Introduction to time-series clustering in child growth data

# Introduction

Childhood undernutrition and obesity, and other diet-related noncommunicable diseases, although opposites co-exist within populations and across the lifecourse. This is called the “burden of malnutrition”[[1]](#footnote-1). Malnutrition, no matter its effect, remains one of the most important health risks even today. The direct relationship between diet and growth is known and established, given that malnutrition can significantly impede child growth. The ability to study a child’s growth and quickly detect problems is a prerequisite in our ability to rectify these problems by correcting the child’s diet. Based on its growth, a child can be characterized normal (its weight and height grow in expected rates), wasted (the child is undernourished with stagnant or lower than normal development) or overweight (the ratio of weight to height is above normal). Using these patterns, we can detect or even predict growth problems and recommend changes in diet. In this unit, you will learn how to extract growth patterns from a given dataset.

# Clustering

In machine learning, pattern identification is performed using clustering. Clustering is considered an unsupervised learning method, which implies that we don’t have a considerable understanding of the data set, we don’t have particular expectations about the clusters and, thus, we cannot aid the algorithm. An opposite example is classification, which is considered a supervised learning method. In this case, we have a dataset for which we already know the class of each data record. We use this dataset to *train* a model, with which we can then classify unlabeled data. In unsupervised clustering, there is no need for a training set.

In order to group data in clusters, we need to define the degree to which data records are similar to each other. Similarity can also be expressed by its opposite, *distance*. Many distance metrics have been defined to compare data, the most well-known is probably the Euclidean distance. In principle, the clustering algorithm will try to group data in clusters with the goal to minimize the distance between data within a cluster and maximize the distance between clusters.

There are various clustering algorithms and we should pick the one that fits our data and our needs with respect to the form of the clustering output. Two of the most popular types of clustering algorithms include hierarchical clustering (the hclust() function in R) and partitioning algorithms, like K-means (kmeans() function in R) or K-medoids (pam() function in R). Hierarchical algorithms organize clusters in a hierarchy as in a tree. As we go from the leaves (individual data points) and we move upwards as the distance increase, we start to group data points or merge clusters together. In the end, all data is grouped in a single cluster at the highest possible distance in the dataset. This method is called agglomerative as it progressively groups data together. In order to determine the desired clustering, we need to define the exact number of clusters we want or the distance threshold above which we stop to merge cluster and data points. In either case, what happens is that a cut-off value is determined based on which we “cut” the resulting hierarchy (called dendrogram). The final clusters are the final merges that happened just before our threshold. The definition of number of clusters (k) or cut-off threshold produces the same results, but from a cognitive perspective the motivation is different; by choosing a k we have an expectation about the number of clusters in our dataset; by choosing a threshold we impose the connectivity of our clusters.

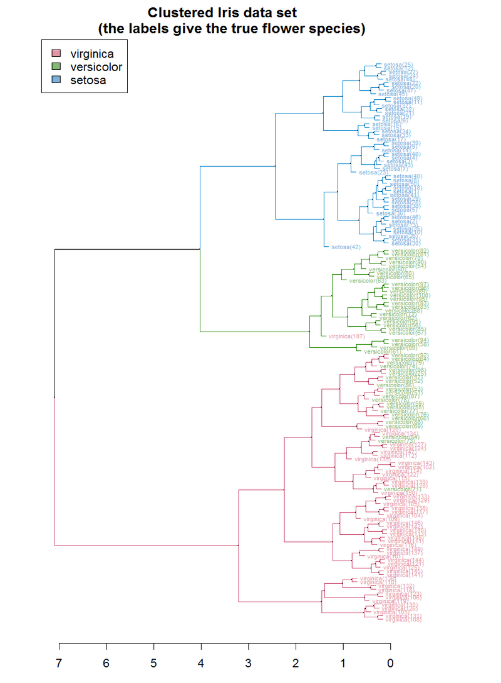


Figure . An example of a dendrogram as the result of a hierarchical clustering algorithm.

In partitioning algorithms, we need to provide only the number of clusters. The algorithm starts with a random or semi-random initial clustering around k centroids, and it progressively tries to improve the quality of the clusters. When the results converge (i.e., there are no new and more cohesive clusters) the result is return. The difference between K-means and K-medoids is that the first considers the “average” data point as the centroid cluster. This point does not necessarily correspond to a real data point from the given data set. On the other side, K-medoids uses real data points as centroids and it chooses the ones that are closest to the (possibly non-existent) average data point. One can claim that the centroids are the representatives of the clusters, i.e., if we were to describe the points of a cluster collectively, we would use the centroid.

Other clustering algorithms include density-based, like DBSCAN, where the algorithms identify dense areas of data, and fuzzy clustering, like Fuzzy C-means, where clusters can overlap with each other.

Task:

1. We are going to explore clustering in R on a well-known dataset and we will build and compare some clustering models.
2. Load the iris dataset from the “datasets” package.
3. You will notice that the dataset contains a column “Species”. This is what we are trying to find, the *class*. Since clustering is an unsupervised learning method, we can exclude this column for now. But keep it handy since we can used for evaluation purposes.

require("datasets")

data("iris") # load Iris Dataset

str(iris) #view structure of dataset

iris.new<- iris[,c(1,2,3,4)]

iris.class<- iris[,"Species"]

head(iris.new)

1. We can now apply our K-means algorithm to our dataset. We know from the description of the data set that there three species of iris. So, we can set the number of clusters to 3.

#apply k-means algorithm with no. of centroids(k)=3

result<- kmeans(iris.new,3)

# gives no. of records in each cluster

result$size

# gives value of cluster center datapoint value(3 centers for k=3)

result$centers

#gives cluster vector showing the cluster where each record falls

result$cluster

1. We can evaluate the results visually by plotting our data and coloring it according to the cluster it belongs. Since we have a multidimensional dataset, it is good to plot the data in different combinations of features to see what pair of dimensions can better explain our datasets in terms of clusters.

par(mfrow=c(2,2), mar=c(5,4,2,2))

# Plot to see how Sepal.Length and Sepal.Width data points have been distributed in clusters

plot(iris.new[c(1,2)], col=result$cluster)

# Plot to see how Sepal.Length and Sepal.Width data points have been distributed originally as per "class" attribute in dataset

plot(iris.new[c(1,2)], col=iris.class)

# Plot to see how Petal.Length and Petal.Width data points have been distributed in clusters

plot(iris.new[c(3,4)], col=result$cluster)

plot(iris.new[c(3,4)], col=iris.class)

1. Since we already have the classes, we can confirm the quality of our output. As we can see by following table, we have 133 true positives over 150 samples, which means about 88% accuracy.

table(result$cluster,iris.class)

## iris.class

## setosa versicolor virginica

## 1 0 47 14

## 2 0 3 36

## 3 50 0 0

1. We can repeat the clustering for a K-medoids algorithm. In R, we can use the pam() function.

pam.res<-pam(iris.new, 3)

# Let’s see an overview of the clustering object

pam.res

# Let’s see the resulting clustering

pam.res$clustering

# Let’s see the 3 medoids

pam.res$medoids

# Let’s see some more information about the clustering.

pam.res$clusinfo

# Let’s check the quality of the clusters

pam.res$silinfo

1. We can visualize the results of pam with clusplot.

clusplot(pam.res, color=TRUE)

1. However, we can use another package “factoextra”[[2]](#footnote-2) for additional plots and visualizations

require(factoextra)

# The clustering plot is similar to the one produced by clusplot()  
fviz\_cluster(pam.res)

# The next visualization can help us find the number of clusters.

# As the within sum of squares drop, the best k is that will not drop WSS  
# much more.

fviz\_nbclust(iris.new, pam, method=”wss”)

# The next visualization plots the distance between the data points.

# First, we need to produce a distance matrix for the iris data.

require(ecodist)

iris.diss<-distance(iris.new, method=”Euclidean”)

fviz\_diss(iris.diss)

# Finally, we can visualize the silhouette information of the clustering

$ to assess its quality.

fviz\_silhouette(silhouette(pam.res))

1. Besides partitioning algorithms, we can also apply hierarchical algorithms. These algorithms require a distance metric instead of the data set directly.

# Execute the hierarchical clustering algorithm...

hc <- hclust(iris.diss)

# We can visualize the results in a dendrogram using a simple plot...

plot(hc, hang=-1)

# ...but it would be nice to have some colour, especially if we know

# the number of clusters.

fviz\_dend(hc, k=3)

# Since we know the number of clusters, we can ask R to cut the

# tree for us.

hc\_3 <- cutree(hc, 3)

You can find more information and instructions on clustering with R here[[3]](#footnote-3) and here[[4]](#footnote-4).

# Growth trajectories and Time-series

Child growth is captured by longitudinal data; we measure a child’s height and weight in frequent intervals from birth to puberty. Every child is then characterized by a growth curve or *trajectory.* In other domains, growth curves are also known as *time-series*. Although point-data clustering is a known and well-defined method, the same cannot be said for time-series clustering. The greatest challenge is how to determine if two trajectories are similar or not. Should we compare them based on their y or x axis distance? Should we compare them based on their shape? In fact, the answer to these questions depends once again on the nature of our data and on our goals. Although this answers the why, it does not answer the how. How can we determine the distance between two trajectories? Is it the average distance between their individual points? Are all points equally important? What happens when the two trajectories have different number of points or measurements at different points in time?

These questions can be answered by the R packages ‘TSclust’[[5]](#footnote-5) and ‘TSrepr’[[6]](#footnote-6). ‘TSrepr’ contains functions that help us *represent* time-series data. Representation can mean one of many things including dimensionality reduction, interpolation, feature extraction and so on. Representation methods can help us treat a trajectory as a single data point, so that we can calculate distances between points and cluster them together. Methods like dimensionality reduction can help us align different trajectories in case they have different measurements or of varying length. The package ‘TSclust’ contains a great number of distance metrics specifically for time-series data. The collection covers many different scenarios. It contains distances that compare the trajectories point by point, that can work with trajectories of different lengths or missing measurements, that can measure the distance based on an underlying model and so on. Besides the distance metrics, the package contains a few functions to validate the results of a clustering given a ground truth. Curiously enough the package does not contain any clustering algorithms. This is because once the trajectories are represented and a distance is measured between them, any clustering algorithm can be applied. Therefore, the package relies on existing algorithms that can accept a distance matrix as input (which is the case for hclust(), kmeans() and pam() in R).

1. https://www.who.int/nutrition/double-burden-malnutrition/en/ [↑](#footnote-ref-1)
2. https://cran.r-project.org/web/packages/factoextra/factoextra.pdf [↑](#footnote-ref-2)
3. http://www.sthda.com/english/wiki/print.php?id=236 [↑](#footnote-ref-3)
4. http://rpubs.com/Nitika/kmeans\_Iris [↑](#footnote-ref-4)
5. https://cran.r-project.org/web/packages/TSclust/TSclust.pdf [↑](#footnote-ref-5)
6. https://cran.r-project.org/web/packages/TSrepr/TSrepr.pdf [↑](#footnote-ref-6)