Perform differential expression analysis on fibrotic and non-fibrotic patients under 4 different treatments on HPC

Author

Sam (Cheng-Hsiang) Lu

Email: Cheng-Hsiang.Lu@cshs.org

Mentor

David Casero

Email: David.Casero@cshs.org

Background

Inflammatory bowel diseases(IBD)

Inflammatory bowel diseases (IBD) are a group of chronic conditions that cause inflammation and damage to the digestive tract. The two main types of IBD are Crohn's Disease and Ulcerative Colitis.

Crohn's disease can affect any part of the digestive tract, like small or large intestine, and can cause symptoms such as abdominal pain, diarrhea, weight loss, and fatigue. It can also cause complications such as fistulas (abnormal

connections between different parts of the intestine) and strictures (narrowing of the intestine).

Ulcerative colitis, on the other hand, affects only the colon and rectum and causes symptoms such as bloody diarrhea, abdominal pain, and a frequent need to pass stools. It can also lead to complications such as inflammation of the skin, eyes, and joints.

Both Crohn's disease and ulcerative colitis are chronic conditions, meaning they can last for a lifetime and require ongoing treatment to manage symptoms and prevent complications.

Induced Pluripotent Stem Cells(iPSCs)

Induced pluripotent stem cells (iPSCs) are a type of stem cell that are generated in the laboratory by reprogramming adult cells, such as skin or blood cells, to a pluripotent state. A pluripotent state means that the cells have the potential to develop into any type of cell in the body, just like embryonic stem cells.

iPSCs offer several advantages as they can be generated from the patient's own cells, avoiding issues with immune rejection, ethical concerns and the need for embryos.

iPSCs can be used to study the underlying causes of diseases, test new drugs and therapies, and potentially generate replacement tissues or organs for transplantation.

Aim

In this project, we will be analyzing RNA-seq data from 19 samples, comprising of 10 samples with fibrotic complications and 9 non-fibrotic samples. Each sample has undergone two runs and 4 different treatments(untreated, TGF-B, TNF-A, and TGF-B+TNF-A), resulting in a total of 151 samples(1 library failed). We used induced pluripotent stem cells (iPSC) to differentiate into myofibroblasts and stimulated the system with different signals to observe its development. The objective is to investigate the effect of four different treatments: untreated, TGF-B, TNF-A, and TGF-B+TNF-A on the development of the system. In the end, we will perform differential expression analysis to identify the genes that are differentially expressed in fibrotic and non-fibrotic samples under 4 treatments.

Pipelines

In HPC

Convert 151 Fastq to Fasta files

First, I put all fasq.gz files in one folder and list all fastq files' name in fastqfiles.txt

```
ls *q.gz > fastqfiles.txt
```

Cut redundant suffix "_R1_trimmed" and list all fastq files' name in libraryname.txt and preffix.txt

```
ls *q.gz | cut -f 1 -d '.' | sed 's/_R1_trimmed//g'  >libraryname.txt
```

```
ls *q.gz | cut -f 1 -d '.' | sed 's/_R1_trimmed//g' > preffix.txt
```

Form a table with 3 columns: fastqfiles.txt libraryname.txt preffix.txt

```
paste fastqfiles.txt libraryname.txt preffix.txt > tofastatable.txt
```

Create small-sized fasta-formatted files. To submit this job to the cluster on HPC, you need to read the file, library, and prefix. Once you have done that, run the script "generatefastaFromFastaqz" which will combine the script "DCfastaqTofastaLibraryId.pl". This results in small-sized fasta-formatted files contain only one header and one sequence per read. You can find all scripts in the "scripts" folder.

```
cat tofastatable.txt | awk {print}' | while read file library preffix ;
do qsub -cwd -o $PWD -e $PWD -l h_data=2048M,h_rt=8:00:00
$HOME/scripts/generatefastaFromFastaqz $file $library $preffix

done
```

This is what each fasta-formatted file would look like

Generate auxiliary files and directories for each sample

Put a list of names of all fasta files in the directory and save them in a text file named “fastafiles.txt”

```
ls *fasta.gz > fastafiles.txt
```

Cut the redundant suffix “.fasta.gz” from the names of all fasta files and generate a new list of file names with the suffix removed in a text file named “targetdirectories.GTF.txt”

```
cat fastafiles.txt |sed 's/.fasta.gz//g' > targetdirectories.GTF.txt
```

Create a separate directory for each sample listed in “fastafiles.txt”

```
cat fastafiles.txt |while read line ; do mkdir
${line/.fasta\.gz/GTFpass1/} ; done
```

Form the submission script called “sendmyof”

Add a shebang line at the beginning of your script file named “sendmyof” to indicate the interpreter that should be used to execute the script

```
echo '#!/bin/bash/' > sendmyof
```

The command below runs the “generatesendscriptSingleGTFParam” script with several input parameters to map the RNA-seq data with STAR. The input parameters include the list of target directories containing the input data (“targetdirectories.GTF.txt”), the subdirectory name (“GTFpass1”), a parameter file containing settings for STAR alignment (“Parameters.txt”), a prefix for output files (“myof”), the path to the STAR index directory (“/home/luc/RNASEQ_MASTER/Hsapiens/GRC38/INDEXES/GRCh38.primary.33.basicselected.STAR2.7.3a/”), the path to the input data directory (“/home/luc/iPSC/MYOFIBROBLAST/”), the amount of free memory to use (“mem_free=32G”), and the number of threads to use (“8”). In the end, it will generate a “processLaneSingleGTFParam” file and run the STAR package in each sample’s folder.

```
./generatesendscriptSingleGTFParam targetdirectories.GTF.txt GTFpass1
Parameters.txt myof
/home/luc/RNASEQ_MASTER/Hsapiens/GRC38/INDEXES/GRCh38.primary.33.basics
elected.STAR2.7.3a/ /home/luc/iPSC/MYOFIBROBLAST/ mem_free=32G 8 >>
sendmyof
```

Change sendmyof into executable mode and run sendmyof

```
chmod a+x sendmyof
. sendmyof
```

It will take less than one day to run through 151 samples and generate each sample a folder which contain every output from STAR

Create a table summarizing the mapping statistics for each sample

Change directory into one sample file which ends with “GTFpass1”. Extract the first column from the mapping statistics file and store it in “temp2.txt”

```
grep "|" 008iP22TGFbM_S71GTFpass1/008iP22TGFbM_S71GTFpass1Log.final.out
| cut -f 1 -d "|" | sed 's/^  *//g' | awk 'NR>3 {print}' > temp2.txt
```

The first column from the mapping statistics file

Create an empty temporary file for storing intermediate results

```
rm tempprev.txt
touch tempprev.txt
```

Extract the total mapped reads from each subsequent mapping statistics file and combine with previous results

```
ls *pass1/*final.out | while read line ; do
grep "|" $line | cut -f 2  > temp.txt
paste tempprev.txt temp.txt > tempnew.txt
mv tempnew.txt tempprev.txt
done
```

Remove the first column and write the final results to a file called

“mappingstatsFirstpass.txt”

```
cut -f 2- tempprev.txt | awk 'NR>3 {print}'  > tempnew.txt
mv tempnew.txt tempprev.txt
paste temp2.txt tempprev.txt > mappingstatsFirstpass.txt
```

The mappingstatsFirstpass.txt would look like this

Compile Counts

Generate a directory called “COUNTS” and copy all gene count files to this folder.

```
mkdir COUNTS
cp *pass1/*PerGene* COUNTS/
```

Clean filenames

```
ls *tab|while read line ; do mv $line ${line/GTFpass1ReadsPerGene.out/}
; done
```

For each count file, extract and create the five count tables

```
ls *.tab | while read line ; do
echo $line
cat $line | awk 'NR==3{print}' | cut -f 2- > ${line/tab/nofeature\.tab}
cat $line | awk 'NR==4{print}' | cut -f 2- > ${line/tab/ambiguous\.tab}
cat $line | awk 'NR>4{print}' | cut -f 2 > ${line/tab/nostrand\.tab}
cat $line | awk 'NR>4{print}' | cut -f 3 > ${line/tab/sense\.tab}
cat $line | awk 'NR>4{print}' | cut -f 4 > ${line/tab/antisense\.tab}
done
```

Make a Geneid list from one of the count tables as “countsannot_GRCh38.primary.Selected.Geneid.txt”

```
ls 008iP22TGFbM_S71.tab | head -1 | while read line; do
cut -f 1 $line | awk 'NR>4{print}' >
countsannot_GRCh38.primary.Selected.Geneid.txt
done
```

Create a file listing the names of all samples as “RBarretTNFATGFBsamples.txt”

```
ls *.sense.tab | sed 's/.sense.tab//g' | tr -s " " "\n" | sed 's/_1//g'
> RBarretTNFATGFBsamples.txt
```

Make count tables for sense, anti-sense, nostrand, ambiguous, and nofeature reads

```
# Combine all sense counts into RBarretTNFATGFB_sense.ALL.cnt
paste *.sense.tab > RBarretTNFATGFB_sense.ALL.cnt

# Combine all antisense counts into RBarretTNFATGFB_antisense.ALL.cnt
paste *.antisense.tab > RBarretTNFATGFB_antisense.ALL.cnt

# Combine all nostrand counts into RBarretTNFATGFB_nostrand.ALL.cnt
paste *.nostrand.tab > RBarretTNFATGFB_nostrand.ALL.cnt

# Combine all ambiguous counts into RBarretTNFATGFB_ambiguous.cnt
cat *ambiguous.tab > RBarretTNFATGFB_ambiguous.cnt

# Combine all nofeature counts into RBarretTNFATGFB_nofeature.cnt
cat *nofeature.tab > RBarretTNFATGFB_nofeature.cnt
```

I am going to use “RBarretTNFATGFB_antisense.ALL.cnt” file for the further analysis

In MATLAB

Transfer data to your local laptop

Read counts, annotation, and mappability

```
RBarretTNFATGFBCnt = textread('RBarretTNFATGFB_antisense.ALL.cnt','');
RBarretsamplesTNFATGFB = textread('RBarretTNFATGFBsamples.txt','%s');
RBarretsampleskeysTNFATGFB = textread('samplekeys_Sam.txt','%s');

% Calculates the sum of the counts in RBarretTNFATGFBCnt, divides the
result by 1000000, and rounds the result to the nearest integer.
RBarretTNFATGFBmeta_seqdepth=round(sum(RBarretTNFATGFBCnt)/1000000);
```

Import annotation

You can find these annotation in the “mappability and R code” folder

```
Gencode_33_Selected_MappSS=textread('mappability and R
code/gencode.v33.Selected.ReadsPerGene.out.MappSS.txt','');
Gencode_33_Selected_MappUS=textread('mappability and R
code/gencode.v33.Selected.ReadsPerGene.out.MappUS.txt','');
Gencode_33_Selected_Geneid=textread('mappability and R
code/gencode.v33.annotation.Selected.geneid.txt','%s\n');
Gencode_33_Selected_Biotype=textread('mappability and R
code/gencode.v33.annotation.Selected.biotype.txt','%s\n');
Gencode_33_Selected_Genename=textread('mappability and R
code/gencode.v33.annotation.Selected.genename.txt','%s\n');
```

Compile counts

First, initializes a new variable called RBarretTNFATGFBTPM with the same count data as RBarretTNFATGFBCnt. Then, iterates over each gene in the count data matrix. For each gene, the corresponding row in RBarretTNFATGFBTPM is updated by dividing the count data by the read counts from the “Gencode_33_Selected_MappSS”, multiplying by 1000, and storing the result in RBarretTNFATGFBTPM.

Finally, iterates over each sample in the TPM data matrix. For each sample, the corresponding column in RBarretTNFATGFBTPM is updated by dividing the values in the column by the sum of the values in the column, multiplying by 1,000,000, and storing the result in RBarretTNFATGFBTPM. This step **normalizes the TPM values** across samples and scales the resulting values to TPM.

```
RBarretTNFATGFBTPM = RBarretTNFATGFBCnt;
for i=1:size(RBarretTNFATGFBCnt,1)
% Divids the gene count matrix RBarretTNFATGFBCnt by
Gencode_33_Selected_MappSS matrix, which is the sum of the transcript
length of each gene
RBarretTNFATGFBTPM(i,:) =
RBarretTNFATGFBCnt(i,:)/Gencode_33_Selected_MappSS(i)*1000;
end
% Sets any NaN or Inf values resulting from the normalization process to
0
RBarretTNFATGFBTPM(isnan(RBarretTNFATGFBTPM)) = 0;
RBarretTNFATGFBTPM(isinf(RBarretTNFATGFBTPM)) = 0;
for i=1:size(RBarretTNFATGFBTPM,2)
% Scale the TPM values so that the sum of expression values across each
sample of the matrix is equal to 1,000,000. This ensures that the
expression values are comparable across different samples and allows
meaningful comparisons of gene expression levels between different
samples.
RBarretTNFATGFBTPM(:,i) =
RBarretTNFATGFBTPM(:,i)/sum(RBarretTNFATGFBTPM(:,i))*1000000;
end
```

Make my first dendrogram

Make a dendrogram to visualize the relationships among samples in the RBarretTNFATGFB dataset based on their gene expression profiles.

1. I generates a random selection of 1,000 genes from the TPM data matrix.
2. Calculates the pairwise distances between the selected genes.
3. Creates a for loop that iterates 9,999 times. For each iteration, a new random selection of 1,000 genes is generated, and the pairwise distances between these genes are added to the previous ‘thisdist’ calculation.
4. Converts the one-dimensional distance vector ‘thisdist’ into a distance matrix ‘thisdistmat’ using the ‘squareform’ function.
5. Generates a hierarchical clustering tree based on the distance matrix ‘thisdistmat’.

Overall, I perform a clustering analysis on a subset of genes in the RBarretTNFATGFB dataset to visualize the relationships among samples based

on their gene expression profiles.

```
thisrand = unique(randi([1 size(RBarretTNFATGFBTPM,1)],1,1000));
thisdist = pdist(RBarretTNFATGFBTPM(thisrand,:)');
for i=1:9999
thisrand = unique(randi([1 size(RBarretTNFATGFBTPM,1)],1,1000));
thisdist = thisdist+pdist(RBarretTNFATGFBTPM(thisrand,:)');
end
thisdistmat = squareform(thisdist/10000);
thistree = seqlinkage(thisdistmat,'average', RBarretsamplesTNFATGFB)
plot(thistree, 'ORIENTATION', 'top')
```

All biotypes counts percents

This part of codes is performing all biotypes counts percents across multiple samples.

```
% Used the unique function and stored the allbiotypes variable.
allbiotypes = unique(Gencode_33_Selected_Biotype);

% Created A cell array allbiotypeslength to store the lengths of each
biotype name.
allbiotypeslength = cell(length(allbiotypes),1);

% Two new matrices, allbiotypescounts and allbiotypescountspercents, are
initialized with zeros. These matrices have dimensions (number of unique
biotypes) x (number of samples in the TPM data). They will be used to
store the number of reads (counts) and the percentage of total reads
(%TPM) for each biotype in each sample.
allbiotypescounts =
zeros(length(allbiotypes),size(RBarretTNFATGFBCnt,2));
allbiotypescountspercents =
zeros(length(allbiotypes),size(RBarretTNFATGFBCnt,2));

for i=1:length(allbiotypes)
%  Found all the indices of Gencode_33_Selected_Biotype that match the
current biotype. Then, returned a vector of **indices** where the
biotype occurs in Gencode_33_Selected_Biotype.
temp = strmatch(allbiotypes{i}, Gencode_33_Selected_Biotype);
```

```
% Stored the length of the temp vector represents the number of genes
with the current biotype in the Gencode_33_Selected_Biotype.
allbiotypeslength{i} = length(temp);
if length(temp)>1
% Sum the expression values for all genes with the current biotype
across all samples. The resulting sums are stored in the corresponding
row of the allbiotypescounts matrix.
allbiotypescounts(i,:) =
sum(RBarretTNFATGFBCnt(strmatch(allbiotypes{i},
Gencode_33_Selected_Biotype),:));
allbiotypescountspercents(i,:) =
allbiotypescounts(i,:)./sum(RBarretTNFATGFBCnt)*100;
end
end

dlmwrite('allbiotypescountspercents.txt',
allbiotypescountspercents,'delimiter','\t')
writetable(cell2table(allbiotypes),'allbiotypes.txt','WriteVariableName
s',0)
```

I Formulated a spreadsheet by using “allbiotypes.txt” and “allbiotypescountspercents.txt” on Excel and add min, max, and average for each biotype. You can find my spreadsheet here. As you can see, protein_coding genes have the average of 98.65% among other biotypes, which is what we want.

Keep only protein coding genes

We are going to keep only protein coding genes for the next step.

```
allbiotypes=unique(Gencode_33_Selected_Biotype);
% Found the 15th unique value, which is protein_coding, of
Gencode_33_Selected_Biotype in the array proteincodingindx.
proteincodingindx = strmatch(allbiotypes{15},
Gencode_33_Selected_Biotype);
biotypeindx = proteincodingindx;

% Created an array additionalgenes contains the indices of genes that
have certain prefixes such as 'MT-', 'H1', 'H2', 'H3', 'H4', 'RPL', or
'RPS' in their names.
additionalgenes = [strmatch('MT-',Gencode_33_Selected_Genename) ;
strmatch('H1',Gencode_33_Selected_Genename);
strmatch('H2',Gencode_33_Selected_Genename);
strmatch('H3',Gencode_33_Selected_Genename);
strmatch('H4',Gencode_33_Selected_Genename) ;
strmatch('RPL',Gencode_33_Selected_Genename) ;
strmatch('RPS',Gencode_33_Selected_Genename)];

% Created an array nonadditionalgenes with the same length as the
Gencode_33_Selected_Genename array.
nonadditionalgenes = 1:length(Gencode_33_Selected_Genename);
% Removed the indices of genes in additionalgenes from the
nonadditionalgenes array.
nonadditionalgenes(additionalgenes) = [];

% mappableindx containing the indices of elements in the
Gencode_33_Selected_MappSS array that are greater than 50.
mappableindx = find(Gencode_33_Selected_MappSS>50);

% a new variable finalIndexGeneric which is the intersection of three
other variables: biotypeindx, nonadditionalgenes, and mappableindx.
finalIndexGeneric =
intersect(biotypeindx,intersect(nonadditionalgenes,mappableindx));
% finds the indices of rows in RBarretTNFATGFBCnt that have a sum
greater than 150.
countindx = find(sum(RBarretTNFATGFBCnt')'>150);

% updates finalIndexGeneric to be the intersection of finalIndexGeneric
and countindx.
finalIndexGeneric=intersect(finalIndexGeneric,countindx);
```

```
% creates a new variable RBarretTNFATGFBCnt_GMask which is a subset of
RBarretTNFATGFBCnt corresponding to the rows indexed by
finalIndexGeneric.
RBarretTNFATGFBCnt_GMask = RBarretTNFATGFBCnt(finalIndexGeneric,:);

Gencode_33_Selected_Geneid_GMask =
Gencode_33_Selected_Geneid(finalIndexGeneric);
Gencode_33_Selected_Genename_GMask =
Gencode_33_Selected_Genename(finalIndexGeneric);
Gencode_33_Selected_MappSS_GMask =
Gencode_33_Selected_MappSS(finalIndexGeneric);
Gencode_33_Selected_MappUS_GMask =
Gencode_33_Selected_MappUS(finalIndexGeneric);
```

```
% normalizes the expression data like we do previously
RBarretTNFATGFBExpression_GMask = RBarretTNFATGFBCnt_GMask;
for i=1:size(RBarretTNFATGFBExpression_GMask,2)
RBarretTNFATGFBExpression_GMask(:,i) =
RBarretTNFATGFBCnt_GMask(:,i)/sum(RBarretTNFATGFBCnt_GMask(:,i))*100000
0;
end
for i=1:size(RBarretTNFATGFBExpression_GMask)
RBarretTNFATGFBExpression_GMask(i,:) =
RBarretTNFATGFBExpression_GMask(i,:)/Gencode_33_Selected_MappSS_GMask(i
)*1000;
end
RBarretTNFATGFBExpression_GMask(isnan(RBarretTNFATGFBExpression_GMask))
= 0;
RBarretTNFATGFBExpression_GMask(isinf(RBarretTNFATGFBExpression_GMask))
= 0;

% RBarretTNFATGFBCPM_GMask contains the expression data normalized only
by CPM, using the same normalization method as the code above.
RBarretTNFATGFBCPM_GMask = zeros(size(RBarretTNFATGFBCnt_GMask));
for i=1:size(RBarretTNFATGFBCnt_GMask,2)
RBarretTNFATGFBCPM_GMask(:,i) =
RBarretTNFATGFBCnt_GMask(:,i)/sum(RBarretTNFATGFBCnt_GMask(:,i))*100000
0;
end

% RBarretTNFATGFBTPM_GMask contains the expression data normalized only
by TPM.
RBarretTNFATGFBTPM_GMask = RBarretTNFATGFBCnt_GMask;
for i=1:size(RBarretTNFATGFBCnt_GMask,1)
RBarretTNFATGFBTPM_GMask(i,:) =
RBarretTNFATGFBCnt_GMask(i,:)/Gencode_33_Selected_MappSS_GMask(i)*1000;
end
RBarretTNFATGFBTPM_GMask(isnan(RBarretTNFATGFBTPM_GMask)) = 0;
RBarretTNFATGFBTPM_GMask(isinf(RBarretTNFATGFBTPM_GMask)) = 0;
for i=1:size(RBarretTNFATGFBTPM_GMask,2)
RBarretTNFATGFBTPM_GMask(:,i) =
RBarretTNFATGFBTPM_GMask(:,i)/sum(RBarretTNFATGFBTPM_GMask(:,i))*100000
0;
end
```

Make dendrogram with only protein coding genes

Perform hierarchical clustering on a subset of the gene expression data stored in the variable RBarretTNFATGFBTPM_GMask with only protein coding genes.

```
thisrand = unique(randi([1 size(RBarretTNFATGFBTPM_GMask,1)],1,1000));
thisdist = pdist(RBarretTNFATGFBTPM_GMask(thisrand,:)');
for i=1:9999
thisrand = unique(randi([1 size(RBarretTNFATGFBTPM_GMask,1)],1,1000));
thisdist = thisdist+pdist(RBarretTNFATGFBTPM_GMask(thisrand,:)');
end
thisdistmat = squareform(thisdist/10000);
thistree = seqlinkage(thisdistmat,'average',
RBarretsampleskeysTNFATGFB)
plot(thistree,'ORIENTATION','top')
```

We can tell that basically the plot is clustered by their treatments.

What is the percent of the top 100 genes

Calculates the top 100 expressed genes in each sample based on their transcript per million (TPM) values in the RBarretTNFATGFBTPM_GMask matrix.

```
% iterates over 151 samples, it first sorts the TPM values of all genes
in descending order and stores the indices of the sorted genes in y. The
top 100 expressed genes in the sample are obtained by selecting the
first 100 indices in y, and these indices are appended to a running list
of all top 100 indices yall.
yall=[];
for i=1:151
[x y]=sort(RBarretTNFATGFBTPM_GMask(:,i),'descend');
yall=unique([y(1:100); yall]);
top100percent(i)=sum(RBarretTNFATGFBTPM_GMask(y(1:100),i))/1000000;
end
```

After calculating the sum of the percent of the top 100 genes, it is 58.25%.

In the end, we store the **Gencode_33_Selected_Geneid_GMask.txt, Gencode_33_Selected_Genename_GMask.txt,**

Gencode_33_Selected_MappSS_GMask.txt, RBarretTNFATGFBTPM_GMask.txt, and RBarretTNFATGFBCnt_GMask.txt for ours further analysis in R.

```
writetable(cell2table(Gencode_33_Selected_Geneid_GMask),'Gencode_33_Sel
ected_Geneid_GMask.txt','WriteVariableNames',0)
writetable(cell2table(Gencode_33_Selected_Genename_GMask),'Gencode_33_S
elected_Genename_GMask.txt','WriteVariableNames',0)
dlmwrite('Gencode_33_Selected_MappSS_GMask.txt',
Gencode_33_Selected_MappSS_GMask,'delimiter','\t')
dlmwrite('RBarretTNFATGFBTPM_GMask.txt',
RBarretTNFATGFBTPM_GMask,'delimiter','\t')
dlmwrite('RBarretTNFATGFBCnt_GMask.txt',
RBarretTNFATGFBCnt_GMask,'delimiter','\t')
```

In R

Install packages

We have to install BiocManager, BiocLite, IHW, DESeq2, and ggplot2. Then, read in RBarretTNFATGFBCnt_GMask.txt, RBarretTNFATGFBsamples.txt, samplekeys_Sam.txt, and Gencode_33_Selected_Genename_GMask.txt.

```
setwd("/Users/LuC/Desktop/Cedars-
Sinai/PROJECTS/IBD_RNASeq/RBARRETTNFATGFB/")#setwd("/Users/samuellu/Des
ktop/Cedars-Sinai/PROJECTS/IBD_RNASeq/RBARRETTNFATGFB/")if
(!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")#BiocManager::install("BiocLite")#BiocMa
nager::install("IHW")#BiocManager::install("DESeq2")#install.packages("
ggplot2")library(DESeq2)library(IHW)library(ggplot2)library(ggrepel)RBa
rretTNFATGFBCntGMask =
as.matrix(read.table("RBarretTNFATGFBCnt_GMask.txt"))sampleNameTNFATGFB
= as.matrix(read.table("RBarretTNFATGFBsamples.txt"))sampleKeyTNFATGFB
= as.matrix(read.table("samplekeys_Sam.txt"))genenames =
as.matrix(read.table("Gencode_33_Selected_Genename_GMask.txt"))```

#### Generate samplekeys_Sam.tab

I have to separate samplekeys_Sam.txt by "\_" to get samplekeys\_Sam.tab before next step. Here are my code in terminal.
```

In terminal

Create an empty file to store the output

touch samplekeys_Sam.tab

Loop over the sample names and split them by "_"

for sample in $(cat samplekeys_Sam.txt); do IFS=_ read -r col1 col2 col3 col4 col5 <<< "$sample" echo -e “$col1\t$col2\t$col3\t$col4\t$col5” >> samplekeys_Sam.tab done

Generate a sampleTableTNFATGFB

The sampleTableTNFATGFB contains Treatment, Line, Pheno, Sex, Pass, Factor, and Batch.

```sampleTableTNFATGFB = read.table("samplekeys_Sam.tab")rownames(sampleTableTNFATGFB)<-sampleKeyTNFATGFBcolnames(sampleTableTNFATGFB)<-c("Treatment","Line","Pheno","Sex","Pass")sampleTableTNFATGFB$Factor <-paste(sampleTableTNFATGFB$Treatment,sampleTableTNFATGFB$Pheno,sep="_")# concatenating the "Line" and "Pass" columns with an underscore separatorsampleTableTNFATGFB$Batch <-paste(sampleTableTNFATGFB$Line,sampleTableTNFATGFB$Pass,sep="_")colnames(RBarretTNFATGFBCntGMask) <-sampleKeyTNFATGFBwrite.table(sampleTableTNFATGFB,file="sampleTableTNFATGFB.txt", sep = "\t", col.names = FALSE)```

```

DESeq2 package

The RBarretTNFATGFBCntGMaskBatch is created with the **Batch** information specified in the design formula, while the RBarretTNFATGFBCntGMaskFactor is created with the **treatment and phenotype** information specified in the design formula.

Next, the DESeq function is used to estimate size factors and dispersion values for the DESeqDataSet objects.

Finally, the varianceStabilizingTransformation function is used to perform variance stabilizing transformation on the DESeqDataSet objects. This transformation is important for reducing the effect of noise and heteroscedasticity in the data, making it more suitable for downstream analyses such as differential gene expression analysis.

```RBarretTNFATGFBCntGMaskBatch <- DESeqDataSetFromMatrix(RBarretTNFATGFBCntGMask, colData= sampleTableTNFATGFB,design= ~Batch)RBarretTNFATGFBCntGMaskBatch <- DESeq(RBarretTNFATGFBCntGMaskBatch)RBarretTNFATGFBCntGMaskFactor <- DESeqDataSetFromMatrix(RBarretTNFATGFBCntGMask, colData= sampleTableTNFATGFB,design= ~Factor)RBarretTNFATGFBCntGMaskFactor <- DESeq(RBarretTNFATGFBCntGMaskFactor)RBarretTNFATGFBCntGMaskBatch_vsd <- varianceStabilizingTransformation(RBarretTNFATGFBCntGMaskBatch,blind=FALSE)RBarretTNFATGFBCntGMaskFactor_vsd <- varianceStabilizingTransformation(RBarretTNFATGFBCntGMaskFactor,blind=FALSE)```

PCA

```# performs principal component analysis (PCA) on the variance-stabilized counts data pcabatch <- prcomp(t(assay(RBarretTNFATGFBCntGMaskBatch_vsd)))# gives the percentage of variance explained by each principal componentpercentVarbatch <- round(100*pcabatch$sdev^2/sum(pcabatch$sdev^2))# pcabatch$rotation is a matrix containing the loadings of the principal components.aloadbatch <- abs(pcabatch$rotation)
# normalizes the loadings in aloadbatch so that each column (i.e., PC) sums to 1. aloadrelativebatch <- sweep(aloadbatch, 2, colSums(aloadbatch), "/")# pcabatch$x is a matrix containing each

```
sample's coordinate on each principal componentpcabatchALL <-
pcabatch$xpcabatchR<- cbind(pcabatchALL,sampleTableTNFATGFB)# centers
the PC1 scores in pcabatchR to have a mean of 0. This is done so that
the PC1 variable can be used as a covariate in the subsequent
differential expression analysis.pcabatchR$PC1 <- scale(pcabatchR$PC1,
center = TRUE)RBarretTNFATGFBCntGMaskPC1 <-
DESeqDataSetFromMatrix(RBarretTNFATGFBCntGMask, colData=
pcabatchR,design= ~PC1)RBarretTNFATGFBCntGMaskPC1 <-
DESeq(RBarretTNFATGFBCntGMaskPC1)RBarretTNFATGFBCntGMaskPC1_vsd <-
varianceStabilizingTransformation(RBarretTNFATGFBCntGMaskPC1,blind=FALS
E)```

PCA plots

The first plot shows the relationship between **PC1 and PC2** colored by
Pheno variable, with the point size indicating the **Treatment**
variable.

The second plot is similar to the first, but the data points are colored
by the **Sex** variable, and the point size indicates the **Treatment**
variable.

The third plot shows the relationship between PC2 and PC9 colored by
Pheno variable, with the point size indicating the Treatment variable.

```ggplot(pcabatchR, aes(PC1, PC2, color= Pheno)) +
geom_point(aes(size= Treatment),alpha=0.6,stroke =
3)+geom_point(aes(size= Pheno),color="black",alpha=0.2) +
xlab(paste0("PC1: ",percentVarbatch[1],"% variance")) +
ylab(paste0("PC2: ",percentVarbatch[2],"% variance")) +
geom_text_repel(aes(label = sampleKeyTNFATGFB),size=4,box.padding   =
0.35, point.padding = 0.5,segment.color = 'grey50')+
theme_bw()ggplot(pcabatchR, aes(PC1, PC2, color= Sex)) +
geom_point(aes(size= Treatment),alpha=0.6,stroke =
3)+geom_point(aes(size= Treatment),color="black",alpha=0.2) +
xlab(paste0("PC1: ",percentVarbatch[1],"% variance")) +
ylab(paste0("PC2: ",percentVarbatch[2],"% variance")) +
theme_bw()ggplot(pcabatchR, aes(PC2, PC9, color= Pheno)) +
geom_point(aes(size= Treatment),alpha=0.6,stroke =
3)+geom_point(aes(size= Treatment),color="black",alpha=0.2) +
xlab(paste0("PC2: ",percentVarbatch[2],"% variance")) +
```
```

ylab(paste0("PC9: ",percentVarbatch[9],"% variance")) + geom_text_repel(aes(label = sampleKeyTNFATGFB),size=4,box.padding = 0.35, point.padding = 0.5,segment.color = 'grey50')+ theme_bw()```

In the PCA plot, there are 4 different groups separate by treatment. We have the CC group(untreated) at the right corner, the TG group(TGFB) at the bottom, TN group(TNFA) at the right corner, and the TT group(TGFB + TNFA) at the top. There are separate by PC1(19%).
A series of bar plots (one for each principal component)

Each bar plot represents the loadings of all samples on a given principal component.
```# The resulting vector coul will contain 12 colors from the "Set3" palette.library(RColorBrewer)coul <- brewer.pal(12, "Set3")# generates colors for a plot based on the batch variablecolors=pcabatchR$Batchallbatches<-unique(pcabatchR$Batch)for (i in 1:38){ colors[pcabatchR$Batch==allbatches[i]]<-coul[i%%12+1]}thinlines=c(seq(4,72,8),75,seq(83,151,8))thicklines=c(seq(8,72,8),79,seq(87,151,8))# first half of the barplot would be the non-fibrotic group and the second part would be the fibrotic group# The order would be CC, TG, TN,TTsamplesorder=c(4,1,3,2,8,5,7,6,28,25,27,26,32,29,31,30,36,33,35,34,40,37,39,38,52,49,51,50,55,53,55,54,68,65,67,66,72,69,71,70,76,73,75,74,80,77,79,78,84,81,83,82,88,85,87,86,116,113,115,114,120,117,119,118,139,136,138,137,143,140,142,141,12,9,11,10,16,13,15,14,20,17,19,18,24,21,23,22,44,41,43,42,48,45,47,46,60,57,59,58,64,61,63,62,92,89,91,90,96,93,95,94,100,97,99,98,104,101,103,102,108,105,107,106,112,109,111,110,124,121,123,122,128,125,127,126,132,129,131,130,135,133,134,147,144,146,145,151,148,150,149)#create 38 barplots and saving each of them as a PNG filefor (i in 1:38) { filename = paste("PC_",i,".png", sep = "")png(filename)barplot(pcabatchALL[samplesorder,i],col=colors[samplesorder],las=2,xaxt='n',space=0) for (i in 1:length(thinlines)) {   abline(v = thinlines[i], col = "black",lty = 3) } for (i in 1:length(thicklines)) {   abline(v = thicklines[i], col = "black",lty = 1) } abline(v = 72, col = "red",lty = 1)dev.off()}write.csv(aloadrelativebatch,file="aloadrelativeMask_batchmodel_filtered.csv")write.csv(pcabatch$x,file="pca_batchmodel_x.csv")```

A red line is drawn at position 72 in order to separate the non-fibrotic
```

```
group and the fibrotic group. Within each patient, the treatment order
would be CC, TG, TN, TT. The color of each bar represents the batch of
the sample, with a unique color assigned to each batch. The vertical
lines on the plot indicate the position of specific loadings, with thin
and thick lines indicating different positions.

For example, this is PC_1.png. From this plot, you can see that the
highest sxpression is closely related with TNF-α. As for the first
patient, compaire to the contorl(untreated), the TNF-α group is much
higher and the TT group (TGF-b+TNF-α) is not that high.

![](/Pics/PCs/PC_1.png)

#### PCA rank matrix

Takes csv files and converts it to the txt files with the second column
onwards. It does this by first removing the first row using awk,
replacing multiple commas with tabs using tr, and removing the first
column using cut.
```

In Mac terminal

cat aloadrelativeMask_batchmodel_filtered.csv | awk ‘NR>1{print}’ | tr -s “,” “ | cut -f 2- > aloadrelativeMask_batchmodel_filtered.clean.txt cat pca_batchmodel_x.csv | awk ‘NR>1{print}’ | tr -s”," " | cut -f 2- > pca_batchmodel_x.clean.txt

```
Read in the preprocessed data files created in the previous steps and
store them in variables pcabatch_samples and pca_loadings, respectively.
```

In Matlab

pcabatch_samples = textread(‘pca_batchmodel_x.clean.txt’,”); pca_loadings = textread(’aloadrelativeMask_batchmodel_filtered.clean.txt’,”);

```
Sort the three columns of pca_loadings in descending order and store the
sorted values in variables x1, x2, and x3, and the corresponding indices
in y1, y2, and y3.
```

%%%%%%%% PCA SUPP %x = pca_loading number, y = its index [x1 y1]=sort(pca_loadings(:,1),‘descend’); [x2 y2]=sort(pca_loadings(:,2),‘descend’); [x3 y3]=sort(pca_loadings(:,3),‘descend’);

```
Determine the rank of each row in the original order for the first three
principal components and store the ranks in a matrix pcarankmatrix.
```

[x y z]=intersect(1:length(y1),y1); pcarankmatrix(:,1)=z; [x y z]=intersect(1:length(y2),y2); pcarankmatrix(:,2)=z; [x y z]=intersect(1:length(y3),y3); pcarankmatrix(:,3)=z;

% contains the rank of each feature in the original order for the first three principal components dlmwrite(‘pcarankmatrix.txt’, pcarankmatrix,‘delimiter’,’)

```
#### Spreadsheet

By making a spread sheet, we can easily manage our data by a single
glance. I used command line, Excel and R to orginize it.

The first sheet built on Excel contains patients order, patients id,
phenotypes, and sex. You can visit the sheet by clicking [here]
(/spreadsheet/Barret_Myofibroblast_TGFTNF_MASTER.xlsx).

![](/Pics/spreadsheet_patient.png)

<br>

The second sheet includes the names and percentages of all biotypes,
along with their respective minimum, maximum, and average values,
providing us with a comprehensive overview.
```

In terminal

paste allbiotypes allbiotypescountspercents > combine_allbiotypes_percents.txt

In R#

allbiotypes_percentssheet2_1 <- list(“Biotypes”)sheet2_2 <- sampleKeyTNFATGFBcombined_sheet2 <- c(sheet2_1, sheet2_2)combined_spreadsheet2 <- as.matrix(read.table(“combine_allbiotypes_percents.txt”))colnames(combined_spreadsheet2) <- combined_sheet2write.table(combined_spreadsheet2,file=“combined_spreadsheet2.txt”, sep = '', row.names = FALSE)

add their respective minimum, maximum, and average values on Excel

```
![](/Pics/spreadsheet_allbiotypes_percents.png)

<br>

The third sheet includes Genename, Geneid, Mapp, PC1, PC2, PC3, and patient's TPM values.
```

In terminal

paste Gencode_33_Selected_Genename_GMask.txt Gencode_33_Selected_Genename_GMask.txt Gencode_33_Selected_Geneid_GMask.txt Gencode_33_Selected_MappSS_GMask.txt pcarankmatrix.txt > combine_test.txt # In R # spreadsheetsheet3_1 <- list(“Genename”,“Genename”,“Geneid”,“Mapp”,“PC1”,“PC2”,“PC3”)sheet3_2<- sampleKeyTNFATGFBcombined_headers <- c(sheet3_1, sheet3_2)combined_spreadsheet <- as.matrix(read.table(“combine_test.txt”))colnames(combined_spreadsheet) <- combined_headerscombined_spreadsheet <- combined_spreadsheet[order(combined_spreadsheet[,1]),] #sort by the first columnwrite.table(combined_spreadsheet,file=“combined_spreadsheet.txt”, sep = '', row.names = FALSE)```

In Excel, we can sort the spreadsheet with PC1, PC2, and so on to see the corelation between the treatment and the expression level in each gene.

Future works

References