



**UNIVERSITA' DEGLI STUDI DI PADOVA**

**DIPARTIMENTO DI SCIENZE ECONOMICHE ED AZIENDALI  
"M.FANNO"**

**DIPARTIMENTO DI SCIENZE STATISTICHE**

**CORSO DI LAUREA MAGISTRALE / SPECIALISTICA IN  
ECONOMICS AND FINANCE**

**TESI DI LAUREA**

**"TENSOR REGRESSION AND THE ROLE OF ASSETS  
INTERCONNECTIONS IN EQUITY PRICING"**

**RELATORE:**

**CH.MO PROF. : MASSIMILIANO CAPORIN**

**LAUREANDO/A: ENRICO PIERINI**

**MATRICOLA N. 1207026**

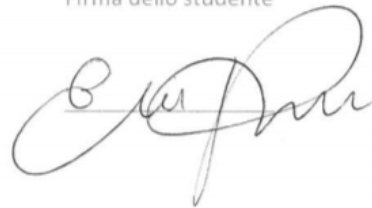
**ANNO ACCADEMICO 2019 – 2020**

Il candidato dichiara che il presente lavoro è originale e non è già stato sottoposto, in tutto o in parte, per il conseguimento di un titolo accademico in altre Università italiane o straniere.

Il candidato dichiara altresì che tutti i materiali utilizzati durante la preparazione dell'elaborato sono stati indicati nel testo e nella sezione "Riferimenti bibliografici" e che le eventuali citazioni testuali sono individuabili attraverso l'esplicito richiamo alla pubblicazione originale.

*The candidate declares that the present work is original and has not already been submitted, totally or in part, for the purposes of attaining an academic degree in other Italian or foreign universities. The candidate also declares that all the materials used during the preparation of the thesis have been explicitly indicated in the text and in the section "Bibliographical references" and that any textual citations can be identified through an explicit reference to the original publication.*

Firma dello studente

A handwritten signature in black ink, consisting of a large, stylized initial 'C' followed by several loops and a long horizontal stroke extending to the right.

I would like to thank several people that have helped me throughout the years:

-my Professor, who provided advice and help with the thesis

-my parents, who gave me the chance to study

-last but not least, my girlfriend Simona, who provided comfort and support whenever needed

I will especially thank my mother, with whom I have faced and won several adversities.

## **Index of Contents:**

- 1. Introduction**
- 2. Section 1: Background and Literature Review**
- 3. Section 2: Tensor Mathematics and Algebra**
- 4. Section 3: Data Description and Preprocessing**
- 5. Section 4: Global Dimensionality Reduction and Tensor Regression**
- 6. Section 5: Conclusions**
- 7. Appendix: Python Source Code**
- 8. References**

## **Introduction:**

### **Tensor Regression and the role of assets interconnections in equity pricing**

We live in challenging and interesting times; and it is indeed during these times that we start to discover how much more complex than expected the world is. Furthering one's knowledge of the complexity of the world around him or her makes it clear that the challenge requires appropriate instruments to explore the overwhelming abundance of information and data that can be gathered. Indeed, taking the new terrible calamity of Covid-19 as an example, data and exploration of it is the only way to face such new challenges. However, researches and scholars are often limited by several factors, including but not limited to:

-higher dimensional data;

-instruments to analyze efficiently the higher dimensional data;

Let us take a step back and think of why one should be concerned of exploring higher dimensional data and give a preliminary definition.

What we try to do when studying a phenomenon is reduce it to its essential parts, so that we can focus on what really matters; in a line, we build a model to describe the phenomenon. However, how informative the data is depends on several factors, especially the hidden links within the data. Typically, when considering the word "data", we think of a matrix or a vector and how the data within the matrix can be exploited to predict the behavior of the variables through the model. It is intuitive then to see that if we could "upgrade" the dimensions of the data, we could find new connections that previously could not be found. On the other hand, we also need the actual instruments to analyze such higher dimensional data. Indeed, this is the problem given by cutting-edge technological innovations such as Machine Learning, a field in which Tensors are the obvious choice. The aim of this work is to apply such data structures to price equity instruments. The reason why we do not want to use the usual instruments is because tensors are not feasibly treated by them; the only way would be to reduce their dimensions until one obtains matrices but then the interconnections among data are destroyed. This nullifies the effort.

Furthermore, whenever one speaks of instruments, he or she usually refers to actual tools but that is not the whole story. What really matters to understand the increasing complexity of the reality in which humanity lives is to develop some "cross-subject skills and competences". This is a kind of mantra that I repeat to myself every day. It is not enough to have teams with several incredibly skilled people without some overall general knowledge. I think that the best way to

explain my point is with an example: suppose that a development team is working on an innovative app for smartphones. There will be people devoted to the commercialization of the app and some others that will be devoted to the actual development. In my opinion, now days we cannot expect people from commercial to not have some degree of understanding of what programming is; at the same time, we cannot expect programmers to have little to no understanding of the economics behind a product. As it usually is in life, the answer is in the middle: developing some knowledge, even superficial, of other subjects is crucial, it is the key to the future, when the walls between subjects that we thought separate become thinner and thinner. Indeed, I tried to abide to this idea and decided to expand my horizons beyond what my course has thought me and enriching my background with several IT and quantitative skills. Thanks to the way the study plan could be organized, I was able to attend some courses from the Mathematical Engineering degree which were:

1. Scientific Computing and Object-Oriented Programming (or SCOOP in short);
2. Stochastic Methods for Finance (or SMF in short);
3. Systems Identification and Data Analysis (out of plan, SIDA in short);

I also attended another “out of plan” course from the Economics and Finance degree which was Banking: Advanced Risk Management.

SCOOP was crucial in the development of the skills needed for this work because it thought me the basics of Machine Learning, Object-Oriented Design and Programming, Algorithms and Data Structures. SMF gave me a lot of insight over the advanced equity pricing models and the mathematics to pick up where I left off during my courses. SIDA provided me with a comprehensive overview of the fields of Statistics, especially when it comes to the math behind Machine Learning models. Finally, Banking: Advanced Risk Management was especially useful for the intensive course in Technical Analysis which shed light over the tools and strategies used in the industry. With such a strong backbone, added to the already strong background in Finance and Economics, I decided to approach the concept of Tensors and apply it to the American stock market, in particular the 100 biggest companies in the Standard and Poor’s 500 equity index, where biggest stands for the highest market capitalization.

For what concerns the applicative part of the thesis, I chose to use the Python programming language, both for its interpretability and clearness but mostly because it is incredibly easy to find free (or extremely cheap) resources online to learn it. The majority of the pre-processing and data gathering has been done on my laptop, using PyCharm as text editor; the following will be a list of specs of my machine:

- CPU: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
- GPU: NVIDIA GeForce GTX 1050 2 GB
- RAM: Samsung 8 GB
- Storage: HDD 1 TB
- OS: Windows 10

Due to the low power of these components and the high computational requirements, the model build-up and computations were performed on the Google Colaboratory online service. It provides remote access to more powerful hardware than mine, while working in an interactive online Python notebook, all free of charge. There is also a premium version which, however, requires a subscription but thankfully I did not need one. It is a very useful virtual environment that comes with several third-party libraries already installed, especially TensorFlow. The latter is the free open-source library that Google developed and maintains to allow anyone to get started in the field of Machine Learning but also to provide a powerful tool for researches all around the globe. As the name suggests, the library is built upon Tensors but I did not use it; I decided to use another one called TensorLy, which I find more intuitive and easy to use.

The outline of this work will be as follows: Section 1 will give some background on these topics, together with some reviews of the literature considered, Section 2 will provide some Tensor mathematics and algebra, Section 3 will describe the data that has been used, Section 4 will be devoted to the description of the algorithms used to obtain the final model, Section 5 will present the results and draw some conclusions.



## Section 1: Background and Literature Review

I will start with some consideration over what is stated by the theory, especially the debate between the Efficient Market Hypothesis and the more recent findings in Behavioural Finance. In reviewing such topics, I will also explain the benefits of addressing these issues with tensor regression and other more complicated models. Indeed, stock movements are highly affected by the information available; the EMH states that there are several agents in the market that behave rationally, which means that changes in the price of stocks is driven only by “unemotional” determinants, namely changes in fundamental value. The agents are constantly updating their beliefs about the future value of their stocks which explains changes in prices; however, agents disagree on their forecasts so the stocks exhibit volatility, but still its value oscillates around the fundamental value. On the other hand, empirical evidence shows that not all agents are rational and stock prices can greatly differ from their intrinsic values, even for long periods of time; prices do not follow a random walk [Lo and MacKinlay 1988]. Recent behavioral finance studies have attributed the non-randomness of stock movements such as overreactions to unfavorable news to investors’ cognitive and emotional biases [Long et al. 1990; Shleifer and Vishny 1997]. Regardless, both agree on the fact that information plays a vital role. This leads to the intuitive idea that having tools that capture the heterogenic information are desirable; Qing Li et al.[2009] have addressed this topic in their work “A Tensor-Based Information Framework for Predicting the Stock Market”. They performed their analysis on the 100 most successful Chinese companies. Indeed they mention the statistical significance of sentiments, news sources, even Twitter messages, which all affect the agents’ decisions and thus propose a model to exploit the information conveyed in these mediums and how they interact with more quantitative data, as it has been studied in great detail by Frank and Antweiler [2004], Schumaker and Chen [2009b], Li et al [2014a, 2014b] and many others. However, very few works describe the joint effects of both quantitative and qualitative information sources. After all, it all boils down to analyzing non-linear connections among data, since linearity has been analyzed countless times, starting with the crucial work of Fama and French [1993]. Nowadays, there are several tools that capture the non-linear relationships among data, Tensors for example, but also Neural Networks and Machine Learning based models in general. Many of these tools were discovered several decades ago, for example the usage of Artificial Neural Networks dates back to something like 1994, as shown by Hsiao-Tien Pao [2008]. He was able to show that ANNs are able to outperform multiple regression analysis when it comes to predict debt ratios, or more in general funding decisions made by firms in Taiwan, during the period between 2000 and 2005. He used as reference several papers

that applied the Machine Learning methodology, and Neural Networks in particular, to address modeling issues in engineering problems back in 1994. His findings are consistent with the previous theory that essentially traces back to Myers [1977] regarding funding decisions of firms. Furthermore, Tensors can help in tracking some aspects that are usually discarded due to the assumption that all relationships in data are linear: such relationships are the so called network effects. In fact, Abalfazl Zareei [2019] showed how risk in portfolios can be originated from some underlying network structure, usually not captured by standard models. He estimates a VAR model and a star-like network structure where it is shown how diversification benefits are lower when there is some asymmetry in the network structure; indeed, the benefits reach the lowest point in the star-like case, which international markets and even U.S. industries have. Another remarkable work that stresses the usefulness of networks in capturing effects that established instruments cannot capture is contained in Andrea Buraschi and Paolo Porchia [2012]. They build up several models using several types of both symmetric and asymmetric networks, showing that, under imperfect information, the fact that a firm that occupies a more central position in the network, or network-centrality, is priced in the cross-section even after controlling for French and Fama factors. The effect of cross-sectional momentum was captured thanks to networks.

Furthermore, one has to mention that the usefulness of such models is greatly penalized by the high computational time needed to train a machine and the problem of scalability. Many times it becomes a burden to create a neural network because the number of parameters to estimate can get incredibly high incredibly fast; this hurts performance and requires powerful machines to obtain good results, all the while reducing generalizability due to the scarcity of data. Tensors are fundamental in building up the most complicated models specifically because their very nature allows to reduce immensely the computational taxation. I will develop further in Section 2 the next consideration but when thinking of a matrix, we usually think of it as a table; In linear algebra, a matrix is an linear object that can be multiplied on both sides; Tensors are linear objects that can be multiplied on additional sides, thus giving more freedom when creating the layers for Neural Networks, reducing the number of parameters to be estimated, the time required to train the model and giving overall advantages.

Such increases in dimensions are especially useful when modelling complex structures when both the number of samples and the number of features is so high that vectors and matrixes cannot capture the entirety of the phenomenon. A clear example is definitely the work of Dean P. Foster, Mark Liberman and Robert A. Stine[2013]. They use several methods to convert text into data which can be fed to the machine and understood by the model. In order to obtain good

predictors, they use several strategies that involve the use of conventional instruments but had a lot of data to work with. A better strategy that works extremely well even on small samples take advantage of the tensor data structure, as in the work of Deng Cai, et al. [2009]: Support Tensor Machines for Text Categorization. In their work, the authors compare the performance of the method that they developed to categorize text with other existing methods. They obtained better performances in terms of better predictions, especially in small samples, compared to usual ways. One could argue that the tasks at hand were different, since featuring text is very different from text categorization, but the analogy still holds because Tensors can be used to store features, not only to speed up calculations. Furthermore, it is a common task in Natural Language Processing and Feature Extraction from Text problems to look for ways that preserve the meaning of the documents. The idea is that one could naively count the number of words in a document and the associated word frequency, store it in a dictionary and then perform whatever analysis has to be done. This of course preserves the meaning of single words but not of entire periods or phrases. This means that the overall meaning of the document is destroyed since meaning also comes from semantics and lexicon, not only the frequency of the word in the text. A natural extension of this concept is to include “embeddings” instead of single words. However, the limit here is that a single embedding does not account for every possible one, thus forcing repetitions, which is wasteful in terms of more memory needed and duplicative information. With Tensors this problem is non-existent because the higher dimensionality allows storage and manipulation of data in ways previously unachievable. I will continue now with the analysis of the Mathematics and Algebra behind tensors.

## Section 2: Tensor Mathematics and Algebra

In this section I want to give some more quantitative background on the topic of Tensors and at the same time provide some explanation on how to use this formalism to deal with problems. In this section I was largely inspired by the work of Justin C. Feng in his *The Poor Man's Introduction to Tensors*, which was of great help to understand really complicated concepts due to the less formal way of explaining them. I should point out that this does not want to be a comprehensive review of the topic but just some starting point for the much broader concept of Tensors. After the *Mathematics behind Tensors*, I will continue by reviewing the common procedures on the topic that greatly helped me in the learning process. In this sense, I will follow the work of Kolda and Bader [2009]. It is by all means the most influential paper that I could find, cited by almost every other author that I have read. Indeed, it can be considered the “go-to” reference. What has really helped me understand this object was the definition provided by Justin C. Feng: “Tensors are objects that eat out something else and spit out scalars”. It is, of course, a very informal definition as I was mentioning earlier, but it made the idea stick in my mind because it immediately made it clear. The fact that Tensors give back scalars is of fundamental importance, in the sense that they behave as if they were scalars; and indeed scalars are very easy to manipulate, they are not subject to coordinate transformations, they are immediately interpretable and understandable, they make very tedious and long computations very easy. However, Tensors have the advantage of retaining much more information rather than a single number. Let us take a step back and proceed accordingly. In order to make it as clear as possible, I will start from more general concepts which are well established in linear algebra and then proceed onwards and link them with Tensors. The first thing to say is that all the geometry that I will discuss further in the text is in a space that respects the Euclidean rules. For now on, I will only refer to this concept as Euclidean space. I should also add that the assumption is maintained even when I try to build up my regression model. In my opinion, it is much easier to understand something if one has a firm grasp of the central ideas and then move on to the details and particular cases. To summarize in one paragraph (which also makes a little clearer the description of Tensors that I used a few lines ago):

“Tensors are mathematical constructs that eat a bunch of vectors and spit out a scalar. The main principle in Tensor analysis is that they are unaffected by coordinate transformations. This means that any equation written in Tensor form is valid in any coordinate system”.

This looks very trivial; but indeed it is the only thing that one has to keep in mind. Of course there are other details that I have skipped, such as the fact that Tensors are linear maps and also that they can “eat” other things called dual vectors, but I will explain everything further down

the way. One thing to always keep in mind is that the index placement in Tensors is of fundamental importance. Since the order of dimensions increases, we need some way to identify each dimension. This is true also when considering vectors or matrixes. In order to identify the objects stored into them, we use indexes. However, we need to make an important difference which is between raised indexes and lowered indexes. They just refer to the position of the index with respect to the symbol used to identify a Tensor. For example:

$$v^i \tag{2.1}$$

In this case, “*i*” would be a raised index. Unless otherwise specified, exponents will not be signs of exponentiation but rather raised indexes. Of course it can be done for matrices as well. Speaking of matrices, one element that is crucial to the topic is the so called Kronecker delta; it is just the components of the identity matrix and it will be indicated by the following symbol:  $\delta_j^i$ . In this case, it will indicate a squared *ixj* matrix; if both *i* and *j* are superscripts or subscripts, it will indicate an element of the matrix. I am putting so much effort into this part because this brings me to a fundamental concept: the Einstein summation convention. It is immediately summed up in a short sentence:

Any time there is a lowered and a raised index, written with the same symbol, a summation is implied.

So for example:

$$M_j^i v^j \tag{2.2}$$

means that the matrix *M* is multiplied on the left of the vector *v*. By Einstein summation convention a sum over the index *j* is implied. I can also make the summation explicit:

$$\sum_{j=1}^n M_j^i v^j \tag{2.3}$$

This becomes very useful in the realm of Tensors, because there will be almost every time a pair of indexes over which one can sum and simplify equations.

As promised previously, I will start from more familiar concepts in linear algebra and then continue towards Tensors.

The most elementary linear object that I can think of is a vector. Later on, it will become very clear that a vector can be considered a particular case of Tensor. We are all acquainted with the concept of a vector: a structure that stores data in each position, it can be a column or a row and it has dimensions  $i \times 1$  (in case of a row vector) or  $1 \times j$  (in case of a column vector). In

computer science, a vector is basically an array that has only 1 dimension. Here I propose a different way to think of vectors, borrowing notation from physics: a vector is a directional derivative.

For example:

$$\vec{v} \cdot \vec{\nabla} f(x^\alpha) = v^i \frac{\delta f}{\delta x^i} \quad (2.4)$$

where  $x^\alpha$  represent the Cartesian coordinates in Euclidean space. In Einstein convention, the index  $i$  in the partial derivative is considered lowered. Removing  $f$ , I obtain the directional derivative operator. Suppose  $i$  is in range of 1 to 3:

$$\vec{v} \cdot \vec{\nabla} = v^i \frac{\partial}{\partial x^i} \quad (2.5)$$

$\leftrightarrow$

$$\vec{v} \cdot \vec{\nabla} = v^1 \frac{\partial}{\partial x^1} + v^2 \frac{\partial}{\partial x^2} + v^3 \frac{\partial}{\partial x^3} \quad (2.6)$$

Now I can rewrite the vector  $v$  explicitly using orthogonal unit vectors  $\widehat{e}_\alpha$ :

$$\vec{v} = v^i \widehat{e}_i \quad (2.7)$$

$\leftrightarrow$

$$\vec{v} = v^1 \widehat{e}_1 + v^2 \widehat{e}_2 + v^3 \widehat{e}_3 \quad (2.8)$$

Looking at the previous example, it appears clear that partial derivatives can be thought of as basis vectors, that is why I claim that vectors can be considered directional derivatives. I should also mention that the basis of partial derivatives is the coordinate basis.

The last thing to say to prove that a vector is a directional derivative operator is that such operator contains the same information contained in the explicit components of the vector. If I use the trivial function  $f(x) = x^3$  (where 3 is an index, not an exponent) and feed it to the directional derivative operator:

$$\begin{aligned}\hat{v} \cdot \vec{\nabla} x^3 &= v^i \frac{\partial x^3}{\partial x^i} = \\ &= v^1 \frac{\partial x^3}{\partial x^1} + v^2 \frac{\partial x^3}{\partial x^2} + v^3 \frac{\partial x^3}{\partial x^3}\end{aligned}\tag{2.9}$$

Coordinates are independent of each other, so the partial derivative for  $i \neq j$  is zero, while it is equal to one for  $i = j$ . This means that I can write:  $\frac{\partial x^i}{\partial x^j} = \delta_j^i$  where delta is the Kronecker delta.

The previous equation becomes then:

$$\vec{v} \cdot \vec{\nabla} x^3 = v^i \partial_i^3 = v^3\tag{2.10}$$

This means that all I have to do to pick out a component of a vector is feed the corresponding coordinate to the directional derivative operator, in fact I could define the components of the vector in the following way:

$$v^i := \vec{v} \cdot \vec{\nabla} x^i\tag{2.11}$$

Now I will drop the arrow to be remark that  $v$  is not a vector but an operator so now I can write:

$$v^i = v(x^i)\tag{2.12}$$

where the right-hand side is the directional derivative operator acting on the coordinate  $x^i$ . These are some important building blocks and I will provide some more in the following lines, starting from the dot product. I can define the latter as:

$$u \cdot v = \delta_{ij} u^i v^j\tag{2.13}$$

where delta is the Kronecker delta and it is equal to one for  $i=j$  and zero otherwise. Indeed the dot product is a particular case of the more general inner product, which can be written in terms of a quantity that we define as the metric. If I call such quantity  $g_{ij}$ :

$$\langle u, v \rangle = g_{ij} u^i v^j\tag{2.14}$$

Since inner products are symmetric, we require the existence of an “inverse metric” defined as the solution to:

$$g^{ik} g_{kj} = \delta_j^i\tag{2.15}$$

From this, it appears clear that the metric in Cartesian coordinates in a Euclidean space is just the Kronecker delta. Before moving on, I want to remind the properties of the inner product: given the vectors  $p$ ,  $u$  and  $v$  and scalar  $a$ , we define the inner product between two vectors on the following four properties:

$$\text{I. } \langle u + p, v \rangle = \langle u, v \rangle + \langle p, v \rangle\tag{2.16}$$

$$\text{II. } \langle au, v \rangle = a\langle u, v \rangle \quad (2.17)$$

$$\text{III. } \langle u, v \rangle = \langle v, u \rangle \quad (2.18)$$

$$\text{IV. } \langle v, v \rangle \geq 0 \text{ for all } v \quad (2.19)$$

The meaning behind the metric is immediately grasped when we start to talk about the basis vectors and how they are subject to the same rules that define the inner product. Recall that

$$v = v^i e_i \quad (2.20)$$

which means that I can always write a vector as the product of its components by the  $i$  linearly independent basis vectors. I am assuming neither that they are orthogonal nor that they are of unit length. Since the basis vectors are indeed vectors, I can take their inner product and obtain the following:

$$g_{ij} = \langle e_i, e_j \rangle \quad (2.21)$$

which is the metric tensor. Thus metric components are just inner products between basis vectors:

$$g_{ij} = \left\langle \frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j} \right\rangle \quad (2.22)$$

This digression was necessary to deliver the punchline that the metric/inner product can be thought of as a linear machine that eats two vectors and spits out a scalar, which closely resembles the initial definition given of a tensor. Now that I have defined vectors, I will take some time to introduce covariant vectors, or dual vectors, as I will refer to them in this way.

The best way to describe them is by taking into account their duality with vectors; indeed you could think of dual vectors as elements that take vectors and give back scalars, just as vectors take dual vectors and give back scalars. One can be very confused by the apparent ambiguity but the real difference comes in the coordinate basis. In fact, the natural set of coordinate basis for dual vectors are differentials, while partial derivatives are the natural set of coordinate basis for vectors. This means that I can write a dual vector as:

$$w = w_i dx^i \quad (2.23)$$

The reason why this is true comes from the differential of a function, which is an example of a dual vector. The differential is defined as:

$$df := \frac{\partial f}{\partial x^i} dx^i \quad (2.24)$$

The components of the differential are just the components of the gradient of the function. Now I can combine everything seen so far in just three equations:



$$v = v^i \frac{\partial}{\partial x^i} \quad (2.25)$$

$$w = w^i dx^i \quad (2.26)$$

$$\langle v, w \rangle = \langle w, v \rangle = w^i v_i = v^i w_i \quad (2.27)$$

Substituting 2.25) and 2.26) in 2.27) one obtains:

$$\langle dx^j, \frac{\partial}{\partial x^i} \rangle = \langle \frac{\partial}{\partial x^i}, dx^j \rangle = \delta_j^i \quad (2.28)$$

Finally, I will give the exact definition of a tensor but most importantly I will deliver some further consideration to the principle that the value of a scalar function at a point is unaffected by coordinate transformations. This is because coordinates are just labels that we give to points in space. A remark is necessary: this is true for scalar functions, which are functions of points, not coordinates. This means that a coordinate transformation can change how a scalar function depends on coordinates but the dependence on points remains unaffected. This then relates to vectors when we consider the fact that vectors are geometric quantities. For instance, they have order and magnitude, borrowing definitions from physics, and how we represent them depends on coordinates. However, they are just labels, geometry does not care how we call points in space. Intuitively, this means that coordinate transformations cannot change the meaning of an inner product or a directional derivative; they both act on a function and yield scalars but the value of a scalar in a point should not depend on the coordinates that we use. On the other hand, the values of vectors and vector components and the values of the gradient components do change. This is what is really interesting: how do components of vectors change under coordinate transformations?

To answer this question, we can take a function  $y^\alpha(x^\alpha)$  which we assume to be invertible so that we can write  $x^\alpha(y^\alpha)$ . I am using the Greek letters to indicate the new coordinates, while I will use Latin letters to indicate the old coordinates. Taking derivatives of these two functions (the quantities  $\frac{\partial x^j}{\partial y^\beta}$  and  $\frac{\partial y^\beta}{\partial x^j}$  respectively), I find the transformation matrices and then the chain rule tells me that they are related in the following way:

$$\frac{\partial x^i}{\partial y^\alpha} \frac{\partial y^\alpha}{\partial x^j} = \frac{\partial x^i}{\partial x^j} = \delta_j^i \quad (2.29)$$

$$\frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^i}{\partial y^\beta} = \frac{\partial y^\alpha}{\partial y^\beta} = \delta_\beta^\alpha \quad (2.30)$$

The last equality in both (2.29) and (2.30) comes from the fact that coordinates are independent of each other. Furthermore, the chain rule tells us that the gradient behaves as:

$$\frac{\partial f}{\partial y^\beta} = \frac{\partial x^j}{\partial y^\beta} \frac{\partial f}{\partial x^j} \quad (2.31)$$

Since the gradient forms the components of the dual vector, the following law of transformation for the components of the dual vector is suggested:

$$w_\beta = \frac{\partial x^j}{\partial y^\beta} w^j \quad (2.32)$$

Then, if  $v$  is a vector, the value of  $w(v) = w_i v^i$ , being it a scalar, cannot be affected by a coordinate transformation, which suggests that the transformation law for a vector must be opposite to the transformation law of the dual vector. Indeed:

$$v^\alpha = \frac{\partial y^\alpha}{\partial x^i} v^i \quad (2.33)$$

$$\begin{aligned} w(v) &= w_\alpha v^\alpha = \frac{\partial x^j}{\partial y^\alpha} w^j \frac{\partial y^\alpha}{\partial x^i} v^i = \\ &= \frac{\partial x^j}{\partial y^\alpha} \frac{\partial y^\alpha}{\partial x^i} w^j v^i = \partial_j^i w^j v^i \\ &= w_j v^i \end{aligned} \quad (2.34)$$

$$w_\alpha v^\alpha = w_i v^i \quad (2.35)$$

Also inner products are invariant under coordinate transformations, which means that also the metric tensor must have its own transformation law. Not only that, but the requirement is also that there exist an inverse tensor metric since the inner product is commutative. Indeed:

$$g_{\alpha\beta} = \frac{\partial x^i}{\partial y^\alpha} \frac{\partial x^j}{\partial y^\beta} g_{ij} \quad (2.36)$$

$$g^{\alpha\beta} = \frac{\partial y^\alpha}{\partial x^i} \frac{\partial y^\beta}{\partial x^j} g^{ij} \quad (2.37)$$

At this point, I can finally give a proper definition of a tensor:

A tensor is a linear map that acquires vectors and/or dual vectors and gives as result scalars.

By linear map, I mean that the tensor must be a linear function of what it acquires, it is linear in each of its arguments and also vanishes if it is fed a vector or a dual vector of zeros. The latter is a less formal way of saying that a tensor is a homogenous function (of degree one). These conditions imply that a tensor that acquires only one vector  $v$  and only one dual vector  $w$  must have the following form:

$$T(w, v) = T_j^i w_i v^j \quad (2.38)$$

For  $T(w, v)$  to be a scalar, it must be that its components transform in following way:

$$T_\beta^\alpha = \frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^j}{\partial y^\beta} T_j^i \quad (2.39)$$

At this point, there is a clear pattern emerging from the equations, which is that raised indices of tensors transform like vector indices while lowered indices transform like dual vector indices.

I will introduce now another important concept, which is the rank of the tensor. It is basically the number of vectors and dual vectors that the tensor acquires. In fact, scalars can be considered rank-0 tensors, vectors can be considered rank-1 tensors and matrices can be viewed as rank-2 tensors. There is no upper limit to the number of vectors and dual vectors that a tensor can acquire.

Now I will remark the reason why I decided to go through all of this explanation:

Tensor equations look the same in all coordinate systems.

As an example, we can consider the following equation:

$$G_j^i = \gamma T_j^i \quad (2.40)$$

where gamma is just a constant and G and T are rank-2 Tensors. Transforming the coordinates:

$$T_\beta^\alpha = \frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^j}{\partial y^\beta} T_j^i \quad (2.41)$$

$$G_\beta^\alpha = \frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^j}{\partial y^\beta} T_j^i \quad (2.42)$$

$$\frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^j}{\partial y^\beta} G_j^i = \gamma \frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^j}{\partial y^\beta} T_j^i \quad (2.43)$$

$$G_\beta^\alpha = \gamma T_\beta^\alpha \quad (2.44)$$

Having defined some basics in terms of Tensor Algebra, it is now the case to move forward and describe some properties of Tensor Calculus. The latter is a little bit trickier because, counter-intuitively, partial derivatives of tensors do not transform like tensors. To show this claim, consider the following quantities:

$$A_j^i := \frac{\partial v^i}{\partial x^j} \quad (2.45)$$

And then the counterpart in the coordinates  $y^\alpha$ :

$$A_\beta^\alpha = \frac{\partial v^\alpha}{\partial y^\beta} \quad (2.46)$$

If  $A_j^i$  and  $A_\beta^\alpha$  are components of a tensor, they would be related by a tensor transformation law, as it was for the following:

$$T_\beta^\alpha = \frac{\partial y^\alpha}{\partial x^i} \frac{\partial x^j}{\partial y^\beta} T_j^i \quad (2.47)$$

However, this is not the case, in fact if I plug the formula  $v^\alpha = \frac{\partial y^\alpha}{\partial x^i} v^i$  into  $A_\beta^\alpha$ , I get:

$$\begin{aligned} A_\beta^\alpha &= \frac{\partial}{\partial y^\beta} \left( \frac{\partial y^\alpha}{\partial x^i} v^i \right) = \\ &= \frac{\partial x^j}{\partial y^\beta} \frac{\partial}{\partial x^j} \left( \frac{\partial y^\alpha}{\partial x^i} v^i \right) = \frac{\partial x^j}{\partial y^\beta} \frac{\partial y^\alpha}{\partial x^i} A_j^i + \frac{\partial x^j}{\partial y^\beta} \frac{\partial^2 y^\alpha}{\partial x^j \partial x^i} v^i \end{aligned} \quad (2.48)$$

The formula above looks very similar to the tensor transformation law but there is an additional component. We could place constraints on the function  $y^\alpha(x^\alpha)$  to make the second derivative of  $y$  equal to zero, but it is better to look for some correction term so that we do not need to add constraints. One way to do it is to construct a new derivative operator that reduces to the usual partial derivative in Cartesian coordinates. Since the term that we want to eliminate depends on  $v$ , we would like something that depends on  $v$  and that makes the additional term vanish when we derive. This leads to the definition of the so called covariant derivative which acts on  $v$  in the following way:

$$\nabla_j v^i = \frac{\partial v^i}{\partial x^j} + \Gamma_{jk}^i v^k \quad (2.49)$$

Capital gamma stands for the so called connection coefficients. The trick is that the connection coefficients do not transform as tensors, instead they transform as:

$$\Gamma_{\beta\gamma}^{\alpha} = \left( \frac{\delta y^{\alpha}}{\delta x^i} \frac{\delta x^j}{\delta y^{\beta}} \frac{\delta x^k}{\delta y^{\gamma}} \right) \Gamma_{jk}^i - \left( \frac{\delta x^j}{\delta y^{\beta}} \frac{\delta^2 y^{\alpha}}{\delta x^j \delta x^i} \right) \frac{\delta x^i}{\delta y^{\gamma}} \quad (2.50)$$

It is immediate to notice how the negative term is exactly what we need to make the additional term in the tensor derivative formula disappear. All that is needed is that  $\Gamma_{jk}^i = 0$ . Everything can be generalized to an arbitrarily high number of correction terms, in fact I will present the covariant derivative formula for a rank-3 tensor:

$$\nabla_k Q^{ijl} = \frac{\delta Q^{ijl}}{\delta x^k} + \Gamma_{km}^i Q^{mj l} + \Gamma_{km}^j Q^{i m l} + \Gamma_{km}^l Q^{i j m} \quad (2.51)$$

So far, I have only defined the covariant derivative in the case of vectors, now I must define the covariant derivative for dual vectors, which I expect to be as:

$$\nabla_k w_i = \frac{\delta w_i}{\delta x^k} + C_{ki}^j w_j \quad (2.52)$$

Recall that the quantity  $v^i w_i$  is a scalar. The covariant derivative acting on a scalar is just the partial derivative:

$$\nabla_k (v^i w_i) = \frac{\partial (v^i w_i)}{\partial x^k} = w_i \frac{\partial v^i}{\partial x^k} + v^i \frac{\partial w_i}{\partial x^k} \quad (2.53)$$

where the product rule has been used in the second equality. Now a good definition for a derivative operator should respect the Leibniz rule, so I want the covariant derivative for dual vectors to be consistent with the following equation:

$$\nabla_k (v^i w_i) = v^i \nabla_k w_i + w_i \nabla_k v^i \quad (2.54)$$

From which I can derive the following:

$$\nabla_k (v^i w_i) - \nabla_k (v^i w_i) = w_i \Gamma_{kj}^i v^j + v^j C_{kj}^i w_i = 0 \quad (2.55)$$

For the latter to be true, I require that  $\Gamma_{kj}^i = -C_{kj}^i$ , which gives me the formula for the covariant derivative of dual vectors:

$$\nabla_k w_i = \frac{\partial w_i}{\partial x^k} - \Gamma_{ki}^j w_j \quad (2.56)$$

Then, as before, one can infer the next terms for higher ranks.

What should be clear now is that the covariant derivative of a tensor transforms as a tensor, which means that we can write derivatives in a form that looks the same in all the coordinate systems.

Now that I have given some introductory notions on tensors, I will proceed and review several methodologies for their manipulation. The majority of what I am going to expose in the next lines is available on the paper by Tamara G. Kolda, Brett W. Bader. [2009]. As I mentioned earlier, this was a fundamental starting point for this work because the authors did a tremendous job in acquiring and storing in one single paper the various methodologies, giving a comprehensive view of the topic. Also some pictures and examples are borrowed from the same paper.

Now I will be a little more formal, because the topic requires it. A tensor is a multidimensional array. More formally, an N-way or Nth-order tensor is an element of the tensor product of N vector spaces, each of which has its own coordinate system. The usability of tensor is extensive and several authors have taken advantage of them. Fields such as psychometrics and chemometrics have been the first to use them and since then a great deal of interest spawned among several other communities, such as the signal processing one, or the data mining field of research. For a complete overview for the literature, there is an extensive list of papers, theses and books in the references section of Kolda and Bader [2009], even of older publications which are hard to find. Another preliminary remark to make is that now I will provide some additional notation, so that I can switch to something less heavy and more readable with respect to what I have used so far. Starting from the very definition, every tensor has an order (or rank, in some papers) which is just the number of dimensions that it has. Even scalars can be considered particular cases of tensors, because they are rank-0 tensors. In the following lines, I will use the boldface lowercase letter to indicate vectors (or rank-1 tensors), capitalized boldface letters for matrixes (or rank-2 tensors, even though an order-2 tensor would be more appropriate, to avoid confusion with the concept of the maximum number of linearly independent columns of a matrix) and higher-order tensors will be directly specified.

Scalars are denoted by lowercase letters, e.g., “a”. The i-th entry of a vector  $\mathbf{a}$  is denoted by  $a_i$ , element (i, j) of a matrix  $\mathbf{A}$  is denoted by  $a_{ij}$ , and element (i, j, k) of a third-order tensor  $\mathbf{X}$  is denoted by  $x_{ijk}$ . Indices typically range from 1 to their capital version, e.g.,  $i = 1, \dots, I$ . The nth element in a sequence is denoted by a superscript in parentheses, e.g.,  $\mathbf{A}(n)$  denotes the n-th matrix in a sequence. Subarrays are just subsets of indexes. The classical example is a column or a row for a matrix. Fibers are a little bit trickier, because they are defined by fixing every

index but one and are the higher order equivalent of matrix rows and columns. Slices are the two-dimensional sections of a tensor, defined by fixing all but two indexes. As it is possible for a matrix or a vector, one can compute the norm of a tensor by summing the squares of all of its elements and then taking the square root of the result. Squaring the norm is equal to computing the inner product of a tensor with itself. From now on I will refer to the number of dimensions as order of the tensor, because I will introduce the  $n$ th order rank-1 tensor:

$$\mathbf{X} = a_1 \circ a_2 \circ \dots \circ a_N \quad (2.57)$$

So, for example  $\mathbf{X} = a \circ b \circ c$  is a third-order rank-one tensor.

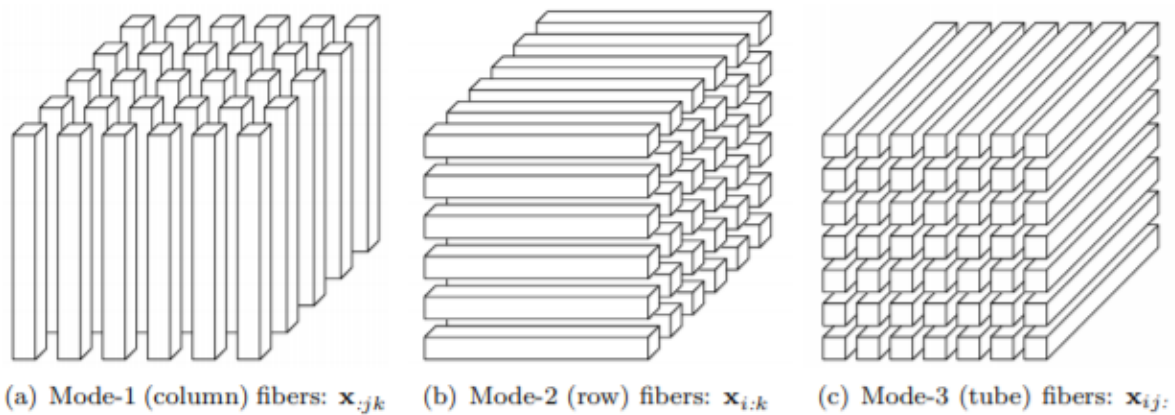


Fig. 1 fibers for each mode of a 3-way Tensor (Kolda and Balder [2009])

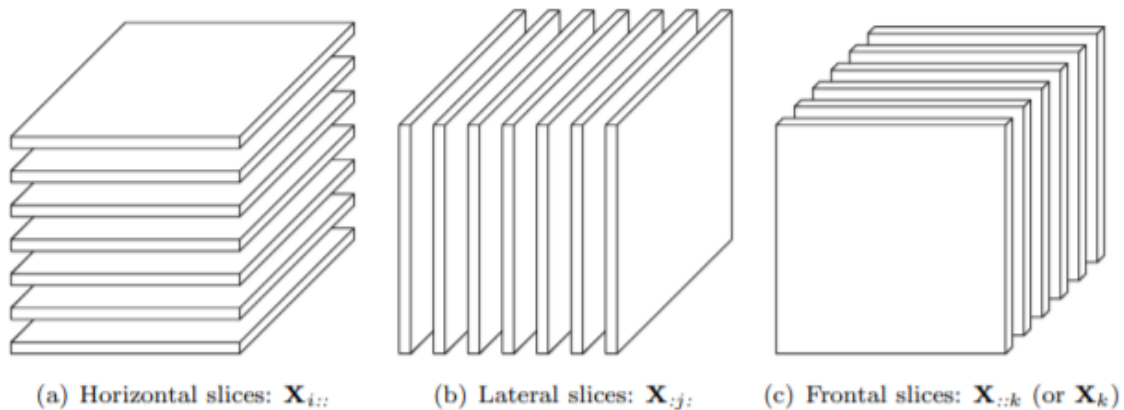


Fig. 2 slices of a 3-way tensor (Kolda and Balder [2009])

Another property that tensors can have is symmetry, which can be total or partial, according to the number of modes. If all the modes are symmetric then the symmetry is defined as

supersymmetry while if two or more modes are symmetric then it is just called symmetry. The property of supersymmetry is of three way tensors and is found when any permutation of the indexes does not impact the elements of the tensor. A tensor can also be diagonal, if the elements on the diagonal are different from zero. It is best explained with a picture\:

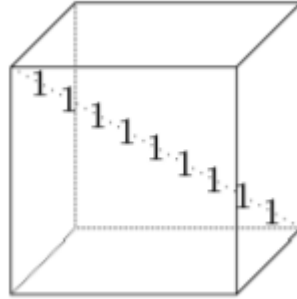


Fig. 3 Superdiagonal 3-way Tensor (Kolda and Balder [2009])

*A tensor  $X \in R^{I_1 \times I_2 \times \dots \times I_k}$  is diagonal if  $x_{i_1 i_2 \dots i_k} \neq 0$  only if  $i_1 = i_2 = \dots = i_k$*

There is also the process of matricization, also known as unfolding or flattening, where the elements of the tensor are rearranged into a matrix. This can be done for any order tensor. The notation is a bit clunky so an example best describes this element. Tensor element  $(i_1, i_2, \dots, i_N)$  maps to matrix element  $(i_n, j)$ , where:

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^N (i_k - 1) J_k \text{ with } J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m \quad (2.58)$$

The concept is easier to understand with an example. Let the frontal slices of the tensor  $X \in R^{3 \times 4 \times 2}$  be

$$X_1 = \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} \text{ and } X_2 = \begin{pmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{pmatrix}$$

Then the three mode-n unfoldings are:

$$X_{(1)} = \begin{pmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \end{pmatrix}$$

$$X_{(2)} = \begin{pmatrix} 1 & 2 & 3 & 13 & 14 & 15 \\ 4 & 5 & 6 & 16 & 17 & 18 \\ 7 & 8 & 9 & 19 & 20 & 21 \\ 10 & 11 & 12 & 22 & 23 & 24 \end{pmatrix}$$



$$X_{(3)} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \dots & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & \dots & 20 & 21 & 22 & 23 & 24 \end{pmatrix}$$

Of course there is also the process of vectorizing a tensor, which is very intuitive, after having seen the process of unfolding.

As it is done for matrixes and vectors, also tensors can be multiplied together. The mindset to have is the same of matrix multiplication: we are used to think of matrices as linear operators that can be multiplied either on the right or on the left, provided that dimensions agree; with tensor we have the same thing but now there are other ways to interact with this linear object and that is why we speak of n-Mode product. According to the mode in interest, the product is defined as the following:

$$(X \bar{x}_n v)_{i_1 \dots i_{n-1} i_{n+1} \dots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \dots i_N} v_{i_n} \quad (2.59)$$

This concept is crucial when considering the change of basis in the case when a tensor defines a multilinear operator.

A fundamental concept in linear algebra is the fact that matrix multiplication has been defined in several ways according to the purpose. That is why we speak of Kronecker, Khatri-Rao and Hadamard products, which I will briefly define, before moving on.

The Kronecker product is defined as  $A \otimes B$  where A is  $I \times J$  and B is  $K \times L$ . The result is a matrix of dimensions  $(IK) \times (JL)$ . An example should clarify everything:

$$\begin{aligned} A \otimes B &= \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1J}B \\ a_{21}B & a_{22}B & \dots & a_{2J}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}B & a_{I2}B & \dots & a_{IJ}B \end{pmatrix} = \\ &= (a_1 \otimes b_1 \ a_1 \otimes b_2 \ \dots \ a_J \otimes b_{L-1} \ a_J \otimes b_L) \end{aligned} \quad (2.60)$$

Then there is the Khatri-Rao product, or “matching columnwise” Kronecker product. Given matrices  $A \in R^{I \times K}$  and  $B \in R^{J \times K}$  then the Khatri-Rao product is denoted by  $A \odot B$ . The resulting matrix will have dimensions  $(IJ) \times K$ :

$$A \odot B = (a_1 \otimes b_1 \ a_2 \otimes b_2 \ \dots \ a_K \otimes b_K) \quad (2.61)$$

There is a particular case when the Khatri-Rao and the Kronecker products are equal, and that is when the matrices are vectors. Finally, the Hadamard product is the elementwise matrix product. Given matrices A and B, both of size  $I \times J$ , the Hadamard product is denoted as  $A * B$  and the result is:

$$A * B = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \dots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \dots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \dots & a_{IJ}b_{IJ} \end{pmatrix} \quad (2.62)$$

Since the main difficulty of using Tensors is the potential higher dimensionality of problems, many times it is mandatory to use a dimensionality reduction technique. There are fundamentally two main tensor decompositions: the CP and the Tucker decomposition. CP stands for CANDECOMP/PARAFAC, which themselves stand for Canonical (CAN) Decomposition (DECOMP) and Parallel (PARA) Factors (FAC) and the Tucker decomposition from the scholar who formalized it. They both have strengths and weaknesses and the choice of which to use, as usual, comes down to the objective that the user is pursuing. It will become clearer as I explain further down the line; for what concerns this work, I have used the Tucker decomposition, which is why for the sake of completeness, I will speak about the CP decomposition without going too much into detail and then move on to the Tucker decomposition.

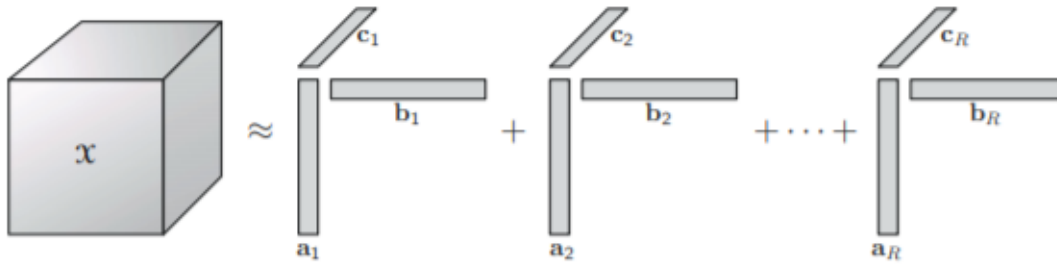


Fig. 4 CP Decomposition of a 3-way tensor (Kolda and Balder [2009])

The CP decomposition factorizes a tensor into a sum of order-1 (or rank-1) tensors. For example:

$$X \approx [A, B, C] \equiv \sum_{r=1}^R a^r \circ b^r \circ c^r \quad (2.63)$$

which is true for a three-way tensor (but it can be generalized to an  $n$ th-order tensor, it is just a matter to take into account all dimensions).

An exact (and ideal) CP decomposition would be the rank decomposition. Indeed, in analogy with the rank of a matrix, the rank of a tensor is the smallest number of rank-1 tensors that generate  $X$  as their sum. The exact decomposition is when  $R$  (the upper limit of the sum) =  $\text{rank}(X)$ . The main difference with respect to the rank of a matrix is that a tensor can have a

different rank over  $\mathbb{R}$  and  $\mathbb{C}$ . However, the main issue with finding the rank of a tensor is that the problem is NP-hard, therefore there is no specific algorithm to calculate it. In fact several ranks have been empirically determined and the tables of fig.5 and fig. 6 below report some of them:

Tensor size	Maximum rank
$I \times J \times 2$	$\min\{I, J\} + \min\{I, J, \lfloor \max\{I, J\}/2 \rfloor\}$
$3 \times 3 \times 3$	5

Fig. 5 maximum ranks for some specific tensors as in Krustal (1989) and J'JaJ'a (1984)

Tensor size	Typical rank
$2 \times 2 \times 2$	$\{2, 3\}$
$3 \times 3 \times 2$	$\{3, 4\}$
$5 \times 3 \times 3$	$\{5, 6\}$
$I \times J \times 2$ with $I \geq 2J$ (very tall)	$2J$
$I \times J \times 2$ with $J < I < 2J$ (tall)	$I$
$I \times I \times 2$ (compact)	$\{I, I + 1\}$
$I \times J \times K$ with $I \geq JK$ (very tall)	$JK$
$I \times J \times K$ with $JK - J < I < JK$ (tall)	$I$
$I \times J \times K$ with $I = JK - J$ (compact)	$\{I, I + 1\}$

Fig. 6 Typical rank by tensor size as in Ten Berge (2000), Kruskal (1983), Ten Berge (1991), Ten Berge and Kiers (1999)

What is done in practice is basically trying to fit several rank-R decompositions and picking the best. The nice thing about the CP decomposition is that it benefits of the uniqueness property. We will see that the Tucker decomposition is not unique but I will explain later why I preferred it. The uniqueness of the decomposition is a nice property to have with respect to the matrix case, since the matrix decomposition is often not unique. In fact, if we consider a matrix Y, its Singular Value Decomposition (of which the CP is the equivalent for tensors) or SVD is:

$$Y = AB^T = \sum_{r=1}^R a_r \circ b_r \quad (2.64)$$

where  $A = U\Sigma$  and  $B=V$ . I can write  $Y=U\Sigma V^T$ . It is equally valid to choose some  $B=VP$  and  $A=U\Sigma P$  with P being an orthogonal, squared matrix. So the only reason why we have a unique

SVD for a matrix is because we impose the orthogonality constraint but it is not necessary in principle. Instead the unique CP decomposition is available under much weaker conditions. The exception is when considering the elementary indeterminacies of scaling and permutation of the rank-1 tensors. The first one refers to the fact that for any  $R \times R$  permutation matrix the CP stays the same, the second deals with the fact that, as long as the scaling factors of the rank-1 tensors sum up to one, the CP is unique for that specific tensor. On the other hand, one may notice that all of this conditions are sufficient but not necessary to the existence of the CP decomposition; several authors have faced the problem and provided necessary conditions, all of which can be found in the paper by Kolda and Bader [2009]. One of the weaknesses of the CP is that there may not be a best rank- $k$  approximation of a tensor. For matrices, one can always find the best rank- $k$  approximation of the matrix  $A$ , which will be the matrix that minimizes the norm of the following difference:

$$\|A - B\| \quad (2.65)$$

it can be shown that the result is the matrix  $B$  which is composed by the first  $k$  factors of the SVD of  $A$ , where  $k$  is the rank of  $A$ . This is not true for tensors and that is why there could be degenerate tensors. Now I will indicate the problem of computing the CP decomposition and then I will indicate an algorithm to compute it. Again, I have not used it but for the sake of completeness, I will decided to talk about it.

Let  $X \in R^{I \times J \times K}$  be a tensor of order three. The goal is to find the  $R$  rank-1 tensors that best approximate  $X$ , in other words:

$$\min_{\hat{X}} \|X - \hat{X}\| \text{ with } \hat{X} = \sum_{r=1}^R \lambda_r a_r \circ b_r \circ c_r = [[\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]] \quad (2.66)$$

where  $\lambda$  is a normalizing vector, since it is very common to let all the vectors have length equal to 1. As mentioned before, there is no unique way to compute the CP, but the leading and most used technique is to use the ALS method or Alternating Least Squares. It is called Alternating because it is very hard to find all three components simultaneously, so what is done is fixing two of the components and then finding the other in each iteration. For example, fixing  $A$ , the problem becomes:

$$\min_{\hat{A}} \|X_{(1)} - \hat{A}(C \odot B)^T\|_F \quad (2.67)$$

which is just the linear least squares problem. In the minimization problem,  $\hat{A} = A \cdot \text{diag}(A)$  and  $X_{(1)}$  just refers to the first frontal slice of the original tensor. The optimal solution is given by:

$$\hat{A} = X_{(1)}[(C \odot B)^T]^\dagger \quad (2.68)$$

which can be rewritten as:

$$\hat{A} = X_{(1)}(C \odot B)(C^T C * B^T B)^\dagger \quad (2.69)$$

due to the property of the form of the Khatri-Rao pseudo-inverse matrix. After this, the same is done for all the other matrices, until some conditions are met, such as maximum number of iterations, little to no improvement in the objective function, and others. The advantage is that during each iteration the pseudo-inverse that has to be computed is of a squared matrix instead of a  $I \times K$  matrix. Be aware, however, of numerical ill-conditioning, which can lead to unexpected results. Finally,  $A$  can be recovered by multiplying on the left the inverse of the diagonalization of the vector  $\lambda$ . In other words, let  $\lambda = \|\widehat{a}_r\|$  and  $a_r = \frac{\widehat{a}_r}{\lambda_r}$  for  $r=1, \dots, R$ . The literature has tried to find alternative ways to the computation of the CP decomposition, trying to avoid the usage of ALS, but it still remains the best way. The following is the algorithm to compute the CP decomposition of an  $N$ -way tensor:

```

procedure CP-ALS( $\mathcal{X}, R$ )
  initialize  $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$  for  $n = 1, \dots, N$ 
  repeat
    for  $n = 1, \dots, N$  do
       $\mathbf{V} \leftarrow \mathbf{A}^{(1)T} \mathbf{A}^{(1)} * \dots * \mathbf{A}^{(n-1)T} \mathbf{A}^{(n-1)} * \mathbf{A}^{(n+1)T} \mathbf{A}^{(n+1)} * \dots * \mathbf{A}^{(N)T} \mathbf{A}^{(N)}$ 
       $\mathbf{A}^{(n)} \leftarrow \mathbf{X}^{(n)} (\mathbf{A}^{(N)} \odot \dots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \dots \odot \mathbf{A}^{(1)}) \mathbf{V}^\dagger$ 
      normalize columns of  $\mathbf{A}^{(n)}$  (storing norms as  $\lambda$ )
    end for
  until fit ceases to improve or maximum iterations exhausted
  return  $\lambda, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)}$ 
end procedure

```

Fig. 7 CANDECOM/PARAFAC through Alternating Least Squares algorithm from Kolda and Balder [2009]

Now I will move on to talk about the Tucker Decomposition and it will be immediately clear why I chose this approach instead of the CANDECOMP/PARAFAC. The Tucker decomposition is known under many names but it is a form of higher order Principal Component Analysis (or HO-PCA). It decomposes a tensor into a core tensor and as many matrices as there are modes, since the core tensor is multiplied (or transformed) by a matrix for each mode. The typical Tucker decomposition look like the following:

$$\begin{aligned}
X &\approx G \times_1 A \times_2 B \times_3 C \\
&= \sum_{p=1}^P \sum_{q=1}^Q \sum_{r=1}^R g_{pqr} a_p \circ b_q \circ c_r = [[G; A, B, C]] \quad (2.70)
\end{aligned}$$

G is the core tensor while A, B and C are the matrices corresponding to the first, second and third mode, respectively. A, B and C are also called factor matrices and can be thought as the principal components of each mode while the core tensor and its entries show the level of interaction between different components.

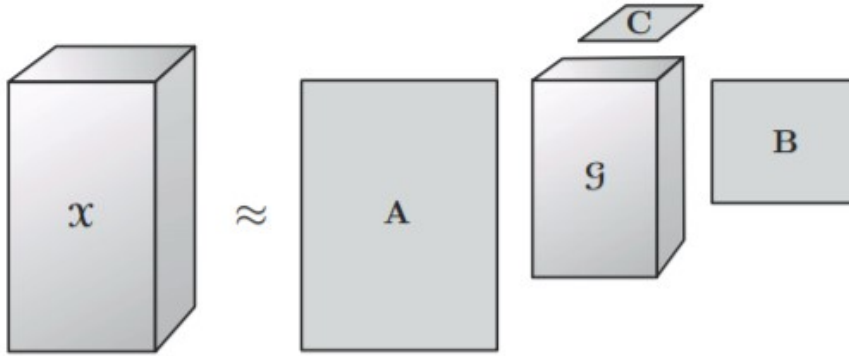


Fig. 8 Tucker decomposition (Kolda and Balder [2009])

It is clear now that the Tucker decomposition was an obvious choice since the scope of this work was to find how several sources of information interact with each other when pricing equity stocks. The core tensor finds exactly that information. Most fitting algorithms to find the Tucker decomposition assume that the columns in the factor matrices are orthogonal but this is not required which allows me to notice that CP can be seen as a special case of Tucker. Indeed, one could think of CP as the case where the core tensor is super diagonal and the number of columns in all the factor matrices is the same. The example that I have provided uses an order-three tensor but this is not required and the problem can be generalized to include N-way tensors. It is worth noting that there are two versions of the Tucker decomposition: Tucker1 and Tucker2. Tucker1 assumes that one of the factor matrices is the identity matrix while Tucker2 assumes that two factors are both the identity matrix which corresponds to the standard two-dimensional PCA, because I can write in matrix form the tensor as:

$$X_{(1)} = AG_{(1)} \quad (2.71)$$

where X is the first frontal slice of the tensor. Now I will proceed to illustrate how one can compute the Tucker decomposition, since there are several ways to do it. The methods that Tucker himself introduced in his paper back in Tucker [1966] were three, all of which with the basic idea that you could find the leading n components that best explained the variability in

each mode, considering such finding independent with respect to all the other components in the other modes. Today such method is known as HOSVD, or Higher Order Singular Value Decomposition. The following algorithm illustrates how to compute it:

```

procedure HOSVD( $\mathcal{X}, R_1, R_2, \dots, R_N$ )
  for  $n = 1, \dots, N$  do
     $\mathbf{A}^{(n)} \leftarrow R_n$  leading left singular vectors of  $\mathbf{X}_{(n)}$ 
  end for
   $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \dots \times_N \mathbf{A}^{(N)T}$ 
  return  $\mathcal{G}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)}$ 
end procedure

```

Fig. 9 Higher Order Singular Value Decomposition to compute the Tucker Decomposition

(Kolda and Balder [2009])

It is very easy to see that the algorithm computes the  $R$  leading left singular values of each frontal slice of the tensor. Then it returns the core tensor and all the matrices with the singular values of all the modes. After this method, De Lathauwer, De Moor and Vanderwalle developed a much more efficient procedure with respect to the previous one that I have stated. Such procedure was called HOOI, or Higher Order Orthogonal Iteration. In this sense, I could define the problem as:

$$\begin{aligned}
 \min_{G, \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}} & \left\| X - \left[ \mathbf{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)} \right] \right\| \\
 \text{subject to } & G \in R^{R_1 \times R_2 \times \dots \times R_N}, \mathbf{A}^{(n)} \\
 & \in R^{I_n \times R_n} \text{ and columnwise orthogonal for } n \\
 & = 1, \dots, N
 \end{aligned} \tag{2.72}$$

By rewriting the above objective function in vectorized form as:

$$\left\| \text{vec}(X) - (\mathbf{A}^{(N)} \otimes \mathbf{A}^{(N-1)} \otimes \dots \otimes \mathbf{A}^{(1)}) \text{vec}(G) \right\| \tag{2.73}$$

It is easy to see that the core tensor  $G$  must have the following form:

$$G = X \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \times_3 \dots \times_N \mathbf{A}^{(N)T} \tag{2.74}$$

This is because the minimization of a norm implies that the lowest point that the function can reach is zero. At this point, we can rewrite the square of the objective function as:

$$\begin{aligned}
& \left\| X - \left[ G; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)} \right] \right\|^2 = \\
& = \|X\|^2 - 2 \langle X, \left[ G; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)} \right] \rangle \\
& \quad + \left\| \left[ G; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)} \right] \right\|^2 = \\
& = \|X\|^2 - 2 \langle X \times_1 \mathbf{A}^{(1)T} \dots \times_N \mathbf{A}^{(N)T}, G \rangle + \|G\|^2 = \\
& = \|X\|^2 - 2 \langle G, G \rangle + \|G\|^2 = \tag{2.75} \\
& = \|X\|^2 - \|G\|^2 = \\
& = \|X\|^2 - \|X \times_1 \mathbf{A}^{(1)T} \times_2 \dots \times_N \mathbf{A}^{(N)T}\|^2
\end{aligned}$$

This means that we can now use ALS to solve iteratively and compute the values for the A matrices. The squared norm of the tensor is constant so the problem of interest is the following:

$$\begin{aligned}
& \max_{\mathbf{A}^{(n)}} \left\| X \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \times_3 \dots \times_N \mathbf{A}^{(N)T} \right\| \\
& \text{subject to } \mathbf{A}^{(n)} \\
& \in R^{I_n \times R_n} \text{ and columnwise orthogonal} \tag{2.76}
\end{aligned}$$

which can be rewritten in matrix form as:

$$\left\| \mathbf{A}^{(n)T} W \right\| \text{ with } W = \mathbf{X}_{(n)} (\mathbf{A}^{(N)} \dots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \dots \otimes \mathbf{A}^{(1)}) \tag{2.77}$$

The solution is just the SVD of W. It will surely converge to a solution, however there is no guarantee that it will be the global optimum nor that it will be a stationary point. Eldèn and Savas [2009] have developed a method that has fewer iterations than HOOI using a Newton-Grassmann optimization strategy (which means applying derivative-based methods) for computing the Tucker decomposition of a three-way tensor. It is, however, more expensive in each iteration due to the computation of the Hessian but it ensures quadratic convergence numerically as the authors demonstrated. This method will converge to a stationary point. As mentioned before, the CP decomposition is unique; Tucker decomposition does not have such property, however I see it as an advantage because it means that one can choose the transformation such that the core tensor can be reduced to values equal to or very close to zero, leaving only the factor matrices. Superdiagonalization is impossible, as Tucker [1966] himself noticed in his original work, but it is possible to try and reduce as much as possible the elements of the core tensor. Finally, I will show the algorithm to compute the HOOI:



```

procedure HOOI( $\mathcal{X}, R_1, R_2, \dots, R_N$ )
  initialize  $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$  for  $n = 1, \dots, N$  using HOSVD
  repeat
    for  $n = 1, \dots, N$  do
       $\mathbf{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\top} \dots \times_{n-1} \mathbf{A}^{(n-1)\top} \times_{n+1} \mathbf{A}^{(n+1)\top} \dots \times_N \mathbf{A}^{(N)\top}$ 
       $\mathbf{A}^{(n)} \leftarrow R_n$  leading left singular vectors of  $\mathbf{Y}_{(n)}$ 
    end for
  until fit ceases to improve or maximum iterations exhausted
   $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\top} \times_2 \mathbf{A}^{(2)\top} \dots \times_N \mathbf{A}^{(N)\top}$ 
  return  $\mathcal{G}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)}$ 
end procedure

```

Fig. 10 Tucker Decomposition through Higher Order Orthogonal Iterations (Kolda and Balder [2009])

The obvious remark is that I used this approach, instead of HOSVD alone, due to the need of computing the decomposition for a stream of three-dimensional tensors and that is also why I choose TensorLy as the Python package to perform the Tucker decomposition. As it is described in the paper that introduced this Python library, together with the online documentation of the library, they implement the HOOI to compute the Tucker decomposition. There are several other decompositions which are all mentioned in Kolda and Balder [2009] which are all somewhat related to the CP and Tucker, however I will not mention them here.

One type of variant of both Tucker and CP decompositions are the NNCP and the NNTucker where NN stands for Non-Negative. Sometimes, it can be useful to impose the non-negativity constraint on the A matrices to improve readability of the results, as it is the case with greyscaled images. There are also some newer models that try to combine aspects of CP and Tucker decompositions, such as the work of De Almeida et. ot. [2006]. They try to express a tensor as a sum of lower rank Tucker tensors. For example:

$$X \approx \sum_{r=1}^R [[G_r; \mathbf{A}_r; \mathbf{B}_r; \mathbf{C}_r]] \quad (2.78)$$

A picture can be more explicative:

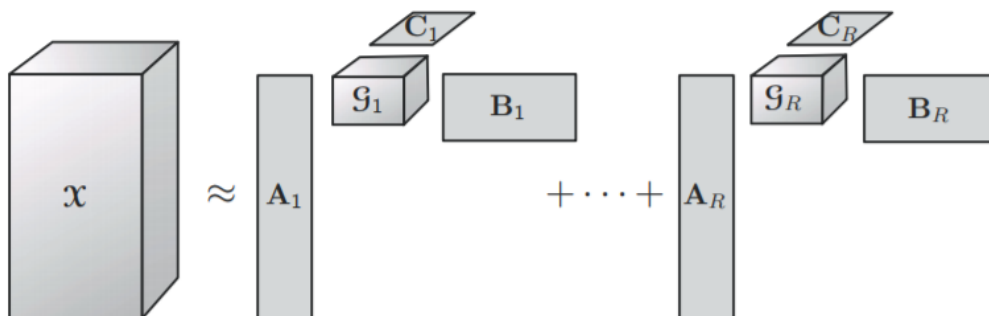


Fig. 11 Block decomposition of a third order tensor

Here I will conclude the current section with some acknowledgements to the various workshop that are available online on YouTube, uploaded by the Institute of Applied Mathematics, which greatly helped in understanding better these concepts. Some of these conferences and workshops were even held by Tamara Kolda herself. Also, I would like to mention the great help coming from the workshops of the Institute for Machine Learning Research, hosted by Anima Anandkumar, who is one of the developers of the TensorLy library.

### 3: Data Description and Preprocessing

In this section, I want to describe my data and to give some context of the various sources of such data, how it was gathered and finally preprocessed. The first thing to say is that my data comes from several sources which I aggregated through my code. Furthermore, I decided to use information coming from three separated types of data, borrowing the idea from the most relevant paper for this thesis which is the work of Qing Li et. ot. [2016]. They try to explain how an investor behaves, having at her disposal information coming from three different sources: quantitative data, such as the Fama and French risk factors, but also two types of textual data, which were a way to score the impact of rational information coming from news on investment decision and some way to capture the sentiment and the irrational behavior, especially through online posts. They performed their study on the Chinese market, however I tried to apply that methodology to the American market, specifically the first 100 companies by market capitalization of the Standard and Poor's 500 equity index. Also, it was a little bit harder to find the data that I needed, while the authors were able to access huge datasets where some part of the preprocessing was already done. Following their example, I built a stream of tensors that had three modes, each mode representing a source of information. In particular, there was the firm-specific mode, which consisted of quantitative data, taken from the Eikon servers. My first attempt was to use monthly aggregated data but that option was quickly disbanded since the effect of news and online forums or social media posts have a very limited effect in time. Thanks to Professor Caporin, I was able to get a much higher frequency dataset, which consisted of the minute-by-minute values of the highest and lowest prices, together with the traded volume for the considered companies. The second mode of the tensors was the so called event-specific mode. It is well established and also particularly intuitive that news have an effect on the market. When the news comes out, professional traders already know what they contain and trade on the basis of it; they discount their preferences though this new information and try to adjust their behavior. This is basically the same thing that happens with posts, of which I will speak about in a second, but the key difference is that news represent the "hard truths". They are not irrational sources of information but rather some proxy of the fundamental value and its changes, so it was a necessary addition to the information space. The third and last mode captures a different set of information coming from tweets. It is already established that many agents on the financial markets are not rational and they trade to entertain themselves or without a lot of knowledge or comprehension of what they are doing. There are also other people who are simply uninterested in the direction of the market and they do not trade for profit but because they have a specific need. Finally, there are some agents that trade on older

news, or more in general they are driven by emotions and are easily influenced by the mood swings of their peers which leads to contagion phenomena among such traders. To sum it up, many traders are not rational and are usually those who lose in the financial markets because experienced traders anticipate their moves or, in many cases, give into the same behavior themselves, generating intra-day trends and profit opportunities to be exploited. To capture such information, I decided to gather several tweets regarding the stocks that I have considered in my work and use them as inputs for my predictive model. The initial remark that I want to make is that I am not predicting the price in a dynamic way, but rather I am assuming market efficiency in each minute so that my model predicts a certain price and then I trade on the basis of the comparison with the actual price. As I mentioned earlier, the quantitative data was the easiest to find, since I had access to the Eikon database but also to Professor Caporin's resources. In order to construct the firm-specific mode I initially gathered all the prices in an Excel spreadsheet where I proceeded to compute the average between the high and the low prices in each minute. After that, I used these prices to compute the Price to Earnings ratio (P\E) and the Price to Book ratio (P\B) and finally I multiplied the average price with the number of shares traded in each minute to obtain the Turnover by Volume. The reason why I chose these explicative variables relies on the fact that they are a proxy of the fundamentals of the firm, which are the primary source of information when deciding in which company to invest. One might say that they are the only information that you need when the time horizon of the investment is long, since all the information contained in the news and the tweets has already been discounted into the prices, which is why the monthly aggregated data was not suitable to my needs. By the way, the average price is my objective variable, I included it in the dataset to upload it in memory, but then I isolate it and store it in a dedicated object, there is no temporal dependency from previous times. I am aware of the fact that this is a really strong assumption to make, which is why there is a lot of room for improvement of this work. To manipulate the dataset I used the pandas python package, which is the go-to library for dataset manipulation. In the appendix of this work, there is the entire source code, indexed by the name of each script. I also avoided deleting the code that manipulated the monthly frequency dataset because I find informative just to look at the overall behavior in the past five years of the stocks that I have considered and also because I wanted to take advantage of the work made by Tummarillo et. ot. [2003] that created the so called Planar Maximally Filtered Graph, or PMFG. As I mentioned in the introduction, the increasing complexity of this world is reflect also in the more complex and yet elegant ways to simplify such complexity and visualize it. The economic environment is subject to this rule as well, one could even say that it has always been very complicated, however today we possess tools that can help us in overcoming challenges that

were previously unthinkable such as that of having an idea of all the possible linkages that might exist among firms and thus have an idea of some underlying structure to the system. By using the correlations among the prices of the firms that I have considered in my work, I was able to build the PMFG and thus visualize the underlying star-like structure that the economic system has. This is not something new and has been proven several times but I find fig.11 very informative:

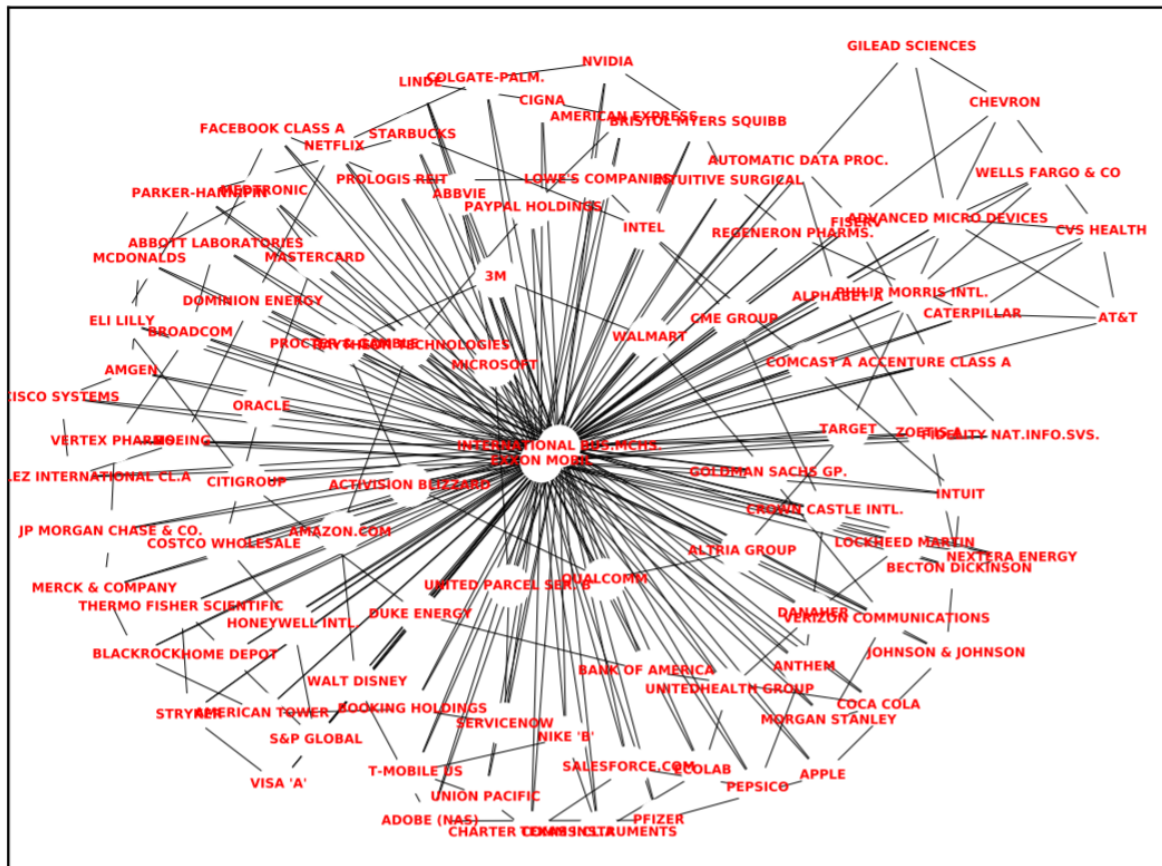


Fig. 11 The PMFG for the first 100 companies of the Standard and Poor's 500, the data used is from the last 5 years monthly aggregated prices

As it was confirmed in other works, financial firms are not positioned in the center of the graph, but actually occupy marginal positions, as Wells Fargo and CO. shows. The central firms are basically two, Exxon mobile and International Bus.MCHS. It is clear that the centrality of this firms depends on how fundamental they are, since oil is the primary source of energy and it powers every machine of the other firms, while International Bus.MCHS is responsible for basically all the logistics and deliveries of goods on road in the US. The key point here is that this graph is preserving planarity (I cannot take out a subgraph without creating a “hole” in the original graph, not even a single node) while also ensuring that each node is linked to its neighbors. Depending on how long the lines are, the degree of how strong the connection is

between two firms changes. However, the firms that are linked to each other for some reason, even though they are not neighbors are captured by this graph. The algorithms to compute the PMFG are readily available on the web. On the other hand, the time frame that I have considered to build up my model goes from 17-08-2020 to 21-08-2020 and I have considered the trading hours from 9:30 to 16:00. This means that there are 420 observations in each day for a total of 1950 observations. Besides the quantitative data, I also gathered qualitative data, specifically news headlines regarding the stocks that I have considered and tweets taken directly from Twitter as the source of the irrational information that generates the short term volatility in the market. Regarding the news headlines, I followed two approaches but eventually settled for the news headlines of each single company. My initial thought was to gather macroeconomic specific news from the Reuters website and so I did. However, I was not satisfied with the results, mainly because I was assuming that such news would affect all the firms, which is true, but I had no way to measure how much these news would impact each firm. I kept the code aside and decided to follow another route, which was to gather data for each firm specifically. That was possible thanks to the website [finviz.com](https://finviz.com). It is easily accessible and I was able to design a scraper relatively easily. From there I gathered all the news that I needed for each company in the time frame that I was considering; indeed, [finviz.com](https://finviz.com) considers relevant only the last week news and discards older news which is assumed to be useless. The python libraries that can help when designing scrapers are various, the most famous is Beautiful Soup, which I have not used since it requires a more advanced knowledge of HTML and can help design scrapers on a lower level. What I have used is Selenium, which is available also for other programming languages, such as R. It allowed me to program my scraper in a higher level, just by providing to the parser the correct tag coming from the HTML source code of the webpage. It is relatively easy to learn how to create a very informal and goal-specific web scraper just by spending a little bit of time researching the web, which is another reason why I chose Python over other programming languages. Not only is it easy to use, but the resources to learn it and to apply it are readily available on the web and anyone can start to learn. Even if the Selenium library is very high level, it still retains a lot of versatility, as long as the class of the HTML webpage is properly specified, otherwise there are some workarounds to use but it can definitely get the job done. In the second script, called “better web scraping for news headlines”, I actually used a mixture of Beautiful Soup and Selenium to reach my goals, because Selenium was too constraining when trying to scrape the news from [finviz.com](https://finviz.com). Another unfortunate obstacle was that [finviz.com](https://finviz.com) allows to search a company through its ticker; if you try to search by name, the website redirects to a list of pages that might be of interest, including articles among other things. To get to the actual webpage of the stock, where there is a conveniently placed list of

news, already divided by date, the research must be by ticker. Indeed, that is why I started to gather them in the very beginning of the script, because I had been using the names of the companies in the rest of the work. One thing that must be remarked is that the ticker was added by string addition to the url, which is in the very beginning of the code. Sometimes, you might encounter some pauses that I have added in the code, which slow it down by a huge factor, but it was necessary, otherwise the code would crash, since the cursor would not find the element specified due to the fact that the webpage was still loading. Also, later on, I applied some correction to the list of tickers, in particular I appended the last company that I had in my dataset, which is Ecolab, I removed the 3rd element, because I had no data for that stock. It was the type A stock of Google, which has "GOOGL" as ticker. This is the preferred stock of Google, which I have discarded and finally I added the letter "C" to the 91th element (because array indexing starts from 0), since the website where I gathered the tickers had some errors in it. Then the real deal starts. In order to use Beautiful Soup, one must create a Request object from the Request class and specify the type of user and the type of permission. The one that I have used is the most generic, which is accepted by every public website. Of course there are more advanced parameters that can be passed but I have yet to dive deeper into them. Once the code puts the request in place, then the server answers back with a response which is then opened and parsed as a HTML written content. From there, the code looks for a specific object which is the news table and stores the content of the table that I have specified for each company ticker in the list that has been fed to the for loop. At this stage, I then store everything into a dictionary, so that I can index the news to the ticker and then parse the relevant information that I want, starting from the news headline itself, then the minute and finally the date of when the article came out. Now that all the news have been correctly parsed and stored, the thing that remains to do is convert them in a format that can be fed to a mathematical model. The way to do that is offered by the techniques developed by the Natural Language Processing field of research. During the years there have been several methods developed to improve the interactions between man and machine, especially considering how a machine could understand natural language, or more properly speaking, how a machine could quickly convert natural language in numbers and then make a prediction on what the next word or phrase or paragraph even could be when interacting with a human. Furthermore, the research has been focused also on how all this textual data could be used to improve predictions or more generally to exploit the information within massive user-generated and online content, broadly speaking. In particular, I have used two methodologies to analyzed text in my work. For the news I have used the VADER lexicon while for the tweets I have used the TFIDF method. I will explain the latter in greater detail further down the section. VADER stands for Valence Aware Dictionary

for sEntiment Reasoning and it was developed by Hutto and Gilbert [2014]. It is capable of measuring the sentiment intensity, not only the sentiment of the text that it analyzes. The idea with VADER is to assign a value that is more continuous than the mere score of the sentiment. Indeed, VADER ranges from a lower limit of -1 which indicates a strongly negative piece of text, to 0 which indicates total neutrality to +1 which indicates a strongly positive text. It can take all possible real values in such interval, giving it the appropriate degree of freedom to illustrate the various positive or negative sentiments that investors can have regarding some news. An example should clarify: some news regarding Microsoft not meeting the analysts' expectations is viewed as negative by investors whose portfolios are exposed to Microsoft, while investors that are more exposed to Apple might find the news positive, since they are competitors. Such diversity is captured in the fact that the score is more of a polarity of the sentiment. The idea behind how VADER works is very simple and it has been constantly updated since its release; it relies on a gold-standard list of lexical features which have been manually gathered by the authors through their own work or by relying on external agents who manually evaluated several sources of text to then teach the model how humans express a sentiment with more or less intensity, instead of just categorizing it as positive, negative or neutral. Another reason why I decided to apply VADER was because it was not yet used on financial news headlines but it generalized very well on several types of textual data. It was first thought for tweets but then it was extended to movie and product reviews and also New York Times articles. As for the news articles, I also followed two approaches: the first one was to divulge my attention to the primary source of tweets, that is why I wanted to scrape Twitter directly but sadly it could not be done with a simple user account. I registered to the Twitter developer program and obtain some credentials which allowed me access to the website. However, there were several limitations imposed to free developer accounts in various ways, such as the number of tweets that could be downloaded and the time frame that could be considered. Also there was no way to measure engagement of tweets, with engagement meaning the number of hearts given to a tweet, the number of retweets or the number of people that simply viewed the tweet. I had no way to control which tweets I was scraping, so I could be getting some tweets that were getting a lot of engagement while some others that were pretty much unknown to most of the users. The engagement was so important because the effect of irrational information is meaningful when it is contagious. If few people behave irrationally, the market simply kicks them out because they lose a lot. However, if a lot of people behave irrationally, then the effect on the market is significant and they are responsible for the short term volatility that allows the generation of small intra-day trends. Furthermore, when there are a lot of irrational agents, the rational traders do not behave rationally but actually give into the



same irrational behavior because it is convenient to do it, to push prices even further and realize even bigger profits as several studies have shown in the Behavioral Finance field. Such herd effect can be measured by how many people have read the tweets; not everyone will be a trader of course but the greater the visibility, the greater the chance that a trader saw that tweet and decided to trade based on that information, among other things. That is why I used another python library which is GetOldTweets3. This library did not have any limitations whatsoever, I could have potentially scraped tweets from a year ago and as many as I wanted, but the most useful method of the TweetCriteria class was setTopTweets, which allowed me to only scrape the most significant tweets in the period considered for each company. I also thought of using VADER on the tweets dataset but I preferred another method which is the Term Frequency Inverse Document Frequency, or TFIDF. The idea was to avoid any pollution of the data by adopting the same methodology of analysis of the two separated sources of information when training the regression model for tensors and price prediction, such as unexpected behavior or synergies which are not present in the underlying structure of the data but rather artificially introduced during the preprocessing phase. The idea behind the TFIDF method is to convert the text into arrays of ones and zeros if a specific feature is present into the dataset. Usually to each feature is assigned a word, with a maximum number of feature specified by the user according to one's needs. I used 2000 features to be sure that every possible true sentiment conveyed in the tweets was correctly classified. However, the TFIDF methodology takes it a step further because it measures not only if a feature is present or not but also how many times such feature is present in the text and it also computes the inverse of the frequency in the document. Thanks to this smoothing factor, information conveyed by rarer features is prioritized while words that are very common receive less weight when creating the feature matrix. The way in which this is achieved is by creating a dictionary, some reference for the method to understand how to indicate the presence of a feature. In many studies, the authors create their own dictionary, especially when it comes to financial information; however, I was not using specifically financial tweets but rather some more generic tweets so I used the stopwords dictionary that comes with the python library NLTK, which stands for Natural Language ToolKit. This library is a must-have when it comes to implementing these methodologies for textual analysis. The stopwords dictionary is also very useful because it gathers all the words in the English language that are of common use, such as determinative and undeterminative articles, which do not add anything valuable to the text rather than making it grammatically correct. In this sense, they are useless for the extraction of the sentiment so they are voluntarily ignored when featurizing the text itself. There are also other elements that do not contribute to the classification of the tweets such as prefixes and suffixes, which can be removed through the Porter Stemmer algorithm;

however, I have not used it. Sometimes, the latter can lead to misleading classification, especially when it mistakes some “slang” or “lingo” in the tweets for elements that do not contribute to the sentiment while in fact they do. Another important thing to remember is that I did not have already classified tweets, so I needed to classify my dataset first and then perform the sentiment analysis. To do that I used another comprehensive dataset of tweets coming from GitHub, which was readily available and free to download. The dataset consisted of 15000 reviews for airline companies in the US. After I parsed all the tweets with regular expressions, I used the entire dataset as a training ground for the Random Forest classifier and then I applied it to my tweets dataset which consisted of 20000 tweets. This is because I instructed the machine to gather the most significant 200 tweets in the time interval that I have specified for all the 100 companies in the dataset. The main reason to use the Random Forest classifier is the so called “wisdom of the crowd”, however the best way to explain how the classifier works, I must explain the meaning of both words. It is random in the sense that it draws randomly tweets from the dataset that I want to classify and it is a forest in the sense that there are several decision trees that classify the tweets according to the dictionary previously created with the TFIDF methodology. However, the final sentiment extracted from the tweet (be it positive, negative or neutral) was the most frequent sentiment in the forest. By that I mean that each decision tree classifies the tweets with a specific sentiment, but the final result will be the one chosen by most of the trees, which is indeed the wisdom of the crowd effect. The actual number of trees that I used is equal to 100. Next thing that must be done was to construct the sentiment mode, since the mere presence of the tweets sentiments was not enough. I wanted to measure the intensity of each sentiment with respect to the less frequent so that I could create even more sources of variability and thus of valuable information. After the sentiment extraction, the most prevalent sentiment was negative, followed by neutral and then positive. This was expected in some way since the markets were experiencing a lot of volatility and the pandemic was still in its peak, people were still adjusting to the smart working necessity while being stuck at home, deprived of holidays. To describe in further detail how I created the sentiment mode, I will refer to a similar methodology used in Qing Li et. ot. [2016], which followed from Qing Li et. ot. [2014]. The idea was to measure the either optimistic, neutral or pessimistic mood regarding a stock in the market together with the specific sentiment towards the single stock. In order to measure the sentiment of a specific stock, I applied the following formula:

$$M^{pos} = \sum_{i=0}^I num\_pos_i + T_i \quad (3.1)$$

where  $T$  with subscript  $i$  stands for a time correction factor given by:

$$T = e^{-\frac{i}{\beta}} \quad (3.2)$$

where I set beta to be equal to 20. The idea is to give more weight to more recent tweets, while older tweets become less and less informative as time passes. In the same way, I computed the neutral market mood and the negative market mood for each specific day that I have considered. Here the assumption that I am making is that the tweets are valid for the entire day, in the sense that I am not differentiating whether a tweets was posted in the morning or in the afternoon. Such a constraint could be relaxed and one could check if there are some better estimates. The reason why I set beta equal to 20 is because it seemed a good way to approximate the delay between posting, reception of the tweets and then the decision to trade according to them, as Qing Li et al. [2016] showed. After having computed the moods of each stock, I also wanted some measure of how intense with respect to the other a sentiment was. That is why I defined the neutral intensity and the negative intensity as:

$$I^{neut} = \frac{M^{neut} - M^{pos}}{M^{neut} - M^{pos}} \quad (3.3)$$

$$I^{neg} = \frac{M^{neg} - M^{pos}}{M^{neg} + M^{pos}} \quad (3.4)$$

where “neg” stands for negative and “neut” stands for neutral. To end this section, I will just mention how I have arranged the data in order to load them into memory and then manipulate the objects with the code. After having performed all of these modifications and transformations, which I have discussed before, I loaded everything in an excel file, organizing the data so that all the variables were somewhat indexed by time and by company. This was done manually, moving cells around an Excel worksheet; I provide a snippet of such file as an example in figure 12, to understand what I mean:

	A	B	C	D	E	F	G	H	I	J	K	L
1	dates	minute	MICROSOFT avg	MICROSOFT P/B	MICROSOFT P/E	MICROSOFT Turnover	MICROSOFT_pos	MICROSOFT_neut	MICROSOFT_neg	MICROSOFT_intensity_neut	MICROSOFT_intensity_neg	compound
2	17/08/2020	09:30:00	209.585	13.46	36.5	77439561.65	10	7	96	0.001335974	-0.005061197	0
3	17/08/2020	09:31:00	209.72	13.46866999	36.52351075	18831597.68	10	7	96	0.001335974	-0.005061197	0
4	17/08/2020	09:32:00	209.9475	13.48328053	36.56313071	13110591.53	10	7	96	0.001335974	-0.005061197	0
5	17/08/2020	09:33:00	209.93	13.48215664	36.56008302	14841001.35	10	7	96	0.001335974	-0.005061197	0
6	17/08/2020	09:34:00	209.7175	13.46850944	36.52307536	10174654.23	10	7	96	0.001335974	-0.005061197	0
7	17/08/2020	09:35:00	209.7725	13.47204165	36.53265382	14075734.75	10	7	96	0.001335974	-0.005061197	0
8	17/08/2020	09:36:00	209.785	13.47284443	36.53483074	22817475.31	10	7	96	0.001335974	-0.005061197	0
9	17/08/2020	09:37:00	209.5275	13.45630723	36.48998616	14470178.68	10	7	96	0.001335974	-0.005061197	0
0	17/08/2020	09:38:00	209.45	13.45133001	36.47648925	19370773.8	10	7	96	0.001335974	-0.005061197	0
1	17/08/2020	09:39:00	209.26	13.4391278	36.44340005	26193701.98	10	7	96	0.001335974	-0.005061197	0

Fig. 12 Snippet of the used dataset and its organization

The idea was to have all the information a time  $t$  regarding company  $i$  very close, so that when reshaping the array in which I stored the loaded data I could still be able to trace back each cell by knowing its position in the array. The resulting matrix was  $1950 \times 1000$ . There are 1000 columns because there are 10 variables regarding each of the 100 companies. After loading the data into memory, I needed to reshape the matrix to create the tensor stream. Before explaining how I did it, let me remark that the operation of reshaping the original matrix into a stream of tensors has nothing to do with Tensor algebra or any other kind of operation that could alter the original data. I am simply taking the data in each cell and repositioning it somewhere else, changing the organization of the data, rather than the structure. The best python library to manipulate arrays, matrices and to solve optimization and linear problems is numpy. There are several other libraries but numpy is the most famous and the choice for beginners who are approaching the python programming language when it comes to linear problems. Other choices include, but are not limited to, pytorch or TensorFlow. I chose numpy also because I used TensorLy, as I mentioned earlier, to perform the decompositions and prepare the ground for the tensor regression; TensorLy allows the user to choose the backend library to rely upon when performing calculations. The most known are all supported but numpy is the default one and I saw no reason to change it, mainly because the other libraries are usually used by those who have to tackle problems of much greater entity; my case was a little smaller in size so I was certain that I would have not had any problems in that aspect. Another small remark is that I want in no way penalize the numpy library, which is very efficient and crucial in several applications. The resulting reshaped array is four dimensional, in particular it was of dimensions  $1950 \times 100 \times 3 \times 3$ . The reason is because I left unaltered the number of rows, since they indexed for time, then 100 is for each company and then the  $3 \times 3$  subgroups have all the explanatory sources of variability (since  $3 \times 3 = 9 = 3$  variables making up the firm-specific mode plus 5 making up the sentiment-specific mode plus 1 making up the event-specific mode). The missing column is the target variable, the average price, which I have stored in a separate object, in particular a

1950x100 matrix, where again the rows are indexed according to time and the columns are just the prices for all the 100 companies that I have considered.

## Section 4: Global Dimensionality Reduction and Tensor Regression

Now that I have described what I did to prepare my data, I can continue by explaining the methodology that I have used to create my regression model. The guidelines that I have followed were from the paper by Qing Li et al. [2016]. A tensor-based information framework for predicting the stock market. Since they applied it to the Chinese stock market, I tried it on the US first 100 companies by market capitalization in the S&P 500 during the period from 17-08-2020 to 21-08-2020. The model is separated in basically two algorithms that further prepare the data and then its output can be considered the theoretical price on which I will design a trading strategy.

The first algorithm was defined of global dimensionality reduction, or GDR in short, which was necessary due to the high dimensionality of the problem. To remark the idea, the data has been now organized in 1950 tensors, each of dimension 100x3x3. I will now illustrate a picture which summarizes the scope of the model and the various logical passages, to help visualize what I am trying to accomplish.

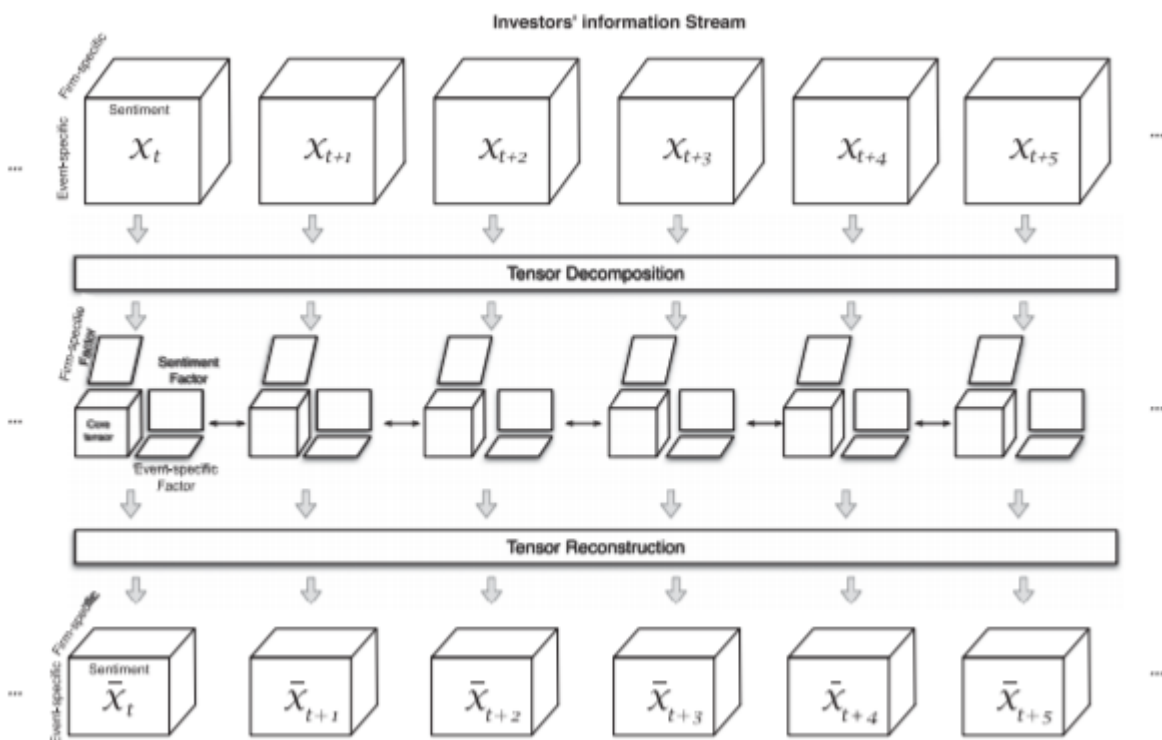


Fig. 13 Phases of the model that I want to build (Quan Li et. al. [2016])

Picture 13 is pretty self-explanatory: starting from the information space, the investor gathers information from the previously mentioned three sources: firm specific, event specific and sentiment specific. The data organized in a tensor at each time  $t$  (which in my case is each minute) is then decomposed through a tensor decomposition technique, which is the Tucker

decomposition for the reasons that I have previously described. Then the tensor is reconstructed to build up the regressors, or the inputs for the tensor regression model. As I also mentioned previously, the assumption here is that the price at time t-1 has no impact on the price at time t. I will indicate now the algorithm and then provide the explanation of the various parts, together with the Mathematics behind it.

---



---

<b>Input:</b>	The training tensor stream $\mathcal{X}_i _{i=1}^N \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and the associated indicators $y_i _{i=1}^N \in \mathbb{R}$ .
<b>Output:</b>	The mapped tensor steam $\tilde{\mathcal{X}}_i _{i=1}^N \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ , where $J_k \leq I_k$ .

---

<b>Step 1:</b>	Calculate the weight matrix $\mathbf{W}$ ;
<b>Step 2:</b>	From $k = 1$ to 3
<b>Step 3:</b>	From $i = 1$ to $N$
<b>Step 4:</b>	Decompose the original tensor $\mathcal{X}_i$ into $C_i \times_1 \mathbf{U}_1^i \times_2 \mathbf{U}_2^i \times_3 \mathbf{U}_3^i$ by Tucker decomposition;
<b>Step 5:</b>	End
<b>Step 6:</b>	$\mathbf{D}_{U_k} = \sum_{i=1}^N d_{i,i} \mathbf{U}_k^i \mathbf{U}_k^{iT}$ ;
<b>Step 7:</b>	$\mathbf{W}_{U_k} = \sum_{i=1}^N \sum_{j=i}^N w_{i,j} \mathbf{U}_k^i \mathbf{U}_k^{iT}$ ;
<b>Step 8:</b>	Obtain $\mathbf{V}_k$ by solving $(\mathbf{D}_{U_k} - \mathbf{W}_{U_k})\mathbf{V}_k = \lambda \mathbf{D}_{U_k} \mathbf{V}_k$ ;
<b>Step 9:</b>	End;
<b>Step 10:</b>	From $i = 1$ to $N$
<b>Step 11:</b>	$\tilde{\mathcal{X}}_i = C_i \times_1 (\mathbf{V}_1^T \mathbf{U}_1^i) \times_2 (\mathbf{V}_2^T \mathbf{U}_2^i) \times_3 (\mathbf{V}_3^T \mathbf{U}_3^i)$ ;
<b>Step 12:</b>	End.

---



---

Fig. 14 Global Dimensionality Reduction Algorithm (Quan Li et. ot. [2016])

As it is indicated at the very top pf the table, the inputs are the tensors that I have described earlier. The output that I expect is a stream of tensors where the dimensions will be lowered. I must say that in my work I found that the dimensions remained the same and this is due to my dataset being not very big, thus all the information could be exploited. Indeed, it was not a problem, since the GDR algorithm tries to reduce the dimensionality but the requisite is that  $J_k \leq I_k$ . The output is defined “mapped” because the actual protagonist of this algorithm is the matrix  $\mathbf{V}$ , computed along the way. This is the regularization matrix, or actually I should say the regularization matrices, since each mode has its own. The idea is that the geometric structure of the tensor must be preserved, or at least there needs to be some trace of it after the original information space is mapped to the new one that has a number of dimensions lower than or equal to the original one. The algorithm starts with the computation of the weighting matrix  $\mathbf{W}$  that I will define properly later on, then a first loop starts according to the number of modes and for each mode every tensor is decomposed through the Tucker decomposition. At that point two matrices for each mode are computed. The matrix  $\mathbf{W}$  was already initialized while the matrix  $\mathbf{D}$  is a diagonal matrix where each value on the diagonal corresponds to the sum of the

corresponding column in the matrix W. Through D and W one can compute the matrix V by solving the following equation:

$$(D_{U_k} - W_{U_k})V_k = \lambda D_{U_k} V_k \quad (4.1)$$

Solving for  $V_k$ :

$$(D_{U_k} - W_{U_k})V_k - \lambda D_{U_k} V_k = 0 \quad (4.2)$$

$$(D_{U_k} - \lambda D_{U_k} - W_{U_k})V_k = 0 \quad (4.3)$$

$$[(1 - \lambda)D_{U_k} - W_{U_k}]V_k = 0 \quad (4.4)$$

Where V in the last equality is just the nullspace of the matrix  $(1 - \lambda)D_{U_k} - W_k$  which is known. The nullspace (or kernel for linear objects other than matrices) is defined as the values of V that make the previous equation equal to zero, avoiding the trivial solution of  $V=0$ . Finally, each tensor is recomposed by multiplying each factor matrix by the corresponding matrix V, for each mode. This makes the role of V even clearer since it means that V is the projection matrix for mode k, so that the new stream of tensors, with a lower dimensionality, can be obtained. This new stream of tensors becomes the input for the Tensor regression model. Now I will present the math behind the algorithm:

The goal is to obtain the correction matrix for each factor, so to compute the matrix V (which has I rows and J columns) for each mode k, the objective function to minimize is:

$$\min_{V_k} J(V_k) = \frac{\sum_{i=1}^N \sum_{j=1}^N \|V_k^T U_k^i - V_k^T U_k^j\| w_{i,j}^2}{\sum_{i=1}^N \|V_k^T U_k^i\|^2 d_{i,i}} \quad (4.5)$$

In the function,  $w_{i,j}$  and  $d_{i,i}$  are elements coming from the matrix W and D respectively. W is an upper triangular matrix that measures the “proximity” of two tensors in the sequence. So if the tensor at time t is close to the tensor at time t-1, then the matrix W will have a 1 in that position, and zero otherwise. More specifically:



$$\begin{cases} 1 & \text{if } i \leq j \text{ and } \frac{|y_i - y_j|}{y_i} \leq 5\% \\ 0, & \text{otherwise} \end{cases}$$

The matrix D, on the other hand is a diagonal matrix with the sums of the columns of W as elements of the diagonal. It is immediate to see how the entire function is designed to accomplish two objectives, the first one is to obtain a correction matrix that is able to project in the new space spanned by the columns of V without losing too much information in the process. In some sense the goal is to maximize the variance as the matrix is projecting the information on the lower dimensional subspace. Assuming that  $V_k^T U_k^i$  is a random variable with mean zero and estimating the probabilities using spectral graph theory from the matrix D (as in Chung [1997]), the denominator is exactly the variance. Minimizing the reciprocal of the variance means to maximize the variance itself, so that the projection is as informative as possible. The other goal is to create a correction factor that is also able to capture the dynamic connections among different tensors. The factors matrices can incorporate by themselves the intrinsic connections within the modes, the core tensor can incorporate the relationships between the modes but there would be no way to find the relationships between two contiguous tensors. The matrix V does exactly that but overcorrection is avoided thanks to the fact that only the tensors that are close enough contribute to the formation of the correction matrix. Such closeness is measured by the elements of the matrix W, which I remind is designed as above to do exactly that. At this point let me define  $V_k^T U_k^i = A^i$ , so that:

$$\begin{aligned}
J(V) &= \frac{\sum_{i=1}^N \sum_{j=1}^N \|A^i - A^j\|^2 w_{i,i}}{\sum_{i=1}^N \|A^i\|^2 d_{i,i}} = \\
&= \frac{\sum_{i=1}^N \sum_{j=i}^N \text{trace}(A^i - A^j)(A^i - A^j)^T w_{i,j}}{\sum_i^N \text{trace}(AA^{iT})d_{i,i}} = \\
&= \frac{\sum_{i=1}^N \sum_{j=i}^N \text{trace}(A^i A^{iT} + A^j A^{jT} - A^i A^{jT} - A^j A^{iT}) w_{i,j}}{\dots} = \\
&= \frac{\text{trace}(A^1 A^{1T} \sum_{j=1}^N w_{1,j} + A^2 A^{2T} \sum_{j=2}^N w_{2,j} \dots + A^N A^{NT} \sum_{j=N}^N w_{N,j} - \sum_{i=1}^N \sum_{j=1}^N A^i A^{jT} w_{i,j})}{\text{trace}(\sum_i^N AA^{iT} d_{i,i})} \\
&= \frac{\text{trace}(\sum_{i=1}^N A^i A^{iT} d_{i,i} - \sum_{i=1}^N \sum_{j=i}^N A^i A^{jT} w_{i,j})}{\text{trace}(\sum_i^N AA^{iT} d_{i,i})} \tag{4.6}
\end{aligned}$$

The key passage is the third one, where the problem is simplified thanks to the property of the trace of a square matrix, or more precisely, the following property:

$$\text{trace}(A^T B) = \text{trace}(AB^T) \tag{4.7}$$

where the dimensions of A and B must be such that the result is a square matrix. It is then possible to interchange the sum operators and the trace operator since they are both linear, to recognize the structure of the matrix D, which I have mentioned earlier. Substituting the original values for A and defining  $D_{U_k} = \sum_{i=1}^N d_{i,i} U^i U^{iT}$  and  $W_{U_k} = \sum_{i=1}^N \sum_{j=1}^N w_{i,i} U^i U^{jT}$ :

$$\begin{aligned}
J(V) &= \frac{\text{trace}(\sum_{i=1}^N V^T U^i U^{iT} V d_{i,i} - \sum_{i=1}^N \sum_{j=i}^N V^T U^i U^{jT} V w_{i,j})}{\text{trace}(\sum_i^N V^T U^i U^{iT} V d_{i,i})} \\
&= \\
&= \frac{\text{trace}(V^T (\sum_{i=1}^N d_{i,i} U^i U^{iT}) V - V^T (\sum_{i=1}^N \sum_{j=i}^N V^T U^i U^{jT} V w_{i,j}))}{\text{trace}(\sum_i^N V^T U^i U^{iT} V d_{i,i})} \tag{4.8} \\
&= \\
&= \frac{\text{trace}(V^T D_U V - V^T W_U V)}{\text{trace}(V^T D_U V)}
\end{aligned}$$

which can be solved, however there are several solutions to this problem. One way to make the solution unique is to add a constraint, which is:

$$\text{trace}(V^T D_u V) = 1 \tag{4.9}$$

The problem now is:

$$\begin{aligned} \min J(V) &= \text{trace}(V^T D_U V - V^T W_U V) & (4.10) \\ \text{s. t. } \text{trace}(V^T D_U V) &= 1 \end{aligned}$$

Now I construct the Lagrangian function to transform the problem from a constrained to an unconstrained minimization with one more parameter to take into account:

$$L(V) = \text{trace}(V^T D_U V - V^T W_U V) - \lambda (\text{trace}(V^T D_U V) - 1) \quad (4.11)$$

From the fact that D and W are symmetric, I can write the following:

$$\begin{aligned} \frac{dL(V)}{dV} &= (V^T (D_U - W_U))^T + (D_U - W_U)V & (4.12) \\ &- \lambda ((V^T D_U)^T + D_U V) = \\ &= ((D_U - W_U)^T + D_U - W_U)V - \lambda (D_U^T - D_U)V = \\ &= 2(D_U - W_U)V - 2\lambda D_U V = 0 \end{aligned}$$

From which I obtain the equation stated at the very beginning of the explanation of the GDR algorithm to compute the matrix V for each mode. The additional parameter lambda is specified by the user and I set it to 1.001 so that convergence and minimization are ensured. I will conclude the explanation of the GDR algorithm by specifying that the projection matrix for each mode strengthens and propagates the correlations among the modes while also incorporating the intertemporal dependencies coming from “close” tensors as indicated previously.

I will continue now with the presentation of the Tensor Regression algorithm and I will follow the same order as I did for the GDR algorithm.

---



---

<b>Input:</b>	The training tensor stream $\mathcal{X}_i _{i=1}^N \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and the associated indicators $y_i _{i=1}^N \in \mathbb{R}$ .
<b>Output:</b>	The mapped tensor steam $\tilde{\mathcal{X}}_i _{i=1}^N \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ , where $J_k \leq I_k$ .

---

<b>Step 1:</b>	Calculate the weight matrix $\mathbf{W}$ ;
<b>Step 2:</b>	From $k = 1$ to 3
<b>Step 3:</b>	From $i = 1$ to $N$
<b>Step 4:</b>	Decompose the original tensor $\mathcal{X}_i$ into $C_i \times_1 \mathbf{U}_1^i \times_2 \mathbf{U}_2^i \times_3 \mathbf{U}_3^i$ by Tucker decomposition;
<b>Step 5:</b>	End
<b>Step 6:</b>	$\mathbf{D}_{U_k} = \sum_{i=1}^N d_{i,i} \mathbf{U}_k^i \mathbf{U}_k^{iT}$ ;
<b>Step 7:</b>	$\mathbf{W}_{U_k} = \sum_{i=1}^N \sum_{j=i}^N w_{i,j} \mathbf{U}_k^i \mathbf{U}_k^{iT}$ ;
<b>Step 8:</b>	Obtain $\mathbf{V}_k$ by solving $(\mathbf{D}_{U_k} - \mathbf{W}_{U_k})\mathbf{V}_k = \lambda \mathbf{D}_{U_k} \mathbf{V}_k$ ;
<b>Step 9:</b>	End;
<b>Step 10:</b>	From $i = 1$ to $N$
<b>Step 11:</b>	$\tilde{\mathcal{X}}_i = C_i \times_1 (\mathbf{V}_1^T \mathbf{U}_1^i) \times_2 (\mathbf{V}_2^T \mathbf{U}_2^i) \times_3 (\mathbf{V}_3^T \mathbf{U}_3^i)$ ;
<b>Step 12:</b>	End.

---



---

Fig. 15 Global Dimensionality Reduction Algorithm (Quan Li et. ot. [2016])

This is fundamentally an SVR problem or Support Vector Regression. As specified, the inputs are the stream of tensors that underwent the process of Global Dimensionality Reduction and the “price indicators”, meaning the matrix of average prices for company  $j$  at time  $i$ . As output, the model will give back the theoretical price, given by the multi-mode product of the input tensor at time  $i$  by each weighting matrix for each mode of the information space, plus a constant term “ $b$ ” which is commonly referred to as the bias. Also the model outputs the slack variables, which are the values that allow tolerance of some violation of the margin that the user specifies. I will explain later on what it means. For the user-specified parameters,  $C$  is the constant and it is commonly chosen equal to one while  $\epsilon$  is the error that the user wants to tolerate. The constant must be a positive number but not necessarily an integer and it measures how much the user wants to allow the violation of the margin, so a lower value of the constant aims at generalizing the model while a higher value aims at classifying each point correctly. On the other hand,  $\epsilon$  indicates the wide the margin can be when classifying points. In my work I have chosen  $C=2$  and  $\epsilon=1$ . The explanation of the body of the algorithm is straightforward because the problem is indeed the Support Vector Regression problem, however it is performed in a higher dimensional context. In the context of SVR, or more generally Support Vector Machines or SVM, there is no way to compute the support vectors all at once but it can be done iteratively, since the problem is minimizing the inner product of the values that map the input matrix to a higher dimensional space, so that the data can be better separated. In general, SVM is very useful both in a supervised learning context but also in an unsupervised learning problem. The difference between the two cases is when the training inputs are labeled. In the

supervised learning case, as it is in my work, SVM tries to find the hyperplane that best separates the data points. I talk about a hyperplane because in the 2-dimensional case, with an input matrix, the hyperplane is a straight line, when considering the linear SVM. The true power of this methodology relies on the fact that they can also find the hyperplane the best separates the points in a non-linear context through the so called “kernel trick”, implicitly mapping the data to higher dimensions. The reason to do it is because some data might not be linearly separable in 2 dimensions but they could be in N dimensions. Indeed, SVM could potentially project the data to an infinite dimensional space. The role of epsilon is in the definition of the so called margin, so the distance from the hyperplane which is tolerated when classifying the data. Points that rely inside the margin are classified according to how close they are to either the negative margin or the positive one. A picture can help focus these ideas.

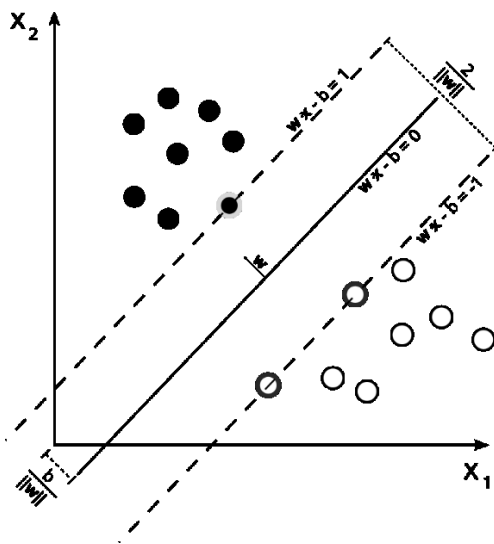


Fig. 16 (left) Support Vector Machine applied to a binary classification problem

(Google image search)

### Support Vector Regression

- Find a function,  $f(x)$ , with at most  $\epsilon$ -deviation from the target  $y$

The problem can be written as a convex optimization problem

$$\min \frac{1}{2} \|w\|^2$$

$$s.t. \quad y_i - w_1 \cdot x_i - b \leq \epsilon;$$

$$w_1 \cdot x_i + b - y_i \leq \epsilon;$$

C: trade off the complexity

What if the problem is not feasible?  
We can introduce slack variables  
(similar to soft margin loss function).



Fig. 17 (right) An example of Support Vector Regression (Google image search)

For the unsupervised learning, the idea is to let the data gravitate to the natural groups that are latent. When considering the Regression problem the idea is not to separate data as much as possible. Also there is a fundamental conceptual difference which is the fact that now the prediction is not aimed at classification but towards a real value, a number. In this sense, there are infinite values that one can find. In regression, the hyperplane is the plane that best fits the data, the latter being within the error margin indicated by epsilon. However, when considering noisy data, and stock prices are a good example of really noisy data, one has to allow some slack, which is why there are slack variables. They allow for some deviation from the chosen

interval around the hyperplane, which is usually a line but could also be a plane or even a higher dimensional object according to the dimensionality of the problem. This is where the constant parameter comes into play because it assigns a different importance to the slack variables according to its magnitude. The closer the constant  $C$  to zero, the closer the baseline case of SVR, the higher it is, the more the user is willing to tolerate deviations from the specified case. In general, what helped me personally to understand the difference is that with SVM classification problems we want the hyperplane just to separate the data and the margin around the hyperplane has to be as empty as possible, while the Regression problem with Support Vector Machines requires the hyperplane to interpolate data as best as it can while keeping as many data points as possible within the margin. From the mathematical point of view the problem is:

$$\begin{aligned} \min_{u,v,b} J(u, v, b) &= \frac{1}{2} \|uv^T\|^2 & (4.13) \\ \text{subject to } & \begin{cases} y_i - \langle X_i, uv^T \rangle - b \leq \varepsilon \\ \langle X_i, uv^T \rangle + b - y_i \leq \varepsilon \end{cases} \end{aligned}$$

The convex optimization problem comes from the specification that the mapping function  $f(X) = u^T X v + b$  can be rewritten as  $f(X) = \langle X, u^T v \rangle + b$  where  $\langle \cdot \rangle$  is the inner product, given the training and test sets composed by the couples  $(X, y)$ . The model complexity is equal to  $\|u^T v\|^2$ . I must remark that this is the specification of the problem in the two dimensional case, but it is immediately generalizable to the  $n$  dimensional case. Assuming that a function that approximates the data with an error less than epsilon exists, the problem is the following:

$$\begin{aligned} \min_{u,v,b,\xi_i,\xi_i^*} J(u, v, b, \xi_i, \xi_i^*) &= \frac{1}{2} \|uv^T\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) & (4.14) \\ \text{subject to } & \begin{cases} y_i - \langle X_i, uv^T \rangle - b \leq \varepsilon + \xi_i \\ \langle X_i, uv^T \rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i^*, \xi_i \geq 0, \quad i = 1, \dots, N \end{cases} \end{aligned}$$

where the slack variables have been introduced in the objective function. Now it is just a problem of finding  $u$  and  $v$  and this can be done by repeatedly optimizing the function for several values of  $u$  and  $v$ , each found from the previous iteration. Starting from  $u$ , it is set equal to a vector of ones:  $u = (1, 1, \dots, 1, 1)$ , so the problem becomes:

$$x_i = X^i u \quad (4.15)$$

$$\beta_1 = \|u\|^2 \quad (4.16)$$

$$\min_{v, b, \xi_i, \xi_i^*} J(v, b, \xi_i, \xi_i^*) = \frac{1}{2} \beta_1 \|v\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (4.17)$$

$$\text{subject to } \begin{cases} y_i - v^T x_i - b \leq \varepsilon + \xi_i \\ v^T x_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i^*, \xi_i \geq 0 \quad i = 1, \dots, N \end{cases}$$

From this optimization one can find v. Following the same procedure, setting the following:

$$\hat{x}_i = X_i v \quad (4.18)$$

$$\beta_2 = \|v\|^2 \quad (4.19)$$

one can find u from (4.20):

$$\min_{u, b, \xi_i, \xi_i^*} J(u, b, \xi_i, \xi_i^*) = \frac{1}{2} \beta_2 \|u\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (4.20)$$

$$\text{subject to } \begin{cases} y_i - u^T \hat{x}_i - b \leq \varepsilon + \xi_i \\ u^T \hat{x}_i + b - y_i \leq \varepsilon + \xi_i^* \\ \xi_i^*, \xi_i \geq 0 \quad i = 1, \dots, N \end{cases}$$

The reason why this algorithm works relies on the fact that the minimization of the objective function in each iteration ensures that a new minimum is always found each time u or v is updated. The convergence criterion is specified by the user. Since it is a squared function zero could be a good stopping criterion but there are also more reliable ways, such as derivative methods. The stopping criterion used in the sk-learn python library, which I have used, is the idea that the iterative procedure stops as soon as there is little to no improvement in the minimization process. However, if computations become too expensive or slow, one can even set a hard limit of say one-thousand iterations. In my code it was not necessary to introduce a hard limit on the iterations but it was necessary to perform some sort of validation, due to the few data available. What I decided to do was performing cross-validation using K-Folds since

I am assuming there is no intertemporal dependency. In my case, the SVR was performed with a constant value of 2 and an error no greater than 1, my kernel was the rbf or radial basis function. This concludes the explanation of the methodology and the model that I have used. Now I will focus on explaining the trading strategy that I have designed based on these findings.

## **Section 5: Conclusions**



Here I will present some conclusions, namely how to use in a trading context the tensor regression and propose a strategy that could be profitable. After having trained the model, I then design a trading strategy using my predictions. The trading strategies that can be used are various such as a simple Moving Average crossing strategy. This strategy is basically the “hello world” of the trading community because it generates a signal to buy each time the long term moving average crosses the short term moving average, indicating a potential positive trend while it generates a signal to sell when the short term moving average crosses the long term one, indicating a potential bear phase of the market. There are also other types of trading strategies such as the mean reversion strategy, where the assumption is that prices tend to return to their average and thus one starts to trade on the basis of the fluctuations or the turtle trading strategy introduced by Dannis which is usually applied to futures. The latter states that you should buy futures on the 20-days low and sell on the 20-days high prices. The strategy that I chose was the mean reversion since I am focusing on the Directional Accuracy as the main metric to decide how to trade. Moreover, the Directional Accuracy that I was able to obtain for all the companies was of approximately 50%. It is equivalent to tossing a coin and guessing if the stock will move up or down but nonetheless the model is able to recognize the movement half of the times. This indicates that further data is required to improve the training of the model. I tried to perform some k-Fold validation, but it was still not enough. In order to check how good the strategy is, I backtested it on my entire dataset and then I designed a portfolio to check profit and losses. I should remind right now that I am not taking into consideration several other factors that are important and have been proven to be informative, namely the fact that there are brokers and market makers that operate into the market, there are delays in the reception and execution of orders, short-selling might be not allowed and finally there are transaction costs which I am not considering. I am assuming that all of the previous factors do not influence the prices but of course it can be an interesting path for improving these results. I previously mentioned the term “backtesting” which refers to the validation of a trading strategy on historical data, acting as if such strategy was implemented some periods of time before its actual implementation and deployment, thus simulating what would have happened in terms of profit and losses over the period considered. A good backtester is made of four essential elements but of course there are also other ways to improve it. My backtester has only the four essential ones:

1. The data handler
2. The strategy
3. The portfolio
4. The execution handler

The data handler is just a fancy name to address the dataset; usually it is called as such because many libraries have a DataHandler class object. The strategy is what I have previously and briefly described, needed to generate the buy and sell signals. The portfolio is necessary to keep track of profit and losses and finally the execution handler is the part of code responsible of instructing the machine to execute orders without the user's interaction. Everything was virtual of course, I did not put any capital at stake. Using my predictions I used the Cerebro class object and thus the backtester python package which allowed me to design and implement the trading strategy on my predictions. Remarkably, the growth of the initial capital over the considered period is exponential. As mentioned previously, there are several constraints that have not been considered, which would likely diminish such enormous returns. Before drawing some conclusions, I will explain with a little more depth the strategy that I have used and the Cerebro and the backtester python library. About the strategy, the idea behind it is almost naïve: prices will return to the average, any deviation can be exploited, thus any stock that is below the market average will be bought while any stock above the average will be sold. The formula on which the strategy is based is the following:

$$w_i = \frac{-(r_i - r_m)}{\sum_k |r_k - r_m|} \quad (5.1)$$

where  $r_m$  is the market return. Each stock has a weight  $w$  attached to it computed relatively to all the other stocks in the portfolio. About the Cerebro and the backtester python library, as the developers state in their documentation page, the idea is to make algorithmic trading simple and straightforward. They were able to design a library which can accept any kind of strategy and then backtest it without any intervention by the user. The library even offers the possibility to develop signals that can be then used by the trader to buy or sell, or skip entirely the trader and place orders autonomously. It is accessible to both professionals, who can exploit several functionalities that are precluded to the retail investors, such as more complex strategies or High-Frequency Trading, and small time traders who want to try and automate some of the time consuming work behind day trading.

In this work I wanted to explore how far I could go in terms of improvement of existing methodologies to trade. Also, I wanted to provide some kind of roadmap to manipulate tensors and apply them to the financial context to try and obtain better predictions of stock prices, which are notoriously hard, even impossible, to predict exactly. There is still a long road ahead, since this work is far from conclusive and was only a first step in a field that has to be explored deeply, with the right tools and methodologies. First of all, the results obtained are based on a dataset which is too small for the raw power that tensors provide. Ideally one should acquire

and store data in the same way that I did for a year or so, group everything and then train the model, however this requires much more computational power. Furthermore, as a beginner programmer, the code in the appendix is in no way modular or replicable, unless one changes all the measurements accordingly. Ideally, the OOP (Object-Oriented Programming) paradigm can be applied and then elaborate a more “pythonic” way to approach this type of analysis. Besides problems strictly related to the amount of data and the code itself, other important remarks and flaws to be addressed in the future are the heavy assumptions imposed to the analysis. The fact that I am able to get such a high return is of course impossible, it is too high for a single week of trades. There are several other roles to be considered when creating the tensor stream for the supervised learning regression model with support vectors such as the role of other institutional investors and their impact on prices, the transaction costs and most importantly intertemporal dependence of prices, to name a few. Once all of these elements have been taken into account one can think of trying other strategies and compare them. One element that emerges is that the role of tensor is crucial in solving complex problems, speeding up calculations and in general providing more useful insights when several sources of information collide in a single information space with higher dimensionality. The following plots (figures 18 and 19) are the results in terms of returns on an initial capital of 1000 US dollars, trading from 9:30 on the 17-08-2020 up until 15:58:00 on the 21-08-2020:

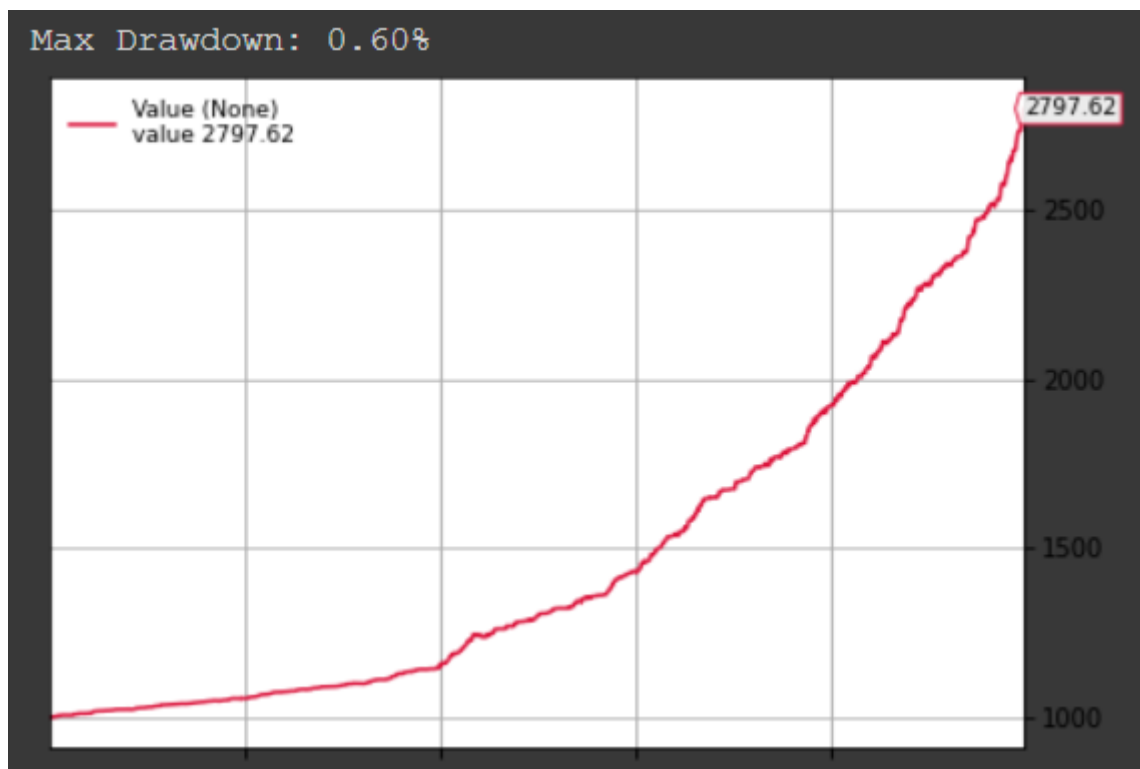


Fig 18 Backtesting the mean reversion strategy with the predictions of the model, weekly return of 179%

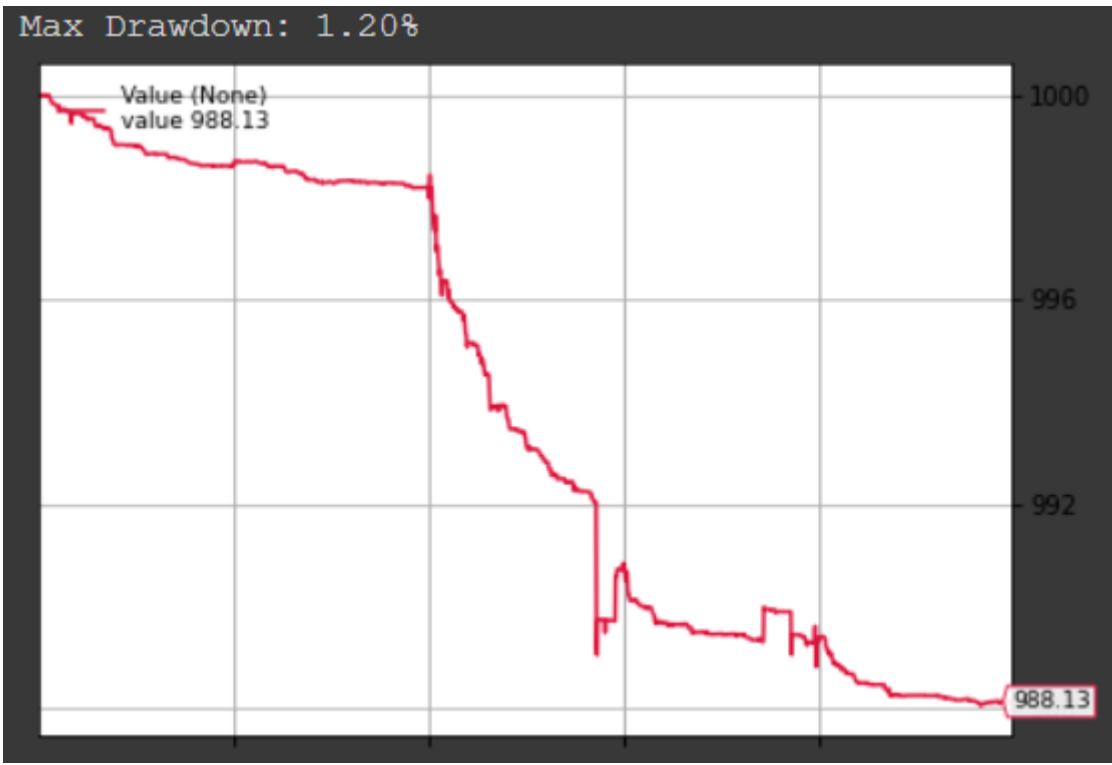


Fig. 19 Backtesting the mean reversion strategy using the actual prices, the loss is approximately -2%

## APPENDIX

All the code that I have written for this work is reported as I used it. I will indicate through titles the names of the scripts.

### Preliminary analysis and graphs

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import planarity

# Custom Functions

def sort_graph_edges(G):
    sorted_edges = []
    for source, dest, data in sorted(G.edges(data=True),
                                     key=lambda x: x[2]['weight']):
        sorted_edges.append({'source': source,
                             'dest': dest,
                             'weight': data['weight']})
    return sorted_edges

def compute_PMFG(sorted_edges, nb_nodes):
    PMFG = nx.Graph()
    for edge in sorted_edges:
        PMFG.add_edge(edge['source'], edge['dest'])
        if not planarity.is_planar(PMFG):
            PMFG.remove_edge(edge['source'], edge['dest'])
        if len(PMFG.edges()) == 3 * (nb_nodes - 2):
            break
    return PMFG

# importing and indexing data

my_data = pd.read_excel(r"C:\Users\user\Desktop\Master's Thesis\data
thesis.xlsx")
my_data["Dates"] = pd.to_datetime(my_data["Dates"],
infer_datetime_format=True)
indexed_data = my_data.set_index(["Dates"])

# removing NaN and isolating labels

headers = indexed_data.columns
indexed_data_noNaN = indexed_data.fillna(0)

# isolating labels only for companies
# restricting dataset to the first 100 firms due to computational problems

headers_only_company = []
for company_name in range(100):
    headers_only_company.append(headers[company_name*4])
df = pd.DataFrame(headers_only_company, columns=["company names"])
df.to_csv("company names.csv")

company_price_data = indexed_data_noNaN[headers_only_company]
```

```

# There are 100 companies so the graph is not very clear

company_price_data_part1 = indexed_data_noNaN[headers_only_company[0:11]]
company_price_data_part2 = indexed_data_noNaN[headers_only_company[11:22]]
company_price_data_part3 = indexed_data_noNaN[headers_only_company[22:33]]
company_price_data_part4 = indexed_data_noNaN[headers_only_company[33:44]]
company_price_data_part5 = indexed_data_noNaN[headers_only_company[44:55]]
company_price_data_part6 = indexed_data_noNaN[headers_only_company[55:66]]
company_price_data_part7 = indexed_data_noNaN[headers_only_company[66:77]]
company_price_data_part8 = indexed_data_noNaN[headers_only_company[77:88]]
company_price_data_part9 = indexed_data_noNaN[headers_only_company[88:]]

# data visualization

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part1)
plt.legend(company_price_data_part1)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part2)
plt.legend(company_price_data_part2)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part3)
plt.legend(company_price_data_part3)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part4)
plt.legend(company_price_data_part4)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part5)
plt.legend(company_price_data_part5)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part6)
plt.legend(company_price_data_part6)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part7)
plt.legend(company_price_data_part7)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part8)
plt.legend(company_price_data_part8)
plt.show()

plt.xlabel("Daily Observations")
plt.ylabel("Closing prices")
plt.plot(company_price_data_part9)
plt.legend(company_price_data_part9)
plt.show()

# PMFG

```

```

corr_matr = np.corrcoef(company_price_data.T)
my_nb_nodes = 100
complete_graph = nx.Graph()
for i in range(my_nb_nodes):
    for j in range(i+1, my_nb_nodes):
        complete_graph.add_edge(i, j, weight=corr_matr[i, j])

my_sorted_edges = sort_graph_edges(complete_graph)

my_PMFG = compute_PMFG(my_sorted_edges, len(complete_graph.nodes))
companies = {}
for i in range(100):
    companies[i] = headers_only_company[i]

nx.draw_networkx(my_PMFG, pos=nx.spring_layout(my_PMFG), with_labels=True,
node_size=150, node_color='w',
                labels=companies, font_size=4, font_color='r',
font_weight='heavy', width=0.1)
plt.savefig("PMFG.pdf")

# descriptive statistics of prices
descriptive_statistics_g1 = company_price_data_part1.describe()
print(descriptive_statistics_g1)
descriptive_statistics_g2 = company_price_data_part2.describe()
print(descriptive_statistics_g2)
descriptive_statistics_g3 = company_price_data_part3.describe()
print(descriptive_statistics_g3)
descriptive_statistics_g4 = company_price_data_part4.describe()
print(descriptive_statistics_g4)
descriptive_statistics_g5 = company_price_data_part5.describe()
print(descriptive_statistics_g5)
descriptive_statistics_g6 = company_price_data_part6.describe()
print(descriptive_statistics_g6)
descriptive_statistics_g7 = company_price_data_part7.describe()
print(descriptive_statistics_g7)
descriptive_statistics_g8 = company_price_data_part8.describe()
print(descriptive_statistics_g8)
descriptive_statistics_g9 = company_price_data_part9.describe()
print(descriptive_statistics_g9)

# Minute Data

# extension = 'txt'
# all_filenames = [i for i in
glob.glob(r'C:\Users\user\PycharmProjects\test\dati compagnia minuto
tickers\*.{}'.format(extension))]
# combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames])
# combined_csv.to_csv("minute_price_data.csv")

data_minute = pd.read_excel(r"C:\Users\user\Desktop\Master's Thesis\data
minute.xlsx")
data_minute["dates"] = pd.to_datetime(data_minute["dates"],
infer_datetime_format=True)
indexed_data_minute = data_minute.set_index(["dates", "minute"])

indexed_data_minute_NoNaN = indexed_data_minute.replace(to_replace=0,
value=np.nan).fillna(method='pad', axis=0)

headers_avg = []
for company_name in headers_only_company:
    headers_avg.append(company_name + " " + "avg")

data_minute_prices = indexed_data_minute[headers_avg]

```

```

indexed_data_minute_NoNaN = indexed_data_minute_NoNaN.drop('minute',
axis=1)

company_price_data_minute_part1 = data_minute_prices[headers_avg[0:11]]
company_price_data_minute_part2 = data_minute_prices[headers_avg[11:22]]
company_price_data_minute_part3 = data_minute_prices[headers_avg[22:33]]
company_price_data_minute_part4 = data_minute_prices[headers_avg[33:44]]
company_price_data_minute_part5 = data_minute_prices[headers_avg[44:55]]
company_price_data_minute_part6 = data_minute_prices[headers_avg[55:66]]
company_price_data_minute_part7 = data_minute_prices[headers_avg[66:77]]
company_price_data_minute_part8 = data_minute_prices[headers_avg[77:88]]
company_price_data_minute_part9 = data_minute_prices[headers_avg[88:]]

plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part1)
plt.legend(company_price_data_minute_part1)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part2)
plt.legend(company_price_data_minute_part2)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part3)
plt.legend(company_price_data_minute_part3)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part4)
plt.legend(company_price_data_minute_part4)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part5)
plt.legend(company_price_data_minute_part5)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part6)
plt.legend(company_price_data_minute_part6)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part7)
plt.legend(company_price_data_minute_part7)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part8)
plt.legend(company_price_data_minute_part8)
plt.show()
plt.xlabel("Minute Observations: 17/08/2020-21/08/2020")
plt.ylabel("Avg. open-close prices")
plt.plot(company_price_data_minute_part9)
plt.legend(company_price_data_minute_part9)
plt.show()

descriptive_statistics_g1 = company_price_data_minute_part1.describe()
print(descriptive_statistics_g1)
descriptive_statistics_g2 = company_price_data_minute_part2.describe()

```



```

print(descriptive_statistics_g2)
descriptive_statistics_g3 = company_price_data_minute_part3.describe()
print(descriptive_statistics_g3)
descriptive_statistics_g4 = company_price_data_minute_part4.describe()
print(descriptive_statistics_g4)
descriptive_statistics_g5 = company_price_data_minute_part5.describe()
print(descriptive_statistics_g5)
descriptive_statistics_g6 = company_price_data_minute_part6.describe()
print(descriptive_statistics_g6)
descriptive_statistics_g7 = company_price_data_minute_part7.describe()
print(descriptive_statistics_g7)
descriptive_statistics_g8 = company_price_data_minute_part8.describe()
print(descriptive_statistics_g8)
descriptive_statistics_g9 = company_price_data_minute_part9.describe()
print(descriptive_statistics_g9)

```

## Webscraping for news headlines

```

from selenium import webdriver
import time
import re
import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

"""This code finds news headlines from reuters for the first 100 companies
of the dataset and then performs
a Vectorization of the words of the headlines. Then they will be used for
building the tensors mode"""

headers_only_company = pd.read_csv("company names.csv")
headers_only_company = headers_only_company.drop("Unnamed: 0", 1)

# importing Chrome driver to automate the scraping
driver =
webdriver.Chrome(r'C:\Users\user\PycharmProjects\test\venv\Lib\site-
packages\selenium\chromedriver.exe')
driver.get('https://www.reuters.com/news/archive/businessnews?view=page&pag
e=5&pageSize=10')

# storing headlines and converting them to text

count = 0
headlines = []
dates = []
for company_name in headers_only_company.values:
    try:
        searchButton = driver.find_element_by_class_name("search-icon")
        searchButton.click()
        time.sleep(1)
        searchField = driver.find_element_by_class_name("search-field")
        searchField.send_keys(company_name)
        searchField.send_keys(u'\ue007')
        time.sleep(1)
        loadMoreButton = driver.find_element_by_class_name("search-result-
more-txt")
        # driver.execute_script("window.scrollTo(0,
document.body.scrollHeight)")
        time.sleep(1)
        loadMoreButton.click()

```

```

        time.sleep(1)
        loadMoreButton.click()
        time.sleep(1)
        loadMoreButton.click()
        time.sleep(1)
        loadMoreButton.click()
        time.sleep(1)
        loadMoreButton.click()
        time.sleep(1)
        loadMoreButton.click()
        time.sleep(1)
        loadMoreButton.click()
        time.sleep(1)
        news_headlines = driver.find_elements_by_class_name("search-result-
title")
        news_dates = driver.find_elements_by_class_name("search-result-
timestamp")
        for headline in news_headlines:
            headlines.append(headline.text)
            print(headline.text)
        for date in news_dates:
            dates.append(date.text)
            print(date.text)
            count = count + 1
        print("CLICKED!!!")
    except Exception as e:
        print(e)
        continue

df = pd.DataFrame(headlines, columns=['Reuters news headline'])
df.drop_duplicates()
df.to_csv('reuters news headlines.csv')

# pre-processing the headlines

processed_headlines = []

for headline in headlines:
    # Remove all the special characters
    processed_headline = re.sub(r'\W', ' ', headline)
    # remove all single characters
    processed_headline = re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_headline)
    # Remove single characters from the start
    processed_headline = re.sub(r'^[a-zA-Z]\s+', ' ', processed_headline)
    # Substituting multiple spaces with single space
    processed_headline = re.sub(r'\s+', ' ', processed_headline,
flags=re.I)
    # Removing prefixed 'b'
    processed_headline = re.sub(r'^b\s+', '', processed_headline)
    # Converting to Lowercase
    processed_headline = processed_headline.lower()
    processed_headlines.append(processed_headline)

# downloading the stopwords dictionary
nltk.download('stopwords')

# text to numeric conversion, min_df=5 and max_df=0.7
tfidfconverter = TfidfVectorizer(max_features=2000, min_df=5, max_df=0.7,
stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(processed_headlines).toarray()

# storing the data for further usage
df = pd.DataFrame(X)
df.set_index(dates)

```

```
df.to_csv('tfidf_text2num.csv')
print('Done!')
```

## Better Webscraping for news headlines

```
# Import libraries
from selenium import webdriver
from urllib.request import urlopen, Request
from bs4 import BeautifulSoup as bs
import time
import pandas as pd
import matplotlib.pyplot as plt
# NLTK VADER for sentiment analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')

finwiz_url = 'https://finviz.com/quote.ashx?t='

tickers = []

driver =
webdriver.Chrome(r'C:\Users\user\PycharmProjects\test\venv\Lib\site-
packages\selenium\chromedriver.exe')
driver.get('https://finasko.com/sp-500-companies/')

stock_symbol = driver.find_elements_by_class_name('stock-symbol')
time.sleep(1)
for i in stock_symbol:
    tickers.append(i.text)
loadmoreButton = driver.find_element_by_css_selector('a.post-page-numbers')
loadmoreButton.click()
time.sleep(1)
stock_symbol_2 = driver.find_elements_by_class_name('stock-symbol')
for j in stock_symbol_2:
    tickers.append(j.text)
stock_symbol_last = 'ECL'
tickers.append(stock_symbol_last)

tickers_not = tickers.pop(3)
tickers[90] = tickers[90] + 'C'

news_tables = {}
count = 0
for ticker in tickers:
    url = finwiz_url + ticker
    print(url)
    count += 1
    print(count)
    req = Request(url=url, headers={'user-agent': 'my-app/0.0.1'})
    response = urlopen(req)
    # Read the contents of the file into 'html'
    html = bs(response)
    # Find 'news-table' in the Soup and load it into 'news_table'
    news_table = html.find(id='news-table')
    # Add the table to our dictionary
    news_tables[ticker] = news_table

parsed_news = []
```

```

# Iterate through the news
for file_name, news_table in news_tables.items():
    # Iterate through all tr tags in 'news_table'
    for x in news_table.findAll('tr'):
        # read the text from each tr tag into text
        # get text from a only
        text = x.a.get_text()
        # splite text in the td tag into a list
        date_scrape = x.td.text.split()
        # if the length of 'date_scrape' is 1, load 'time' as the only
element

        if len(date_scrape) == 1:
            time = date_scrape[0]

        # else load 'date' as the 1st element and 'time' as the second
        else:
            date = date_scrape[0]
            time = date_scrape[1]
        # Extract the ticker from the file name, get the string up to the
1st '_'
        ticker = file_name.split('_')[0]

        # Append ticker, date, time and headline as a list to the
'parsed_news' list
        parsed_news.append([ticker, date, time, text])

# Instantiate the sentiment intensity analyzer
vader = SentimentIntensityAnalyzer()

# Set column names
columns = ['ticker', 'date', 'time', 'headline']

# Convert the parsed_news list into a DataFrame called
'parsed_and_scored_news'
parsed_and_scored_news = pd.DataFrame(parsed_news, columns=columns)

# Iterate through the headlines and get the polarity scores using vader
scores =
parsed_and_scored_news['headline'].apply(vader.polarity_scores).tolist()

# Convert the 'scores' list of dicts into a DataFrame
scores_df = pd.DataFrame(scores)

# Join the DataFrames of the news and the list of dicts
parsed_and_scored_news = parsed_and_scored_news.join(scores_df,
rsuffix='_right')

# Convert the date column from string to datetime
parsed_and_scored_news['date'] =
pd.to_datetime(parsed_and_scored_news.date).dt.date

plt.rcParams['figure.figsize'] = [100, 60]

# Group by date and ticker columns from scored_news and calculate the mean
mean_scores = parsed_and_scored_news.groupby(['ticker', 'date']).mean()

# Unstack the column ticker
mean_scores = mean_scores.unstack()

# Get the cross-section of compound in the 'columns' axis
mean_scores = mean_scores.xs('compound', axis="columns").transpose()

```

```

# Plot a bar chart with pandas
mean_scores.plot(kind='bar')
plt.grid()
plt.show()

parsed_and_scored_divided = []
for i in range(0, 10000, 100):
    parsed_and_scored_divided.append(parsed_and_scored_news.iloc[i:i+100,
:])

for i in range(len(parsed_and_scored_divided)):
    parsed_and_scored_divided[i].to_csv('news_n_{}.csv'.format(i))

parsed_and_scored_news.to_csv('parsed and scored news.csv')

for i in range(100):
    definitive_df = pd.concat([parsed_and_scored_divided,
pd.read_csv('news_n_{}'.format(i))])

```

## Twitter scraping for tweets

```

import tweepy
from tweepy import OAuthHandler
import re
import time
import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier

""" This code is to obtain tweets from the social network, using the tweepy
library.

WARNING: IT TAKES AT LEAST 11 HOURS TO GATHER THE TWEETS

It also classifies the sentiment conveyed by the scraped tweets, taking
advantage of a freely available
data-set that can be found at
https://raw.githubusercontent.com/kolaveridi/kaggle-Twitter-US-Airline-Sentiment-/master/Tweets.csv
"""

# twitter api keys
# Bearer_token =
'AAAAAAAAAAAAAAAAAAAAAAAAAII1PGQEAAAAAAZE2GLdMtpC2h7muRQppVpzsAhGc%3DTIsv9pbSTtD1
uxJcdqbU7wAFTXdZ9hrZau2QSoIqssNrk4hWQG'

# Twitter keys for accessing their API

twitter_api_key = 'dXPA73iP7fYljlU0NI19uECRD'
twitter_secret_api_key =
'RnzSQaOYErrefyCB2HPkE1kOAdt2h308FX7VvLOWfbm3d0Kikn'
twitter_access_token = '1171770799-DnWcJr2CrS11fWao2qpq09pq9x5vU77gG9iV6cv'
twitter_access_token_secret =
'yR7dOz4gn5PDZ35nhl1UaYDtSFSAOkqUcWxid3ghK8h3e8'

authorizer = OAuthHandler(twitter_api_key, twitter_secret_api_key)
authorizer.set_access_token(twitter_access_token,

```

```

twitter_access_token_secret)

api = tweepy.API(authorizer, wait_on_rate_limit=True)

# importing data for company names

prices = pd.read_excel(r"C:\Users\user\PycharmProjects\Scoop_project\data
thesis.xlsx")
prices["Dates"] = pd.to_datetime(prices["Dates"],
infer_datetime_format=True)
indexed_data = prices.set_index(["Dates"])

headers = indexed_data.columns
indexed_data_noNaN = indexed_data.fillna(0)

headers_only_company = []
for company_name in range(100):
    headers_only_company.append(headers[company_name*4])

# scraping tweets

all_tweets = []
count = 0
for company_name in headers_only_company:
    search_query = company_name
    print(company_name)
    time.sleep(1)
    for tweet_object in tweepy.Cursor(api.search, q=search_query+ " -
filter:retweets",
                                     lang='en',
result_type='recent').items(150):
        time.sleep(2)
        all_tweets.append(tweet_object.text)
        count += 1
# saving the dataset for further usage later on

all_tweets_df = pd.DataFrame(all_tweets, columns=['scraped tweets'])
all_tweets_df.to_csv("all_tweets.csv")

# download the stopwords dictionary
nltk.download('stopwords')

# importing another dataset which was already classified
support_tweets =
pd.read_csv("https://raw.githubusercontent.com/kolaveridi/kaggle-Twitter-
US-Airline-Sentiment-/master/Tweets.csv")

X = support_tweets.iloc[:, 10].values
y = support_tweets.iloc[:, 1].values

# data pre-processing
support_processed_tweets = []

for support_tweet in range(0, len(X)):
    # Remove all the special characters
    support_processed_tweet = re.sub(r'\W', ' ', str(X[support_tweet]))
    # remove all single characters
    support_processed_tweet = re.sub(r'\s+[a-zA-Z]\s+', ' ',
support_processed_tweet)
    # Remove single characters from the start
    support_processed_tweet = re.sub(r'\^[a-zA-Z]\s+', ' ',
support_processed_tweet)
    # Substituting multiple spaces with single space

```

```

support_processed_tweet = re.sub(r'\s+', ' ', support_processed_tweet,
flags=re.I)
# Removing prefixed 'b'
support_processed_tweet = re.sub(r'^b\s+', '', support_processed_tweet)
# Converting to Lowercase
support_processed_tweet = support_processed_tweet.lower()
support_processed_tweets.append(support_processed_tweet)

# text to numeric conversion using TF-IDF
tfidfconverter = TfidfVectorizer(max_features=2000, min_df=5, max_df=0.7,
stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(support_processed_tweets).toarray()

# text sentiment classification, remark: using all the dataset because
these are only support tweets
text_classifier = RandomForestClassifier(n_estimators=100, random_state=0)
text_classifier.fit(X, y)

processed_tweets = []
sentiments = []

# assigning the interesting tweets their sentiment
for tweet in all_tweets:
# Remove all the special characters
processed_tweet = re.sub(r'\W', ' ', tweet)
# remove all single characters
processed_tweet = re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_tweet)
# Remove single characters from the start
processed_tweet = re.sub(r'^[a-zA-Z]\s+', ' ', processed_tweet)
# Substituting multiple spaces with single space
processed_tweet = re.sub(r'\s+', ' ', processed_tweet, flags=re.I)
# Removing prefixed 'b'
processed_tweet = re.sub(r'^b\s+', '', processed_tweet)
# Removing https
processed_tweet = re.sub(r'https:\\[a-zA-Z]+', '', processed_tweet)
# Converting to Lowercase
processed_tweet = processed_tweet.lower()
sentiment =
text_classifier.predict(tfidfconverter.transform([processed_tweet]).toarray
())
processed_tweets.append(processed_tweet)
sentiments.append(sentiment)

# joining the two lists to create the data-set to be later used
df_1 = pd.DataFrame(processed_tweets, columns=['processed tweets'])
df_2 = pd.DataFrame(sentiments, columns=["sentiment"])
processed_tweets_df = df_1.join(df_2)
processed_tweets_df.to_csv('processed tweets.csv')
print('Done!')

```

## Better Twitter scraping for tweets

```

import time
import re
import pandas as pd
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
import GetOldTweets3 as got

```

```

prices = pd.read_excel(r"C:\Users\user\PycharmProjects\Scoop_project\data
thesis.xlsx")
prices["Dates"] = pd.to_datetime(prices["Dates"],
infer_datetime_format=True)
indexed_data = prices.set_index(["Dates"])

headers = indexed_data.columns
indexed_data_noNaN = indexed_data.fillna(0)

headers_only_company = []
for company_name in range(100):
    headers_only_company.append(headers[company_name*4])

since_date = '2020-08-17'
until_date = '2020-08-22'
# Creation of query object
count = 200
# Creation of list that contains all tweets
text_tweets = []
dates = []
for company_name in headers_only_company:
    text_query = company_name
    tweetCriteria =
got.manager.TweetCriteria().setQuerySearch(text_query).setSince(since_date)
.setUntil(until_date).setMaxTweets(count).setTopTweets(True)
    time.sleep(1)
    print("don't worry, I'm working on it...")
# Creating list of chosen tweet data
tweets = got.manager.TweetManager.getTweets(tweetCriteria)
for tweet in tweets:
    text_tweets.append(tweet.text)
    print(tweet.text)
    dates.append(tweet.date)
    print(tweet.date)
    time.sleep(1)

all_tweets_df = pd.DataFrame(data=text_tweets, columns=['scraped tweets'])
all_dates_df = pd.DataFrame(data=dates, columns=['dates'])
all_tweets_and_dates = all_tweets_df.join(all_dates_df)
all_tweets_and_dates.to_csv("all tweets.csv")

# download the stopwords dictionary
nltk.download('stopwords')

# importing another dataset which was already classified
support_tweets =
pd.read_csv("https://raw.githubusercontent.com/kolaveridi/kaggle-Twitter-
US-Airline-Sentiment-/master/Tweets.csv")

X = support_tweets.iloc[:, 10].values
y = support_tweets.iloc[:, 1].values

# data pre-processing
support_processed_tweets = []

for support_tweet in range(0, len(X)):
    # Remove all the special characters
    support_processed_tweet = re.sub(r'\W', ' ', str(X[support_tweet]))
    # remove all single characters
    support_processed_tweet = re.sub(r'\s+[a-zA-Z]\s+', ' ',
support_processed_tweet)
    # Remove single characters from the start
    support_processed_tweet = re.sub(r'^[a-zA-Z]\s+', ' ',

```



```

support_processed_tweet)
    # Substituting multiple spaces with single space
    support_processed_tweet = re.sub(r'\s+', ' ', support_processed_tweet,
flags=re.I)
    # Removing prefixed 'b'
    support_processed_tweet = re.sub(r'^b\s+', '', support_processed_tweet)
    # Converting to Lowercase
    support_processed_tweet = support_processed_tweet.lower()
    support_processed_tweets.append(support_processed_tweet)

# text to numeric conversion using TF-IDF
tfidfconverter = TfidfVectorizer(max_features=2000, min_df=5, max_df=0.7,
stop_words=stopwords.words('english'))
X = tfidfconverter.fit_transform(support_processed_tweets).toarray()

# text sentiment classification, remark: using all the dataset because
these are only support tweets
text_classifier = RandomForestClassifier(n_estimators=100, random_state=0)
text_classifier.fit(X, y)

processed_tweets = []
sentiments = []

# assigning the interesting tweets their sentiment
for tweet in text_tweets:
    # Remove all the special characters
    processed_tweet = re.sub(r'\W', ' ', tweet)
    # remove all single characters
    processed_tweet = re.sub(r'\s+[a-zA-Z]\s+', ' ', processed_tweet)
    # Remove single characters from the start
    processed_tweet = re.sub(r'^[a-zA-Z]\s+', ' ', processed_tweet)
    # Substituting multiple spaces with single space
    processed_tweet = re.sub(r'\s+', ' ', processed_tweet, flags=re.I)
    # Removing prefixed 'b'
    processed_tweet = re.sub(r'^b\s+', '', processed_tweet)
    # Removing https
    processed_tweet = re.sub(r'https:\\[a-zA-Z]+', '', processed_tweet)
    # Converting to Lowercase
    processed_tweet = processed_tweet.lower()
    sentiment =
text_classifier.predict(tfidfconverter.transform([processed_tweet]).toarray
())
    processed_tweets.append(processed_tweet)
    sentiments.append(sentiment)

# joining the two lists to create the data-set to be later used
df_1 = pd.DataFrame(processed_tweets, columns=['processed tweets'])
df_2 = pd.DataFrame(sentiments, columns=["sentiment"])
processed_tweets_df = df_1.join(df_2)
processed_tweets_df = processed_tweets_df.join(all_dates_df)
processed_tweets_df['dates'] = pd.to_datetime(processed_tweets_df['dates'],
infer_datetime_format=True)
processed_tweets_df = processed_tweets_df.set_index(['dates'])
processed_tweets_df.to_csv('processed tweets.csv')
print('Done!')

```

## Sentiment mode construction

```

import pandas as pd
import math
import re

```

```

import openpyxl
"""
This code imports all the data-sets previously constructed and then uses
the processed_tweets.csv file to further
develop the sentiment mode to be fed to the tensors. The description of the
computation of the moods is reported below.
There is also an additional part required to further clean up the tweets,
which had some parts that were not eliminated
during pre-processing.
"""

# importing data
processed_tweets_and_sentiment = pd.read_csv('processed_tweets.csv')
my_data = pd.read_excel(r"C:\Users\user\Desktop\Master's Thesis\data
thesis.xlsx")
my_data["Dates"] = pd.to_datetime(my_data["Dates"],
infer_datetime_format=True)
indexed_data = my_data.set_index(["Dates"])

headers = indexed_data.columns
indexed_data_noNaN = indexed_data.fillna(0)

# isolating labels only for companies
# restricting dataset to the first 100 firms due to computational problems

headers_only_company = []
for company_name in range(100):
    headers_only_company.append(headers[company_name*4])

# data-set with only factors
headers_factors = indexed_data_noNaN.drop(headers_only_company, 1)

# additional processing of tweets to remove links and other useless words
# for i in range(len(processed_tweets_and_sentiment)):
#     processed_tweets_and_sentiment['processed_tweets'][i] =
re.sub(r'https', r'\', processed_tweets_and_sentiment['processed
tweets'][i])
#     processed_tweets_and_sentiment['processed_tweets'][i] =
re.sub(r'\\s+[a-zA-Z]+\s', r'\',
processed_tweets_and_sentiment['processed_tweets'][i])
#     processed_tweets_and_sentiment['processed_tweets'][i] = re.sub(r'\\[a-
zA-Z0-9]+', r'', processed_tweets_and_sentiment['processed_tweets'][i])
#     processed_tweets_and_sentiment['processed_tweets'][i] = re.sub(r'^\s',
r'', processed_tweets_and_sentiment['processed_tweets'][i])

# processed_tweets_and_sentiment.to_csv('processed_tweets.csv')

# dataset of sentiment
# sentiments = processed_tweets_and_sentiment['sentiment']
# print(sentiments.head())

# building up the stock mood, the market mood and the intensity
# splitting the data-set in the number of tweets for each company, there
are almost 20k tweets, so 205 per company

# sentiment = sentiments.values.tolist()
splitted_sentiment = []
for i in range(0, len(processed_tweets_and_sentiment), 205):
    splitted_sentiment.append(processed_tweets_and_sentiment[i:i+205])
del splitted_sentiment[100]
# creating the data-set with sentiment for each tweet for each company

```

```

# overall number of positive, neutral and negative tweets
num_pos_tweets = 0
num_neut_tweets = 0
num_neg_tweets = 0
for df in splitted_sentiment:
    for sentiment in df['sentiment']:
        if sentiment == 'positive':
            num_pos_tweets += 1
        elif sentiment == 'neutral':
            num_neut_tweets += 1
        else:
            num_neg_tweets += 1
print('positive tweets: ', num_pos_tweets, 'neutral tweets: ',
num_neut_tweets, 'negative tweets: ', num_neg_tweets)

# computing the number of positive, neutral and negative tweets for each
stock
# First, I create three DataFrames for each sentiment and then use isin()
to put True instead of the desired sentiment
# Second, I loop through the columns and sum over them to get the number of
times the desired sentiment is present,
# then I store everything in a dedicated list
date_list = ['2020-08-21 ', '2020-08-20 ', '2020-08-19 ', '2020-08-18 ',
'2020-08-17 ']
pos_sums = []
neut_sums = []
neg_sums = []
count = 0
for df in splitted_sentiment:
    # df.drop('processed tweets', axis=1)
    count += 1
    print(count)
    df = df.reset_index()
    for i in range(len(df)):
        df['dates'][i] = re.sub(r'(\d+)*\+', r'', df['dates'][i])
        df['dates'][i] = re.sub(r'(\d+)*:', r'', df['dates'][i])
        df['dates'][i] = re.sub(r'00', r'', df['dates'][i])
    df_pos = df.loc[df['sentiment'] == 'positive'].isin(['positive'])
    df_neut = df.loc[df['sentiment'] == 'neutral'].isin(['neutral'])
    df_neg = df.loc[df['sentiment'] == 'negative'].isin(['negative'])
    for date in date_list:
        pos_sums.append(df_pos.loc[df['dates'] == date]['sentiment'].sum())
        neut_sums.append(df_neut.loc[df['dates'] ==
date]['sentiment'].sum())
        neg_sums.append(df_neg.loc[df['dates'] == date]['sentiment'].sum())

time_factor = []
for i in range(5):
    time_factor.append(1/math.exp(i/20))
    # weighted mood for each stock
weighted_pos = []
weighted_neut = []
weighted_neg = []
for i in range(0, 500, 5):
    weighted_pos.append(pos_sums[i]*time_factor[0])
    weighted_pos.append(pos_sums[i + 1] * time_factor[1])
    weighted_pos.append(pos_sums[i + 2] * time_factor[2])
    weighted_pos.append(pos_sums[i + 3] * time_factor[3])
    weighted_pos.append(pos_sums[i + 4] * time_factor[4])
    weighted_neut.append(neut_sums[i] * time_factor[0])
    weighted_neut.append(neut_sums[i + 1] * time_factor[1])
    weighted_neut.append(neut_sums[i + 2] * time_factor[2])
    weighted_neut.append(neut_sums[i + 3] * time_factor[3])

```

```

weighted_neut.append(neut_sums[i + 4] * time_factor[4])
weighted_neg.append(neg_sums[i] * time_factor[0])
weighted_neg.append(neg_sums[i + 1] * time_factor[1])
weighted_neg.append(neg_sums[i + 2] * time_factor[2])
weighted_neg.append(neg_sums[i + 3] * time_factor[3])
weighted_neg.append(neg_sums[i + 4] * time_factor[4])

# conversion in Series to later usage
# df_pos = pd.Series(data=weighted_pos, index=headers_only_company)
# df_neut = pd.Series(data=weighted_neut, index=headers_only_company)
# df_neg = pd.Series(data=weighted_neg, index=headers_only_company)

# Mood intensity for individual stocks
inten_neut = []
inten_neg = []
for i in range(500):
    inten_neut.append((weighted_pos[i] -
weighted_neut[i]) / (sum(weighted_pos) + sum(weighted_neut)))
    inten_neg.append((weighted_pos[i] - weighted_neg[i]) / (sum(weighted_pos)
+ sum(weighted_neg)))

keys = []
for company_name in headers_only_company:
    keys.append(company_name + '_' + 'pos')
    keys.append(company_name + '_' + 'neut')
    keys.append(company_name + '_' + 'neg')
    keys.append(company_name + '_' + 'intensity_neut')
    keys.append(company_name + '_' + 'intensity_neg')

sentiment_mode_dataset = {}
for i in range(0, 500, 5):
    sentiment_mode_dataset[keys[i]] = weighted_pos[i:i+5]
    sentiment_mode_dataset[keys[i+1]] = weighted_neut[i:i+5]
    sentiment_mode_dataset[keys[i+2]] = weighted_neg[i:i+5]
    sentiment_mode_dataset[keys[i+3]] = inten_neut[i:i+5]
    sentiment_mode_dataset[keys[i+4]] = inten_neg[i:i+5]

# Put everything in the data-set for the sentiment mode
sentiment_mode = pd.DataFrame(data=sentiment_mode_dataset)

sentiment_mode.to_excel('sentiment mode.xlsx')

```

At this point I stopped using PyCharm and started using Google Colaboratory. I should also remark that the scripts titled “better ...” are the effective ones, in the sense that it is from them that my data comes from, the others have been added for completeness and because they could be helpful to the reader.

### GDR\_and\_TensorRegression\_and\_MR\_trading\_strategy

```

"""GDR_and_TensorRegression_and_MR_trading_strategy.ipynb

```

*Automatically generated by Colaboratory.*

*Original file is located at*

```

https://colab.research.google.com/drive/1WGz9QgYkSy4Mw7Ekbvqpb5RlIpZK5FfV
"""

```

```

!pip install tensorly
!pip install scipy

```

```

!pip install backtrader

# Commented out IPython magic to ensure Python compatibility.
# %tensorflow_version 2.x
import math
import pandas as pd
import numpy as np
import scipy
from scipy import linalg
from scipy.optimize import minimize
import openpyxl
import tensorly as tl
from tensorly.decomposition import tucker
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

def tensor_distance(matrix):
    # Custom function to compute the matrix W, further explanation in the
    # code below
    tmp = np.zeros(shape=(1949, 1949))
    for i in range(1949):
        for j in range(99):
            tmp[i][j] = (abs(matrix[i+1][j+1]-matrix[i][j])/matrix[i][j])
            if tmp[i][j] > 0.05:
                tmp[i][j] = 0
            elif tmp[i][j] <= 0.05:
                tmp[i][j] = 1
    return tmp

def mape(actual, pred):
    # Custom function to create the Mean Absolute Percentage Error Metric
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100

def directional_accuracy(actual, pred):
    # Custom function to compute the mean directional accuracy
    actual, pred = np.array(actual), np.array(pred)
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(pred[1:] -
pred[:-1])).astype(int))

# monthly data import, highly inefficient but I needed the company names

my_data = pd.read_excel("data thesis.xlsx")
my_data["Dates"] = pd.to_datetime(my_data["Dates"],
infer_datetime_format=True)
indexed_data = my_data.set_index(["Dates"])

headers = indexed_data.columns
# indexed_data_noNaN = indexed_data.fillna(method='backfill', axis=1)

# isolating labels only for companies
# restricting dataset to the first 100 firms due to computational problems
headers_only_company = []
for company_name in range(100):
    headers_only_company.append(headers[company_name*4])

# minute data, actual dataset to be used
data_minute = pd.read_excel("data minute.xlsx")

```

```

data_minute["dates"] = pd.to_datetime(data_minute["dates"],
infer_datetime_format=True)
indexed_data_minute = data_minute.set_index(["dates", "minute"])

indexed_data_minute_NoNaN = indexed_data_minute.replace(to_replace=0,
value=np.nan).fillna(method='pad', axis=0)

headers_avg = []
variables_to_drop = []
for company_name in headers_only_company:
    headers_avg.append(company_name + " " + "avg")
    variables_to_drop.append(company_name + " " + "open")
    variables_to_drop.append(company_name + " " + "high")
    variables_to_drop.append(company_name + " " + "low")
    variables_to_drop.append(company_name + " " + "close")
    variables_to_drop.append(company_name + " " + "volume")

data_minute_prices = indexed_data_minute_NoNaN[headers_avg]

# this will be my dependent variable, my "Y"
# y = data_minute_prices.to_numpy()

# data-set with the factors data
# this is one mode, the firm specific one
# firm_mode = indexed_data_minute_NoNaN.drop(variables_to_drop,
axis=1).reset_index()

# now import the sentiment mode
# sentiment_mode = pd.read_excel('sentiment mode.xlsx').reset_index()

# creating the definitive dataset and importing the event mode
# df_1 = pd.concat([firm_mode, sentiment_mode], axis=1)
# news = []
# for i in range(100):
#     news.append('news_n_{}.csv'.format(i))
# all_news = pd.concat(map(pd.read_csv, news), axis=1)
# definitive_df = pd.concat([df_1, all_news], axis=1)
# definitive_df = definitive_df.drop(['index', 'time', 'neg', 'neu', 'pos',
'headline', 'date', 'ticker', 'Unnamed: 0'], axis=1)
# tensor_data = definitive_df.fillna(method='backfill', axis=0)
# tensor_data = tensor_data.set_index(['dates', 'minute'])
# tensor_data.to_csv('tensor_data.csv')

# there was a mistake in the specification of the columns so I had to
adjust it manually
# that is why there is another call to the read_excel method on a
tensor_data.xlsx file
# tensor_data_new = pd.read_excel('tensor_data.xlsx')
# tensor_data_new = tensor_data_new.fillna(method='backfill', axis=0)
# tensor_data_new = tensor_data_new.set_index(['dates', 'minute'])
# tensor_data_new.to_csv('tensor_data.csv')

tensor_data_new = pd.read_csv('tensor_data.csv')
tensor_data_new = tensor_data_new.set_index(["dates", "minute"])
y = data_minute_prices.to_numpy()

y.shape

# Global Dimensionality Reduction
# construct the matrix W to check the proximity of two tensors in the
sequence
# It is an upper triangular matrix, where if two tensors are close the
entry is a 1,

```

```

# otherwise the entry is 0
W = tensor_distance(y)
W = np.nan_to_num(W)
W = np.tril(W)

# compute the matrix D, which is a diagonal matrix made up by
# the sums over the columns of W
sums_over_cols = []
for column in W:
    sums_over_cols.append(column.sum())
D = np.diag(sums_over_cols)

# Convert the data to a numpy ndarray
X = tensor_data_new.drop(headers_avg, axis=1)
X = X.to_numpy()
X_reshaped = np.reshape(X, (1950, 100, 3, 3))
X_reshaped_NoNaN = np.nan_to_num(X_reshaped)
tensors = tl.tensor(X_reshaped_NoNaN)

# lists to store the quantities in the algorithm
core_storage = []
factors_1_storage = []
factors_2_storage = []
factors_3_storage = []
D_u1_storage = []
D_u2_storage = []
D_u3_storage = []
W_u1_storage = []
W_u2_storage = []
W_u3_storage = []

for i in range(1949):
    # Tucker decomposition for the stream of input tensors, the rank has
    # been
    # chosen arbitrarily
    core, factors = tucker(tensors[i], ranks=[100, 100, 100])
    # appending the core tensor
    core_storage.append(core)
    # appending the first three factors
    factors_1_storage.append(factors[0])
    factors_2_storage.append(factors[1])
    factors_3_storage.append(factors[2])
    # appending the D matrix for every first factor of every tensor in the
    # stream
    # the same for the other two factors
    D_u1_storage.append(D[i][i]*(np.dot(factors_1_storage[i],
factors_1_storage[i].T)))
    D_u2_storage.append(D[i][i]*(np.dot(factors_2_storage[i],
factors_2_storage[i].T)))
    D_u3_storage.append(sums_over_cols[i]*(np.dot(factors_3_storage[i],
factors_3_storage[i].T)))
    # appending the W matrix for every first factor of every tensor in the
    # stream
    # the same is done for the other factors
    W_u1_storage.append(W[i][i]*(np.dot(factors_1_storage[i],
factors_1_storage[i].T)))
    W_u2_storage.append(W[i][i]*(np.dot(factors_2_storage[i],
factors_2_storage[i].T)))
    W_u3_storage.append(W[i][i]*(np.dot(factors_3_storage[i],
factors_3_storage[i].T)))

# computation of the actual D and W matrixes for each factor
D_1 = sum(D_u1_storage)

```

```

D_2 = sum(D_u2_storage)
D_3 = sum(D_u3_storage)
W_1 = sum(W_u1_storage)
W_2 = sum(W_u2_storage)
W_3 = sum(W_u3_storage)
# now the code to obtain V_k
# the value for gamma was chosen arbitrarily
gamma = 1.0001
A_1 = np.subtract((1-gamma)*D_1, W_1)
A_2 = np.subtract((1-gamma)*D_2, W_2)
A_3 = np.subtract((1-gamma)*D_3, W_3)
# b_1 = np.zeros(len(A_1))
# b_2 = np.zeros(len(A_2))
# func_1 = lambda x: np.linalg.norm(np.dot(-A_1,x))
# func_2 = lambda x: np.linalg.norm(np.dot(-A_2,x)-b_2)
# func_3 = lambda x: np.linalg.norm(np.dot(A_3,x)-b)
V_1 = scipy.linalg.null_space(A_1, rcond=1)
V_2 = scipy.linalg.null_space(A_2, rcond=1)
V_3 = scipy.linalg.null_space(A_3, rcond=1)

tensors_reconstruct = []
factors_projected = []
for i in range(1949):
    # build-up the Tensor stream back but by multiplying the matrix V_k for
    all
    # the modes U_k in all the tensors
    tmp = []
    tmp.append(np.matmul(V_1.transpose(), factors_1_storage[i]))
    tmp.append(np.matmul(V_2.transpose(), factors_2_storage[i]))
    tmp.append(np.matmul(V_3.transpose(), factors_3_storage[i]))
    factors_projected.append(tmp)
    tensors_reconstruct.append(tl.tucker_to_tensor((core_storage[i],
factors_projected[i])))

# This ends the implementation of the first algorithm for global
dimensionality reduction
# Now I need to implement the regression part

tensors_reduced = np.array(tensors_reconstruct)

tensors_reduced.shape

# Now tensor-based regression learning
# The first thing to do is to set the weights for the factors
# It said that they had to be random unit vectors, so I computed them
# random_1 = np.random.randn(100)
# random_2 = np.random.randn(3)
# random_3 = np.random.randn(3)
# random_1_norm = np.linalg.norm(random_1)
# random_2_norm = np.linalg.norm(random_2)
# random_3_norm = np.linalg.norm(random_3)
# weight_hat_1 = random_1/random_1_norm
# weight_hat_2 = random_2/random_2_norm
# weight_hat_3 = random_3/random_3_norm
# According to the paper, the betas are equal to the norm squared of the
weight_hat vectors
# This cannot be because the weight_hat vectors are unit vectors
# Their norm is therefore equal to one, so I assume it was a mistake or
maybe something wrong
# that I did along the way. For the rest of the code, I use the random unit
vectors.

# betas = [np.linalg.norm(weight_hat_1)**2,

```



```

#         np.linalg.norm(weight_hat_2)**2,
#         np.linalg.norm(weight_hat_3)**2]

# betas = [weight_hat_1**2, weight_hat_2**2, weight_hat_3**2]

# ORIGINAL, DO NOT MODIFY
# Now tensor-based regression learning
# The first thing to do is to set the weights for the factors
# It said that they had to be random unit vectors, so I computed them
# According to the paper, the betas are equal to the norm squared of the
weight_hat vectors
# This cannot be because the weight_hat vectors are unit vectors
# Their norm is therefore equal to one, so I assume it was a mistake or
maybe something wrong
# that I did along the way. For the rest of the code, I use the random unit
vectors.
y = y[0:1949, :]
# Here I start with the algorithm
tensor_regressors = []
for j in range(1949):
    random_1 = np.random.randn(100)
    random_2 = np.random.randn(3)
    random_3 = np.random.randn(3)
    random_1_norm = np.linalg.norm(random_1)
    random_2_norm = np.linalg.norm(random_2)
    random_3_norm = np.linalg.norm(random_3)
    w_hat_1 = random_1/random_1_norm
    w_hat_2 = random_2/random_2_norm
    w_hat_3 = random_3/random_3_norm
    betas = [w_hat_1**2, w_hat_2**2, w_hat_3**2]
    regressors = tl.tenalg.multi_mode_dot(tensors_reduced[j], betas)
    tensor_regressors.append(regressors)
tensor_regressors = np.array(tensor_regressors)
# regressors_all_companies.append(tensor_regressors)

tensor_regressors = tensor_regressors.reshape((1949, 1))
tensor_regressors.shape

weights = []
biases = []
predictions = []
actual_values = []
for i in range(100):
    X = tensor_regressors
    y_company = y[:, i]
    constant = 5
    #I need the Support Vector Machines to continue, I need the sklearn.svm
package
    # support_regression = make_pipeline(StandardScaler(), SVR(C=constant,
kernel='rbf', epsilon=0.1))
    # scaler = StandardScaler()
    support_regression = SVR(C=constant, kernel='rbf', epsilon=2)
    cross_validate = cross_val_predict(support_regression, X, y_company,
cv=10)
    # model = sklearn.multioutput.MultiOutputRegressor(support_regression)
    # X_train_scaled = scaler.fit_transform(X_train)
    # X_test_scaled = scaler.fit_transform(X_test)
    # y_train_scaled = scaler.fit_transform(y_train.reshape((1561, 1)))
    # y_test_scaled = scaler.fit_transform(y_test.reshape((388,1)))
    support_regression.fit(X=X, y=y_company)
    coef_dec_func = support_regression.dual_coef_
    bias = support_regression.intercept_
    actual_values.append(y_company)

```

```

weights.append(coef_dec_func)
biases.append(bias)
predictions.append(cross_validate)

print('company n.{}: '.format(i), headers_only_company[i])
MSE = mean_squared_error(y_true=y_company.reshape((-1, 1)),
y_pred=cross_validate)
RMSE = math.sqrt(MSE)
MAE = mean_absolute_error(y_company.reshape((-1, 1)),
y_pred=cross_validate)
MAPE = mape(y_company.reshape((-1, 1)), cross_validate)
dir_acc = directional_accuracy(y_company.reshape((-1, 1)),
cross_validate)
r_2 = r2_score(y_company.reshape((-1, 1)), cross_validate)
# print('MSE: ', MSE)
print('RMSE: ', RMSE)
print('MAE: ', MAE)
print('MAPE: ', MAPE)
print('DA: ', dir_acc)
print('R2: ', r_2)

predictions = pd.DataFrame(data=predictions)
predictions_transposed = predictions.transpose()
predictions_transposed["periods"] = data_minute["dates"].astype(str) + ' '
+ data_minute["minute"].astype(str)
predictions_transposed["periods"] =
pd.to_datetime(predictions_transposed["periods"],
infer_datetime_format=True)
indexed_predictions =
predictions_transposed.set_index(predictions_transposed["periods"])
indexed_predictions = indexed_predictions.drop(["periods"], axis=1)
indexed_predictions.columns = headers_only_company

for company_name in headers_only_company:

indexed_predictions[company_name].to_csv('{}_trading.csv'.format(company_name))

import matplotlib.pyplot as plt

plt.plot(actual_values[0], label='actual price')
plt.plot(indexed_predictions["MICROSOFT"].values, label='predicted price')
plt.legend()

# Trading strategy, MEAN REVERSION
import backtrader as bt
from datetime import datetime as dt
class CrossSectionalMR(bt.Strategy):
    def prenext(self):
        self.next()

    def next(self):
        # only look at data that existed yesterday
        available = list(filter(lambda d: d, self.datas))

        rets = np.zeros(len(available))
        for i, d in enumerate(available):
            # calculate individual daily returns
            rets[i] = (d.close[0]- d.close[-1]) / d.close[-1]

        # calculate weights using formula
        market_ret = np.mean(rets)

```

```

weights = -(rets - market_ret)
weights = weights / np.sum(np.abs(weights))

for i, d in enumerate(available):
    self.order_target_percent(d, target=weights[i])

cerebro = bt.Cerebro(stdstats=False)
cerebro.broker.set_coc(True)

for company_name in headers_only_company:
    data = bt.feeds.GenericCSVData(
        dataname=f'{company_name}_trading.csv',
        dtformat=('%Y-%m-%d %H:%M:%S'),
        datetime=0,
        close=1,
        time=-1,
        open=-1,
        high=-1,
        low=-1,
        volume=-1,
        openinterest=-1,
        nullvalue=0.0,
        plot=False
    )
    cerebro.adddata(data)

cerebro.broker.setcash(1_000)
cerebro.addobserver(bt.observers.Value)
cerebro.addanalyzer(bt.analyzers.SharpeRatio, riskfreerate=0.0)
cerebro.addanalyzer(bt.analyzers>Returns)
cerebro.addanalyzer(bt.analyzers.DrawDown)
cerebro.addstrategy(CrossSectionalMR)
results = cerebro.run()

print(f"Norm. Return:
{results[0].analyzers.returns.get_analysis()['rnorm100']:.2f}%")
print(f"Max Drawdown:
{results[0].analyzers.drawdown.get_analysis()['max']['drawdown']:.2f}%")
cerebro.plot()[0][0]

for i in range(100):
    tmp = pd.DataFrame(actual_values[i],
                       index=predictions_transposed["periods"])
    tmp.to_csv('{}_trading_original.csv'.format(i))

import backtrader as bt
from datetime import datetime as dt
class CrossSectionalMR(bt.Strategy):
    def prenext(self):
        self.next()

    def next(self):
        # only look at data that existed yesterday
        available = list(filter(lambda d: d, self.datas))

        rets = np.zeros(len(available))
        for i, d in enumerate(available):
            # calculate individual daily returns
            rets[i] = (d.close[0]- d.close[-1]) / d.close[-1]

        # calculate weights using formula
        market_ret = np.mean(rets)

```

```

weights = -(rets - market_ret)
weights = weights / np.sum(np.abs(weights))

for i, d in enumerate(available):
    self.order_target_percent(d, target=weights[i])

cerebro = bt.Cerebro(stdstats=False)
cerebro.broker.set_coc(True)

for i in range(100):
    data = bt.feeds.GenericCSVData(
        dataname=f'{i}_trading_original.csv',
        dtformat=( '%Y-%m-%d %H:%M:%S' ),
        datetime=0,
        close=1,
        time=-1,
        open=-1,
        high=-1,
        low=-1,
        volume=-1,
        openinterest=-1,
        nullvalue=0.0,
        plot=False
    )
    cerebro.adddata(data)

cerebro.broker.setcash(1_000)
cerebro.addobserver(bt.observers.Value)
cerebro.addanalyzer(bt.analyzers.SharpeRatio, riskfreerate=0.0)
cerebro.addanalyzer(bt.analyzers>Returns)
cerebro.addanalyzer(bt.analyzers.DrawDown)
cerebro.addstrategy(CrossSectionalMR)
results = cerebro.run()

print(f"Norm. Return:
{results[0].analyzers.returns.get_analysis()['rnorm100']:.2f}%")
print(f"Max Drawdown:
{results[0].analyzers.drawdown.get_analysis()['max']['drawdown']:.2f}%")
cerebro.plot()[0][0]

```

### References:

1. Boyer John M., Wendy J. Myrvold. On the Cutting Edge: Simplified  $O(n)$  Planarity by Edge Addition. *Journal of Graph Algorithms and Applications* <http://jgaa.info/> vol. 8, no. 3, pp. 241–273 (2004).
2. Buraschi Andrea and Paolo Porchia. 2012. Dynamic Networks and Asset Pricing. AFA 2013 San Diego Meetings Paper.
3. Chung Fan R. K.. 1997. *Spectral Graph Theory*. Vol. 92. American Mathematical Society.
4. De Almeida A., G. Favier, and J. ao Mota, The Constrained Block-PARAFAC Decomposition. Presentation at TRICAP2006, Chania, Greece. Available online at [http://www.telecom.tuc.gr/~nikos/TRICAP2006main/TRICAP2006\\_Almeida.pdf](http://www.telecom.tuc.gr/~nikos/TRICAP2006main/TRICAP2006_Almeida.pdf), June 2006.
5. Fama Eugene F. and Kenneth R. French. 1993. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics* 33, 1 (1993), 3–56.
6. Fazeli Arvand Using Deep Learning for Predicting Stock Trends 2019
7. Feng Justin C.. The Poor Man’s Introduction to Tensors. Freely available at [https://web2.ph.utexas.edu/~jcfeng/notes/Tensors\\_Poor\\_Man.pdf](https://web2.ph.utexas.edu/~jcfeng/notes/Tensors_Poor_Man.pdf).
8. Foster Dean P., Mark Liberman, Robert A. Stine. 2013. Featurizing Text: Converting Text into Predictors for Regression Analysis. CUNY Data Mining Initiative.
9. Frank Murray Z. and Werner Antweiler. 2004. Is all that talk just noise? The information content of internet stock message boards. *Journal of Finance* 59, 3 (2004), 1259–1294.
10. Hagberg Aric A. Daniel A. Schult and Pieter J. Swart. “Exploring network structure, dynamics, and function using NetworkX”. in *Proceedings of the 7th Python in Science*

- Conference (SciPy2008). Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008.
11. Hassan Jain, Mohibul, Nimisha Goyal, Tanmay Goel 2019 A System for Comparative Analysis of Different Stock Prediction Methodologies Ishan
  12. Hutto C.J.. Eric Gilbert VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text.. Copyright © 2014, Association for the Advancement of Artificial Intelligence ([www.aaai.org](http://www.aaai.org)).
  13. Huy Phan Anh and Andrzej Cichocki. Tensor decompositions for feature extraction and classification of high dimensional datasets. *Nonlinear Theory and Its Applications IEICE* · January 2010.
  14. J. JaJ' a', Optimal evaluation of pairs of bilinear forms, *SIAM J. Comput.*, 8 (1979), pp. 443–462
  15. Kisil Ilia, Giuseppe G. Calvi, and Danilo P. Mandic. Tensor Valued Common and Individual Feature Extraction: Multi-dimensional Perspective. (2017).
  16. Kolda Tamara G., Brett W. Bader. *Tensor Decomposition and Application* (2009): Society for Industrial and Applied Mathematics
  17. Kossaifi Jean. Yannis Panagakis. Anima Anandkumar. Maja Pantic. TensorLy: Tensor Learning in Python. *Journal of Machine Learning Research* 20 (2019) 1-6
  18. Kruskal J. B., Rank, decomposition, and uniqueness for 3-way and N-way arrays, in *Multiway Data Analysis*, R. Coppi and S. Bolasco, eds., North-Holland, Amsterdam, 1989, pp. 7–18.
  19. Kruskal J. B., Statement of Some Current Results about Three-Way Arrays, manuscript, AT&T Bell Laboratories, Murray Hill, NJ. Available at <http://three-mode.leidenuniv.nl/pdf/k/kruskal1983.pdf>, 1983
  - Lexin Li. Hongtu Zhu. *J Am Stat Assoc.* 2013.
  20. Li Qing, Yuanzhu Chen, Li Ling Jiang, Ping Li, and Hsinchun Chen. 2016. A tensor-based information framework for predicting the stock market. *ACM Trans. Inf. Syst.* 34, 2, Article 11 (February 2016), 30 pages.
  21. Li Qing. TieJun Wang. Ping Li. Ling Liu. Qixu Gong. Yuanzhu Chen. The effect of news and public mood on stock movements. *Information Sciences* 278 (2014) 826–840
  22. Nguyen Lam. Examine different neural networks' efficiency in predicting stock prices.
  23. Oku Kenta, Shinsuke Nakajima, Jun Miyazaki, and Shunsuke Uemura. 2006. Context-aware SVM for context-dependent information recommendation. In *Proceedings of the 7th International Conference on Mobile Data Management*. IEEE, 109

24. Panzica, Roberto Calogero (2018): Idiosyncratic volatility puzzle: The role of assets' interconnections, SAFE Working Paper, No. 228, Goethe University Frankfurt, SAFE Sustainable Architecture for Finance in Europe, Frankfurt a. M,
25. Pao Hsiao-Tien, "A comparison of neural network and multiple regression analysis in modeling capital structure", *Expert Systems with Applications* 35, 2008
26. Recent Advances in Big Data and Deep Learning: Proceedings of the INNS Big Data and Deep Learning Conference INNSBDDL 2019, held at Sestri Levante, Genova, Neural Networks Society Book.
27. Rosin Paul L.. Techniques for Assessing Polygonal Approximations of Curves. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, VOL. 19, NO. 6, JUNE 1997.
28. Schumaker Robert P. and Hsinchun Chen. 2009a. A quantitative stock prediction system based on financial news. *Information Processing & Management* 45, 5 (2009), 571–583.
29. Schumaker Robert P. and Hsinchun Chen. 2009b. Textual analysis of stock market prediction using breaking financial news: The AZF in text system. *ACM Transactions on Information Systems* 27, 2 (2009), 12:1– 12:19.
30. Schumaker Robert P. and Hsinchun Chen. 2009b. Textual analysis of stock market prediction using breaking financial news: The AZFin text system. *ACM Transactions on Information Systems* 27, 2 (2009), 12:1– 12:19.
31. Schumaker Robert P., Yulei Zhang, Chun-Neng Huang, and Hsinchun Chen. 2012. Evaluating sentiment in financial news articles. *Decision Support Systems* 53, 3 (2012), 458–464.
32. Shleifer Andrei and Robert W. Vishny. 1997. The limits of arbitrage. *Journal of Finance* 52, 1 (1997), 35–55.
33. Sinha Malo, P., , A., Korhonen, P., Wallenius, J., & Takala, P. (2014). Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65(4), 782-796.
34. Ten Berge J. M. F., Kruskal's polynomial for  $2 \times 2 \times 2$  arrays and a generalization to  $2 \times n \times n$  arrays, *Psychometrika*, 56 (1991), pp. 631–636.
35. Ten Berge J. M. F., Partial uniqueness in CANDECOMP/PARAFAC, *J. Chemometrics*, 18 (2004), pp. 12–16.
36. Ten Berge J. M. F., Partial uniqueness in CANDECOMP/PARAFAC, *J. Chemometrics*, 18 (2004), pp. 12–16.

37. Ten Berge J. M. F., The typical rank of tall three-way arrays, *Psychometrika*, 65 (2000), pp. 525–532.
38. Tetlock Paul C., Maytal Saar-Tsechansky, and Sofus Macskassy. 2008. More than words: Quantifying language to measure firms' fundamentals. *Journal of Finance* 63, 3 (2008), 1437–1467.
39. Tumminello Michele, Fabrizio Lillo, Rosario N. Mantegna. 2010. Correlation, hierarchies and networks in financial markets. *Journal of Economic Behavior and Organization*.
40. W. Lo Andrew and Archie Craig MacKinlay. 1988. Stock market prices do not follow random walks: Evidence from a simple specification test. *Review of Financial Studies* 1, 1 (1988), 41–66.
41. Zareei Abalfazl, “Network Origin of Portfolio Risk”, *Journal of Banking and Finance*, 2019
42. Zhou Hua. *Tensor Regression with Applications in Neuroimaging Data Analysis*.



