Bulk RNA-seq analysis using R (Rsubread)

RNA Sequencing

RNA sequencing (RNA-Seq) uses the capabilities of high-throughput sequencing methods to provide insight into the transcriptome of a cell. RNA-Seq provides far higher coverage and greater resolution of the dynamic nature of the transcriptome.

Prequisites For R

Windows OS (at least 8GB RAM) with working command line

Install R(version "4.4") - https://cran.r-project.org/bin/windows/base/

Install RStudio - https://posit.co/download/rstudio-desktop/

After installing R run the following command to install Rsubread:

```
if (!require("BiocManager", quietly = TRUE))

install.packages("BiocManager")

BiocManager::install("Rsubread")

BiocManager::install("edgeR")

BiocManager::install("limma")
```

The input files for the RNAseq analysis are to be downlaoded from link : https://figshare.com/s/f5d63d8c265a05618137 OR from the R folder

- SRR1552444.fastq.gz
- SRR1552445.fastq.gz
- SRR1552454.fastq.gz
- SRR1552455.fastq.gz

**Download the reference genome from this link :

https://figshare.com/s/f5d63d8c265a05618137 The following files are te refrence files named as chr1_mm10 and the index file named: chr1_mm10.files, chr1_mm10.00.b.tab and chr1_mm10.00.b.array.

Download all the files given under the R folder.

Set the working directory in R to where you have downloaded all the files:

```
setwd("C:/path/to/your/directory")
```

Step 1 : Loading R Packages

Rsubread provides functions for read alignment and feature counting. It is particularly useful for handling large RNA-seq datasets efficiently.

Limma is an R/Bioconductor software package that provides an integrated solution for analysing data from gene expression experiments. It contains rich features for handling complex experimental designs and for information borrowing to overcome the problem of small sample sizes.

edgeR is a Bioconductor package for differential expression analysis of digital gene expression data.

```
library(Rsubread)

library(limma)

library(edgeR)
```

Step 2 : QC of the raw reads

FastQc to check the quality of raw reads

Download from the link below:

https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Step 3 : Listing FASTQ Files for Analysis

FASTQ files are a common format used to store raw sequence data from high-throughput sequencing experiments. Each FASTQ file contains sequences of nucleotides along with their corresponding quality scores. FastQ files contains raw sequencing reads. Each file represents reads from a specific sample. This step is important because it allows us to identify all the FASTQ files in the directory, ensuring that we process all available samples.

```
reads1 <- list.files(path=".", pattern="*.fastq.gz$")
```

Step 4 : Building Index For Reference Genome

Indexing a reference genome is the process of generating a set of data structures that facilitate rapid access to specific locations within the genomic sequence. Indexing is crucial for improving the performance and speed of downstream analyses, such as alignment and variant detection.

```
buildindex(basename="chr1_mm10",reference="chr1.fa")
```

Step 5 : Alignment

The next step is to align the RNA-seq reads to the reference genome using the "align" function from the Rsubread package. The input format is indicated as "FASTQ" to show the input files are in FASTQ format. This process involves mapping the sequence of nucleotides in the genome into a format that can be efficiently searched, enabling bioinformatics tools to quickly locate and align sequencing reads to the reference genome.

The output format is specifies as "BAM" to indicate that the output should be in BAM format. The RNA seq reads are aligned with indexed reference genome.

```
align(index="chr1_mm10", readfile1=reads1, input_format="FASTQ", output_format="BAM")
```

Step 6 : BAM File

A BAM file (*.bam) is the compressed binary version of a SAM file that is used to represent aligned sequences. BAM files store aligned sequence data, which includes information on where each read maps to the reference genome. BAM files contain a header section and an alignment section: Header—Contains information about the entire file, such as sample name, sample length, and alignment method. Alignments in the alignments section are associated with specific information in the header section. Alignments—Contains read name, read sequence, read quality, alignment information, and custom tags. The read name includes the chromosome, start coordinate, alignment quality, and the match descriptor string.

The alignments section includes the following information for each or read pair:

- RG: Read group, which indicates the number of reads for a specific sample.
- BC: Barcode tag, which indicates the demultiplexed sample ID associated with the read.
- SM: Single-end alignment quality.
- AS: Paired-end alignment quality.
- NM: Edit distance tag, which records the Levenshtein distance between the read and the reference.
- XN: Amplicon name tag, which records the amplicon tile ID associated with the read.

```
bamfiles <-list.files(path=".",pattern = "*.BAM$")
```

Step 7 : Feature Counts

FeatureCounts is a function under RSubread used in bioinformatics for counting the number of reads (from RNA sequencing) that map to genomic features such as genes, exons, or genomic regions. It is part of the Subread package.

This step is to count the number of reads that map to each gene using the "featureCounts" function.

This step is crucial because it provides the raw data for differential expression analysis.

```
fc <- featureCounts(files=bamfiles,annot.inbuilt="mm10")

names(fc)

fc$stat

head(fc$annotation)

write.csv(fc$counts, file = "path/to/your/directory")

write.csv(fc$annotation, file = "path/to/your/directory")

write.csv(fc$targets, file = "path/to/your/directory")
```

Step 8 : Loading Sample Information from the CSV File

Reading sample information provides the metadata needed for differential expression analysis, linking read counts to experimental conditions and ensuring accurate statistical analysis. A sample file needs to be created with the information given in the image and save the file with .csv extension. Sample Info is used to describe the experimental condition associated with each sample. Conditions being control and treatment.

Control : Untreated or baseline state

Treatment : Manipulated for experiment

```
sampleInfo <- read.table("sample_info.csv", header=TRUE, sep=",", row.names=1)
```

STEP 9: Differential Gene Expression

Differential expression is the process of determining the differences in gene expression levels between different biological conditions. It identifies which set of genes are expressed at different levels under varying experimental conditions, such as treatments, time points, or disease states.

edgeR stores data in a simple list-based data object called a DGEList. This type of object is easy to use because it can be manipulated like any list in R.

dgeFull is a variable here in which we are saving DGEList.

```
dgeFull                                                <-DGEList(counts=fc$counts,
gene=fc$annotation[,c("GeneID","Length")],group=sampleInfo$condition)
```

The code below filters out genes that have zero counts across all samples.

```
dgeFull <- DGEList(dgeFull$counts[apply(dgeFull$counts, 1, sum) != 0,
],group=dgeFull$samples$group)

head(dgeFull$counts)
```

The normLibSizes function normalizes the library sizes in such a way to minimize the log-fold changes between the samples for most genes. The default method for computing these scale factors uses a trimmed mean of M-values (TMM) between each pair of samples.

TMM stands for Trimmed Mean of M-values. It is a normalization method that adjusts for compositional differences between samples. TMM aims to make the majority of genes have similar expression levels across samples by trimming extreme values and calculating a scaling factor for each sample.

We call the product of the original library size and the scaling factor the **effective library size**, i.e., the normalized library size. The effective library size replaces the original library size in all downsteam analyses.

```
dgeFull <- calcNormFactors(dgeFull, method="TMM")

dgeFull$samples

head(dgeFull$counts)

eff.lib.size <- dgeFull$samples$lib.size*dgeFull$samples$norm.factors

normCounts <- cpm(dgeFull)
```

The pseudo-counts represent the equivalent counts would have been observed had the library sizes all been equal, assuming the fitted model. The pseudo-counts are computed for a specific purpose, and their computation depends on the experimental design as well as the library sizes.

```
pseudoNormCounts <- log2(normCounts + 1)
```

The function plotMDS draws a multi-dimensional scaling plot of the RNA samples in which distances correspond to leading log-fold-changes between each pair of RNA samples. The leading log-fold-change is the average (root-mean-square) of the largest absolute log-fold changes between each pair of samples.

```
plotMDS(pseudoNormCounts)
```

estimateCommonDisp Estimate Common Negative Binomial Dispersion by Conditional Maximum Likelihood

The estimateCommonDisp function in the edgeR package is used to estimate a common dispersion parameter for a set of counts following a negative binomial distribution. This helps in accurately modeling the data and improving the reliability of downstream analyses, such as identifying differentially expressed genes.

```
dgeFull <- estimateCommonDisp(dgeFull)
```

EstimateTagwiseDisp Estimate Empirical Bayes Tagwise Dispersion Values

The estimateTagwiseDisp function refines the RNA-seq data analysis by providing gene-specific dispersion estimates using the empirical Bayes method. This process enhances the accuracy of differential expression analysis by accounting for the unique variability of each gene.

```
dgeFull <- estimateTagwiseDisp(dgeFull)

dgeFull
```

The exact test is a statistical method used in RNA-seq data analysis to identify differentially expressed genes between experimental groups. This test compares the read counts for each gene between groups, taking into account the estimated dispersion.By performing the exact test, one can determine which genes show statistically significant differences in expression between conditions, providing insights into the underlying biological processes and responses.

```
dgeTest <- exactTest(dgeFull)

dgeTest

write.csv(dgeTest, file = "path/to/your/directory")

hist(dgeTest$table[,"PValue"], breaks=50)

hist(dgeTestFilt$table[,"PValue"], breaks=50)
```

Bulk RNA-seq analysis using Python

Pyrpipe Setup and Usage Guide

Introduction

This guide will walk you through setting up a Conda environment for using **Pyrpipe** and related bioinformatics tools. Pyrpipe is a Python package designed for the reproducible analysis of RNA-Seq data. It integrates various popular RNA-Seq analysis tools into a streamlined workflow.

Prerequisites

We need Miniconda for this. Miniconda is a minimal installer for Conda, which is a package management and environment management system. It provides a lightweight alternative to the full Anaconda distribution. Anaconda is a comprehensive open-source distribution of Python and R designed specifically for scientific computing, data analysis, and machine learning. It includes a wide range of pre-installed packages and tools for data science and scientific computing, making it a popular choice for many researchers and developers.

Ensure you have Conda installed on your system. If you don't have Conda, you can install it by following the instructions on the Conda website- https://docs.anaconda.com/miniconda/

Step 1: Configure Conda Channels

Before installing Pyrpipe, ensure that Conda channels are added in the correct order:

bash

```
conda config --add channels defaults

conda config --add channels bioconda

conda config --add channels conda-forge
```

This will ensure that the necessary packages are sourced from the correct channels.

Step 2: Create a New Conda Environment

Next, create a new Conda environment named `pyrpipe` with Python 3.8:

bash

```
conda create -n pyrpipe python=3.8

conda activate pyrpipe
```

Step 3: Install Pyrpipe and Required Tools

With the environment activated, install Pyrpipe along with the required bioinformatics tools:

bash

```
conda install -c bioconda pyrpipe star=2.7.7a sra-tools=2.10.9
stringtie=2.1.4 trim-galore=0.6.6 orfipy=0.0.3 salmon=1.4.0
```

This command installs the following tools:

- **Pyrpipe**: Main Python package for RNA-Seq analysis.
- **STAR**: RNA-Seq read aligner.
- **SRA-Tools**: Tools for working with sequence data from NCBI's Sequence Read Archive.
- **StringTie**: Transcriptome assembly and quantification.
- **Trim Galore**: A wrapper around Cutadapt and FastQC for quality control and trimming.
- **Orfipy**: Tool for finding open reading frames (ORFs).
- **Salmon**: Quantification of transcript abundance.

Step 4: Download Test Sample Files

To get started with Pyrpipe, you'll need a sample dataset. Run the following commands in your bash terminal to download the necessary files:

bash

```
wget                                ftp://ftp.ensemblgenomes.org/pub/release-
→46/plants/fasta/arabidopsis_thaliana/dna/Arabidopsis_thaliana.TAIR10.d→na.
toplevel.fa.gz

gunzip Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.gz

wget                                ftp://ftp.ensemblgenomes.org/pub/release-
46/plants/gtf/arabidopsis_thaliana/Arabidopsis_thaliana.TAIR10.46.gtf.gz

gunzip Arabidopsis_thaliana.TAIR10.46.gtf.gz
```

These commands download the reference genome and GTF file for *Arabidopsis thaliana* from the Ensembl Plants database.

Simple RNA-Seq Processing Pipeline

RNA-Seq processing with Pyrpipe is straightforward. The following Python script provides a basic example of using Pyrpipe on publicly available RNA-Seq data:

python

```
#define some variables
run_id = 'SRR976159'
working_dir = 'example_output'
```

```
gen = 'Arabidopsis_thaliana.TAIR10.dna.toplevel.fa'
ann = 'Arabidopsis_thaliana.TAIR10.46.gtf'
star_index = 'star_index/athaliana'

#initialize objects
#creates a STAR object to use with threads
star = mapping.Star(index=star_index, genome=gen, threads=4)

#use Trim Galore for trimming
trim_galore = qc.Trimgalore()

#Stringtie for assembly
stringtie = assembly.Stringtie(guide=ann)

#create SRA object; this will download FASTQ if it doesn't exist
srr_object = sra.SRA(run_id, directory=working_dir)

#create a pipeline using the objects
srr_object.trim(trim_galore).align(star).assemble(stringtie)

#The assembled transcripts are in srr_object.gtf
print('Final result', srr_object.gtf)
```

Explanation of the Code

1. **Imports the required Pyrpipe modules.**
2. **Lines 3 to 7** define variables for reference files, the output directory, and the STAR index. The output directory will be used to store the downloaded RNA-Seq data and will be the default directory for all the results.
3. **Creates a STAR object**: It takes the index and genome as parameters. It will automatically verify the index, and if an index is not found, it will use the genome to build one and save it to the index path provided.
4. **Creates a Trimgalore object**.
5. **Creates a Stringtie object**.
6. **Creates an SRA object**: This represents the RNA-Seq data. If the raw data is not available on disk, it auto-downloads it via `fasterq-dump`.
7. **Pipeline creation**:
    - `trim()`: Takes a QC type object and performs trimming via `qc.perform_qc` method. The trimmed FASTQ files are updated in the SRA object.
    - `align()`: Takes a mapping type object and performs alignment via `mapping.perform_alignment` method. The resulting BAM file is stored in `SRA.bam_path`.
    - `assemble()`: Takes an assembly type object and performs assembly via `mapping.perform_assembly` method. The resulting GTF file is stored in `SRA.gtf`.

This pipeline downloads FASTQ files from NCBI-SRA, uses Trim Galore for trimming, STAR for alignment to the reference genome, and Stringtie for assembly.