Self Organizing Maps (SOM): Example using RNAseq reads

Part 2: Running PCA and SOM

Principle Component Analysis

When running SOM, it is sometimes helpful to first run PCA to see the general spread of your dataset. Later you can map your SOM cluster results back onto the PCA and see if your PCA clusters can be further defined by the gene expression patterns resulting from you SOM results.

Required Libraries

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
library(reshape)
library(kohonen)
```

```
## Loading required package: class

##
## Attaching package: 'class'

## The following object is masked from 'package:reshape':
##
##     condense

## Loading required package: MASS
```

```
library(RColorBrewer)
```

Read in data

First read in file that came from part 1 `allGeneList.csv`. This is a list of genes from all DE analysis in WT that were performed previously. They were all cancatenated, then duplicate genes were removed. In addition the mean was calculated from the replicates of each type.

The first step is to get the data into the right format. First column being the genes, while the subsequent columns are the different libraries (type).

```
mostDEgenes <- read.csv("../data/allGeneList.csv")
head(mostDEgenes)
```

```
##     type genotype N       mean         sd         se               gene
## 2   Ambr       wt 3  17.193379  21.125010 12.1965304 Solyc00g005070.1.1
## 3 Aother       wt 5   4.523554   1.835113  0.8206874 Solyc00g005070.1.1
## 4   Bmbr       wt 4  12.645609  18.655600  9.3277998 Solyc00g005070.1.1
## 5 Bother       wt 4   3.461582   1.789494  0.8947468 Solyc00g005070.1.1
## 6   Cmbr       wt 6   4.180826   2.818555  1.1506704 Solyc00g005070.1.1
## 7 Cother       wt 3 105.966982 168.647913 97.3689180 Solyc00g005070.1.1
```

```
mostDEgenes <- mostDEgenes[c(7, 1, 4)] #keep only needed columns (gene, type, mean)

#Change from long to wide data format
mostDEgene.long <- cast(mostDEgenes, gene ~ type, value.var = mean, fun.aggregate = "mean")
```

```
## Using mean as value column.  Use the value argument to cast to override this choice
```

```
mostDEgene.long <- as.data.frame(mostDEgene.long) #transformation.
names(mostDEgene.long) #check
```

```
## [1] "gene"   "Ambr"   "Aother" "Bmbr"   "Bother" "Cmbr"   "Cother"
```

```
# set up for PCA
scale_data <- as.matrix(t(scale(t(mostDEgene.long[c(2:7)]))))

#Principle Component Analysis
pca <- prcomp(scale_data, scale=TRUE)

summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5      PC6
## Standard deviation     1.3989 1.1164 1.0575 0.9617 0.8679 9.21e-16
## Proportion of Variance 0.3262 0.2077 0.1864 0.1541 0.1255 0.00e+00
## Cumulative Proportion  0.3262 0.5339 0.7203 0.8745 1.0000 1.00e+00
```

```
pca.scores <- data.frame(pca$x)
```

I put all the information from each analysis back together throughout the process. Makes visualiztion easier down the line.

```
# Add back to original so everything is together.
data.val <- cbind(mostDEgene.long, scale_data, pca.scores)
head(data.val)
```

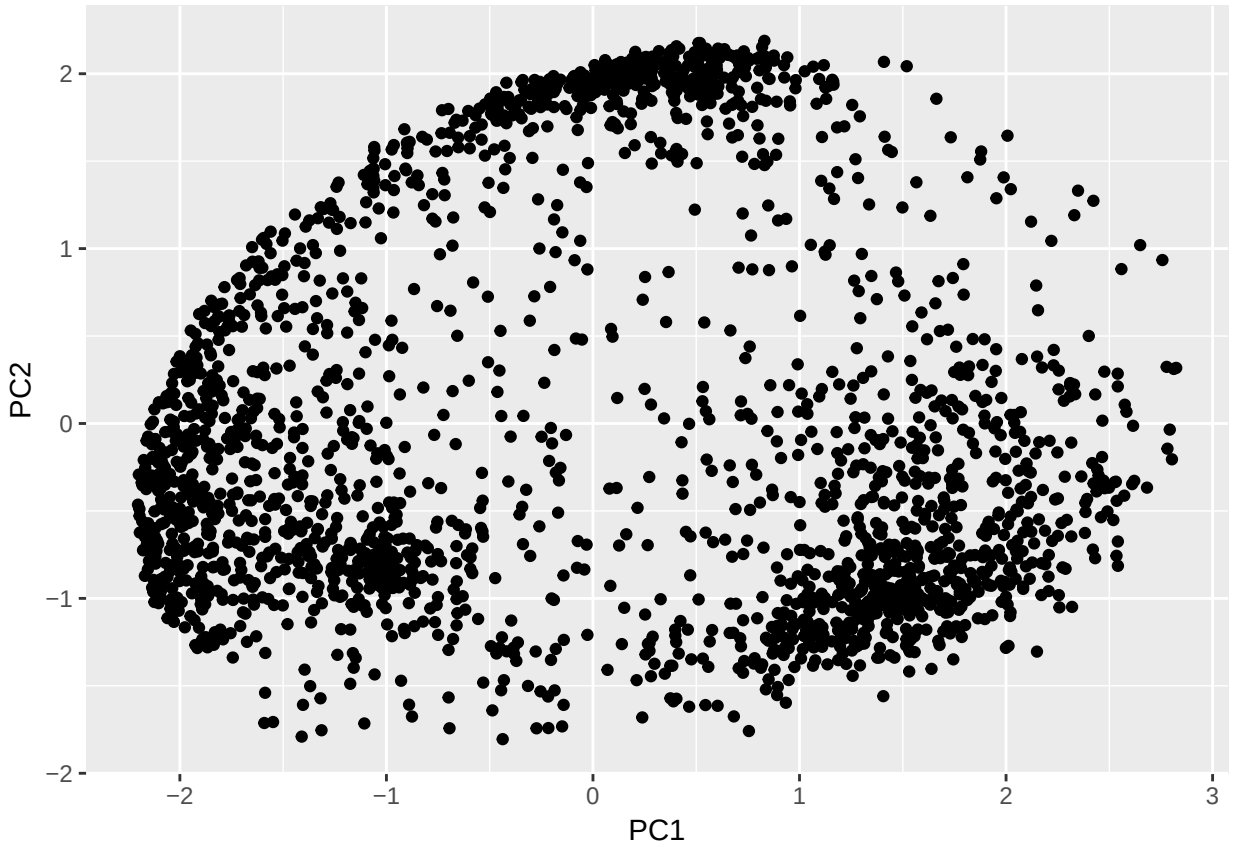
```
##                  gene         Ambr      Aother        Bmbr     Bother
## 1 Solyc00g005070.1.1   17.1933786   4.5235545  12.6456091   3.461582
## 2 Solyc00g005080.1.1   16.0214526  10.1276589   7.5621826   8.495561
## 3 Solyc00g005840.2.1   19.7457984  14.0743283  11.4830294  83.513288
## 4 Solyc00g005870.1.1    0.1335935   0.6413574   3.1240810   2.778641
## 5 Solyc00g006470.1.1 2455.5110955 605.8620487 108.0946960 360.481814
## 6 Solyc00g006670.2.1   66.9760013   5.9946868   0.6022608   7.070824
##         Cmbr     Cother       Ambr     Aother        Bmbr     Bother
## 1   4.180826 105.966982 -0.1857292 -0.5008022 -0.29882308 -0.5272113
## 2   8.413141  25.028796  0.5010396 -0.3641074 -0.74069236 -0.6036822
## 3  15.811238  13.981084 -0.2380746 -0.4399334 -0.53216284  2.0315363
## 4   1.780475  12.461880 -0.7372084 -0.6255713 -0.07971921 -0.1556676
## 5 499.447526 390.760289  2.0023986 -0.1524161 -0.73230794 -0.4382806
## 6  11.065297   4.677613  2.0225935 -0.4000473 -0.61427544 -0.3572950
##         Cmbr     Cother        PC1        PC2         PC3        PC4
```

```
## 1 -0.5093251  2.0218909  0.4123820  1.9720828  0.62320725 0.23135121
## 2 -0.6157807  1.8232230  0.4094212  1.4972020  1.14022326 0.75216342
## 3 -0.3781133 -0.4432521 -0.9687199 -0.9405744 -0.45944789 0.47066745
## 4 -0.3751246  1.9732912  0.2002039  2.0773175  0.08514379 0.01114138
## 5 -0.2763875 -0.4030066  1.1163748 -1.0607219  0.54496613 0.93810521
## 6 -0.1986042 -0.4523715  1.3269062 -1.0423991  0.32411614 0.97528268
##          PC5           PC6
## 1  0.1378721 -1.110223e-15
## 2 -0.1591747 -1.332268e-15
## 3  1.6558976  1.498801e-15
## 4  0.5230315 -9.992007e-16
## 5 -0.4747755 -5.551115e-16
## 6 -0.2106054 -1.665335e-16
```

Visualizing the PCA

By eye you can see my data is clustered into three major clusters.

```
p <- ggplot(data.val, aes(PC1, PC2))
p + geom_point()
```



Self Organizing Map

Running SOM large

The data

Clustering is performed using the `som()` function on the scaled gene expression values.

```
# Check where the scaled gene expression values are.
names(data.val)
```

```
##  [1] "gene"   "Ambr"   "Aother" "Bmbr"   "Bother" "Cmbr"   "Cother"
##  [8] "Ambr"   "Aother" "Bmbr"   "Bother" "Cmbr"   "Cother" "PC1"
## [15] "PC2"    "PC3"    "PC4"    "PC5"    "PC6"
```

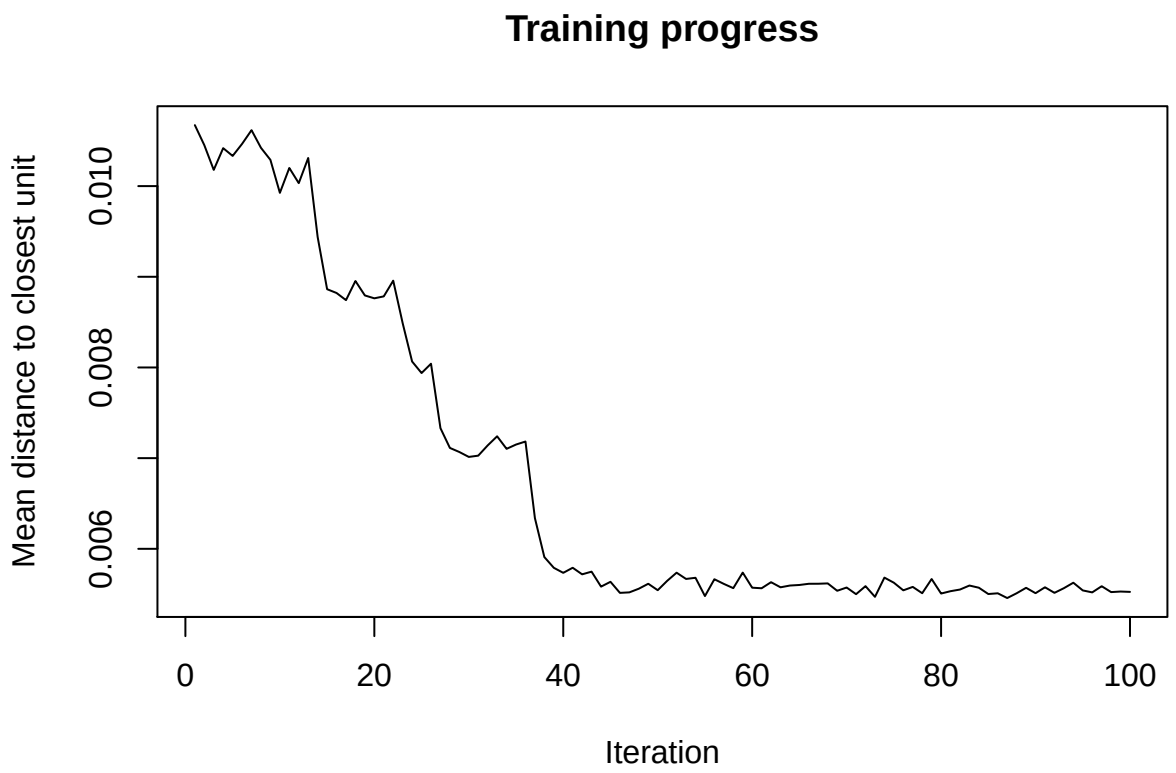
```
#Subset for SOM in a matrix.
#som() only works on matrices NOT dataframes
#subset only the scaled gene expression values
som.data <- as.matrix(data.val[,c(8:13)])

# Set seed, just make sure you keep the same.
# Has to do with the randomization process.
set.seed(2)

#This is where you change the size of the map
som <- som(data=som.data, somgrid(6,6,"hexagonal"))

summary(som)
```

```
## som map of size 6x6 with a hexagonal topology.
## Training data included; dimension is 2249 by 6
## Mean distance to the closest unit in the map: 0.4130783
```

Training Plot ("changes") - Large

This shows a hundred iterations. Training decreases with iterations and plateaus at around 40 iterations. Ideally you want the the training to reach a minimum plateau.

```
plot(som, type ="changes")
```

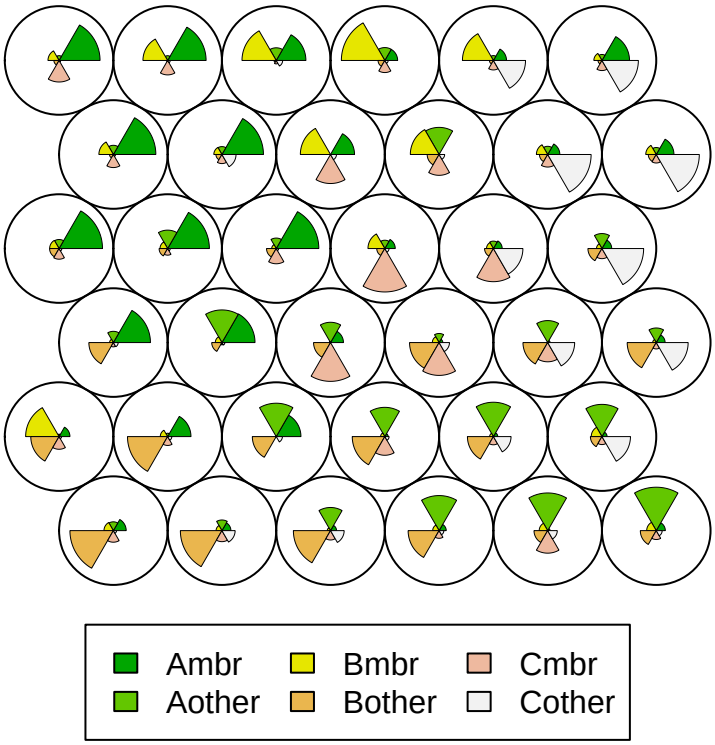


Code Plot - Large

The the code plot shows each cluster and the node wieght vectors or "codes" associated with each node. These are made up of the original normalized values of the original values used to generate the map. You should see patterns of clustering.

The fan chart in the center of the clusters reveals the characteristics that define how the genes were clustered into each particular cluster. For instance if one cluster has only one large fan piece, say for Bother, this is telling us that most of the genes in this cluster were grouped because of similar normalized gene count value of the Bother region. We do not know the degree, it could mean all these genes are up-regulated or down-regulated in the Bother region, but we do not know which at this point.

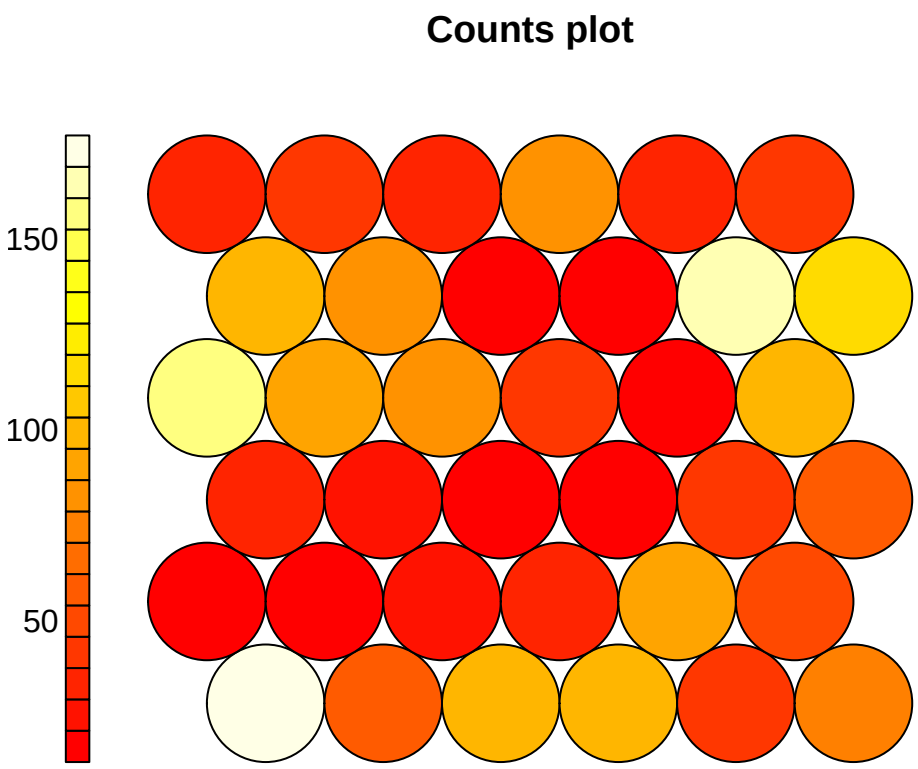
```
plot(som, type = "codes")
```



Count Plot - Large

This tells you how many genes are in each of the clusters. The count plot can be used as a quality check. Ideally you want a uniform distribution. If there are some peaks in certain areas, this means you should likely increase the map size. If you have empty nodes you should decrease the map size [1].

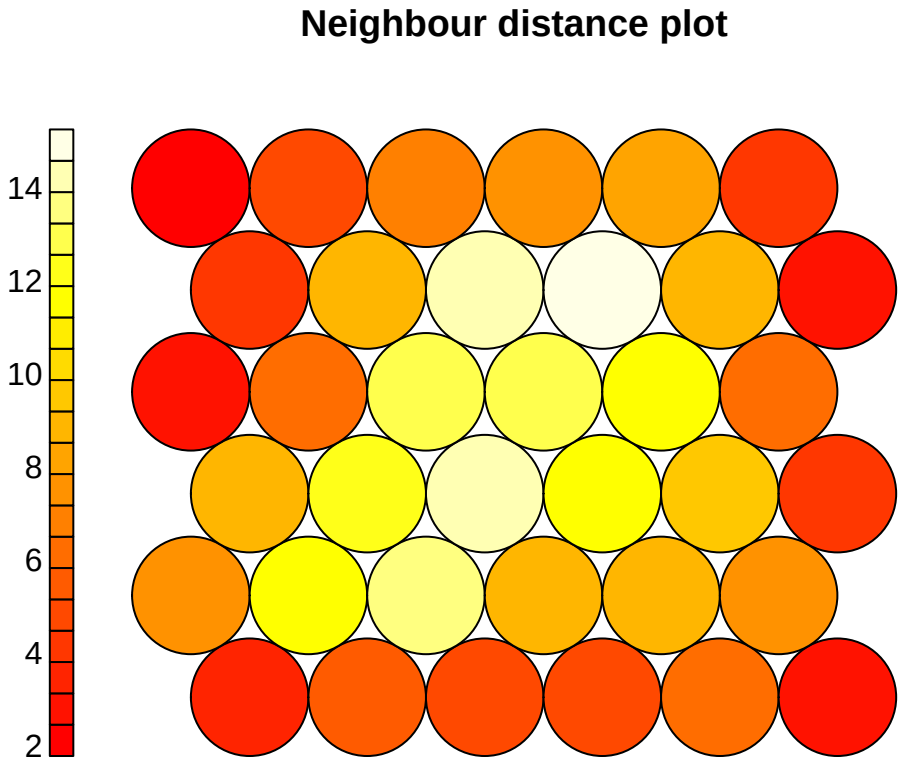
```
plot(som, type = "counts")
```



Distance Neighbour Plot - Large

This is sometimes called the "U-Matrix", it can help identify further clustering. Areas of low neighbour distance indicate groups of nodes that are similar and the further apart nodes indicate natural "borders" in the map.

```
plot(som, type="dist.neighbours")
```

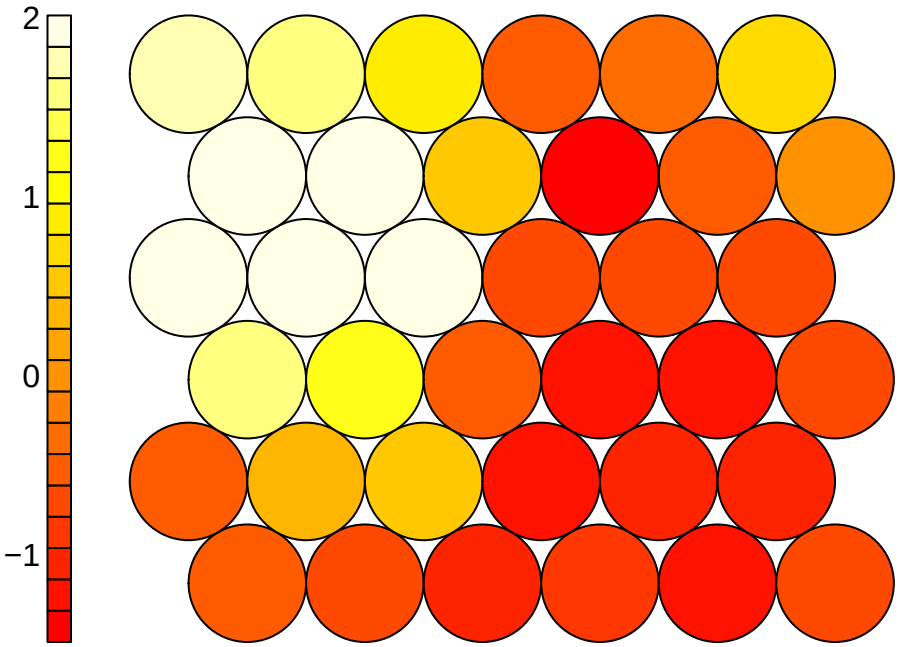


Heatmaps - large

This shows the distribution of each type

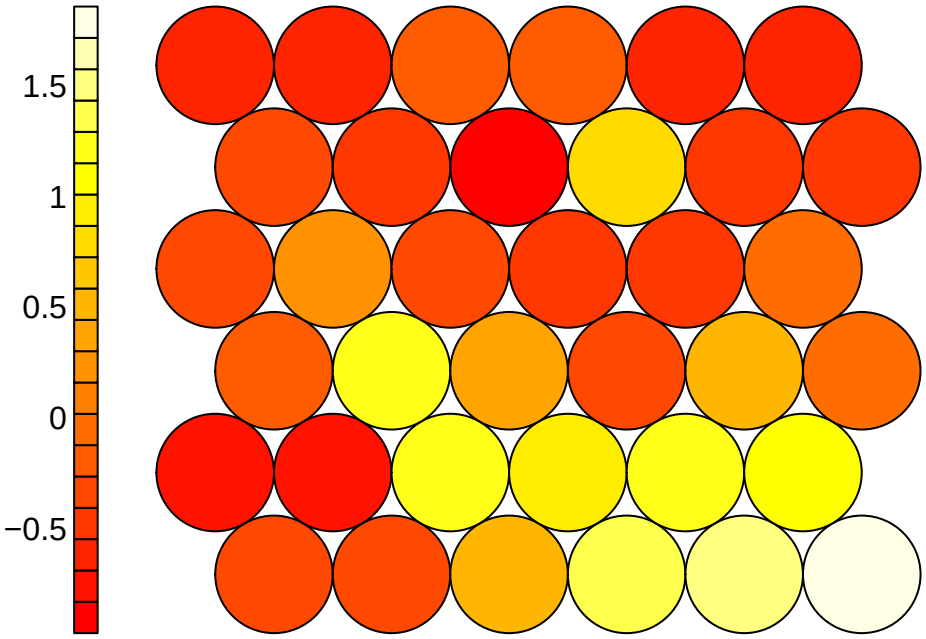
```
#changed to dataframe to extract column names easier.
som$data <- data.frame(som$data)

#This is just a loop that plots the distribution of each tissue type across the map.

for (i in 1:6){
  plot(som, type = "property", property = som$codes[,i], main=names(som$data)[i])
  print(plot)
  }
```

Ambr

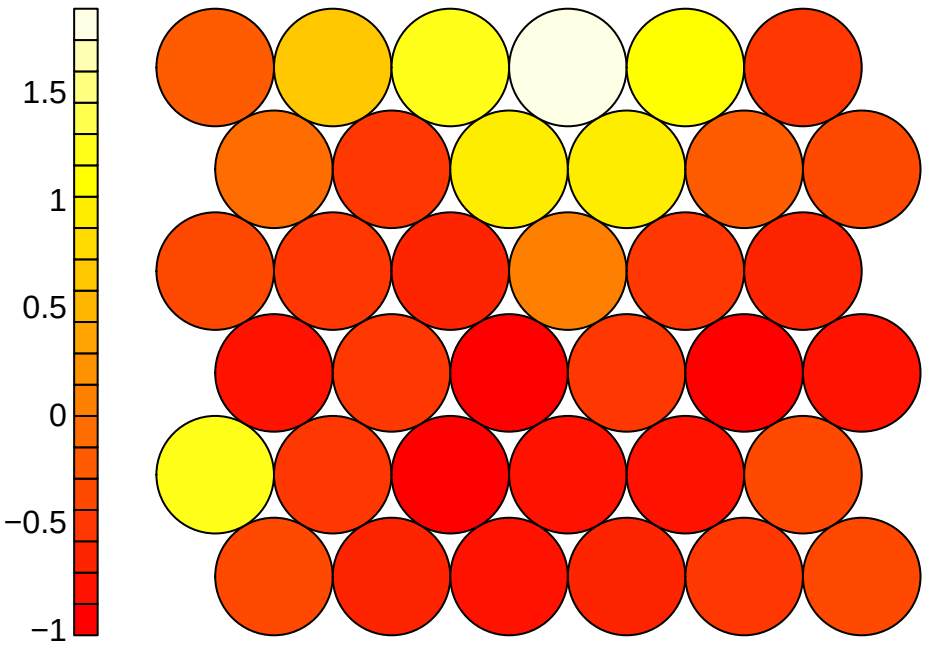



```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7fe4b45a1e78>
## <environment: namespace:graphics>
```

Aother

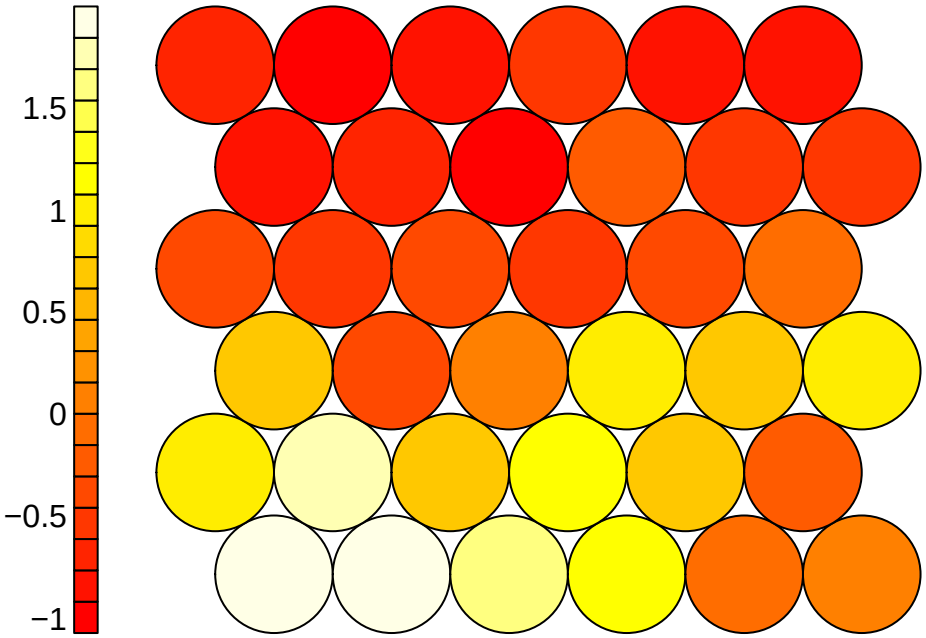


```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7fe4b45a1e78>
## <environment: namespace:graphics>
```



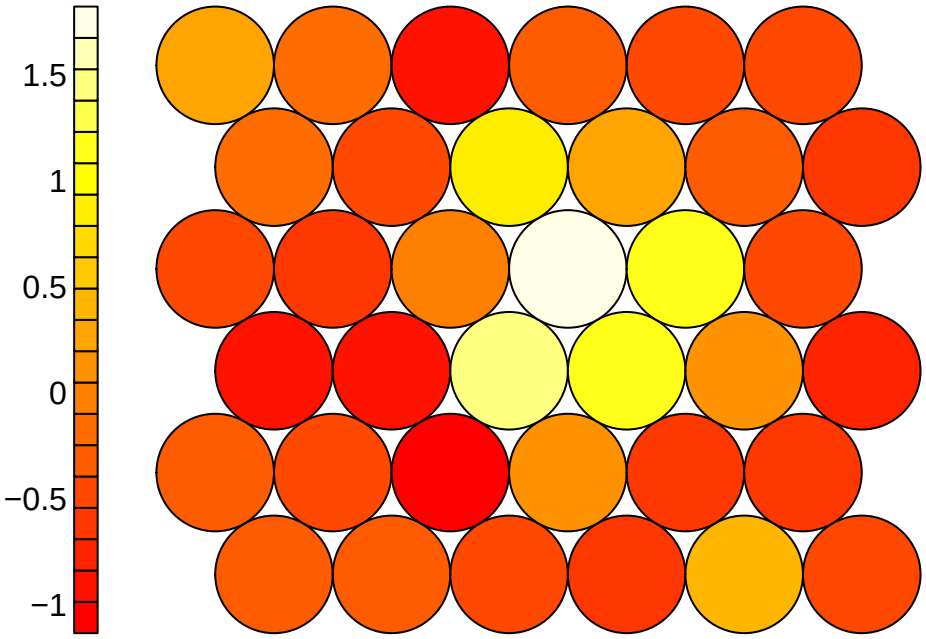

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7fe4b45a1e78>
## <environment: namespace:graphics>
```

Bother

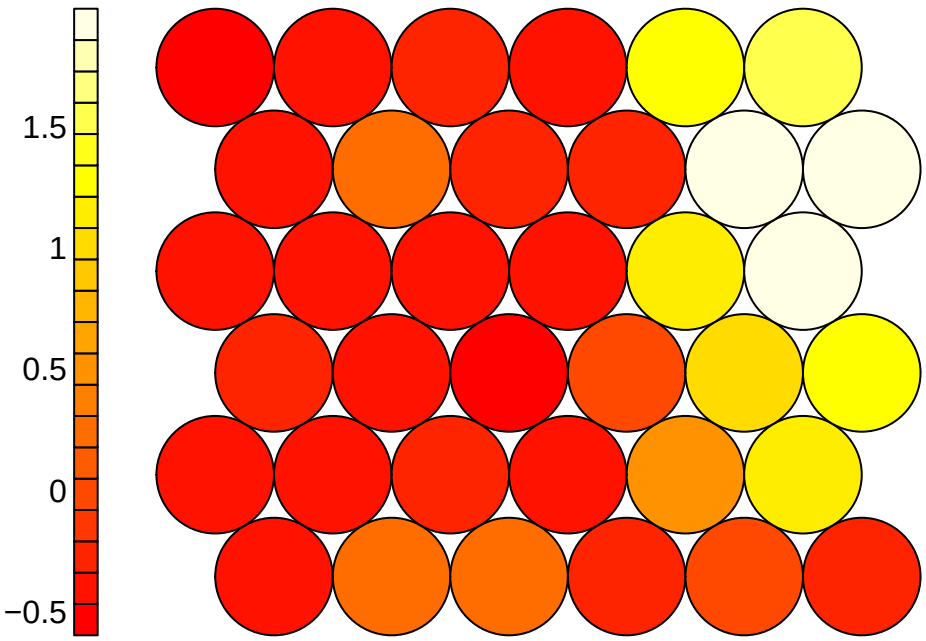



```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7fe4b45a1e78>
## <environment: namespace:graphics>
```

Cmbr



```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7fe4b45a1e78>
## <environment: namespace:graphics>
```



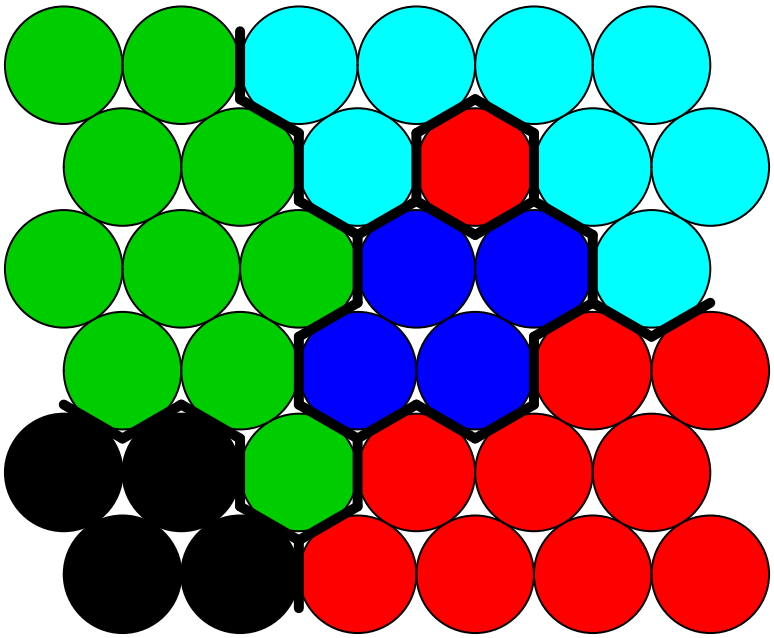

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7fe4b45a1e78>
## <environment: namespace:graphics>
```

Further clustering

You can further split the clusters into a smaller set of clusters using hierarchical clustering.

```
## use hierarchical clustering to cluster the codebook vectors
som_cluster <- cutree(hclust(dist(som$codes)), 5)
# plot these results:
plot(som, type="mapping", bgcol = som_cluster, main = "Clusters")
add.cluster.boundaries(som, som_cluster)
```

Clusters



```
# Attach the hierchal cluster to the larger dataset data.val.
som_clusterKey <- data.frame(som_cluster)
som_clusterKey$unit.classif <- c(1:36)

data.val <- cbind(data.val,som$unit.classif,som$distances)
head(data.val)
```

```
##                  gene         Ambr      Aother        Bmbr     Bother
## 1 Solyc00g005070.1.1.1   17.1933786   4.5235545  12.6456091   3.461582
## 2 Solyc00g005080.1.1.1   16.0214526  10.1276589   7.5621826   8.495561
## 3 Solyc00g005840.2.1.1   19.7457984  14.0743283  11.4830294  83.513288
## 4 Solyc00g005870.1.1.1    0.1335935   0.6413574   3.1240810   2.778641
## 5 Solyc00g006470.1.1.1 2455.5110955 605.8620487 108.0946960 360.481814
## 6 Solyc00g006670.2.1.1   66.9760013   5.9946868   0.6022608   7.070824
##         Cmbr     Cother       Ambr     Aother        Bmbr     Bother
## 1   4.180826 105.966982 -0.1857292 -0.5008022 -0.29882308 -0.5272113
## 2   8.413141  25.028796  0.5010396 -0.3641074 -0.74069236 -0.6036822
## 3  15.811238  13.981084 -0.2380746 -0.4399334 -0.53216284  2.0315363
## 4   1.780475  12.461880 -0.7372084 -0.6255713 -0.07971921 -0.1556676
## 5 499.447526 390.760289  2.0023986 -0.1524161 -0.73230794 -0.4382806
## 6  11.065297   4.677613  2.0225935 -0.4000473 -0.61427544 -0.3572950
##         Cmbr     Cother        PC1        PC2         PC3        PC4
## 1 -0.5093251  2.0218909  0.4123820  1.9720828  0.62320725 0.23135121
## 2 -0.6157807  1.8232230  0.4094212  1.4972020  1.14022326 0.75216342
## 3 -0.3781133 -0.4432521 -0.9687199 -0.9405744 -0.45944789 0.47066745
## 4 -0.3751246  1.9732912  0.2002039  2.0773175  0.08514379 0.01114138
## 5 -0.2763875 -0.4030066  1.1163748 -1.0607219  0.54496613 0.93810521
## 6 -0.1986042 -0.4523715  1.3269062 -1.0423991  0.32411614 0.97528268
##          PC5           PC6 som$unit.classif som$distances
```

```
## 1  0.1378721 -1.110223e-15              30    0.02231733
## 2 -0.1591747 -1.332268e-15              36    0.29893594
## 3  1.6558976  1.498801e-15               1    0.06933208
## 4  0.5230315 -9.992007e-16              29    0.24555383
## 5 -0.4747755 -5.551115e-16              21    0.05962820
## 6 -0.2106054 -1.665335e-16              21    0.04081959
```

```
#Merge data.val with som_clusterKey
##change data.val to match som_cluster key
names(data.val)[20] <- "unit.classif"

data.val <- merge(data.val, som_clusterKey, by.x = "unit.classif" ) #ignore warning, this is what you w
```

```
## Warning in merge.data.frame(data.val, som_clusterKey, by.x =
## "unit.classif"): column names 'Ambr', 'Aother', 'Bmbr', 'Bother', 'Cmbr',
## 'Cother' are duplicated in the result
```

```
# Write out your data at the end to save your results of the SOM
#write.table(data.val, file="../data/analysis1.som.data.txt")
```

Visualize on PCA

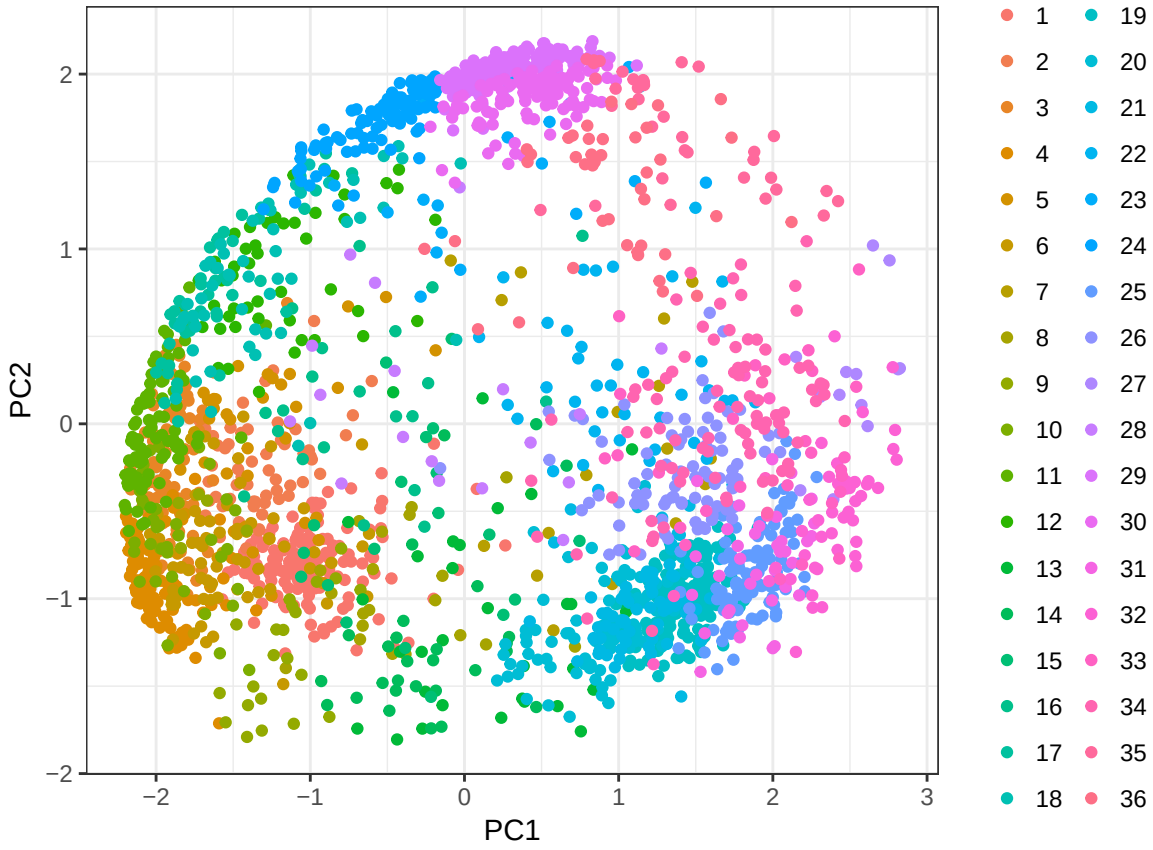
```
# read in data
# data.val <- read.table("../data/analysis1.som.data.txt",header=TRUE)

names(data.val)
```

```
##  [1] "unit.classif"  "gene"          "Ambr"          "Aother"
##  [5] "Bmbr"          "Bother"        "Cmbr"          "Cother"
##  [9] "Ambr"          "Aother"        "Bmbr"          "Bother"
## [13] "Cmbr"          "Cother"        "PC1"           "PC2"
## [17] "PC3"           "PC4"           "PC5"           "PC6"
## [21] "som$distances" "som_cluster"
```

```
dim(data.val)
```

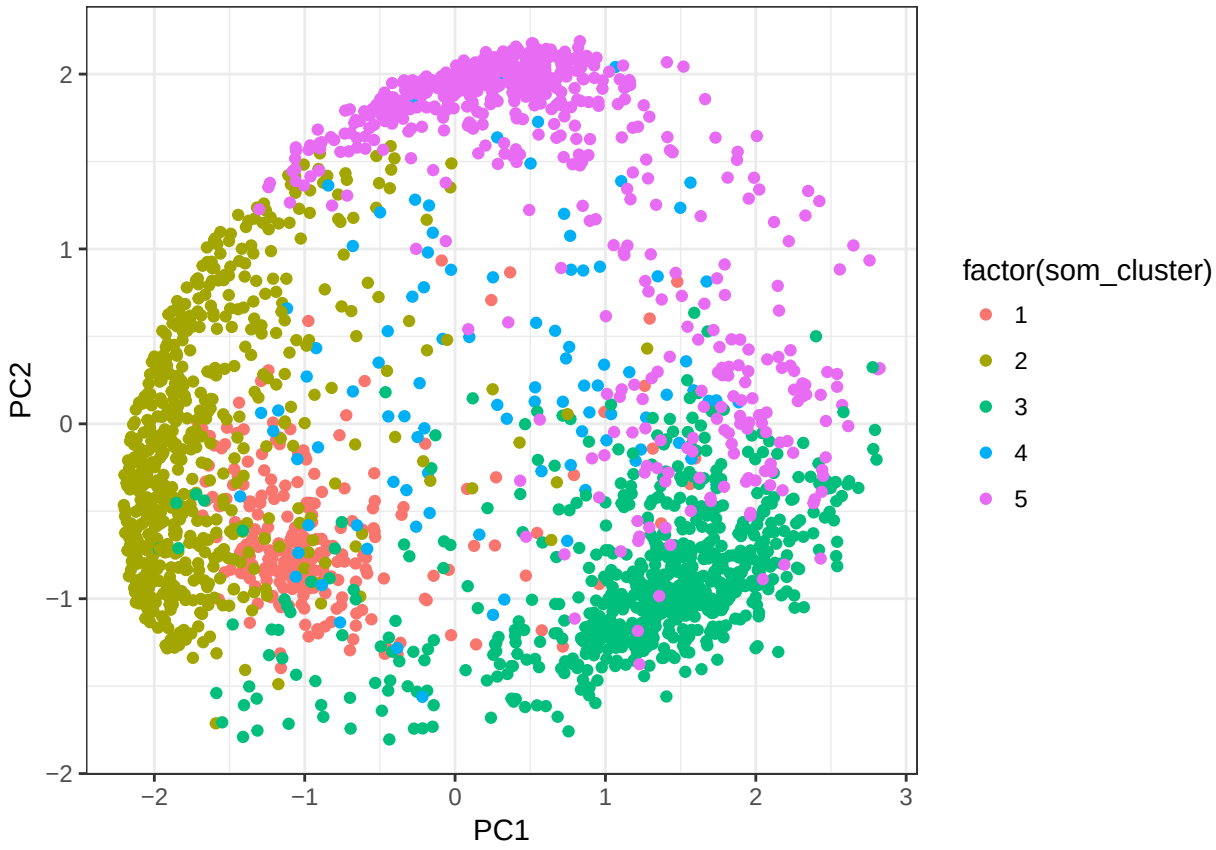
```
## [1] 2249   22
```

```
p <- ggplot(data.val, aes(PC1, PC2, colour=factor(unit.classif)))
p + geom_point() + theme_bw()
```



Notice I am plot.data$unit.classif, which is the clusters generated when we used `som()`. You can also use the assignments made from the hierarchical clustering.

```
p <- ggplot(data.val, aes(PC1, PC2, colour=factor(som_cluster)))
p + geom_point() + theme_bw()
```



Activity

The size of the map is something that may cause differences in the genes that are clustered. The only way to see how this is affects what we see is to compare the clusters of a small and large map.

Make a smaller SOM map using this data and visualize the differences on the PCA.

Acitvity Conclusions

Using a small map size (3,2), I found they cluster according to tissue type. This makes the interpretation of the results pretty straight forward. My only worry is that the map might not be large enough, considering the Kohenen Package Manual Vignette [1] suggests that you pick the size of the map based on count distribution, the goal being an even distribution, with no "peak" counts in any one cluster while also having no empty clusters.

References:

1. Kohenen Package Manual PDF