3.3 Grouping by Similarities

Ming Lu

2022-04-02

{r setup, include=FALSE} knitr::opts\_chunk$set(echo = TRUE)

In any given stock exchange market all business with shares outstanding are classified by industrial sectors (or industry) such as banking, technology, automotive, energy, and so on, and according to the business primary activities. In this, and other forms of classifying business and their stocks (by the nature of their share holders, by income statements, by revenues, etc.), there is an underlying idea of similarity for forming groups, or dissimilarity for separating elements, that goes beyond the straightforward structural dependencies among prices, or returns, or other statistics, although it is expected that being similar under any of these more general (and often subjective) criteria implies some form of structural dependency. For example, investors generally expect that the prices of stocks from the same market and industry should eventually move together. To investigate these broader forms of similarity classification of financial time series, the appropriate statistical technique to call for is cluster analysis

Clustering is an unsupervised classification method which groups elements from a dataset under some similarity relation, in such a way that elements in the same group are closely similar while any two elements in different groups are far apart according to the stated similarity criterion. The choice of similarity relation depends on the nature of the data and the statistical properties we would like to deduce from them. A way to mathematically realized the notions of “closely” and “far apart” is to endow the similarity relation with the characteristics of a distance measure. Hence it is the dissimilarity between two observations that is determined by computing the distance separating the observations, and the clustering procedures are designed so as to maximized inter-clusters distances (dissimilarity) and minimized the withina-cluster distances (similarity as minimal dissimilarity). In the following section we review the fundamentals of data clustering and some of the most commonly used methods. The intention is to give the reader the minimal background so that following afterwards more specialized literature on the subject can carry on research on clustering, in particular related to financial time series.

# 3.3.1 Basics of Data Clustering

Clustering algorithms divide the data into groups or clusters. According to how this group division is performed a clustering method belongs to one of two major categories: hierarchical or partitional. Hierarchical methods begin with a gross partition of the data, which can be either to consider all the data as one single cluster, or each data element as a separate cluster, and successively produce other partitions using the previously established clusters and applying a similarity criterion, until certain convergence conditions are met. The case where the method begins with each element being a cluster by itself, and then in successive steps these are join together to form larger clusters, is called agglomerative. The other case, which begins with all data in one cluster and makes successive divisions to get smaller clusters, is called divisive. Partitional methods begin with a predetermined number of clusters (chosen arbitrarily or with some heuristic), and applying a similarity criterion elements are moved from one cluster to another, until convergence conditions are met. As commented before, it is more sound to work with a distance measure to quantify the dissimilarity between pairs of data elements.

Formally, given data as a set of vectors in , where each component is a feature or attribute, a distance metric on this space of real vectors is a bivariate function , which satisfies the following properties: (1) , for all (positivity). (2) if, and only if, (reflexivity). (3) (symmetry). (4) , for all (triangle inequality).

On distance measures. The most commonly used distance measures for classifying data whose features are continuous belong to the family of norms:

for . Among these, special attention is given to , the Euclidean distance (which is more common to denote it by , i.e. without the subscript 2), and , known as the “Manhattan” distance (denoted ). A reason for the preference of norm for , and not higher is that for larger it is more severe the tendency of the largest-scale feature to dominate the others, a problem that might be alleviated by normalization of the features (cf. Jain et al. 1999).

Example 3.2 To compare two stocks, and , it is usual to take as their vectors of features their respective time series of returns observed through some period of time, and . Each observed return value is a feature of the stock, and we can apply some norm to quantify the dissimilarity among these stocks’ features; for example, use the Euclidean distance

In this case one is measuring the spread among the return history of the pair of stocks.

In Aggarwal et al. (2001) it is shown that as increases the contrast between the -distances to different data points in high dimensional space is almost null. Hence, for classifying data in high dimensional space with -norms, and in particular classifying financial time series over long periods of time, seems preferable to use lower values of . This is a form of the curse of dimensionality suffered by the metric. Thus, to cope with this problem, the Manhattan distance should be preferable over Euclidean; Euclidean preferable over -norm, and so on. As a step further Aggarwal et al. (2001) propose to use a fractional -norm, where is taken as a fraction smaller than 1 , and show that this metric is indeed better at displaying different degrees of proximity among objects of high dimension. Also, the and norms are sensitive to noise and outliers present in the time series, so here too some form of normalization or filtering of the data should be needed.

Another family of distance measures of interest are those based on linear correlation. Note that the correlation coefficient is not by itself a metric, so it must be composed with other functions to turn it into a metric. The following two compositions give correlation-based distance metrics:

and for ,

However, as it has been noted in Hastie et al. (2009, , if observations are normalized then ; so that clustering based on linear correlation is equivalent to that based on squared Euclidean distance. As a consequence these distances based on correlation suffer from the same curse of dimension for norm mentioned above.

There are many others distance measures designed for quantifying similarity among objects of various nature: the Mahalanobis distance which corrects data with linearly correlated features, the maximum norm and the cosine distance are other measures for analyzing similarity of numerical data; for comparing sets and strings there are the Hamming distance, the Jaccard dissimilarity, Lempel-Ziv and information distances, and many others. For an encyclopedic account of distances see Deza and Deza (2009), but to keep within the context of statistical classification compare with Hastie et al. (2009).

# 3.3.2 Clustering Methods

The general goal of clustering can be formally stated as the following decision problem:

Given a finite set of data points, a dissimilarity or distance function , an integer and an objective function defined for all -partitions of , to find a -partition with minimum objective function value, i.e., a partition of such that is a -partition of Depending on the dimension of the data elements (i.e., the number of features we want to compare), the number of clusters, and for various objective functions , this problem is, more often than not, NP-hard (see, e.g., Brucker 1977; Gonzalez 1985; Garey and Johnson 1979). Hence, clustering algorithms are build with the goal of optimizing the underlying objective function which is usually a measure of dissimilarity between clusters, and solutions are often approximations, not the best possible.

The computational complexity of the clustering problem has motivated a lot of research on clustering techniques. We shall limit our presentation to two of the most popular methods, the agglomerative hierarchical clustering algorithm and the k-means partitional algorithm, and then show the equivalence of the clustering problem to the graph partition problem. This important link allows to export a broad box of graph combinatorial tools to aid for the solution of data clustering problems.

## 3.3.2.1 Hierarchical Clustering

In hierarchical clustering algorithms the dissimilarity between clusters is quantified by defining a distance between pairs of clusters as a positive real-valued function of the distance between elements in the clusters. Then, besides being agglomerative or divisive, we get further variants of the hierarchical clustering algorithms by changing the definition of the inter-clusters distance (which by being the objective function to optimize, it has an influence on the quality and precision of the resulting clusters). The most popular variants of inter-cluster distances are the single-link, completelink, average-link and Ward distance (also known as minimum variance), defined as follows. Given a set of data points in and a distance metric, the distance between two clusters, and (subsets of ), is denoted . Then is - single-link if ; - complete-link if ; - average-link if , where is the cardinality of . - Ward (or minimum variance) if , where as before, is the centroid of , and is the squared Euclidean norm (Eq. (3.14)), which is the underlying distance between elements. 5 The centroid of is defined as the point in such that for all in .

The agglomerative hierarchical clustering algorithm is a bottom-up procedure beginning at level 0 with each element being its own cluster, and recursively merging a pair of clusters from previous level into a single cluster subject to minimizing the inter-cluster distance, as given by some form of as above, or other. The algorithm stops when all data elements are in one cluster, and this marks the top level. The details of this clustering heuristic are presented as Algorithm 3.1.

The algorithm generates a nested hierarchy of partitions , which is best visualized as a tree. This tree is obtained by representing every merge done in step 4 by an edge drawn between the pair of clusters. In fact, many implementations of the agglomerative hierarchical clustering algorithm does exactly that and outputs such a tree, called dendrogram. Then, a specific partition is chosen by the user by cutting the tree at a desired level , which corresponds to choosing the level partition . A standard rule of thumb to compute this cut is to take the mean of the interclusters distances associated with the clustering method, and computed throughout the algorithm. Finally, note that the time complexity of this algorithm is of the order , where is the number of data elements to classify.

R Example 3.5 In this R Lab we show how to do an agglomerative hierarchical clustering of 34 stocks belonging to the main Spanish market index IBEX. For each stock we consider its series of daily returns, observed from 1 Dec 2008 to 1 Feb 2009 , which amounts to 40 observations; each series of returns is a column in a table labelled Ibex0809. Agglomerative hierarchical clustering analysis can be done in with the function hclust (d, method, …) from the package stats. The parameter is a dissimilarity structure produced by function dist (a distance measure), and method can be any of the four agglomerative methods. We experiment with all four methods and use as our dissimilarity measure the correlation-based distance from Eq. (3.15), namely . Enter in your console the following instructions

## 3.3.2.2 Partitional Clustering

3.3.2.2 Partitional Clustering

sum of squared errors, and under this criterion the partitional clustering problem amounts to: For a given set of data points and a given integer , find a set of centers so as to minimized the sum of squared errors , where is the cluster whose center is . Finding an optimal global solution to this problem is NP-hard (Drineas et al. 2004); hence we have to content ourselves with heuristics giving approximate solutions. The -means algorithm (Algorithm 3.2) is the simplest and most popular method for finding solutions to the partitional clustering problem based on squared error. It begins with a random selection of centers, and builds neighbors (clusters) around these centers by assigning points that minimizes the sum of squared distances. It updates the centers and repeats the clustering process until the squared error ceases to decrease significantly, or a preassigned number of iterations is reached.

The popularity of -means is due to its simplicity and its fast performance. Strictly speaking, the time complexity of -means is , where is the number of data elements, is the number of clusters and is the number of iterations. However, and are usually fixed in advance, so the algorithm has in practice linear time complexity .

The major disadvantage of -means is that it is sensible to the selection of the initial partition, and so often converge to a local minimum solution.

## 3.3.2.3 Graph Based Clustering

The problem of clustering any data set can be seen as a problem of partitioning a weighted graph in some optimal way. Given data set and a similarity relation on (i.e. , and is not necessarily a distance metric), define the similarity weighted graph , whose vertex set is and edge set is given by , and the edge has weight . Then the problem of clustering becomes the graph partition problem: To find a partition of such that edges between two different parts have low weight (hence, elements in different clusters are dissimilar from each other), and edges within the same part have high weight (i.e., elements within the same cluster are similar to each other). As an optimization problem this graph partition problem can be stated in terms of a very simple objective function, defined for all partitions of . Given a partition of , the of this partition is defined as

where . Now, the graph partition problem can be formalized as the Min-cut problem:

Min-cut: Given a graph , to find a partition of which minimizes .

This problem can be solve efficiently (i.e., deterministically in polynomial time) for , since in this case it is equivalent to the maximum network flow problem by the max-flow min-cut theorem (Papadimitriou and Steiglitz 1998, Chap. 6). For you may get by with cleverly iterating the algorithm for the case in the different parts. However, in practice, these network flow solutions of Min-cut often give clusters with only one element and one (or very few) clusters containing the

rest of the elements. To prevent from these imbalanced solutions one can sharpen the objective function with some form of normalization that forces the clusters to take in more than one element. One possible such normalization is to consider the objective function

Note that the minimum of the sum is attained when all are equal. Thus, with ut as objective function, solutions to the Min-cut problem consist of balanced clusters in terms of number of elements. The downside of this approach is that it turns the Min-cut into an NP-hard problem. In general any attempt to balance the sizes of the clustering solutions will make the graph partition a harder problem (see Wagner and Wagner 1993).

# 3.3.3 Clustering Validation and a Summary of Clustering Analysis

The most common and used empirical form of evaluating a clustering method is to create some artificial data set and for which one knows or forces to have a clustering fitting our desired objectives, i.e. create a controlled experiment. Then test the clustering method on this artificial data and see if it produces a solution identical to the already known clustering. For time series data the artificial set is usually obtained by doing simulations with a time series model, as an ARMA or GARCH, to be studied in Chap. 4. This is the type of evaluation criteria where we know before hand some ground truth clustering, and we just use some cluster similarity measure to quantify the coincidence of the solution resulting from our clustering method with this ground truth. This can be formally done as follows. Let C = {C1,…,Ck } be the set of clusters obtained by some clustering algorithm whose performance we want to evaluate against a previously known true clustering V = {V1,…, Vk }. Then a natural measure of evaluation is to calculate the proportion of coincidence , or similarity measure defined as

where . Other measures for comparing clusters are refinements of Eq. (3.19).

On the other hand, if we do not know (or are unable to build) a true sample of our clustering objective, then the evaluation criteria is limited to comparing the solutions obtained by the method, with the modeling assumptions; that is, to assess somehow if

the lowest intra-cluster and highest inter-cluster distances are attained. A typical way of doing this evaluation is by setting the intra- and inter-cluster distances as functions of the number of cluster, and so focus on determining the appropriate number of clusters. The gap statistic is one of many goodness of clustering measures based on estimating (see Hastie et al. $2009, \S 14.3 .11$ and references therein). The gap statistic is programmed in by the function clusGap from the package cluster. Other model selection criteria for determining are listed in the survey by Liao . To recap this section, let us summarize the four stages of clustering analysis: (1) Decide on data representation, which include cleaning, normalization or weighting of the features, and a general preparation to accommodate for the algorithm to be used. (2) Define a proximity measure, which reflects the relation of similarity use to compare the data, and preferably is a distance metric. (3) Choose a clustering strategy. (4) Validate the results: usually by comparison to benchmarks, known or simulated solutions; performing the results of the clustering method on the data for which we know before hand the correct partition.

For financial time series, data representation refers to considering returns, or some other standardization of the series, or some model fitting the data, among other forms of representation. Cleaning refers to removing noise, missing data or outliers. On defining appropriate proximity measures we have already commented some advantages and disadvantages for norms, correlation based distances, and others. Clustering methods you will find that the most recurred ones are the agglomerative hierarchical clustering and -means; other clustering methods and techniques for validating results are discussed in (Jain et al. 1999; Jain and Dubes 1988; Hastie et al. 2009; Liao 2005).

# 3.3.4 Time Series Evolving Clusters Graph

Financial returns evolve in time, and consequently when trying to cluster financial instruments by similarities on their return behavior (or any of its moments), it would be desirable to have a representation of this dynamic. In this section we study a graphical tool for monitoring the temporal evolution of clusters of financial time series; that is, a representation of the clusters in movement. Through this graph some useful links between graph combinatorics and financial applications arise, and this allow us to use some known combinatorial methods from graph theory to produce sound answers for problems about financial markets. The general construction of this time evolving clusters graph follows. Temporal graph of clusters. Let be a set of financial time series and let be the time period selected for analysis. Make a partition of into successive

sub-periods of time , where and the length of each sub-period are arbitrarily chosen.

We have fixed a clustering algorithm and, for each , apply it to the fragments of time series in observed in sub period of time . Let be the collection of clusters (a clustering) obtained in each temporal interval , and let . Let , be the clusters in with at least two elements, and be the reunion of elements that were not placed in any cluster by the clustering algorithm acting on time segment , for . We define a directed graph with vertex set the collection of subsets

and weighted edge set

that is, we link a cluster in the time segment with a cluster in the next time segment as long as their intersection is non empty, and the cardinality of their intersection is the weight assigned to the link. The sets remain as isolated vertices. We call the Temporal Graph of Clusters (TGC) for the set of time series in the time segmentation .

Example As an example of interest of a TGC consider the 34 stocks of IBEX treated in R Example , represented by daily returns series observed from 1 June 2008 to 1 Aug 2009 . We divide this time period in seven successive bimonthly subperiods June 2008,1 Aug 2008], Aug 2008, 1 Oct 2008] [1 June 2009,1 Aug 2009]. Consider as clustering goal to find among these stocks those whose returns series have similar monotone behavior, so as to obtain potential candidates for a pairs trading strategy. 9 Therefore, for the dissimilarity metric it is reasonable to work with some measure based on Kendall’s correlation coefficient , since this reflects best the monotone relationship between random variables, and then apply the agglomerative hierarchical clustering with the complete or Ward method. However, since the order in which the data elements are selected has an influence in the resulting clustering from the hierarchical method, we shall consider the following proximity measure (based on ) that avoids this “order of selection problem”. For each pair of stocks and define

where is a set comprised of all stocks whose series of returns , on the same time span of the returns of , have (sample) correlation with a higher than a positive ; that is,

The is determined from the sample estimation given by Eq. (3.8) and depends on the sample size. For example, for 40 days period one should take , the asymptotic quantile of the distribution of , to ensure a significant correlation. The measure is the Jaccard distance, and the reader should verify that is a metric. One can see that if and are similar, in the sense that their respective groups and have almost the same members, then is close to 0 ; whilst if and greatly differ then is close to 1 . Note that is a stronger measure of similarity than taking direct point-wise distance based on correlation (as given by Eqs. (3.15) and (3.16)), since it says that two stocks are similar, not only if they are correlated but also correlated to almost all the same stocks.

Next, on each sub-period , we apply agglomerative hierarchical clustering with distance given by Eq. (3.22), where the decision of the partition is made by the standard procedure of cutting at mid level the dendrogram representation of the hierarchical clustering. After the cut, the non-clustered elements go into . The full TGC is shown in Fig. 3.5: the green boxes (boxes in first row from bottom) show the time periods for each return series; the pink boxes (boxes in second row from bottom) collect companies with correlations below our established threshold (hence, no conclusion can be drawn for these); and the blue boxes (all remaining boxes above second row from bottom) represent non singleton clusters, which also contain a label to indicate its position in the adjacency matrix representation of the graph; for example, cluster 12 is {ELE, IBLA}. The edges are weighted by the intersection measure.

Graph combinatorics applied to financial analytics. The TGC provides a framework for solving problems relevant to financial analytics with techniques proper to graph combinatorics. For example, by considering the cardinality of the intersection of the clusters as the weight for the edges in the TGC, we can seek to identify the most persistent or stable clusters of asset returns through time: these would be located on the paths of heaviest weight (considering the weight of a path as the sum of the weights of the edges that comprise it). We formalized this problem as follows.

Stable clusters: consist on finding the clusters (or sub-clusters) of asset returns that appear more frequently in consecutive intervals of time. On the TGC this amounts to finding the heaviest paths from the first time interval to the last. The problem of deciding the heaviest weighted path in a graph is in general NP-hard; however, on acyclic directed graphs polynomial time solutions are known using dynamic programming (Garey and Johnson 1979, problem ND29). We shall make use of such algorithmic methodology for producing a solution to this problem on . The relevance of this graph problem to financial analytics is of a large extent. For example, it helps detect those stocks that correlate through different but con-

tinuous time intervals, and hence become candidates for a pairs trading investment strategy. Another interesting graph combinatorial problem we can formulate on the TGC is Stock cover: consist on finding the smallest possible set of stocks returns that intersects every (non singleton) cluster in the TGC G . This is the Hitting Set problem (Garey and Johnson 1979, problem SP8), which is a form of the Vertex Cover problem on graphs, both NP-hard. Hence, the best we can aim for is an approximate solution. The interest of this computational query is to discover those stocks that essentially represent the market behavior through a sequence of time periods; thus, conforming a portfolio that covers the market. We present efficient algorithms to solve these problems.

Algorithm for the Stable Clusters problem. Given a positive integer and TGC , with (excluding the isolated vertices ), we want to find the first heaviest paths starting at any cluster with no ingoing edge.

Let be the sequence of time segments which partition the total time period . We view the edges in as directed going from time segment to time segment . Then we add an extra (dummy) cluster after time period , and which serves as a sink in the sense that every cluster that do not have an outgoing edge (i.e. is terminal) is connected by an edge of weight 1 to . Now, a brute force solution is to find all paths that end in (say by Breadth First Search), order them by weight and keep the heaviest. This, however, would incur in exponentially many comparisons, as we can have a worst case containing many paths, where is the maximum number of clusters per time segment and is the number of time segments. Instead we apply a bottom-up approach (aka dynamic programming) where at step , we look at time segment and define, for each cluster , a heap (i.e. a tree with nodes having a positive value or key) where the root, labelled , has key 0 , its descendent will be all heaps defined in previous time segments , for which there is an edge from to , and the keys of all leaves of these descendent heaps are updated by adding the weight of the edge , denoted Observe that the total weight of a path in the heap of root is the key of the leaf or initial node of such path in the heap. We order these leaves by key value and keep the paths for the leaves with highest key, removing the paths of the remaining leaves before proceeding to step of algorithm. By adding a sink to the graph, that is an extra dummy set , the algorithm will collect at the last and extra step all -first heaviest paths in the heap , as the paths from the leaves to the root. Algorithm summarizes this strategy. We use the following abbreviations for functions whose implementation we do not specify but are obvious: for a vertex keeps the key value of in a heap, which should be clear from context; for a heap , leaves is the set of leaves of heap ; for two heaps and , append , returns a heap made by drawing an edge from the root of heap to the root of heap .

One can see that for time segments and a maximum of many clusters per time segment, the algorithm has a worst-case running time bound of , which is much better than the naive solution explained at the beginning.

Algorithm for the Stock Cover problem. Given a TGC , with , we want to find a (minimum) set of stocks that intersects every (non singleton) cluster in . We apply the following greedy strategy: pick the stock that shows up more often in the collection of clusters ; then remove the clusters containing this stock (i.e. covered by the stock). Repeat with remaining clusters.

In the pseudo-code implementation of the above strategy (Algorithm 3.4) we define an incidence matrix for the set of clusters , where rows are labelled by the companies ticker and columns labelled by the clusters. represents the set of tickers (which is given as input). Then there is a 1 in entry and , if stock represented by ticker is in cluster , or a 0 otherwise. By we denote the dimension of matrix , which is the number of rows multiplied by the number of columns.

The running time of this algorithm is bounded by ; that is by . More importantly, the size of the solution is not too big with respect to the size of the full portfolio of companies involved; in fact, it is half the size of the original portfolio, as shown below. Proposition 3.1 . Proof Each cluster contains at least two stocks. The worst situation is that any stock appears in at most one cluster. The algorithm selects a stock (of highest weight) and removes the cluster where it belongs together with at least another stock that will not enter the solution set (otherwise its weight is higher than ’s and would have been selected before). Thus, at most one half of the stocks enters .

Thus, our algorithm guarantees a solution of size at most one half of the size of the set of companies П. On the other hand, it is important to note that we, and nobody up to today, can not guarantee a solution of minimum size with current computer technology, since the problem of finding the minimum size vertex cover (of which our stock cover problem is a replica) is in general NP-hard (Garey and Johnson 1979). The hard way to find this minimum cover, following our greedy strategy, would be to check all possible combinations of assets that appear more often in the TGC in the first round of the while loop in the algorithm (lines 4-11), call this maximum number of appearances , together with those that appear more often in the second round, say number of times, and so on until all sets are covered. This amounts to combinations, where is at most half of the number of assets, by Proposition 3.1. Finally, a given cover of a given size might not necessarily be unique

in its components. The composition of a cover depends on the order of selection of the elements in done by the algorithm. Example 3.4 Back to Example 3.3. On the TGC computed for the set of stocks from the principal index IBEX of the Madrid Stock Exchange (Fig.3.5) applying Algorithm with , we found the following three heaviest paths (the number on top of the arrow is the weight of the corresponding edge): The first path weights 66 and the second heaviest weights 59 and the third weights 55. We have highlighted in boldface the indices of the sets where the paths differ. On these heaviest paths we can find the subsets of companies with higher tendency to correlate through time (correlation in movement). Once we computed a heaviest path we can take the intersection of all the sets belonging to the path in order to detect those assets that tend to behave similarly, in terms of their series of returns, most of the time. Doing this on the first heaviest path we found that ACS, BBVA and SAN remain consistently together through the clustering; hence any two of these three assets make a good candidate for a pairs trading strategy. However, the two banks SAN and BBVA tend to have a stronger correlation at all sub-periods.

Applying the Stock Cover algorithm (Algorithm 3.4) we obtain the following cover: , IBLA, IBR, MAP, IDR, GAM

These eight companies represent a set of stocks that intersects all non isolated clusters in the TGC for IBEX through the period considered. It constitute a portfolio that would have replicated the behavior of the Spanish Exchange index throughout those fourteen months. Due to the computational hardness of the vertex cover problem, we can not guarantee in any feasible known way that this set of stocks is of minimum size. The best we can do is to repeatedly permute the order of the stocks ticker in the list and run the algorithm again, hoping to find in some of the possible exponentially many trials a better solution. We have been lucky with this brute force method and found another cover of smaller size (seven elements) by permuting to the first place in the list. , IBLA, ELE, GAM, IBR, MAP, IDR For the sake of completeness, one may add to the cover found by the algorithm those stocks that are not in any cluster, such is the case of GRF.

There are many other graph combinatorial problems that can resemble questions on managing financial portfolios, and their algorithmic solutions be of used in answering the financial problems. One more example is given in Note 3.5.10. We encourage the reader to explore this subject further.