Research project: Expression analysis of intestinal tumor organoids derived from different driver gene genotypes

Moneeb I. Bajwa

Summer 2018

Intro

This project aims to replicate a prior study (see: http://cancerres.aacrjournals.org/content/78/5/1334) done on cancer derived organoids in order find the expression patterns of genes in the progression of cancer. The study focused on the expression patterns of organoids with cancer driving mutations on *Apc*, *Kras* and *Tgfbr2* genes. The goal of this project is to utilize and explore various bioinformatics tools and methods on this study's raw SRA data from NCBI. Mainly, figures similar to Figure 5 of the study are what this report focuses on creating.

Obtaining data

To obtain the data, SRA sequences were downloaded from NCBI (https://www.ncbi.nlm.nih.gov/bioproject/PRJDB5631). The following code was used to simplify the process:

```
#!/bin/bash
#Set a variable with the file containing SRA accessions
filename="SRR_Acc_List.txt"
#Create a function to read the file line-by-line
getAll(){
    while read -r line
    do
        #Get the FASTQ
        fastq-dump $line --split-3 --gzip -O Fastq/
    done < "$filename"
}
#Call the function in the background
getAll 1>get.log 2>get.err &
```

The sample replicates (totaling three in number) were downloaded in batches to save space; meaning the first sample set was downloaded and the eventual counts were made, after which they were deleted and the next sample set was downloaded.

Aligning and counts

STAR was then used on the raw reads to align the reads to the genome and obtain counts on genes. The genome and annotation were first loaded and prepared as follows:

```
#!/bin/sh
STAR --runThreadN 5 --runMode genomeGenerate --genomeDir genome_dir \
--genomeFastaFiles GRCm38.primary_assembly.genome.fa \
--sjdbGTFfile gencode.vM18.primary_assembly.annotation.gtf \
```

```
--limitGenomeGenerateRAM 95444909440 \
1>prepStar.log 2>prepStar.err &
```

The `--limitGenomeGenerateRAM 95444909440` parameter was specified based on feedback on an initial run, as the default parameter of 31G was not sufficient. The genome file as well as the GTF file were obtained from GENCODE.

After the genome directory was prepared, the FASTQ files were then aligned and counts were performed as follows:

```
#!/bin/bash
#Initialize variable for directory of fastq files
fastqPath="Fastq/"
#Initialize variables to contain the suffix for reads
Suffix=".fastq"
#Loop through all the fastq files in $fastqPath
for InFile in $fastqPath*$Suffix
do
        #Remove path from filename and assign to pathRemoved
        pathRemoved="${InFile/$fastqPath/}"
        #Remove the read suffix from $pathRemoved and assign to sampleName
        sampleName="${pathRemoved/$Suffix/}"
        STAR --runThreadN 4 --genomeDir genome_dir \
        --readFilesIn $fastqPath$sampleName$Suffix \
        --quantMode GeneCounts \
        1>$sampleName.log 2>$sampleName.err &
done
```

The resulting tab-delimited file looked as such:

```
1 N_unmapped      3026943 3026943 3026943
2 N_multimapping  15069738 15069738        15069738
3 N_noFeature     2047858 60334802        2178629
4 N_ambiguous     2172782 26847   759295
5 ENSMUSG00000102693.1    0       0       0
6 ENSMUSG00000064842.1    0       0       0
7 ENSMUSG00000051951.5    4       0       4
...
36 ENSMUSG00000033845.13   6229    2       6227
37 ENSMUSG00000102275.1    11      0       11
38 ENSMUSG00000025903.14   4597    3       4594
39 ENSMUSG00000104217.1    0       0       0
40 ENSMUSG00000033813.15   927     1       926
41 ENSMUSG00000062588.4    25      0       25
```

The correct count column is the fourth, as the reads were single-ended and strand specific, so it could not be the second column; and based on the low counts in the third column, that could easily be excluded.

Data prep

Before diving straight into differential expression analysis, we must first obtain, clean and prepare the required data.

Getting gene names

In order to get the output file sorted out to include gene names, the following code was used with AKW on the annotation file:

```
awk -F "\t" '{if ($3 == "gene") print $9}' gencode.vM18.primary_assembly.annotation.gtf |
  awk -F " " '{print $2,$6}' | awk -F "; " 'OFS="\t" {print $1, $2}' | tr -d '"\;' > ensmus.txt
```

To better explain what this code is doing, let us look at a snippet of the GTF file:

```
  20804 chr1    HAVANA  gene    46011757        46012020        .       -       .
        gene_id "ENSMUSG00000090682.4"; gene_type "processed_pseudogene";
        gene_name "RP23-183P22.2"; level 2; ha
  20805 chr1    HAVANA  transcript      46011757        46012020        .       -       .
        gene_id "ENSMUSG00000090682.4"; transcript_id "ENSMUST00000182781.1";
        gene_type "processed_pseud
```

The code before the first pipe is splitting all the fields by tabs and getting all fields which are labelled as `gene` in the 3rd field. Then those lines are further processed by splitting by space and only retrieving the `gene_id` and `gene_name`. After that, those fields are cleaned further and sent to the output file. The result is a file containing a list of gene names along with the corresponding gene IDs:

```
ENSMUSG00000102266.1    RP24-141F19.1
ENSMUSG00000101112.1    RP24-141F19.2
ENSMUSG00000100697.1    RP24-141F19.3
ENSMUSG00000099877.6    Pinc
ENSMUSG00000098697.1    Gm27315
```

Load into R

Next we load both the counts file and gene names file into R to get the gene names matching with counts and gene IDs:

```
gene_and_id = read.delim("C:/Users/Moneeb/Downloads/ensmus.txt")
#sample set 1
counts73<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089573ReadsPerGene.out.tab")
counts75<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089575ReadsPerGene.out.tab")
counts80<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089580ReadsPerGene.out.tab")
counts81<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089581ReadsPerGene.out.tab")
counts85<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089585ReadsPerGene.out.tab")
counts88<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089588ReadsPerGene.out.tab")
#sample set 2
counts72<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089572ReadsPerGene.out.tab")
counts76<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089576ReadsPerGene.out.tab")
counts78<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089578ReadsPerGene.out.tab")
counts82<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089582ReadsPerGene.out.tab")
counts84<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089584ReadsPerGene.out.tab")
counts89<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089589ReadsPerGene.out.tab")
#sample set 3
counts74<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089574ReadsPerGene.out.tab")
counts77<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089577ReadsPerGene.out.tab")
counts79<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089579ReadsPerGene.out.tab")
counts83<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089583ReadsPerGene.out.tab")
counts86<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089586ReadsPerGene.out.tab")
counts87<- read.delim("C:/Users/Moneeb/Downloads/counts/DRR089587ReadsPerGene.out.tab")
head(gene_and_id)
```

```
##   ENSMUSG00000102693.1 RP23.271O17.1
## 1 ENSMUSG00000064842.1       Gm26206
## 2 ENSMUSG00000051951.5          Xkr4
## 3 ENSMUSG00000102851.1 RP23-317L18.1
## 4 ENSMUSG00000103377.1 RP23-317L18.4
## 5 ENSMUSG00000104017.1 RP23-317L18.3
## 6 ENSMUSG00000103025.1  RP23-115I1.6
```

```
head(counts87)
```

```
##             N_unmapped X15289259 X15289259.1 X15289259.2
## 1       N_multimapping  17817767    17817767    17817767
## 2          N_noFeature   2684648    69888629     2848401
## 3          N_ambiguous   2596032       36124      919105
## 4 ENSMUSG00000102693.1         0           0           0
## 5 ENSMUSG00000064842.1         0           0           0
## 6 ENSMUSG00000051951.5        10           0          10
```

```
nrow(gene_and_id)
```

```
## [1] 54231
```

```
nrow(counts87)
```

```
## [1] 54235
```

```
#get only unique gene names
gene_and_id=gene_and_id[!duplicated(gene_and_id[2]),]
nrow(gene_and_id)
```

```
## [1] 54165
```

```
#get only the correct stranded column (4th column) and ids (1st column)
counts73=counts73[-c(2,3)]
counts75=counts75[-c(2,3)]
counts80=counts80[-c(2,3)]
counts81=counts81[-c(2,3)]
counts85=counts85[-c(2,3)]
counts88=counts88[-c(2,3)]

counts72<- counts72[-c(2,3)]
counts76<- counts76[-c(2,3)]
counts78<- counts78[-c(2,3)]
counts82<- counts82[-c(2,3)]
counts84<- counts84[-c(2,3)]
counts89<- counts89[-c(2,3)]

counts74<- counts74[-c(2,3)]
counts77<- counts77[-c(2,3)]
counts79<- counts79[-c(2,3)]
counts83<- counts83[-c(2,3)]
counts86<- counts86[-c(2,3)]
counts87<- counts87[-c(2,3)]
head (counts87)
```

```
##             N_unmapped X15289259.2
## 1       N_multimapping    17817767
## 2          N_noFeature     2848401
```

```
## 3          N_ambiguous      919105
## 4 ENSMUSG00000102693.1           0
## 5 ENSMUSG00000064842.1           0
## 6 ENSMUSG00000051951.5          10
```

Now we must provide the naming specifications based on tumor sample type for each count column. The names are abbreviations of the cancer driving genes that are mutated (e.g. "AK" standing for the combination of *Apc* and *Kras* being mutated):

```
 #Give columns names so columns are identifiable.
colnames(counts73)<-c("gene_id","A1")
colnames(counts75)<-c("gene_id","AK1")
colnames(counts80)<-c("gene_id","AKP1")
colnames(counts81)<-c("gene_id","AKT1")
colnames(counts85)<-c("gene_id","ATP1")
colnames(counts88)<-c("gene_id","AKTP1")

colnames(counts72)<-c("gene_id","A2")
colnames(counts76)<-c("gene_id","AK2")
colnames(counts78)<-c("gene_id","AKP2")
colnames(counts82)<-c("gene_id","AKT2")
colnames(counts84)<-c("gene_id","ATP2")
colnames(counts89)<-c("gene_id","AKTP2")

colnames(counts74)<-c("gene_id","A3")
colnames(counts77)<-c("gene_id","AK3")
colnames(counts79)<-c("gene_id","AKP3")
colnames(counts83)<-c("gene_id","AKT3")
colnames(counts86)<-c("gene_id","ATP3")
colnames(counts87)<-c("gene_id","AKTP3")
head(counts87)
```

```
##                gene_id    AKTP3
## 1       N_multimapping 17817767
## 2          N_noFeature  2848401
## 3          N_ambiguous   919105
## 4 ENSMUSG00000102693.1        0
## 5 ENSMUSG00000064842.1        0
## 6 ENSMUSG00000051951.5       10
```

```
colnames(gene_and_id)<-c("gene_id","gene_name")
head(gene_and_id)
```

```
##                gene_id     gene_name
## 1 ENSMUSG00000064842.1       Gm26206
## 2 ENSMUSG00000051951.5          Xkr4
## 3 ENSMUSG00000102851.1 RP23-317L18.1
## 4 ENSMUSG00000103377.1 RP23-317L18.4
## 5 ENSMUSG00000104017.1 RP23-317L18.3
## 6 ENSMUSG00000103025.1  RP23-115I1.6
```

Next we must merge all counts:

```
MyMerge <- function(x, y){
  df <- merge(x, y, by= "gene_id")
  return(df)
}
```

```
merged <- Reduce(MyMerge, list(
                gene_and_id,
                counts72,
                counts73,
                counts74,
                counts75,
                counts76,
                counts77,
                counts78,
                counts79,
                counts80,
                counts81,
                counts82,
                counts83,
                counts84,
                counts85,
                counts86,
                counts87,
                counts88,
                counts89)
                )
head(merged)
```

```
##                 gene_id gene_name    A2   A1    A3  AK1   AK2   AK3  AKP2
## 1  ENSMUSG00000000001.4     Gnai3 15144 8644 14153 9051 10998 15136 14332
## 2 ENSMUSG00000000003.15      Pbsn     0    0     0    0     0     0     0
## 3 ENSMUSG00000000028.15     Cdc45  2920 1878  3383 1887  2428  2786  1690
## 4 ENSMUSG00000000031.16       H19    38    9    12   20    23    17    11
## 5 ENSMUSG00000000037.16     Scml2    34    6   102   17    47    62    58
## 6 ENSMUSG00000000049.11      Apoh    11    1    46    2     1     1     5
##    AKP3  AKP1  AKT1  AKT2  AKT3  ATP2 ATP1  ATP3 AKTP3 AKTP1 AKTP2
## 1 19793 12931 17268 14936 23715 10788 3956 15679 18779 18053 18143
## 2     0     0     0     0     0     0    0     0     0     0     0
## 3  2811  1941  2790  3190  5137  2101 1634  4497  3700  4172  4315
## 4    41    36   108    37   172   467  132   159    43     3     7
## 5    83   103    83    41   143   198   99   147    41    15    26
## 6     9     1     3     1     4     2    3     3     0     4     2
```

```
nrow(merged)
```

```
## [1] 54165
```

```
sub_merged=merged[c(3:ncol(merged))]
rownames(sub_merged)<-merged$gene_name
head(sub_merged)
```

```
##          A2   A1    A3  AK1   AK2   AK3  AKP2  AKP3  AKP1  AKT1  AKT2
## Gnai3 15144 8644 14153 9051 10998 15136 14332 19793 12931 17268 14936
## Pbsn      0    0     0    0     0     0     0     0     0     0     0
## Cdc45  2920 1878  3383 1887  2428  2786  1690  2811  1941  2790  3190
## H19      38    9    12   20    23    17    11    41    36   108    37
## Scml2    34    6   102   17    47    62    58    83   103    83    41
## Apoh     11    1    46    2     1     1     5     9     1     3     1
##        AKT3  ATP2 ATP1  ATP3 AKTP3 AKTP1 AKTP2
## Gnai3 23715 10788 3956 15679 18779 18053 18143
```

```
## Pbsn      0     0    0     0     0     0     0
## Cdc45  5137  2101 1634  4497  3700  4172  4315
## H19     172   467  132   159    43     3     7
## Scml2   143   198   99   147    41    15    26
## Apoh      4     2    3     3     0     4     2
```

DESeq2

PCA plot

Let us now use DESeq2 and make a PCA plot:

```
library("DESeq2")

colDat=c("A", "A", "A", "AK", "AK", "AK", "AKP", "AKP","AKP","AKT",
         "AKT","AKT", "ATP","ATP", "ATP", "AKTP", "AKTP", "AKTP" )
colDat=as.matrix(colDat)
rownames(colDat)<-c("A2", "A1", "A3", "AK1", "AK2", "AK3","AKP2",
                    "AKP3", "AKP1", "AKT1", "AKT2", "AKT3", "ATP2",
                    "ATP1", "ATP3", "AKTP3", "AKTP1", "AKTP2")
colnames(colDat)<-"organoid_class"
head(colDat)
```

```
##     organoid_class
## A2  "A"
## A1  "A"
## A3  "A"
## AK1 "AK"
## AK2 "AK"
## AK3 "AK"
```

```
ddsAll=DESeqDataSetFromMatrix(countData = sub_merged,
                                    colData = colDat,
                                    design = as.formula(~organoid_class))
nrow(counts(ddsAll))
```

```
## [1] 54165
```

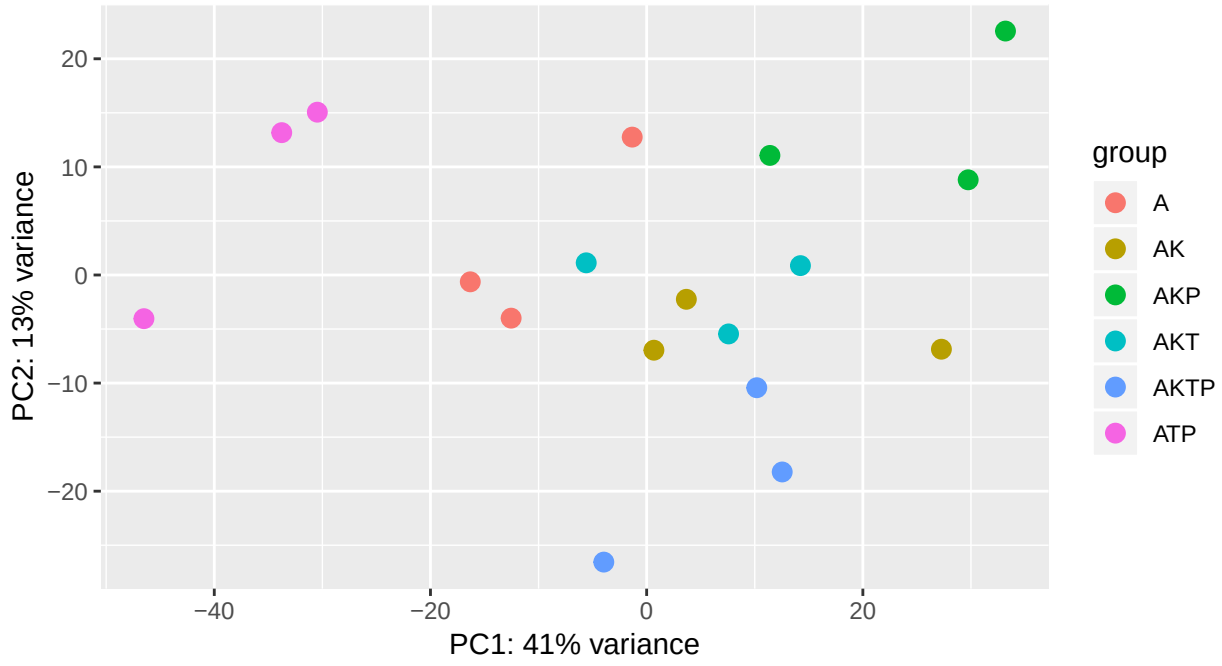
```
#get rid of genes with no/little expression
ddsAll <- ddsAll[ rowSums(counts(ddsAll)) > 10 ]

nrow(counts(ddsAll))
```

```
## [1] 24998
```

```
ddsAll <- DESeq(ddsAll)

#get regularized log and plot
rldAll <- rlog(ddsAll, blind=FALSE)
plotPCA(rldAll, intgroup= "organoid_class")
```



We can see the grouping patterns of similar groups, and the pattern matches that as seen in the paper.

Venn Diagram

Let's prepare the data to get a Venn diagram of the AK-common DEGs:

```
#get genes with greatest variance
topVarGenes <- head( order( rowVars( assay(rldAll) ), decreasing=TRUE ), 2000 )
topLogGenes=assay(rldAll)[ topVarGenes, ]
top_gene_names=rownames(topLogGenes)

resAKvA <- as.data.frame(results(ddsAll, name="organoid_class_AK_vs_A" ))
resAKPvA <-  as.data.frame(results(ddsAll,name="organoid_class_AKP_vs_A"))
resAKTvA <-  as.data.frame(results(ddsAll, name="organoid_class_AKT_vs_A"))
resAKTPvA <-  as.data.frame(results(ddsAll, name="organoid_class_AKTP_vs_A"))
resATPvA <-  as.data.frame(results(ddsAll, name="organoid_class_ATP_vs_A"))

AK=subset(resAKvA, select= log2FoldChange)
colnames(AK)<-"AK"
AKP=subset(resAKPvA, select= log2FoldChange)
colnames(AKP)<-"AKP"
AKT=subset(resAKTvA,select= log2FoldChange)
colnames(AKT)<-"AKT"
AKTP=subset(resAKTPvA, select= log2FoldChange)
colnames(AKTP)<-"AKTP"
ATP=subset(resATPvA, select= log2FoldChange)
```

```r
colnames(ATP)<-"ATP"

#merge on gene names
merge1=merge(AKP, AK, by=0)
AKT$Row.names <- rownames(AKT)
merge2=merge(merge1, AKT, by = "Row.names")
AKTP$Row.names <- rownames(AKTP)
logvalues=merge(merge2, AKTP, by = "Row.names")

logvalues[is.na(logvalues)]<-0
head(logvalues)
```

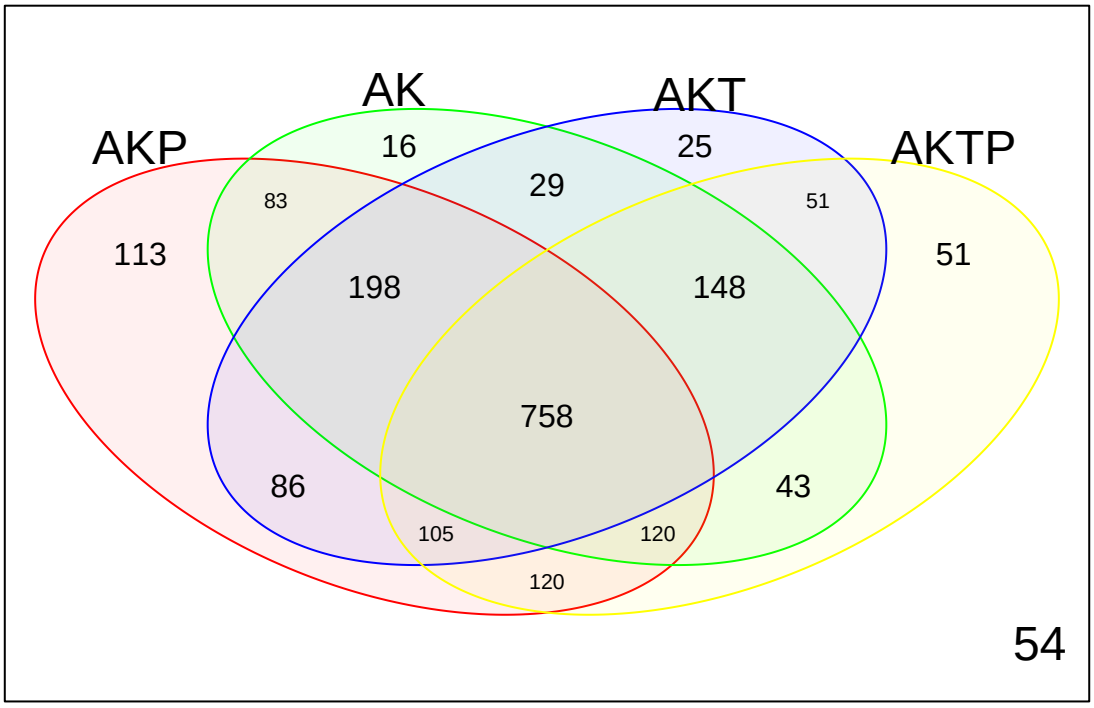
```
##        Row.names        AKP         AK        AKT        AKTP
## 1 00R_AC107638.2 -0.7033426  0.2374644 -1.5227726  0.40294188
## 2  0610009O20Rik -0.6742699 -0.2994587 -0.3070563 -0.09393477
## 3  1110020A21Rik -0.2368418 -0.3830357 -0.1506346 -0.67160865
## 4  1110036E04Rik  0.9693813  1.7338340  0.3037279  0.53079780
## 5  1600014C23Rik  1.9900549  1.1194663  2.2373049 -0.51694310
## 6  1700018M17Rik  2.4366846  2.4303145  2.1215488  0.58837928
```

```r
#to use for later in the pipeline, store as logvalues1
logvalues1=logvalues

logvalues=logvalues[which(logvalues$Row.names %in% top_gene_names),]

#get binary table of DEGs
logvalues[2:ncol(logvalues)]=as.integer(abs(logvalues[2:ncol(logvalues)]) > .6)
```

```r
library(limma)
vcounts=vennCounts(logvalues[2:ncol(logvalues)])
vennDiagram(vcounts, circle.col=c("red", "green", "blue", "yellow"))
```



It is important to mention that the AKTPF samples were not included in this result as was used in the original study. Also, depending on what level of stringency used, these results may be different (as compared to the published paper in which this study was conducted).

GO enrichment

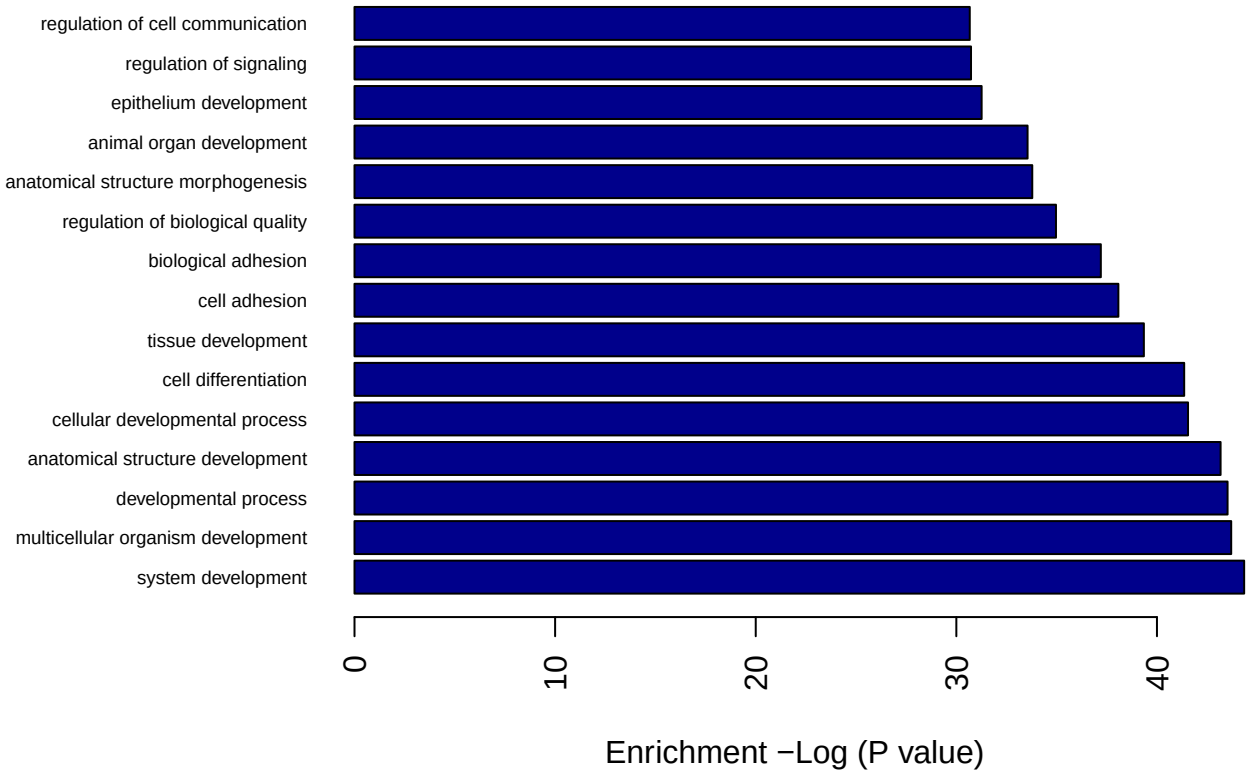
The gene list of AK-common genes was then used in g:Profiler:

```
AK_common_genes =
  logvalues$Row.names[which(logvalues$AKT != 0
                            & logvalues$AKTP != 0
                            & logvalues$AK != 0
                            & logvalues$AKP != 0)]
write.csv(
  AK_common_genes,
  "gene_list.txt",
  col.names = F,
  row.names = F,
  sep = "\n",
  quote = F
  )
```

The resulting enrichment is loaded into R and a graph is then able to be plotted:

```
go_data<- read.delim("C:/Users/Moneeb/Downloads/gprofiler_results_1069390895760.txt")
go_data=data.frame(go_data$Description, go_data$p.Val)
go_data_sorted=go_data[with(go_data, order(go_data.p.Val)),]
```

```
go_data_log <- transform(go_data_sorted, go_data.p.Val=-log(go_data.p.Val,2) )
go_data_log=go_data_log[!duplicated(go_data_log[1]),]
row.names(go_data_log)=go_data_log$go_data.Description
go_data_log<- go_data_log[-1]
go_data_log=as.matrix(go_data_log)
par(mar=c(4, 9, 2, 0.1))
barplot(go_data_log[1:15,], horiz=T, col="darkblue",cex.names=0.6, las=2,
        xlab = "Enrichment -Log (P value)")
```

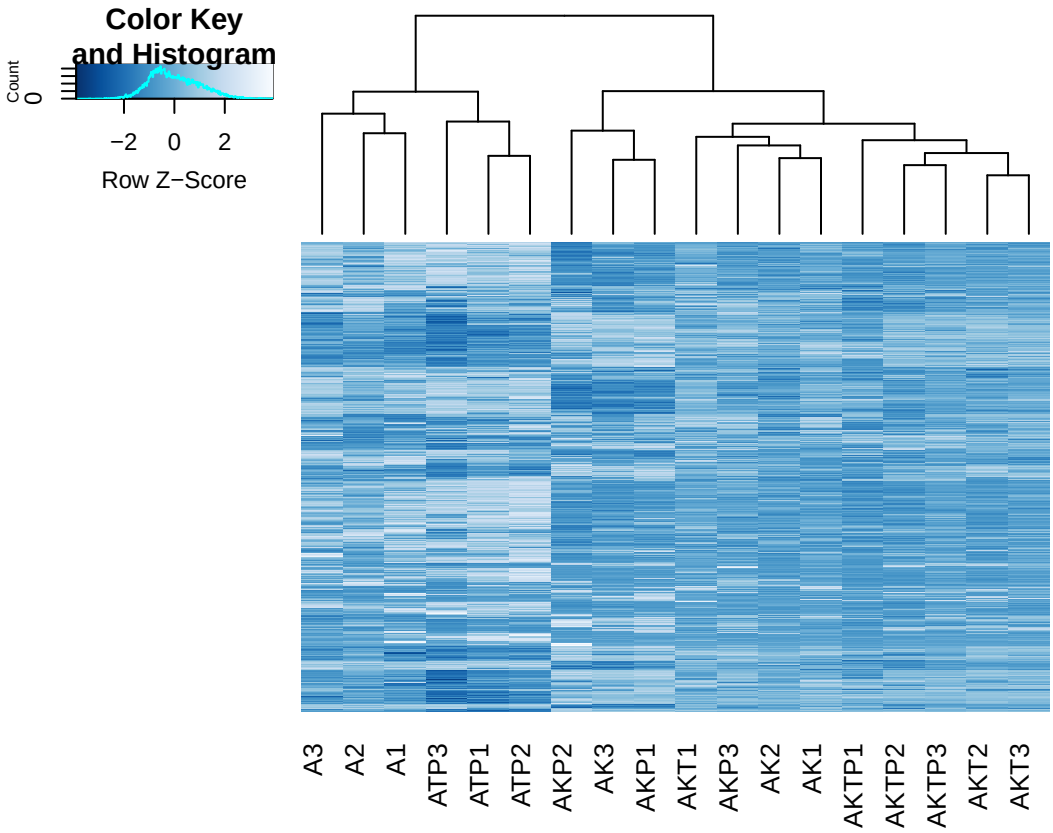


Heatmap

Now let us finish with a heatmap using the AK-common genes found in the Venn diagram step:

```
library("RColorBrewer")
library(gplots)
colors <- colorRampPalette(rev(brewer.pal(9, "Blues")))(255)
topLogGenes=topLogGenes[,order(colnames(topLogGenes))]

heatmap.2(topLogGenes[which(rownames(topLogGenes) %in% AK_common_genes),]
, labRow = NA, col=colors, scale="row",
trace="none", dendrogram="column"
)
```



We can see the similar expression patterns of AK-common samples.

Appendix

This appendix includes the initial project workflow. The initial idea was to construct a trancriptome for the reads, but the fact that the reads were only 36 bp long caused the abandonment of that initial plan. The alignment was less than 3% when performed in this manner.

Trimming reads

```
#!/bin/bash
#Initialize variables with desired output path
OutPath="Trimmed/"
mkdir -p $OutPath
trimAll(){
#Loop through all the zipped fastq files
for InFile in *.fastq.gz
    do
      #Trim adapter sequences. Output should be sent to paired/unpaired directory accordingly.
        nice -n 19 java -jar /usr/local/programs/Trimmomatic-0.36/trimmomatic-0.36.jar SE \
        -threads 4 -phred33 \
        $InFile \
        $OutPath$InFile \
        HEADCROP:0 \
        ILLUMINACLIP:Overrepresented_list.fa:2:30:10 \
        LEADING:20 TRAILING:20 SLIDINGWINDOW:4:30 MINLEN:36
    done
}
trimAll 1>trimAll.log 2>trimAll.err &
```

The over-represented sequences were found with FastQC.

Aligning reads

The GMAP database was built with the genome as follows:

```
#!/bin/sh
gmap_build -D . \
-d M18GmapDb \
GRCm38.primary_assembly.genome.fa \
1>Gmap_Build.log 2>Gmap_Build.err &
```

An index of intron splice sites was constructed with the gene annotation file as so:

```
#!/bin/sh
nice -n19 iit_store \
-G gencode.vM18.primary_assembly.annotation.gff3 \
-o GmapIIT \
1>buildIIT.log 2>buildIIT.err &
```

```
#!/bin/sh
#Initialize variable for directory of paired trimmed fastq files
fastqPath="Trimmed/"
#Initialize variables to contain the suffix for reads
Suffix=".fastq"
```

```
#Initialize variable with desired output paths
samOutPath="sam/"
alignAll(){
#Loop through all the fastq files in $fastqPath
for InFile in $fastqPath*$Suffix
        do
                #Remove path from filename and assign to pathRemoved
                pathRemoved="${InFile/$fastqPath/}"
                #Remove the read suffix from $pathRemoved and assign to sampleName
                sampleName="${pathRemoved/$Suffix/}"
                nice -n 19 gsnap \
                -A sam \
                -s GmapIIT.iit \
                -D . \
                -d M18GmapDb \
                $fastqPath$sampleName$Suffix \
                1>$samOutPath$sampleName.sam 2>$samOutPath$sampleName.err
        done
}
alignAll &
```

Sorting & merging

The resulting SAM files from the alignment were sorted to BAM format like so:

```
#!/bin/bash
#Initialize sam directory variable
samPath="sam/"
#Initialize variable containing bam directory/outpath
outPath="bam/"
#Initialize variable to contain the suffix
Suffix=".sam"
sortAll(){
#Loop through all the sam files in $samPath
for InFile in $samPath*$Suffix
    do
        #Remove path from filename and assign to pathRemoved
        pathRemoved="${InFile/$samPath/}"
        #Remove suffix from $pathRemoved and assign to sampleName
        sampleName="${pathRemoved/$Suffix/}"
        samtools sort \
        $samPath$sampleName$Suffix \
        -o $outPath$sampleName.sorted.bam \
        1>$outPath$sampleName.sort.log 2>$outPath$sampleName.sort.err
    done
}
sortAll &
```

The BAM files were then merged for subsequent processing in Trinity:

```
#!/bin/sh
ls bam/*.bam > bamIn.txt
samtools merge -b bamIn.txt AllBam.bam \
1>merge.log 2>merge.err &
```

Assembly

The resulting merged BAM file was then used for assembly with Trinity:

```
#!/bin/sh
nice -n19 /usr/local/programs/\
trinityrnaseq-2.2.0/\
Trinity --genome_guided_bam AllBam.bam \
--genome_guided_max_intron 10000 \
--max_memory 10G --CPU 4 \
1>trin.log 2>trin.err &
```

The results were then analyzed:

```
#!/bin/sh
/usr/local/programs/trinityrnaseq-2.2.0/\
util/TrinityStats.pl \
trinity_out_dir/Trinity-GG.fasta
```

Here we can see the results:

```
bajwa.m@defiance[Research_Project]# ./analyzeTrinity.sh


################################
## Counts of transcripts, etc.
################################
Total trinity 'genes':  1099
Total trinity transcripts:      1099
Percent GC: 48.13

########################################
Stats based on ALL transcript contigs:
########################################

        Contig N10: 567
        Contig N20: 481
        Contig N30: 403
        Contig N40: 334
        Contig N50: 298

        Median contig length: 266
        Average contig: 308.97
        Total assembled bases: 339563


#####################################################
## Stats based on ONLY LONGEST ISOFORM per 'GENE':
#####################################################

        Contig N10: 567
        Contig N20: 481
        Contig N30: 403
        Contig N40: 334
        Contig N50: 298

        Median contig length: 266
        Average contig: 308.97
        Total assembled bases: 339563
```

Abundance Estimation

Next, abundance estimation was performed to prepare for the subsequent differential expression analysis.

First the bowtie2 index was prepared for the impending alignment:

```
#!/bin/bash
trinityPath='/usr/local/programs/trinityrnaseq-2.2.0'
#Run, align, and estimate abundance with --prep_reference option
#to build bowtie2 index
nice -n19 $trinityPath/util/align_and_estimate_abundance.pl \
--transcripts Trinity-GG.fasta --prep_reference \
--aln_method bowtie2 --est_method eXpress \
--trinity_mode --output_dir prepAbun --seqType fq
```

The trimmed samples were then aligned to the created transcriptome and the stats were attained:

```
#!/bin/bash
transcriptome=Trinity-GG.fasta
outPath=xprs_gg
trinityPath='/usr/local/programs/trinityrnaseq-2.2.0'
inPath='Trimmed/'
Suffix='.fastq'
mkdir -p xprs_gg
logSuffix='.log'
errSuffix='.err'
for Reads in $inPath*$Suffix
do
    sample="${Reads/$inPath/}"
    sample="${sample/$Suffix/}"
    nice -n19 $trinityPath/util/align_and_estimate_abundance.pl \
    --transcripts Trinity-GG.fasta --aln_method bowtie2 --est_method eXpress \
    --trinity_mode --output_dir $outPath$sample --seqType fq --SS_lib_type F \
    --single $Reads --thread_count 1 \
    1>$outPath$sample$logSuffix 2>$outPath$sample$errSuffix &
done
```

The alignment stats were then written to a CSV file:

```
#!/bin/bash
xprsPath=xprs_gg
#Set a variable with the xprs output file extension
xprsExt='.err'
#Set a variable with the output filename
outFile='tranAlignStats.csv'
#Set a variable with the search text around the desired data
searchText='overall alignment rate'
#Write a header line in csv format
echo 'Sample,TranAlignPct' > $xprsPath$outFile
#grep the search text and pipe to a while loop to process line-by-line
grep "$searchText" $xprsPath*$xprsExt | while read -r line ;
do
    #Remove search text from the output
    line="${line/$searchText/}"
    #Remove file path from the output
    line="${line/$xprsPath/}"
```

```
    #Remove file extension from the output
    line="${line/$xprsExt/}"
    #Replace colon separator with comma
    line="${line/:/,}"
    #Remove % sign
    line="${line/\%/}"
    #Append output in csv format
    echo $line >> $xprsPath$outFile
done
```
