10X single cell sequencing data analysis pipeline protocol

1. Raw UMI

Raw UMI matrix can be downloaded from:

http://dna.engr.uconn.edu/10x/TS19/RUN1-WholeEmbryo/filtered\_gene\_bc\_matrices/mm10/

Underlined part can be edited to get access to other samples. The matrix is either in .tsv or .mtx format.

Download the .tsv file and rename, then read into R and trim:

TS19.WE <- read.table("TS19.WE.tsv", header=TRUE)

row.names(TS19.WE) <- TS19.WE$Gene

TS19.WE$Gene <- NULL

corner(TS19.WE,c=4)

AAACATACAAGAGT.1 AAACATACAAGATG.1 AAACATACAGATGA.1 AAACATACATCGTG.1

0610007P14RIK 0 1 0 0

0610009B22RIK 1 0 1 1

0610009L18RIK 0 0 0 0

0610009O20RIK 1 0 0 0

0610010F05RIK 1 0 1 0

After checking the matrix, genes should be on rows and barcode on column. For the convenience of downstream analysis, we prefer to add prefix to barcode:

colnames(TS19.WE) <- paste("TS19.WE", colnames(TS19.WE), sep = "\_")

By doing so, each barcode in this table starts with TS19.WE\_, thus differentiate it from other samples.

Note: when trimming the data table, you want to use ENSM ID rather than the gene names shown above, use the convert table gene.tsv in this link:

http://dna.engr.uconn.edu/10x/TS19/RUN1-WholeEmbryo/filtered\_gene\_bc\_matrices/mm10/

2. Quality control (QC)

The purpose of QC is to get rid of bad libraries and doublets. The distribution of sum\_UMI of all cells fits a 3-normal distribution model. Our hypothesis is, the first norm-distribution (at low UMI) is bad library, the second is good library (also majority of population), the third can be doublets. To get rid of the first and third normal distribution, we set 95% confidence interval for first and third distribution, and discard anything below low interval and above high interval.

To do so, first we calculate the sum\_UMI of each sample:

sum\_UMI <- colSums(TS19.WE)

corner(as.data.frame(sum\_UMI))

sum\_UMI

TS19.WE\_AAACATACAAGAGT.1 2622

TS19.WE\_AAACATACAAGATG.1 2563

TS19.WE\_AAACATACAGATGA.1 4212

TS19.WE\_AAACATACATCGTG.1 5122

TS19.WE\_AAACATTGCAGGAG.1 6403

Save this as a .csv file, open in JMP and calculate log(sum\_UMI) in 3rd column.

To get distribution and 3-norm distribution fit:

Select log(sum\_UMI) column, Analyze -> Distribution, click the red triangle on histogram -> Continuous Fit -> Normal Mixture -> Normal 3 Mixture. In fitted normal 3 mixture, right click and make into data table. In this table, we need lower 95% of first peak and higher 95% of third peak, anything below and above this frame will be discarded. To get QC’ed cells, go back to data table, Row -> Data Filter -> select log value column and add -> type in the number from 3-norm fit -> close window ->Table -> Subset -> make sure the ‘Selected Rows’ box is checked -> OK. Save and rename this new table in .csv format for further analysis.

Trim the raw UMI by QC information:

# TS19.QC.WE is the QC info .csv, first column is barcode after QC.

# TS19.WE is the raw UMI.

TS19.QC.WE <- read.csv ("UMI\_QC.Subset.of.TS19.WE.UMI.csv", header=T)

TS19.QC.WE.UMI <- TS19.WE[,TS19.QC.WE$Column.1]

Then save the file as .csv for further analysis.

In our pipeline, we joined all samples in TS19 together:

joined.UMI <- merge(TS19.QC.WE.UMI, TS19.QC.HT.UMI, by=0, all=TRUE) # merge by row names (by=0 or by="row.names")

joined.UMI[is.na(joined.UMI)] <- 0 # replace NA values

The code above is not verified, and should be repeated to join all 4 UMI tables.

3. Normalization, center and scale

The purpose of this step is to fit the input criteria of downstream PCA, as it prefers a norm-distributed data sets. And as the UMI counts vary from one cell to another, we need to normalize the UMI counts by using a scale factor:

By applying this scale factor to all cells, the UMI will be normalized.

bc\_sums<-colSums(TS19.3NM.QC.rawUMI)

median\_sum<-median(bc\_sums)

corner(as.data.frame(bc\_sums))

bc\_sums

TS19.WE\_AAACATACAAGAGT.1 2622

TS19.WE\_AAACATACAAGATG.1 2563

TS19.WE\_AAACATACAGATGA.1 4212

TS19.WE\_AAACATACATCGTG.1 5122

TS19.WE\_AAACATTGCAGGAG.1 6403

barcode\_norm\_TS19<-sweep(TS19.3NM.QC.rawUMI,2,median\_sum/bc\_sums,"\*")

After normalization, a log transform will be applied. Centering and scaling will be performed in PCA function.

To do log transformation:

log\_bcnorm\_TS19<-log2(1+barcode\_norm\_TS19) # need +1 UMI "pseudocount" to avoid NA entries for zeros

TS19.ALL.QC.bcNorm.log2.UMI <- log\_bcnorm\_TS19

Save this as TS19.ALL.bcNorm.log2.UMI.

Depending on when you delete genes with all 0 expressions (before or after norm-log transformation), the UMI you get might be slightly different. Typically we do this after the step above.

To delete genes with all 0 expressions:

TS19.ALL.QC.bcNorm.log2.UMI.sum <- rowSums(TS19.ALL.QC.bcNorm.log2.UMI)

n\_row <- nrow(TS19.ALL.QC.bcNorm.log2.UMI) # delete genes that have sum\_UMI as 0

count <- c()

for (i in 1:n\_row) {

if (TS19.ALL.QC.bcNorm.log2.UMI.sum[i] == 0) {

count <- c(count,i)

}

}

TS19.ALL.QC.bcNorm.log2.UMI <- TS19.ALL.QC.bcNorm.log2.UMI [-count,]

VERY important to note that cellranger Rkit drops genes with zero detection before normalization.

4. Extract 100PC and 3tSNE

The purpose of this step is to reduce dimensions of our data table from ~20k dimensions to 3 dimensions, to visualize in 3D space.

To do PCA, we use prcomp function in {stats} package. Note that as we want to study cells, we need to put cells on rows and genes on columns. Which means we need to transpose the table:

TS19.ALL.QC.bcNorm.log2.UMI.t <- t(TS19.ALL.QC.bcNorm.log2.UMI)

corner(TS19.ALL.QC.bcNorm.log2.UMI.t,c=3)

ENSMUSG00000051951 ENSMUSG00000089699 ENSMUSG00000102343

TS19.WE\_AAACATACAAGAGT.1 0 0 0

TS19.WE\_AAACATACAAGATG.1 0 0 0

TS19.WE\_AAACATACAGATGA.1 0 0 0

TS19.WE\_AAACATACATCGTG.1 0 0 0

TS19.WE\_AAACATTGCAGGAG.1 0 0 0

TS19.ALL.QC.norm.prcomp <- prcomp(TS19. ALL.QC.bcNorm.log2.UMI.t,center=T,scale=T)

Note that the center and scale step is done here in function prcomp.

We can save the PCA table by calling the object TS19.ALL.QC.norm.prcomp, we can get the PCs using TS19.ALL.QC.norm.prcomp$X, or we can get the loadings with TS19.ALL.QC.norm.prcomp$rotation. Save these tables for further analysis, and we need 100PCs in columns with cell barcode as column name, name it TS19.ALL.QC.norm.prcomp.100PC.csv.

Then we can do tSNE based on 100 PCs using function Rtsne in package {Rtsne}:

TS19.ALL.QC.norm.prcomp.3tSNE <- Rtsne(TS19.ALL.QC.norm.prcomp.100PC, dims = 3, theta=0.0, check\_duplicates = FALSE, pca = FALSE)

Again, we can save the tSNE table by calling TS19.ALL.QC.norm.prcomp.3tSNE$Y, save this table as tSNE table.

Lastly, we put 100PCs and 3tSNEs together to make a master table, the dimension of the table is 26305 (cell number) by 104 (1 barcode + 100PCs + 3tSNEs).

You can find this master table named with TS19.ALL.QC.cells.pca100.tsne.csv.

5. Clustering

We’re using two different methods to cluster our cells, IBS and HSC.

5.1 Iterative binary splitting clustering

This is a clustering method based on Ward Hierarchical Clustering. More precisely, Ward’s minimum variance method which minimizes the total within-cluster variance. One concern here is that, tSNE is not a Euclidean distance based space while Ward method is usually applied on distance matrix, though we’ve tested a lot on this. In general, IBS will run a few iterations. At first iteration, it divides the whole data set into 2 clusters. At the second iteration, it takes each of the two clusters individually and further divide into 2 clusters. We repeat this for 8 iterations which gives us 256 clusters at 8th iteration. There’re a couple of advantages using this approach, including flexibility and being able to track relationships between clusters.

To do IBS manually in JMP, we can:

Open the .csv and click on Analyze -> Clustering -> Hierarchical, select 3 tSNE columns and click “Y, Columns”, get rid of the check box ‘Standardize Data’, make sure it’s using Ward and OK. On Cluster map, click on red triangle -> number of columns, enter 2 and save clusters. Go to the new column, right click -> Standardize Columns Attributes -> Attributes dropbox ->  Select All -> Data Type dropbox -> Character -> OK, then rename it as L1. Go back to Cluster map and select Redo -> Relaunch analysis, select L1 and click ‘by’. This will re-cluster based on our L1 result. Now we have 2 maps, to do everything in batch, hold ‘Ctrl’ and click on red triangle -> number of columns, enter 2 twice and save clusters. To merge two columns together, we change the new column from Numeric to Character as described before, and create a new column -> Column Properties -> Formula -> Edit formula -> Character -> Concat, select L1 and Cluster by L1, to generate out new column L2. By repeating this, we will cluster the table to Level8.

To reduce human labor in doing this, I wrote a script in JMP to batch run IBS:

//L1

myPath = "C:/Users/NelsonLab/Desktop/scratch/tSNE\_subset/";

dt = Open( myPath || "TS13.ch2.3NM.QC.tSNE.csv");

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),Method( "Ward" ));

obj << Save Clusters;

column("Cluster") << Data Type (Character);

col = Column(5); //5 is the first column you want to put cluster results

col << Set Name("Level1");

//L2

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level1 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level1") << Data Type (Character);

column("Level1") << Data Type (Character);

dt << New Column("Level2", Character,Formula( :Level1 || :Cluster By Level1));

dt << Save();

//L3

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level2 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level2") << Data Type (Character);

column("Level2") << Data Type (Character);

dt << New Column("Level3", Character,Formula( :Level2 || :Cluster By Level2));

dt << Save();

//L4

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level3 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level3") << Data Type (Character);

column("Level3") << Data Type (Character);

dt << New Column("Level4", Character,Formula( :Level3 || :Cluster By Level3));

dt << Save();

//L5

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level4 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level4") << Data Type (Character);

column("Level4") << Data Type (Character);

dt << New Column("Level5", Character,Formula( :Level4 || :Cluster By Level4));

dt << Save();

//L6

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level5 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level5") << Data Type (Character);

column("Level5") << Data Type (Character);

dt << New Column("Level6", Character,Formula( :Level5 || :Cluster By Level5));

dt << Save();

//L7

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level6 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level6") << Data Type (Character);

column("Level6") << Data Type (Character);

dt << New Column("Level7", Character,Formula( :Level6 || :Cluster By Level6));

dt << Save();

//L8

obj = Hierarchical Cluster( Y( :tSNE1, :tSNE2, :tSNE3 ),Number of Clusters( 2 ) ,Standardize Data( 0 ),By( Level7 ),Method( "Ward" ));

obj << Save Clusters;

dt = Current Data Table();

column("Cluster By Level7") << Data Type (Character);

column("Level7") << Data Type (Character);

dt << New Column("Level8", Character,Formula( :Level7 || :Cluster By Level7));

dt << Save();

This script is for 8 level IBS, if needed you can add more levels. Modify mypath and dt before use.

You can find output table named as TS19.ALL.QC.3tSNE.IBS.

The results come with redundancy, for example, you will see a column named ‘Level X’ and a column named ‘Clustered by Level X’, the latter column is not necessary thus we can delete before we proceed.

**6. Differential gene expression**

First, we’ve tried our home made DGE formula to score each gene using: DGE score =

Here " " refers to mean UMI of a cluster, " " refers to grand mean UMI of the whole data set.

This is a score completely based on fold change, and obviously has its limitation. The fold change does not include the information about how many cells are positive for one specific gene, in other words, even if only one cell express a gene, the entire cluster might have a relative high mean UMI.

To overcome this, we applied R package ‘Monocle’ which is designed for scRNA seq. Also according to a paper comparing different DGE tools, Monocle seems to have very nice performance.

To run Monocle, we need to label each cell. In a specific run for cluster X, all cells from cluster X should be labelled, and all other cells should be labelled in a different way (for example, ‘X’ and ‘non\_X’). In this case, we are essentially comparing our target cluster with all other cells in data set. In the output of Monocle, it has p-value for each gene and related FDR. As we only care about up-regulated genes,

Using following script to run Monocle:

# this is a general Monocle script, run monocle on any cluster.

# make sure you have master UMI in environment

# this UMI should have 26305 cells

UMI\_raw <- read.csv("TS19.3NM.QC.rawUMI\_with\_name.csv", stringsAsFactors = F)

row.names(UMI\_raw) <- UMI\_raw$X

UMI\_raw$X <- NULL

# log transformed UMI

UMI\_raw\_log <- read.csv("TS19.ALL.3NM.QC.norm.log.UMI.csv", stringsAsFactors = F)

row.names(UMI\_raw\_log) <- UMI\_raw\_log$X

UMI\_raw\_log$X <- NULL

# drop 0 expression genes in raw\_UMI

UMI\_raw.sum <- rowSums(UMI\_raw)

n\_row <- nrow(UMI\_raw) # delete genes that have sum\_UMI as 0

count <- c()

for (i in 1:n\_row) {

if (UMI\_raw.sum[i] == 0) {

count <- c(count,i)

}

}

UMI\_raw <- UMI\_raw[-count,] # dimension 20347 by 26305

# drop o expression genes in log\_UMI

UMI\_raw\_log.sum <- rowSums(UMI\_raw\_log)

n\_row <- nrow(UMI\_raw\_log) # delete genes that have sum\_UMI as 0

count <- c()

for (i in 1:n\_row) {

if (UMI\_raw\_log.sum[i] == 0) {

count <- c(count,i)

}

}

UMI\_raw\_log <- UMI\_raw\_log[-count,] # dimension 20347 by 26305

# now prepare IBS cluster index

cluster\_table <- read.csv("TS19.ALL.QC.3tSNE.IBS.csv", stringsAsFactors = F)

cluster\_table$Column.1 <- gsub('-', '\\.', cluster\_table$Column.1) # dimension 26305 by 20, L1 at 6, L2 at 8, L3 at 10...

# create feature data, this can be used in any run

feature\_data <- matrix(nrow=20347, ncol=1)

row.names(feature\_data) <- row.names(UMI\_raw)

colnames(feature\_data) <- "gene\_short\_name"

feature\_data <- as.data.frame(feature\_data)

feature\_data$gene\_short\_name <- row.names(feature\_data)

# add gene short name

library(plyr)

feature\_data[,1] <- mapvalues(feature\_data[,1], from=conversion$Column.1, to=conversion$Column.2) # replace in dictionary style

# error message may show up because it's not 1-to-1 match

# Now I want to run Monocle DGE on all LevelX clusters

Monocle\_IBS\_level <- function (cluster) { # cluster should be written in manner like "Level5"

n\_col <- which(colnames(cluster\_table) == cluster)

cluster\_table\_temp <- cluster\_table[,c(2,n\_col)]

cluster\_list <- unique(cluster\_table\_temp[,2])

n\_clusters <- length(cluster\_list)

# create phenoData for monocle input, rows are cells, columns are annotations. ( this is in the loop)

# ATTENTION: MAKE SURE THE ORDER OF THE CELLS MATCH

library(monocle)

for (i in 1:n\_clusters) {

pheno\_data <- cluster\_table\_temp

cluster\_name <- cluster\_list[i] # grab cluster ID of i cluster

pheno\_data$Cluster <- pheno\_data[,2] # create an extra column

pheno\_data[which(pheno\_data[,2] != cluster\_name),3] <- paste("non",cluster\_name, sep="\_") # set new column either belongs to cluster i or not

row.names(pheno\_data) <- pheno\_data[,1]

pheno\_data[,1:2] <- NULL

# create object

pd <- new("AnnotatedDataFrame", data = pheno\_data)

fd <- new("AnnotatedDataFrame", data = feature\_data)

monocle\_data <- newCellDataSet(as.matrix(UMI\_raw),

phenoData = pd, featureData = fd, expressionFamily=negbinomial.size())

print(paste("starting estimate on", cluster\_name, sep=" "))

monocle\_data <- estimateSizeFactors(monocle\_data)

monocle\_data <- estimateDispersions(monocle\_data)

print(paste("starting DGE on", cluster\_name, sep=" "))

diff\_exp <- differentialGeneTest(monocle\_data, fullModelFormulaStr = "~Cluster", cores = 8)

write.csv(diff\_exp,paste("diff\_exp\_", cluster\_name, ".csv", sep=""))

print(i)

}

}

Monocle\_IBS\_level("Level6")

# now you have the diff\_exp data finished by Monocle

# next step is to calculate fold change

# a file with all IBS mean\_UMI should be loaded first:

mean\_UMI\_all <- read.csv("Mean\_UMI\_all\_IBS.csv", stringsAsFactors = F, check.names = F) #20347 rows(genes)

# make sure master UMI table is still loaded (UMI\_raw)

# calculate global mean\_UMI

UMI\_mean <- rowMeans(UMI\_raw\_log)

UMI\_mean <- as.data.frame(UMI\_mean)

# #############################################

#now set wd as where you store diff\_exp data#

#############################################

# read in from folder

temp = list.files(pattern="\*.csv")

n\_clusters <- length(temp)

# make output matrix

output <- matrix(nrow=400, ncol=n\_clusters)

output <- as.data.frame(output)

# make another output for sum\_UMI table

output\_sumUMI <- matrix(nrow=ncol(UMI\_raw), ncol=n\_clusters)

output\_sumUMI <- as.data.frame(output\_sumUMI)

rownames(output\_sumUMI) <- colnames(UMI\_raw)

# for each diff\_exp file, apply q-value filter and then FC filter

for (i in 1:n\_clusters) {

diff\_data <- read.csv(temp[i], stringsAsFactors = F)

cluster\_name <- strsplit(temp[i], "\_")[[1]][3] # here I'm grabbing cluster name from file name, split by '\_' here

cluster\_name <- strsplit(cluster\_name, "\\.")[[1]][1] # then split by '.'

# now filter data by q-value threshhold 0.05

diff\_data <- diff\_data[which(diff\_data$qval <= 0.05),]

# now pullout mean\_UMI of target cluster from mean\_UMI\_all

cluster\_mean <- mean\_UMI\_all[,cluster\_name]

cluster\_mean <- as.data.frame(cluster\_mean)

row.names(cluster\_mean) <- row.names(UMI\_mean)

FC <- cluster\_mean/UMI\_mean

colnames(FC) <- "FC"

# add a column to diff\_data to store FC

diff\_data$FC <- 0

# pullout FC info

for (j in 1:nrow(diff\_data)) {

diff\_data[j,7] <- FC[which(row.names(FC) == diff\_data[j,1]),1]

}

# sort by FC

diff\_data\_sort <- diff\_data[with(diff\_data, order(-FC)), ]

# filter data by FC threshold 2

diff\_data\_sort <- diff\_data\_sort[which(diff\_data\_sort$FC >= 2),]

write.csv(diff\_data\_sort, paste(cluster\_name, "diff\_with\_FC.csv", sep="\_"))

# pullout top 400 to output

if (nrow(diff\_data\_sort) >= 400) {

gene\_list <- diff\_data\_sort$gene\_short\_name[1:400]

ENS\_list <- diff\_data\_sort$X[1:400]

} else {

gene\_list <- diff\_data\_sort$gene\_short\_name

ENS\_list <- diff\_data\_sort$X

length(gene\_list) <- 400

}

# now the ENS ID of top400 genes stored in ENS\_list, can be used to calculate sum\_UMI

colnames(output\_sumUMI)[i] <- cluster\_name

UMI\_subset <- UMI\_raw\_log[ENS\_list,]

output\_sumUMI[,i] <- colSums(UMI\_subset)

# save to output

output[,i] <- gene\_list

colnames(output)[i] <- cluster\_name

}

#writ out query list

write.csv(output,"query\_gene\_list\_l5.csv")

write.csv(output\_sumUMI,"sum\_UMI\_DGE\_l5.csv")

It is preferred to run on a server and set multiple cores for running Monocle. It typically takes 20 minutes for each cluster.

In the script we applied internal threshold for fold change >2 and q-value < 0.05 (FDR). For each cluster, top 400 genes were selected for downstream analysis. If <400 genes left then all those genes selected.

7. Batch GA query

We used to have a php tool to run this, but Flora is not working. As an alternative way, we have a python script doing the job.

The python script will read each column from query\_gene\_list.csv from step 6 output and send the genes to Gene Analytics. To run the script, you must modify the number of clusters and input file name in script. This script calls firefox browser, it should be properly installed. After finishing all query, each will open a download window. In other words, you should have N windows pending after running for N clusters. Manually click save or use a mouse clicker could help to save all those GA reports.

8. GA report post process

In this step, we need to extract useful information from multiple GA reports and summarize all of them into one .csv file.

8.1 Rename downloaded GA reports and sanity check

First we have to rename all GA reports. They were downloaded with a name of time stamp. This can be useful when checking if proper number of reports were downloaded, sometimes there’s a server lag or loss of connection, which lead to several clusters not properly queried. If number of GA reports not equal to number of clusters, then you can check the time stamp to see which cluster is missed. One way to get around this is to modify the sleep() time in python script. Try to increase the time so that there’s a lower chance of bad query due to connection/responding issue.

When we have all GA reports we need in a folder, we can use another script called applyNames\_revised.py to rename all these reports by calling:

python applyNames\_revised.py keyFile dataFolder

Here the keyFile contains all names of the clusters starting from 3rd column. DataFolder is the folder with all GA reports.

8.2 Aggregating reports with background subtraction

To do background subtraction, we ran 1000 random trials with 400 gene input. For cell type and tissue (tissue is not considered in our final version pipeline, but can be added), we use z-score subtraction. To be more specific, we calculate the mean and standard deviation of each possible match, then use formula:

to determine the z-score.

For GO:Biological Processes and phenotype, we cannot retrieve all possible entities by 1000 random trials. So there will be missing data for ‘std’, as a result we cannot use z-score background subtraction. Instead we use direct subtraction: .

By running an R script, it reads in all excel sheets in assigned directory, and specifically extract information (cell type, GOBP and phenotype) from designated sheet. For cell type, one entity may appear multiple times as they come from different anatomical compartments. The script will sum all those scores to make a final score, then do background subtraction. At the end it will output a table for each category, with columns as clusters and rows are all possible matches.

8.3 Trim for JMP input

This part is also integrated into the same R script. Because JMP word cloud does not support scores as weight, we have to turn the scores into word frequency. For example, if term X gets a score of 5.5 in one cluster, we will time the score by 10 to get 55 so that we have a resolution of 1 decimal place. Then we would repeat term X 55 times (55 rows of term X). As a result, the score 5.5 will be turned into word frequency.

In output file, clusters are on columns. And rows are repeated words weighted by scores. Another output file will be generated as ‘phrase cloud’ input. In that file, we replace the space between the words in one entity with ‘\_’, so that we can treat the entire phrase as our input. To do these, we have to replace “, “, “-“, “/” and “ “ with “\_”, this is also integrated in script.

9. Generate word/phrase cloud

We do this in JMP. Manually you can select the column you need and go to analyze-> text explorer. In tokenizing you want ‘basic’ so that you can have both phrase clouds and word clouds. Then in output window you need to select specific words as ‘stop words’ as they don’t provide information we need. Words like ‘cell’, ‘positive’, ‘negative’ can be added to stop words. Then click on the red triangle and in display option check word cloud. From the word cloud panel change the layout to ordered.

Obviously we cannot manually do all clusters by ourselves. A JMP script will output all word clouds with all columns (except the first). To run the script, you need to modify the input/output path and file names, as well as number of clusters. You need to run both word and phrase cloud for all categories.