

M3C: Monte Carlo Reference-based Consensus Clustering

Christopher R John

13/02/2020

Summary

Genome-wide expression data is used to stratify patients into classes using clustering algorithms for precision medicine. The Monti consensus clustering algorithm (Monti et al., 2003) is a widely applied method to identify the number of clusters (K) through the principle of stability selection. This algorithm works by resampling and clustering the data for each K and a $N \times N$ consensus matrix is calculated, where each element represents the fraction of times two samples clustered together. A perfectly stable matrix would consist entirely of 0s and 1s, representing all sample pairs always clustering together or not together over resampling iterations. The next step is to compare the stability of these consensus matrices to decide K . The Proportion of Ambiguous Clustering (PAC) score (Senbabaoglu et al., 2014) has been proposed to assess consensus matrix stability for each K , however, it has bias towards greater values of K . This is due to a general problem with this type of consensus clustering algorithm that occurs because as K increases the consensus matrix converges towards a matrix of perfect stability simply by chance. The alternative well used delta K metric to find K is subjective as it relies on finding an elbow point and has been demonstrated to be inferior to the PAC score. Monte Carlo reference-based consensus clustering (M3C) (John et al., 2018) was made to solve these problems by comparing the real stability scores with those expected under a random model. M3C uses a Monte Carlo simulation to generate null distributions of stability scores along the range of K which, by comparing with the real stability scores, are used to decide the optimal K and reject the null hypothesis $K=1$.

Prerequisites

M3C recommended spec

For the Monte Carlo simulation method (method=1), M3C is best run on a relatively new and fast multi-core computer. If the machine is not particularly fast, the Monte Carlo iterations parameter may be reduced to 5-15.

The regularised consensus clustering method (method=2) that uses a penalty term to eliminate overestimation of K is faster, but the user will not get p values.

M3C requires

A matrix or data frame of normalised continuous expression data (e.g. microarray, RNA-seq, methylation arrays, protein arrays) where columns equal samples and rows equal features. M3C's reference will work better if the feature data is approximately normally distributed across biological replicates.

For example, for RNA-seq data, VST or rlog transformed count data, $\log_2(\text{CPM})$, $\log_2(\text{TPM})$, and $\log_2(\text{RPKM})$, are all acceptable forms of normalisation.

The data should be filtered to remove features with no or very low signal, and filtered using variance to reduce dimensionality (unsupervised), or p value from a statistical test (supervised). We include a feature filter function that uses variance which is demonstrated later in the vignette. This variance function should be applied after removing the tendency of the variance to increase with the mean (e.g. after VST or \log_2 transformation).

Outliers should be removed, we include an easy to use PCA function which has a text label parameter to help remove outliers (type ?pca for more information).

We recommend M3C only be used to cluster datasets with high numbers of samples (e.g. 60-1000). M3C is mainly aimed at large patient cohort datasets such as those produced by TCGA and other consortia. Because of the high complexity of the algorithm and the type of consensus matrix it makes, it is not that well suited for single cell RNA-seq data, better to use SC3 or Spectrum, for example.

M3C also accepts optionally

Annotation data frame, where every row is a patient or sample and columns refer to meta-data, e.g. age, sex, time until death, etc. M3C will automatically rearrange the annotation to match the clustering output and add the consensus cluster grouping to it. Note, this only works if the IDs (column names in data) match the entries in a column called “ID” in the user supplied annotation data frame.

Example I: TCGA glioblastoma dataset

The M3C package contains the glioblastoma (GBM) cancer microarray dataset for testing (reduced to 50 samples randomly). The original optimal cluster decision was 4. First, we load M3C which also loads the GBM data.

```
library(M3C)
# now we have loaded the mydata and desx objects (with the package automatically)
# mydata is the expression data for GBM
# desx is the annotation for this data
```

Exploratory data analysis

This is an important exploratory step prior to running M3C. It is best to remove extreme outliers. It is also important to be aware of the assumptions of PAM, K-means, and HC. K means, PAM, and HC assume the following;

- i) Clusters are approximately spherical - not severely elongated in one direction (anisotropic) or non-Gaussian
- ii) Clusters are approximately equal in variance

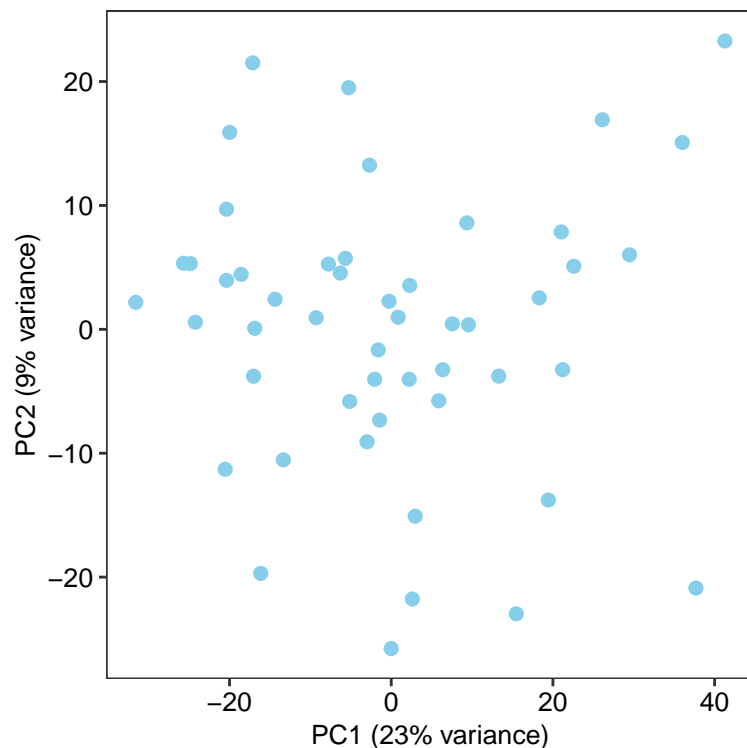
There is more info about the assumptions of clustering algorithms here: (<http://scikit-learn.org/stable/modules/clustering.html>)

Spectral clustering may be used to cluster more unusual structures using M3C, but this normally is not necessary. Ward’s hierarchical clustering (HC) is faster than these 3 algorithms and is an option included in M3C, although in practice PAM and KM usually perform better.

It is also important to check that batch effects and other uninteresting sources of variation do not drive the main variation in the data. The PCA function has a labels flag for adding categorical and continuous variables to the below plot (type ?pca). For a more quantitative analysis of these variables before clustering there is the ‘plot_drivers’ function from the biplotr package which produces a nice plot (<https://github.com/dswatson/biplotr>).

The data should have been transformed to be homoskedastic (or nearly so) by this point (e.g. vst or log2), otherwise the larger features will have larger variances and dominate the PCA and cluster analysis. If the data contains different kinds of features, e.g. both protein and miRNA, then they should also be z-score normalised to be comparable.

```
pca(mydata, legendtextsize = 10, axistextsize = 10, dotsize=2)
#> ***PCA wrapper function***
#> running...
#> done.
```



Running M3C

In our example, we run the algorithm using the default settings (25x Monte Carlo iterations and 100x inner replications). We have found the results generally stable using these parameters. If the user wants results quickly, it is reasonable to lower the Monte Carlo iterations parameter to 5-20x. The inner replications needs to be at least 100x, changing this up to 250x will lead to increased result stability on some datasets.

We use the PAC objective function here to minimise, but, we also have provided an entropy objective function demonstrated later in the vignette. It is the entropy objective function that is used as default. The subsequent steps are the same for either objective function, but entropy does not require setting a window or first calculating a CDF.

It is recommended to save the workspace after M3C if the user is working with a large dataset because the runtimes can be quite long. M3C uses by default PAM with Euclidean distance in the consensus clustering loop because we have found this runs fast with good results.

```
# for vignette
res <- M3C(mydata, des = desx, removeplots = TRUE, iters=25,
           objective='PAC', fsize=8, lthick=1, dotsize=1.25)
#> ***M3C***
#> method: Monte Carlo simulation
#> objective: pac
#> clustering algorithm: pam
#> annotation: yes
#> running simulations...
#> done.
#> running consensus cluster algorithm for real data...
#> done.
#> optimal K: 4
```

The scores and p values are contained within the `res$scores` object. We can see below the RCSI reaches a

maxima at $K = 4$, the p value supports this optimal K decision. This means the null hypothesis that $K = 1$ can be rejected for this dataset because we have achieved significance ($\alpha=0.05$) versus a dataset with no clusters.

```
res$scores
#>      K PAC_REAL PAC_REF      RCSI RCSI_SE MONTECARLO_P BETA_P
#> 1  2 0.6408163 0.5997061 -0.095389785 0.05071640 0.53846154 0.59389848
#> 2  3 0.4473469 0.5413224 0.170837061 0.04204458 0.23076923 0.18779576
#> 3  4 0.3526531 0.4743837 0.291862822 0.01984532 0.03846154 0.00378522
#> 4  5 0.3191837 0.4038204 0.230255475 0.02041057 0.03846154 0.01672057
#> 5  6 0.3061224 0.3408000 0.102214062 0.02078191 0.19230769 0.15858514
#> 6  7 0.2946939 0.2952163 -0.002312789 0.01847110 0.50000000 0.49948899
#> 7  8 0.2751020 0.2622694 -0.051777632 0.01816156 0.73076923 0.70591383
#> 8  9 0.2563265 0.2377796 -0.078919842 0.01787017 0.76923077 0.81139105
#> 9 10 0.2244898 0.2157061 -0.044229336 0.01911714 0.57692308 0.67487555
#>      P_SCORE
#> 1 0.22628778
#> 2 0.72631422
#> 3 2.42190885
#> 4 1.77674885
#> 5 0.79973750
#> 6 0.30147408
#> 7 0.15124831
#> 8 0.09076978
#> 9 0.17077631
```

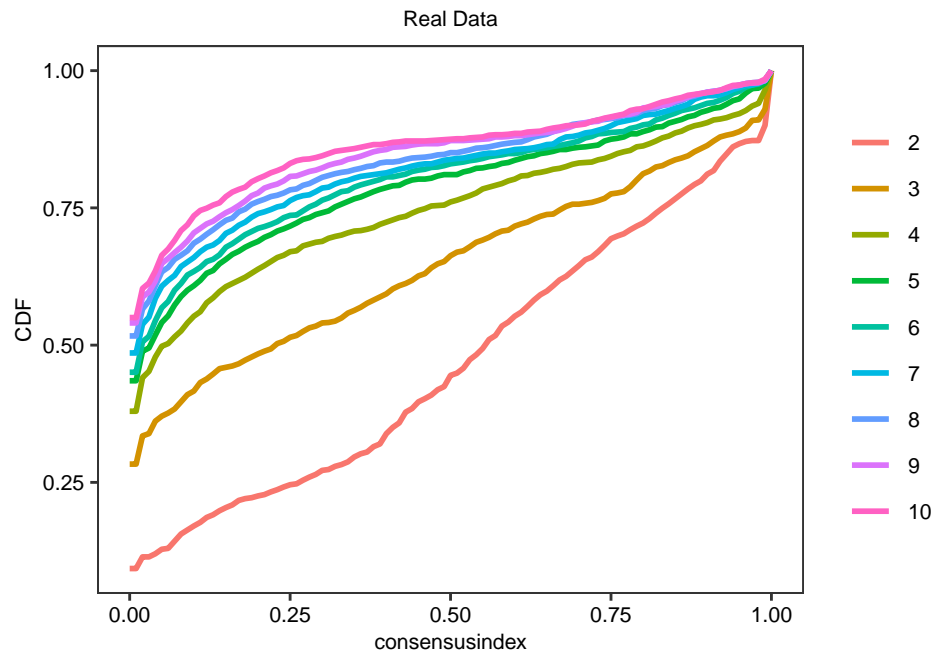
Also important is the relationship between the clinical variables and the discovered clusters. In this data we want to compare with a categorical variable so perform a chi-squared test. We are reassured to see below $K=4$ is highly significant.

```
for (k in seq(2,10)){
  myresults <- res$realdataresults[[k]]$ordered_annotation
  chifit <- suppressWarnings(chisq.test(table(myresults[c('consensuscluster','class')]))))
  print(chifit$p.value)
}
#> [1] 8.977979e-07
#> [1] 2.961138e-12
#> [1] 5.479839e-14
#> [1] 7.124742e-13
#> [1] 1.29408e-11
#> [1] NaN
#> [1] NaN
#> [1] NaN
#> [1] NaN
```

Now we will take a look at some of the plots M3C generates.

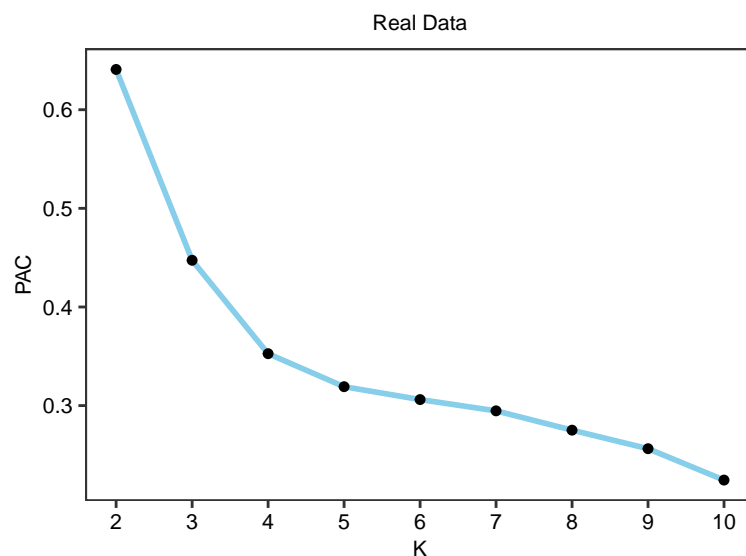
This is a CDF plot of the consensus matrices for the test data from $2 \dots \max K$. In the perfect case there would be a flat line to indicate a matrix of just 0s and 1s. CDF flatness can be quantified using the PAC metric (Senbabaoglu et al., 2014), or entropy can be used directly on the probabilities (see later section). In the CDF and following PAC plot we can see the inherent bias of consensus clustering where as K increases so does the apparent stability of the results (or CDF plot flatness), this we correct for by using a reference. This makes the method more sensitive to detection of the underlying structure in noisy data.

```
res$plots[[1]]
```



This figure below shows the PAC score (Senbabaoglu et al., 2014), we can see an elbow at $K = 4$ which is suggestive this is the best K . However, the bias of consensus clustering can be seen here as the PAC score naturally tends towards lower values as K increases (see above plot), making selecting K without taking this into account subject to bias. Selecting the minimal PAC score will only work when the clusters are very well separated.

```
res$plots[[2]]
```

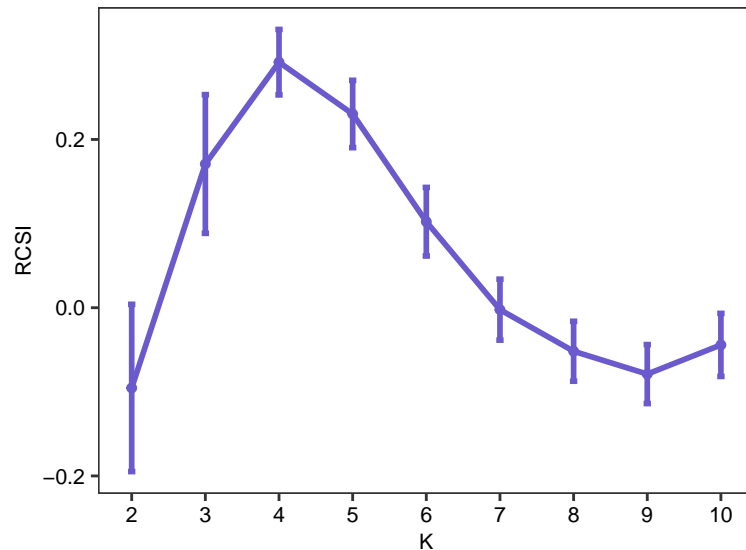


Another option would be to use the original delta K metric here from the Monti method, however, identifying the elbow in this plot or 'value before the floor' is often very subjective, especially on noisy datasets like commonly occur in precision medicine. This metric has been shown to perform poorly on real data (Senbabaoglu et al., 2014) and like the PAC score has the theoretical issue of not accounting for CDF

convergence as K increases.

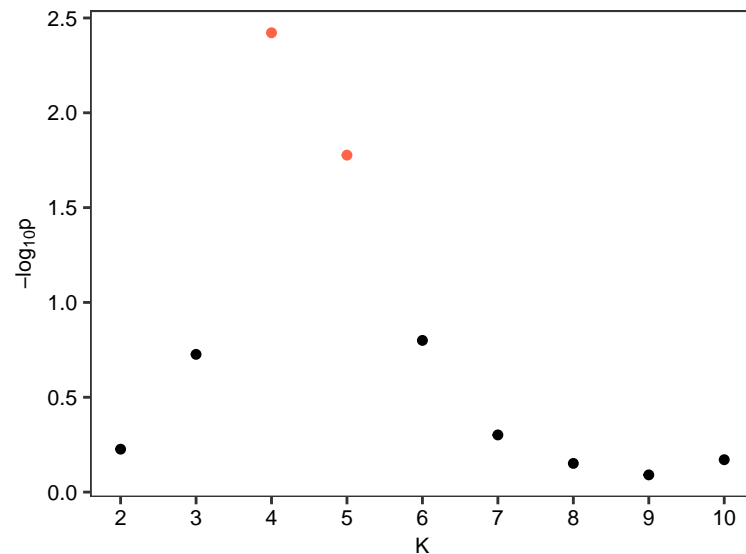
We derive the Relative Cluster Stability Index (RCSI) and associated 95% confidence intervals which take into account the reference PAC scores using the reference mean. This metric is better than the PAC score for deciding class number, where the maximum value corresponds to the optimal K . In this example the RCSI has an optima at $K=4$. Either the RCSI or p values can be used to select K , in the Bioinformatics paper we used M3C with the p values to select K (results in Supplementary Information).

```
res$plots[[4]]
```



Finally, we calculate a p value from the distribution, here we display the p values from the beta distribution. If none of the p values reach significance over a reasonable range of K (e.g. 10), then we accept the null hypothesis. In the GBM dataset, we can see $K = 4$ reaches significance with an alpha of 0.05, therefore we can reject the null hypothesis $K=1$ for the GBM dataset.

```
res$plots[[3]]
```



Now we are pretty convinced there are 4 clusters within this dataset which are not likely simply to have occurred by chance alone.

We can turn to examine the output objects that M3C generates. These allow heatmap generation for publications.

Understanding M3C outputs

The 3 lines below extract the ordered (according in clustering results) expression data and the ordered annotation data from the results object after running M3C for a 4 cluster solution. If we, for example, wanted to extract the data for a 5 cluster solution from the M3C results list, we would simply replace 4 in the below lines to 5. We then take a look at the annotation object M3C outputs, a consensus cluster column has been added by M3C.

```
data <- res$realdataresults[[4]]$ordered_data
annon <- res$realdataresults[[4]]$ordered_annotation
ccmatrix <- res$realdataresults[[4]]$consensus_matrix
head(annon)
```

| | <i>consensuscluster</i> | <i>class</i> |
|------------------------|-------------------------|--------------|
| #> TCGA.02.0048.01A.01 | 1 | Proneural |
| #> TCGA.08.0517.01A.01 | 1 | Proneural |
| #> TCGA.08.0350.01A.01 | 1 | Proneural |
| #> TCGA.08.0524.01A.01 | 1 | Proneural |
| #> TCGA.06.0648.01A.01 | 1 | Proneural |
| #> TCGA.02.0432.01A.02 | 1 | Proneural |

Example code for making consensus matrix heatmaps from M3C output with ComplexHeatmap.

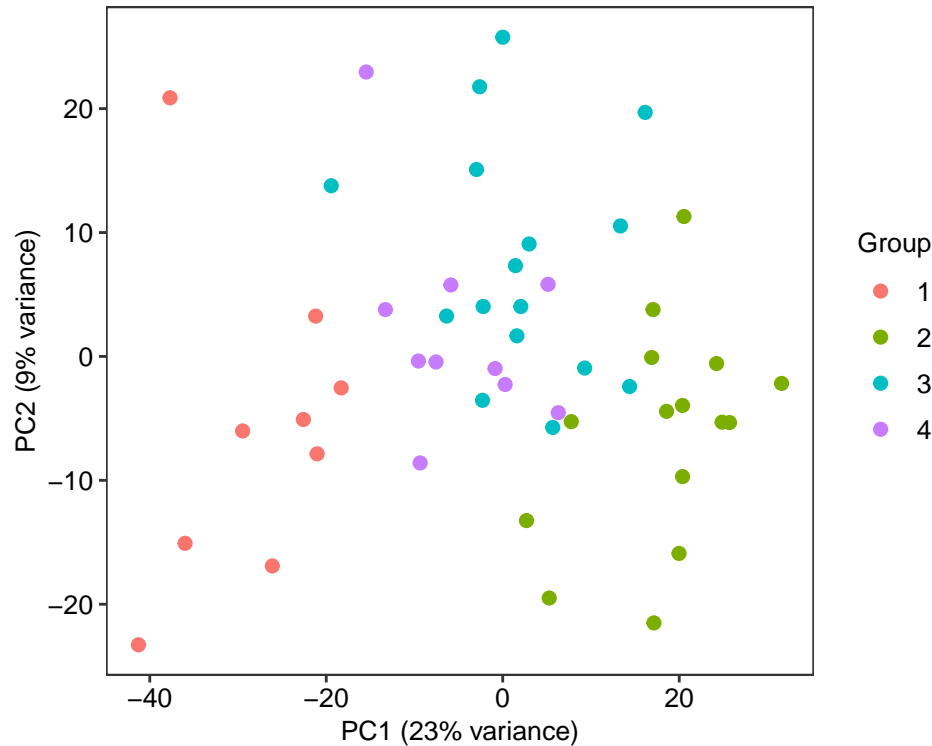
```
# library(ComplexHeatmap)
# ccl <- list()
# x <- c("skyblue", "gold", "violet", "darkorchid", "slateblue", "forestgreen",
#       "violetred", "orange", "midnightblue", "grey31", "black")
# names(x) <- as.character(seq(1,11,by=1))
# for (i in seq(2,10)){
#   # get cc matrix and labels
#   ccmatrix <- res$realdataresults[[i]]$consensus_matrix
#   annon <- res$realdataresults[[i]]$ordered_annotation
#   # do heatmap
#   n <- 10
#   seq <- rev(seq(0,255,by=255/(n)))
#   palRGB <- cbind(seq,seq,255)
#   mypal <- rgb(palRGB,maxColorValue=255)
#   ha = HeatmapAnnotation(
#     df= data.frame(Cluster=as.character(annon[,1])), col = list(Cluster=x))
#   ccl[[i]] <- Heatmap(ccmatrix, name = "Consensus_index", top_annotation = ha,
#     col=mypal, show_row_dend = FALSE,
#     show_column_dend = FALSE, cluster_rows = FALSE, cluster_columns = FALSE,
#     show_column_names = FALSE)
# }
```

Viewing consensus matrices manually should not be used to decide K, better to use the RCSI or p values that M3C provides to quantify the mixing proportions in the consensus matrix versus the null model.

Visual check of consensus cluster structure

A last analysis we recommend to do is to examine how the clusters are arranged in principal component space. It is also possible to run t-SNE and UMAP on the data, see ?tsne or ?umap.

```
pca(data, labels=annon$consensuscluster, legendtextsize = 10, axistextsize = 10, dotsize=2)
#> ***PCA wrapper function***
#> running...
#> done.
```



Doing a visual check like this allows the user to ensure there are no obvious artifacts that have occurred.

Example II: Regularised consensus clustering

Running regularised consensus clustering

The second method minimises $\log(\text{PAC})$ subject to λK to find the optimal K . The resultant metric is called the Penalised Cluster Stability Index (PCSI). To tune λ , we use the log-likelihood and search a sequence of λ to find the K with the higher log-likelihood. A good predefined λ value to use is 0.1, this can be used by setting `tunelambda` to `FALSE`, this is the fastest way of running M3C - without this tuning.

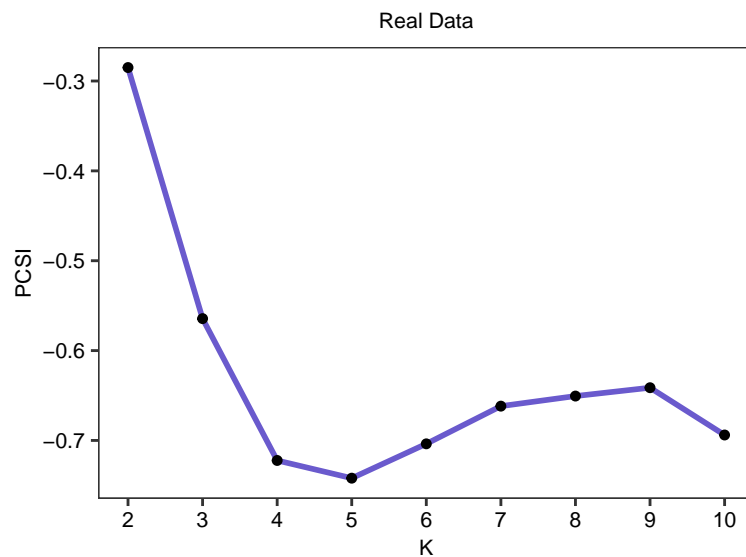
The log-likelihood is calculated through clustering once from $2 \dots \text{maxK}$ then this gives the ground truth probabilities, the perturbed probabilities for each K are those made from the resampling algorithm. Ultimately, the better K (or λ) is that which gives the minimal distance between these two sets of probabilities. We divide through by N to normalise the log-likelihood and only select the positive class because this reduces the bias of the algorithm.

```
res <- M3C(mydata, method = 2, objective='PAC', fsize=8, lthick=1, dotsize=1.25)
#> ***M3C***
#> method: regularised consensus clustering
#> objective: pac
#> clustering algorithm: pam
#> annotation: none
#> running consensus cluster algorithm for real data...
```



```
#> done.
#> tuning lambda: 0.02
#> K = 10 , log-likelihood -0.36506995090234
#> tuning lambda: 0.04
#> K = 10 , log-likelihood -0.36506995090234
#> tuning lambda: 0.06
#> K = 10 , log-likelihood -0.36506995090234
#> tuning lambda: 0.08
#> K = 5 , log-likelihood -0.228027219885955
#> tuning lambda: 0.1
#> K = 4 , log-likelihood -0.314250369108803
#> optimal lambda: 0.08
#> optimal K: 5
```

```
res$plots[[3]]
```



For this method, it is faster than running a Monte Carlo simulation so we could increase the ‘repsreal’ parameter to e.g. 250 for potential for increased result stability.

Example III: Entropy objective function

Treating the consensus matrix elements as probabilities makes entropy a very suitable objective function for consensus clustering. We use information entropy and aim to minimise it to find K, where lower values correspond to less uncertainty in the system (more stability during resampling). Whilst this function often gives pretty similar results to the PAC score, it makes better theoretical sense because it does not rely on a subjective window, neither does it rely on calculating a CDF first and a metric from this CDF second.

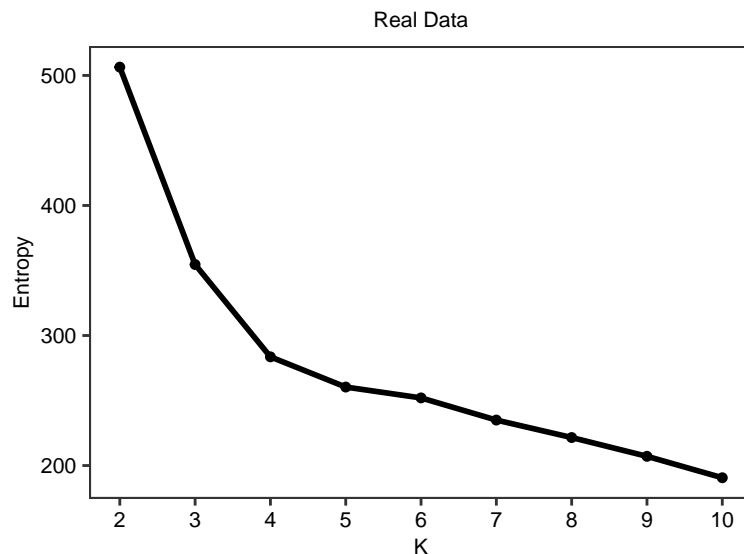
Entropy can be used with the Monte Carlo simulation or the regularised consensus clustering method we provide.

```
res <- M3C(mydata, method = 2, fsize=8, lthick=1, dotsize=1.25)
#> ***M3C***
#> method: regularised consensus clustering
#> objective: entropy
#> clustering algorithm: pam
#> annotation: none
```

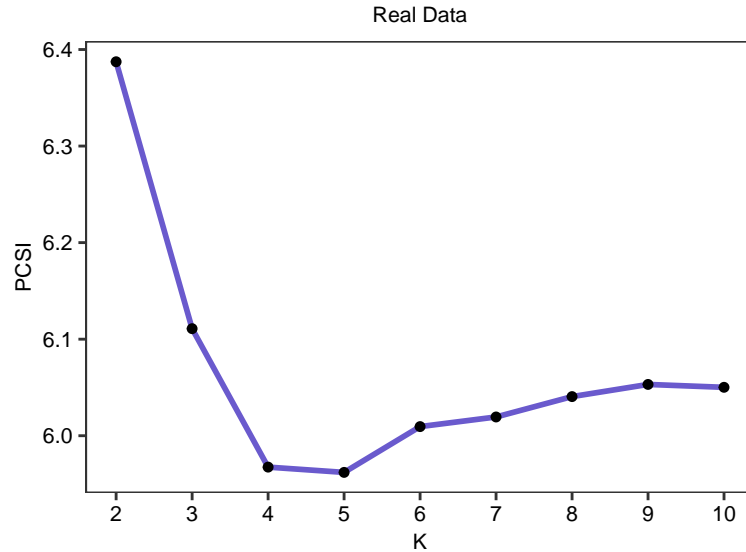
```
#> running consensus cluster algorithm for real data...
#> done.
#> tuning lambda: 0.02
#> K = 10 , log-likelihood -0.36506995090234
#> tuning lambda: 0.04
#> K = 10 , log-likelihood -0.36506995090234
#> tuning lambda: 0.06
#> K = 10 , log-likelihood -0.36506995090234
#> tuning lambda: 0.08
#> K = 5 , log-likelihood -0.228027219885955
#> tuning lambda: 0.1
#> K = 4 , log-likelihood -0.314250369108803
#> optimal lambda: 0.08
#> optimal K: 5
```

This is entropy not corrected for bias. We can see it is subject to the same problem the PAC score is as it tends to decrease with increasing K.

```
res$plots[[2]]
```



```
res$plots[[3]]
```



Additional functions

Filtering features by variance

A simple feature filter function has been included in M3C that uses one of variance, median absolute deviation (MAD), Pearson's coefficient of variation (A), or its second order derivative (A2) to filter the data. MAD will yield similar results to variance, however, because it uses the median instead of the mean it will be more robust to outliers. This will be the preferable metric in many cases. The statistics for each feature (and the filtered data itself) are saved in a convenient data frame in the results list for plotting and further analysis.

Below, we set the percentile to filter by and input the data. The code below extracts the 10% most variable features using MAD as the filtering metric. In this case the example data has been already been filtered for variance then scaled additionally, so the results are not intended to be interpreted as it is just an example. The statistics for the topN features will be printed to the console. Type `?featurefilter` for more information.

```
filtered_results <- featurefilter(mydata, percentile=10, method='MAD', topN=5)
#> ***feature filter function***
#> extracting the most variable: 10 percent
#> features to start with: 1740
#> performing calculations for median absolute deviation
#> printing topN most variable features with statistics...
#>      feature      mean      var      sd      MAD
#> POSTN POSTN  0.2967294  7.301561  2.702140  3.205915
#> LTF      LTF   0.0117520  4.580893  2.140302  3.046446
#> DCX      DCX  -0.0115316  3.583629  1.893047  2.706227
#> TMSL8    TMSL8  0.0846228  4.988073  2.233399  2.652757
#> NNMT     NNMT   0.2344958  3.510178  1.873547  2.527811
#> features remaining: 174
```

Closing comments

In this document, we have seen that M3C provides a rigorous approach for selecting the number of clusters in the data. We have found in our analyses this approach results in increased performance relative to other methods and the removal of systematic bias inherent in the results. M3C's methodological and software developments are primarily built on the work of Tibshirani et al. (2001), Senbabaoglu et al. (2014), and

Monti et al. (2003). Notably, it is best to use M3C after careful consideration of the format of the data, its normalisation and transformation, correction for any batch effects, and in conjunction with dimensionality reduction tools such as PCA and t-SNE to confirm the structure.

During the development of M3C, we also devised a flexible method and associated CRAN package for generating Gaussian clusters called, `clusterlab` (<https://cran.r-project.org/web/packages/clusterlab/index.html>). This tool should prove useful in testing the performance of class discovery algorithms.

For fast clustering of multi-omic data, we recommend `Spectrum` (<https://cran.r-project.org/web/packages/Spectrum/index.html>) (John et al., 2019). `Spectrum` is a self-tuning spectral clustering algorithm that can integrate heterogenous data sources and reduce noise. It uses a tensor product graph data integration and diffusion procedure, a density-aware kernel, and contains an optional data compression method to cluster 1000s of samples quickly. The method is also effective at single view cluster analysis.

For analyses demonstrating the efficiency of M3C, please see either the Supplementary Information of the Bioinformatics spectral clustering manuscript where it was tested in parallel with `Spectrum` on several TCGA RNA-seq datasets, with the p value method used to select K, or the M3C manuscript which is on bioRxiv.

M3C code is hosted here: <https://github.com/crj32/M3C>.

References

Christopher R John, David Watson, Michael R Barnes, Costantino Pitzalis, Myles J Lewis, `Spectrum`: fast density-aware spectral clustering for single and multi-omic data, *Bioinformatics*, , btz704, <https://doi.org/10.1093/bioinformatics/btz704>

John, Christopher Robert, et al. “M3C: A Monte Carlo reference-based consensus clustering algorithm.” *bioRxiv* (2018): 377002.

John, Christopher R., et al. “`Spectrum`: Fast density-aware spectral clustering for single and multi-omic data.” *BioRxiv* (2019): 636639.

Monti, Stefano, et al. “Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data.” *Machine learning* 52.1 (2003): 91-118.

Kvålseth, Tarald O. “Coefficient of variation: the second-order alternative.” *Journal of Applied Statistics* 44.3 (2017): 402-415.

Senbabaoglu, Yasin, George Michailidis, and Jun Z. Li. “Critical limitations of consensus clustering in class discovery.” *Scientific reports* 4 (2014): 6207.

Tibshirani, Robert, Guenther Walther, and Trevor Hastie. “Estimating the number of clusters in a data set via the gap statistic.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001): 411-423.