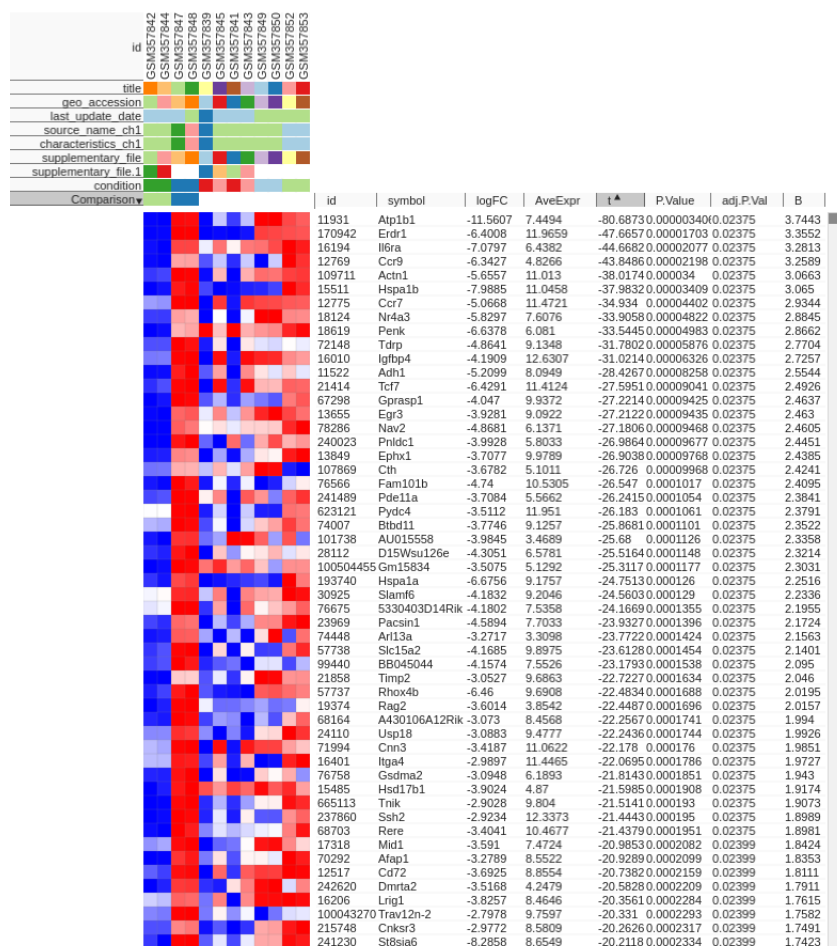


Рисунок 7 – Результат работы инструмента LimmaTool на данных GSE14308

### 3.2.2. Запуск приложения

На данный момент существует два варианта запуска приложения *phantasus*. В данном разделе будут описаны оба.

#### 3.2.2.1. Единый R-пакет *phantasus*

Так как проект теперь существует в виде единого *git*-репозитория, его легко можно использовать как полноценный R-пакет, содержащий в себе в том числе и файлы для веб-приложения. С помощью функции, представленной на листинге 11, можно запускать веб-приложение *phantasus* непосредственно из R.

Соответственно, чтобы запустить приложение, необходимо выполнить код, представленный на листинге 12 и перейти на <http://localhost:8000> в браузере.

```

1 limmaAnalysis <- function(es, rows = c(), columns = c(), fieldValues) {
2   assertthat::assert_that(length(columns) == length(fieldValues) || length(
3     columns) == 0)
4   rows <- getIndicesVector(rows, nrow(exprs(es)))
5   columns <- getIndicesVector(columns, ncol(exprs(es)))
6   fieldName <- "Comparison"
7   fieldValues <- replace(fieldValues, fieldValues == '', NA)
8   new.pdata <- pData(es)[columns,]
9   new.pdata[[fieldName]] <- as.factor(fieldValues)
10  new.pdata <- new.pdata[!is.na(new.pdata[[fieldName]]),]
11  new.sampleNames <- row.names(new.pdata)
12  es.copy <- es[rows, new.sampleNames]
13  pData(es.copy) <- new.pdata
14  fData(es.copy) <- data.frame(row.names=row.names(es.copy))
15  es.design <- model.matrix(~0 + Comparison, data = pData(es.copy))
16  colnames(es.design) <- gsub(pattern = fieldName,
17    replacement = '',
18    x = make.names(colnames(es.design)))
19  fit <- lmFit(es.copy, es.design)
20  fit2 <- contrasts.fit(fit, makeContrasts(B - A,
21    levels=es.design))
22  fit2 <- eBayes(fit2)
23  de <- topTable(fit2, adjust.method="BH", number=Inf)
24  de <- de[row.names(fData(es.copy)),]
25  f <- tempfile(pattern = "de", tmpdir = getwd(), fileext = ".bin")
26  writeBin(protolite::serialize_pb(as.list(de)), f)
27 }

```

Листинг 9 – Реализация анализа дифференциальной экспрессии в R-пакете phantastus

```

1 git clone --recursive https://github.com/ctlab/phantastus.git

```

Листинг 10 – Клонирование репозитория проекта phantastus

### 3.2.2.2. Docker-образ phantastus

На [hub.docker.com](https://hub.docker.com) существует автоматический репозиторий, привязанный к *git*-репозиторию *phantastus*. Для каждой перекомпиляции он использует Dockerfile с листинга Б, расположенный в репозитории.

На данный момент существуют две ветки Docker-образа:

- *master* — компиляция происходит из *master*-веток составляющих проекта, чаще всего эти скомпилированные образы стабильны и отправляются в открытый доступ для использования;
- *develop* — компиляция происходит из *develop*-веток составляющих проекта, эти образы используются для тестирования всего прило-

```

1 servePhantassus <- function(host, port,
2                             staticRoot=system.file("www/phantassus.js", package="
3                             phantassus"),
4                             cacheDir=tempdir()) {
5   options(phantassusCacheDir=cacheDir)
6   app <-
7     Rook::URLMap$new(
8       "/ocpu"=opencpu:::rookhandler("/ocpu"),
9       "/?"=Rook::Static$new(
10        urls = c('/'),
11        root = staticRoot
12      ))
13   httpuv::runServer(host,
14                     port,
15                     app=app)
16 }
17

```

Листинг 11 – Функция для запуска приложения из R

```

1 library(phantassus)
2 example(servePhantassus)

```

Листинг 12 – Запуск приложения в качестве R-пакета

жения в целом, тестирования нового функционала и не предназна-  
чены для использования на серверах.

Чтобы загрузить *Docker*-образ, нужно воспользоваться командой с  
листинга 13

```

1 docker pull dzenkova/phantassus

```

Листинг 13 – Загрузка Docker-образа phantassus

Для запуска *Docker*-контейнера, необходимо воспользоваться ко-  
мандой с листинге 14.

```

1 docker run -t -d -p "8000:80" -v /mnt/data/phantassus-cache:/var/phantassus/
  cache dzenkova/phantassus

```

Листинг 14 – Запуск Docker-контейнера

### 3.2.3. Кэш для данных из GEO

Независимо от способа запуска, данные, загруженные из *GEO*, кэ-  
шируются в определенной папке (сейчас это неизменяемое значение:  
/var/phantassus/cache), чтобы не было необходимости перескачи-  
вать их заново.

### 3.3. Настройка с помощью Apache

В обоих из представленных вариантах запуска, приложение запускается в формате *single-user*, соответственно, настройка деятельности приложения в формате *multi-user* осуществляется возможностями *Apache*. Конфигурационный файл можно увидеть в приложении В.

#### 3.3.1. Переадресация OpenCPU-сервера

Как было сказано в разделе 1.3.2.1 обзора, *OpenCPU*-сервер работает быстрее в однопользовательском режиме. Сервер запускается на определенном порту (например, 8001), после для корректной работы приложения, происходит переадресация запросов с /осри на //localhost:8001/осри и наоборот.

#### 3.3.2. Балансировщик для multi-user соединения

Из-за того, что используется однопользовательский *OpenCPU*-сервер, несколько людей, использующих веб-приложение *phantasus* одновременно, вынуждены ждать, пока закончится запрос для одного.

Чтобы избежать такой ситуации, запускается четыре одинаковых экземпляра *OpenCPU*-сервера и с помощью *Apache*-балансировщика можно получать доступ к *R*-серверу параллельно.

### Выводы по главе 3

В данной главе были рассмотрены реализованные инструменты и методы анализа:

- загрузка данных из *Gene Expression Omnibus*;
- метод главных компонент и его визуализация;
- кластеризация методом *kmeans*;
- анализ дифференциальной экспрессии.

Также были описаны технические подробности реализации веб-приложения: инструкции для запуска, варианты использования и детали настройки веб-приложения на сервере.

## ЗАКЛЮЧЕНИЕ

В ходе данной работы веб-приложение *Morpheus* было расширено дополнительным функционалом с помощью реализованных в *R* и в *Bioconductor* статистических методов и методов анализа экспрессии генов. Веб-приложение *phantasus* было выпущено в открытый доступ и им можно пользоваться по адресу <https://artyomovlab.wustl.edu/phantasus>, либо же запускать самостоятельно в виде *R*-пакета или *Docker*-контейнера, как это было описано в главе 3. Проект *phantasus* используется в:

- лаборатории Максима Артемова в Washington University in St. Louis;
- лаборатории Laurent Yvan-Charvet в Université Nice Sophia Antipolis.

Также проект был продемонстрирован на семинаре по системной биологии в Сиднее (10-13 апреля 2017) и в Санкт-Петербурге (14-19 мая 2017).

Полученное веб-приложение соответствует всем поставленным требованиям.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 *R Core Team*. R: A Language and Environment for Statistical Computing / R Foundation for Statistical Computing. — Vienna, Austria, 2014. — URL: <http://www.R-project.org/>.
- 2 Orchestrating high-throughput genomic analysis with Bioconductor / Huber [и др.] // Nature Methods. — 2015. — Т. 12, № 2. — С. 115–121. — URL: <http://www.nature.com/nmeth/journal/v12/n2/full/nmeth.3252.html>.
- 3 *Falcon S., Morgan M., Gentleman R.* An Introduction to Bioconductor's ExpressionSet Class. — 2006. — URL: <https://www.bioconductor.org/packages/devel/bioc/vignettes/Biobase/inst/doc/ExpressionSetIntroduction.pdf>. [Электронный ресурс].
- 4 *Gould J.* GENE-E. — URL: <http://www.broadinstitute.org/cancer/software/GENE-E/>. [Электронный ресурс].
- 5 *Gould J.* morpheus.js. — URL: <https://clue.io/morpheus.js/>. [Электронный ресурс].
- 6 Mutation Annotation Format. — URL: <https://wiki.nci.nih.gov/display/TCGA/Mutation+Annotation+Format+%28MAF%29+Specification/>. [Электронный ресурс].
- 7 GCT. — URL: <http://software.broadinstitute.org/cancer/software/genepattern/file-formats-guide#GCT/>. [Электронный ресурс].
- 8 Gene Matrix Transposed file format. — URL: [http://software.broadinstitute.org/cancer/software/gsea/wiki/index.php/Data\\_formats#GMT:\\_Gene\\_Matrix\\_Transposed\\_file\\_format\\_.28.2A.gmt.29/](http://software.broadinstitute.org/cancer/software/gsea/wiki/index.php/Data_formats#GMT:_Gene_Matrix_Transposed_file_format_.28.2A.gmt.29/). [Электронный ресурс].
- 9 The Cancer Genome Atlas Pan-Cancer analysis project / T. C. G. A. R. Network [и др.] // Nat Genet. — 2013. — Окт. — Т. 45, № 10. — С. 1113–1120. — ISSN 1061-4036. — URL: <http://dx.doi.org/10.1038/ng.2764> ; Commentary.

- 10 *Зайцев К.* Проектирование и реализация веб-сервиса для анализа экспрессии генов. — 2015. — URL: <http://is.ifmo.ru/diploma-theses/2015/bachelor/zaycev/zaycev.pdf>.
- 11 shiny: Web Application Framework for R / W. Chang [и др.]. — 2017. — URL: <https://CRAN.R-project.org/package=shiny>; R package version 1.0.3.
- 12 Hypertext transfer protocol–HTTP/1.1: тех. отч. / R. Fielding [и др.]. — 1999.
- 13 *Ooms J.* The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns // arXiv:1406.4806 [stat.CO]. — 2014. — URL: <http://arxiv.org/abs/1406.4806>.
- 14 *Horner J.* rApache: Web application development with R and Apache. — 2013. — URL: <http://www.rapache.net/>.
- 15 *Holdener III A. T.* Ajax: The Definitive Guide. — First. — O'Reilly, 2008. — ISBN 9780596528386.
- 16 *Merkel D.* Docker: Lightweight Linux Containers for Consistent Development and Deployment // Linux J. — Houston, TX, 2014. — Март. — Т. 2014, № 239. — ISSN 1075-3583. — URL: <http://dl.acm.org/citation.cfm?id=2600239.2600241>.
- 17 Docker Hub. — URL: <https://hub.docker.com/>. [Электронный ресурс].
- 18 GitHub. — URL: <https://github.com/>. [Электронный ресурс].
- 19 Bitbucket. — URL: <https://bitbucket.org/>. [Электронный ресурс].
- 20 Apache HTTP Server Project. — URL: <https://httpd.apache.org>. [Электронный ресурс].
- 21 *Varda K.* Protocol buffers: Google's data interchange format // Google Open Source Blog, Available at least as early as Jul. — 2008.
- 22 *Edgar R., Domrachev M., Lash A. E.* Gene Expression Omnibus: NCBI gene expression and hybridization array data repository // Nucleic acids research. — 2002. — Т. 30, № 1. — С. 207–210.

- 23 *Davis S., Meltzer P.* GEOquery: a bridge between the Gene Expression Omnibus (GEO) and BioConductor // Bioinformatics. — 2007. — T. 14. — C. 1846–1847.
- 24 *dcode.* Protocol Buffers for JavaScript (& TypeScript). — URL: <https://github.com/dcodeIO/ProtoBuf.js/>. [Электронный ресурс].
- 25 *Ooms J.* protolite: Fast and Simple Object Serialization to Protocol Buffers. — 2017. — URL: <https://CRAN.R-project.org/package=protolite> ; R package version 1.6.
- 26 *Wickham H.* ggplot2: Elegant Graphics for Data Analysis. — Springer-Verlag New York, 2009. — ISBN 978-0-387-98140-6. — URL: <http://ggplot2.org>.
- 27 *Inc. P. T.* Collaborative data science. — 2015. — URL: <https://plot.ly>.
- 28 *R Core Team.* R: A Language and Environment for Statistical Computing / R Foundation for Statistical Computing. — Vienna, Austria, 2017. — URL: <https://www.R-project.org/>.
- 29 *limma* powers differential expression analyses for RNA-sequencing and microarray studies / M. E. Ritchie [и др.] // Nucleic Acids Research. — 2015. — T. 43, № 7. — e47.



## ПРИЛОЖЕНИЕ А. ПРОТОКОЛ СЕРИАЛИЗАЦИИ В PROTOBUF

R-пакет protolite использует стандартный протокол сериализации, представленный на листинге А. Этот же протокол было решено использовать и при сериализации на клиенте для однообразия и для корректного разбора сообщений как на клиенте от сервера, так и на сервере от клиента.

```

1 package rexp;
2
3 option java_package = "org.godhuli.rhipe";
4 option java_outer_classname = "REXPProtos";
5
6 message REXP {
7     enum RClass {
8         STRING = 0;
9         RAW = 1;
10        REAL = 2;
11        COMPLEX = 3;
12        INTEGER = 4;
13        LIST = 5;
14        LOGICAL = 6;
15        NULLTYPE = 7;
16        NATIVE = 8;
17    }
18    enum RBOOLEAN {
19        F=0;
20        T=1;
21        NA=2;
22    }
23
24    required RClass rclass = 1;
25    repeated double realValue = 2 [packed=true];
26    repeated sint32 intValue = 3 [packed=true];
27    repeated RBOOLEAN booleanValue = 4;
28    repeated STRING stringValue = 5;
29
30    optional bytes rawValue = 6;
31    repeated CMPLX complexValue = 7;
32    repeated REXP rexpValue = 8;
33
34    repeated string attrName = 11;
35    repeated REXP attrValue = 12;
36    optional bytes nativeValue = 13;
37 }
38 message STRING {
39     optional string strval = 1;
40     optional bool isNA = 2 [default=false];
41 }
42 message CMPLX {
43     optional double real = 1 [default=0];
44     required double imag = 2;
45 }

```

## ПРИЛОЖЕНИЕ Б. DOCKERFILE

```

1 FROM ubuntu
2 RUN apt-get -y update && apt-get -y dist-upgrade && apt-get -y install \
3     software-properties-common \
4     git \
5     libcairo2-dev \
6     libxt-dev \
7     libssl-dev \
8     libssh2-1-dev \
9     libcurl4-openssl-dev \
10    apache2 \
11    locales && \
12    apt-add-repository -y ppa:opencpu/opencpu-1.6 && \
13    apt-get -y update && apt-get -y install opencpu-lib
14
15 RUN touch /etc/apache2/sites-available/opencpu2.conf
16 RUN printf "ProxyPass /ocpu/ http://localhost:8001/ocpu/\nProxyPassReverse /ocpu/\n" >> /etc/apache2/sites-available/opencpu2.conf
17 RUN a2ensite opencpu2
18
19 RUN sh -c 'echo "deb http://cran.rstudio.com/bin/linux/ubuntu trusty/" >> /etc/apt/sources.list'
20 RUN gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
21 RUN gpg -a --export E084DAB9 | apt-key add -
22 RUN apt-get -y update && apt-get -y install \
23     r-base \
24     libprotobuf-dev \
25     protobuf-compiler \
26     r-cran-xml
27
28 RUN git clone --recursive https://github.com/ctlab/phantasus /root/phantasus
29 RUN cp -r /root/phantasus/inst/www/phantasus.js /var/www/html/phantasus
30 RUN R -e 'source("https://bioconductor.org/biocLite.R"); install.packages("XML", repo = "http://cran.gis-lab.info"); biocLite("Biobase"); biocLite("limma"); biocLite("org.Mm.eg.db")'
31 RUN R -e 'install.packages("devtools", repo = "http://cran.gis-lab.info"); library(devtools); install_github("hadley/scales"); install_github("assaron/GEOquery"); install("/root/phantasus")'
32
33 RUN a2enmod proxy_http
34 EXPOSE 80
35 EXPOSE 443
36 EXPOSE 8004
37 RUN locale-gen en_US.UTF-8
38 ENV LANG en_US.UTF-8
39 ENV LANGUAGE en_US:en
40 ENV LC_ALL en_US.UTF-8
41
42 RUN mkdir -p /var/phantasus/cache
43 VOLUME ["/var/phantasus/cache"]
44
45 CMD service apache2 start && \
46     R -e 'opencpu::opencpu$start(8001)' && \
47     tail -F /var/log/opencpu/apache_access.log

```

Листинг Б.1 – Dockerfile для Docker-образа веб-приложения phantasus

## ПРИЛОЖЕНИЕ В. КОНФИГУРАЦИОННЫЙ ФАЙЛ APACHE

```

1 ProxyPass /phantasus balancer://phantasus-servers/phantasus
2 ProxyPassReverse /phantasus balancer://phantasus-servers/phantasus
3
4 <Location "/phantasus">
5     Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=
        BALANCER_ROUTE_CHANGED
6 </Location>
7
8 <Proxy balancer://phantasus-servers/>
9     BalancerMember http://localhost:8000/ route=1
10    BalancerMember http://localhost:8001/ route=2
11    BalancerMember http://localhost:8002/ route=3
12    BalancerMember http://localhost:8003/ route=4
13    ProxySet stickysession=ROUTEID
14 </Proxy>
15
16 <Location "/ocpu">
17     Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=
        BALANCER_ROUTE_CHANGED
18 </Location>
19
20 ProxyPassMatch ^/ocpu/(.*/.*)$ balancer://phantasus-servers/ocpu/$1
21 ProxyPassReverse /ocpu/ balancer://phantasus-servers/ocpu

```