COmbined Mapping of Multiple clUsteriNg ALgorithms (COMMUNAL): A Robust Method for Selection of Cluster Number K

Albert Chen
Stanford University

Timothy E Sweeney
Stanford University

Olivier Gevaert
Stanford University

February 11, 2015

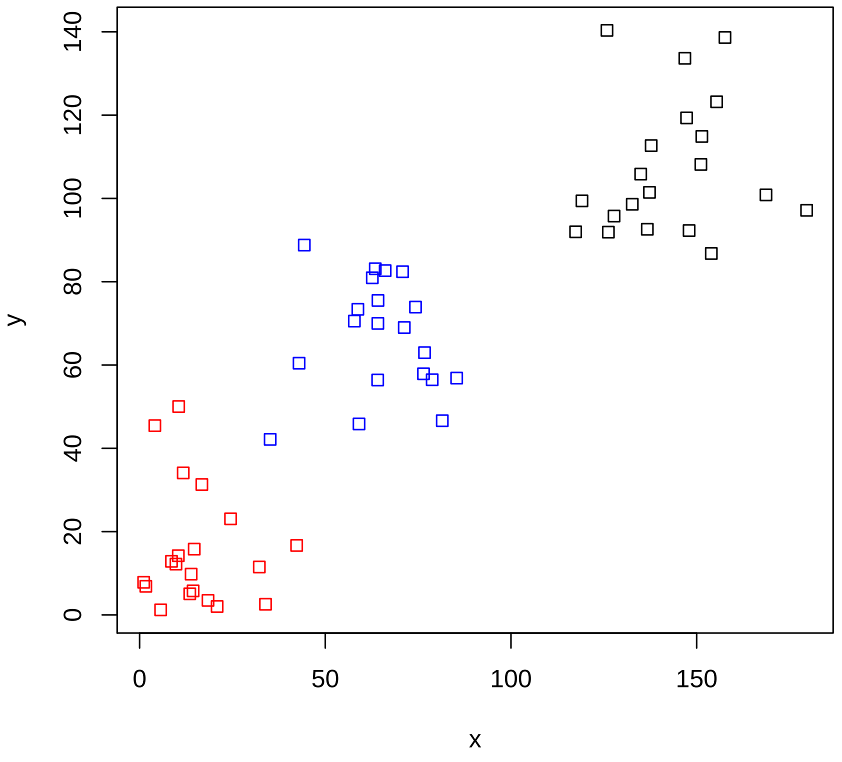
Introduction

We are often interested to understand the underlying structure of our data. One such way is to group the data into clusters of similar data points. There are many algorithms that can find such clusters (e.g. kmeans, hierarchical, sota), and we aggregate their results to arrive at a more robust result. However, we don't initially know what the best number of clusters is for the data. For biological data, we may believe there is a 'true' number of clusters with some biological meaning. We would like to discover the optimal number of clusters k. After discovering this, we can combine the cluster assignments of each clustering algorithm (with k clusters) to arrive at a robust clustering. This package allows for both these steps: first identifying the best value of k, and then combining the cluster assignments from multiple algorithms to identify 'core' clusters in the data.

Tutorial

Let's start with a small example. We first create an artificial data set with 3 distinct clusters. We would hope to "discover" that the optimal $k = 3$.

```
> ## create artificial data set with 3 distinct clusters in two dimensions
> set.seed(1)
> V1 = c(abs(rnorm(20, 2, 20)), abs(rnorm(20, 65, 15)), abs(rnorm(20, 140, 20)))
> V2 = c(abs(rnorm(20, 2, 20)), abs(rnorm(20, 65, 15)), abs(rnorm(20, 105, 20)))
> data <- t(data.frame(V1, V2))
> colnames(data) <- paste("Sample", 1:ncol(data), sep="")
> rownames(data) <- paste("Gene", 1:nrow(data), sep="")
> plot(V1, V2, col=rep(c("red", "blue", "black"), each=20), pch=rep(c(0,1,2), each=100),
+      xlab="x", ylab="y")
```



Identifying k

The first step is to run a few clustering algorithms on our data, trying each value of k we think is reasonable. The function `COMMUNAL` runs the clustering algorithms. The main arguments are the values of k (number of clusters) to try, and the clustering algorithms to use. The available clustering algorithms are `hierarchical, kmeans, diana, fanny, som, model, sota, pam, clara, agnes, ccp-hc, ccp-km, ccp-pam, nmf`. In this list, `nmf` corresponds to `nmf` in package `NMF`, `ccp-xx` corresponds to `xx` in package `ConsensusClusterPlus`, and the rest match to the algorithm of the same name in package `clValid`. By default, it runs `hierarchical` and `kmeans`.

By default, seven different validation measures are evaluated (`Connectivity`, `dunn`, `wb.ratio`, `g3`, `g2`, `pearsongamma`, `avg.silwidth`, `sindex`). Other validation metrics are available as well. Each of these is a measure of how well a particular clustering separates the data into clusters. With the exception of "Connectivity", which is calculated by `clValid::connectivity`, these are calculated with `fpc::cluster.stats`.

These settings can be adjusted with the `clus.methods` and `validation` parameters. Here we run the clustering for each value of k between 2 and 5 inclusive.

```
> ## run COMMUNAL with defaults
> library(COMMUNAL)
> ks <- seq(2,5)
> result <- COMMUNAL(data=data, ks=ks)
```

The return value is an object of (reference) class `COMMUNAL` (same name as the function). The default print method shows the k's used, and the original call.

```
> result

Reference class "COMMUNAL"

        ks: 2, 3, 4, 5

        Call: COMMUNAL(data = data, ks = ks)

(Use the getClustering method to extract data frame of cluster assignments)
```

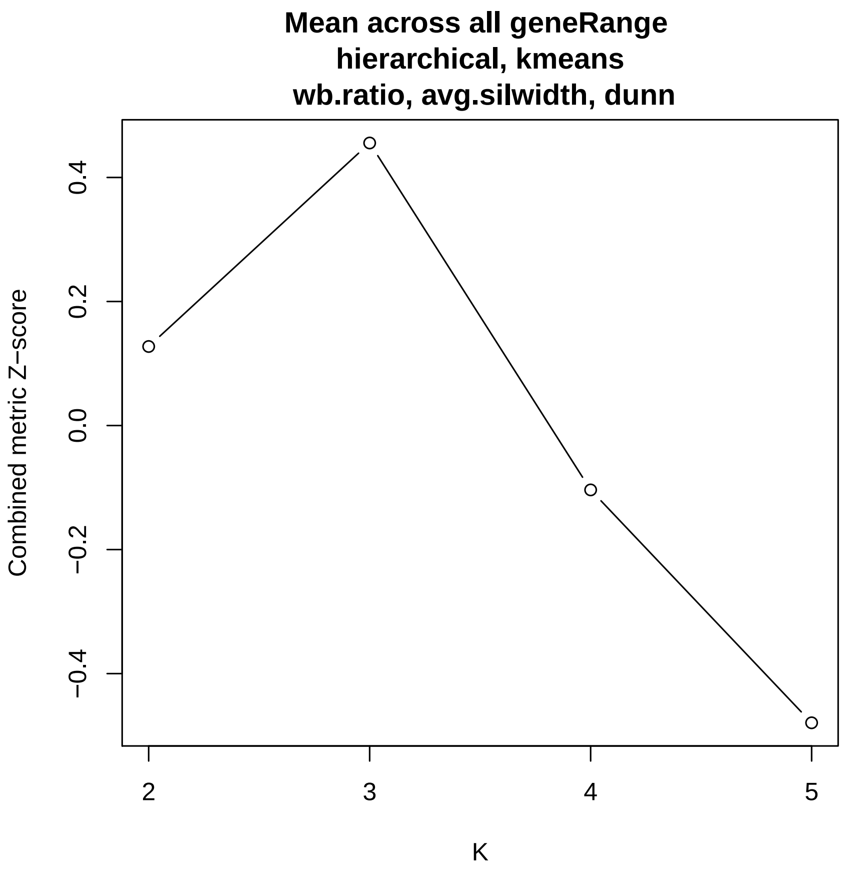
For each algorithm and value of k, the validation measures are computed. They can be accessed like this.

```
> str(result$measures)

 num [1:8, 1:4, 1:2] 0 0.4491 0.3634 0.0415 0.8807 ...
 - attr(*, "dimnames")=List of 3
  ..$ : chr [1:8] "Connectivity" "dunn" "wb.ratio" "g3" ...
  ..$ : chr [1:4] "2" "3" "4" "5"
  ..$ : chr [1:2] "hierarchical" "kmeans"
```

Let's visualize these validation scores to pick the optimal value of k. We're going to use the results from hierarchical and kmeans clustering, and only consider the validation metrics `wb.ratio`, `avg.silwidth`, and `dunn` (which are generally the most useful, in our testing). The validation metrics are standardized and averaged into a single score, such that a higher plotted value is better. Note that for some metrics, like `wb.ratio`, a lower value is naturally better; this is handled internally by pre-multiplying the standardized values by -1 before aggregating the metrics into a single score.

The `plotRange3D` function produces a plot of the aggregated validation score for each value of k. This normally produces a 3D plot using the package "rgl". Note that in this example we don't utilize the third dimension, so we just get the 2D plot. See section `Identifying` k `- additional tools` for an example of how the 3D plot is used. The optimal value of k corresponds to the peak, in this case at $k = 3$.

```
> result.list <- list(list(result), ncol(data))
> goodAlgs <- c('hierarchical', 'kmeans')
> goodMeasures <- c('wb.ratio', 'avg.silwidth', 'dunn')
> values <- plotRange3D(result.list, ks, goodAlgs, goodMeasures)
```



Refer to the documentation for additional options for `COMMUNAL` and more usage examples.

Extracting Core Clusters

After looking at these results, we are satisfied that $k = 3$ is optimal. The next step is to extract the cluster assignments for $k = 3$ with `getClustering`. It returns a data frame whose rows are the samples and columns are the clustering algorithm names. Each entry is the cluster assignment of a sample from the respective algorithm. This is the input used in identifying 'core' clusters.

```
> clusters <- result$getClustering(k=3)
> table(clusters)
```

```
            kmeans
hierarchical  1  2  3
           1 20  0  0
           2  1 19  0
           3  0  0 20
```

From the table, you can see that sometimes kmeans and hierarchical clustering disagree, but in general the clusters fall into three groups of about equal size (just as expected). Now we combine the results to get final cluster assignments. This shows that the algorithms agree on all but 1 of the assignments (which ends up in cluster 0)

```
> # find 'core' clusters
> mat.key <- clusterKeys(clusters, k=3)
> examineCounts(mat.key)
```

```
     percent.agreement sample.counts percent.remaining.if.removed
[1,]                50             1                         0.98
[2,]               100            59                         0.00
```

```
> core <- returnCore(mat.key, agreement.thresh=50) # find 'core' clusters

A total of 1 samples were rejected as not robustly clustered.
98.3 % samples remain.

> table(core) # the 'core' clusters

core
 0  1  2  3
 1 20 19 20

> head(core) # the cluster assignments

Sample1 Sample2 Sample3 Sample4 Sample5 Sample6
    "1"     "1"     "1"     "1"     "1"     "1"
```

Now let's consider a more involved example of how `clusterKeys` and `returnCore` are useful. Consider the following cluster assignments. Overall the algorithms agree that there are three clusters, but differ in how they label the clusters. They disagree about the cluster of the last point.

```
> k <- 3
> clusters <- data.frame(
+   alg1=as.integer(c(1,1,1,1,1,2,2,2,2,2,3,3,3,3,1)),
+   alg2=as.integer(c(1,1,1,1,1,3,3,3,3,3,2,2,2,2,1)),
+   alg3=as.integer(c(3,3,3,3,3,1,1,1,1,1,2,2,2,2,2))
+ )
```

`clusterKeys` reindexes the labels for each algorithm to make the agreement more apparent.

```
> mat.key <- clusterKeys(clusters, k)
> mat.key # cluster indices are relabeled

      alg1 alg2 alg3
 [1,]    1    1    1
 [2,]    1    1    1
 [3,]    1    1    1
 [4,]    1    1    1
 [5,]    1    1    1
 [6,]    2    2    2
 [7,]    2    2    2
 [8,]    2    2    2
 [9,]    2    2    2
[10,]    2    2    2
[11,]    3    3    3
[12,]    3    3    3
[13,]    3    3    3
[14,]    3    3    3
[15,]    1    1    3
```

The next step is to synthesize these into "core" clusters. The clusters are assigned by majority vote. If not enough algorithms agree, based on a user-defined threshold, the cluster is left undetermined. `examineCounts` shows how many samples would be undetermined at various threshold levels.

```
> examineCounts(mat.key)

     percent.agreement sample.counts percent.remaining.if.removed
[1,]          66.66667             1                         0.93
[2,]         100.00000            14                         0.00
```

Now we use a threshold to retrieve the "core" clusters. The default threshold is 50%, meaning that more than 50% of the algorithms must agree. In this case, if we use the 50% threshold, then all points are assigned to some cluster.

```
> core <- returnCore(mat.key, agreement.thresh=50) # find 'core' clusters

A total of 0 samples were rejected as not robustly clustered.
100 % samples remain.

> table(core) # the 'core' clusters

core
1 2 3
6 5 4
```

However, if we require all algorithms to agree, then one point is undetermined (hence labeled as cluster 0).

```
> core <- returnCore(mat.key, agreement.thresh=99)

A total of 1 samples were rejected as not robustly clustered.
93.3 % samples remain.

> table(core) # 0 is undetermined

core
0 1 2 3
1 5 5 4
```

Identifying k - additional tools

The third dimension of the validation measure plot is used to show the results of multiple runs of `COMMUNAL` on different subsets of the data. Often there are extraneous dimensions which make it harder to cluster the data. One solution is to only use the dimensions in which the data points have the highest variance when clustering, since these are the dimensions in which the data are most separated.

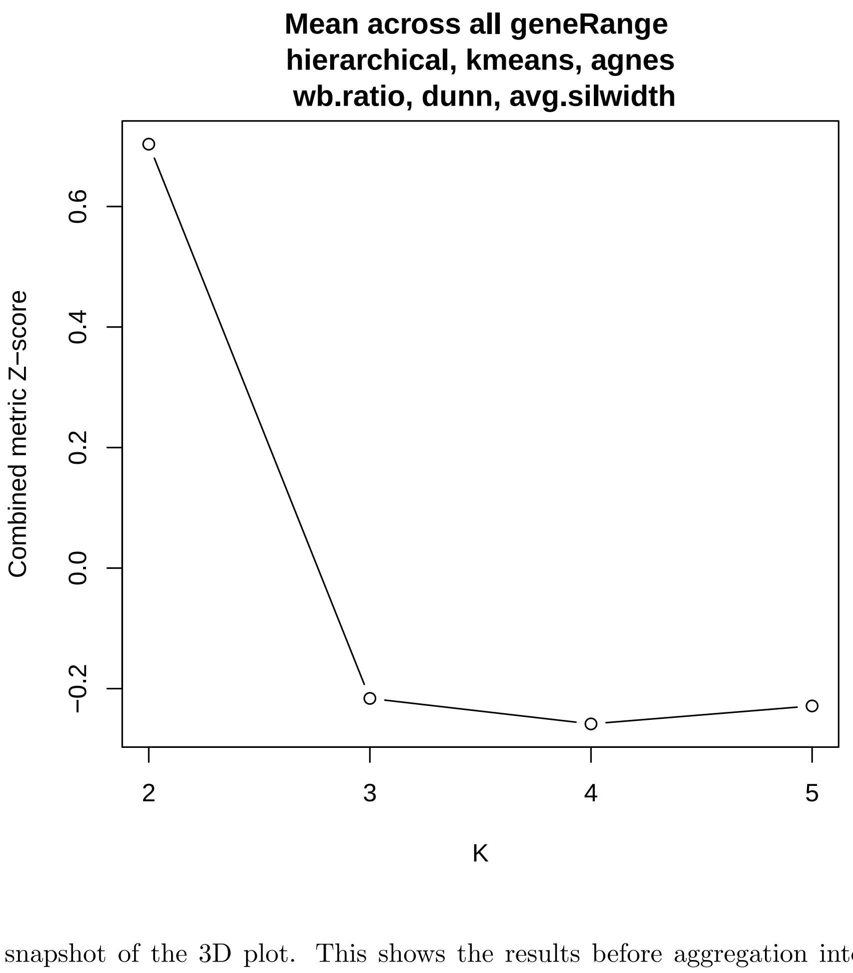
So for each run, we subset the data and cluster using just the x dimensions with the highest variance. Then the validation scores from all runs are plotted together on a 3D plot. Additionally, the scores from all the runs are aggregated to get a score for each k, and these are shown in the 2D plot. Ideally the algorithms will work the regardless of the number of dimensions and we'll identify a consistently optimal number of clusters. If not, we might get different results, in which case we'd want to check the cluster assignments and raw validation metrics for further assessment.

The `clusterRange` function provides a harness to `COMMUNAL` for this purpose. Here I load some breast cancer data for 533 tissues and their expression levels of 100 genes. We'll use 50, 70, 85, and 100 genes (i.e. dimensions) for clustering the tissues.

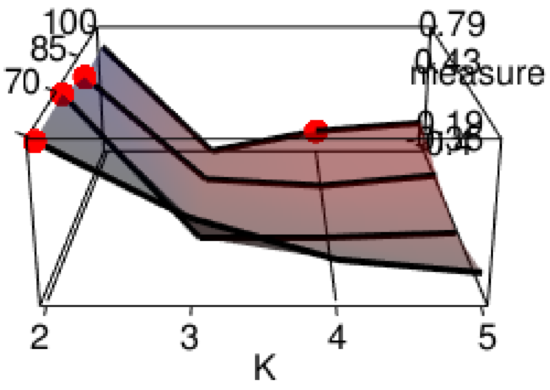
```
> data(BRCA.100)
> algs <- c("hierarchical", "kmeans", "agnes")
> measures <- c('wb.ratio', 'dunn', 'avg.silwidth')
> varRange <- c(50, 70, 85, 100)
> ks <- 2:5
> range.results <- clusterRange(dataMtx=BRCA.100, varRange=varRange,
+                               ks = ks,
+                               clus.methods = algs,
+                               validation = measures)

.         .         .         .
```

Now `range.results` contains the results from running `COMMUNAL` on the subsetted data. We can generate the plots using `plotRange3D`. These indicate that two clusters is best. (As it turns out, if we cluster using more than 100 genes, we find that the data better separate into 3 clusters.)

```
> plot.data <- plotRange3D(range.results, ks, algs, measures, filename='snapshot.png')
```



Here is a snapshot of the 3D plot. This shows the results before aggregation into the 2D plot above. The red dots mark the most concave non-edge point, if the function is concave somewhere, or the absolute maximum otherwise. The z-axis has labels for Tukey's five number summary of all the values. For an interactive 3D example of `plotRange3D`, you may run this code yourself or see the help page for `plotRange3D`.


100
85
70
0.79
0.43
measure
0.19
-0.05
-0.4
2
3
4
5
K