SOUND
Statistical Multi-Omics Understanding

Project Deliverable

Project acronym: **SOUND**	GA number: **633974**
Project title: **Statistical Multi-Omics Understanding of Patient Data**	
Funding Scheme: Collaborative Project (H2020-PHC-2014-2015/H2020-PHC-2014-two-stage) Health, novel medical developments	
Project start date: **01 September 2015**	Duration: **36 months**
Project's coordinator: Dr Wolfgang Huber (European Molecular Biology Laboratory, Heidelberg)	

D9.1 Open-source software with updated Renjin and code library of benchmarks and technical report describing most significant performance bottlenecks

Due date of deliverable: Month 18 - 28.02.2017
Actual submission date: 23.02.2017

Organization name of lead contractor for this deliverable: BeDataDriven (BDD)
Organization name of other involved partners: IDMEC, UHZ, EMBL
Personnel involved: *Parham Solaimani Kartalaei, Phil Cheng, André Veríssimo, Susana Vinga, Mitch Levesque, Wolfgang Huber, Maarten-Jan Kallen and Alexander Bertram*

Project co-funded by the European Commission within the H2020 Program (2015-2018)		
Dissemination Level		
PU	Public	x
PP	Restricted to other program participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Background

Renjin [1] is an interpreter for the R language written in Java. This allows running of an R interpreter in the Java Virtual Machine (JVM) of Platform-as-a-Service providers such as *Google App Engine* and *Microsoft Azure App Service*, and seamless integration with existing enterprise Java solutions. The compatibility of Renjin with the GNU R interpreter, which serves as reference implementation of R language, will be improved to allow the use of bioinformatic workflows and Bioconductor packages as part of the SOUND action.

Research progress

As part of task 9.1 we created a (open) library of benchmarks based on real use cases which will be used in measuring performance improvements [2]. Detailed description and analysis of the benchmarks is provided as annex of this document under the title “Renjin Benchmarks Documentation”. We extended our continuous integration system to include automated progress and regression testing. This also includes a dashboard that reports, in real-time, the number of improvements and regressions in package builds, compiling, and tests [3]. We instrumented GNU R and also used CPU profiling tools to analyse the CPU and cache memory usage by GNU R while running the benchmarks.

Overview deliverable progress

We developed the library of benchmarks and included the technical report describing the benchmark performance at CPU and CPU cache memory level (see annex). We collaborated with our partners from the University Hospital Zurich (Mitch Levesque's team), from the Institute of Mechanical Engineering Lisbon (Susana Vinga's team) and from EMBL Heidelberg (Wolfgang Huber's team) to include some of the most often used bioinformatics analysis for clinical cancer data. The R packages included in the benchmarks are heavily used by other SOUND partners. These benchmarks will allow an incremental and meaningful way of the Renjin development, instead of randomly supporting packages.

Future plans

The benchmarks and the profiling environment will help us to identify the typical data access patterns, most efficient data structure designs, and development of implicit optimizations (task 9.3). As part of D9.2, we will extend support for the S4 object system and improve the toolchain for automated compiling of C/C++/Fortran code to Java bytecode, which runs on the JVM.

References

[1] http://www.renjin.org

[2] http://github.com/bedatadriven/renjin-benchmarks

[3] http://packages.renjin.org/qa/dashboard

Annex

Renjin Benchmarks Documentation

Annex

Renjin Benchmarks Documentation

BeDataDriven B.V.

January 15, 2017

CHAPTER

ONE

INTRODUCTION

1.1 Benchmarks for Renjin

The purpose of this project is to collect *benchmarks* for Renjin[1], the R interpreter in the Java Virtual Machine. This document includes a general introduction to each of these benchmarks. For the purpose of this project, a *benchmark* is a program (or a collection of scripts) which performs a particular calculation in R. The reference for these benchmarks is the performance of the reference interpreter for the R programming language, namely GNU R[2]. We sometimes also compare with other interpreters such as TERR[3] and pqR[4].

Benchmarks are used to evaluate a number of metrics:

- can Renjin execute the full benchmark? In other words: is Renjin's interpreter sufficiently compatible with GNU R to complete the full benchmark?
- how does Renjin's performance compare with the performance of the GNU R interpreter?
- where are the computational bottlenecks in the benchmark?
- what is the nature of these bottlenecks and how can this be addressed in Renjin?

For Renjin, we are mostly interested in longer running programs which you typically find in daily situations and less in micro-benchmarks which are commonly used in computer programming. The *Bioinformatics* chapter contains a collection of bio-informatic workflows which represent a wide range of calculations and some of which require large data sets as their input.

1.2 Updates to this document

This is a live document which means that we add and update information as the project progresses. A recent version of the contents of this document can be browsed online at http://bedatadriven.github.io/renjin-benchmarks/.

The source code of this document is available at https://github.com/bedatadriven/renjin-benchmarks.

Benchmarks are executed by our automated build system using a variety of interpreters and the results are reported at http://packages.renjin.org/benchmarks.

[1] http://www.renjin.org
[2] https://www.r-project.org
[3] https://docs.tibco.com/products/tibco-enterprise-runtime-for-r
[4] http://www.pqr-project.org/

CHAPTER

TWO

BIOINFORMATICS

2.1 Introduction

Here we have gathered bioinformatic workflows used in different fields of biology. For each benchmark we have included a brief description containing information about its usage, package dependencies, and sources of input data. Moreover, each benchmark is stored in a seperate folder with defined structure and includes information about the package and dataset versions used.

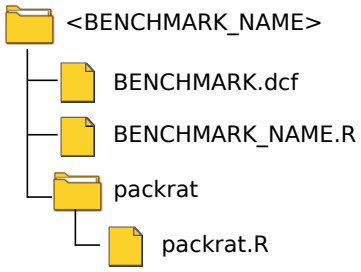


Figure 2.1: The folder name is the name of the benchmark and the file containing the R code. Each folder contains a Debian Control File containing Title and Description tags and for each input file a File, URL, and Hash tags. In addition, there is "packrat.R" file containing the detailed information about each package that is used in the workflow.

So far the benchmarks include analysis of gene expression profiling (array and sequencing based methods), protein analysis, gene set enrichment analysis, network analysis, analysis of clinical data, and visualization of these data.

List of bioinformatic benchmarks included in this repository.

Benchmark name	**Benchmark description**
affy	Analysis of MicroArray data
clinical_eslII	Analysis of clinical data (according to book ESL 2nd edition)
clinical_livercohort	Analysis of clinical data from liver cohort study
generate_count	Generate count tables from RNAseq data
integration_igraph	Integrate different biological data with iGraph
integration_livercohort	Integrate data from liver cohort study with another data source
microarray	Analysis of MicroArray datasets
mutation	Analysis of SNP data
rnaseq_deseq2	Analysis of RNAseq data using DESeq2
rnaseq_preprocess	Preprocessing of RNAseq files to generate count data from FASTA files
rppa	Analysis of Reverse Phase Protein Array (RPPA) data
simulated_geo_matrix	Analysis of simulated gene expression and patient meta data
survival_simple	Simple survival analysis
survival_tcga	Survival analysis using TCGA data
tcga_browser	Modules used in a shiny app to visualize TCGA data

We accept contributions to this list through Pull Requests (PR) after we have validated the requested benchmark for correctness and if the workflow is not covered by existing benchmarks.

2.2 Workflows

2.2.1 Affy

Mutations in DNA that is coding for proteins (genes) or regulatory elements are often the cause of most diseases. Affymatirx arrays allow quantification of specific sequences of DNA or RNA in biological samples such as blood, tissue, or tumors. These arrays give the relative levels of specific transcript (between two samples) so that these can be compared (between eg. healty vs diseased tissues or before vs after treatment).

To do so, DNA sequences unique to specific genes/mutations are printed as small spots on a glass surface (arrays), with each spot containing thousends of copies of a single sequence. The DNA from each sample is than tagged with different colors of flourecent molecules (red or green) and hybredized with the array. This way, DNA from sample can bind to sequences on the surface that are antisense of its own sequence (specific binding). The array undergoes several washing steps to remove the none-specifically bound DNA sequences, which bind loosly. The plate is than scanned and the flourecence intensity for each color at each spot is recorded. This intensity is an indicator of the ammount of DNA bound and probably the levels of that specific DNA present in the sample. The information regarding the specific sequence printed on each spot, control spots, and other array specific information is stored in a CDF format/file. By comparing the intensity of red to green at each spot you can know which sample contained more of that specific transcript.

This workflow is provided by ArrayAnalysis.org[1] and its code has been merged into single R script. It performs array normalization and differential expression analysis, and plots the important Quality Control plots. Dataset used in this workflow is from a study by Ramsey JE et al 2013[2] in which the effects of presence/absence of ZXDC gene before and during differentiation of a white blood cell type is studied. The corresponding data can be downloaded from Gene Expression Omnibus repository using accession number GSE45417. For more information about this workflow please visit ArrayAnalysis.org[3].

Packages and Dependencies

There are 12 packages (mainly affymatrix array analysis related) used in this workflow, which depend on 28 additional packages from CRAN and Bioconductor (dependencies)

Used packages:

- *Bioconductor*: ArrayTools, affy, affycomp, affyPLM, affypdnn, bioDist, simpleaffy, affyQCReport, plier, yaqcaffy
- *CRAN*: gdata, gplots

Package dependencies:

- *Bioconductor*: limma, Biobase, BiocInstaller, BiocGenerics, zlibbioc, preprocessCore, affyio, gcrma, Biostrings, XVector, S4Vectors, IRanges, genefilter, AnnotationDbi, annotate, GenomeInfoDb, affyPLM
- *CRAN*: xtable, KernSmooth, DBI, XML, lattice, RSQLite, RColorBrewer, caTools, bitops, gtools, survival

License

[1] http://www.arrayanalysis.org
[2] http://dx.doi.org/10.1016/j.molimm.2013.07.001
[3] http://www.arrayanalysis.org
[4] http://www.apache.org/licenses/LICENSE-2.0

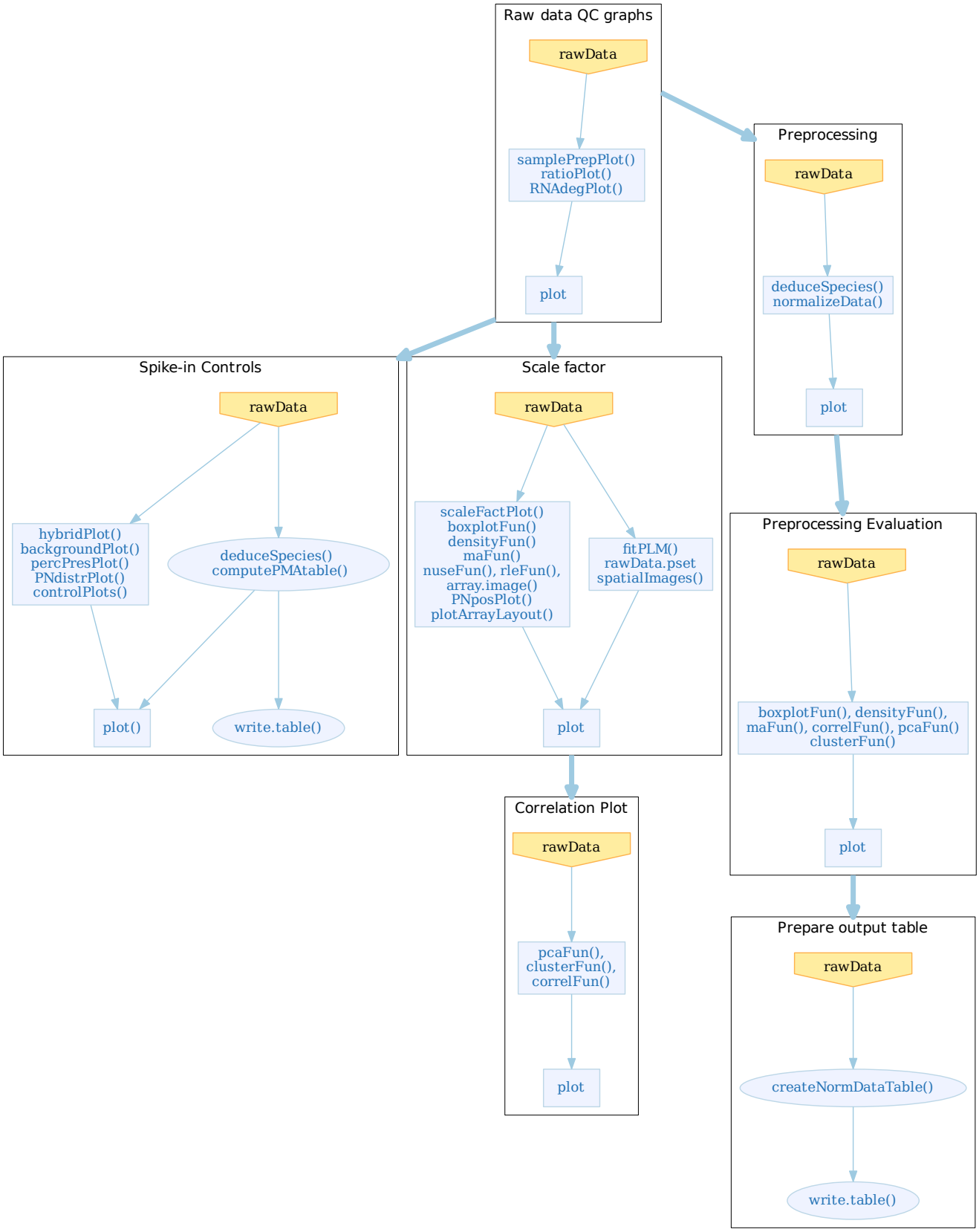


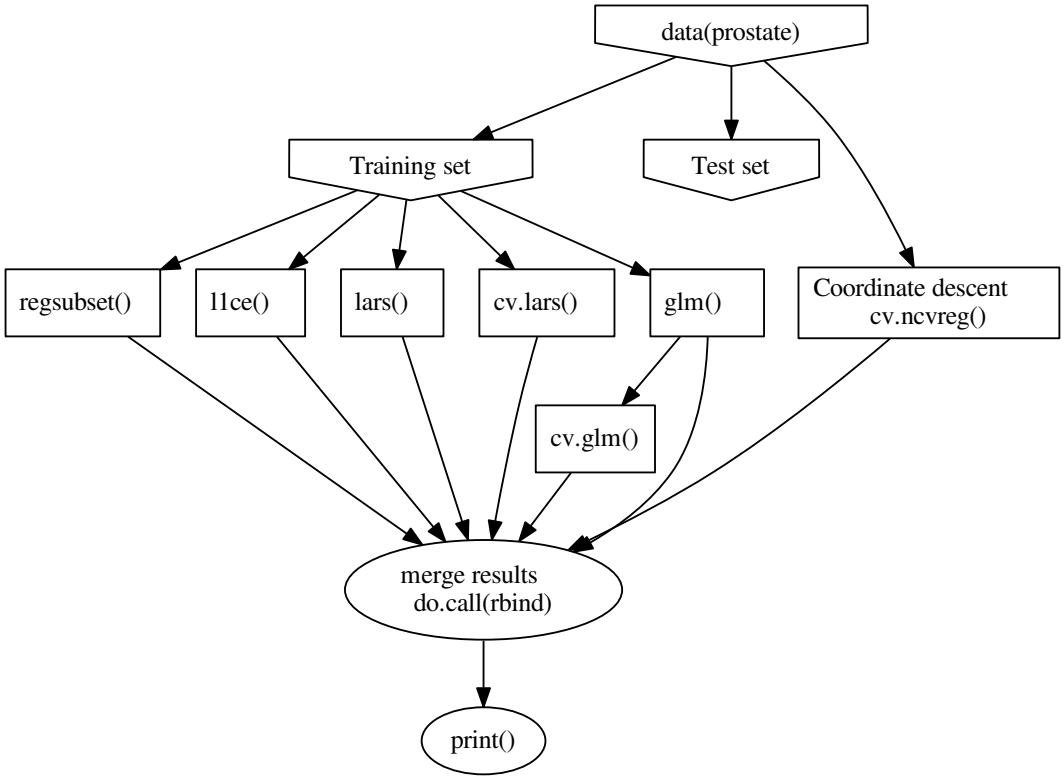
Figure 2.2: Diagram of 'affy' analysis workflow.

2.2.2 Clinical esIII (Essential Statistics Learning 2nd Edition)

In clinical studies, information about disease state and progression of a large number of patients is collected together with patient information that could (technically) be used for early diagnosis and or prognosis of disease. In this workflow clinical prostate cancer data of 97 patients is used to identify molecular markers that correlate with disease stage. The analysis is mainly inspired by the book 'Essential Statistical Learning[5]'.

Analysis used in this workflow are cross-validation for Lasso penalized regression fit and best predictive variable identification ('ncvreg' and 'leaps' packages); linear regression and fitting with L1 constraints ('stats' and 'lasso2' packages); Lasso penalized Least Angle Regression with cross validation ('lars' package); and fitting General Linear Model ('stats' package).

Figure 2.3: Workflow to identify prostate cancer markers using clinical data.



Packages and Dependencies

There are 6 packages used in this workflow, which depend on 2 additional packages from CRAN (dependencies)

Used packages:

- *CRAN*: ncvreg, boot, lars, lasso2, mda, leaps

Package dependencies:

- *CRAN*: class, MASS

[5] http://statweb.stanford.edu/ tibs/ElemStatLearn/

Data

source datasets from http://cran.r-project.org/web/packages/lasso2/lasso2.pdf

- **Prostate dataset:**
  - Hastie, T., Tibshirani, R., and Friedman, J. (2001). The Elements of Statistical Learning. Springer.
  - Stamey, T., et al. (1989). Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. II. Radical prostatectomy treated patients. Journal of Urology, 16: 1076-1083.

License

[6] https://github.com/biolion/genbench
[7] http://www.gnu.org/licenses/gpl.html

2.2.3 Clinical (liver cohort)

Single nucleotide ploymorphisms (SNPs) are single basepair mutations in DNA that occur with certain frequency within the population. These mutations might be the cause of a disease or due to their close vecinity to genetic abration they might be co-transfered with the cause.

In the study by Schadt EE et al., 2008[8] more than 780 thousand SNPs were analyzed in more than 400 human liver samples. Since a mutation could cause a phenotype that in time can cause a disease, the phenotypic information about the patients was included as well.

In this workflow, these data will be used as input for machine learning classification using SVM (from e1071 package). Data is devided into training and test sets and predictive models are generated for different features, these models are then tested using test dataset and the results are stored in a list.

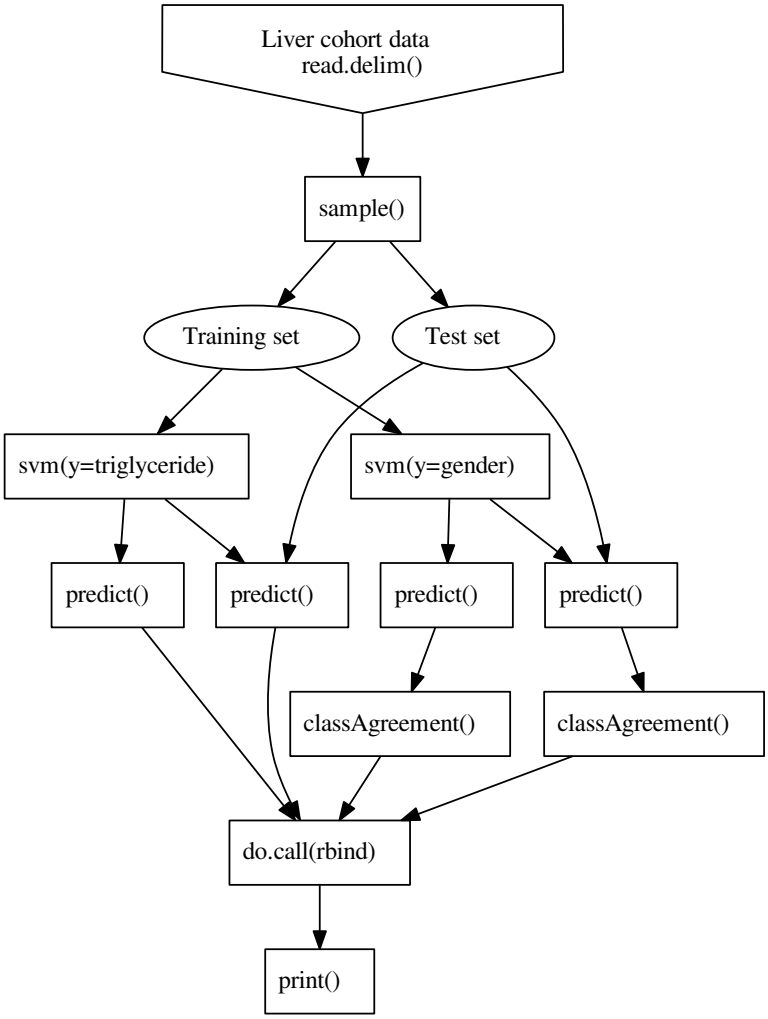


Figure 2.4: Analytic workflow for clinical liver cohort.

Packages and Dependencies

There is 1 package used in this workflow, which depends on 2 additional packages from CRAN (dependencies)

[8] http://dx.doi.org/10.1371/journal.pbio.0060107

Used packages:

- *CRAN*: e1071

Package dependencies:

- *CRAN*: class, MASS

Data

Downloaded from https://www.synapse.org/ in June 2015

License

[9] https://github.com/biolion/genbench
[10] http://www.gnu.org/licenses/gpl.html

2.2.4 Generate count (From RNAseq BAM files)

The amount of RNA produced by a specific gene is used as a surrogate marker for detection the activity, of that gene. With Current highthroughput sequencing technologies (such as in this case RNAseq), all (fragmented) RNA molecules from a biological sample are sequenced. These sequences are then matched to the annotated genome sequence to identify to which gene each sequenced fragment belongs. More sequenced fragments mapping to a gene in one samples as compared to another, means that the specific gene had a higher activity.

The first step in this workflow is reading the relevant parts of genome annotation file and converting it the a list of ranges of feature of interest such as exon (IRanges, GenomicRanges). Sequencing products previously aligned to genome are imported using Rsamtools. Mapping of mapped sequences to genes as defined in the anotation file is the next step. Followed by aggragation of data by summing the number of mapped fragments per gene.

Packages and Dependencies

There are 5 packages used in this workflow, which depend on 6 additional packages from Bioconductor and CRAN (dependencies)

Used packages:

- **Bioconductor**: BiocGenerics, GenomicRanges, IRanges, Rsamtools
- *CRAN*: parallel

Package dependencies:

- *Bioconductor*: Biostrings, GenomeInfoDb, XVector, S4Vectors, zlibbioc
- *CRAN*: bitops

Data

Bam files are single-end RNAseq data from Kartalaei PS et al, 2015.

License

[11] http://www.apache.org/licenses/LICENSE-2.0

2.2.5 Integration: iGraph

Genes are sequences of DNA that encode functional molecules (such as RNAs or Proteins). These functional molecules most of time interact and affect eachothers function. Therefor, to understand the role of a specific gene in a disease, scientist need to know about all the published functions.

Given many published scientific manuscripts about any given gene, the National Center for Biotechnology Information (NCBI)[12] has introduced GeneRIF[13]. GeneRIF provides scientist a simple mechanism to add functional annotation to known genes based on scientific publications. These annotations are than processed and approved by GeneRIF staff. All these functional annotations are freely accessible and can be used by anyone to extract functions of a gene or genes with specific function.

This workflow imports human Phophatases and Kinases and the publications. And uses 'igraph' to store this information and to plot the relation between these publications as well as interaction network between molecules.

Packages and Dependencies

There are 7 packages used in this workflow, which depend on 35 additional packages from CRAN (dependencies)

Used packages:

- *CRAN*: igraph, XML, R.utils, plyr, reshape, utils, sqldf

Package dependencies:

- *CRAN*: magrittr, irlba, Matrix, NMF, lattice, foreach, gridBase, pkgmaker, reshape2, stringr, colorspace, doParallel, digest, rngtools, ggplot2, RColorBrewer, cluster, registry, codetools, iterators, xtable, plyr, Rcpp, stringi, scales, gtable, MASS, proto, dichromat, labeling, munsell, RSQLite, gsubfn, chron, DBI

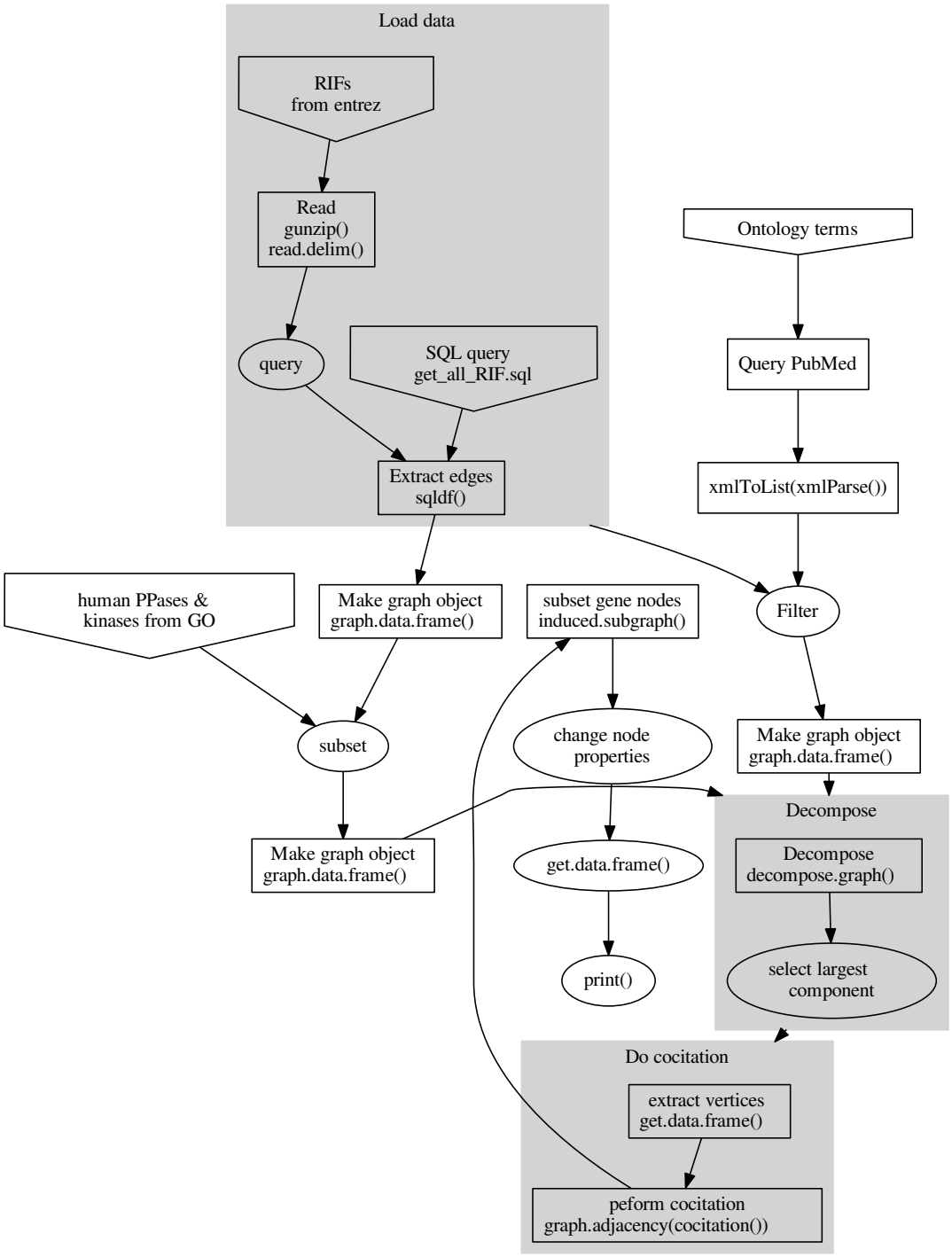
Data

- Basic GeneRIF data: ftp://ftp.ncbi.nih.gov/gene/GeneRIF/generifs_basic.gz
- SQL script written by Ieuan Clay: get_all_RIF.sql
- Human kinases and phosphatases protein: GeneOntology.org list of human (Taxonomy id: 9606) genes using terms GO:0050222 (protein kinase activity) and GO:0004721 (phosphoprotein phosphatase activity).

License

[12] http://www.ncbi.nlm.nih.gov
[13] http://www.ncbi.nlm.nih.gov/gene/about-generif
[14] https://github.com/biolion/genbench
[15] http://www.gnu.org/licenses/gpl.html

Figure 2.5: Workflow for integration liver cohort data.



2.2.6 Integration: Liver cohort

Recent high-throughput technologies have caused an explosion in the number of cohort studies which monitor and record medical (phenotype, genetic), social (phenotype), and biological (phenotype, genotype, transcriptional) information of large number of human participants for a long period of time. This allows to identify the variables that, for example, correlate with occurrence of diseases. Researchers hope that the causative events that have led to the disease are among the highest correlating variables. Given the limited number of validation experiments that a scientist can perform during his/her career, being able to narrow down the number of candidate variables without loosing the causative variable is of utmost importance. From population health perpective, earlier diagnosis of disease or slightly better prediction of disease progress (higher accuracy and precision) can have significant effects on population health and its costs. Machine learning algorithms can integrate information from multiple sources and are, therefor, most widely used.

This workflow uses data acquired in a cohort study by Schadt EE et al., 2008[16] on over 400 human liver samples. First gene expression, mutation and phenotype data from are cleaned up by removing variables containing missing data and patients which lack expression, mutation, or phenotype data. Multiple machine leaning algorithms such as Support Vector Machines, Native Bayesian, and Robust regression are than used to create predictive models.

Support Vector Machine ('e1071' package) is used to train a model (classification and regression) using random sample of 2/3 of samples (training set) and tested with the remaining 1/3 of samples (test set). Model is trained based on independent variables such as age and liver triglyceride levels, and dependent variables such as activity of nine liver enzymes.

Using enzyme activity information, heatmaps ('stats' package) are generated to visualize correlation between enzymes and correlation between patients. For clustering of patients based on enzyme activity data, heatmap are grouped based on Principal Component Analysis results (prcomp from 'stats' package). Naive Bayesian classifier ('e1071' package) is used to cluster samples based on gene expression profile with aldehyde oxydase levels or liver enzyme activity as class vector (independent variable).

Furthermore, Robust linear model ('MASS' package) is used to train model using liver triglyceride levels and gene expression levels of genes with highest variance. A random sample of 25 genes are selected from 1000 genes with the highest variance and used in combination with triglyceride level phenotype. This is done in 50 iteration and the iteration with highest correlation in training and test sets is recorded.

Packages and Dependencies

There are 3 packages used in this workflow, which depend on 1 additional package from CRAN (dependency)

Used packages:

- *CRAN*: stats, e1071, MASS

Package dependencies:

- *CRAN*: class

Data

from Human Liver Cohort (Synapse ID: syn4499)[17]

License

[16] http://doi.org/10.1371/journal.pbio.0060107
[17] https://www.synapse.org/#!Synapse:syn4499
[18] https://github.com/biolion/genbench
[19] http://www.gnu.org/licenses/gpl.html

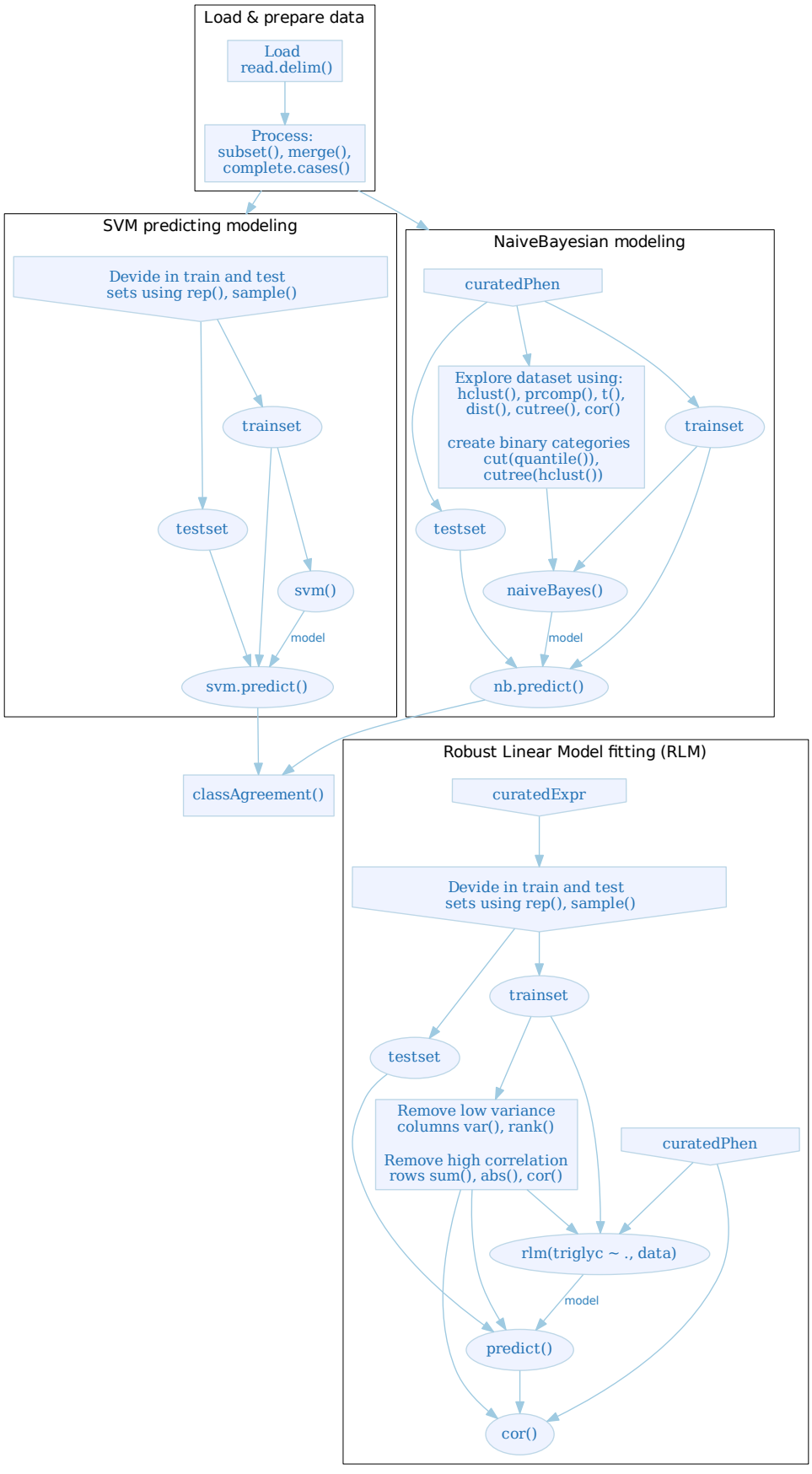


Figure 2.6: Diagram for integration livercohort benchmark.

2.2.7 Microarray

Mutations in DNA that is coding for proteins (genes) or regulatory elements are often the cause of most diseases. Affymatirx arrays allow quantification of specific sequences of DNA or RNA in biological samples such as blood, tissue, or tumors. These arrays give the relative levels of specific transcript (between two samples) so that these can be compared (between eg. healty vs diseased tissues or before vs after treatment).

To do so, DNA sequences unique to specific genes/mutations are printed as small spots on a glass surface (arrays), with each spot containing thousends of copies of a single sequence. The DNA from each sample is than tagged with different colors of flourecent molecules (red or green) and hybredized with the array. This way, DNA from sample can bind to sequences on the surface that are antisense of its own sequence (specific binding). The array undergoes several washing steps to remove the none-specifically bound DNA sequences, which bind loosly. The plate is than scanned and the flourecence intensity for each color at each spot is recorded. This intensity is an indicator of the ammount of DNA bound and probably the levels of that specific DNA present in the sample. The information regarding the specific sequence printed on each spot, control spots, and other array specific information is stored in a CDF format/file. By comparing the intensity of red to green at each spot you can know which sample contained more of that specific transcript.

This workflow uses limma[20] package for analysis of microarray data to analyse the data from Ramsey et al 2013[21]. The raw data is normalized and differential gene expression analysis and a simple gene set testing is performed as described in limma vignet.

Packages and Dependencies

There are 4 packages used in this workflow, which depend on 5 additional packages (dependencies).

Used packages:

- *Bioconductor*: Biobase, affy, hgu133plus2cdf, limma

Package dependencies:

- *Bioconductor*: BiocGenerics, BiocInstaller, zlibbioc, preprocessCore, affyio

Data

From Gene Expression Omnibus repository accession ID GSE45417[22].

License

- Copyright (c) 2005 Gordon Smyth based on Limma package[23]
- Copyright (c) 2015 Ieuan Clay based on code from genbench[24]
- Copyright (c) 2015-2016 BeDataDriven B.V. License: GPL version 2 or higher[25]

[20] http://www.bioconductor.org/packages/release/bioc/html/limma.html
[21] http://doi.org/10.1016/j.molimm.2013.07.001
[22] http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE45417
[23] http://www.bioconductor.org/packages/release/bioc/html/limma.html
[24] https://github.com/biolion/genbench
[25] http://www.gnu.org/licenses/gpl.html

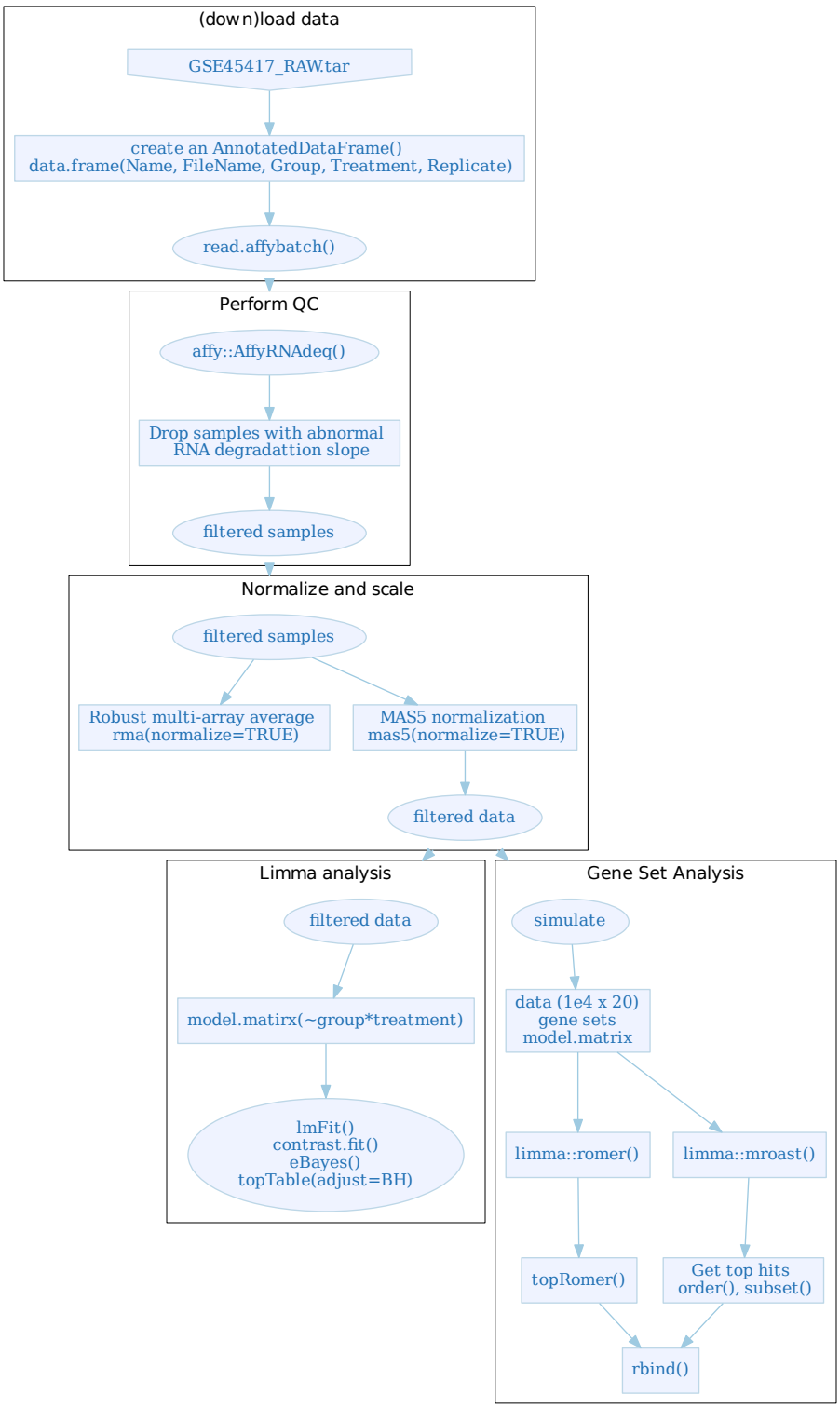
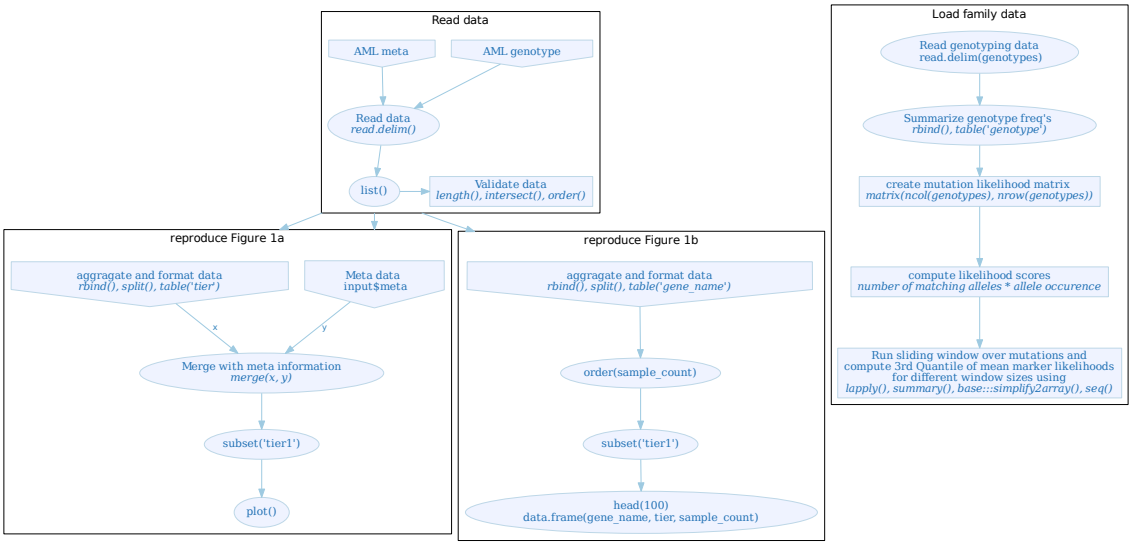


Figure 2.7: Diagram for the microarray benchmark.

2.2.8 Mutation

Two main types of genetic studies are population and familial/pedigree studies. In this workflow, individual mutation information is used to determine the relatedness between individuals and data from The Cancer Genome Atlas landmark paper on most common AML mutations[26] is used to reproduce some of the figures in this publication.

Figure 2.8: Diagram of mutation analysis workflow.



Packages and Dependencies

There are 2 core packages used in this workflow, which have no dependencies.

Used packages:

- *Core*: stats, utils

Data

The familial data was obtained from the nice people at Genomes Unzipped[27], who make their own genomic data[28] publicly available.

The dataset we are using comes from the 23andme v2[29] sequencing service.

Though the individuals are not related, this data can still be used to perform some typical tests carried out on pedigree studies, such as determining "relatedness" between individuals.

[26] http://www.doi.org/10.1056/NEJMoa1301689
[27] http://genomesunzipped.org/members
[28] http://genomesunzipped.org/data
[29] https://www.23andme.com/

member	dataset id	link
Daniel MacArthur	DGM001	http://s3.amazonaws.com/gnz.genotypes/DGM001_genotypes.zip
Luke Jostins	LXJ001	http://s3.amazonaws.com/gnz.genotypes/LXJ001_genotypes.zip
Dan Vorhaus	DBV001	http://s3.amazonaws.com/gnz.genotypes/DBV001_genotypes.zip
Caroline Wright	CFW001	http://s3.amazonaws.com/gnz.genotypes/CFW001_genotypes.zip
Kate Morley	KIM001	http://s3.amazonaws.com/gnz.genotypes/KIM001_genotypes.zip
Vincent Plagnol	VXP001	http://s3.amazonaws.com/gnz.genotypes/VXP001_genotypes.zip
Jeff Barrett	JCB001	http://s3.amazonaws.com/gnz.genotypes/JCB001_genotypes.zip
Jan Aerts	JXA001	http://s3.amazonaws.com/gnz.genotypes/JXA001_genotypes.zip
Joe Pickrell	JKP001	http://s3.amazonaws.com/gnz.genotypes/JKP001_genotypes.zip
Don Conrad	DFC001	http://s3.amazonaws.com/gnz.genotypes/JKP001_genotypes.zip
Carl Anderson	CAA001	http://s3.amazonaws.com/gnz.genotypes/CAA001_genotypes.zip
Ilana Fisher	IPF001	http://s3.amazonaws.com/gnz.genotypes/IPF001_genotypes.zip

The population study data is from the TCGA consortium publication TCGA, 2013[30], publication data archive[31], mutation and annotation (maf[32]), and patient meta data[33].

License

[30] http://www.doi.org/10.1056/NEJMoa1301689
[31] https://tcga-data.nci.nih.gov/docs/publications/laml_2012/
[32] http://tcga-data.nci.nih.gov/docs/publications/laml_2012/genome.wustl.edu_LAML.IlluminaGA_DNASeq.Level_2.2.12.0.tar.gz
[33] http://tcga-data.nci.nih.gov/docs/publications/laml_2012/clinical_patient_laml.tsv
[34] https://github.com/biolion/genbench
[35] http://www.gnu.org/licenses/gpl.html

2.2.9 RNAseq DESeq2

All the functions that take place within a cell are performed through proteins. These proteins are coded within the DNA (Deoxyribonucleic acid) of the cell. A gene is a sequence of DNA that encodes for a particular protein. In order to make the necessary proteins, the transcriptional machinary of a cell makes special copies of the respective genes that can be translated to protein sequences. These special copies are called messenger RNAs (Ribonucleic acids).

The amount of mRNA produced by a specific gene is used as a surrogate marker for quantification of the gene activity. With Current highthroughput sequencing technologies (such as in this case RNAseq), all (fragmented) RNA molecules from a biological sample are sequenced. These sequences are then matched to the annotated genome sequence to identify to which gene each sequenced fragment belongs. More sequenced fragments mapping to a gene in one samples as compared to another, means that the specific gene had a higher activity.

This workflow uses the exploratory analysis of RNAseq data using many well stablished Bioconductor packages such as DESeq2, Rsamtools, and GenomicAlignments as described in Love MI et al., 2015[36]

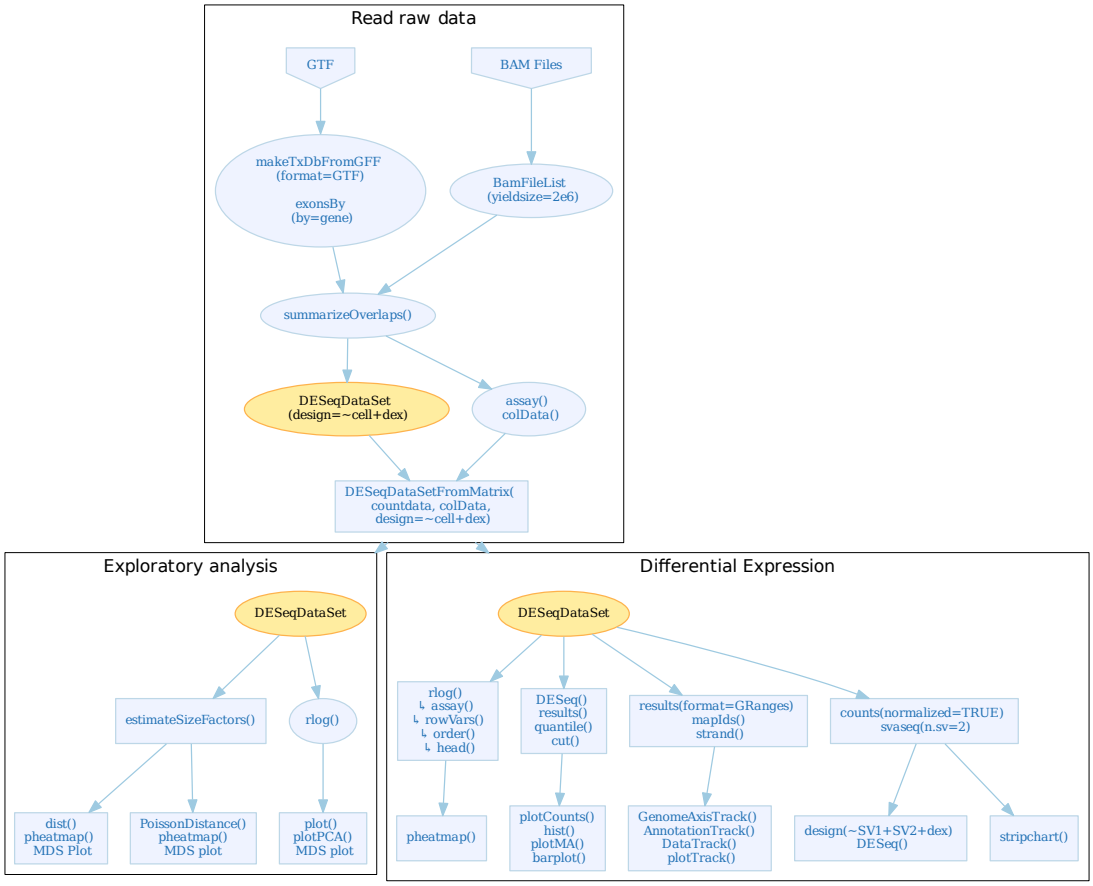


Figure 2.9: Diagram of RNAseq analysis using DESeq2.

Packages and Dependencies

There are 17 packages used in this workflow, which depend on 79 additional packages (dependencies).

Used packages:

[36] http://doi.org/10.12688/f1000research.7035.1

- *Bioconductor*: Rsamtools, GenomicFeatures, GenomicAlignments, BiocParallel, DESeq2, genefilter, AnnotationDbi, org.Hs.eg.db, ReportingTools
- *CRAN*: pheatmap, RColorBrewer, PoiClaClu, ggplot2, Gviz, fission, sva, fission

Package dependencies:

- *Bioconductor*: BiocGenerics, Biostrings, GenomeInfoDb, XVector, S4Vectors, rtracklayer, Biobase, biomaRt, IRanges, zlibbioc, GenomicRanges, geneplotter, ggbio, PFAM.db, limma, edgeR, GSEABase, GOstats, VariantAnnotation, BSgenome, OrganismDbi, biovizBase, GO.db
- *CRAN*: bitops, RCurl, RSQLite, DBI, XML, snow, futile.logger, Rcpp, RcppArmadillo, Hmisc, locfit, plyr, scales, reshape2, gtable, digest, MASS, proto, Formula, lattice, gridExtra, nnet, acepack, latticeExtra, cluster, rpart, foreign, survival, annotate, dichromat, labeling, munsell, stringr, xtable, colorspace, magrittr, stringi, Category, R.utils, hwriter, knitr, GGally, graph, Matrix, RBGL, AnnotationForge, evaluate, markdown, yaml, highr, formatR, reshape, mime, matrixStats, mgcv, nlme

Data

RNAseq bam-files from Solaimani Kartalaei P, (2014)[37]

License

[37] http://www.doi.org/10.1084/jem.20140767
[38] http://www.doi.org/10.12688/f1000research.7035.1

2.2.10 Reverse phase protein array (rppa)

Processes within a cell are mainly performed by proteins. Special copies of a gene, called messenger RNAs (mRNAs), are produced and transported to the cells protein synthesis machinery to produce the corresponding protein. The more mRNA molecules of a gene cause the more of its corresponding protein is produced. However, this is not always a linear relationship. Presence or absence of some molecules (including other proteins) can inhibit or enhance the production of a protein.

High-throughput technologies such as micro-arrays and RNA/DNA sequencing technologies measure the level of mRNA as a surrogate marker for protein level. There are a few highthroughput technologies that allow direct measurement of the protein levels. The reverse phase protein array (RPPA) is one such technology. To perform RPPA, cell/tissue lysate of interest is printed as small dots (droplets) of the same size on surface of a special plate. Each dot on the plate is then stained with fluorochrome conjugated antibodies against one specific protein. The plate is then washed and scanned to detect the fluorescence. Since most antibodies bind to multiple proteins due to non-specific binding, the requirement for antibodies which only detect a single protein in one of the limitation of RPPA.

In this workflow, RPPA dataset[39] from TCGA consortium is clustered using R hierarchical clustering and K-means clustering from 'stats' package.

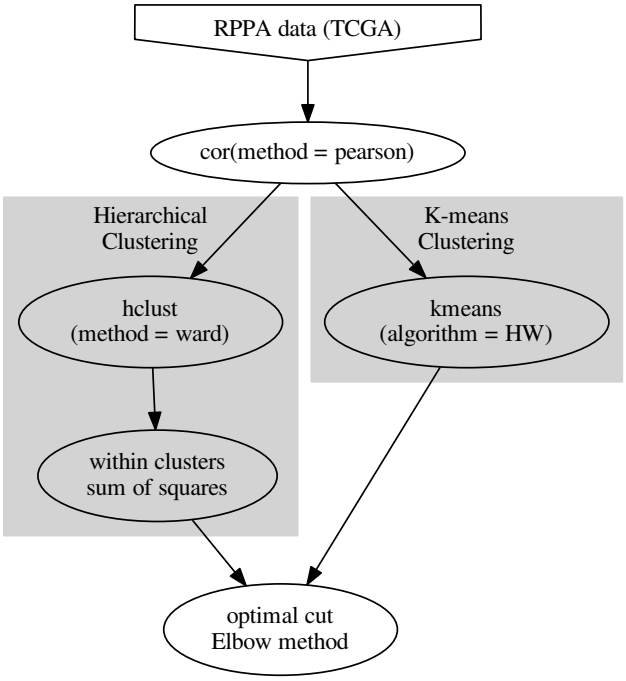


Figure 2.10: Diagram of 'rppa' workflow.

Packages and Dependencies

There is ony 1 package used in this workflow, which has no dependencies.

Used packages:

- *CORE*: stats

[39] http://tcga-data.nci.nih.gov/docs/publications/TCGApancan_2014/RPPA_input.csv

Data

- RPPA data: http://tcga-data.nci.nih.gov/docs/publications/TCGApancan_2014/RPPA_input.csv

License

[40] https://github.com/biolion/genbench
[41] http://www.gnu.org/licenses/gpl.html

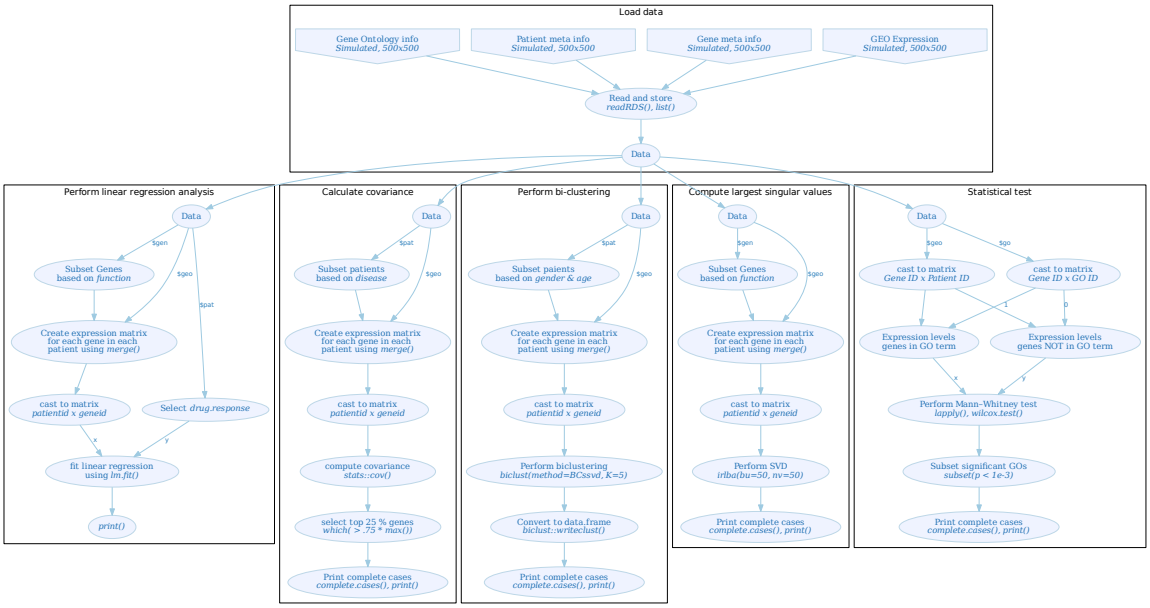
2.2.11 Simulated GEO matrix

Mutations in DNA that is coding for proteins (genes) or regulatory elements are often the cause of most diseases. Affymatirx arrays allow quantification of specific sequences of DNA or RNA in biological samples such as blood, tissue, or tumors. These arrays give the relative levels of specific transcript (between two samples) so that these can be compared (between eg. healty vs diseased tissues or before vs after treatment).

To do so, DNA sequences unique to specific genes/mutations are printed as small spots on a glass surface (arrays), with each spot containing thousends of copies of a single sequence. The DNA from each sample is than tagged with different colors of flourecent molecules (red or green) and hybredized with the array. This way, DNA from sample can bind to sequences on the surface that are antisense of its own sequence (specific binding). The array undergoes several washing steps to remove the none-specifically bound DNA sequences, which bind loosly. The plate is than scanned and the flourecence intensity for each color at each spot is recorded. This intensity is an indicator of the ammount of DNA bound and probably the levels of that specific DNA present in the sample. The information regarding the specific sequence printed on each spot, control spots, and other array specific information is stored in a CDF format/file. By comparing the intensity of red to green at each spot you can know which sample contained more of that specific transcript.

For this workflow a simulated dataset was used containing patient meta-information, gene expression levels (microarray), and gene ontology information for 500 patients and 500 genes. This workflow performs Standard statistical methods such as linear regression model (lm.fit from stats package), covariance ('cov' from 'stats' package), biclustering ('biclust' package), SVM ('irlba' package), and differential gene expression using two sample Wilcoxon test (also know as 'Mann-Whitney' test) are performed.

Figure 2.11: Diagram of Simulated GEO Matrix analysis workflow.



Packages and Dependencies

There are 3 packages used in this workflow, which depend on 7 additional packages from CRAN (dependencies)

Used packages:

- *CRAN*: biclust, s4vd, irlba

Package dependencies:

- *CRAN*: lattice, colorspace, MASS, flexclust, modeltools, biclust, Matrix

Data

This workflow uses simulated data from GenBase data_generator.py[42] with 500 as size of columns and rows. Here you can read about the original GenBase study by Taft R et al., 2014[43].

License

- Copyright (c) 2015 MIT DB Group based on code from GenBase[44]
- Copyright (c) 2015 Hannes Mühleisen based on code from GenBase (fork)[45]
- Copyright (c) 2015 Ieuan Clay based on code from genbench[46]
- Copyright (c) 2015-2016 BeDataDriven B.V. License: GPL version 2 or higher[47]

[42] https://github.com/mitdbg/genbase/blob/master/data/data_generator.py
[43] http://dx.doi.org/10.1145/2588555.2595633
[44] https://github.com/mitdbg/genbase/blob/master/code/R_benchmark/vanilla_R_benchmark.R
[45] https://github.com/hannesmuehleisen/genbase/blob/master/code/R_benchmark/vanilla_R_benchmark.R
[46] https://github.com/biolion/genbench
[47] http://www.gnu.org/licenses/gpl.html

2.2.12 Survival simple

All cancers are caused by mutations in DNA. The Cancer Genome Atlas (TCGA)[48] is a project that begun in 2005 and aims to catalogue the mutations that cause cancer in patients. It uses highthroughput sequencing technologies and bioinformatics to achive this gaol. Beside genomic data, TCGA also records and provides detailed anonymize meta information about each patient (such as disease stage, patient age, sex, healt, etc) which are very valuable for identification of risk factors.

In this workflow, survival analysis is performed on TCGA patient survival data with BAZ2A gene mutations as predictor. The survival package[49] is one of the most used packages for survival analysis in R.

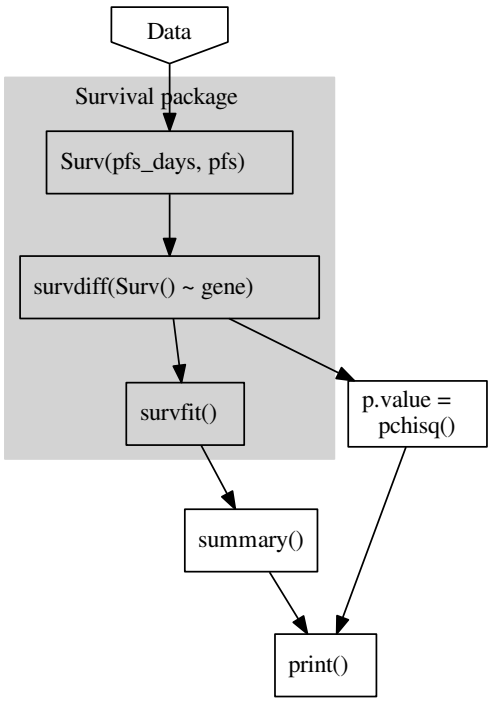


Figure 2.12: Diagram of 'Survival Simple' workflow.

Packages and Dependencies

There is only 1 package used in this workflow, which has no additional dependencies.

Used packages:

- *CRAN*: survival

Data

- pat.gene.rda: patient survival data from TCGA consortium processed and formatted by Phil Cheng

[48] http://cancergenome.nih.gov/
[49] https://cran.r-project.org/web/packages/survival/index.html

License

License: GPL version 2 or higher[50]

[50] http://www.gnu.org/licenses/gpl.html

2.2.13 Survival TCGA

All cancers are caused by mutations in DNA. The Cancer Genome Atlas (TCGA)[51] is a project that begun in 2005 and aims to catalogue the mutations that cause cancer in patients. It uses highthroughput sequencing technologies and bioinformatics to achive this gaol. Beside genomic data, TCGA also records and provides detailed anonymize meta information about each patient (such as disease stage, patient age, sex, healt, etc) which are very valuable for identification of risk factors.

In this workflow, survival analysis is performed on TCGA meta and survival data using the Cox regression model. The glmnet package[52] is one of the most efficient packages for such an analysis in R.

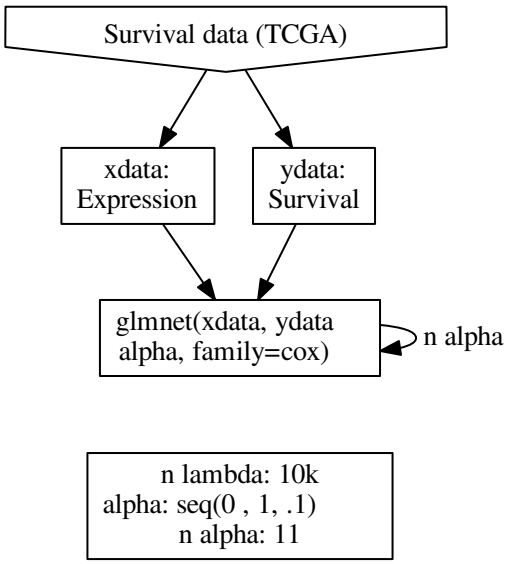


Figure 2.13: Diagram of Survival TCGA workflow.

Packages and Dependencies

There are 2 packages used in this workflow, which depend on 5 additional packages from CRAN (dependencies)

Used packages:

- *CRAN*: glmnet, survival

Package dependencies:

- *CRAN*: foreach, Matrix, codetools, iterators, lattice

Data

- Patient survival data from TCGA consortium provided by Andre Verissimo.

[51] http://cancergenome.nih.gov/
[52] https://cran.r-project.org/web/packages/glmnet/index.html

License

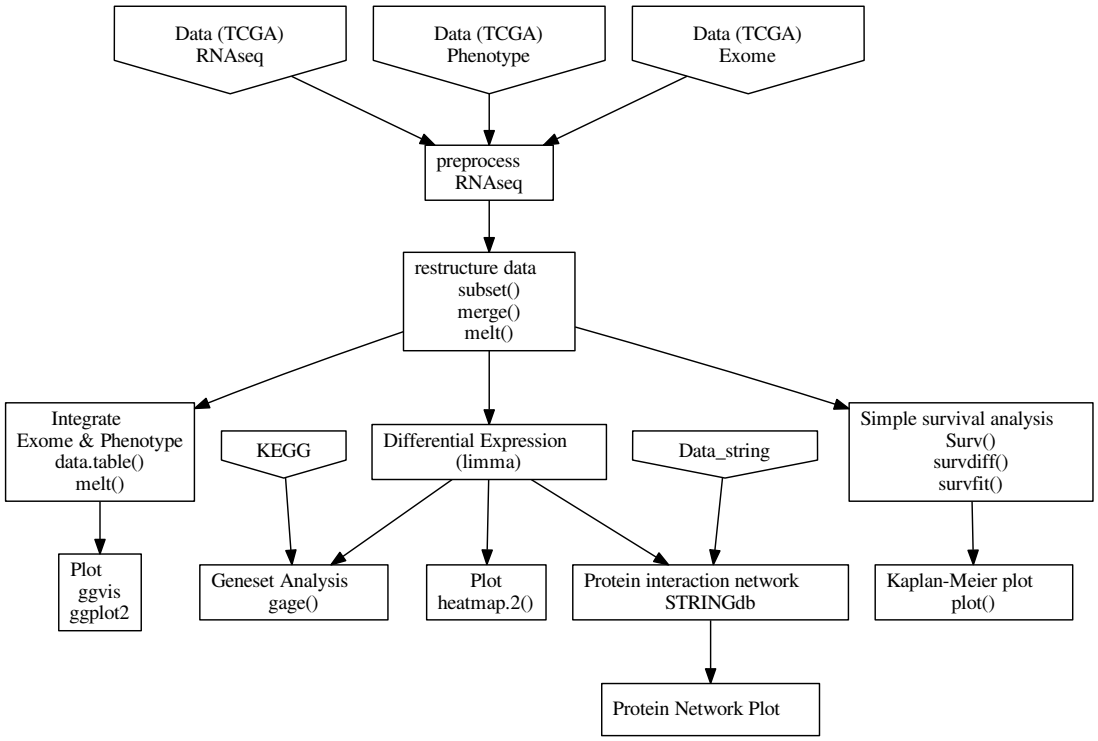
[53] andre.verissimo@tecnico.ulisboa.pt
[54] http://www.gnu.org/licenses/gpl.html

2.2.14 TCGA browser

All cancers are caused by mutations in DNA. The Cancer Genome Atlas (TCGA)[55] is a project that begun in 2005 and aims to catalog the mutations that cause cancer in patients. DNA sequence, gene expression profile, and other relevant patient information are collected for identification of risk factors and better understanding the disease mechanism.

First, gene expression profiles (the amount of RNA transcripts produced by each gene), DNA mutations, and patient information which have been previously retrieved from TCGA repository are loaded. These data are stored as data frames using 'data.table' package which is one of the most efficient (memory- and speed wise) packages for handeling large data frames. Significant differences in gene activity (expression) are calculated using the 'limma' package. This type of analysis result often in large lists of differentially expressed, each of these genes is in turn is involved in a few and sometimes many biological processes. In analysis of gene expression data, it is a common practice to statistically test whether there are biological processes/pathways whose genes are significantly overrepresented using a gene set analysis (GSA) method. This gives us a view at the level of biological processes. In this workflow 'gage' package has been used which uses Generally Applicable Gene-set Enrichment GAGE method[56]. Gene transcripts are translated to proteins and proteins can interact and inhibit or activate each others function or expression level. Information about protein-protein interactions are continuously added to STRING database[57] which is accessible through 'STRINGdb' package and used to show the interaction between the genes with significant differential expression.

Figure 2.14: TCGA browser shiny app modules.



Packages and Dependencies

There are 24 packages used in this workflow, which depend on 71 additional packages (dependencies)

[55] http://cancergenome.nih.gov/
[56] http://doi.org/10.1186/1471-2105-10-161
[57] http://string-db.org/

Used packages:

- *Bioconductor*: limma, edgeR, gage, STRINGdb,
- *CRAN*: ncvreg, boot, lars, lasso2, mda, leaps, data.table, reshape2, ggplot2, magrittr, survival, googleVis, plyr, grid, d3heatmap, ggvis, RColorBrewer, DT, jsonlite
- *Github*: rCharts

Package dependencies:

- *Bioconductor*: BiocGenerics, GenomeInfoDb, S4Vectors, Biobase, IRanges, Biostrings, AnnotationDbi, KEGGREST, XVector
- *CRAN*: irlba, Matrix, NMF, lattice, foreach, gridBase, pkgmaker, stringr, colorspace, doParallel, digest, rngtools, cluster, registry, codetools, iterators, xtable, Rcpp, stringi, scales, gtable, MASS, proto, dichromat, labeling, munsell, RSQLite, gsubfn, chron, DBI, graph, png, httr, zlibbioc, R6, jsonlite, curl, mime, RCurl, igraph, hash, gplots, plotrix, sqldf, bitops, gtools, KernSmooth, caTools, gdata, class, RJSONIO, htmlwidgets, dendextend, base64enc, yaml, htmltools, whisker, dplyr, assertthat, lazyeval, shiny, httpuv

Data

- Exome, Patient, RNAseq data from TCGA consortium and processed by Phil Cheng
- Protein data downlaoded from STRINGdb package for tax_id 9606 (Human)

License

[58] http://www.gnu.org/licenses/gpl.html

2.3 Performance in GNU R

2.3.1 Profiling benchmarks with GNU R

Profiling environment

We used the following environment to test the profile and analyse the benchmarks with GNU R.

GNU R instrumentation results

We instrumented GNU R by addition of timers to functions that are used for calling external C, C++, and Fortran code (.C, .Call, and .External functions). This information provides us with an overview of where the most time is spent in each workflow.

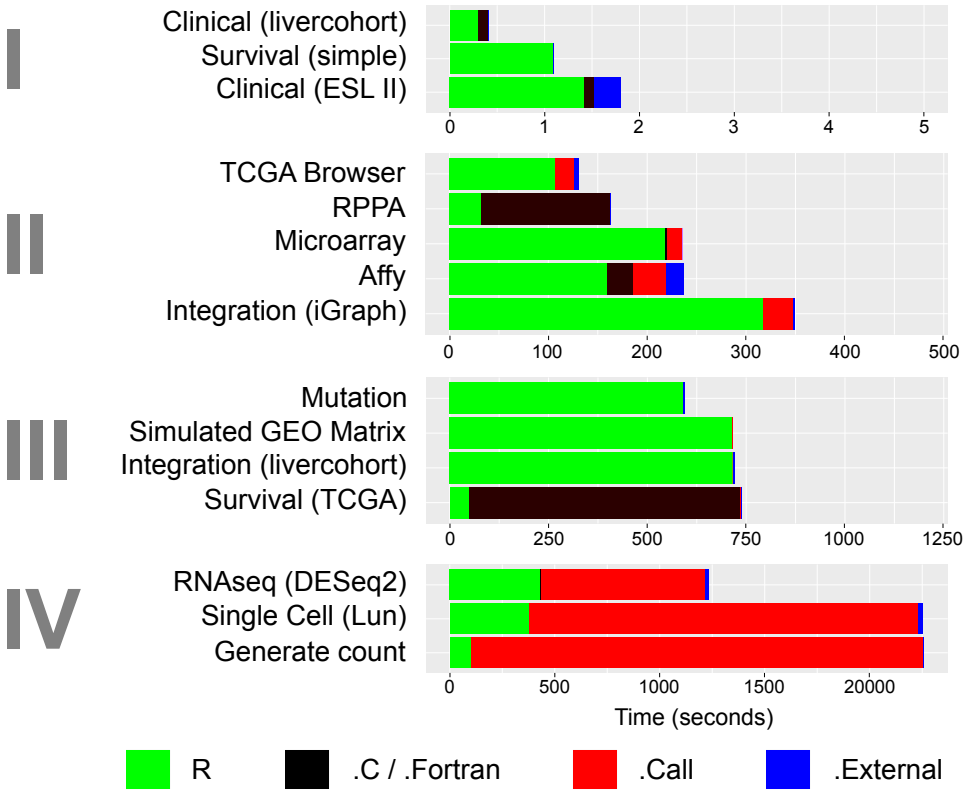


Fig. 2.16: Running of benchmarks on instrumented GNU R. Distribution of time spent in executing R code (green) or native code through .C/.Fortran (black), C++ through .Call (red), .External (blue) interfaces. The benchmarks are sorted and grouped based on running time; < 1 minute (group I), between 1 – 8 minutes (group II), 8 – 20 minutes (group III), and > 20 minutes (group IV).

The running time of the workflows ranges from 0.5 second (Clinical liver cohort) to 37 minutes (generate_count). In workflows with long running times, most of time is spent in native code such as C/Fortran (RPPA, Survival TCGA) or C++ (Generate count, RNAseq_DESeq2, Single cell Lun). Indeed, it has become a common practice to rewrite slow parts of R code in native languages. However, having to rewrite an algorithm (partially) in other languages is an extra burden for scientists in terms of learning new programming languages and maintaining such hybrid code base.

CPU Profiling with GNU R

Analysis run fastest when the number of operations can be reduced and by making optimal use of the available (hardware) resources (CPU, memory, etc). There are, therefore, two kind of optimizations that can be built into

the interpreter, reducing the number of operations (overhead), and improving the usage of for example CPUs. We have analysed the CPU, and CPU cache memory usage while running each of the workflows.

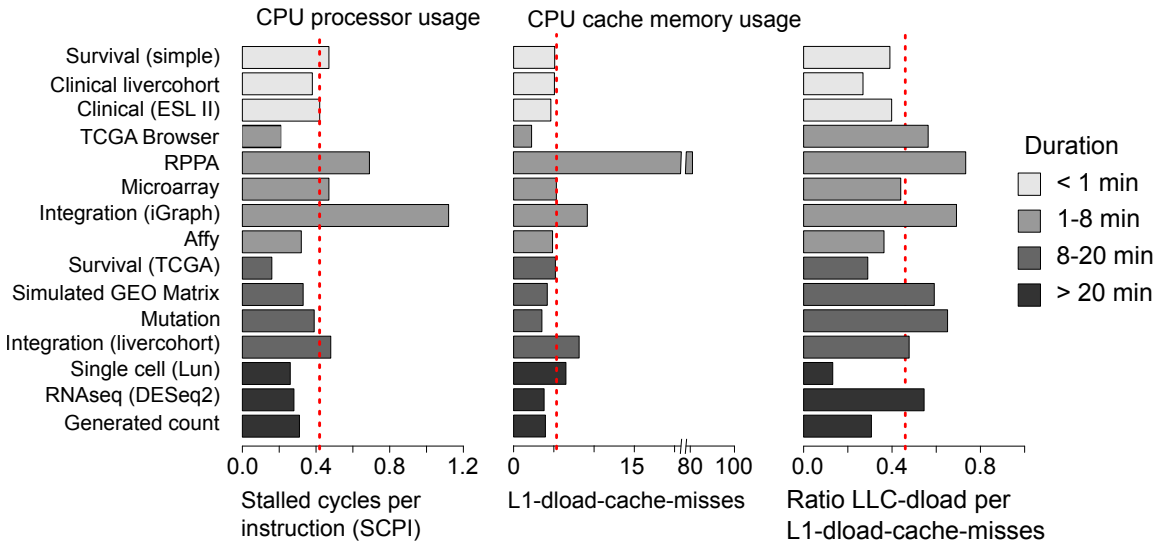


Fig. 2.17: Analysis of the CPU and CPU cache memory usage by benchmarks. (Left) Stalled cycles per instruction (SCPI) indicating how many operations waited for necessary data to be loaded in the cache. Red dashed line is the mean SCPI (trimmed 10%). (Middle) Percentage of L1-dcache-load-misses in each benchmark with red dashed line indicating 10% trimmed mean of all benchmarks. (Right) The ratio of number of lookups in slowest cache with number of L1-dcache-load-misses, indicating what part of data that were not found in fastest cache memory were found in the slowest cache (LLC) and not in intermediate caches (L2).

For each operation the required data is loaded in a super fast memory on the CPU chip named L1-cache. If the data necessary for the operation is not found, the CPU searches on the slower (but larger) caches such as L2, L3, and LCC (each level is slower but larger than previous one). If the data/instruction is not found on L1 cache and need to be looked up in other caches, this is called L1-dcache-load-misses.

We observed between 2-81% L1-cache-load-misses while running the workflows with lowest cache-misses being observed in 'TCGA Browser' and the highest in 'RPPA'. In addition, between 13-71% of L1-dcache-load-misses had to be looked up in the slowest LCC cache. All this suggest significant gains can be achieved by optimizing processor cache usage. The metrics instructions per CPU cycle (IPC) and stalled cycles per instruction (SCPI) indicates how efficient the processor cycles are being used. Ideally you want to reduce SCPI and increase the IPC. An IPC lower than 1 is considered inefficient code. This is the case for only one of the workflows (integration_igraph) with 'rppa' being on the border (1.01 instructions per cycle, data not shown). Likewise 'Integration (iGraph)' and 'RPPA' have the highest SCPI values. This is surprising since 'RPPA' is implemented in C and was expected to be efficient. More detailed analysis of the 'RPPA' code reveals that most of the time is spend in 'kmeans' algorithm which is implemented in C. 'kmeans' is a very popular algorithm for partitioning samples in a defined number of clusters based on properties of each sample. It is interesting to know whether we can replace the current naive 'kmeans' C code with a more cache-memory aware algorithm written in R or Java.

Summary

The collected benchmarks cover a variety of analysis commonly performed in the field of biomedical research. The benchmarks are highly variable in their running time, efficient use of resources, size of input data, and implementation of algorithms in native languages such as C, C++, and Fortran. These benchmarks could, therefore, assist in identification of computational bottlenecks, causes of inefficient uses of available hardware, incompatibility with the standard R R interpreter. In addition, the efficiently running benchmarks could help us to detect regressions during development of Renjin. Improvement of IPC/SCPI, running time, and reduction CPU cache misses are valuable metrics for this purpose.