Swiss Finance Institute

Master in Financial Engineering

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

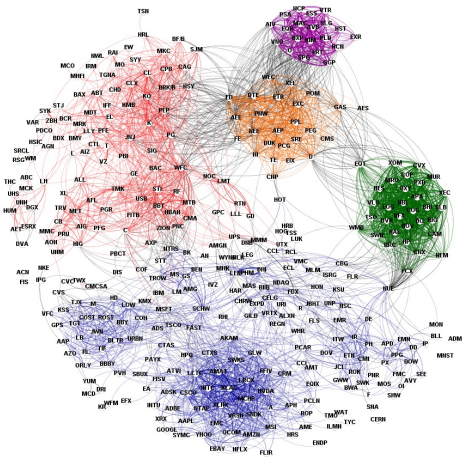
NAFORA

A Network Analysis of Stock Communities and Market States

Benjamin Gaudin

Financial Engineering Master Thesis

August 2016



Company : Nafora SA

FE supervisor: Prof. Semyon Malamud

Company supervisor: Jan Macháček

Abstract

We aim at implementing a trading strategy based on market states. Their definition in Marsili (2002) revealed predictability in the returns of stocks. To unveil these states, we analyze a large data set of daily returns, and use a clustering algorithm based on the Louvain method. Beforehand, we test its effectiveness by clustering stocks into communities. Along the way, we submit an improvement of the process of the algorithm, and a revision from previous work of the null hypothesis in the modularity for correlation matrices. We propose a track to adjust the definition of the modularity in order to obtain a more intuitive partition. We find that 1) assets in the same community behave similarly across states and 2) for the stocks and the period considered, the predictability is absent, making the market, in this view, efficient.

Acknowledgments

We would like to warmly thank Prof. Damien Challet of HEC for his recommendations that founded the subject of this thesis, as well as directing us to the main reference papers, and for his course 'Inefficient Markets' that gives excellent insights into the world of algorithmic trading.

We are very grateful to Dr. Matteo Marsili for his answers, and for sharing precious information and intuition.

We thank a lot Prof. Semyon Malamud of EPFL and Jan Macháček of Nafora for their valuable time spent supervising this thesis.

Thank you also to Elizabeth Silver, Foort Hamelink, and Gergely Bacsó for their careful proofreading.

Last but not least, we wish to express special thanks to everyone at Nafora, for the resources and freedom provided, for their support, generosity, and enlightening discussions, in particular Samer Darwiche, Jan Macháček, Franky Sleuyter, Quentin Bossard, Ahmed Trabelsi, Patrick Fodor, Gabriel Garcia, Yilin Hu, and Kanchan Mopari.

Contents

1 Introduction **6**

2 Literature Review and Data **8**

3 Algorithm **11**

3.1 Louvain method 11

3.2 A proposed improvement 14

4 Clustering of Stocks **16**

4.1 Significant values of the edges 16

4.1.1 Significant values for positively weighted networks 16

4.1.2 Significant values for correlation matrices 17

4.2 Market mode 24

4.3 Modularity for correlation matrices 25

4.4 Results 26

4.4.1 Extension of the modularity 28

5 Clustering of Days **32**

5.1 Measure of similarity 32

5.2 Results 35

6 Strategy Research **39**

6.1 Predictive power of the states 39

6.2 An intraday strategy 42

7 Conclusion **46**

Bibliography **47**

Appendices **51**

A Code of the Modified Louvain Method **52**

B Predictability Plots **58**

List of Figures

3.1 Network representation of a correlation matrix 13

4.1 Reduced effectiveness of the time parameter 21
4.2 Distribution of the eigenvalues of the empirical correlation matrix 22
4.3 Visualization of a dimension reduction 24
4.4 Layer representation of the steps of the algorithm 27
4.5 Network representation of the S&P500 constituents 31

5.1 Distribution of eigenvalues of the correlation matrix between days 35
5.2 Visualization of the market states 37
5.3 Exponential decay of the number of edges 38
5.4 Temporal distribution of the market states 38

6.1 Predictability of the states 40
6.2 Dynamics of portfolios 44

Chapter 1

Introduction

The last two years alone have seen the creation of as much as 90% of the data available in the world (IBM, 2016). This phenomenon, which has been developing over the past couple of decades, is called *Big data*, and it fuels the quantity of financial data accessible. This huge amount of data makes it essential to find techniques that are able to filter a few but highly relevant portions of information out of a continuously increasing noisy background. Subsequently, these can be used by quantitative traders in an attempt to leverage this information to construct their strategies. Technically, a trading strategy is a function that takes a subset of the currently available information as input and outputs positions on instruments to hold at each time. Successful trading strategies are based on quantitative relationships among entities. They are founded on a set of assumptions that the trader expects to hold in the future, when the strategy is running.

A major turn in the history of trading strategies has been the automatization of the orders sent to an exchange. This gave rise to a new way of managing money based on algorithms (McGowan, 2010). They allow computers to send orders without human intervention. In the execution of orders, computers can go much faster to profit from short-lived inefficiencies and are not subject to human failure. Algorithmic trading relies on the discovery of statistical relationships that emerge from data. These data can be anything from news, quarterly reports, annual reports, analyst forecasts, weather reports, volume traded, historical prices, to any other data sources that seem usable. The more data researchers have at their disposal, the more anomalies they can detect, and also

verify whether these anomalies are consistent over time. If this is the case, then an algorithmic strategy can be implemented. To name of few, these strategies can be based on mean-reversion, trending, pure arbitrage, market making, or pattern recognition.

In this work, we use a subset of the available data to define *market states*, which were originally suggested by Marsili (2002), who found that they contained predictability in price movements. Such predictability in a model is appealing in the purpose of designing a trading strategy. We define market states as the identification of statistically significant classifications of daily market-wide activity. Our strategy makes the simple assumption that similar patterns on a set of assets occur in different periods of time. Our goal is to find out whether this assumption holds with respect to the definition of the patterns we look at.

The main focus of this thesis lies in the technicality of constructing these states, which are groups of similar days. To achieve this, we need to form a similarity measure, and turn subsequently to cluster analysis to form those groups. For this purpose, we use the Louvain method (Blondel et al., 2008), which is an algorithm based on graph theory. It relies on the widely used modularity optimization (Newman, 2004), adapted here to correlation matrices (MacMahon and Garlaschelli, 2015).

In chapter 2, we discuss in more detail the kind of predictability that can be found from market states, based on the work of Marsili (2002), together with the different kinds of clustering models that currently exist, and the data we use for our analysis. In chapter 3, we describe the algorithm that we selected, the Louvain method, and propose an improvement. In chapter 4, we verify that the algorithm produces an intuitive partition of the data into groups by applying it to finding groups of similar stocks. We will see that we need to extract the significant value of the correlation between two stocks by constructing a null hypothesis based on random matrix theory (Laloux et al., 1999; Plerou et al., 2002). Along the way, we analyze the results and tackle the fundamentals of the theory. Based on these investigations, we submit novel revisions to previous works (MacMahon and Garlaschelli, 2015), as well as possible extensions of the modularity. In chapter 5, we apply our algorithm to days to detect the sought-after market states, and look for returns predictability from the model in chapter 6.

Chapter 2

Literature Review and Data

Marsili (2002) finds that assigning days to a few number of states contains some predictability in the next day returns. From a subset of size $N = 2000$ of the most actively traded stocks in the New York Stock Exchange, he finds that many of them have a predictability larger than noise. Knowing the state of today has much more predictive power on the returns of tomorrow compared to the returns in a day randomly picked in the future. This is interesting for a trading strategy: if we are able to predict tomorrow's return for many assets with an accuracy larger than that coming from random picks, then it is possible to profit from that, with a minimized risk since the portfolio takes positions on several stocks.

The states are defined by the daily returns, so for each day, a vector of size N is used to assign the day to a state. Marsili (2002) uses a clustering algorithm to group the days together as a way to reduce the dimensionality of the problem.

Data clustering provides an answer to the question: "Given a set of N objects, how to classify them into groups, so that objects belonging to the same group are more similar to each other than objects in other groups?" The algorithms used to solve such a task can be supervised, i.e. the structure of the data is predefined by the user, or unsupervised, in which case the cluster structure emerges directly from the data. They can tackle the problem either from a top-down approach, where the data set is split into smaller groups, or from a bottom up approach, where the objects are merged together. They come with a cost (or benefit) function, that they try to minimize (or maximize), together with a

measure of similarity or distance between objects.

There is a vast number of clustering algorithms, which comes from the fact that the very notion of 'cluster' is not precisely defined (Estivill-Castro, 2002). The only unequivocal definition that is agreed on is that a cluster is a group of data objects. The algorithm must subsequently be chosen with respect to the partition sought after, and the nature of the data.

If the clusters are already well separated in the space in which they are represented, centroid models can be used, like the k-means algorithm (Kanungo et al., 2002), which represents each cluster by a single mean vector. Distribution-based models can also give good results in this situation. If we can construct a sensible distance metric between objects, a hierarchical clustering algorithm, which connects objects step by step, can be suitable. Such a kind of algorithm was used in Marsili (2002). If the density of the data seems to be an important property to form differentiated clusters, then density models (Ester et al., 1996; Ankerst et al., 1999) may be relevant. An interesting group of algorithms focuses on detecting clusters in graphs, which can be applied to data that can be represented in such a way.

In this thesis, we choose one clustering algorithm that we describe in the next chapter. We will apply it to stocks so that we can judge from a qualitative point of view that the groups formed by the algorithm are legitimate. Then, we will apply it to days and detect market states. Since the number and the structure of groups are not parameters that we can naturally see for stocks, and especially for days, the algorithm has to be unsupervised. We also require that it works with a minimum number of parameters (so that the result relies as little as possible on arbitrary choices), and that it can handle positive together with negative measures.

The data set we use are the daily closing price series of the constituents of the S&P500 as of 2016-02-29 provided by Bloomberg. For our analysis of clustering stocks and days,

we take the period of 2003-01-01 to 2006-12-31 which contains $T = 1013$ days, to be sure of having enough data left for backtesting afterwards. We are aware that it is not the best practice to take the current components and look at their time series in the past. By doing that, we select only the stocks that will be successful enough to survive in the future. This is known as 'survivorship bias'. We presume however that for the purpose of testing our clustering method, this has limited effects. For the strategy implementation, we are more careful and take the most recent data to match the most recent S&P500 components.

From the 504 components given by the Bloomberg terminal, we select only those for which there are less than 40% of missing data. We end up with a set of $N = 416$ stocks. The price time series are adjusted for stock splits and dividends. Data points are taken as the prices of the last trade in a day. When a price is missing, we take the last available price. We compute the log-returns that we will simply call 'returns' thereafter, and apply a 95% winsorization. This means that we cap the outliers above the 97.5th percentile to the 97.5th percentile, and those below the 2.5th percentile to the 2.5th percentile. We do this operation in order to filter out potentially wrong data sent by Bloomberg, and to minimize the effect of outliers on estimators, like the sample correlation, that are sensitive to them.

Chapter 3

Algorithm

Our goal is to group together objects that are similar, whether they are stocks or days. We need a measure to quantify the quality of a given partition. Moreover, since comparing all possible combinations of N objects into $M \leq N$ groups is computationally intractable once one works with large sets of data, a heuristic algorithm [1] must be used.

To do so, we can represent our system (an ensemble of stocks or days) as a network, which is a collection of *nodes* joined in pairs by *edges*. The nodes are the elements of the system, while the edges are a measure of closeness, or similarity, between two nodes. A matrix $\mathbf{A}$ can be represented as a network, whose entry A_{ij} is the edge between the nodes i and j. The matrix $\mathbf{A}$ can be for example a correlation matrix (see Figure 3.1). The benefit of representing our system as such is that we are now able to use recent techniques used in graph theory.

We note $\mathcal{P} = \{P_1, \cdots, P_M\}$ the partition of the data into M communities. We assign to each of them a community identifier $\sigma(P_a) = a$. The partition can also be expressed with an N-dimensional vector $\vec{\sigma}$, whose component σ_i is equal to the identifier of the community to which the node i belongs to (node $i \in P_a \Leftrightarrow \sigma_i = \sigma(P_a)$).

3.1 Louvain method

A popular algorithm used to extract the community structure of large, positively weighted networks, is the Louvain algorithm (Blondel et al., 2008). It can also be used with negative weights as we will see later. It is an unsupervised, merging algorithm. It attempts to find

[1] A heuristic algorithm generally sacrifices the guarantee to find the best possible partition for the sake of speed, and in this case feasibility.

an optimal partition $\vec{\sigma}$ of the network, so that nodes in the same community are the most alike. The optimal partition is reached when a certain quantity, called *modularity* is maximized. The modularity is a measure of the quality of a particular partition of the network. It is expressed as:

$$Q(\vec{\sigma}) = \frac{1}{F} \sum_{i,j} B_{ij} \cdot \delta(\sigma_i, \sigma_j), \tag{3.1}$$

where B_{ij} is the strength of the link between the nodes i and j, and F is a scaling factor so that partitions of networks of different size can be compared on the same basis. $\delta(\sigma_i, \sigma_j)$ is the delta function, which is equal to 1 if $\sigma_i = \sigma_j$ or 0 if $\sigma_i \neq \sigma_j$. This ensures that only nodes in the same community contribute to the sum. The diagonal terms B_{ii} are added once, no matter what the partition is, while the off-diagonal terms $B_{ij}, j \neq i$ are added twice if the nodes i and j belong to the same community, or not added otherwise.

We should point out here that the values fed to the modularity Q are B_{ij}. They are not the original values A_{ij} of the edges in the network, but rather a measure of how significant these edges A_{ij} are. We see in the next chapter how to extract the significant value B_{ij} from A_{ij}.

The algorithm proceeds in two phases, as follows: initially, it puts each node in its own community. Then it moves the nodes one by one to the community they 'like the best'. Once a node finds the community it likes the best, it stays there, and the algorithm continues the process with another node. 'Like the best' means maximizing the local gain in modularity resulting from moving the node k from community P_a to community P_b:

$$\Delta Q^{k(P_a \to P_b)} = \frac{2}{F} \Big[\sum_{j \neq k} B_{kj} \cdot \delta\big(\sigma_j, \sigma(P_b)\big) - \sum_{i \neq k} B_{ki} \cdot \delta\big(\sigma_i, \sigma(P_a)\big) \Big], \tag{3.2}$$

where the factor 2 comes from the fact that we sum each link twice in the modularity. This process is done iteratively until no additional move is able to increase the modularity. At this stage, groups of nodes have been formed. This is the first phase.

In the second phase, the algorithm tries to merge the groups together, as it did for single nodes. The strength of the link between two groups P_a, P_b, is simply the sum of

the links between nodes in the two groups

$$B_{P_a,P_b} = \sum_{ij} B_{ij} \cdot \delta\big(\sigma_i, \sigma(P_a)\big) \cdot \delta\big(\sigma_j, \sigma(P_b)\big). \tag{3.3}$$

This second phase is repeated until a (local) optimal partition which has the (local) maximum modularity is found. Figure 3.1 gives an example of the process of the algorithm.

The computation of the modularity can either take into account the diagonal numbers (the self-loops) or not, with no change in the optimization of the moves. These numbers only add a constant term to the modularity and do not affect the maximization.

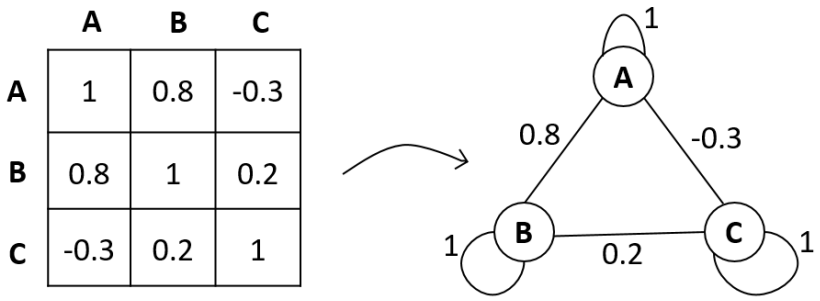


Figure 3.1: Network representation of a correlation matrix, with three objects A, B, C. The off-diagonal terms give the weights to the edges of the network. These are counted twice if the two nodes they link belong to the same community, or not counted at all if it is not the case. The diagonal terms are represented as self-loops. They are counted once, no matter the partition. Here is an illustration of the first phase of the algorithm. Initially, all nodes are in their own community. For the sake of simplicity, let $F = 1$ for now. The modularity is 3. A is sequentially moved to each of the two communities. The move that maximizes the modularity is kept. Moving A to B and A to C gives respectively modularities of 4.6 and 2.4, so A moves to the community of B. The algorithm then tries to move B, etc., until no move is able to increase the modularity.

The algorithm is greedy, in the sense that it makes the local optimal choice at each step with the hope of finding a global optimum. It is efficient, but might give slightly different partitions depending on the order it picks nodes to move. A solution is to randomize the order in which the nodes are picked and to repeat the algorithm several times to ultimately keep the partition with the highest modularity.

At this stage, we want to say a word about the performance of modularity maximization. Although it is widely used, an important issue is that modularity lacks a clear global maximum, and admits a large number of high-scoring partitions, close to

the optimum (Good et al., 2010). Another issue it faces is the resolution limit, which is the fact that modularity maximization often fails to detect communities smaller than a certain scale (Fortunato and Barthémy, 2007). The resulting partition may not coincide with the most intuitive one. While we will be able to circumvent this second issue in the following chapters by running an additional algorithm on top of the Louvain method, any results must be interpreted cautiously, having these limitations in mind.

3.2 A proposed improvement

After running the algorithm, we noticed an issue in the final community partition. Indeed, it can happen that in some communities the sum of the incident edges to a node would sum up to a negative value. A node for which the sum of its edges linked to the nodes in a community is negative cannot be inserted in this community, since this would decrease the modularity. This situation occurs only if this specific node was incorporated in the community before most of the other nodes.

Removing this node from the community would actually *increase* the modularity. However, the original Louvain algorithm forces the node to join a community already formed by other nodes. It is possible that moving it to any other community would decrease the modularity even further. That is why it stays in a community that would not accept it otherwise, if it were not already part of the community. This is what can make a node be misassigned.

The improvement of the algorithm we propose and implemented [2] is to create an extra empty community, so that nodes that do not fit in their original community anymore can be freed from it. This move will increase the modularity locally. Because the Louvain algorithm is a greedy algorithm, one can reasonably expect this local improvement to lead to the improvement of the modularity of the final partition. For every move, we allow the node to go back to a community where it would be the only member. In detail, if the

[2] The reader can find the code in Appendix A.

gain of only removing a node k from its community P_a is positive, i.e.

$$\Delta Q^{k(P_a \to \text{out})} = -\frac{2}{F} \sum_{i \neq k} B_{ki} \cdot \delta\big(\sigma_i, \sigma(P_a)\big) > 0, \tag{3.4}$$

then we also consider as possible the move of this node to a new, empty community. If the current partition $\mathcal{P}$ is formed by M communities, the new community σ_k of the node k is found with

$$\sigma_k = \begin{cases} \sigma(\hat{P}) & \text{if } \Delta_k(\hat{P}) > 0 \\ M+1 & \text{otherwise,} \end{cases} \tag{3.5}$$

where

$$\hat{P} = \arg\max_{P \in \mathcal{P}} \Delta_k(P) = \arg\max_{P \in \mathcal{P}} \sum_{i \neq k} B_{ki} \cdot \delta\big(\sigma_i, \sigma(P)\big), \tag{3.6}$$

where $M+1$ is the community identifier of a new community P_{M+1}.

Chapter 4

Clustering of Stocks

In this section, we discuss what values B_{ij} have to be passed to the modularity. Then, we test our algorithm by applying the clustering on stocks. We can thus verify whether the stocks in one community also share financial similarity.

4.1 Significant values of the edges

Detecting communities means grouping nodes that are statistically more related to each other than with the rest of the nodes. The link B_{ij} between the nodes i and j has to measure how significant their relation is. This measure can be either positive if the nodes are similar, or negative if they are different. Mathematically, it means that the edge A_{ij} of the original network has to be compared to the null hypothesis $\langle A_{ij} \rangle$, which is the value that is expected if the link between i and j comes purely from randomness. Therefore, the matrix $\mathbf{B}$ has entries equal to

$$B_{ij} = A_{ij} - \langle A_{ij} \rangle. \tag{4.1}$$

4.1.1 Significant values for positively weighted networks

The original Louvain algorithm (Blondel et al., 2008) considers large, positively weighted networks. These can be used to represent social networks, biological networks, citation networks, or the world wide web for example. In such networks, the degree k_i (the number of links) of each node i specifies the structure of the system. The relation A_{ij} between two nodes is discrete and is the number of links that associate the two nodes (usually 0

or 1, but it can take larger integer values). The null hypothesis is expressed as (Newman, 2004; Blondel et al., 2008)

$$\langle A_{ij} \rangle = \frac{k_i k_j}{F}, \tag{4.2}$$

where $F = \sum_{ij} A_{ij}$ is twice the number of links present in the network. $\langle A_{ij} \rangle$ is the probability that an edge exists between nodes i and j if we preserve the degrees of nodes in our network but connect them otherwise at random. The modularity is thus given by:

$$Q(\vec{\sigma}) = \frac{1}{F} \sum_{i,j} \Big[A_{ij} - \frac{k_i k_j}{F} \Big] \delta(\sigma_i, \sigma_j). \tag{4.3}$$

A weight B_{ij} will be positive if the number of links A_{ij} is larger than average (in other words, if the density $\frac{A_{ij}}{F}$ is larger than the average density $\frac{k_i k_j}{F^2}$), or will be negative if the links are more sparse than average. Therefore, the maximization of the modularity tends to put together nodes with a density of links larger than average, while it will pull apart nodes with density of links lower than average.

There exist packages of this algorithm in several programming languages like Matlab or Python. However, they cannot be used as such when we deal with correlation matrices, so we need to modify the algorithm. We will see why below.

4.1.2 Significant values for correlation matrices

In this thesis, we use the Pearson correlation to measure the similarity between two time series. The matrix $\mathbf{C}$ of pairwise correlations acts as the matrix of similarity $\mathbf{A}$. For the correlation to be a sound measure of similarity, the data must have linear dependencies and be temporally stationary, which does not generally hold for assets returns (Danielsson, 2011). This is particularly important for techniques like eigenvalue decomposition that we use below, because they work the best when these assumptions are met. Additionally, correlation is sensitive to the time resolution of the data (MacMahon and Garlaschelli, 2015; Borghesi et al., 2007). Moreover, it only considers the two first moments, while higher moments like skewness and kurtosis are neglected. These moments also are risk factors (Lempérière et al., 2014) that should preferably be taken into account. Nonetheless, this measure is still the most widely used. Improving it is an important and open problem, but this is beyond the scope of this thesis, so this is simply a warning of the limitation of the Pearson correlation for the reader to bear in mind.

Networks formed from correlation matrices must be handled differently than positively weighted networks, since their structure is fundamentally different. Indeed, a single node is related to all the other nodes by edges, which can take positive and negative values, ranging from -1 to +1. The degree of the nodes is not a relevant measure in this case anymore. The null hypothesis must be adjusted accordingly.

An empirical correlation matrix can be written as

$$\mathbf{C} = \frac{1}{T}\mathbf{X}^\top\mathbf{X}, \tag{4.4}$$

where $\mathbf{X}$ is a $T \times N$ matrix of normalized log-returns [1] consisting of N time series of length T, and where $^\top$ is the transpose sign. For financial assets, the distributions of the time series $\vec{x_i}$ (which are the columns of $\mathbf{X}$) are approximately Gaussian (Bachelier, 1900; Black and Scholes, 1973), with a mean of 0 and a standard deviation of 1 by construction. Therefore, they can be thought to be drawn from the same distribution, i.e. to be identically distributed. If these times series are independent, then they are also uncorrelated. Their empirical correlations, however, would not exactly equate to zero in this case: they will have a small, residual correlation. What we want is to quantify this residual correlation coming from randomness, which will constitute the null hypothesis.

An ingenious technique of quantifying this randomness component comes from random matrix theory (RMT). What RMT tells us is that, for a matrix $\mathbf{X_0}$ fulfilling the requirements of independent, identically distributed (i.i.d.) entries with mean 0 and standard deviation $\sigma < \infty$, the eigenvalues [2] of the matrix $\mathbf{C_0} = \frac{1}{T}\mathbf{X_0}^\top\mathbf{X_0}$ follow a specific distribution (in the limits $T, N \to \infty$ and $q = T/N \in (0, +\infty)$). This distribution is known as the

[1] Let $\vec{r_i} \equiv (r_i(1), r_i(2), \ldots, r_i(T))$ be the time series of log-returns of asset i. Let μ_i and σ_i be the usual sample mean and standard deviation of $\vec{r_i}$. Then, the normalized log-return time series of i is defined as $\vec{x_i} \equiv (\frac{r_i(1)-\mu_i}{\sigma_i}, \frac{r_i(2)-\mu_i}{\sigma_i}, \ldots, \frac{r_i(T)-\mu_i}{\sigma_i})$.

[2] A real symmetric matrix, like a correlation matrix, can be decomposed in an orthogonal basis as $\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$, where $\mathbf{V}$ is an orthogonal matrix whose the i^{th} column is the eigenvector v_i of $\mathbf{C}$, and $\mathbf{\Lambda}$ is a diagonal matrix whose i^{th} coefficient is the eigenvalue λ_i. The eigenvalues are ranked in increasing order: $\lambda_{min} = \lambda_1 \leq \cdots \leq \lambda_N = \lambda_{max}$.

Marchenko-Patsur (MP) distribution (Laloux et al., 1999; Plerou et al., 2002) and reads:

$$\rho(\lambda;\sigma^2,q) = \begin{cases} \frac{q}{\sigma^2}\frac{\sqrt{(\lambda_+-\lambda)(\lambda-\lambda_-)}}{2\pi\lambda}, & \text{if } \lambda_+ \leq \lambda \leq \lambda_- \\ (1-q)^+, & \text{if } \lambda = 0 \\ 0, & \text{otherwise} \end{cases}, \tag{4.5}$$
$$\lambda_\pm(\sigma^2,q) = \sigma^2\left[1 \pm \sqrt{1/q}\right]^2,$$

where here $\sigma^2 = 1$ since it is the variance of normalized data. Notice that the distribution has a point mass at 0 if $T < N$. The interesting characteristic predicted by eq. (4.5) is that all the eigenvalues are contained within the range delimited by λ_- and λ_+, which are both non-negative. For N and T finite, there is a small probability to find eigenvalues outside this range, which goes to zero when N, T become large. Any eigenvalue lying above this threshold λ_+ would carry information about a significant correlation structure in the data (Laloux et al., 1999; Plerou et al., 2002).

For the empirical correlation matrix $\mathbf{C}$, the eigenvalues do not necessarily follow the distribution of eq. (4.5), since its correlation structure may not be trivial. Rather, the distribution is formed by a random and a non-random part. Not all the eigenvalues will belong to the random part, because some of them are significant [3]. The question that arises is: how to separate random from significant eigenpairs? We will use RMT for that. It will give is a threshold λ_+ which will be the value separating randomness from significance.

Let us call $N_\rho < N$ the number of random eigenvalues. The eigenvectors related to these eigenvalues span the space of random co-movements of the returns. These eigenvalues will not explain the total variance $\sigma^2 = 1$ of the data, but only a fraction of it, that

[3] The fraction of the variance carried by the k^{th} eigenvalue is $\frac{\lambda_k}{\sum_{i=1}^N \lambda_i}$. For correlation matrices, $N = Tr(\mathbf{C}) = Tr(\mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top) = Tr(\mathbf{V}^\top\mathbf{V}\mathbf{\Lambda}) = Tr(\mathbf{\Lambda}) = \sum_{i=1}^N \lambda_i$. We can decompose the initial variance into its random and non-random ensembles of eigenvalues, respectively called ρ and $\bar{\rho}$:

$$\sigma^2 = \sum_{i:\lambda_i\in\rho} \frac{\lambda_i}{N} + \sum_{i:\lambda_i\in\bar{\rho}} \frac{\lambda_i}{N}. \tag{4.6}$$

we call σ_ρ^2 [4]. Therefore, we expect these N_ρ eigenvalues to follow a MP distribution, but with different values for N and σ^2. We will see later that the parameter T also has to be adjusted. How can we find the value of these parameters?

The shape of the empirical distribution contains the information we need. The idea (Laloux et al., 1999) is to let the parameter σ^2 in the theoretical MP distribution (eq. (4.5)) vary from 0 to 1, and select the value $\hat{\sigma}^2$ for which the theoretical distribution fits the empirical distribution at best. Then, the value for N will be equal to the number of eigenvalues falling within the theoretical bulk.

In Figure 4.2b, the magenta dashed curve is the MP distribution fitted on the empirical distribution. Here we keep the parameter N fixed, which is a reasonable approximation since typically $N_\rho \approx N$ (Laloux et al., 1999).

Besides, the parameter T also needs to be adjusted. The explanation we advance relies on the fact that financial markets have periods of low and high volatility, a fact that violates the i.i.d. assumption of RMT. This effect affects the time parameter T, as illustrated in the following example (see Figure 4.1). Let us consider the case where we have two instruments, A and B, which have constant correlation over time, and two periods of equal length, T_1 and T_2, but with different volatility regimes: in T_2, the volatility of both assets decrease by, say, 10 times compared to the volatility in T_1. Now let $\tilde{a}_t$ and $\tilde{b}_t$ be the respective returns of instrument A and B, and $\bar{a}$ and $\bar{b}$ their respective means. Let $a_t = \tilde{a}_t - \bar{a}$ and $b_t = \tilde{b}_t - \bar{b}$. Computing the sample correlation gives

$$\begin{aligned} c_{AB} &= \frac{\sum_{t\in T_1} a_t \cdot b_t + \sum_{t\in T_2} a_t \cdot b_t}{\sqrt{(\sum_{t\in T_1} a_t^2 + \sum_{t\in T_2} a_t^2) \cdot (\sum_{t\in T_1} b_t^2 + \sum_{t\in T_2} b_t^2)}} \\ &\approx \frac{\sum_{t\in T_1} a_t \cdot b_t}{\sqrt{\sum_{t\in T_1} a_t^2 \cdot \sum_{t\in T_1} b_t^2}}. \end{aligned} \tag{4.7}$$

We see that the contribution of returns in T_2 is very small and can be neglected such that only returns in T_1 define the value of the correlation. This induces the *effectiveness* of T to be reduced. Fewer data points are effectively used in the computation of the

[4] $\sigma_\rho^2 := \sum_{i:\lambda_i\in\rho} \frac{\lambda_i}{N}$ is the sum of the N_ρ normalized noise eigenvalues, and equal to the fraction of variance coming from randomness.

correlation. Missing data affect T in the same manner. By extension, T can also be seen as a fitting parameter, because of this effect of volatility correlation through assets.

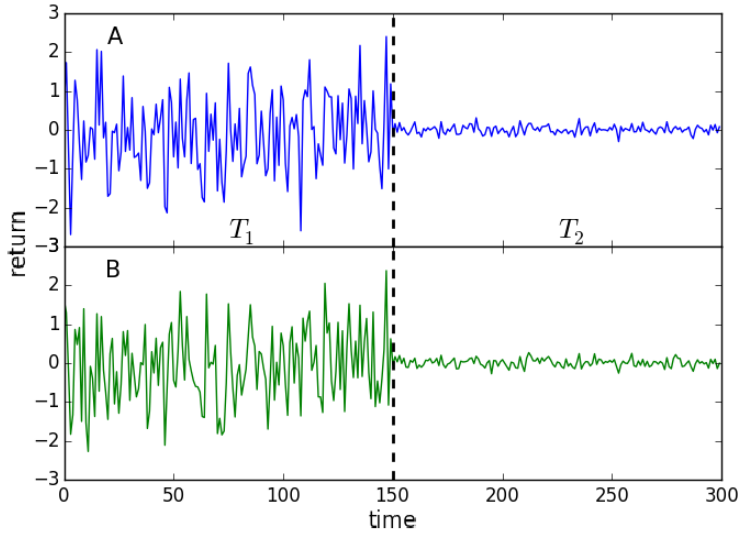
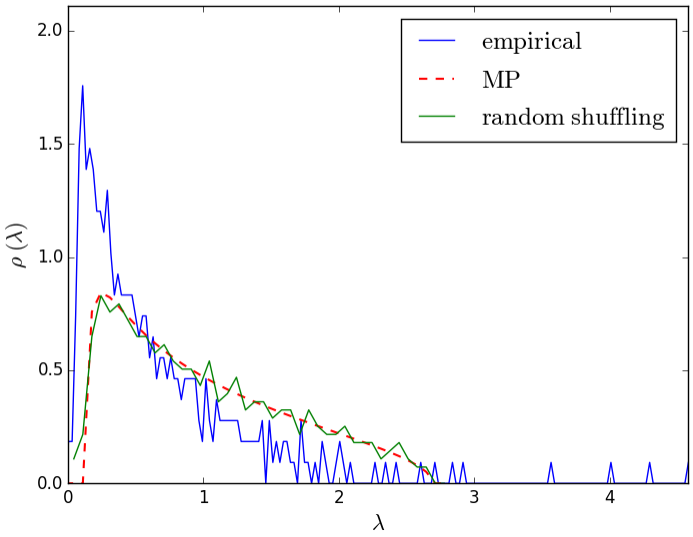


Figure 4.1: Returns of two fictive assets. The two time series A and B are built to have a correlation of 0.8. In period T_1, the volatility of both time series is ten times larger than in T_2. The only effective period that has a significant effect on the empirical correlation is T_1. If the correlation in T_2 was different, the effect would not translate into a significant change in the computed correlation. By extrapolation, it suggests that the correlation of the volatilities of assets in the market reduces the value of the parameter T.

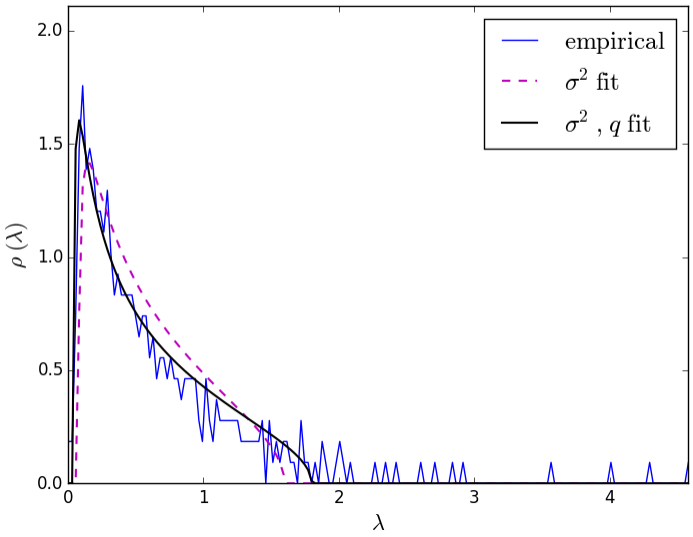
The discussion above shows that three parameters, σ^2, N, and T, need to be adjusted. Since $q = T/N$, we may only consider q and σ^2 as fitting parameters. The resulting distribution after letting σ^2 and q be free parameters is the continuous black curve in Figure 4.2b. Let us call their values respectively $\hat{\sigma}^2$ and $\hat{q}$. The threshold eigenvalue that we find after fitting the empirical distribution is a function of these two parameters, i.e. $\lambda_+(\hat{\sigma}^2, \hat{q})$. We obtain $\hat{\sigma}^2 = 0.577$ and $\sigma_\rho^2 = 0.518$, meaning that more than half of the correlation of the data is just noise [5]. Also, the number of eigenvalues outside the bulk is $N_s = 24$, which is only 5.8% of the total number, and $\hat{q} = 1.86$ instead of $\frac{T}{N} = 2.44$.

[5] Note that $\hat{\sigma}^2 \neq \sigma_\rho^2$. $\hat{\sigma}^2$ is the variance parameter of the eigenvalue distribution of the covariance matrix of N_ρ uncorrelated time series, whose variance equals $\hat{\sigma}^2$, while σ_ρ^2 is the variance explained in the original correlation matrix by the N_ρ eigenvalues arising from noise. We have that

$$\hat{\sigma}^2 = \sum_{i=1}^{N_\rho} \frac{\lambda_i}{N_\rho} > \sum_{i=1}^{N_\rho} \frac{\lambda_i}{N} = \sigma_\rho^2. \tag{4.8}$$



(a) Eigenvalues distribution from shuffled returns.



(b) Fit of the empirical eigenvalue distribution.

Figure 4.2: Distribution of the eigenvalues of the empirical correlation matrix (blue curve). λ are the eigenvalues and $\rho(\lambda)$ the probability density. The largest eigenvalue is at 103.85 (not shown), far away from the threshold of the bulk. **(a)**: The returns were shuffled to destroy the correlation structure. The green curve is the eigenvalue distribution of the correlation matrix built from the shuffled time series. The red dotted curve is the MP prediction assuming all time series are uncorrelated (q=T/N and σ^2=1). It confirms the agreement of RMT for uncorrelated data. **(b)**: The magenta dotted curve is obtained by letting the parameter σ^2 vary and allows to obtain a good fit. The plain black curve is fitting the empirical distribution even better. It is obtained by letting σ^2 and q vary. The threshold eigenvalue is at $\lambda_+(\hat{\sigma}^2, \hat{q}) = 1.81$, above which the eigenvalues are relevant and explain a significant correlation structure in the data.

From this, we obtain a threshold eigenvalue $\lambda_{fit} = \lambda_+(\hat{\sigma}^2, \hat{q})$ below which all the eigenvalues are considered as emerging from noise, and above which as capturing a significant correlation structure in the data. Therefore, the part of the correlation value between two time series coming from noise is found with:

$$\mathbf{C_r} = \sum_{i:\lambda_i \leq \lambda_{fit}}^{N} \lambda_i |v_i\rangle\langle v_i|, \tag{4.9}$$

where v_i is the eigenvector corresponding to the eigenvalue λ_i, and where we use the physicists' bra-ket notation $\langle\cdot|$, $|\cdot\rangle$ to signal respectively a row and a column vector. These eigenvalues are the ones below the bulk delimited by the MP distribution in Figure 4.2b. This matrix is the contribution of noise in the correlation links, and constitute our null hypothesis $\langle \mathbf{C} \rangle = \mathbf{C_r}$. Subtracting it from the initial correlation matrix is similar to zero-

ing out the eigenvalues in question, or equivalently, projecting [6] the data to the remaining (significant) eigenvectors. We end up with a measure of significance of correlation links, and the significant strength between nodes, after taking care of the random part in the values of the correlations, is given by:

$$\mathbf{B_r} = \mathbf{C} - \mathbf{C_r}. \tag{4.10}$$

MacMahon and Garlaschelli (2015) suggest using the threshold $\lambda_+(\sigma = 1, q = \frac{T}{N})$ and have for null hypothesis:

$$\mathbf{C_r} = \sum_{i:\lambda_i \leq \lambda_+(1,q)}^{N} \lambda_i |v_i\rangle\langle v_i|. \tag{4.11}$$

They argue that this is the value expected if the correlation was purely random, by analogy with the original null hypothesis in the modularity of positively weighted networks, which is the expected value if nodes were linked at random. Indeed, if we shuffle the returns of the time series (i.e. permute them with uniform probability), the correlation structure is destroyed. The eigenvalue distribution resulting from these shuffled returns follows well the MP prediction with $\sigma^2 = 1$ and $q = T/N$ (respectively the green curve and the red dashed curve on Figure 4.2a), which means that all correlation relations emerge, indeed, from noise. In this case, the threshold eigenvalue is indeed $\lambda_+(1, q)$.

However, we want to take extra care here, because the noise eigenvalues predicted by the shuffling are not related to the same eigenvectors, whether we consider the shuffled case or the original case. Moreover, eigenvalues only have a meaning when they are compared to the other eigenvalues of the same dataset, so it is dangerous to compare eigenvalues from two structurally different datasets. We think that the threshold predicted by the fitting of the empirical distribution is more accurate, because it relies on true (not shuffled) data and only extracts eigenvectors that span the space of random correlation.

[6] See Figure 4.3 for an example of data projection and dimension reduction in 2D.

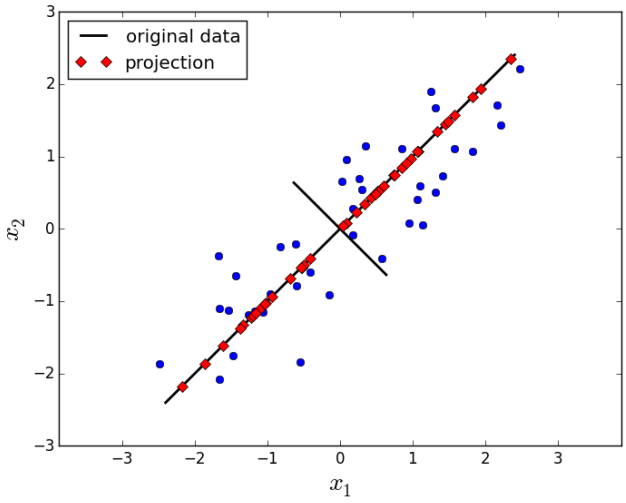


Figure 4.3: A visualization of dimension reduction in 2D using principal component analysis (PCA). The blue dots are the original data points of two simulated correlated variables x_1 and x_2. The PCA decomposition gives two eigenvectors whose directions are given by the rotated axes. The red diamonds are the orthogonal projection of the blue dots on the principal eigenvector, and are obtained by putting to zero the eigenvalue of the second eigenvector. Instead of two coordinates, only one is now needed to describe a given point: the data is reduced from 2D to 1D. In higher dimensions, we project the data on the significant eigenvectors, so we subtract $\mathbf{C_r}$ from $\mathbf{C}$.

4.2 Market mode

The dynamics of stocks are strongly correlated to a common factor, which can be named the *market factor* or *market mode*. This causes most assets to have pairwise positive correlations, and induces that the algorithm will put all the assets in the same community. To be able to detect a finer structure of correlation, we need to remove the effect of this market mode.

One can think of several ways to remove the market mode. One can remove, for each asset, the average cross-sectional return [7] of each day, or alternatively the return of the SP500 index, weighted eventually by the regression factor β, i.e. the sensitivity of an instrument to the common factor (Borghesi et al., 2007). An other method will be mentioned and used in the next chapter. The technique we select, because it is the most natural to implement in our framework, consists in zeroing out the largest eigenvalue λ_{max} (MacMahon and Garlaschelli, 2015). For the period considered, the maximum eigenvalue

[7] The average of the returns of each asset for a day.

is 103.5, or $\frac{103.5}{416} = 24.9\%$ of the total variance, which is a significant value. The related eigenvector v_{max} is expected to have loadings of the same sign. This can be seen as a portfolio which is long in all the assets, i.e. which tracks the market. The reason is that stocks that belong to the same community share a common behavior, so they should have a positive correlation to the most important factor of the community. However this is not necessarily the case, especially if the second eigenvalue is close to the maximum one. If this is the case, the matrix

$$\mathbf{C_m} = \lambda_{max}|v_{max}\rangle\langle v_{max}| \tag{4.12}$$

is a matrix with only positive entries, so removing its effect on $\mathbf{C}$ comes down to removing a positive quantity to the links between assets, hence making small links turn negative. Rather than removing a fixed quantity, it removes a quantity dependent on the exposure of the two nodes to the common factor. If one wants to remove the market mode, the weights that shall be passed to the Louvain algorithm are expressed with the matrix

$$\mathbf{B_m} = \mathbf{C} - \mathbf{C_r} - \mathbf{C_m}. \tag{4.13}$$

Because eigenvectors are orthogonal, this operation does not affect the random eigenpairs. We can do both operations of removing the market mode and the noise at once. An interesting aspect of the communities formed after removing the market mode is that they are *residually* anti-correlated (they are originally correlated, but become anti-correlated if we omit the main common factor), because their resulting link is negative (MacMahon and Garlaschelli, 2015).

4.3 Modularity for correlation matrices

We derived in the previous section how to compute the significant links B_{ij} to be fed to the Louvain algorithm. We need one more thing, which is the scaling factor F. We use

$$F = \sum_{i,j} C_{ij}, \tag{4.14}$$

which is equal to the volatility of the sum of all the normalized time series $x_{tot}(t) = \sum_{i=1}^{N} x_j(t)$ (MacMahon and Garlaschelli, 2015). This quantity gets larger when the stocks become more correlated on average, and when we include more stocks. Dividing by

this quantity scales down these two effects, so the modularity can be compared between networks computed with different numbers of stocks as well as between periods in which the general correlation is different. It is a positive quantity, unless our data set contains only two perfectly anti-correlated time series (in which case F=0 and the modularity computation breaks). Except for this special case, the modularity

$$Q(\vec{\sigma}) = \frac{1}{F} \sum_{i,j} B_{ij} \cdot \delta(\sigma_i, \sigma_j) \tag{4.15}$$

is a valid measure. The matrix $\mathbf{B}$ is to be chosen from either $\mathbf{B_r}$, or $\mathbf{B_m}$ if one wants to also remove the market mode.

4.4 Results

We apply the Louvain algorithm in several steps as proposed by MacMahon and Garlaschelli (2015). At the beginning, we subtract $\mathbf{C_r}$ from $\mathbf{C}$ to adjust the weights from the noise (eq. (4.10)), and apply the Louvain method. At this stage, we obtain a set of communities. Then, to reveal a finer partition, we consider each of the communities P, which are expressed by a correlation matrix $\mathbf{C}^{(\mathrm{P})}$. For each of them, we compute the matrix $\mathbf{C}_\mathbf{r}^{(\mathrm{P})}$ and the matrix $\mathbf{C}_\mathbf{m}^{(\mathrm{P})}$, which represents the 'market', or global mode, of the community P. For each community P that we want to split, we pass to the clustering algorithm the matrix

$$\mathbf{B}_\mathbf{m}^{(\mathrm{P})} = \mathbf{C}^{(\mathrm{P})} - \mathbf{C}_\mathbf{r}^{(\mathrm{P})} - \mathbf{C}_\mathbf{m}^{(\mathrm{P})}. \tag{4.16}$$

A good sanity check here when subtracting $\mathbf{C}_\mathbf{m}^{(\mathrm{P})}$ is to plot the distribution of the loadings of $v_{max}^{(i)}$ and make sure they all have the same sign, to ensure that we only remove positive quantities from the links [8]. We repeat this second step as many times as wanted, as long as the sub-communities are not too small. Indeed, RMT only works for large dimensional data, and starts to break down when we approach values $N, T \approx 100$ (a value that *we* set empirically, and below which we decided not to divide the community further).

[8] If the correlation matrix $\mathbf{C}^{(\mathrm{P})}$ has uniquely positive entries, then the Perron-Frobenius theorem ensures that $v_{max}^{(P)}$ has loadings of the same sign. However, it may happen that some entries are negative, so a check by hand is necessary. This can occur for the smallest communities, in which the first and second maximum eigenvalues are close to each other. We did not encounter this issue for the results in this thesis so we did not tackle this problem.

Figure 4.4, is a graphical representation of three steps of our algorithm. The stocks are progressively split into finer communities. The benefit of this iterative algorithm is that it overcomes one of the main drawbacks of the modularity optimization, in the sense that it circumvents the resolution limit (the fact that modularity maximization often fails to detect communities smaller than a certain scale (Fortunato and Barthémy, 2007)). If we want a finer partition, we can split the current communities into sub-communities.

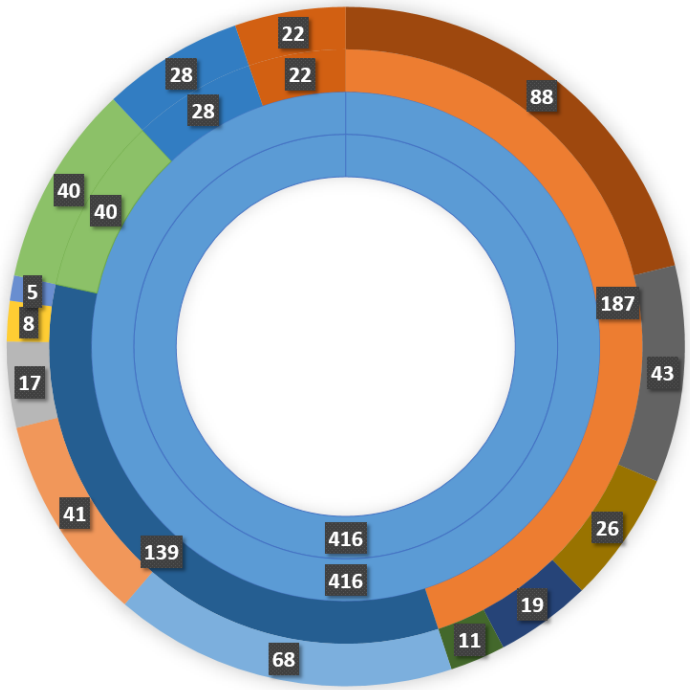


Figure 4.4: Layer representation of the steps of the algorithm. Each layer is a step of the algorithm, and the numbers are the number of stocks in the associated community. The inner layer is the initial data set containing the 416 stocks. The second layer is the partition of the initial data set, after taking care of the contribution of the noise. One notices that all of the stocks have been put in one unique community, which means that all the stocks follow the same market mode. The third layer is the result after also taking care of the market mode: five communities materialize. The fourth layer appears after removing the global mode, and the noise, of the largest communities. We do not alter the small communities, as we cannot use RMT for small datasets.

Figure 4.5 and Table 4.1 are generated using the partition given in the third layer (second step of the algorithm) from the center of Figure 4.4, in which five communities are formed. Figure 4.5 is the representation of the network, partitioned into these five communities. In Table 4.1, we classified the stocks according to their sector after the GICS

	community 0	community 1	community 2	community 3	community 4
Consumer Discretionary	51	12			
Consumer Staples	3	28	1		
Energy			35		
Financials	18	41			21
Health Care	15	33			1
Industrials	39	17			
Information Technology	43	2			
Materials	18	3	3		
Telecommunication Services		3			
Utilities			1	28	
Total	187	139	40	28	22

Table 4.1: Repartition of stocks in the third layer of the algorithm. We classified them by their GICS sector (left column). One sees the good overlap between the qualitative and the quantitative classifications. All industries are confined at maximum within three communities out of five. Communities 0 and 1 are the largest and are less well defined. The colors of the communities are the same as in Figure 4.5.

(Global Industry Classification Standard) and their communities found by the algorithm. We find a good overlap of these two classifications: stocks that are qualitatively alike also cluster together. This demonstrates that the algorithm is successful in clustering similar objects based on their quantitative relationships.

4.4.1 Extension of the modularity

Lastly, we want to say a word about the final partition produced by the algorithm. Although it generally produces reasonable communities, surprisingly one can sometimes find two closely related instruments in two different clusters. This casts doubt on the applicability of the original definition of the modularity by Newman and Girvan (2004) for positively weighted networks to measuring convincingly the quality of a partition for correlation matrices. The original measure is based on the intuition that nodes that have a density of links larger than average should be forming a community. This idea applied to a correlation matrix will tend to give a larger score to the move of a node into a large community, even if it is less correlated *on average* than another, smaller, community.

As an example, let us suppose that the algorithm is asked to move a node n to the best among two communities, 1 and 2. Community 1 contains 20 nodes that each have a correlation of 0.1 with n. Community 2 contains 2 nodes that each have a correlation

of 0.9 with n. Intuitively, we want the algorithm to put the node n into community 2, however the move to community 1 gets a higher score with this definition of modularity (2.0 vs. 1.8). This problem typically arises when the cardinalities of the communities are inhomogeneous. For correlation matrices, a negative link may exist between communities 1 and 2. The two communities will never be merged, so n will never be in community 2, although it is very strongly correlated with its components. Such a final partition contradicts the intuitive outcome, so there is a need to improve the quantitative measure of the pertinence of a move and of the partition.

To tackle this, we could for example try to consider the *average* correlation of the node n to the communities 1 and 2, or take the correlation of n with the portfolios formed by the nodes each of the communities. Depending on the order the algorithm moves the nodes, this could result in qualitatively better clusters. However, by doing so, it can happen, for some initial configurations, that the algorithm gets stuck in a loop where it moves a few nodes back and forth from a community to another, and the algorithm does not converge, so we did not pursue with this idea.

Another idea is to apply a non-linear, increasing function to the links, in order to give more importance to large correlation values. It can be pertinent to look at the edges in a network as a set of forces driving the movement of the nodes. In Newman's framework, the force $\mathcal{F}_{ij}$ exerted by an edge between two nodes i, j follows a linear law. It is expressed as $\mathcal{F}_{ij} = \frac{2}{F} B_{ij}$. Then, optimizing the move of a node i to a community P_a with respect to the modularity change $\Delta Q^{(i \to P_a)}$ is exactly equivalent to letting the node i join the community where the attracting force $\mathcal{F}_i(P_a)$ is the strongest:

$$\Delta Q^{(i \to P_a)} = \frac{2}{F} \sum_{j \neq i} B_{ij} \cdot \delta\big((\sigma_j, \sigma(P_a)\big) = \mathcal{F}_i(P_a). \tag{4.17}$$

We could as well model this force differently, and use for example a quadratic force. In this case, the force between a node i and a community P_a would be

$$\mathcal{F}_i(P_a) = \frac{2}{F} \sum_{j \neq i} B_{ij}^2 \cdot sign(B_{ij}) \cdot \delta\big(\sigma_j, \sigma(P_a)\big) = \frac{2}{F} \sum_{j \neq i} \frac{B_{ij}^3}{|B_{ij}|} \cdot \delta\big((\sigma_j, \sigma(P_a)\big). \tag{4.18}$$

The related modularity to optimize is in this framework

$$Q^{quadratic}(\vec{\sigma}) = \frac{1}{F}\sum_{ij}\frac{B_{ij}^3}{|B_{ij}|}\delta(\sigma_i,\sigma_j). \tag{4.19}$$

It gives more importance to large correlation coefficients relative to small ones.

We actually used this framework for other projects at Nafora, and obtained more intuitive clusters. In this thesis however, we keep the original modularity definition, because we did not find any misallocations in the clustering of stocks, and also because we considered this idea only after the whole analysis had been conducted.

Providing the user with different force models gives him the freedom to choose the one that suits his problem best. Such an approach is actually used in some popular network drawing algorithms (Noack, 2004; Fruchterman and Reingold, 1991; Jacomy et al., 2014). These algorithms construct non-physical force models, and aim at minimizing the energy of the system.

The adaptation of the modularity of a network built on a correlation matrix seems to be an important and interesting work to carry out, and has not been done yet to our knowledge, except for works that focused on reviewing the null hypothesis (MacMahon and Garlaschelli, 2015; Gómez et al., 2009). Here we give an idea coming from empirical findings, which might need firmer support from a theoretical framework.

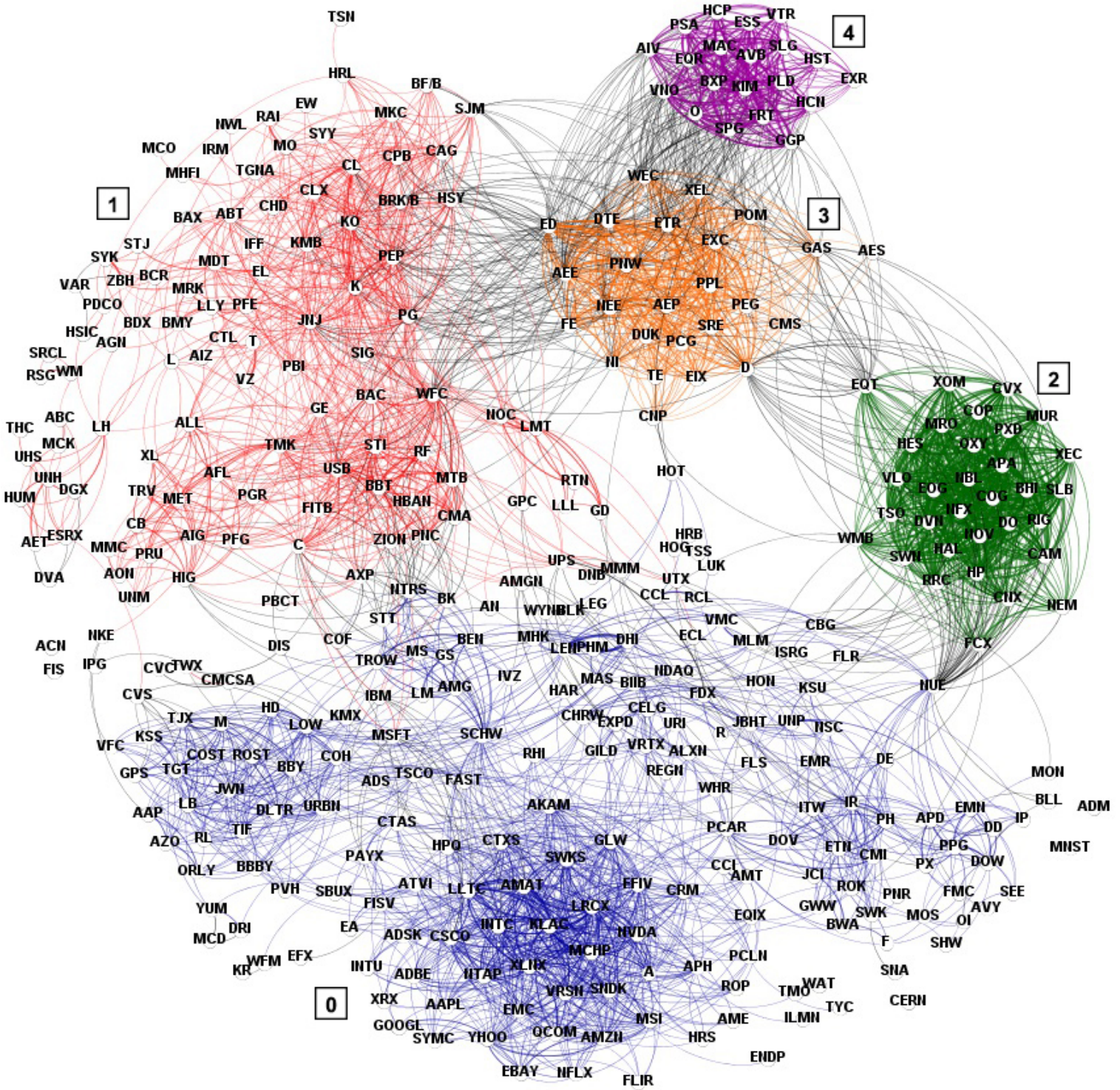


Figure 4.5: Network representation of 416 stocks of the S&P500, using the partition of the third layer. The numbers represent the communities. An edge between two nodes indicates a significant correlation. A colored edge means that the two nodes belong to the same community, revealed by the algorithm. The colors of the communities are the same as in Table 4.1. The closer two nodes are, the more correlated they are. The stocks are represented by their ticker symbol. For example, if one looks at the bottom of the network, one notices that comparable IT companies like Apple, Alphabet, eBay, Netflix, Amazon, and Yahoo, are in the same community and close to each other. The network is obtained using the Fisher transformation (Fisher, 1915) of the weights of $\mathbf{B_m}$ with a threshold at 1.25 standard error, that is fed to the ForceAtlas2 algorithm implemented in the Gephi software (Jacomy et al., 2014).

Chapter 5

Clustering of Days

In the previous chapters, we presented the design of an algorithm that clusters together stocks, based on the similarity of their behavior as expressed by the Pearson correlation. In this chapter, we proceed to do the same with days, i.e. group together days in which stocks behaved in a similar fashion. By similar behavior, we mean that stocks have comparable returns in two different days. To do so, the core questions one faces are: what makes two days be alike, how to quantify this relationship, and, once this measure is defined, how to detect whether the value is statistically significant. For the purpose of creating a trading strategy, we are interested in fast-changing states, rather than general states that would describe periods of months or years in the same state where one can focus on the variation of a rolling, short-term correlation matrix (Münnix et al., 2012).

5.1 Measure of similarity

One intuitive way of classification is to cluster all days in which almost all stocks went up in one group, in another group where they went down, in a third group where, say, the IT sector went up and the energy sector went down, etc. Another way is to study the *relative* behavior of the stocks, regardless of the direction and magnitude of the move of the market. In this case for example, a group would be formed by days in which the IT sector outperformed the market, and where the energy sector underperformed the market, no matter what the market did in these days. This second way reads the relative information between data points, not the absolute one. Borghesi et al. (2007) shows that removing the market dynamics gives much better information from the clustering, and

Marsili postulates [1] that there are very good reasons to think that relative motion and the center of mass should be subject to different dynamics. For these reasons, we select this second way for the future development of our analysis.

For a given day, we have a series of returns of each asset, which can be represented as an N-dimensional vector of cross-sectional returns. The Pearson correlation between the vectors of cross-sectional returns is the measure that we will use as the measure of similarity between days. This measure indeed fulfills the requirements of removing the market dynamics, it increases (decreases) if an asset behaves in a similar (different) fashion in two days, hence it will be more (less) likely to find these two days in the same cluster. However, we cannot simply take the raw data and compute the Pearson coefficient since the time series of different assets have different volatility. If we naively do that, the assignment to a state would only be based on the most volatile stocks. We have to normalize the time series beforehand.

Marsili (2002) proposes such a transformation, which consists in iteratively normalizing the data: at each step, one normalizes the time series for each asset, followed by the normalization of the cross-sectional series for each day. It reads as follows:

$$\begin{aligned} x_i^{(2k+1)}(t) &= \frac{x_i^{(2k)}(t) - \left\langle x_i^{(2k)} \right\rangle}{\sqrt{\left\langle \left(x_i^{(2k)} - \left\langle x_i^{(2k)} \right\rangle \right)^2 \right\rangle}}, \\ x_i^{(2k+2)}(t) &= \frac{x_i^{(2k+1)}(t) - \left[x^{(2k+1)}(t) \right]}{\sqrt{\left[\left(x^{(2k+1)}(t) - \left[x^{(2k+1)}(t) \right] \right)^2 \right]}}, \end{aligned} \tag{5.1}$$

where $x_i^{(0)}(t)$ is the log-return of asset i on day t, $\langle \cdot \rangle$ is the time average, $[\cdot]$ is the average across assets, and k is the iteration step. The first equation normalizes the time series while the second one normalizes the cross-sectional returns. The normalized data are obtained as the limit $x_i^{(\infty)}(t) = \lim_{n\to\infty} x_i^{(n)}(t)$. As a side effect, this transformation removes the volatility clustering property of the market (Cont, 2001), since it averages

[1] Marsili, personal communication 2016.

the volatility of each day to one. The matrix of transformed returns converges rapidly to a stable value, so in practice we stopped the process after 500 iterations to have the approximation $\mathbf{X}^{(\infty)} \approx \mathbf{X}^{(500)}$. This operation allows us to use the correlation matrix of these transformed returns as a similarity measure:

$$\mathbf{M} = \frac{1}{N}\mathbf{X}^{(\infty)}\mathbf{X}^{(\infty)\top}. \tag{5.2}$$

Notice that the transpose sign $^\top$ has changed position compared to eq. (4.4), meaning that we transpose the matrix and indeed compute the correlation between days.

Because we deal with a correlation matrix, we can use RMT as done previously to take out the contribution of randomness to detect the significance of a link between two days. In the previous chapter, the N objects were the stocks, and the T sampling units (observations) were the days. Here, the objects are the days and the sampling units the stocks, and we will respectively name them T' and N'. The number of objects is larger than the number of observations (T'>N'), so there is a point mass at zero in the distribution of the eigenvalues (see Figure 5.1). This is because the matrix has a rank of N' and dimension T', so there are $T' - N'$ zero eigenvalues. We argued before that both the volatility parameter σ^2 and the observations-to-objects ratio parameter q had to be fitted. For the latter parameter, we claimed that this was possible because of the fluctuating volatility of the market, which affects the effective number of observations T. However now, the parameter of observations N' represents stocks, so the argument cannot hold. Therefore, we only fit σ^2, find the threshold value $\lambda_+(\hat{\sigma}^2)$, and adjust T' afterwards to the number of eigenvalues within the bulk predicted by RMT. Similarly as before, we obtain a 'noise' matrix built from the noise eigenvalues

$$\mathbf{M_r} = \sum_{i:\lambda_i \leq \lambda_+(\hat{\sigma}^2)}^{T} \lambda_i |v_i\rangle\langle v_i|, \tag{5.3}$$

and get the weights used by the clustering algorithm with

$$\mathbf{B_M} = \mathbf{M} - \mathbf{M_r}. \tag{5.4}$$

For splitting clusters into sub-clusters, we take the series of cross-sectional returns of the days inside a given cluster i, compute the matrix $\mathbf{M}^{(\mathrm{i})}$ from the transformed data $\mathbf{X}_{(i)}^{(\infty)}$, and the noise matrix $\mathbf{M}_{\mathbf{r}}^{(\mathrm{i})}$. This allows us to get the weights

$$\mathbf{B}_{\mathbf{M}}^{(\mathrm{i})} = \mathbf{M}^{(\mathrm{i})} - \mathbf{M}_{\mathbf{r}}^{(\mathrm{i})} \tag{5.5}$$

to pass to the clustering algorithm. We do not need to remove the largest eigenvalue in this part. The reason is that the iterative operation of normalizing $\mathbf{X}_{(i)}^{(0)}$ already does the job of removing the most prominent pattern of the cluster i. Note that we did not use this technique when clustering stocks only because we did not want to alter too much the data by normalizing the volatility of each day.

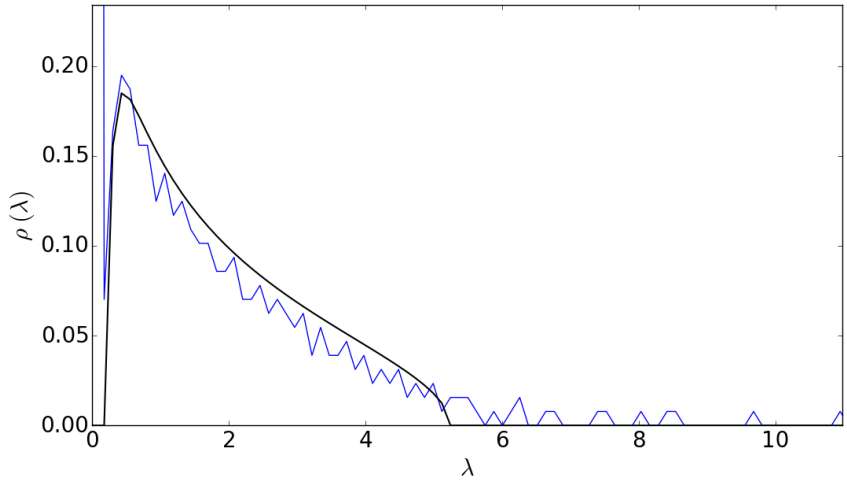


Figure 5.1: Distribution of eigenvalues of the correlation matrix between days. The empirical distribution (blue) is fitted with the theoretical MP distribution (black), by letting σ^2 be a free parameter, and adjusting afterwards the number T' to be equal to the number of eigenvalues below the theoretical bulk. One notices the point mass at zero, which comes from the fact that the number of sampling units N' is smaller than the number of objects T'.

5.2 Results

We run the algorithm described above, and stop it at the second step. The first step has two large communities of days of roughly equal size, contrary to the stock clustering, where it had only one cluster containing all the objects. At each step, we discarded communities whose size were smaller than 1% of the original number of days (1013), that

is, communities have to contain more than 10 days to be considered as a state. The reason is that too few observations of a state do not lead to a reliable definition of this state.

We define $\langle r_i|\omega\rangle$ the (non-normalized) average return of a stock i in a given state ω, and $\langle \tilde{r}_i|\omega\rangle$ the average excess return over the market ($\tilde{r}_i(t) = r_i(t) - r_m(t)$, where r_m is the average return of the assets). To visualize a state, we plot the values $\langle \tilde{r}_i|\omega\rangle$ in one state against another in Figure 5.2. The reason is simply because it is more convenient to represent them in two dimensions, rather than in one or three dimensions. Two states plotted against each other have nothing in common. A point represents a stock, and its position on the graph depends on its average return in the two considered states. The colors indicate to which community the stock belongs to, and are similar to the color of Figure 4.5 and Table 4.1.

The first fact one can notice is that points of the same color are placed in the same region in each of the graphs. Stocks belonging to the same community are grouped in very clear clusters. This means that stocks that were found to have a similar behavior as measured by their correlation, and were put in the same community, also have a similar behavior when one looks at a particular state. This shows that our algorithm is able to detect very definite states.

For example, state 3 and state 8 are dominated by large excess returns (respectively positive and negative) of community 2. State 1 is characterized by downward moves of communities 2, 3, and 4, and an upward move of community 0, while the opposite happens in state 9. In states 4 and 6, community 4 is down significantly. In state 5, communities 2, 3, and 4 are up.

It is interesting to see (Figure (5.3)) that the number of links in the matrix $\mathbf{M}$ of the network of days that are above a certain threshold decreases very quickly as the threshold increases. In fact, it seems to decay exponentially fast. Such a relation is always interesting to find, since it generally reveals important properties of the two related quantities, in particular in natural science. This law describes for example the rate of many enzyme-catalyzed biochemical reactions, the electric charge contained in a capacitor over time, heat transfer, radioactive decay, etc. In our case, this means that the probability of finding two days in which m assets have comparable returns decreases exponentially with respect

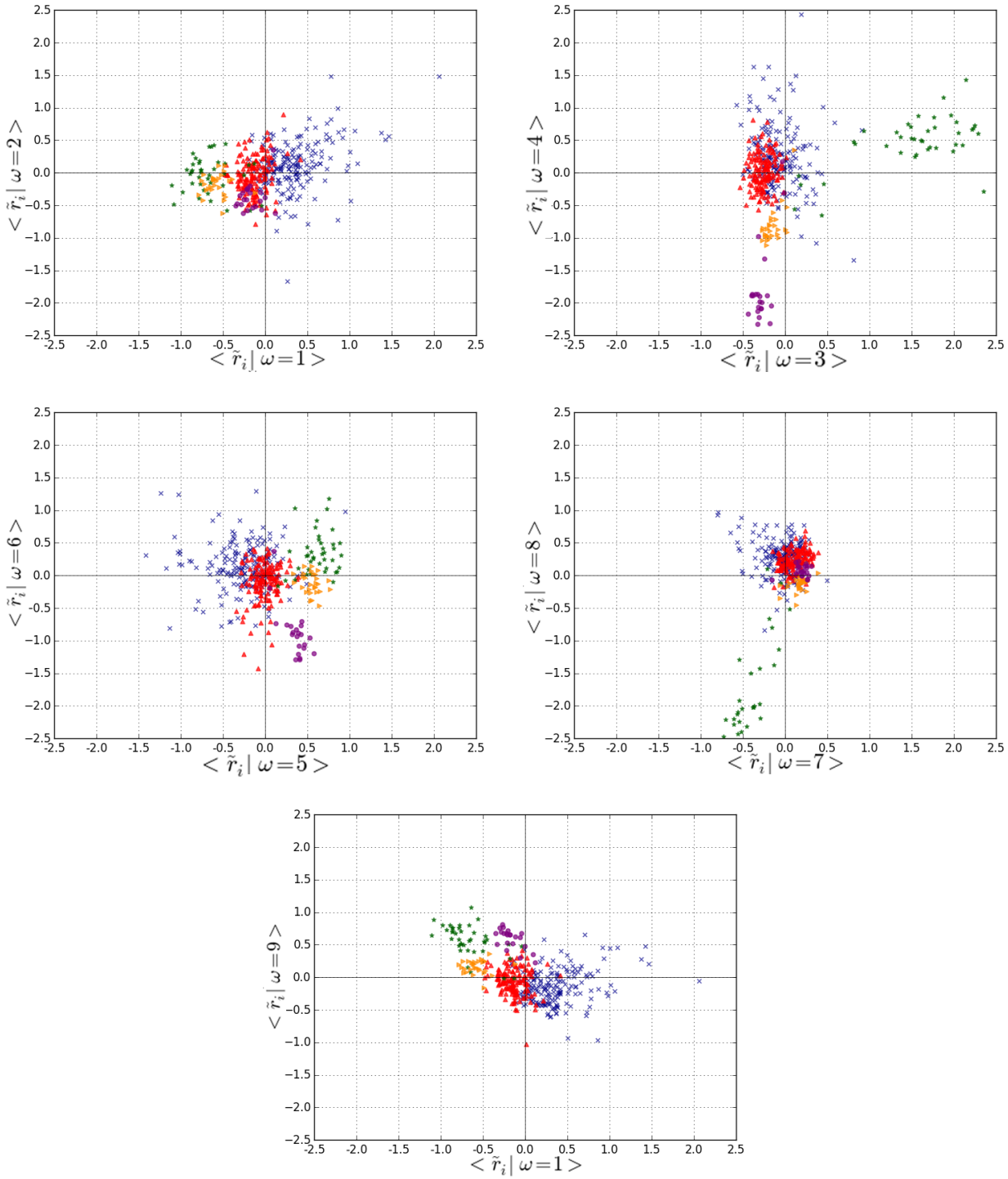


Figure 5.2: Visualization of the market states. The states are obtained after two iterations of the clustering algorithm. Each point represent a stock, and its position is its mean excess return in the two considered states. For example, looking at the middle left plot (states 5 and 6), we see that stocks in the purple community have on average a positive excess return of about 0.5% in state 5 and -1% in state 6. The behavior of each community in different states is very well-defined. Communities 0, 1, 2, 3, 4, are respectively shown with blue crosses, red up-triangles, green stars, orange left-triangles, and purple dots.

to m.

Figure 5.4 shows the time series of the market states during the studied period (Jan. 2003 - Dec. 2006). There are 9 states, ordered by the number of times they occurred, plus a state 0 containing the days that were discarded because they were not distributed to a state large enough. They seem to be homogeneously distributed in the studied period. In the next section, we will study whether this time series has a predictability power to forecast (excess) returns.

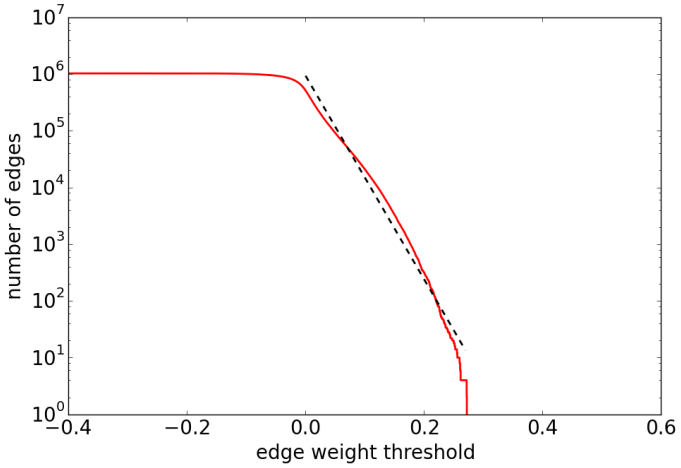


Figure 5.3: Exponential decay of the number of edges whose values are above a certain threshold in the network of days. The black dashed curve is a linear fit, in the semi-log plot, of the decreasing part of the number of edges.

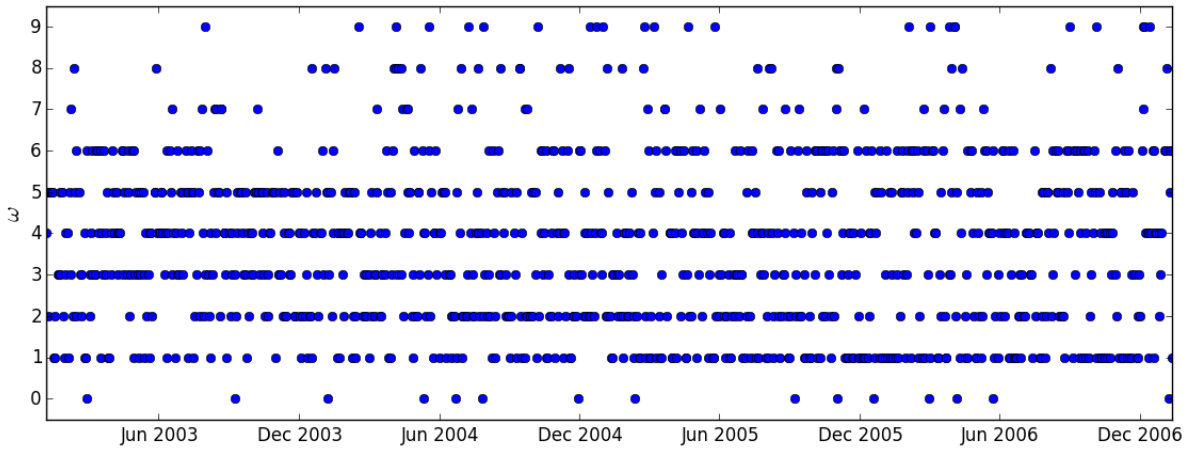


Figure 5.4: Temporal distribution of the market states. There is one point for each day. The states are ordered by their size. The number of days in each state, from 1 to 9, is respectively 180, 166, 157, 145, 140, 125, 31, 30, and 24, plus 15 days that are in the unassigned (random) state 0.

Chapter 6

Strategy Research

6.1 Predictive power of the states

In this chapter, we analyze the market states revealed by the algorithm with the objective of constructing a trading strategy. We study whether the state in a day is useful to predict the excess return (the return over the market return) of the stocks for the following day. If we are able to predict the excess returns of tomorrow, we can build a long-short portfolio according to our prediction, which would make our position market-neutral by construction, i.e. hedged against a global movement of the market. We can also try to predict the absolute return, which would result in a portfolio that would go long in the stocks with a positive expected return and short otherwise. The measure of predictability that we use relies on the work conducted in Marsili (2002), who found that their states had predictability power. We define it as a weighted average signal-to-noise ratio for each stock i expressed as:

$$H_i(k) = \sqrt{\sum_{\omega} \rho_{\omega} \frac{\left\langle \tilde{r}_i(t+k) - \left\langle \tilde{r}_i \right\rangle \middle| \omega(t) = \omega \right\rangle^2}{\left\langle \left(\tilde{r}_i(t+k) - \left\langle \tilde{r}_i \right\rangle \right)^2 \middle| \omega(t) = \omega \right\rangle}}, \tag{6.1}$$

where ρ_{ω} is the frequency with which state ω occurs, $\left\langle \cdot \right\rangle$ denotes as before the time average, and $\left\langle \cdot \middle| \omega(t) = \omega \right\rangle$ denotes the average over the days in state ω. $H_i(k)$ is large if the excess returns have consistently the same sign, and small if they are randomly drawn. More importance is given to larger states. The average excess return of the stock is removed to only detect abnormal excess returns.

In Figure 6.1, we plot the distribution of $H_i(k)$ for values $k = 0$, $k = 1$, and $k = 20$. We are interested in the distribution for $k = 1$, which is the predictability for the next day. We benchmark it against the distribution at $k = 0$, which is the maximum predictability we can obtain with our states (we predict today's return knowing today's state), and against the distribution at $k = 20$, where we assume that the predictability of the return 20 days from today should be very close to insignificant.

We find that for no stocks $H_i(1)$ is significantly larger than $H_i(20)$, meaning that prediction is very difficult, if possible at all. This result is somewhat surprising since it is not in line with what was discovered in Marsili (2002), who found that $H_i(1)$ was significant for many stocks.

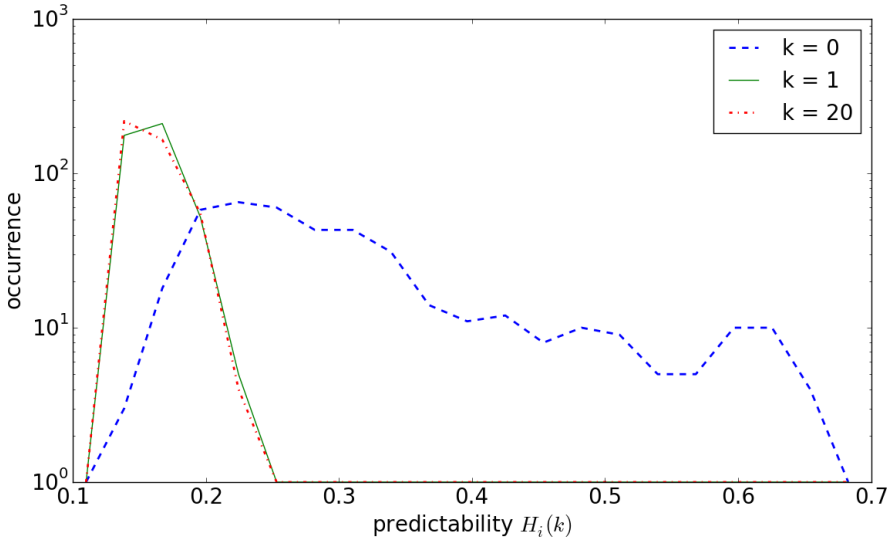


Figure 6.1: Absence of predictability of the states. The graph shows the distribution of predictability H_i for $k = 0, 1, 20$. $k = 0$ and $k = 20$ act as benchmarks for $k = 1$. The first one is the maximum predictability distribution reachable from the model of states. The second one is the distribution if we assume that there is no predictability. The distribution for $k = 1$ follows the distribution of $k = 20$, revealing the absence of predictability of the market states.

We shall list the differences between the aforementioned paper (Marsili, 2002) and our work, to have a view of why our result differs:

1. Marsili (2002) uses a different, unsupervised algorithm, based on the maximization of a likelihood function (Giada and Marsili, 2001, 2002). This algorithm would

probably find a different partition of the dataset we study. However, since both their and our algorithms are unsupervised clustering algorithms, and since they both use the same measure of similarity, we do not think that this causes the end results to be contradictory, although we cannot claim that this has no effect. To have another partition, we computed the predictability distribution iterating the algorithm a third time by splitting states larger than 100 days, but this gave no manifest change (see Appendix, fig. B.1).

2. Instead of using end-of-day returns, Marsili (2002) uses open-to-close returns to both define the states and predict the returns. We ran the simulation with these intraday returns, but found a similar distribution (see Appendix, fig. B.2).

3. Instead of predicting excess returns $\tilde{r}$, Marsili (2002) predicts the raw returns r. Once again, the plot does not substantially change (see Appendix, fig. B.3).

4. We use 416 stocks of the S&P500 over $T = 1013$ days, while Marsili (2002) uses 2000 stocks, selected based on their trading activity, over $T = 2358$ days. It is possible that the stocks that showed predictability in their work were the ones not included in the S&P500. This could be interesting to investigate further, but we have not conducted this research.

5. The period they analyze starts on 1$^{\text{st}}$ January 1990 and ends on 30$^{\text{th}}$ April 1999, while ours starts on 1$^{\text{st}}$ January 2003 and ends on 1$^{\text{st}}$ January 2006. Here might be the main source of dissimilarity of the predictive power of the two models. In the recent decade, there has been a surge in the amount of trading carried out by algorithmic trading strategies (McGowan, 2010), as more electronic exchanges opened. These algorithms have made market inefficiencies disappear faster. A strategy that had some predictive power in the past may not work anymore in this fast-changing, computer led, new environment.

Resulting from the discussion above, we conclude that, with our market state sequence, it seems not to be possible to construct a trading strategy based on end-of-day prices, which would produce a profit-and-loss curve better than random trades.

6.2 An intraday strategy

The market states were not successful at predicting the returns of the next day, but might be suitable for an intraday strategy. This is what will be tested in this section. Our idea is to imagine that if the state is quite stable through a day, then it is possible to compute the state in the beginning of the day, and build an expectation of what the returns should be at the end of the day. A technical problem we immediately face is that, here at Nafora, we do not have a database for intraday prices of stocks, so we have to rely on data from Bloomberg, which only provides intraday data for the past 140 days. The analyzed period spans from 2015-9-15 to 2016-3-28. We also consider end-of-day data for the period between 2014-3-4 and 2016-3-28. These dates include the period for which we have intraday data, plus 400 days preceding 2015-9-15. Theses additional 400 days will be used to define states. We pick this number large enough in order to have enough days to be able to construct states, but not too large so that we do not build a prediction of the returns of a day from data from far away in the past. Our dataset is more recent than for the work carried out before, so by only keeping stocks for which we have data more than 90% of time, we end up with a set of 493 stocks.

We compute the close-to-10am [1] returns ('morning' returns) as well as the close-to-close returns for the period between 2014-3-4 and 2016-3-28. For a day between 2015-9-15 and 2016-3-28, there is by construction an overlap between these two returns since the latter comprises the data of the former. To check the stability of a state through the day, we concatenate the data set of the two kinds of returns (540 close-to-close returns and 140 morning returns), and classify them with our algorithm, stopping at the second step. It appears that the days using the morning returns are classified 54.3% of the time in the same cluster as the same day using the close-to-close return. This is much higher than the 18.4% we would find if the morning returns were randomly placed [2]. We explain this by the fact that, if there is a jump in the price of an asset at the opening of the market,

[1] New York Stock Exchange time.

[2] To obtain this number, we keep the classification in states: we obtain 6 states plus a random state. The morning returns are distributed over the states as: (34, 33, 25, 19, 15, 12, 2), with the last number being the number of morning returns placed in the random state. We then randomly permute the date of the morning returns. This gives an instance of a random draw. From this, we can compute the fraction of dates finding themselves in the same group than the same date using close-to-close returns. We use the Monte Carlo method by repeating this draw 50000 times, and obtain a distribution with mean 18.4% and standard deviation 3.2%. Notice that the actual classification with 54.3% of similarity is far away from this range.

the return will be likely to be of the same sign at the close as in the morning, and hence make it more likely that the day remains in the same state. Notice that, although the classification is significantly better than a random draw, it will succeed only slightly more than half of the time to correctly predict the state at the end of the day.

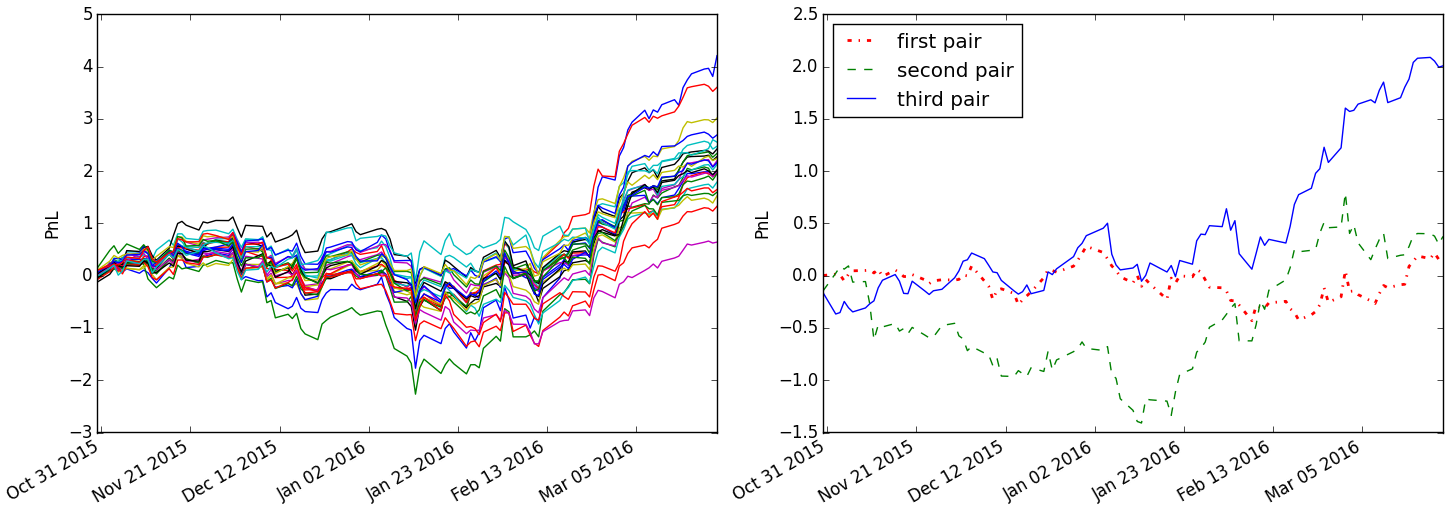
The strategy is built as follows: it considers a day ('today' thereafter) and looks at the returns of the stocks in the morning of this day. It also looks at the close-to-close returns in the past 400 days. These days, including today, are clustered into states, so that today belongs to a defined state. The assumption that we make is that the state will remain the same at the end of the day. The mean excess returns of the stocks for the days in today's state are computed: this provides us with an expectation of the excess returns for the close excess returns of today. If today's state contains less than 10 days, not counting today, we do not make a prediction, because we judge that basing a prediction on too few similar days is not very reliable.

The next step consists in computing the difference of the normalized excess returns of today with the normalized excess expected returns. By excess returns, we mean, as earlier, the returns over the market return. By normalized returns, we mean that we divide the returns by the standard deviations of the close-to-close returns over the past 400 days.

We call this difference the σ_{away} of a stock, which is how many standard deviations the stock is away form its mean in today's state. We classify the stocks according to their σ_{away}, in a decreasing order. The stocks with a positive σ_{away} can be considered as currently overperforming the state, while those with a negative σ_{away} can be seen as currently underperforming the state. We divide them into 25 equally weighted portfolios of 19-20 stocks, according to their σ_{away} ranking. The idea is that the assets shall revert to their usual behavior according to the defined state of today, so a portfolio of overperforming stocks should go down while a portfolio of underperforming stocks is expected to go up.

We open our positions at 10 am by investing 1 dollar in each portfolio, and close them at the close of the day. The next day, we move the 400-day window by one day to take into account the most recent days, and reiterate the strategy. Tracking the performance of each of the portfolios, whose components change every day, results in the profit-and-loss

curve seen in Figure 6.2a. If the strategy works, it is likely that we get clearly separated curves, with the overperformer portfolio performing the worst, and the underperformer portfolio performing the best. This is not what we obtain, meaning that at best, the idea has to be refined, or that at worst, that it does not work.



(a) Performance of the 25 portfolios.

(b) Performance of trading portfolios in long/short pairs.

Figure 6.2: Profit & Loss of 25 portfolios of 19 or 20 stocks, reconstructed every day, according to their σ_{away} ranking. The bid-ask spread and the transaction costs are not taken into account. **(a)**: The curves are generated by investing every day one dollar in each of the portfolios. The expected distinction of drift of the portfolios, according to their over/underperforming ranking, is not detected, making it difficult to implement the strategy. **(b)**: The PnLs are generated by going short one dollar in the portfolio ranked 1^{st} (overperformers) and long one dollar in the portfolio ranked 25^{th} (underperformers) for the 'first pair', short in the portfolio ranked 2^{nd} and long in the portfolio ranked 24^{th} for the 'second pair', and short in the portfolio ranked 3^{rd} and long in the portfolio ranked 23^{th} for the 'third pair'. The Sharpe ratios are respectively 0.4, 0.7, 3.2.

By refining it, we could think of not trading stocks for which there have been news reported on a particular day. In such a case, the move in the price could be large but would be explained by fundamental reasons rather than be revealing a statistically abnormal return. Therefore, we should not expect the price to revert to the expectation constructed from past data. The idea can be to avoid trading the portfolios at the extremes of the ranking, so that we do not trade stocks that have large abnormal returns because of an individual fundamental event. As shown in Figure 6.2b, by trading the portfolio of the most overperforming stocks (short) against that of the most underperforming stocks

(long), we get a Sharpe ratio [3] of 0.4, not taking into account any friction cost (bid-ask spread and transaction costs). If we trade the next (second) portfolio of overperforming stocks against the next (second) portfolio of underperforming stocks, we get a Sharpe ratio of 0.7. If we take the third layer portfolios, we get a Sharpe of 3.2. By going long and short in different portfolios provides us a hedge against a global market move. Although the last Sharpe is a good number, picking the third layer portfolios based on this number would be:

(a) overfitting, because there are no good reasons to select the third pair of portfolios. The portfolios are arbitrarily constructed with 20 stocks so the strategy may not work with another number of components, and such an in-sample optimization cannot be reliable (we can always find good Sharpe ratios in-sample);

(b) lower in-sample when we take into consideration the trading costs. If we assume a cost of 5 bps for a transaction (so 10 bps per day), we still get a Sharpe ratio of 3.0;

(c) statistically not reliable, because it only takes into account 140 round-trips.

Moreover, the scarcity of the data prevents us from conducting a proper out-of-sample backtest.

In this chapter, we attempted at finding predictability using the market states that our algorithm discovered. Our definition of market states however has not allowed us to find a signal to construct a profitable strategy. The predictability for the next day that we expected from Marsili (2002) was absent with our clustering algorithm, the considered period, and the considered stocks. In recent years, the expansion of algorithms trading in the market made it more efficient. Strategies that could have encountered success in the past may not work with more recent data.

We turned afterwards to an intraday strategy, assuming that we could predict the state of the day by extrapolating a snapshot of the state at 10 am. We found positive in-sample Sharpe ratios, but we were unable to make a judgment of their validity because we could not conduct a backtest by lack of data.

[3] We compute the Sharpe ratio as $\mu/\sigma \cdot \sqrt{(252)}$, where μ and σ are the mean and standard deviation of the daily returns.

Chapter 7

Conclusion

In this thesis, we have improved an unsupervised clustering algorithm, and have applied it to find communities of stocks and of days. This algorithm is based on the widely used method of modularity maximization, and applies the Louvain algorithm for this purpose. We have refined existing models (MacMahon and Garlaschelli, 2015) for quantifying the null hypothesis in the modularity for correlation matrices after conducting an in-depth analysis of random matrix theory and its parameters σ^2 and q. We have submitted an improvement on the process of the Louvain algorithm itself, by giving the possibility to a node to leave a community without joining another, if this move increases the modularity. Because the Louvain algorithm is a greedy algorithm that tries to take the best action at every step, this local optimization is reasonably expected to result in the increase of the modularity of the final partition. Besides, since the algorithm we have used on top of the Louvain algorithm proceeds iteratively by splitting formed communities into sub-communities, we have been able to circumvent the resolution limit which is one of the main drawbacks of modularity maximization.

Although the communities formed by the resulting partition of the data do contain similar instruments from a qualitative point of view, we have pointed out that a few instruments may be misplaced. By that we mean that two very similar instruments may belong to two different communities. We have suggested that this might be caused by the heterogeneity of the size of the communities, giving more attraction power to large communities, and may result in a node moving to a large community rather than

a smaller one, although it is more similar to the nodes of the latter. This raised the question of how to overcome this issue. We have proposed an idea to modify the measure of similarity between nodes by looking at the moves of nodes as force-driven moves. We have suggested that this force can be chosen in such a way so that it optimizes the communities from a qualitative point of view. We used a quadratic form of the modularity for other works in Nafora and found more intuitive communities. We leave the choice of this force as an open question for further research.

By applying the algorithm to cluster days, we have detected remarkably well-defined states, in which communities of stocks have distinctive behavior. We have found groups of days in which assets behave in the same way relative to each other. However, this model did not contain any predictive power, and we were not able to construct a trading strategy based on that, as we had expected from previous work (Marsili, 2002). The surge in the recent decade of the amount of trading conducted by algorithmic strategies may explain the absence of predictability in the more recent dataset we have studied: they have made market inefficiencies vanish more quickly. This has shown that, in this respect, markets have become more efficient.

Bibliography

Ankerst, M., Breunig, M. M., Kriegel, H.-P. and Sander, J. (1999). Optics: Ordering points to identify the clustering structure, *SIGMOD Rec.* **28**(2): 49–60.

Bachelier, L. (1900). Théorie de la spéculation, *Annales scientifiques de l'École Normale Supérieure* **17**: 21–86.

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities, *Journal of Political Economy* **81**(3): 637–54.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R. and Lefebvre, E. (2008). Fast unfolding of communities in large networks, *Journal of Statistical Mechanics: Theory and Experiment* **2008**(10): P10008.

Borghesi, C., Marsili, M. and Miccichè, S. (2007). Emergence of time-horizon invariant correlation structure in financial returns by subtraction of the market mode, *Phys. Rev. E* **76**: 026104.

Cont, R. (2001). Empirical properties of asset returns: stylized facts and statistical issues, *Quantitative Finance* **1**(2): 223–236.

Danielsson, J. (2011). *Financial Risk Forecasting: The Theory and Practice of Forecasting Market Risk with Implementation in R and Matlab*, Finance, Wiley.

Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise, *Kdd*, Vol. 96, AAAI Press, pp. 226–231.

Estivill-Castro, V. (2002). Why so many clustering algorithms: A position paper, *SIGKDD Explor. Newsl.* **4**(1): 65–75.

Fisher, R. A. (1915). Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population, *Biometrika* **10**(4): 507–521.

Fortunato, S. and Barthémy, M. (2007). Resolution limit in community detection, *Proceedings of the National Academy of Sciences* **104**(1): 36–41.

Fruchterman, T. M. J. and Reingold, E. M. (1991). Graph drawing by force-directed placement, *Softw. Pract. Exper.* **21**(11): 1129–1164.

Giada, L. and Marsili, M. (2001). Data clustering and noise undressing of correlation matrices, *Physical Review E* **63**(6): 061101.

Giada, L. and Marsili, M. (2002). Algorithms of maximum likelihood data clustering with applications, *Physica A: Statistical Mechanics and its Applications* **315**(3): 650–664.

Gómez, S., Jensen, P. and Arenas, A. (2009). Analysis of community structure in networks of correlated data, *Phys. Rev. E* **80**: 016114.

Good, B. H., Montjoye, Y.-A. d. and Clauset, A. (2010). Performance of modularity maximization in practical contexts, *Physical Review E* **81**(4): 046106.

IBM (2016). Bringing big data to the enterprise. Retrieved 2016-03-04.
URL: *http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html*

Jacomy, M., Venturini, T., Heymann, S. and Bastian, M. (2014). Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software, *PLoS ONE* **9**(6): 1–12.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R. and Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation, *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(7): 881–892.

Laloux, L., Cizeau, P., Bouchaud, J.-P. and Potters, M. (1999). Noise dressing of financial correlation matrices, *Phys. Rev. Lett.* **83**: 1467–1470.

Lempérière, Y., Deremble, C., Nguyen, T.-T., Seager, P. A., Potters, M. and Bouchaud, J.-P. (2014). Risk premia: Asymmetric tail risks and excess returns, *Available at SSRN 2502743* .

MacMahon, M. and Garlaschelli, D. (2015). Community detection for correlation matrices, *Phys. Rev. X* **5**: 021006.

Marsili, M. (2002). Dissecting financial markets: sectors and states, *Quantitative Finance* **2**(4): 297–302.

McGowan, M. J. (2010). The rise of computerized high frequency trading: Use and controversy, *Duke Law & Technology Review* **16**: 1.

Münnix, M. C., Shimada, T., Schäfer, R., Leyvraz, F., Seligman, T. H., Guhr, T. and Stanley, H. E. (2012). Identifying states of a financial market, *Nature - Scientific Reports* **2**: 644 EP.

Newman, M. E. (2004). Analysis of weighted networks, *Phys. Rev. E* **70**(5): 056131.

Newman, M. E. and Girvan, M. (2004). Finding and evaluating community structure in networks, *Physical review E* **69**(2): 026113.

Noack, A. (2004). *An Energy Model for Visual Graph Clustering*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 425–436.

Plerou, V., Gopikrishnan, P., Rosenow, B., Amaral, L. A. N., Guhr, T. and Stanley, H. E. (2002). Random matrix approach to cross correlations in financial data, *Phys. Rev. E* **65**: 066126.

Appendices

Appendix A

Code of the Modified Louvain Method

We share here the Python code that we developed to implement the central clustering algorithm used throughout this thesis. It is based on the Louvain method, but with the adequate modularity for correlation matrices, and with the additional possibility for a node to move to an empty community.

```
"""
Apply clustering after:
"Community detection for correlation matrices", Mel MacMahon, Diego Garlaschelli, 2015
http://arxiv.org/abs/1311.1924

Based on a modified Louvain method. Original Louvain method is from:
"Fast unfolding of communities in large networks", Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte,
Etienne Lefebvre, 2008
http://arxiv.org/pdf/0803.0476

The code is inspired by the Matlab implementation of the Louvain method written by Antoine Scherrer. It can be found at:
https://perso.uclouvain.be/vincent.blondel/research/louvain.html

The additional possibility, brought by this thesis, for a node to move to an empty community is implemented.
"""

import pandas as pd
import numpy as np
from joblib import Parallel, delayed

__author__ = 'Benjamin'
```

```
def cluster(m, verbose=True, shuffle=True, recursive=True, m_init=pd.DataFrame()):
    """ Detects communities by maximizing the modularity of the partition, using the modified Louvain method.
    :param m: Matrix of similarity (between days or stocks). Modified to account for noise and possibly the global mode.
    :type m: pandas.DataFrame
    :param verbose: If 1, the algorithm will print what it is doing.
    :type verbose: bool
    :param shuffle: Randomize the order in which the nodes are moved. This can yield different partitions.
    :type shuffle: bool
    :param recursive: 0, gives the first pass communities. 1, returns the partition with maximal modularity.
    :type recursive: bool
    :param m_init: Initial (non-modified) matrix. Used to compute the modularity. Does not impact the clustering.
    It only impacts the modularity as a scaling factor.
    :type m_init: pandas.DataFrame
    :return: List of dictionaries. The elements of the list contain a dictionary for one pass. Each dictionary contains:
    'COM': the community partition, 'SIZE': the size of each community, 'MOD': the modularity of the given partition,
    'Niter': the number of iterations of the pass, 'end': handles the case that m not possible to decompose.
    :type: list
    """

    assert isinstance(m, pd.DataFrame), "m must be a pandas.DataFrame"
    index = list(m.columns.values)
    if m_init.empty:
        m_init = m
    c_norm = float(np.abs(m_init).sum().sum())
    n = m.shape[0]

    ending = 0
    if float(c_norm) == 0. or n == 1:
        print "NO MORE DECOMPOSITION"
        ending = 1
        community = [{"end": ending}]
        return community

    n_iter = 0

    com = np.arange(n)  # Each node is put in an individual community at the beginning.
    # ------------------------------------------- Start of the first phase -------------------------------------------
    gain = True
    same_partition = True
    range_n = range(n)
    extra_community = range(n)  # extra community identifiers
    while gain:
        gain = False
        # randomize the order in which the nodes are moved. This can yield different partitions.
        if shuffle:
            np.random.shuffle(range_n)
        for i in range_n:
            com_i = com[i]  # community of stock i
            g = np.zeros(n) - 1  # gain vector
            best_increase = 0
            com_new = com_i
            com[i] = -1
            com_i_nodes = np.where(com == com_i)[0]  # array of the indices of the stocks in community i

            i_col = m.iloc[:, i].values

            c_ij1 = i_col[com_i_nodes]  # strength of the links of i in its previous community
```

```
        q_removed = - c_ij1.sum()/c_norm  # modularity gain from removing i from its previous community
        # if removing a node from its community increases the modularity,
        # then we consider the possibility of a move to an empty community
        extra = False
        if q_removed > 0:
            best_increase = q_removed
            # find empty community identifiers, and choose one at random
            free_com_id = np.array(extra_community)[np.in1d(extra_community, com, invert=True)]
            c_new_t = np.random.choice(free_com_id)
            extra = True

        nb = np.where(i_col)[0]  # take the neighbours of i (nodes with a non-zero link)
        if shuffle:
            np.random.shuffle(nb)  # it does not have to be shuffled. The result would only change if two target
        # communities yield the same modularity increase.
        for j in nb:
            com_j = com[j]
            if com_i != com_j and j != i:  # check all neighbours j of i except those in its own community (faster)
                if g[com_j] == -1:  # we do not consider several times the same community (faster)
                    com_j_nodes = np.where(com == com_j)[0]  # array of the indices of the stocks in community j

                    c_ij2 = i_col[com_j_nodes]  # strength of the links of i in the community of j
                    q_added = c_ij2.sum()/c_norm  # modularity gain from adding i in the community of j
                    q_increase = q_removed + q_added  # modularity gain from this operation
                    g[com_j] = q_increase

                    if q_increase > best_increase:
                        best_increase = q_increase
                        c_new_t = com_j  # temporary new community
                        extra = False
        # print str(i) + " best increase: " + str(best_increase)
        if best_increase > 0:  # if the maximum gain of modularity is positive, we change the community if i
            com_new = c_new_t  # new community is the one with the best increase in modularity
            if verbose:
                print "moved %d to %d" % (i, com_new)
                if extra:
                    print 'MOVED TO AN EXTRA COMMUNITY!'

        com[i] = com_new  # i is placed in the community for which the gain is maximum.
        # If no positive gain is possible, i stays in its original community.

        if com_new != com_i:  # If no community changed, we get out of the while loop.
            gain = True  # If any of the stock changed community, we stay in the loop.
            same_partition = False

    n_iter += 1
    if verbose:
        if gain:
            print "END OF ITERATION: %d" % n_iter
        else:
            print 'ITERATIONS ARE COMPLETED'

n_iter -= 1
com, com_size = reindex_communities(com)
community = [{"COM": pd.DataFrame(com, index), "SIZE": com_size, "MOD": modularity(com, m, c_norm), "Niter": n_iter,
              "end": ending, "same partition": same_partition}]
# community is a list of dictionaries. Each element of the list contains the information of one pass.
if verbose:
    print "MODULARITY IS: %.9f" % community[0]["MOD"]
```

```
if not recursive:
    return community
# ------------------------------------------------------------------------------------------------------------------------
# ------------------------------------------ Start of the second phase -----------------------------------------------
# ------------------------------------------------------------------------------------------------------------------------
else:
    m_new = m  # m_new will contain the strength of the links between the hyper-nodes,
    # ie. self-loops and renormalized interactions.
    com_current = com  # Community of the hyper-nodes.
    com_full = com  # Community of the single name nodes.
    k = 2
    if k == 2 and verbose:
        print "PASS NO.1 IS COMPLETED"

    while True:
        m_old = m_new
        n_node = m_old.shape[0]  # Number of hyper-nodes.

        com_unique = np.unique(com_current)

        n_com = len(com_unique)  # number of communities
        ind_com = np.zeros((n_com, n_node)) - 1  # The lines will contain the indices of the hyper-nodes in comm. i
        ind_com_full = np.zeros((n_com, n)) - 1  # The lines will contain the indices of the nodes in comm. i

        for i in xrange(n_com):
            ind = np.where(com_current == i)[0]
            ind_com[i, range(len(ind))] = ind
        for i in xrange(n_com):
            ind = np.where(com_full == i)[0]
            ind_com_full[i, range(len(ind))] = ind

        m_new = np.zeros((n_com, n_com)) - 1
        m_new = pd.DataFrame(m_new)
        for i in xrange(n_com):
            for j in xrange(i, n_com):
                ind1 = ind_com[i, :]
                ind2 = ind_com[j, :]
                ind1 = ind1[np.where(ind1 >= 0)].astype(int)  # nodes in community m
                ind2 = ind2[np.where(ind2 >= 0)].astype(int)  # nodes in community n
                m_new.ix[i, j] = m_new.ix[j, i] = m_old.ix[ind1, ind2].sum().sum()  # renormalized interactions

        # new communities of the hyper-nodes
        community_t = cluster(m_new, verbose, shuffle, recursive=False, m_init=m_init)
        end = community_t[0]["end"]

        if not end:
            same_partition = community_t[0]["same partition"]
            com_full = np.zeros(n) - 1
            com_current = community_t[0]["COM"].T.values[0]
            for i in xrange(n_com):
                ind1 = ind_com_full[i, :]
                ind1 = ind1[np.where(ind1 >= 0)].astype(int)
                com_full[ind1] = com_current[i]  # new communities of the single-name nodes

            com_full, com_size = reindex_communities(com_full)
            community_new = {"COM": pd.DataFrame(com_full, index), "SIZE": com_size,
                             "MOD": modularity(com_full, m, c_norm),
                             "Niter": community_t[0]["Niter"], "end": end, "same partition": same_partition}

            if same_partition:
```

```
                    if verbose:
                        print "identical segmentation => END\nTOTAL PASSES: %d" % (k - 1)
                    return community
                else:
                    community.append(community_new)
                    if verbose:
                        print "PASS NO. %d IS COMPLETED" % k
            else:
                if verbose:
                    print "EMPTY MATRIX OR NO MORE MERGING POSSIBLE => END"
                return community
            k += 1


def reindex_communities(com):
    """ Reindexes the communities according to their size. 0 is now the larger community.
    :param com: Array of communities.
    :return: com_reindex : Re-indexed array of communities.
    :return: com_size : Size of the communities, in decreasing order.
    """
    com_reindex = np.zeros(len(com)) - 1
    com_unique = np.unique(com)  # the sorted unique elements of com
    size = np.zeros(len(com_unique)) - 1
    for i in range(0, len(com_unique)):
        size[i] = len(com[np.where(com == com_unique[i])])  # number of elements in each community

    com_size = -np.sort(-size)  # sort the sizes in decreasing order
    idx = np.argsort(-size)  # indices that sort size to get com_sizes: size[idx] = com_size

    for i in range(0, len(com_unique)):
        com_reindex[np.where(com == com_unique[idx[i]])] = i  # assign the new identification number to communities

    com_reindex = com_reindex.astype(int)
    com_size = com_size.astype(int)

    return com_reindex, com_size


def modularity(com, m, c_norm):
    """ Computes the modularity of the partition.
    :param com: Array of communities.
    :param m: Matrix of link weights (matrix modified for noise and possibly global mode).
    :param c_norm: Sum of all the elements in the original matrix.
    :return: Modularity of the partition.
    """
    com_unique = np.unique(com)
    modularity_ = 0.
    for i in range(len(com_unique)):
        com_i = np.where(com == com_unique[i])[0]
        modularity_ += m.ix[com_i, com_i].sum().sum()
    modularity_ /= c_norm
    return modularity_


def cluster_iter(m, iterations, debug=1, shuffle=1, verbose=0, jobs=-1, recursive=True):
    """ The Louvain algorithm is sensitive to the order in which we pick the nodes to move.
    This can give different partitions.
    We iterate the algorithm and select the partition that give the largest modularity.
    :param m: Matrix (=network) to cluster.
    :type m: pandas.DataFrame
```

```
    :param iterations: Number of desired iterations.
    :type iterations: int
    :param shuffle: Randomize the order in which the nodes are moved. This can yield different partitions.
    :type shuffle: bool
    :param verbose: (bool)
    :param debug: (bool)
    :param jobs: Number of multiprocessors used. If you want to use other programs while running, set it to 1.
    :type jobs: int
    :return: Same as the cluster function.
    """

    param = [(m, verbose, shuffle, recursive) for _ in range(iterations)]

    cs = Parallel(n_jobs=jobs)(delayed(wrapper)(p) for p in param)
    mod = [cs[i][-1]["MOD"] for i in range(iterations)]
    t = np.where(mod == max(mod))[0]  # select largest modularity
    t = t[0]
    if debug:
        print "Modularities: " + str(mod)
        print "Position of max modularity: " + str(t)
    return cs[t]


def wrapper(param):
    return cluster(*param)
```

Appendix B

Predictability Plots

We show here three plots of the predictability for different states and type of returns. The three of them exhibit an absence of predictability for the next day.

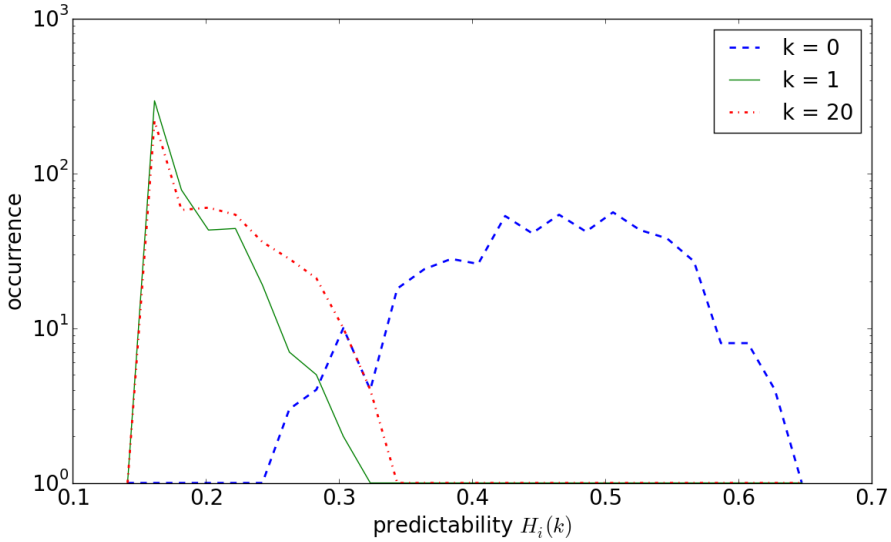


Figure B.1: Predictability for finer states. Instead of using 9 states, we split those that contain more than 100 days and obtain 21 states.

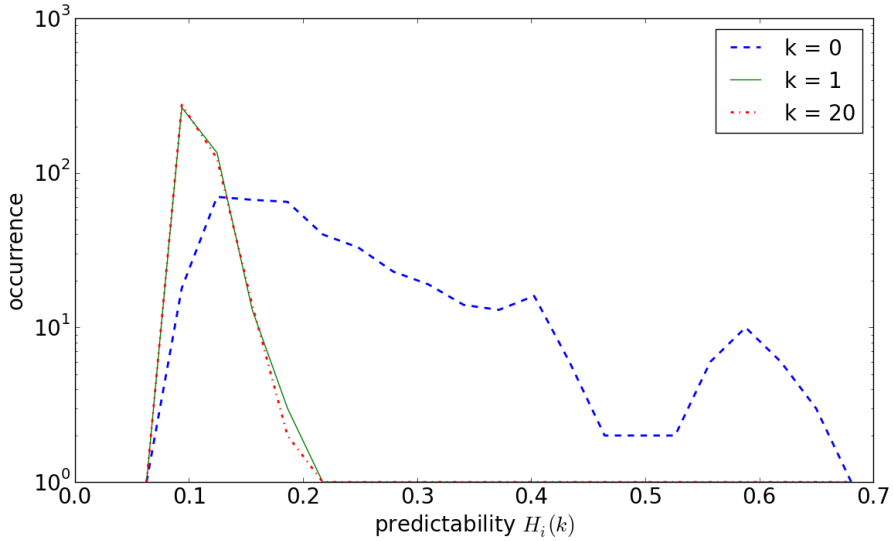


Figure B.2: Predictability of open-to-close returns. Instead of using end-of-day returns, we use open-to-close returns.

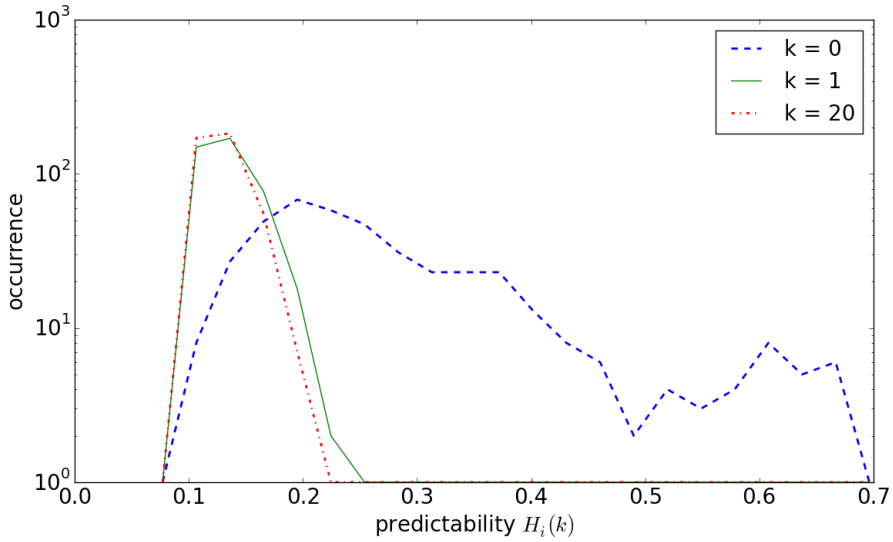


Figure B.3: Predictability of raw returns. Instead of using the excess returns $\tilde{r}$ over the market, we use the raw returns r.