MA5113 Assignment 2: Class/Biomarker Discovery

Ty Medina

Part 1: Data Analysis Procedure

This report is an application of the R package "multiClust" on a breast cancer expression dataset for the National Cancer Control Programme, originally produced by Hatzis et al. 2011 and available through GEO at accession number GSE25066. Results are attached, and be discussed in Part 2.

Loading the Libraries

Necessary libraries, including multiClust, are first loaded.

```
library("Biobase")
library("multiClust")
library("preprocessCore")
library("ctc")
library("gplots")
library("dendextend")
library("graphics")
library("grDevices")
library("amap")
```

Importing the Data

The dataset from GEO accession GSE25066 is read in from a prepared text file. While the R package geoQuery can be used to directly import from GEO, this text file has specifically been edited to remove comments and annotations from its header region, as these can cause incompatibilities with multiClust. The data itself consists of 22,283 microarray probes, measuring gene expression levels of 508 patient tumor biopsies.

```
raw <- as.matrix(read.table("C:/Users/Tyler/Desktop/NOW/GSE25066_noheader.txt",
                            header=T, sep="\t", row.names=1, as.is=1))
```

Control Affymetrix probes are then removed from the data, as they are included only for quality control, not sample measurement.

```
# Removes probes containing "AFF", such as AFFX-CreX-5_at
expr_data <- raw[!grepl("AFF", rownames(raw)),]
```

In addition to expression levels, the patient clinical outcome file is also read in, which contains event time and type for patient Distant Relapse-Free Survival (labeled Disease Free Survival here). This will be used for the generation of Kaplan-Meier survival plots after performing cluster analysis.

```
outcomes <- read.delim2(file="C:/Users/Tyler/Desktop/NOW/GSE25066-DFS-Clinical-Outcome.txt",
                        header=T)
```

Transforming the Data

Before analyzing the expression data, it must first be normalized and transformed. Quantile normalization allows expression levels to be compared between microarray chips, each chip representing a sample. By transforming each probes's value according to an average distribution representative of all samples, global differences and artifacts between chips can be avoided, allowing more accurate comparison.

Note that the normalize.quantiles function from preprocessCore normalizes data values *in situ*, causing the input dataframe to be edited permanently without creating a new object. Though the "copy" argument can be set to true, this will only produce a replicate without row or column names while still editing the original data permanently, and so is not used.

```
normalize.quantiles(expr_data, copy=F)
```

Log transformation then allows gene expression levels to be more realistically and accurately represented. Because recorded expression data is absolute, and not relative, three gene expression measurements of 0.5, 2, and 8 do not easily reflect the fact that the first and third levels *both* show a x4 difference compared to the second measurement. Log transforming the data more readily indicates these relative differences in expression. Additionally, the log transform only considers the magnitude of the difference such that the difference itself is held to be the important measurement, regardless of the direction.

It should be noted that values must be positive and non-zero to perform a logarithm. As such, the data is checked for any negative or zero values. If found, the entire dataset is shifted by the lowest value, plus a small addition to ensure that all values are positive and non-zero.

```
if (min(expr_data) <= 0) {
  expr_shift = expr_data + abs(min(expr_data)) + 0.00001}

expr_log <- t(apply(expr_shift, 1, log2))
```

Choosing the Number of Probes

Clustering is performed using only a subsection of the entire microarray probeset, both to reduce the level of background noise contributed by non-correlated insignificant probes and to reduce computational load. MultiClust provides four methods for inputting the number of probes to select:

- Fixed: The user provides a fixed number of genes to select. This is often set to 1000.
- Percent: The user provides a desired percentage of the total probes to use. This is often set to 1%.
- Polynomial: The multiClust package uses an adaptive polynomial regression method to calculate the optimal number of probes to select based on mean and variance.
- Gaussian Mixture Model (GMM): The multiClust package uses Gaussian Mixture Modelling to determine the number of probes to select based on calculated data substructures. Note that a false discovery rate must be provided, which is often set at the standard 0.05.

The four method calls are shown below, each returning a simple integer numeric object. Note that the adaptive GMM method has a calculation time of approximately 3 hours for this data set. Each method was run using its default or common values to analyze the effect of each on the end result.

```
gene_num_fix <- number_probes(input="GSE25066", data.exp=expr_log,
                              Fixed=1000, Percent=NULL, Poly=NULL, Adaptive=NULL) # Returns 1000
gene_num_per <- number_probes(input="GSE25066", data.exp=expr_log,
                              Fixed=NULL, Percent=1, Poly=NULL, Adaptive=NULL) # Returns 223
gene_num_pol <- number_probes(input="GSE25066", data.exp=expr_log,
                              Fixed=NULL, Percent=NULL, Poly=T, Adaptive=NULL) # Returns 1257
```

```
gene_num_gmm <- number_probes(input="GSE25066", data.exp=expr_log,
                              Fixed=NULL, Percent=NULL, Poly=NULL, Adaptive=T, cutoff=0.01) # Returns 4
```

Choosing the Number of Clusters

In addition to choosing a number of gene probes to use for cluster analysis, the number of clusters to form must also be provided in one of two ways, both of which are used in this report:

- Fixed: The user specifies a fixed number of clusters to form. This value is often informed by *a priori* knowledge of the data.
- Gap Statistic: The goodness of clustering of several numbers of clusters is calculated, culminating in an optimal number of clusters to form based off of bootstrapped PCA rotations.

In previous usages of this dataset, 3 clusters were chosen, despite the fact that there are only 2 obvious clusters from literature. Regardless, it is useful to observe the effect of seemingly extra clusters to observe any substructures that may exist.

The gap statistic measurement returned an optimal cluster number of 8. However, multiClust issued a non-converge error upon completion, suggesting that an optimal number of clusters may not exist, or more likely that upon reaching the multiClust maximum cluster number of 8 without converging on a value, multiClust simply returned the last value. This may indicate that 8 is not a particularly useful number of clusters at all. It should also be noted that the gap statistic calculation time for this dataset is extremely long, completing bootstrapping after approximately 13 hours.

```
cluster_num_fix <- number_clusters(data.exp=expr_log, Fixed=3, gap_statistic=NULL) # Returns 3
cluster_num_stat <- number_clusters(data.exp=expr_log, Fixed=NULL, gap_statistic=T) # Returns 8
```

Ranking the Probes

The multiClust ranking analysis ranks gene probes according to some measure of variance. Four methods are included:

- SD Rank: Each probe is ranked by its standard deviation across all samples.
- CV Rank: Each probe is ranked by its coefficient of variation (the ratio of its standard deviation to mean) across all samples.
- CV Guided: Each probes is ranked by a novel method that guides the CV_Rank method along the coefficient of variation of the entire dataset.
- Polynomial: As in the probe number selection, second-order polynomial regression is used to rank probes based on their standard deviation and mean.

Each ranking method then produces a list of probes with high variance, as measured according to the method. The length of each list is dictated by the probe number selection (eg. fixed, percent, etc.). Note that the polynomial ranking will default to the polynomial gene number selection, and will not accept other values higher than this value. For this reason, using the GMM number selection method with polynomial ranking is not possible ($4458 > 1257$).

An example ranking call is shown below and takes the following arguments:

- input: A string with an arbitrary user-defined prefix name for files generated.
- data.exp: The expression array to rank.
- probe_number: The number of probes to choose after ranking. This can be supplied by the output of the probe number function.

- probe_num_selection: An arbitrary string describing the number selection method. Used solely for the purpose of file naming.
- method: The desired gene ranking method (eg. SD_Rank)

```
probe_ranking(input = "Example",
              data.exp = my_data,
              probe_number = num_object,
              probe_num_selection = "Fixed",
              method = "SD_Rank")
```

Selected probe distributions are shown at the end of this report. The ranking function also produces files containing the ranked probes and heatmap files viewable with Treeview. Representative excerpts from the heatmaps for the three selected rankings are attached after this report.

Clustering the Selected Probes

Machine learning algorithms can be used to identify patterns with the probe expression levels. Several methods are available for these purposes, including the two methods available in multiClust:

- K-Means Clustering: Probes are assigned to clusters such that the variance within each cluster is minimized.
- Hierarchical Clustering: Probes are split or joined into clusters that minimize a distance metric progressively in a tree-like fashion, until the desired number of clusters is achieved.

The hierarchical clustering used by multiClust also has the option to specify distance, linkage type, and gene distance type for calculations. Based on performance in similar studies, the following values were used for all hierarchical clustering, and are the defaults in multiClust:

- Linkage: Ward.D2
- Distance: Euclidean
- Gene Distance: Correlation

All other arguments take strings specifying extra information that used only for file naming. An example hierarchical clustering call is shown below:

```
cluster_analysis(sel.exp = this_rank,
                 cluster_type = "HClust",
                 distance = "euclidean",
                 linkage_type = "ward.D2",
                 gene_distance = "correlation",
                 num_clusters = my_gap_statistic_value,
                 data_name = "My_Dataset_Name",
                 probe_rank = "I_Used_SD_Rank",
                 probe_num_selection = "I_Used_GMM_Ranking",
                 cluster_num_selection = "I_Used_Gap_Statistic_Cluster_Number")
```

Kaplan-Meier Survival Curves

After clustering the data, survival curves are created to show survival per cluster. An example survival curve call is shown below, and survival curves for each clustering are attached at the end of this report. Note that all fields besides the clustered object are solely for naming files and the graph title itself.

```
surv_analysis(samp_cluster = my_hierarchical_cluster_analysis,
              clinical = "GSE25066-DFS-Clinical-Outcome.txt",
              survival_type = "DRFS",
              data_name = "My_Dataset_Name",
              cluster_type = "I_Used_Hierarchical_Clusters",
              distance = "I_Used_Euclidean",
              linkage_type = "I_Used_Ward_Linkage",
              probe_rank = "This_Was_Made_Using_SD_Rank",
              probe_num_selection = "GMM_Was_Used",
              cluster_num_selection = "I_Used_the_Gap_Statistic_Method")
```

Function Definition and Outcomes

To compare each of the possible methods of probe number selection, ranking, clustering type, and clustering number, a function was defined below that was then passed into a basic "apply" function in R. In this way, all (64 combinations - 4 GMM/poly = 60) possible combinations were performed.

Ultimately the two probe number selection methods and two ranking methods that yielded the overall lowest average survival curve P-values for both KMeans and Hierarchical clustering were found to be Fixed and GMM Adaptive probe number selection, and CV Guided and Polynomial probe ranking.

KMeans, 3 clusters

- CV Guided Rank, Fixed Gene Number: 1.7E-6
- CV Guided Rank, GMM Gene Number: 4.4E-7
- Polynomial Rank, Fixed Gene Number: 4.4E-8

Hierarchical, 3 clusters

- CV Guided Rank, Fixed Gene Number: 2.2E-3
- CV Guided Rank, GMM Gene Number: 2.2E-5
- Polynomial Rank, Fixed Gene Number: 5.9E-4

KMean, 8 clusters

- CV Guided Rank, Fixed Gene Number: 7.3E-7
- CV Guided Rank, GMM Gene Number: 8.8E-6
- Polynomial Rank, Fixed Gene Number: 1.2E-6

Hierarchical, 8 clusters

- CV Guided Rank, Fixed Gene Number: 4.4E-4
- CV Guided Rank, GMM Gene Number: 2.7E-4
- Polynomial Rank, Fixed Gene Number: 1.0E-8

The function definition is shown below:

```
# Function definition.
rank_function <- function(X, selection_method, num_object, tag = "") {

  # A desired edit to the file names can be added through the "tag" argument.
  this_name <- paste(tag, "GSE25066", sep = "")
```

```
# Main ranking function.
this_rank <- probe_ranking(input = this_name,
                           data.exp = expr_log,
                           probe_number = num_object,
                           probe_num_selection = selection_method,
                           method = X)

# The following are the hierarchical and kmeans clustering, using fixed cluster number
#and statistical cluster number, and generating survival curves for each.
clustK_fix <- cluster_analysis(sel.exp = this_rank,
                               cluster_type = "Kmeans",
                               distance = NULL,
                               linkage_type = NULL,
                               gene_distance = NULL,
                               num_clusters = cluster_num_fix,
                               data_name = this_name,
                               probe_rank = X,
                               probe_num_selection = selection_method,
                               cluster_num_selection = "Fixed_Clust_Num")
survK_fix <- surv_analysis(samp_cluster = clustK_fix,
                           clinical = "GSE25066-DFS-Clinical-Outcome.txt",
                           survival_type = "DFS",
                           data_name = "GSE25066",
                           cluster_type = "K-Means",
                           distance = NULL,
                           linkage_type = NULL,
                           probe_rank = X,
                           probe_num_selection = selection_method,
                           cluster_num_selection = "Fixed_Clust_Num")

clustK_stat <-cluster_analysis(sel.exp=this_rank,
                               cluster_type = "Kmeans",
                               distance = NULL,
                               linkage_type = NULL,
                               gene_distance = NULL,
                               num_clusters = cluster_num_stat,
                               data_name = this_name,
                               probe_rank = X,
                               probe_num_selection = selection_method,
                               cluster_num_selection = "Statistic_Clust_Num")
survK_stat <- surv_analysis(samp_cluster = clustK_stat,
                           clinical = "GSE25066-DFS-Clinical-Outcome.txt",
                           survival_type = "DFS",
                           data_name = "GSE25066",
                           cluster_type = "K-Means",
                           distance = NULL,
                           linkage_type = NULL,
                           probe_rank = X,
                           probe_num_selection = selection_method,
                           cluster_num_selection = "Statistic_Clust_Num")

clustH_fix <- cluster_analysis(sel.exp = this_rank,
                               cluster_type = "HClust",
```

```
                                  distance = "euclidean",
                                  linkage_type = "ward.D2",
                                  gene_distance = "correlation",
                                  num_clusters = cluster_num_fix,
                                  data_name = this_name,
                                  probe_rank = X,
                                  probe_num_selection = selection_method,
                                  cluster_num_selection = "Fixed_Clust_Num")
  survH_fix <- surv_analysis(samp_cluster = clustH_fix,
                             clinical = "GSE25066-DFS-Clinical-Outcome.txt",
                             survival_type = "DFS",
                             data_name = "GSE25066",
                             cluster_type = "HClust",
                             distance = "euclidean",
                             linkage_type = "ward.D2",
                             probe_rank = X,
                             probe_num_selection = selection_method,
                             cluster_num_selection = "Fixed_Clust_Num")

  clustH_stat <- cluster_analysis(sel.exp = this_rank,
                                  cluster_type = "HClust",
                                  distance = "euclidean",
                                  linkage_type = "ward.D2",
                                  gene_distance = "correlation",
                                  num_clusters = cluster_num_stat,
                                  data_name = this_name,
                                  probe_rank = X,
                                  probe_num_selection = selection_method,
                                  cluster_num_selection = "Statistic_Clust_Num")
  survH_stat <- surv_analysis(samp_cluster = clustH_stat,
                             clinical = "GSE25066-DFS-Clinical-Outcome.txt",
                             survival_type = "DFS",
                             data_name = "GSE25066",
                             cluster_type = "HClust",
                             distance = "euclidean",
                             linkage_type = "ward.D2",
                             probe_rank = X,
                             probe_num_selection = selection_method,
                             cluster_num_selection = "Statistic_Clust_Num")
}
```

Applying the Function

The apply loops are set up as follows:

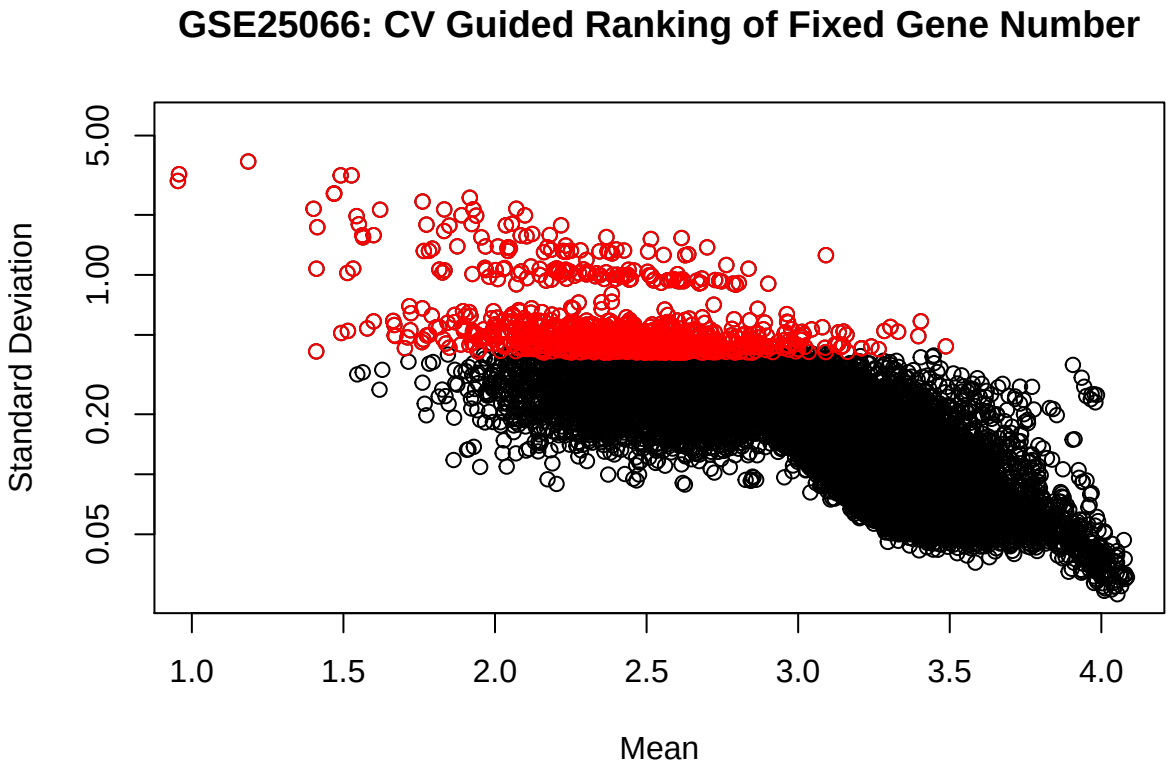
```
# List of the ranking methods to loop through.
rank_methods = c("SD_Rank", "CV_Rank", "CV_Guided", "Poly")

# Important note: "Tags" are added to the file names. The polynomial ranking method does
#not include the probe number selection method in the file name, so all 4 polynomial rankings
#generated would otherwise have the same filename and overwrite each other.
lapply(X=rank_methods, rank_function, selection_method="Fixed", num_object=gene_num_fix, tag="1_")
```

```
lapply(X=rank_methods, rank_function, selection_method="Percent", num_object=gene_num_per, tag="2_")
lapply(X=rank_methods, rank_function, selection_method="Poly", num_object=gene_num_pol, tag="3_")

# The ranking methods list is recreated, omitting the polynomial method since it is
#incompatible with the higher number chosen through GMM.
rank_methods = c("SD_Rank", "CV_Rank", "CV_Guided")
lapply(X=rank_methods, rank_function, selection_method="GMM", num_object=gene_num_gmm, tag="4_")
```

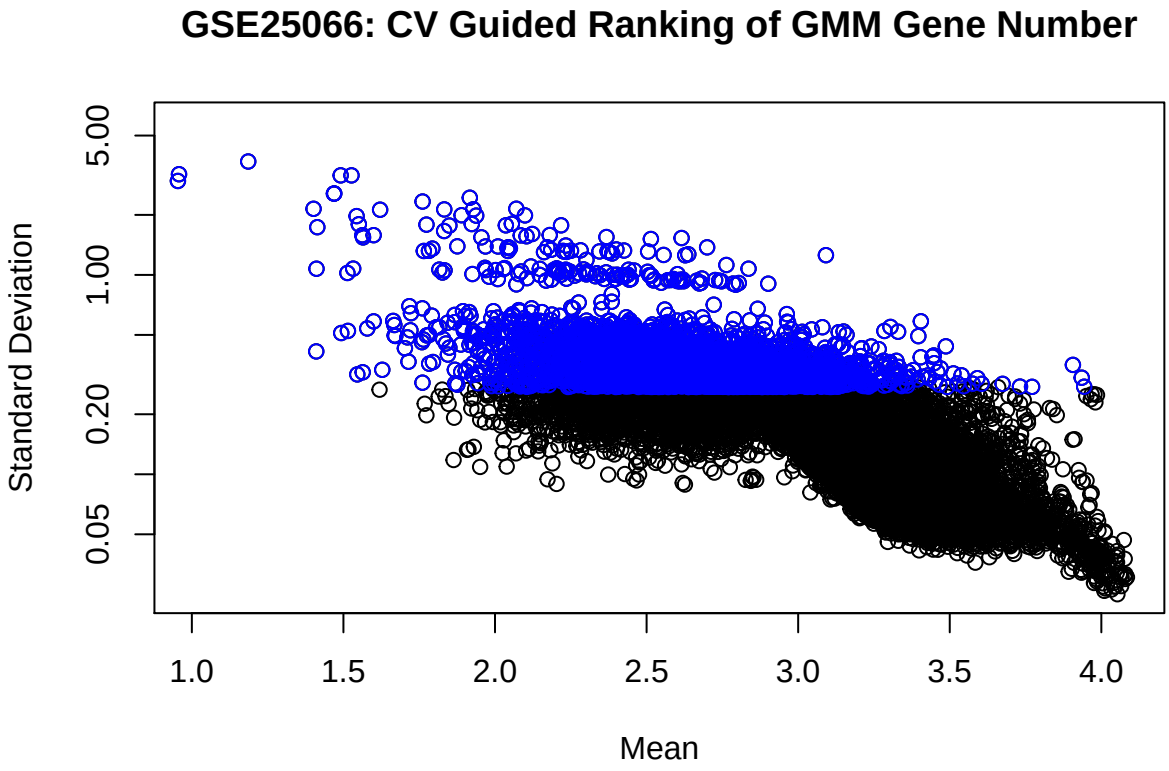
Selected Probe Distributions

The following shows the variance distributions of the probeset. Colored probes indicate selected probes per method.

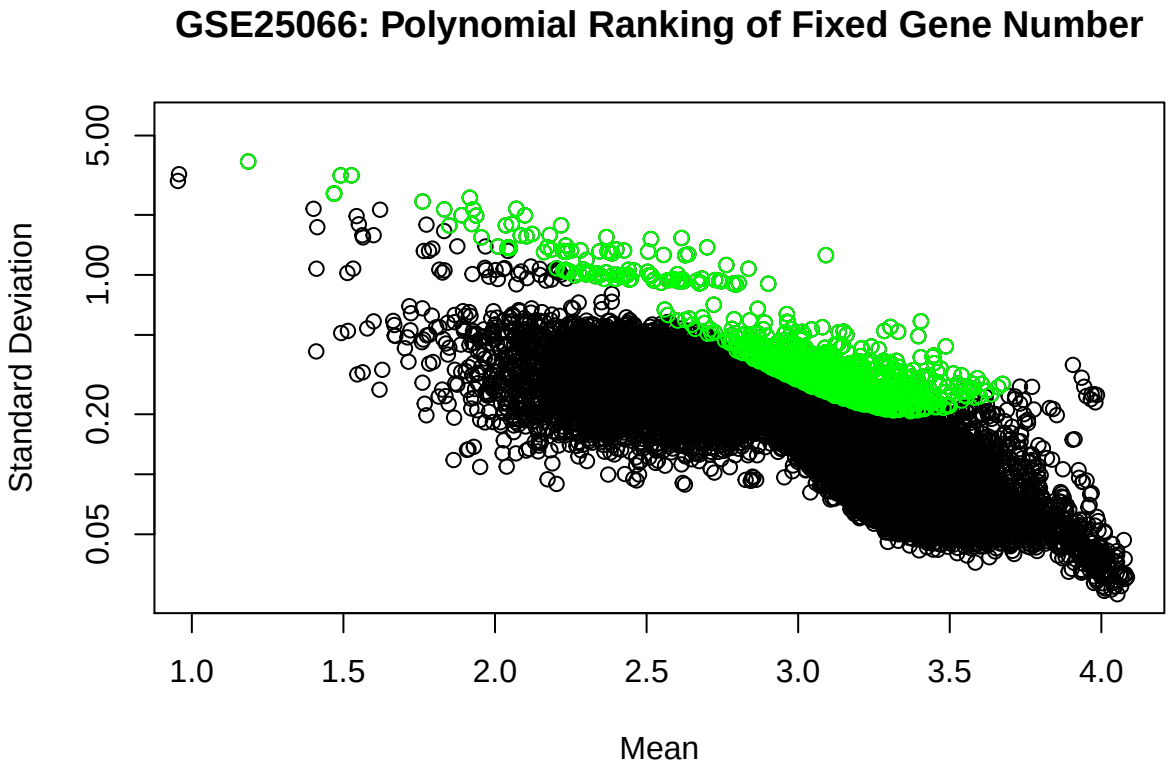
```
means <- apply(expr_log, 1, mean)
sds <- apply(expr_log, 1, sd)
CVG_Fix <- as.matrix(read.table("C:/Users/Tyler/Desktop/NOW/Rankings/1_GSE25066.Fixed.CV_Guided_Selecte
CVG_GMM <- as.matrix(read.table("C:/Users/Tyler/Desktop/NOW/Rankings/4_GSE25066.GMM.CV_Guided_Selected_
Poly_Fix <- as.matrix(read.table("C:/Users/Tyler/Desktop/NOW/Rankings/1_GSE25066.Poly_Selected_Gene_Exp

cvgf <- rownames(CVG_Fix)
cvggmm <- rownames(CVG_GMM)
polyf <- rownames(Poly_Fix)

plot(means, sds,
     col="black", log="y",
     xlim=c(1,max(means)),
     main="GSE25066: CV Guided Ranking of Fixed Gene Number",
     xlab="Mean",
     ylab="Standard Deviation")
points(apply(expr_log[cvgf,], 1, mean),
       apply(expr_log[cvgf,], 1, sd),
       col="red")
```




```
plot(means, sds,
     col="black", log="y",
     xlim=c(1,max(means)),
     main="GSE25066: CV Guided Ranking of GMM Gene Number",
     xlab="Mean",
     ylab="Standard Deviation")
points(apply(expr_log[cvggmm,], 1, mean),
       apply(expr_log[cvggmm,], 1, sd),
       col="blue")
```




```
plot(means, sds,
     col="black", log="y",
     xlim=c(1,max(means)),
     main="GSE25066: Polynomial Ranking of Fixed Gene Number",
     xlab="Mean",
     ylab="Standard Deviation")
points(apply(expr_log[polyf,], 1, mean),
       apply(expr_log[polyf,], 1, sd),
       col="green")
```



Conclusions

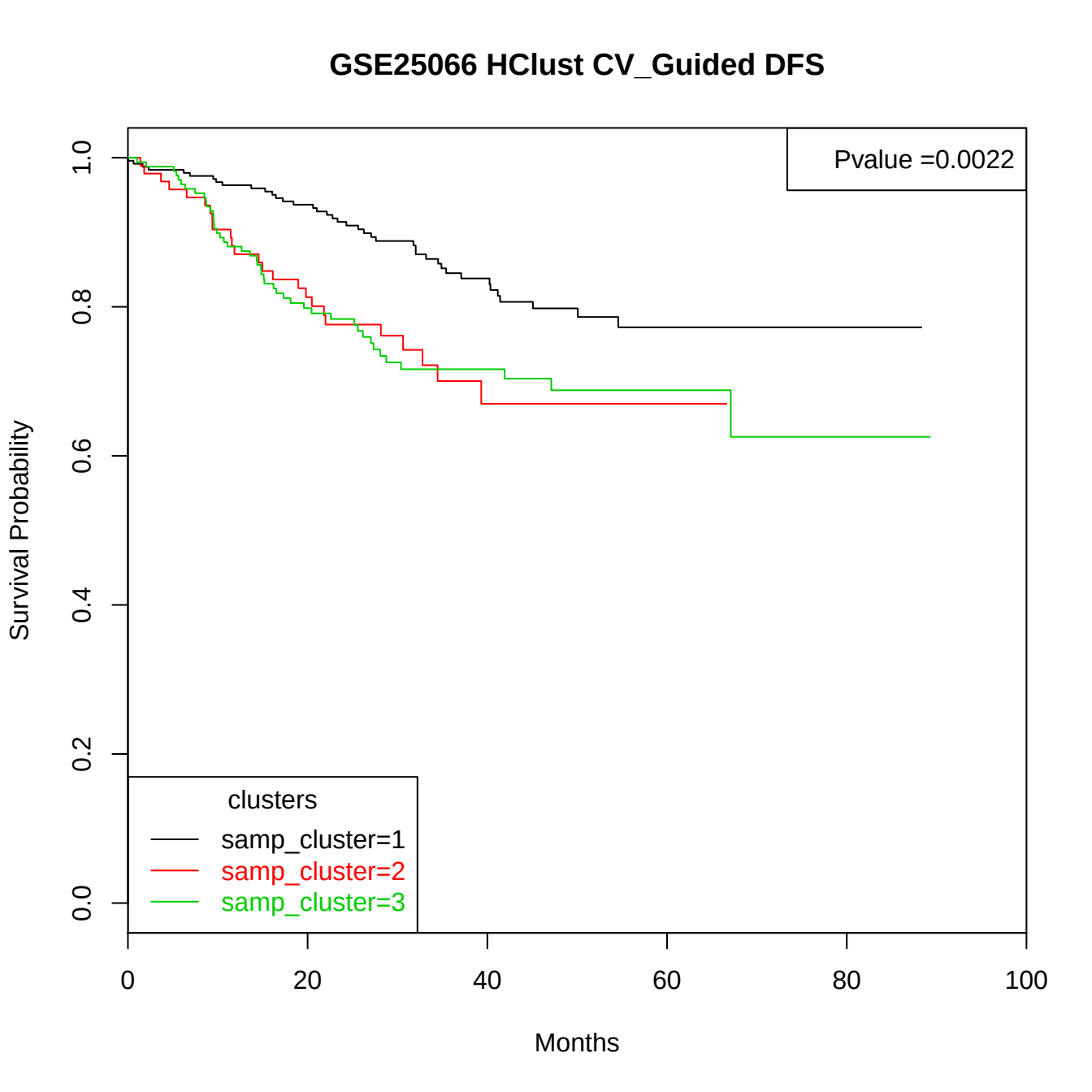
This application of multiClust on a cancer dataset has presented several benefits and setbacks. As a concept, multiClust provides multiple functionalities in one package, making it a convenient one-stop analysis tool. Its novel CV guided ranking also proved to produce generally favorable and improved results. However, in practice the package falls short of its potential. While it has the ability to run multiple methods on the same data, it fails to implement any way to do so automatically, and lacks any comparison tools. Furthermore, it fails as a pipelining tool due to the limited data objects it produces. For example, the gene number selection function requires the user to input the desired method, then returns the number value as a numeric object; every downstream function requires the user to input a string indicating the gene number selection method that was used to produce the data up until that point, culminating in the survival function requiring 10 arguments, only 2 of which are actually used for analysis, and 8 of which are simply for naming. If each function returned an object that also included some kind of annotations containing the parameters used to generate the object (eg. if fixed probe number selection of 1000 probes returned [1000, fixed] instead of just the number 1000), the entire package would be enormously streamlined. Inconsistent file naming schemes and an inconsistent, error-ridden vignette also contribute to multiClust's less-than-desirable performance.

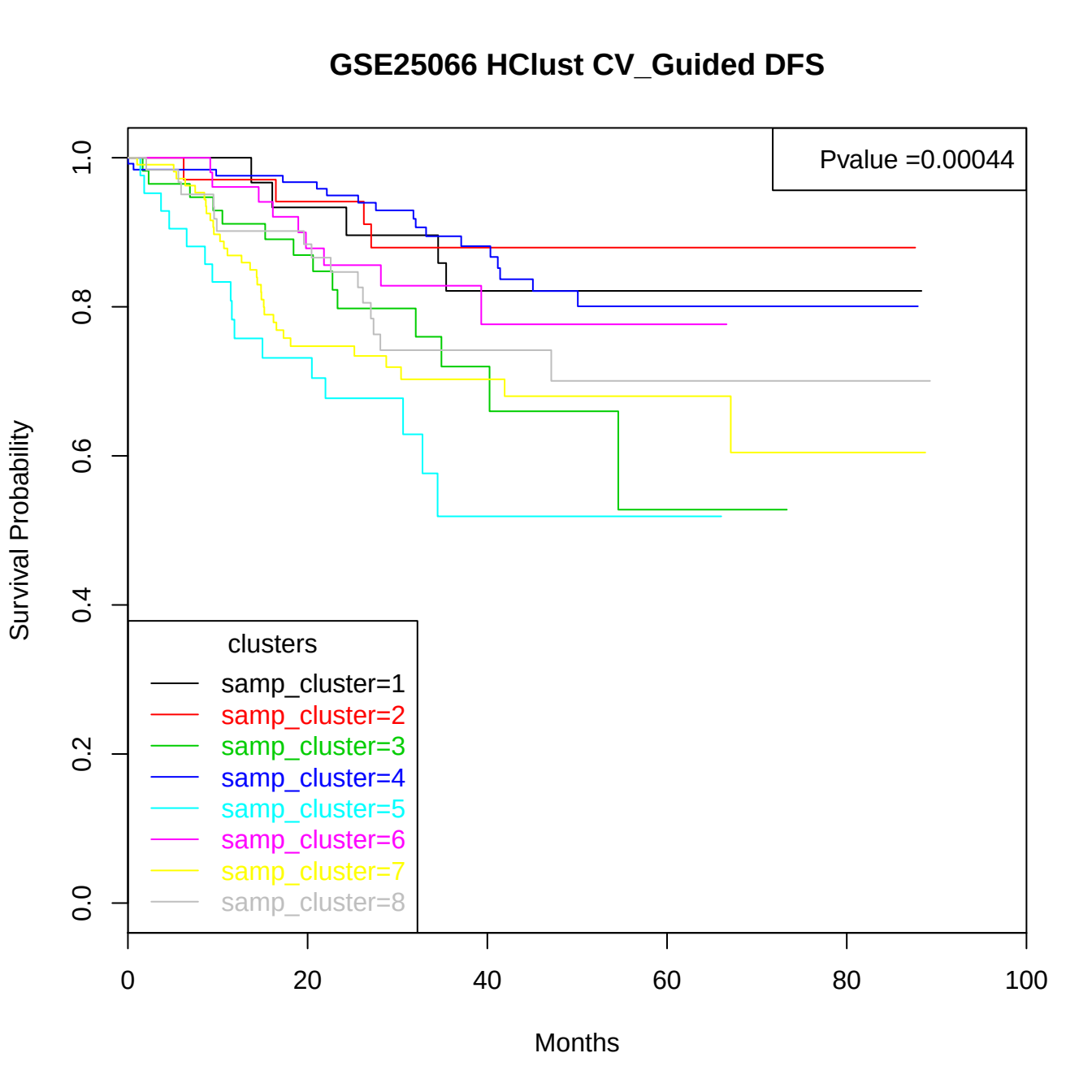
The quality of the data generated also leaves something to be desired. For example, cluster assignments are arbitrary, meaning that cluster names (eg. cluster 1, 2, 3, etc.) are not comparable between clustering methods. For example, all else being equal, comparing hierarchical and Kmeans 3-cluster assignments is not possible, because while the clustering may have been produced the exact same sample clusters, their assigned numbers may be (1, 2, 3) and (2, 1, 3) respectively, resulting in zero *nominally* matching clusters.

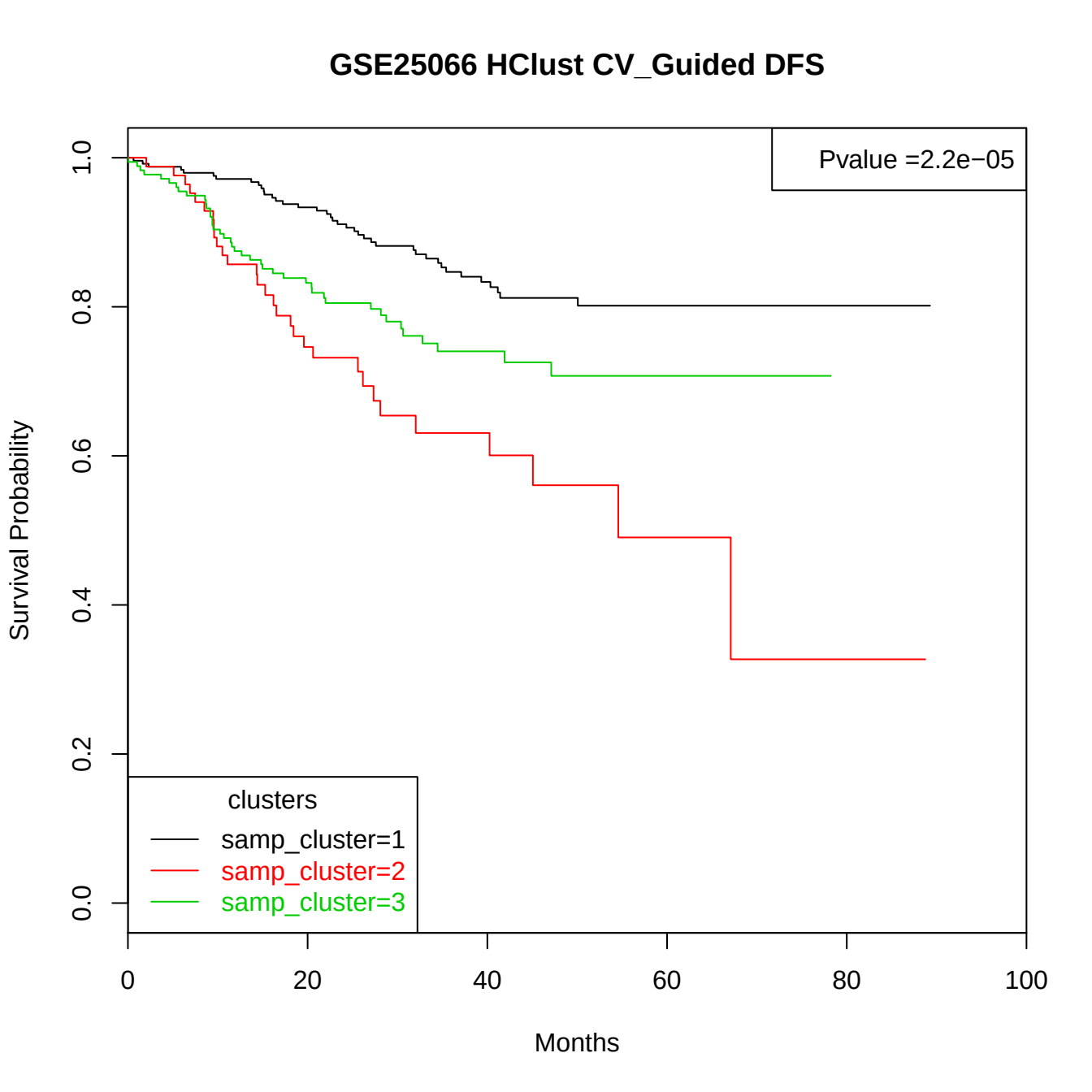
Other issues arose throughout the analysis as well, resulting in more serious complication. The gene lists

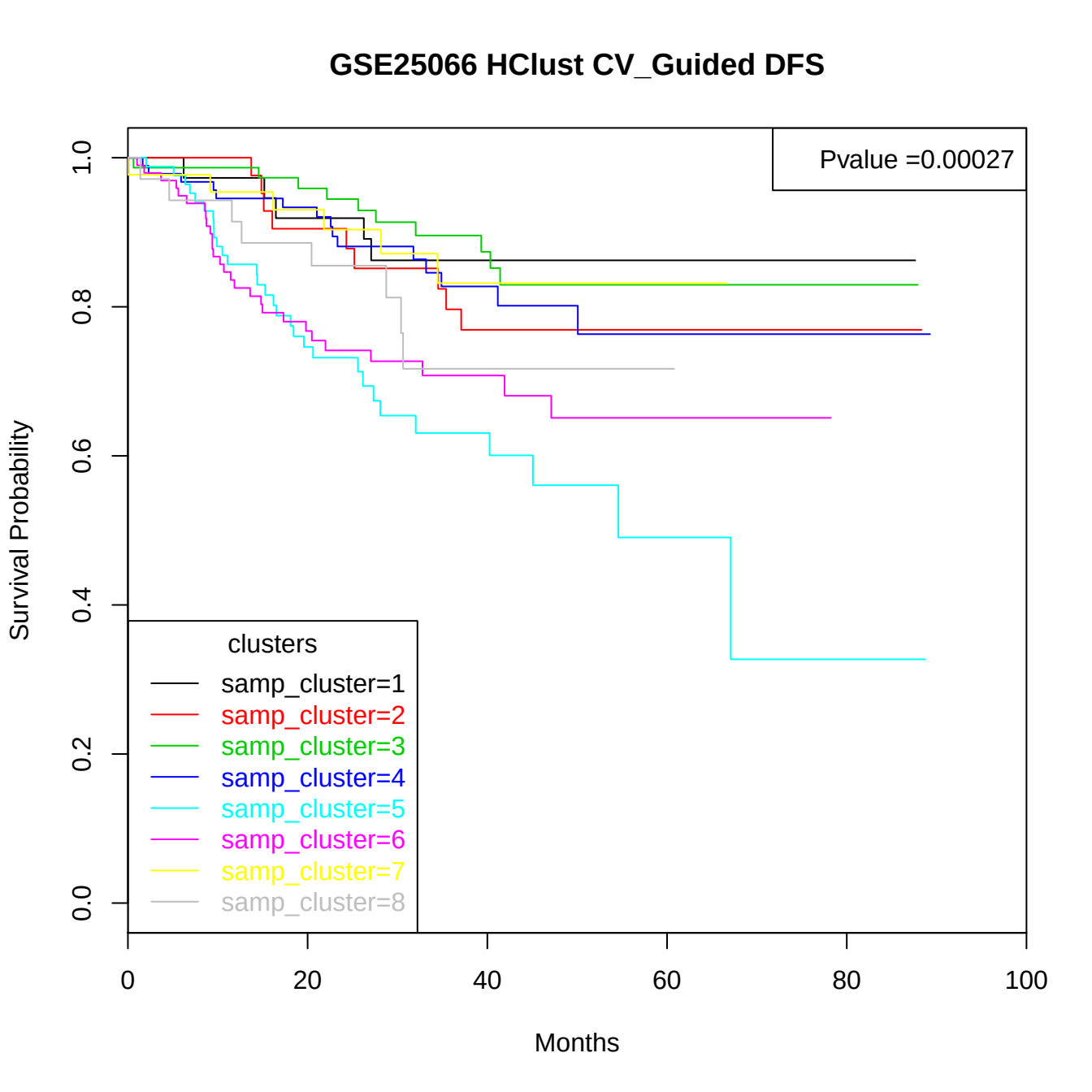
returned by the ranking function are often not actually sorted by whatever ranking criteria was input, and are instead listed in numerical order, leaving the user guessing as to which of the selected probes showed the best ranking criteria. The expression values in these lists also do not match the input expression levels, casting doubt on the usability of these lists for any computational purposes.

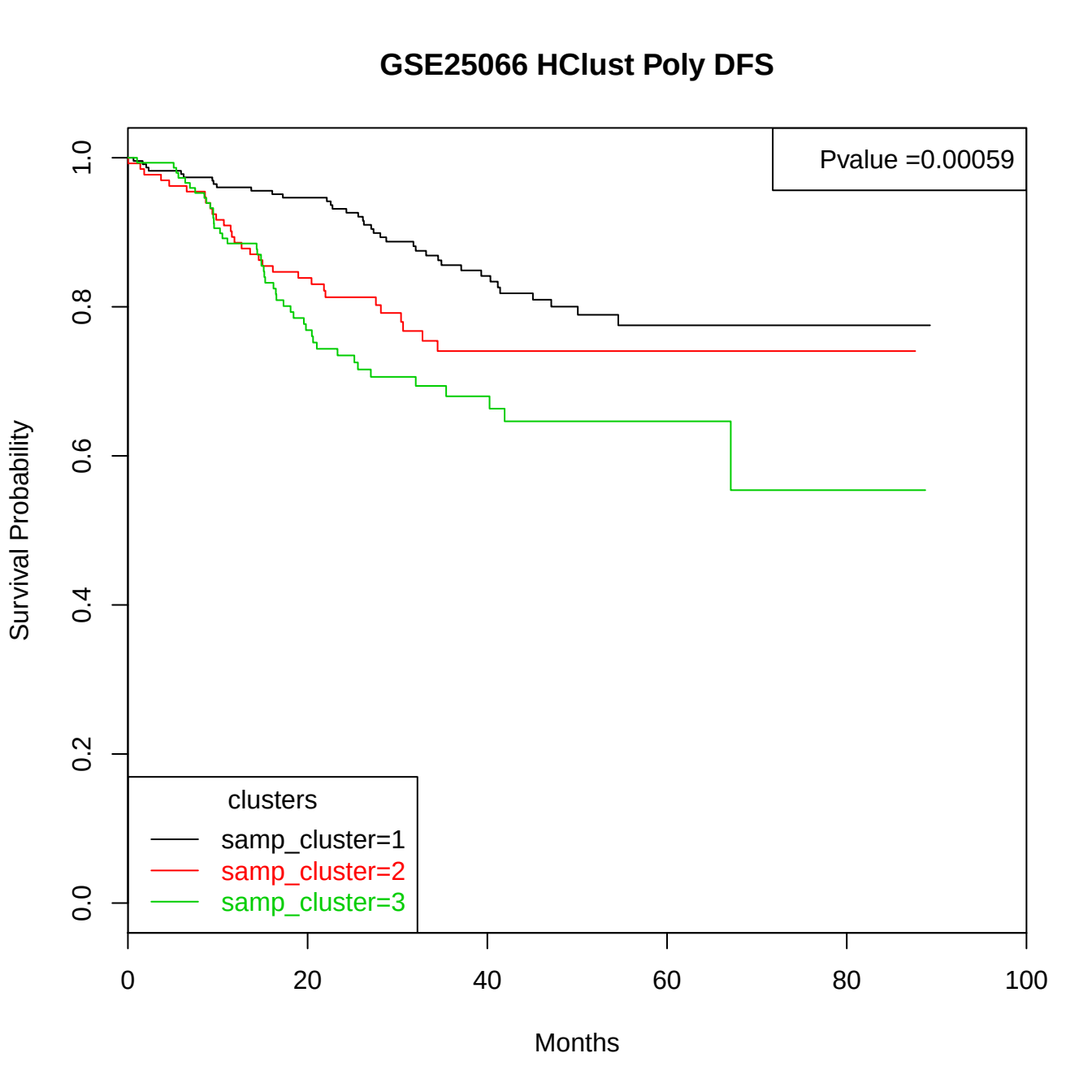
Overall, multiClust shows potential, but currently causes more problems than it solves.

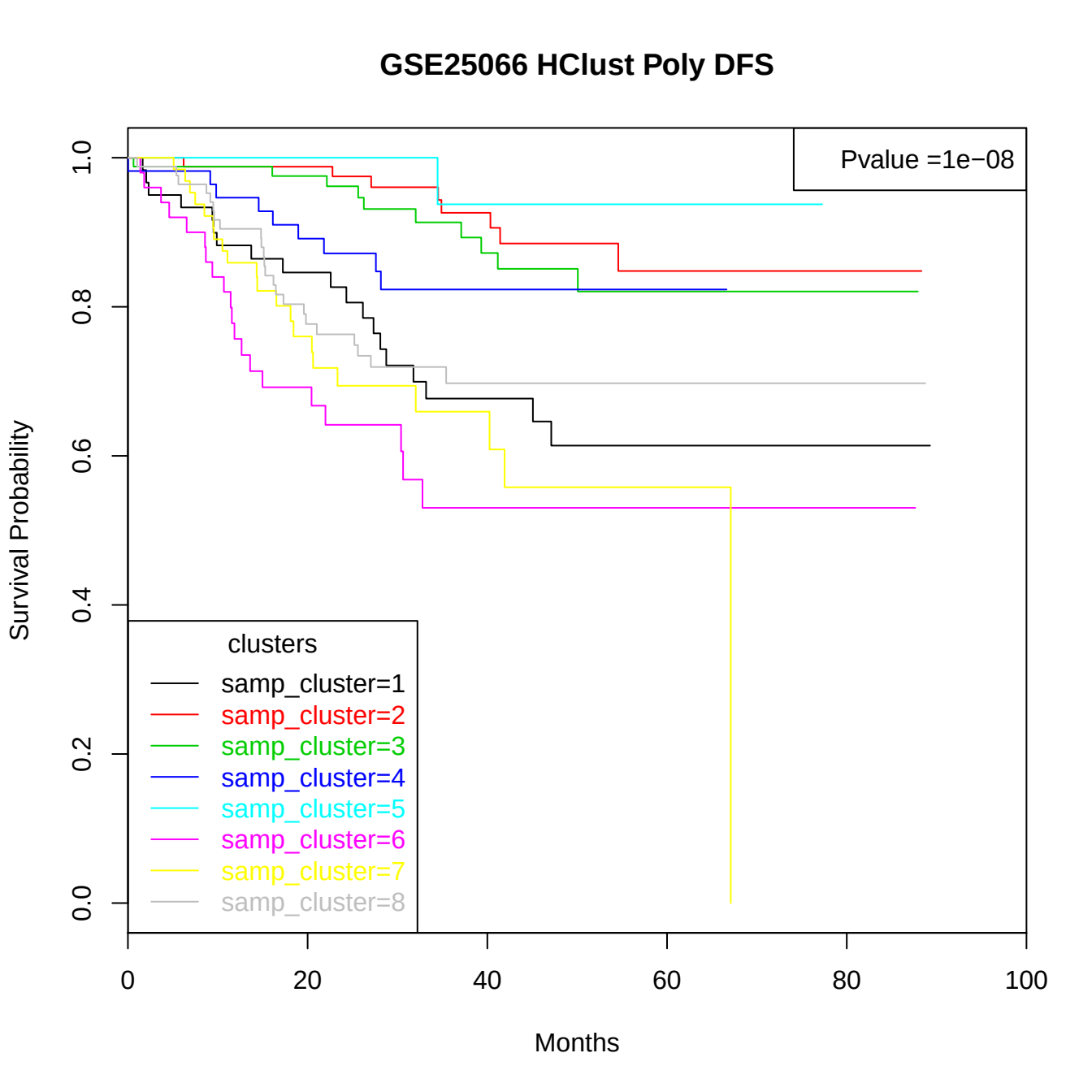

GSE25066 HClust CV_Guided DFS
Pvalue =0.0022
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
0
20
40
60
80
100
Months

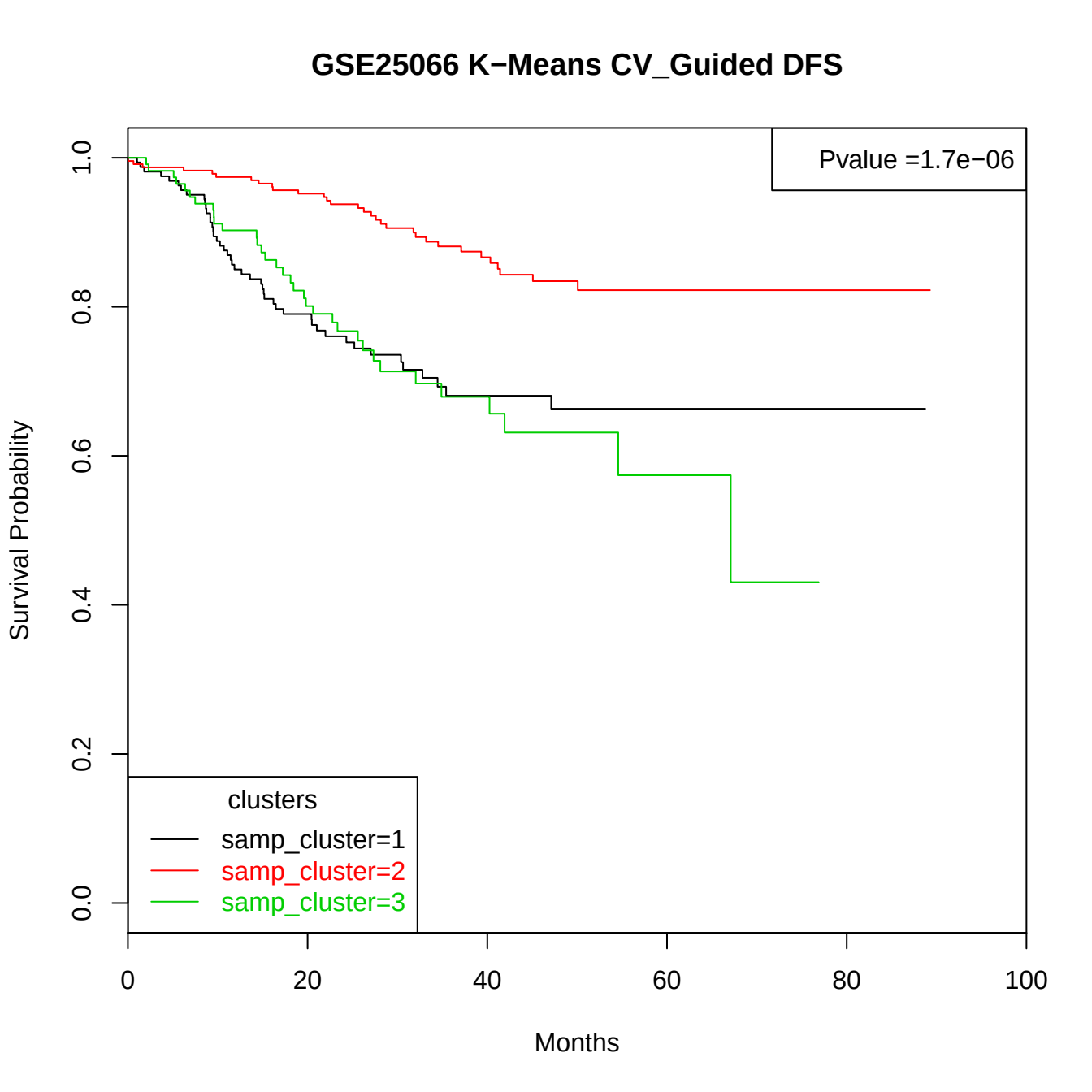

GSE25066 HClust CV_Guided DFS
Pvalue =0.00044
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
samp_cluster=4
samp_cluster=5
samp_cluster=6
samp_cluster=7
samp_cluster=8
0
20
40
60
80
100
Months

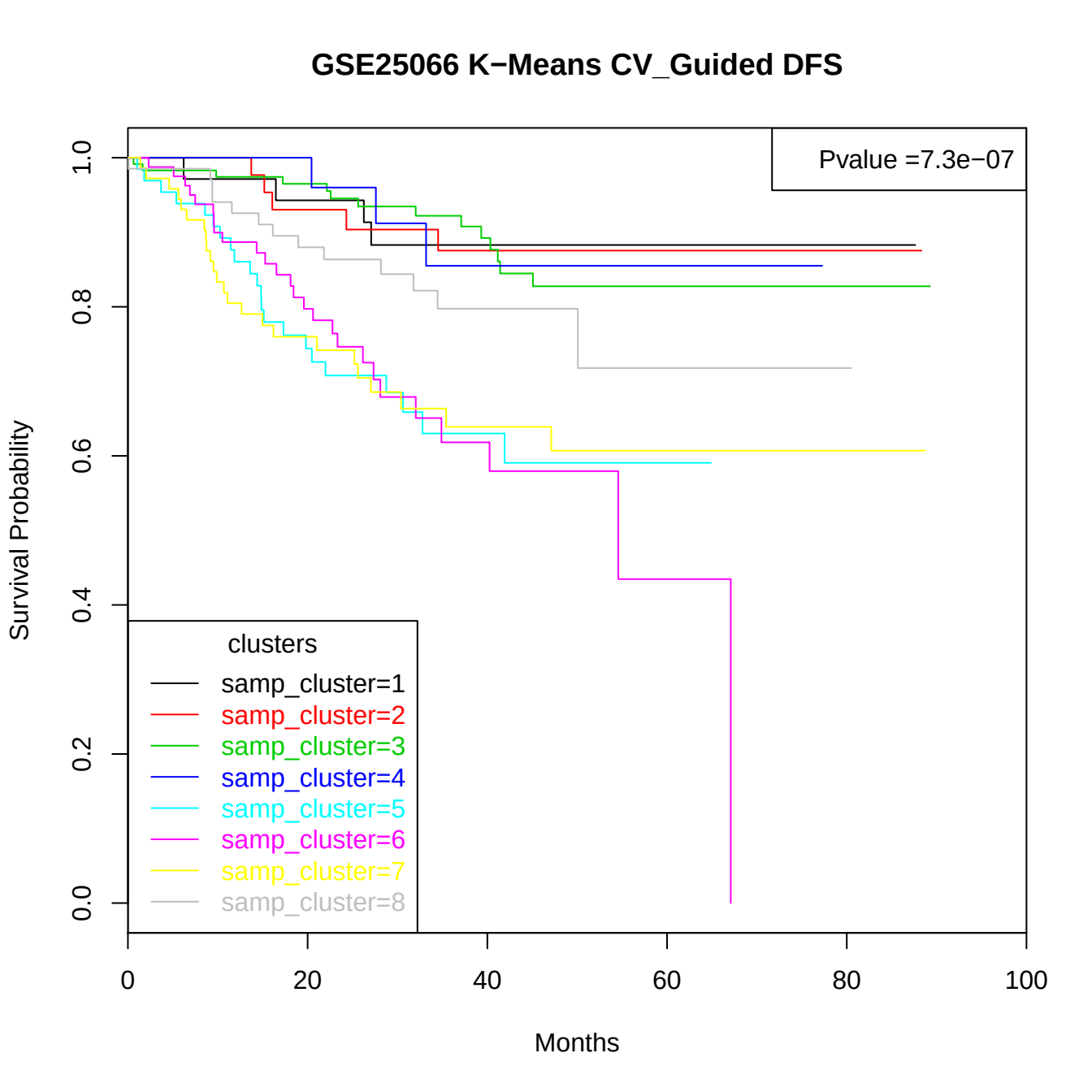

GSE25066 HClust CV_Guided DFS
Pvalue =2.2e−05
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
0
20
40
60
80
100
Months

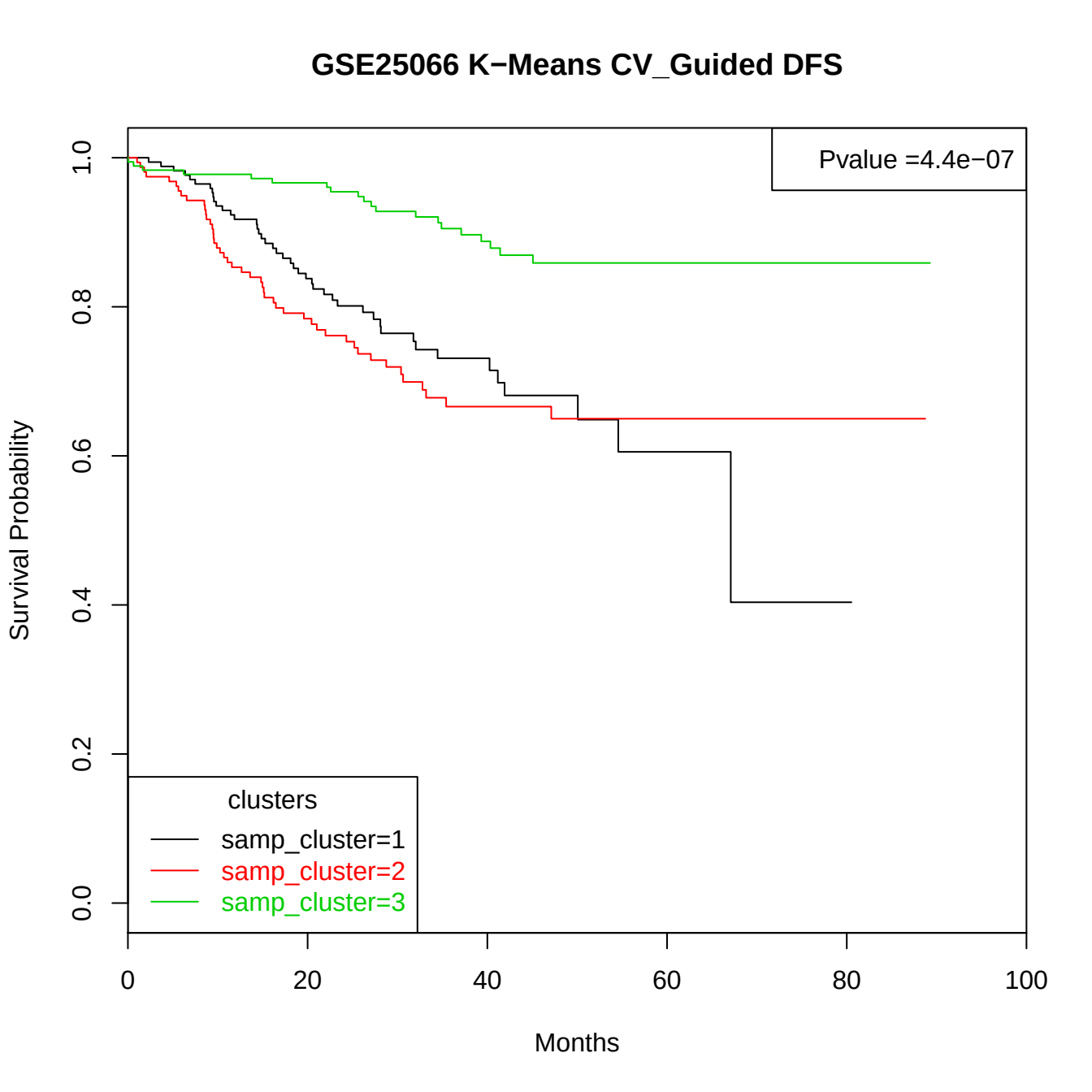

GSE25066 HClust CV_Guided DFS
Pvalue =0.00027
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
samp_cluster=4
samp_cluster=5
samp_cluster=6
samp_cluster=7
samp_cluster=8
0
20
40
60
80
100
Months

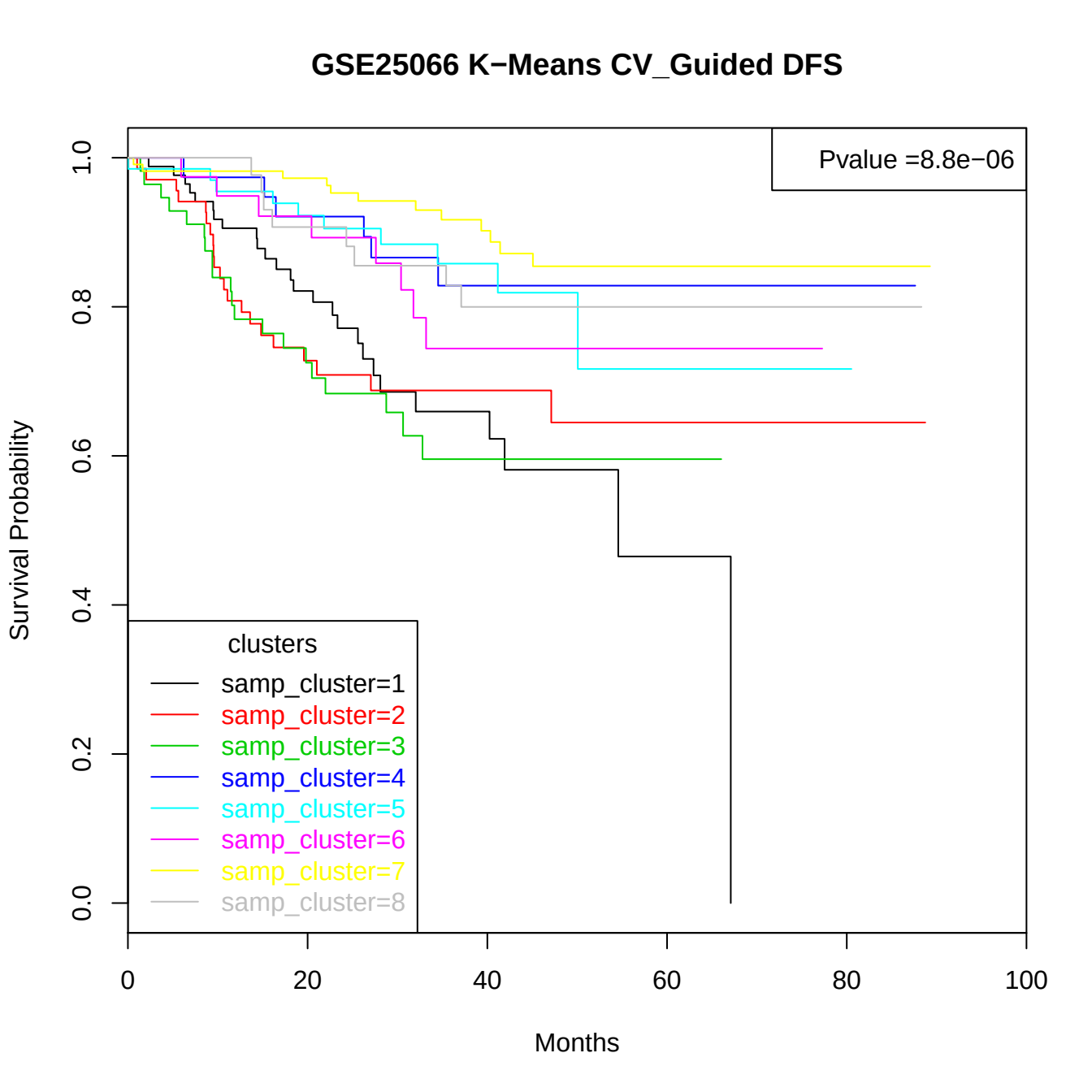

GSE25066 HClust Poly DFS
Pvalue =0.00059
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
0
20
40
60
80
100
Months

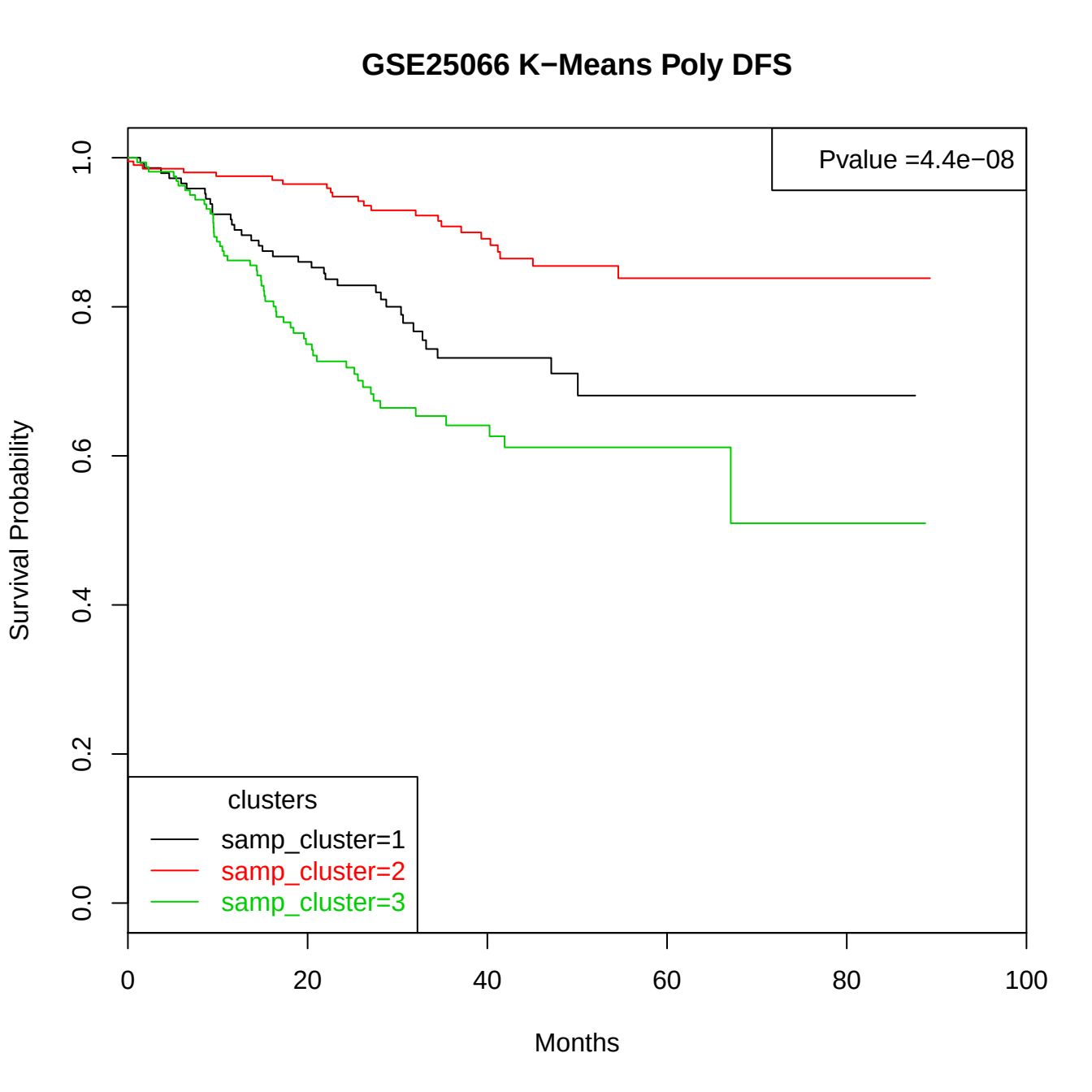

GSE25066 HClust Poly DFS
Pvalue =1e−08
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
samp_cluster=4
samp_cluster=5
samp_cluster=6
samp_cluster=7
samp_cluster=8
0
20
40
60
80
100
Months

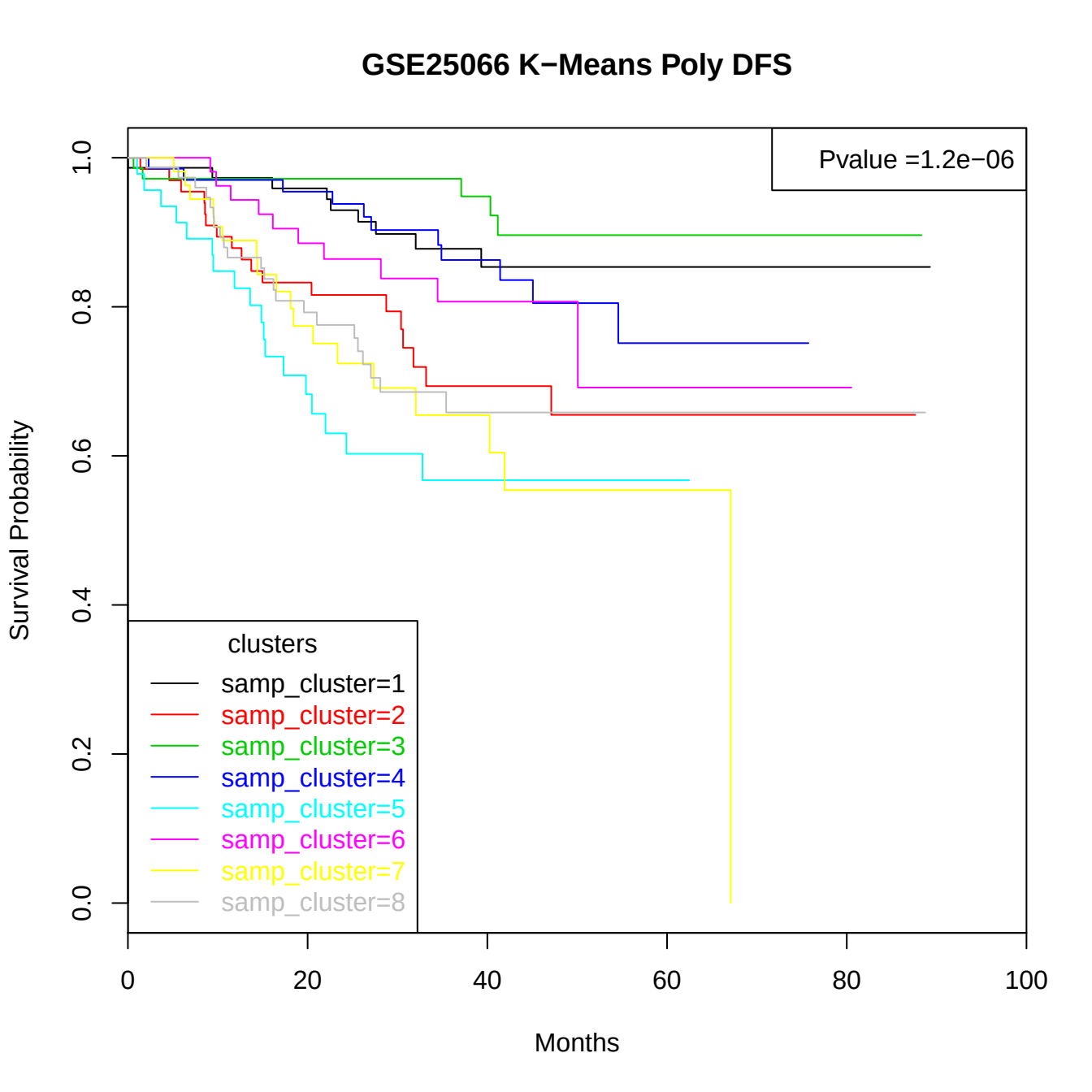

GSE25066 K−Means CV_Guided DFS
Pvalue =1.7e−06
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
0
20
40
60
80
100
Months

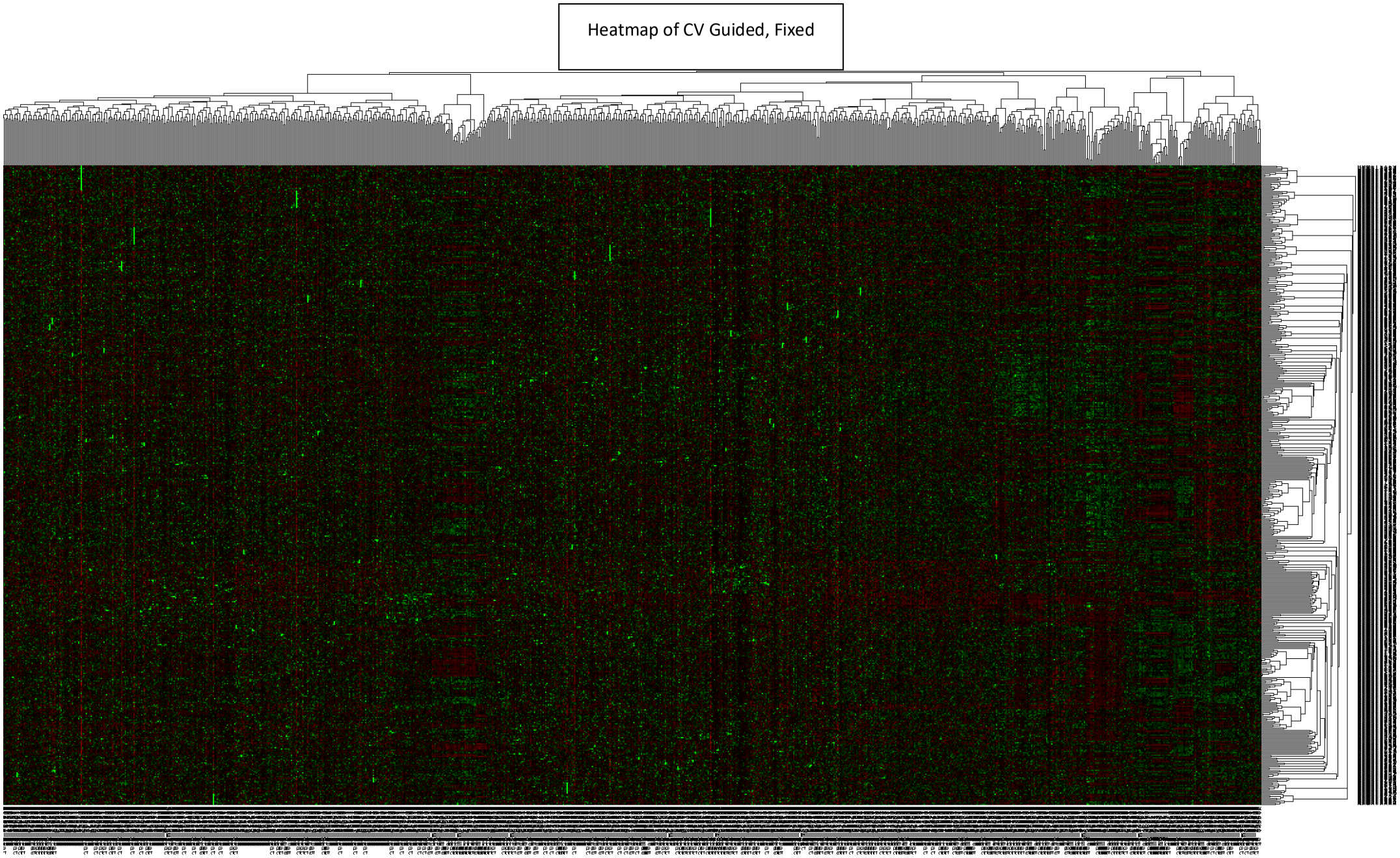

GSE25066 K−Means CV_Guided DFS
Pvalue =7.3e−07
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
samp_cluster=4
samp_cluster=5
samp_cluster=6
samp_cluster=7
samp_cluster=8
Survival Probability
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Months


GSE25066 K−Means CV_Guided DFS
Pvalue =4.4e−07
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
0
20
40
60
80
100
Months

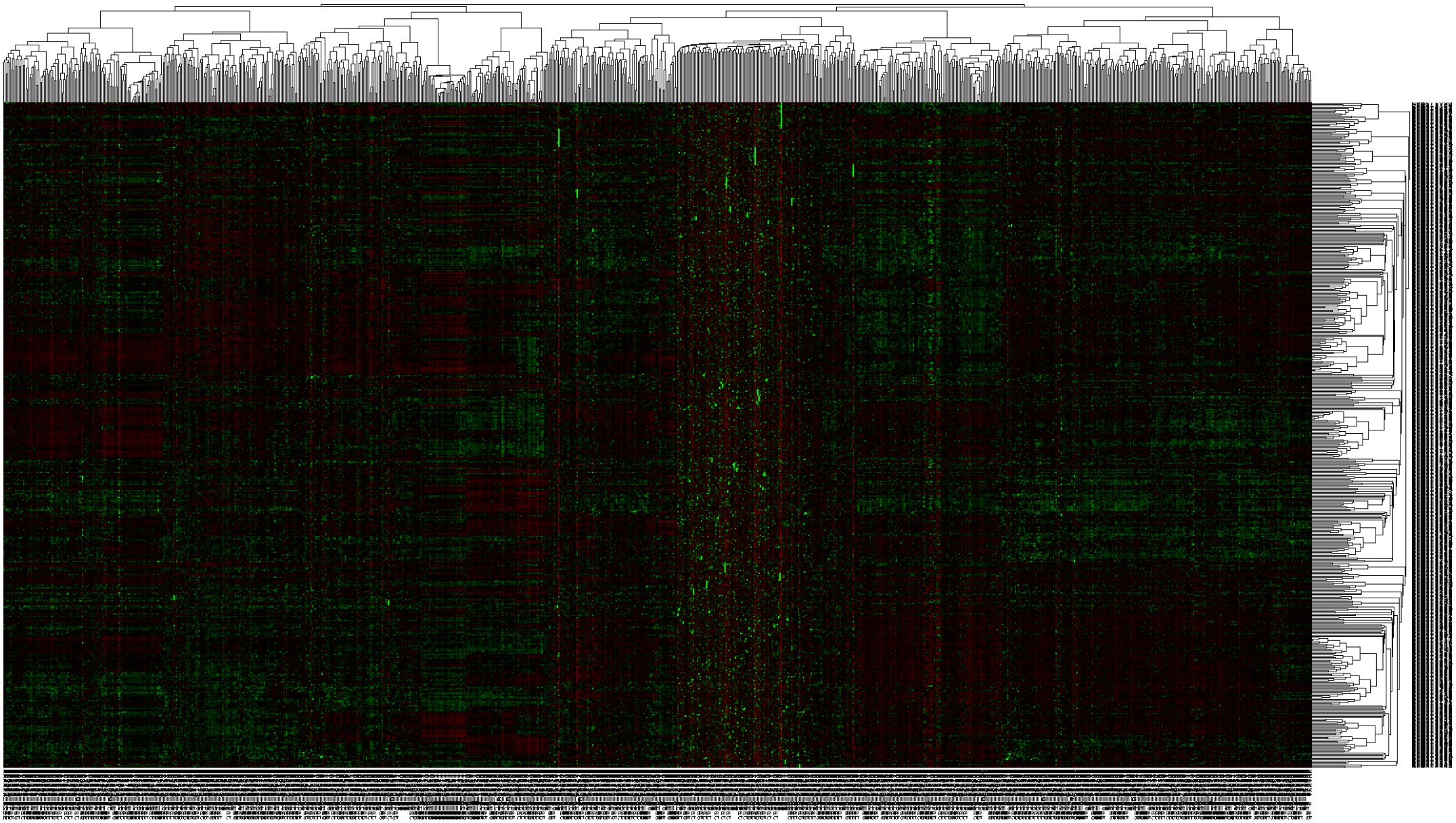

GSE25066 K−Means CV_Guided DFS
Pvalue =8.8e−06
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Months
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
samp_cluster=4
samp_cluster=5
samp_cluster=6
samp_cluster=7
samp_cluster=8


GSE25066 K−Means Poly DFS
Pvalue =4.4e−08
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
0
20
40
60
80
100
Months


GSE25066 K−Means Poly DFS
Pvalue =1.2e−06
Survival Probability
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Months
clusters
samp_cluster=1
samp_cluster=2
samp_cluster=3
samp_cluster=4
samp_cluster=5
samp_cluster=6
samp_cluster=7
samp_cluster=8


Heatmap of CV Guided, Fixed

Heatmap of Poly Ranking, Fixed



Heatmap of CV Guided, GMM