`timeclust`: A Function for Sequential Clustering of Numerical and Timestamp Data

Matej Poliaček

supervised by
Dr Yashar Moshfeghi

Contents

List of Tables

List of Figures

Declaration

I confirm that this assignment is my own work and that I have:

- Read and understood the guidance on plagiarism in the Student Handbook, including the University of Glasgow Statement on Plagiarism
- Clearly referenced, in both the text and the bibliography or references, all sources used in the work
- Fully referenced (including page numbers) and used inverted commas for all text quoted from books, journals, web etc.
- Provided the sources for all tables, figures, data etc. that are not my own work
- Not made use of the work of any other student(s) past or present without acknowledgement. This includes any of my own work, that has been previously, or concurrently, submitted for assessment, either at this or any other educational institution, including school (see overleaf at 31.2)
- Not sought or used the services of any professional agencies to produce this work
- In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with University regulations.

I am aware of and understand the University's policy on plagiarism and I certify that this assignment is my own work, except where indicated by referencing, and that I have followed the good academic practices noted above.

1 Introduction

The motivation for the `timeclust` function arose from the need to cluster timestamped multidimensional data, where the timestamp serves as a secondary source of separation between the data points. Such static data could be an outcome of a past data stream, where the clustering has not been performed in real time as the data were collected. Existing R packages only offer solutions for real time clustering of incoming data, for example *rEMM: Extensible Markov Model for Data Stream* by Michael Hahsler and Margaret H. Dunham, *A Framework for Clustering Evolving Data Streams* by Charu C. Aggarwal et. al, and related works.

The problem can be addressed by creating a two-step clustering function employing methods available in R. To complement these for better functionality, a `k-POD` clustering algorithm is also used in the function (Chi et al., 2016). Finally, to ensure that the function can be tested on several dataset, the `MixSim` package was used and expanded upon, to generate timestamped multidimensional data from varying distributions, in order to provide a basis for clusters.

2 Methods

As no suitable dataset was available, the data used to test the `timeclust` function were simulated. The clustering is then performed by one of the available algorithms and parameters depending on the choice of the user. The performance of the clustering in comparison to the true division of data was then evaluated using Adjusted Rand Index.

2.1 Data Simulation - Expanding the `MixSim` package

The need to create custom simulated data arose from the necessity to evaluate the performance of the clustering algorithm and immediate lack of suitable datasets. For this purpose, the varying origin of the data that will serve as the basis for clustering must be known to the user. The `MixSim` package provides simulation of data drawn from separate Gaussian distributions with user-defined overlap. The function outputs a desired number of observation across specified dimensions. The resulting data can then also retain the "true" labels, based on their source distribution (Melnykov et al., 2016). Generating multiple datasets also allows to test the function on data with varying features.

The package was utilised and customised for the purposed of the clustering problem. This function expands the `MixSim` package to provide additional functionality.

2.1.1 Inputs

- `dim` - positive integer - number of dimensions of each observation (represented by columns)
- `clust` - positive integer - number of true clusters for output data
- `size` - positive integer - number of observation in the data (rows)
- `overlap` - positive double - overlap value to be passed to `MixSim` initialisation, smaller numbers generate more separated clusters. Works reliably up to the value 0.75, after which the simulation times become long
- `timestamp` - boolean - indicate whether randomised timestamp data should be generated
- `unique_timestamps` - boolean - if `TRUE`, observations can have the same timestamp
- `regular` - boolean - if `TRUE`, timestamps will be spaced out equally, otherwise they will be random
- `date_start` - date of the format `'yyyy/mm/dd'` (including apostrophes) - the earliest date point, from which timestamps will be generated. Can remain `NULL` if timestamp is set to `FALSE`.
- `date_end` - date of the format `'yyyy/mm/dd'` (including apostrophes) - the latest date point, to which timestamps will be generated. Can remain `NULL` if timestamp is set to `FALSE`.
- `NA_val` - positive integer - the number of missing values to generate in the data set. If 0 or `NULL` (default), no missing values will be generated

2.1.2 Output

The function returns the resulting data frame. It is thus necessary to save the output into a variable e.g.:

```
simulated_output <- create_data(...)
```

The `dim`, `clust`, `size` and `overlap` values are passed to the `MixSim` function to generate the initial complete dataset with the desired dimensions, observations, clustering and overlap. After the complete data is returned, NA values are generated and replace random entries, if the value of `NA_val` is specified to be greater than 0. A column of "true" cluster membership is also stored.

If `timestamp` is set to `TRUE`, the next step will generate a vector of timestamp data, containing a timestamp for each observation (row). These will be either completely randomly spaced, random but unique (i.e. one observation per day) if `unique_timestamps` is `TRUE`, or regularly spaced if `regular` is set to `TRUE`. The timestamps are then appended to the rest of the data as the first column. The entire data frame is then sorted based on the timestamp data, in an ascending fashion.

2.2 `timeclust` function

The *timeclust* function utilised methods available natively in R, namely hierarchical clustering and k-means clustering. An external package `k-POD` is also employed to handle cases where the numerical data contains missing values.

The timeclust fucntion uses native R clustering fuctions to perform a 2 stage clustering of timestamp-labelled data. The two stages are used to cluster numerical data and timestamp data respectively.

The methods of unsupervised learning are described below. Furthermore, to evaluate the performance of the clustering algorithms the Adjusted Rand Index will be used.

2.2.1 Unsupervised Learning

Unsupervised learning is a process of discovering patterns in the dataset. When the pattern to be discovered is groups of data with similar characteristics, the process is referred to as clustering (Bishop, 2006). Unsupervised learning problems do not have any groups or classes laid out prior to the analysis (as opposed to supervised learning). Unsupervised learning thus does not require extensive input from an expert in the field in order to determine the class labels, and furthermore, such labels could prove to be unreliable in terms of discovering the groups in the data (Murphy, 2012). In this problem, the data labels are present, however the algorithms are blind to them when performing the clustering - their purpose is to enable posterior evaluation of the algorithms.

2.2.1.1 Clustering with K-Means

K-means clustering is a method of identifying groups of data in a multidimensional space by partitioning. The data is partitioned into a number clusters, so that the inter-point distances of the data points in the cluster are small compared with the distances to points outside of the cluster. The goal is to find an assignment of data points to clusters, as well as a set of vectors representing the cluster centres, so that the sum of the squares of the distances of each data point to its closest cluster centre is a minimum.

By using a binary indicator variable that indicates to which cluster the data point belongs, an objective function can be constructed which represents the sum of the squares of the distances of each data point to the centre of the cluster it is assigned to. An iterative approach is then employed to find values for the indicator variable and cluster centres in order to find the minimum of this function.

The iterative process consists of two steps. To initialise, the cluster centres are randomly selected. In the first step, the function is minimised with respect to the cluster indicator variable and the cluster means remain fixed. Then, in step two, the indicator variable is unchanged and the function is minimised by optimising the cluster centres. These two steps are then repeated until convergence (Bishop, 2006).

The primary issue of this methodology lies in its initialisation, and thus on the choice of numbers of clusters. One method of deciding the number of clusters is to plot the reconstruction error of the dataset set versus the number of clusters, and to try to identify a "knee" or "kink" in the curve. The idea is that for a number of clusters (that is less than the "true" number of clusters) the rate of decrease in the error function will be high, since groups that are being separated should not be grouped together to begin with. If, however, the number of clusters is larger than the "true" number of clusters, "natural" clusters are being split, which does not reduce the error as significantly. Moreover, identifying the aforementioned "kinks" can be problematic, since the plotted function may decrease gradually (Murphy, 2012). Interpreting such output therefore requires subjective reasoning, which may lead to varying results.

2.2.1.2 K-Means for data with missing entries - `k-POD` package

`k-POD` clustering is an alternative to deleting or imputing data to eliminate missing entries. Since k-means requires a complete data matrix, the potential missing values must be addressed before applying the algorithm. This will either involve removing incomplete observations and thus reducing the dataset, or imputation of values by estimating the missingness mechanism.

`k-POD` clustering does not require assumptions on missingness pattern, and thus prevents incorrect imputation which might result in incorrect clustering. It works with the missing values directly, so removing observations is also not necessary. Instead, the missing data are addressed by developing a simple majorization-minimalization algorithm. The basic strategy behind an MM algorithm is to convert a hard optimization problem into a sequence of simpler ones. In effect, the `k-POD` clustering algorithm works by updating the unobserved portion of data based on initialised values. Then a k-means algorithm is used to perform the clustering. The results is used to update the initialised values and the process is then repeated until convergence. More detailed breakdown of the algorithm can be found in the paper by Chi et al. (2016). `k-POD` clustering algorithm is available in the `kpodclustr` package.

2.2.1.3 Hierarchical Clustering

Hierarchical clustering is a deterministic clustering method executed by creating a nested tree of partitions. As a deterministic method, it does not require an initial specification of clusters and thus is not sensitive to the initial conditions required for other methods (e.g. mainly flat clustering methods).

Hierarchical clustering builds on the input given by distance of the objects to be clustered. The term distance signifies a mathematical measure of dissimilarity. There are various methods of calculating the distance, dependent on the type of the variable. For real-valued attributes, Murphy (2012) provides Euclidean distance and a more robust city block distance as examples of common attribute dissimilarity functions.

Two main approaches to constructing clusters hierarchically can be considered, both of which use the measure of dissimilarity as output. The first of the two, the bottom-up method, merges most similar groups at each step. The second approach - the top-down method - splits groups using different criteria. Both bottom-up and top-down hierarchical clustering methods will cluster the data regardless of whether a structure is present in the data or not. In the event there is no structure in the data, the resulting clusters are merely random noise of the data. The choice between the top-down and the bottom-up approach does not optimise any well-defined objective function.

In the bottom-up method it is necessary to establish how the dissimilarity between groups of objects should be defined. There are 3 variants and each of them can potentially give different results.

- Single linkage (nearest neighbour clustering) - the distance between two groups is defined as the distance between the two closest members of the group. When merging two clusters, the two closest members of the clusters are connected. Since only the two closest members are considered, this pair is sufficient to consider the two groups to be close together. This can lead to the violation of the compactness property, i.e. the property that a all the observations within the group should be similar to each other.
- Complete linkage (furthest neighbour clustering) - the distance between two groups is defined as the distance between the two most distant members of the group. In this case, two groups are considered close based on all the observations in the union, and only when all of them are relatively similar. This tends to result in small (compact) clusters.
- Average linkage - this method measures the average distance between all pairs. The resulting clusters have a tendency of being relatively compact and relatively far apart. However, due to presence of the averaging, changes to the measurement scale pose a risk of changing the result. This is not the case with single and complete linkage, as they are invariant to monotonic transformations of the member pairs. This approach is preferred in practice.

The process of merging clusters can be displayed using a dendrogram (a binary tree). The objects start at the bottom of the diagram individually and are merged together according to the selected approach. The distance between the merging points (i.e. the height of the branch) represents the dissimilarity between the groups joined. The top of the chart is the group containing all of the objects. Cutting the tree at a given height results in a clustering of a given size.

In the `timeclust` function, the linkage and distance methods can be specified by the user, in case hierarchical clustering is applied to data. The choice of these methods will depend on the format and the nature of the data.

2.2.1.4 Adjusted Rand Index

The **adjusted Rand index** is a method of evaluating similarity between two vectors of labels. It can be used to evaluate the performance of a clustering algorithm in case that true cluster labels are known.

The **Rand index** is derived by dividing the sum of true positives and true negatives by the sum of true positives, true negatives, false positives and false negatives, a summary statistic is obtained that ranges from 0 to 1. The **adjusted Rand index** takes the evaluation of (index - expected index) divided by (max index - expected index). The resulting value will also lie on the interval from 0 to 1, where 0 indicates no match between the input label vectors, and 1 indicating a perfect match (Murphy, 2012).

2.2.2 Inputs

- `clust_data` - a data frame or a matrix containing real numerical values
- `time_data` - a vector of dates containing timestamps of the data
- `clust_method` - the method for clustering numerical data - either `"hclust"`, `"kmeans"` or `"kpod"` (including quotation marks)
- `k` - positive integer - the desired number of clusters for the numerical data
- `dist_m` - the distance method for the numerical data, in case `clust_method` is set to `"hclust"` - any native R method is possible (e.g. `"euclidean"`, `"maximum"`, `"manhattan"`, etc., including quotation marks)

- `hclust_m` - the linkage method for the numerical data, in case `clust_method` is set to `"hclust"` - any native R method is possible (e.g. `"single"`, `"complete"`, `"average"`, etc., including quotation marks)
- `time_method` - the method for clustering timestamp data - either `"hclust"`, `"kmeans"` or `"kpod"` (including quotation marks)
- `k_t` - positive integer - the desired number of clusters for the timestamp data
- `dist_m_t` - the distance method for the timestamp data, in case `time_method` is set to `"hclust"` - any native R method is possible (e.g. `"euclidean"`, `"maximum"`, `"manhattan"`, etc., including quotation marks)
- `hclust_m_t` - the linkage method for the timestamp data, in case time_method is set to "hclust" - any native R method is possible (e.g. `"single"`, `"complete"`, `"average"`, etc., including quotation marks)
- `timefirst` - boolean - if $^{\text{TM}}$, numerical data is clustered first, then timestamp based clusters are determined within the numerical clusters (and vice versa for `TRUE`)

2.2.3 Output

The function returns a data frame with two columns named `timeLabels` and `dataLabels` that need to be stored, e.g.:

```
timeclust_output_clusters <- timeclust(...)
```

The function uses built-in R functions `hclust` and `kmeans` to perform clustering. In case missing values are observed in the numerical data, the method used is k-POD clustering, from the `kpodclustr` package. The functions performs clustering in two stages. The method of clustering can be specified for either of these individually, as can be the desired number of clusters, the distance method employed and the linkage used if hierarchical clustering is used. The order is depends on whether the setting of the `timefirst` parameter.

If `timefirst` is set to `FALSE`, the numerical data saved in `clust_data` is clustered first using the clustering method defined in `clust_method`, with the number of clusters specified in `k`. If hierarchical clustering is selected in `clust_method`, the parameter `dist_m` can be specified to choose a method for distance measurement (default: `"euclidean"`), and `hclust_m` can be specified to select linkage method (default: `"average"`). Distance and linkage are passed to the built-in `hclust` function, therefore any supported methods can be used. Subsequently, the `time_data` is clustered using the clustering method defined in `time_method`, with the number of clusters specified in `k_t`. `k_t` clusters are determined within each of the numerical clusters individually. Again, if hierarchical clustering is selected in `time_method`, the parameter `dist_m_t` can be specified to choose a method for distance measurement (default: `"euclidean"`), and `hclust_m_t` can be specified to select linkage method (default: `"average"`).

Alternatively, if `timefirst` is set to `TRUE`, the timestamp data saved in `time_data` is clustered first using the clustering method defined in `time_method`, with the number of clusters specified in `k_t`. If hierarchical clustering is selected in `time_method`, the parameter `dist_m_t` can be specified to choose a method for distance measurement (default: `"euclidean"`), and `hclust_m_t` can be specified to select linkage method (default: `"average"`). As in the previous case, distance and linkage are passed to the built-in `hclust` function, therefore any supported methods can be used. Afterwards, the numerical data stored in `clust_data` are clustered within each of the clusters from the previous clustering step. These are clustered into clusters stored in `k`. If hierarchical clustering is chosen, distance and linkage can be specified with `dist_m` and `hclust_m` as before.

3 Numerical Analysis

The data simulation and clustering function will be demonstrated and then tested. First, a complete two dimensional data set will be used to show the functionality of the data simulation and clustering. Similarly, a 2D set with some missing values will be used. The complete 2D data will also demonstrate the clustering when timestamp data is processed first.

Afterwards, the results of two test runs - one with varying dimensions of simulated data and one with varying overlap of the distributions (and thus clusters) - will be presented to evaluate the performance of the `timeclust` function.

3.1 Basic Demonstration - 2D Data

The data simulation is run with 2 dimensions and 5 clusters, i.e. resulting in 2 numerical vectors drawn from 5 separate distributions. The `size` setting results in 100 observations. The overlap is set to 0.01, which should provide a reasonable separation between the distributions, and thus clusters. The timestamp data will be drawn from dates starting 01/01/2000 and ending 01/01/2010. Since `unique_timestamps` and `regular` are not specified, their default values will be used (both `FALSE` in this case). This means that the timestamps are generated with replacement and with random interval between individual timestamp. Furthermore `NA_val` is not specified, meaning that the default value (`NULL`) will be used, i.e. no missing values will be present in the dataset. The code used thus is:

```
test_run_2D <- create_data(dim=2, clust=5, size=100, overlap=0.01, timestamp = TRUE,
                date_start = "2000/01/01", date_end = "2010/01/01")
```

The `timeclust` algorithm is used with the following parameters: first the numerical and timestamp data are set to appropriate , columns of the data frame. The method for clustering numerical data is set to `"kmeans"`, with 5 target clusters. The method for clustering timestamp data is hierarchical clustering using euclidean distance and complete linkage. The desired number of clusters for timestamp data is 2. The `timefirst` parameter is set to `FALSE`, indicating the numerical data is clustered first, and the clusters for timestamp data are then determined within these clusters.

```
test_clust_2d <- timeclust(clust_data = test_2d[,2:3], time_data = test_2d[,1],
 clust_method = "kmeans", k=5, time_method = "hclust", dist_m_t = "euclidean",
             hclust_m_t = "complete", k_t = 2, timefirst = FALSE)
```

The resulting clustering of the numerical data can be then compared with the true clusters of the data, stemming from the origin distribution. The method used is the `adjustedRand` from the package `clues`, which is a requirement of the `kpodclustr` package.

```
     Rand
0.9852525
```

This value is very close to 1, indicating near perfect clustering compared to the origin of the data. Plotting the data with colour indicating origin in one plot and cluster assignment in another allows visual

inspection of clustering performance. Figure 3.1 shows the original data distinguished by their origin distribution, whereas Figure 3.2 shows the results of clustering using `timeclust` function. Figure 3.2 also distinguishes the points further with different label shapes to show the further in-cluster division based on the assignment to either of the 2 clusters stemming from the timestamp data. Visual inspection shows that a small number of points was mis-assigned do to their proximity to or presence inside a cluster of another group. Table 3.1 shows the resulting assignment to clusters - in case of a 100% correct clustering, there would only be one entry per row/column. In this case there are two entries where the cluster was determined incorrectly in comparison to the origin distribution.

	1	2	3	4	5
1	0	0	15	1	0
2	0	15	0	1	0
3	22	0	0	0	0
4	0	0	0	0	25
5	0	0	0	21	0

Table 3.1: Assignment of the complete dataset points to clusters vs. their true cluster membership based on the origin distribution

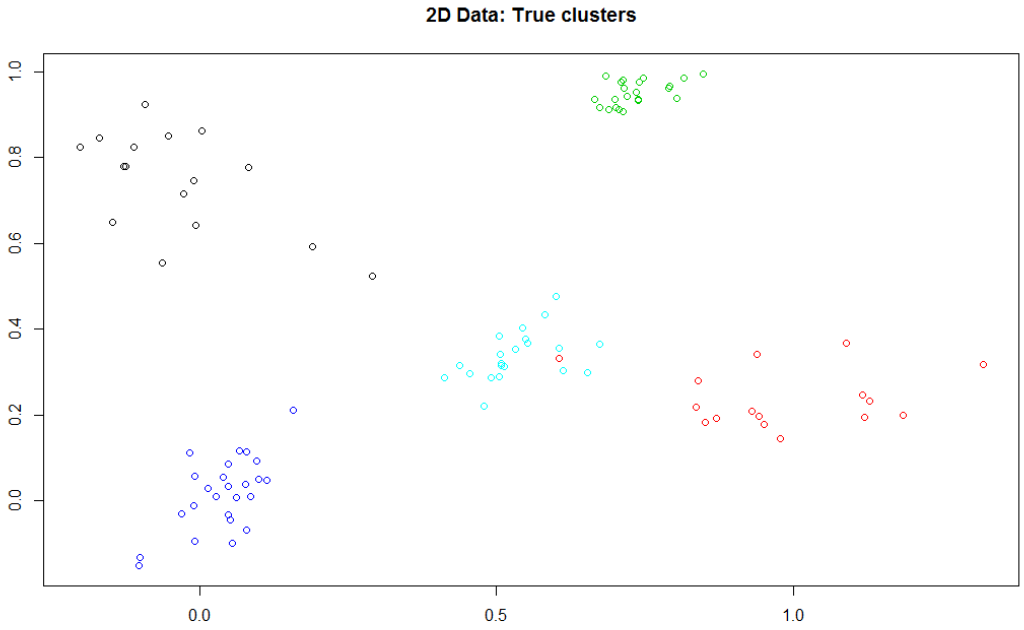

Figure 3.1: 2D Data, colour displaying true cluster membership

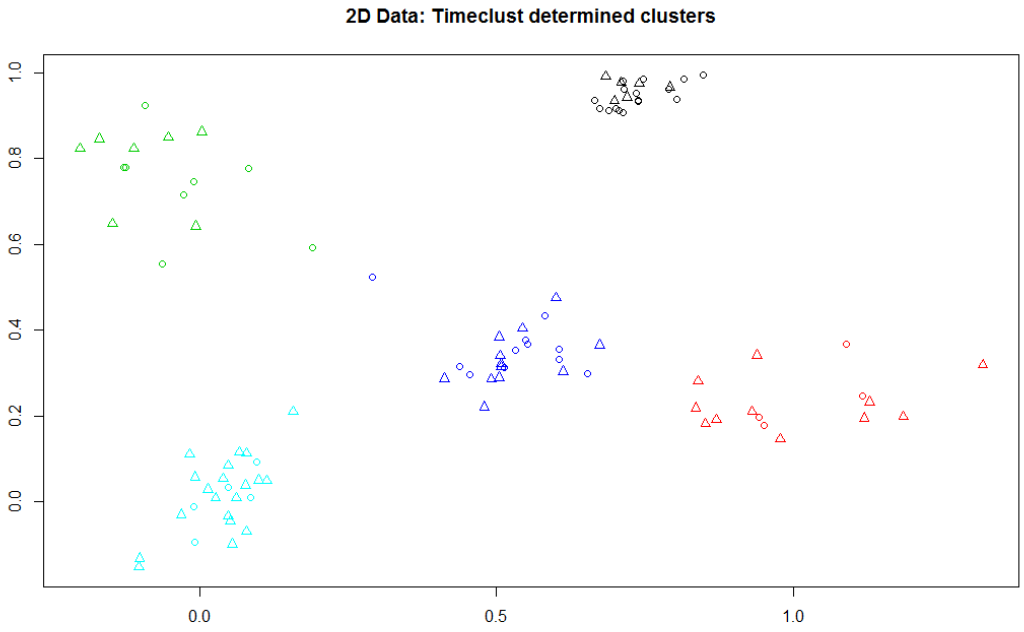

Figure 3.2: 2D Data clustered with `timeclust`, colour displaying cluster membership, shape representing cluster membership corresponding to timestamp clustering

Similarly, a dataset is generated with some values missing. The parameters are set identically to the previous case, with the addition of specifying `NA_val` to 10. Since this is a new simulation, the data will differ from previous case

```
test_run_2D_NAs <- create_data(dim=2, clust=5, size=100, overlap=0.01, NA_val=10,
      timestamp = TRUE, date_start = "2000/01/01", date_end = "2010/01/01")
```

The clustering algorithm is similar to the previous example, with the difference of pointing at the correct dataset and `time_method` being set to `"kmeans"`. Note that the methods have been deliberately set to k-means to demonstrate that the function will identify missing values and thus will switch to k-POD clustering.

The resulting adjusted Rand index can be seen below. Furthermore, the Table 3.2 shows the assignment of the points to clusters compared to their origin, and Figures 3.3 and 3.4 show the graphical representation of the origins and clustering. The table shows that 6 data points were assigned incorrectly. The mis-assigned points may contain the missing values, therefore they may be missing from the plots. However, the adjusted Rand index is close to 1 and vast majority of points were assigned to the correct cluster. The performance is thus only slightly hindered by the missing values.

```
     Rand
0.9377778
```

	1	2	3	4	5
1	12	0	0	0	1
2	0	26	0	0	2
3	1	0	0	0	23
4	2	0	14	0	2
5	0	0	0	17	0

Table 3.2: Assignment of incomplete data to clusters vs. their true cluster membership based on the origin distribution

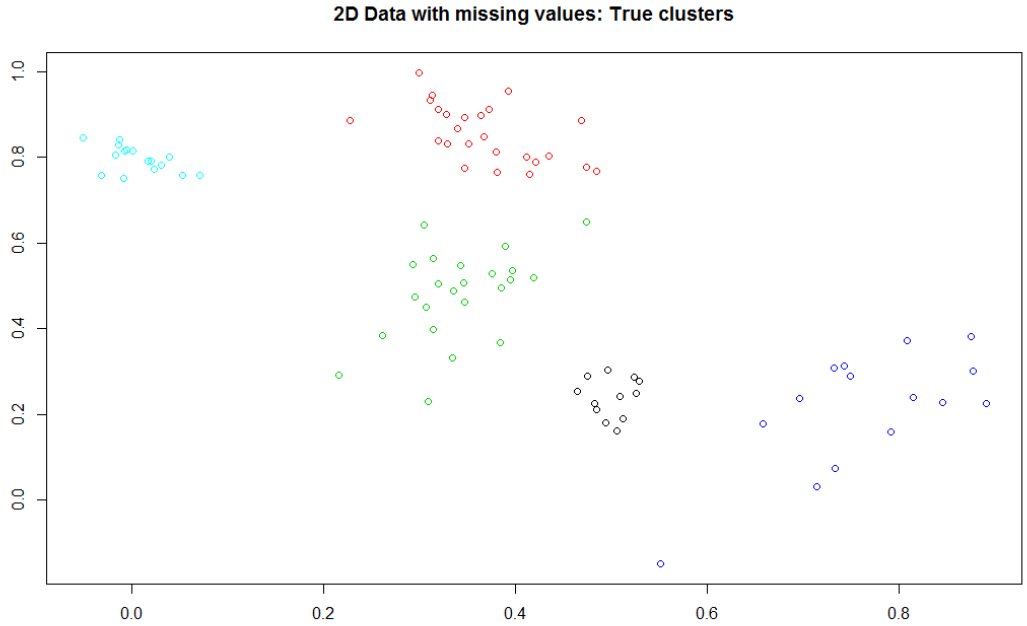

Figure 3.3: 2D Data with missing entries, colour displaying true cluster membership

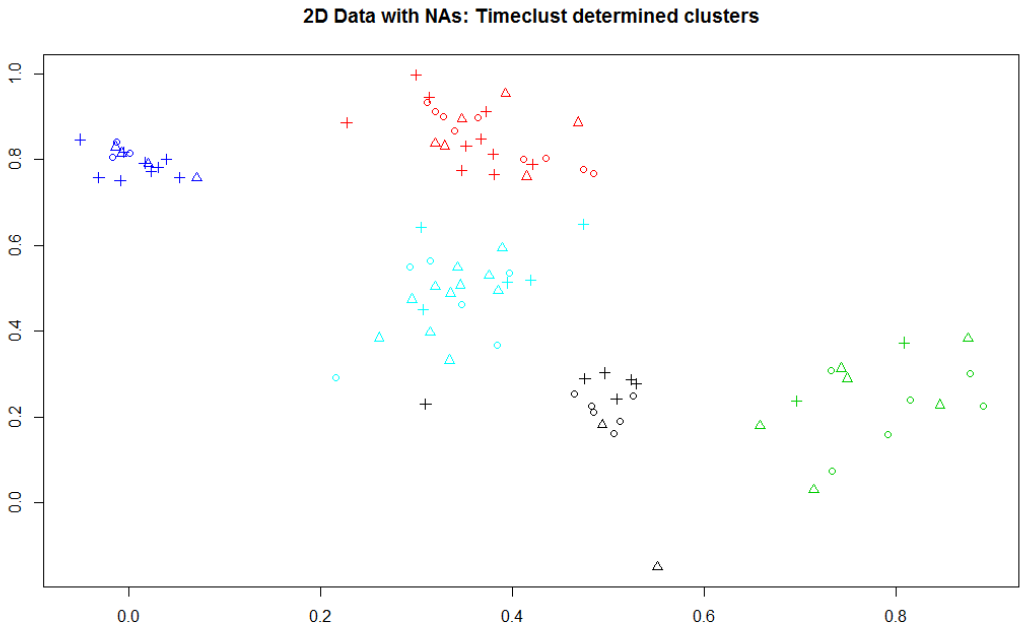

Figure 3.4: 2D Data containing missing values clustered with `timeclust`, colour displaying cluster membership, shape representing cluster membership corresponding to timestamp clustering

Finally, for demonstration purposes, the `timeclust` function is displayed with the parameter `timefirst` set to `TRUE`. No evaluation of the performance will be used, as clustering according to the timestamps is not meaningful given the way the dataset is constructed. The dataset used is the complete dataset from the first example. The timestamp data is split into 3 clusters with hierarchical clustering, and within each of these 3 further clusters are determined with k-means based on the numerical data. Figure 3.5 shows the resulting clustering using the following code:

```
test_clust_2d_time <- timeclust(clust_data = test_2d[,2:3], time_data = test_2d[,1],
    clust_method = "kmeans", k=3, time_method = "hclust", dist_m_t = "euclidean",
                hclust_m_t = "complete", k_t = 3, timefirst = TRUE))
```

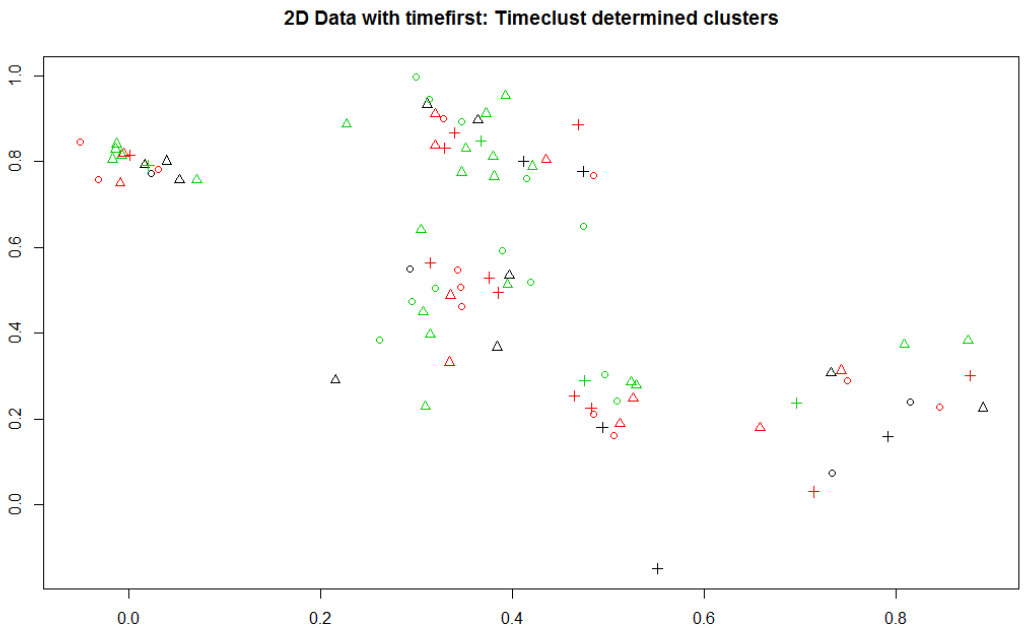

Figure 3.5: 2D Data clustered by timestamps first, colour displaying cluster assignment by timestamp data and label shape representing in-cluster grouping according to numerical data

3.2 Varying Data - Performance with Increasing Dimensions & Overlap

To test the performance of the `timeclust` function using all 3 available algorithms, a loop was set up to gradually increase the dimensions of the data (with other parameters fixed), performing clustering with each of the 3 methods at each iteration. The data will come from 5 distributions with overlap set to 0.1 and will contain 200 observations. The clustering will be performed on numerical data first to provide meaningful results. For hierarchical clustering, the methods remain default, i.e. the distance used is euclidean and the linkage is complete.

Each iteration of the loop increased the number of dimensions in the data simulation function by 1. The loop controlling the number of dimensions starts at 2 and closes at 30 dimension. A nested loop repeats the clustering 30 times for the same dimensions of the data, to obtain a more robust measure of the adjusted Rand index. In each iteration the data is used to determine the adjusted Rand index by comparing the true cluster membership with the resulting assignment (according to numerical data) from k-means, hierarchical clustering, and k-pod. The 30 measurements for each method are then averaged to provide a mean adjusted Rand index value for the given dimension of the data. The evolution of the mean adjusted Rand index values for each of the methods can be seen in Figure 3.6 below.

The performance of k-means and k-POD peaks at 4 dimensions, and at 3 dimensions for hierarchical clustering. As expected, the performance worsens as the number of dimensions increases. In comparison, the performance of the three algorithms appears to be comparable.

Similarly, the test is performed for increasing value of overlap. All 3 algorithms are tested again, with identical averaging of the adjusted Rand index across 30 iterations. The data generated consists of 5 numerical dimensions and has 200 observations. The initial value of overlap is 0.1. The outer loop will

iterate 10 times, increasing the overlap by 0.05 at each iteration. The maximum value of overlap reached thus is 0.55, as higher values resulted in long simulation times.

Figure 3.7 below shows the evolution of mean adjusted Rand index with increasing overlap of origin distributions.

The mean adjusted Rand index drops as the overlap increases. This behaviour is intuitive, as it becomes increasingly more difficult for the algorithms to determine the membership of the data points that are well mixed. The performance appears to decay similarly for all three algorithms, and appears to follow a pattern similar to exponential decay.

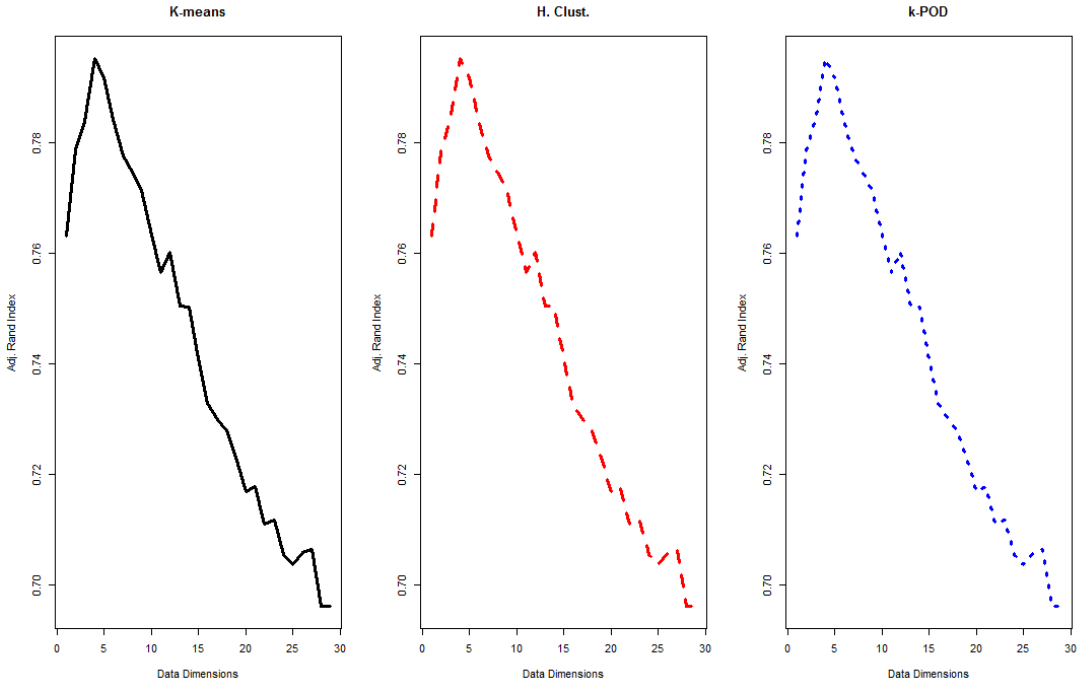

Figure 3.6: Evolution of mean adjusted Rand Index according to the number of data dimensions for each of the 3 algorithms available in `timeclust` function

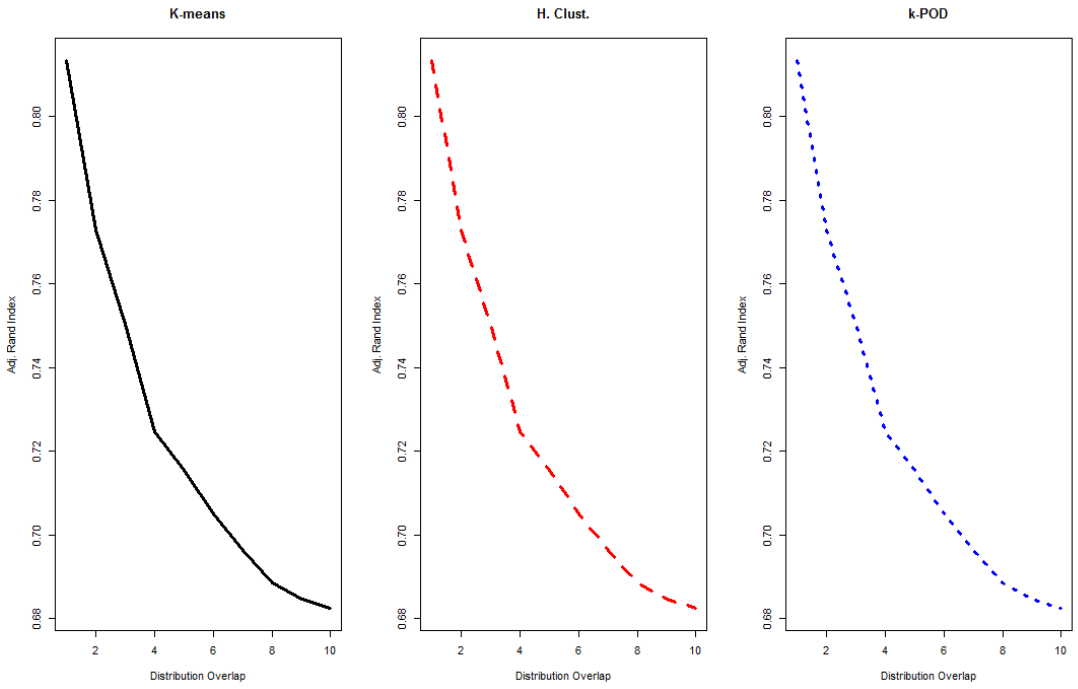

Figure 3.7: Evolution of mean adjusted Rand Index according to the value of distribution overlap for each of the 3 algorithms available in `timeclust` function

4 Conclusion

The `timeclust` function allows the user to perform clustering in 2 sequences of clustering algorithms, where one of the sequences clusters the numerical data and the other clusters timestamp data. The second clustering will happen in-group of the clusters from the first step. As it was demonstrated, the algorithms available - k-means, hierarchical clustering and k-POD - behave intuitively with increasing dimensions of the data and with increasing overlap of the origin distributions. Since the function is built by using the built-in algorithms, it is expected that the function will operate as effectively as the algorithms would on their own.

4.1 Limitations & Further Work

The limitations of the `timeclust` function stem from the common limitations related to clustering problems. The user must still choose the desired number of clusters based on some assumptions. The k-means algorithm used could be expanded or replaced by a better algorithm that does not induce randomness due to initialisation - for example k-means++.

Further work should consider the further implications of timestamp data - these could work as an additional source of similarity/distance to the numerical data in situations, where only one clustering algorithm would be run. Furthermore, the algorithm could be expanded to operate on dynamic data sets, where timestamped observations are received in real time. Such application would require a method for evaluating the appropriate number of cluster and algorithms employed, to ensure adaptability of the function. Furthermore, the choice of methods for hierarchical clustering is also dependent on the user. Different contexts may require different methods, thus the adaptivity of the function should also consider these options when clustering dynamic data.

Bibliography

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, pages –. Springer Science+Business Media, LLC, Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA.

Chi, J. T., Chi, E. C., and Baraniuk, R. G. (2016). k-POD - A Method for k-Means Clustering of Missing Data.

Melnykov, V., Chen, W.-C., and Maitra, R. (2016). MixSim: An R Package for Simulating Data to Study Performance of Clustering Algorithms. *Journal of Statistical Software*.

Murphy, K. P. (2012). *Machine Learning - A Probabilistic Perspective*, chapter 1.3, 25.1, pages 9–16, 875–906. The MIT Press, Cambridge, Massachusetts.