Machine Learning for Financial Products Recommendation

Baptiste Barreau

université PARIS-SACLAY

Machine Learning for Financial Products Recommendation

Apprentissage Statistique pour la Recommandation de Produits Financiers

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°573
Interfaces : Approches interdisciplinaires, fondements, applications et innovation
Spécialité de doctorat : Mathématiques appliquées
Unité de recherche : Université Paris-Saclay, CentraleSupélec, Mathématiques et Informatique pour la Complexité et les Systèmes, 91190, Gif-sur-Yvette, France.
Référent : CentraleSupélec

Thèse présentée et soutenue à Gif-sur-Yvette, le 15 septembre 2020, par

Baptiste BARREAU

Composition du Jury

Michael BENZAQUEN Professeur, École Polytechnique	Président
Charles-Albert LEHALLE Head of Data analytics, Capital Fund Management, HDR	Rapporteur & Examinateur
Elsa NEGRE Maître de conférences, Université Paris-Dauphine, HDR	Rapporteur & Examinatrice
Eduardo ABI JABER Maître de conférences, Université Paris 1	Examinateur
Sylvain ARLOT Professeur, Université Paris-Sud	Examinateur
Damien CHALLET Professeur, CentraleSupélec	Directeur de thèse
Sarah LEMLER Maître de conférences, CentraleSupélec	Co-Directrice de thèse
Frédéric ABERGEL Professeur, BNP Paribas Asset Management	Invité
Laurent CARLIER Head of Data & AI Lab, BNPP CIB Global Markets	Invité

Thèse de doctorat

NNT : 2020UPAST010

Résumé

L'anticipation des besoins des clients est cruciale pour toute entreprise — c'est particulièrement vrai des banques d'investissement telles que BNP Paribas Corporate and Institutional Banking au vu de leur rôle dans les marchés financiers. Cette thèse s'intéresse au problème de la prédiction des intérêts futurs des clients sur les marchés financiers, et met plus particulièrement l'accent sur le développement d'algorithmes *ad hoc* conçus pour résoudre des problématiques spécifiques au monde financier.

Ce manuscrit se compose de cinq chapitres, répartis comme suit:

- Le chapitre 1 expose le problème de la prédiction des intérêts futurs des clients sur les marchés financiers. Le but de ce chapitre est de fournir aux lecteurs toutes les clés nécessaires à la bonne compréhension du reste de cette thèse. Ces clés sont divisées en trois parties: une mise en lumière des jeux de données à notre disposition pour la résolution du problème de prédiction des intérêts futurs et de leurs caractéristiques, une vue d'ensemble, non exhaustive, des algorithmes pouvant être utilisés pour la résolution de ce problème, et la mise au point de métriques permettant d'évaluer la performance de ces algorithmes sur nos jeux de données. Ce chapitre se clôt sur les défis que l'on peut rencontrer lors de la conception d'algorithmes permettant de résoudre le problème de la prédiction des intérêts futurs en finance, défis qui seront, en partie, résolus dans les chapitres suivants;

- Le chapitre 2 compare une partie des algorithmes introduits dans le chapitre 1 sur un jeu de données provenant de BNP Paribas CIB, et met en avant les difficultés rencontrées pour la comparaison d'algorithmes de nature différente sur un même jeu de données, ainsi que quelques pistes permettant de surmonter ces difficultés. Ce comparatif met en pratique des algorithmes de recommandation classiques uniquement envisagés d'un point de vue théorique au chapitre précédent, et permet d'acquérir une compréhension plus fine des différentes métriques introduites au chapitre 1 au travers de l'analyse des résultats de ces algorithmes;

- Le chapitre 3 introduit un nouvel algorithme, *Experts Network*, i.e., réseau d'experts, conçu pour résoudre le problème de l'hétérogénéité de comportement des investisseurs d'un marché donné au travers d'une architecture de réseau de neurones originale, inspirée de la recherche sur les mélanges d'experts. Dans ce chapitre, cette nouvelle méthodologie est utilisée sur trois jeux de données distincts: un jeu de données synthétique, un jeu de données en libre accès, et un jeu de données provenant de BNP Paribas CIB. Ce chapitre présente aussi en plus grand détail la genèse de l'algorithme et fournit des pistes pour l'améliorer;

- Le chapitre 4 introduit lui aussi un nouvel algorithme, appelé *History-augmented collaborative filtering*, i.e., filtrage collaboratif augmenté par historiques, qui proposes d'augmenter les approches de factorisation matricielle classiques à l'aide des historiques d'interaction

des clients et produits considérés. Ce chapitre poursuit l'étude du jeu de données étudié au chapitre 2 et étend l'algorithme introduit avec de nombreuses idées. Plus précisément, ce chapitre adapte l'algorithme de façon à permettre de résoudre le problème du *cold start*, i.e., l'incapacité d'un système de recommandation à fournir des prédictions pour de nouveaux utilisateurs, ainsi qu'un nouveau cas d'application sur lequel cette adaptation est essayée;

- Le chapitre 5 met en lumière une collection d'idées et d'algorithmes, fructueux ou non, qui ont été essayés au cours de cette thèse. Ce chapitre se clôt sur un nouvel algorithme mariant les idées des algorithmes introduits aux chapitres 3 et 4.

La recherche présentée dans cette thèse a été conduite via le programme de thèses CIFRE, en collaboration entre le laboratoire MICS de CentraleSupélec et la division Global Markets de BNP Paribas CIB. Elle a été conduite dans le cadre du mandat de l'équipe Data & AI Lab, une équipe de science des données faisant partie de la recherche quantitative et dévouée à l'application des méthodes d'apprentissage statistique aux problématiques des différentes équipes de Global Markets.

Mots-clés. Apprentissage statistique, finance, systèmes de recommandation, clustering supervisé, systèmes de recommandation dépendants du contexte, systèmes de recommandation dépendants du temps, apprentissage profond, réseaux de neurones.

Abstract

Anticipating clients' needs is crucial to any business — this is particularly true for corporate and institutional banks such as BNP Paribas Corporate and Institutional Banking due to their role in the financial markets. This thesis addresses the problem of future interests prediction in the financial context and focuses on the development of *ad hoc* algorithms designed for solving specific financial challenges.

This manuscript is composed of five chapters:

- Chapter 1 introduces the problem of future interests prediction in the financial world. The goal of this chapter is to provide the reader with all the keys necessary to understand the remainder of this thesis. These keys are divided into three parts: a presentation of the datasets we have at our disposal to solve the future interests prediction problem and their characteristics, an overview of the candidate algorithms to solve this problem, and the development of metrics to monitor the performance of these algorithms on our datasets. This chapter finishes with some of the challenges that we face when designing algorithms to solve the future interests problem in finance, challenges that will be partly addressed in the following chapters;

- Chapter 2 proposes a benchmark of some of the algorithms introduced in Chapter 1 on a real-word dataset from BNP Paribas CIB, along with a development on the difficulties encountered for comparing different algorithmic approaches on a same dataset and on ways to tackle them. This benchmark puts in practice classic recommendation algorithms that were considered on a theoretical point of view in the preceding chapter, and provides further intuition on the analysis of the metrics introduced in Chapter 1;

- Chapter 3 introduces a new algorithm, called *Experts Network*, that is designed to solve the problem of behavioral heterogeneity of investors on a given financial market using a custom-built neural network architecture inspired from mixture-of-experts research. In this chapter, the introduced methodology is experimented on three datasets: a synthetic dataset, an open-source one and a real-world dataset from BNP Paribas CIB. The chapter provides further insights into the development of the methodology and ways to extend it;

- Chapter 4 also introduces a new algorithm, called *History-augmented Collaborative Filtering*, that proposes to augment classic matrix factorization approaches with the information of users and items' interaction histories. This chapter provides further experiments on the dataset used in Chapter 2, and extends the presented methodology with various ideas. Notably, this chapter exposes an adaptation of the methodology to solve the cold-start problem and applies it to a new dataset;

- Chapter 5 brings to light a collection of ideas and algorithms, successful or not, that were experimented during the development of this thesis. This chapter finishes on a new

algorithm that blends the methodologies introduced in Chapters 3 and 4.

The research presented in this thesis has been conducted under the French CIFRE Ph.D. program, in collaboration between the MICS Laboratory at CentraleSupélec and the Global Markets division of BNP Paribas CIB. It has been conducted as part of the Data & AI Lab mandate, a data science team within Quantitative Research devoted to the applications of machine learning for the support of Global Markets' business lines.

Keywords. Machine learning, finance, recommender systems, supervised clustering, context-aware recommender systems, time-aware recommender systems, deep learning, neural networks.

Contents

Acknowledgements

I would like to express my gratitude to Frédéric Abergel and Laurent Carlier for providing me with the opportunity to conduct this Ph.D. thesis within BNP Paribas Corporate and Institutional Banking. Thanks to Frédéric Abergel for accepting to be my academic Ph.D. supervisor, and for his initial and precious guidance on my research. Many thanks to Damien Challet, who took over the supervision of my thesis for the last two years and who, thanks to his academic support and his deep and thoughtful questions, fueled me with the inspiration I required to pursue this work. Thanks also to Sarah Lemler for following my work, even remotely, and for her help when I needed it.

I would also like to thank the fantastic Global Markets Data & AI Lab team as a whole and, more particularly, its Parisian division for their welcome and all the good times we spent together during these past three years. Many thanks to Laurent Carlier for being my industrial Ph.D. supervisor and our countless and fruitful discussions on our research ideas. Thanks to Julien Dinh for following my work and examining it with his knowledgeable scrutiny. Special thanks to William Benhaim, François-Hubert Dupuy, Amine Amiroune, the more recent Jeremi Assael and Alexandre Philbert, and our two graduate program rotations, Victor Geoffroy and Jean-Charles Nigretto, for our fruitful discussions and our pub crawls that, without doubt, fueled the creativity required for this work. Many thanks also to the interns Lola Moisset, Dan Sfedj, Camille Garcin, and Yassine Lahna, with whom I had the chance to work on topics related to the research presented in this manuscript and deepen my knowledge of the presented subjects.

I am sincerely grateful to my family as a whole and more particularly to my parents, Annick and Bruno Barreau, for their love and support during my entire academic journey, which ends with this work. My deepest thanks go to the one who shares my life, Camille Raffin, for her unconditional support and her heartwarming love, in both good and bad times.

Chapter 1

Recommendations in the financial world

Mathematics have made their appearance in corporate and institutional banks in the 90s with the development of quantitative research teams. Quantitative researchers, a.k.a. *quants*, are devoted to applying all tools of mathematics to help the traders and salespeople, a.k.a. *sales*, of the bank. Quants have historically focused on developing tools and innovations for the use of traders — the recent advances of machine learning, however, have allowed them to provide new kinds of solutions for the previously overlooked salespeople. Notably, deep learning methodologies allow assisting sales with their daily tasks in many ways, from automating repetitive tasks by leveraging natural language processing tools to providing decision-support algorithms such as *recommender systems*. This thesis provides an in-depth exploration of recommender systems, their application to the context of a corporate and investment bank, and how we can improve models from literature to better suit the characteristics of the financial world.

This chapter introduces the context and research question examined in this thesis (Section 1.1), elaborates on and explores the data we use (Section 1.2), gives an overview of recommender systems (Section 1.3), sets the evaluation strategy that we use in all further experiments (Section 1.4) and provides insights into the challenges we face in a financial context (Section 1.5).

1.1 Predicting future interests

This section introduces the financial context, defines the recommendation problem that we want to solve, and provides first insights into the datasets we study.

1.1.1 Enhancing sales with recommender systems

BNP Paribas CIB is a *market maker*. The second Markets in Financial Instruments Directive, MiFID II, a directive of the European Commission for the regulation of financial markets, defines a market maker as "a person who holds himself out on the financial markets on a continuous basis as being willing to deal on own account by buying and selling financial instruments against that person's proprietary capital at prices defined by that person," see (The European Commission, 2014, Article 4-7).

Market makers play the role of liquidity providers in the financial markets by quoting both buy and sell prices for many different financial assets. From the perspective of a market maker, a.k.a. the *sell side*, we call *bid* the buy price and *ask* the sell price, with $p_{bid} < p_{ask}$. Consequently, on the *buy side*, a client has to pay the ask price to buy a given product. The business of a market

maker is driven by the *bid-ask spread*, i.e., the difference between bid and ask prices — the role of a market maker is therefore not to invest in the financial markets but to offer investors the opportunity to do so. The perfect situation happens when the market maker directly finds buyers and sellers for the same financial product at the same time. However, this ideal case is not common, and the bank has to suffer *risk* from holding products before they are bought by an investor or from stocking products to cope with demand. These assets, called *axes*, need to be carefully managed to minimize the risk to which the bank is exposed. This management is part of the role of traders and sales, and has notably been studied in Guéant et al. (2013).

Sales teams are the interface between the bank and its clients. When a client wants to buy or sell a financial asset, she may call a sales from a given bank, or directly interact with market makers on an electronic platform such as the Bloomberg Dealer-to-Client (D2C) one (Bloomberg, 2020). It can happen that a client does not want to buy/sell a given product at the moment of the sales' call, but shows interest in it. Sales can then register an *indication of interest* (IOI) in BNPP CIB systems, a piece of valuable information for further commercial discussions, and for the prediction of future interests. On D2C platforms, clients can request prices from market makers in two distinct processes called *request for quotation* (RFQ) and *request for market* (RFM). RFQs are directed, meaning that these requests are specifically made for either a buy or sell direction, whereas RFMs are not. RFMs can be seen as more advantageous to the client, as she provides less information about her intent in this process. However, for less liquid assets such as bonds, the common practice is to perform RFQs. Market makers continuously stream bid and ask prices for many assets on D2C platforms. When a client is interested in a given product, she can select n providers and send them an RFQ — providers then respond with their final price, and the client chooses to buy or sell the requested product with either one of these n providers, or do nothing. Only market makers among these n receive information about the client's final decision. A typical value of n is 6, such as for the Bloomberg D2C platform. Fermanian et al. (2016) provide an extensive study along with a modeling of the RFQ process. Responding to calls and requests is the core of a sales' work. But sales can also directly contact clients and suggest relevant trade ideas to them, e.g., assets on which the bank is axed and for which it might offer a better price than its competitors. This proactive behavior is particularly important for the bank as it helps manage financial inventories and better serve the bank's clients when their needs are correctly anticipated.

Correctly anticipating clients' needs and interests, materialized by the requests they perform, is therefore of particular interest to the bank. Recommender systems, defined as "an information-filtering technique used to present the items of information (video, music, books, images, Web-sites, etc.) that may be of interest to the user" (Negre, 2015), could help us better anticipate these requests and support the proactive behavior of sales by allowing them to navigate the complexity of markets more easily. Consequently, the research problem that we try to solve in this thesis is the following:

At a given date, which investor is interested in buying and/or selling a given financial asset?

This thesis aims at designing algorithms that try to solve this open question while being well-suited to the specificities of the financial world, further examined in Section 1.5.

1.1.2 An overview of bonds and options

The mandate of the Data & AI Lab covers multiple asset classes, from bonds to options on equity or foreign exchange market, as known as FX. In-depth knowledge of these financial products is not required to understand the algorithms that are developed in this thesis. However, it is essential to know what these products and their characteristics are to design relevant methodologies. This section consequently provides an overview of the financial assets that will be covered in this thesis. It does not require prior knowledge of mathematical finance, and only aims at providing insights into the main characteristics and properties of these assets. An in-depth study of financial assets can be found in Hull (2014).

Bonds

It is possible to leverage financial markets to issue debt in the form of *bonds*. Fabozzi (2012, Chapter 1) defines a bond as "a debt instrument requiring the issuer to repay to the lender the amount borrowed plus interest over a specified period of time." There are three main types of bonds, defined by their issuer — municipal bonds, government bonds, and corporate bonds. The definitions and examples provided in this section are freely inspired by Hull (2014, Chapter 4).

Bonds, as other financial assets, are issued on *primary markets*, where investors can buy them for a limited period of time. In the context of equities, *initial public offerings* (IPO) are an example of a primary market. The *issuance date* of a bond corresponds to its entry on the *secondary market*. On the secondary market, investors can freely trade bonds before their *maturity date*, .i.e., their expiry date. Consequently, bonds have a market price determined by bid and ask, and usual features related to its evolution can be computed.

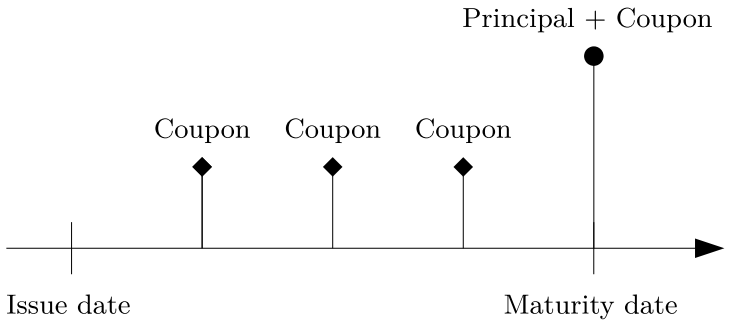


Figure 1.1: Cash flows of a bond.

The interest rate of a bond is usually defined annually. Interests are either paid periodically as *coupons* or in full at maturity, in which case the bond is called a *zero-coupon*. The cash flows of a bond are summed up in Fig. 1.1. Classically, coupons are paid on a semiannual basis. Bonds are expressed in percent, i.e., the value of a bond at issuance is considered to be 100. For example, the value of a semiannual coupon bond with an interest rate of 10% a year after issuance is $100 * 1.05 * 1.05 = 110.25$. If we consider a compounding of m times per annum, the terminal value of an investment of 100 at rate R after n years is given by $100 * (1 + R/m)^{mn}$. Considering the $m \to \infty$ limit, it can be shown that the terminal value converges to $100 * e^{Rn}$. This is known as continuous compounding and is used for further computations.

The theoretical price of a bond is defined as the present value of the future cash flows received by the owner of the bond. Future cash flows are discounted using the Treasury zero-coupon rates maturing at the time of these cash flows, as Treasury bonds are considered risk-free. For instance, in the case of a 2-year bond with a 6% rate delivering semiannual coupons, the

theoretical price of the bond p_t, expressed in basis points, is defined as $p_t = 3e^{-T_{0.5}*0.5} + 3e^{-T_{1.}*1.} + 3e^{-T_{1.5}*1.5} + 103e^{-T_{2.}*2.}$, where T_n is the n-year Treasury zero-coupon rate. The *yield* y of a bond is defined as the single discount rate that equals the theoretical bond price, i.e., in basis points, $p_t = 3e^{-y*0.5} + 3e^{-y*1.} + 3e^{-y*1.5} + 103e^{-y*2.}$. The yield can be found using iterative methods such as Newton-Raphson. High yields can come from either high coupons or a low theoretical bond value — its interpretation is consequently subject to caution. It is, however, a useful quantity to understand bonds, and *yield curves* representing yield as a function of maturity for zero-coupon government bonds are often used to analyze rates markets. The bond *spread* corresponds to the difference, expressed in basis points, between the bond yield and the zero-coupon rate of a risk-free bond of the same maturity.

Another useful quantity is the *duration* of a bond. Noting $c_i, i \in [\![1; n]\!]$ the cash flow at time t_i, the theoretical bond price can be written $p_t = \sum_{i=1}^{n} c_i e^{-yt_i}$, where y denotes the yield. The duration of the bond is then defined as

$$D_t = \frac{\sum_{i=1}^{n} t_i c_i e^{-yt_i}}{p_t} = \sum_{i=1}^{n} t_i \left[\frac{c_i e^{-yt_i}}{p_t} \right]. \tag{1.1}$$

Duration can be understood as a weighted average of payment times with a weight corresponding to the proportion of the bond's total present value provided by the i-th cash flow. It consequently measures how long on average an investor has to wait before receiving payments.

Finally, bonds also receive a credit rating representing the credit worthiness of their issuer. Credit rating agencies such as Moody's or Standard & Poors publish them regularly for both corporate and governmental bonds. As they are meant to indicate the likelihood that the debt will be fully repaid, investors use such ratings to guide their investment choices.

More information about interest rates in general and bonds in particular can be found in Hull (2014, Chapter 4). In this work, we cover corporate and G10 bonds, i.e., governmental bonds from countries of the G10 group (Germany, Belgium, Canada, United States, France, Italy, Japan, Netherlands, United Kingdom, Sweden, Switzerland).

Options

Hull (2014, Chapter 10) defines options as financial contracts giving holders *the right* to buy or sell an underlying asset by (or at) a certain date for a specific price. The right to buy an underlying asset is named a *call option*, and the right to sell is named a *put option*. The price at which the underlying asset can be bought/sold is called the *strike* price. The date at which the underlying asset can be bought/sold is called the *maturity* date, as in bonds. If the option allows exercising this right at any time before maturity, it is said to be *American*. If the right can only be exercised at maturity, the option is said to be *European*. American options are the most popular ones, but we consider here European options for ease of analysis. Investors have to pay a price to acquire an option, regardless of the exercise of the provided right.

Options can be bought or sold, and can even be traded. Buyers of options are said to have *long positions*, and sellers are said to have *short positions*. Sellers are also said to *write* the option. Figure 1.2 shows the four different cases possible and their associated payoffs.

To understand the interest of options, let us take the example of an investor buying a European call option for 100 shares of a given stock with a strike price of 50€ at a price of 5€ per share. Assume the price at the moment of the contract was 48€. If at maturity, the price of the stock is 57€, the investor exercises its right to buy 100 shares of the stock at 50€ and directly sells them. She makes a net profit of 700-500=200€. If the investor would have invested the same amount in the stock directly, she would have made a profit of 9€ per share and could have

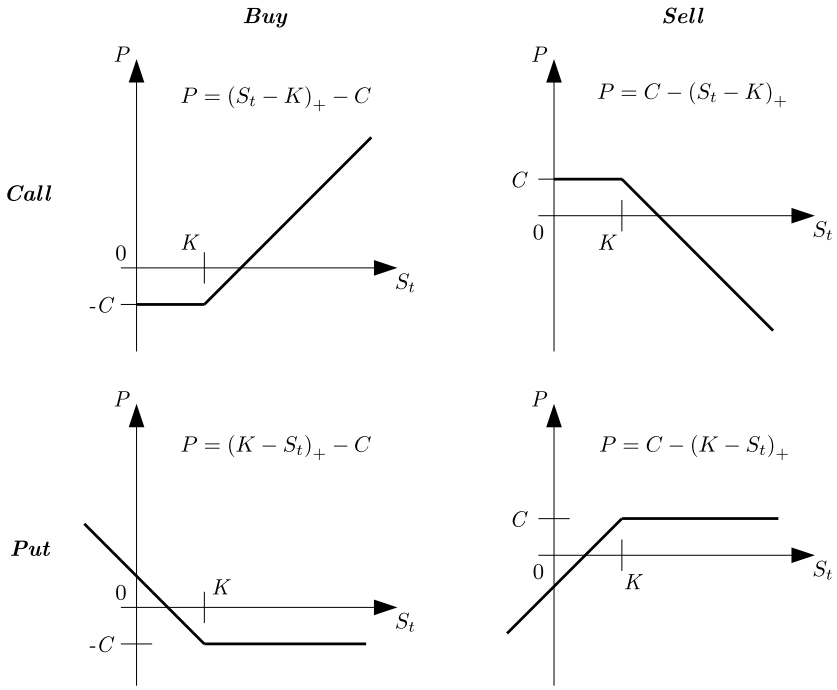


Figure 1.2: The four different options payoff structures. $(\cdot)_+$ denotes here $\max(\cdot, 0)$.

bought only 4 of them, making a net profit of 36€. Options are *leverage* instruments: for the same amount of money invested, they provide the opportunity to obtain far better returns at the cost of higher risk. We see in this example that at prices below 55€, the options investor would start losing money, whereas the buyer of the shares would still make a profit.

In the case of equity options, Hull (2014, Chapter 11) enumerates six factors affecting the price of an option:

- the current price of the equity, noted S_0;
- the strike price K;
- the time to maturity T;
- the volatility of the underlying σ;
- the risk-free interest rate r;
- the dividends D expected to be paid, if any.

Using the *no-arbitrage principle* stating that there are no risk-free investments insuring strictly positive returns, an analysis of these factors allows deriving lower and upper bounds on the option price (Hull, 2014, Chapter 11). The arbitrage analysis allows as well to uncover the *call-put parity*. This property links call and put prices by the formula:

$$C_{call} + Ke^{-rT} + D = C_{put} + S_0, \tag{1.2}$$

considering that stocks with no dividends have $D = 0$.

The price of an option can be derived analytically using stochastic calculus. If we assume, among other hypotheses that one can find in Hull (2014, Chapter 15), that stock prices follow a geometric Brownian motion $dS_t = S_t(\mu dt + \sigma dB_t)$ where μ is the *drift* and σ the *volatility*, we can derive the Black-Scholes equation (Black and Scholes, 1973):

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} - rC = 0, \tag{1.3}$$

where C is the price of the (European) option, S the price of the underlying, σ the volatility of the underlying and r the risk-free rate. Notably, this equation links the price of the option to the volatility of the underlying. Using the bounds found in the arbitrage analysis, the equation can be used both to derive the price of an option using the *historical volatility* computed from past prices of its underlying and to determine the *implied volatility* from the option current price. Implied volatility is an indicator that is used for markets analysis, and can serve as input in more complex models.

Moreover, calls and puts can be combined in *option strategies* to form new payoff structures. Classic combinations include the following:

- *Call spread.* A call spread is the combination of a long call at a given strike and a short call at a higher strike. The payoff corresponds to the one of a call, with a cap on the maximum profit possible. Call spreads allow benefiting from a moderate increase in price, at a lower cost than a classic call option.
- *Put spread.* A put spread is the combination of a short put at a given strike and a long put at a higher strike. The payoff corresponds to the one of a put, with a cap on the maximum profit possible. Put spreads allow benefiting from a moderate decrease in price, at a lower cost than a classic put option.
- *Straddle.* Buying a straddle corresponds to buying a call and a put with the same strike prices and maturities on the same underlying. This position is used in high volatility situations when an investor reckons that the underlying price might evolve a lot in either direction.
- *Strangle.* Buying a strangle corresponds to buying a call and a put with the maturities on the same underlying, but with different strike prices. This position is used in the same situation as a straddle, but can be bought at a lower price.

1.2 Understanding data

We introduced the global characteristics of the RFQ datasets that are studied in this thesis in Section 1.1. However, an in-depth exploration of the data is required to understand the challenges that lie within data: from this exploration follow insights that can guide the construction of an algorithm tailored to face these challenges. Consequently, this section explains the data sources at hand and provides an exploratory analysis of a particularly important characteristic of the datasets we study in this thesis.

1.2.1 Data sources

To solve the problem exposed in Section 1.1, we have various sources of data at our disposal. For all the asset classes that we may consider, we count three data categories.

- **Client-related data.** When an investor becomes a BNPP CIB client, she fills *Know Your Customer* (KYC) information that can be used in our models, such as her sector of activity, her region, ..., i.e., a set of *categorical* features. Due to their size and/or the nature of their business, some clients may also have to periodically report the content of their portfolio. This data is particularly valuable, as an investor cannot sell an asset she does not have. Consequently, knowing their portfolio allows for more accurate predictions — in particular, in the sell direction.
- **Asset-related data.** Financial products are all uniquely identified in the markets by their International Securities Identification Number (ISIN, 2020). Depending on the nature of the asset, categorical features can be derived to describe it. For instance, bonds can be described through their issuer, also known as the ticker of the bond, the sector of

activity of the latter, the currency in which it was issued, its issuance and maturity dates, ... Market data of the assets can also be extracted, i.e., numerical features such as the price, the volatility or other useful financial indicators. Usually, these features are recorded on a daily basis at their *close value*, i.e., their value at the end of the trading day. For options, both market data for the option and its underlying can be extracted.

- **Trade-related data.** The most important data for recommender systems is the trade-related one. This data corresponds to all RFQs, RFMs, and IOIs that were gathered by sales teams (see Section 1.1.1). These requests and indications of interest provide us with the raw interest signal of a client for a given product. As the final decision following a request is not always known, we use the fact that a client made a request on a particular product as a positive event of interest for that product.

These different sources are brought together in various ways depending on the algorithms that we use, as exposed in Section 1.3 and in the next chapters of this thesis.

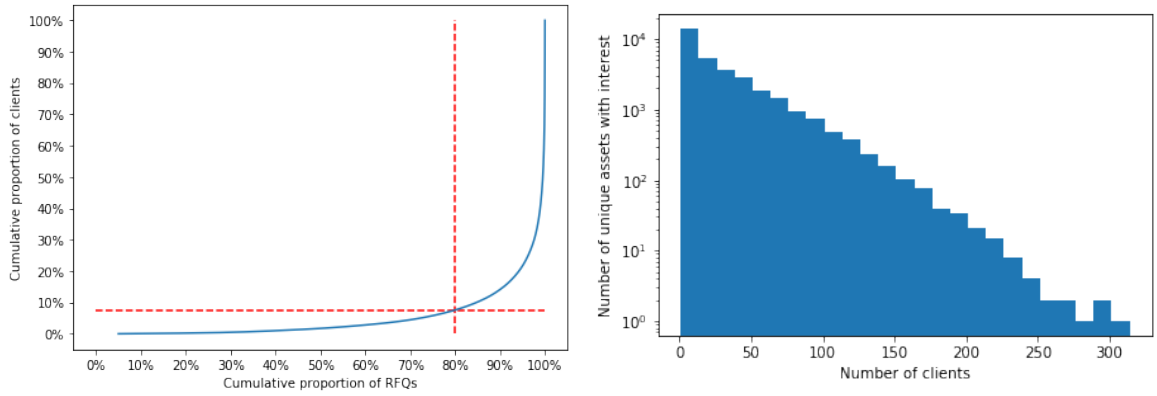
1.2.2 Exploring the heterogeneity of clients and assets

An essential characteristic of our RFQ datasets, common to many fields of application, is heterogeneity. Heterogeneity of activity is deeply embedded in our financial datasets and appears on both the client and asset sides. We explore here the case of the corporate bonds RFQ database, studying a portion ranging from 07/01/2017 to 03/01/2020. In this period, we count roughly 4000 clients and 30000 bonds. Of these 4000 clients, about 23% regularly disclose the content of their bond portfolios — this information is consequently difficult to use in a model performing predictions for all clients.

Let us begin with the heterogeneity of clients. A glimpse at heterogeneity can be taken by looking at a single client: the most active client accounted for 5% of all performed RFQs alone, and up to 6.4% when taking into account all its subsidiaries. The heterogeneity of clients can be further explored with Fig. 1.3. In Fig. 1.3a, we see that 80% of the accounted RFQs were made by less than 10% of the client pool. Most of BNPP CIB business is consequently done with a small portion of its client reach: a recommender system could help better serve this portion of BNPP clients by providing sales with offers tailored to their needs. The histogram shown in Fig. 1.3b provides another visualization of this result. We see here that a few clients request prices for thousands of unique bonds, whereas over the same period, some clients only request prices for a handful of bonds. Clients are consequently profoundly heterogeneous. Note however that this heterogeneity, to some extent, might only be perceived as such as we do not observe the complete activity of our clients and only what they do on platforms on which BNPP CIB is active.

We now give a closer look at the heterogeneity of bonds. In particular, here, the most active bond accounted for 0.3% of all RFQs, and up to 1% when aggregating all bonds emitted by the same issuer. A global look at these statistics is presented in Fig. 1.4a. To avoid bias from expiring products, we only account in this graph for bonds with issue and maturity dates strictly outside of the considered period. We see here that 80% of the RFQs were done on more than 30% of the considered assets. The heterogeneity of bonds is consequently less pronounced that the heterogeneity of clients. There are however more pronounced popularity effects, as can be seen in Fig. 1.4b. Here, we see that only a small portion of bonds are shared among the client pool. Most bonds can be considered as *niche*, and are only traded by a handful of clients. Consequently, even though bonds are, on average, more active than clients, their relative popularities reveal their heterogeneity.

In Fig. 1.3 and 1.4, **(b)** graphs both show the *sparsity* of this dataset. Most clients only trade a handful of bonds, and most bonds are only traded by a handful of clients. It can



(a) **(b)**

Figure 1.3: **(a)** Cumulative proportion of clients as a function of the cumulative proportion of RFQs, with clients ordered in descending order of activity. We see that 80% of RFQs account for less than 10% of the clients. **(b)** Histogram representing the number of unique assets of interest to a particular client. The x-axis shows the number of clients in the bucket corresponding to a specific number of unique assets. A few clients show interest in more than a thousand bonds, whereas the largest buckets show interest for a couple ones only.

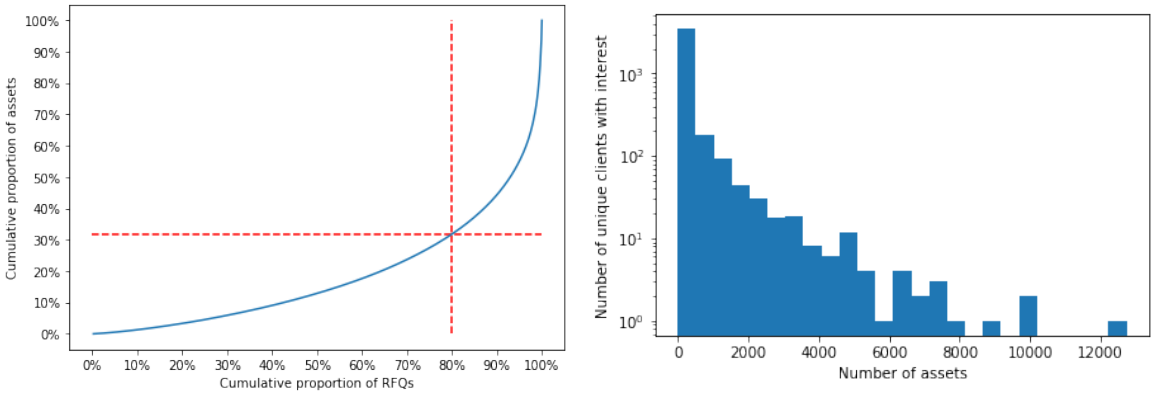
consequently become difficult to build recommender systems able to propose the entire asset pool to a given client, and reciprocally. This phenomenon is referred to in the literature as the *long tail* problem (Park and Tuzhilin, 2008), which refers to the fact that for most items, very few activity is recorded. Park and Tuzhilin (2008) proposes a way to tackle this by splitting the item pool in a head and tail parts and train models on each of these parts, the tail model being trained on clustered items. More broadly, the long tail is tackled by systems that favor *diverse* recommendations, i.e., recommendations that are made outside of the usual clients' patterns. Diversity is, however, hard to assess: as recommendations outside of clients' patterns cannot be truly assessed on historical data, one has to monitor their efficiency with A/B testing. Section 1.4 introduces metrics that give a first insight into the diversity of a model without having to fall back on live monitoring.

1.3 A non-exhaustive overview of recommender systems

For the remainder of this thesis, the terms clients and users (resp. financial products/assets and items) are used interchangeably to match the vocabulary used in recommender systems literature.

Digitalization brought us in a world where more and more of our decisions are taken in spite of information overload. Recall that Negre (2015) defines recommender systems as "an information-filtering technique used to present the items of information (video, music, books, images, Websites, etc.) that may be of interest to the user." Recommender systems have consequently gained much traction as they help users navigating problems overcrowded with potential choices. They can be found almost everywhere, from booking a hotel room (e.g., with Booking.com (Bernardi et al., 2015)), a flat (e.g., with AirBnB (Haldar et al., 2019)), shopping online (e.g., on Amazon.com (Sorokina and Cantu-Paz, 2016))... to watching online videos (e.g., on YouTube (Covington et al., 2016)) or films (e.g., on Netflix (Koren, 2009a)). This section provides a non-exhaustive overview of recommender systems to lay the ground on which we build algorithms well-suited to the challenges of the financial world.

Recommender systems are built from the feedback that users provide about their items of



(a) **(b)**

Figure 1.4: **(a)** Cumulative proportion of assets as a function of the cumulative proportion of RFQs, with assets ordered in descending order of activity. To avoid bias from expiring products, we only consider bonds alive in the whole period studied in this graph. We see that 80% of RFQs account for roughly 30% of the assets. **(b)** Histogram representing the number of unique clients interested in a particular asset. The x-axis shows the number of assets in the bucket corresponding to a specific number of unique clients. We see that a few assets are traded by lots of clients, whereas lots of assets were of interest for a small portion of the clients pool only.

interest. Users can provide two sorts of feedback depending on the considered application: either an *explicit* or an *implicit* one. Explicit feedback corresponds to the ratings users give to items, such as a "thumbs up" on YouTube or a ten-star rating on IMDb. Implicit feedback corresponds to any feedback that is not directly expressed by users, such as a purchase history, a browsing history, clicks on links... : the interest of the user for the item is implied from her behavior. As explained in Section 1.1.1, our goal is to predict the future RFQs of BNPP CIB clients, given the RFQs they made and the RFQs made by everyone else. RFQs are also an example of implicit feedback: requesting the price of a product does not say anything about the client's appeal for that product, or what the client ultimately chooses to do with it. As implicit feedback is our setting of interest in this thesis, we focus on it from now on.

In the implicit feedback setting, recommender systems can be understood as mappings $f : t, u, i \longrightarrow x_{ui}^t$, where t is a given time-step, $u \in U$ is a given client, $i \in I$ is a given product and x_{ui}^t represents the interest of client u for product i at t. In most applications, the interest's dependence on time is ignored. x_{ui}^t is usually interpreted either as a *probability* of interest or as a *score* of interest, depending on the chosen mapping f. For a given client u, at a given t, the corresponding *recommendation list* is the list of all products ranked by their score in descending order, i.e., as $\left(x_{u(|I|-i)}^t, i \in I\right)$ with $(\cdot)$ the order statistics. For some real-world applications, e.g., YouTube (Covington et al., 2016), the cardinality of I prevents from computing scores for all $i \in I$ and workarounds are used to compute scores for a relevant portion of items only. In our setting of interest, the cardinalities of both U and I allow for computing the scores of all *(client, product)* pairs — we consequently focus on finding well-suited f mappings.

There are two recommender systems paradigms (Koren et al., 2009):

- **Content-filtering.** This approach characterizes clients and/or products through *profiles*, which are then used to match clients with relevant products and conversely. A classical formulation involves learning a mapping $f_u : t, i \longrightarrow x_{ui}^t$ for each $u \in U$, with classic supervised learning approaches where the items' profiles are used as inputs to the f_u's. Conversely, mappings could depend on items and use as inputs users' profiles. An al-

ternative approach is to use a global mapping $f : t, u, i \longrightarrow x_{ui}^t$ using as input, e.g., a concatenation of u and i profiles at t, and features depending on both u and i.

- **Collaborative filtering.** The term, coined by Goldberg et al. (1992) for Tapestry, the first recommender system, refers to algorithms making predictions about a given client using the knowledge of what all other clients have been doing. According to Koren et al. (2009), the two main areas of collaborative filtering are *neighborhood methods* and *latent factor models*. Neighborhood methods compute weighted averages of the rating vectors of users (resp. items) to find missing (user, item) ratings, e.g., using user-user (resp. item-item) similarities as weights. Latent factor models characterize users and items with vector representations of given dimension d inferred from observed ratings and use these vectors to compute missing (user, item) ratings. These two approaches can be brought together, e.g., as in (Koren, 2008).

Content- and collaborative filtering are not contradictory and can be brought together, e.g., as in Basilico and Hofmann (2004). Learning latent representations of users and items, as is done in latent factor models, has advantages beyond inferring missing ratings. These representations can be used *a posteriori* for clustering, visualization of user-user, item-item or user-item relationships and their evolution with time,... These applications of latent factors are particularly useful for business, e.g., to understand how BNP Paribas CIB clients relate to each other and consequently reorganize the way sales operate. For that reason, the collaborative filtering algorithms we study here are mainly latent factor models.

We now examine how baseline recommender systems can be obtained, develop on the content filtering approach with a specific formulation of the problem as a classification one, expand on some collaborative filtering algorithms and provide insights into other approaches for recommender systems.

1.3.1 Benchmarking algorithms

Baselines are algorithms that serve as reference for determining the performance of a proposal algorithm. Consequently, baselines are crucial for understanding how well an algorithm is doing on a given problem. In computer vision problems such as the ImageNet challenge (Russakovsky et al., 2015), the baseline is a human performance score. However, obtaining a human performance baseline is more complicated in recommender systems as the goal is to automate a task that is inherently not doable by any human: it would require at the very least an in-depth knowledge of the full corpus of users and items for a human to solve this recommendation task.

A simple recommender system baseline is the *historical* one. Using a classical train/test split of the observed events, a first baseline is given by the mapping $f_{hist} : (u, i) \longrightarrow r_{ui}$ with $r_{ui} \in \mathbb{N}$ the number of observed events for the (u, i) couple on the train split. The score can also be expressed as a frequency of interest, either on the user- or item-side; i.e., on the user-side as $r_{ui} / \sum_{i \in I} r_{ui}$. The historical model can also be improved using the temporal information, considering that at a given t, closer interests should weigh more in the score. An example of such mapping is

$$f_{hist} : (t, u, i) \longrightarrow \sum_{t'=1}^{t-1} e^{-\lambda(t+1-t')} r_{ui}^{t'} \tag{1.4}$$

where $r_{ui}^{t'}$ denotes the number of observed events for the (u, i) couple at t', and λ is a decay factor. Note that the sum stops at $t-1$ for prediction at t since observed events at t cannot be taken into account for predicting that time-step. In the following, we mainly use the most simple instance of baseline where the score of a (u, i) couple is given by their number of interactions observed in the training split.

1.3.2 Content-filtering strategies

We previously defined content filtering algorithms as mapping profiles of users, items, or both of them to the interest x_{ui}^t. The difficulty of this approach lies in the construction of users and items' profiles, as they require data sources other than the observation of interactions between the two. As exposed in Section 1.2, we have at our disposal multiple data sources from which to build such profiles. Concretely, each of the three sources outlined in Section 1.2.1 leads to a profile — one can consequently build a user profile, an item profile, and an interaction profile from the available data.

The profile of a user (resp. item) is built from the numerical and categorical information available describing her (resp. its) state. Numerical features such as the buy/sell frequency of a user or the price of an item can be directly used. However, categorical features, e.g., the user's country or the company producing the item, require an encoding for an algorithm to make use of them. Categories can be encoded in various ways, the most popular ones in machine learning being *one-hot* encoding, *frequency* encoding and *target* encoding. One-hot encoding converts a category $c \in [\![1; K]\!]$ to the c-th element e_c of the canonical basis of $\mathbb{R}^K$, $e_c = (0, \dots, 1, \dots, 0) \in \mathbb{R}^K$, i.e., a zero vector with a 1 in the c-th position. Frequency encoding converts a category $c \in [\![1; K]\!]$ to its frequency of appearance in the training set. Target encoding converts a category $c \in [\![1; K]\!]$ to the mean of targets y observed for particular instances of that category. One-hot encoding is a simple and widespread methodology for dealing with categorical variables; frequency and target encoding, to the best of our knowledge, could be more qualified of "tricks of the trade" used by data science competitors on platforms such as Kaggle (Kaggle Inc., 2020). One-hot encodings are, by essence, sparse and grow the number of features used by a model by as many categories there are per categorical feature. Deep learning models usually handle categorical features through *embeddings*, i.e., trainable latent representations of a size specifically chosen for each feature. These embeddings are trained along with all other parameters of the neural networks.

Provided with embeddings of categorical features describing the state of a particular user (resp. item), a profile of this user (resp. item) can be obtained by the concatenation of the numerical features and embeddings related to her. These profiles are then used to learn mappings f_i for all $i \in I$ (resp. f_u for all $u \in U$), or can be concatenated to learn a global mapping $f : (t, u, i) \longrightarrow x_{ui}^t$. One can also add cross-features depending on both u and i, such as the frequency at which u buys or sells i. Indeed, in the financial context, it is usual for investors to re-buy or re-sell in the future products they bought or sold previously — cross-features describing the activity of a user u on an item i are consequently meaningful. A good practice for numerical features is to transform them using *standard scores*, as known as z-scores. A feature x is turned into a z-score z with $z = \frac{x-\mu}{\sigma}$, where μ and σ are the mean and standard deviation of x, computed using the training split of the data. Z-scores can also be computed on categories of a given categorical feature, using μ_c and σ_c for all $c \in [\![1; K]\!]$, provided that we have enough data for each category to get meaningful means and variances. The distribution of a feature can vary much with categories: for instance, the buying frequency of a hedge fund is not the same as the one of a central bank. Using z-scores modulated per category of a categorical feature consequently helps to make these categories comparable.

In the implicit feedback setting, the recommendation problem can be understood as a classification task. Provided with a profile, i.e., a set of features x describing a particular (t, u, i) triplet, we want to predict whether u is interested in i or not at t, i.e., predict a binary target $y \in \{0, 1\}$. If, as mentioned in Section 1.1.1, we want to predict whether u is interested in buying and/or selling i, the target y can consequently take up to four values corresponding to a lack of interest, an interest in buying, in selling or in doing both, the last case happening only when the time-step is large enough to allow for it — the size of the time-step usually depends

on the liquidity of the considered asset class, as mentioned in Section 2.2.4. Classification is a supervised learning problem for which many classic machine learning algorithms can be used. A classic way to train a classifier is to optimize its *cross-entropy* loss, defined for a sample x as

$$l(x) = -\sum_{i=1}^{C} y_i(x) \log \hat{y}_i(x) \ ,$$

a term derived from maximum likelihood where C is the total number of classification classes, $y_i(x) = \mathbb{1}_{y(x)=i}$ and $\hat{y}_i(x)$ the probability of class i outputted by the model for sample x. A variation of cross-entropy, the *focal loss* (Lin et al., 2017), can help in cases of severe class imbalance. Focal loss performs an adaptive, per-sample reweighing of the loss, defined for a sample x as

$$l_{focal}(x) = -\sum_{i=1}^{C} (1 - \hat{y}_i(x))^{\gamma} y_i(x) \log \hat{y}_i(x) \ , \tag{1.5}$$

with γ an hyperparameter, usually chosen in the $]0; 4]$ range and depending on the imbalance ratio, defined in the binary case as the ratio of the number of instances in the majority class and the ones in the minority class. In imbalance settings, the majority class is easily learnt by the algorithm and accounts for most of the algorithm loss performance. The $(1 - \hat{y}_i(x))^{\gamma}$ term counterbalances this by disregarding confidently classified samples, and focuses the sum on "harder-to-learn" ones. Consequently, when the imbalance ratio is high, higher values of γ are favored.

The classifier can take many forms, from logistic regression (Hastie et al., 2009, Chapter 4) to feed-forward neural networks (Goodfellow et al., 2016). To date, one of the most widespread classes of algorithms is gradient tree-boosting algorithms (Hastie et al., 2009, Chapters 9,10). Popular implementations of gradient tree-boosting are XGBoost (Chen and Guestrin, 2016) and LightGBM (Ke et al., 2017). In a classic supervised learning setting with tabular data, these algorithms most often obtain state-of-the-art results without much effort. However, to handle time, these algorithms have to rely on hand-engineered features, as they only map a given input x to an output y. Instead of relying on hand-engineered features incorporating time, we could let the algorithm learn by itself how features should incorporate it using neural networks. Fawaz et al. (2019) provides a review of deep learning algorithms tackling time series classification. Two neural network architectures can be used to handle time series: recurrent neural networks, such as the Long Short-Term Memory (Hochreiter and Schmidhuber, 1997), or convolutional neural networks (see Appendix A.4). These networks map an input time series to a corresponding output one. For instance, in our case, we could use as input a series of features describing the state of a (u, i) pair at every considered time-step and as output the observed interest at that time. Convolutional neural networks are mostly used in visual recognition tasks. Their unidimensional counterpart used in natural language processing (NLP) tasks is, however, also appropriate for time series classification. WaveNet-like architectures (Oord et al., 2016), illustrated in Fig. 1.5, are particularly relevant to handle short- and long-term dependencies in time series through their causal, dilated convolutions.

Content-filtering is consequently a very diverse approach in terms of potential algorithms to use, as it can be seen as re-framing the problem of recommendation to the supervised learning task of classification. Its main shortcomings are linked to the usage of features. One needs to have features, i.e., external data sources, to build profiles, which is not a given in all recommender systems. Moreover, when profiling the (t, u, i) triplet, hand-engineering cross-features can lead to overfitting the historical data. An overfitted algorithm would then only recommend past interactions to a user: in most application cases, such recommendations are valueless for a user.

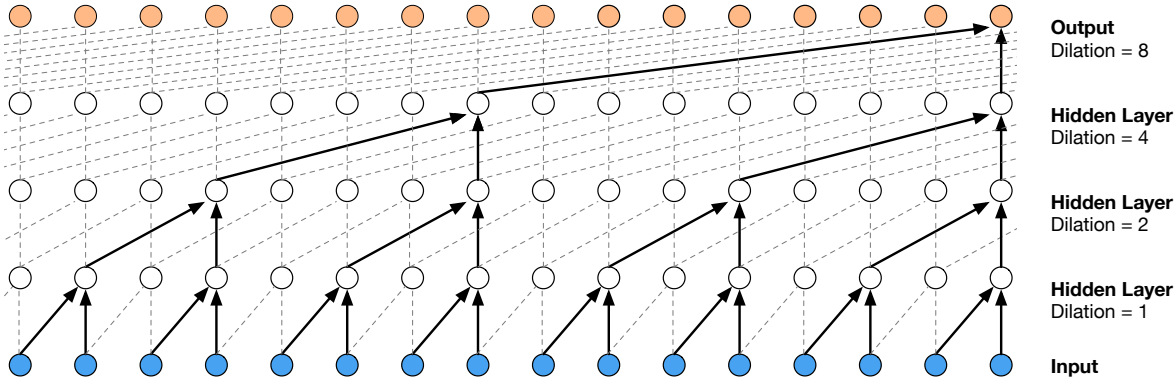


Figure 1.5: The WaveNet architecture, using causal and dilated convolutions. Causal convolutions relate their output at t to inputs at $t' \leq t$ only. Dilated convolutions, as known as *à trous* convolutions, only consider as inputs to a particular output index the multiples of the dilation factor at that index. Dilated convolutions extend the *receptive field* of the network, i.e., the period of time taken into account for the computation of a particular output. WaveNets used stacked dilated convolutions, with a dilation factor doubled up to a limit and repeated, e.g., 1, 2, 4, 16, 32, 1, 2, 4, 16, 32, ... to get the widest receptive field possible. Illustration taken from Oord et al. (2016).

1.3.3 Collaborative filtering strategies

The collaborative filtering approach refers to algorithms using global information consisting of all observed (t, u, i) interactions to make local predictions about a given unobserved (t, u, i) triplet. As a matter of fact, recommendation tasks can be understood as predicting links in a bipartite graph $G_t = (U, I, E_t)$ with U the set of users, I the set of items and E_t the set of edges connecting them, which may evolve with time depending on the studied problem. E_t can be summarized as an adjacency matrix $A^t \in \mathbb{R}^{|U|\times|I|}$ where $a^t_{ui} = 1$ if $(u, i) \in E_t$, 0 else. When E_t evolves with time, the global information can be summarized in a three-dimensional, adjacency tensor $A \in \mathbb{R}^{T\times|U|\times|I|}$ with T the total number of time-steps considered and $A[t; u; i] = A^t[u; i]$. Note that A^t, and by extension A, is generally *sparse*, meaning that most of its entries are 0 — only a small portion of all the (u, i) couples are observed with a positive interaction. See Fig. 1.6 for an illustration of a bipartite graph and its associated adjacency matrix.

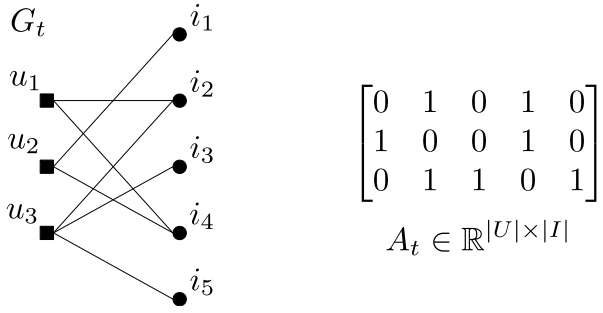


Figure 1.6: An example of bipartite graph and its associated adjacency matrix, where $|U| = 3$ and $|I| = 5$.

Consequently, one can think of collaborative filtering algorithms as learning to reconstruct the full adjacency matrix/tensor from an observed portion of it. We examine here algorithms that particularly influenced the work presented in this thesis. An extensive review of collaborative filtering can be found in (Su and Khoshgoftaar, 2009), and a survey of its extensions in (Shi

et al., 2014).

Matrix factorization and related approaches

Matrix factorization models learn to decompose the adjacency matrix into two matrices of lower dimension representing respectively users and items. For now, we disregard the influence of time, setting $t = 0$ and writing for convenience $A := A_0$. A matrix factorization model learns latent, sparse and lower-rank representations of users and items $P \in \mathbb{R}^{|U| \times d}, Q \in \mathbb{R}^{|I| \times d}$ such that

$$A \approx PQ^T \tag{1.6}$$

with d an hyperparameter of the algorithm usually chosen as $d \ll |U|, |I|$. This equation is illustrated in Fig. 1.7. It follows that a given user u is mapped to a latent representation $p_u \in \mathbb{R}^d$, and a given item i respectively to $q_i \in \mathbb{R}^d$. The recommendation score of a given (u, i) couple corresponds to the scalar product of their latent representations, $x_{ui} = \langle p_u, q_i \rangle \approx a_{ui}$.

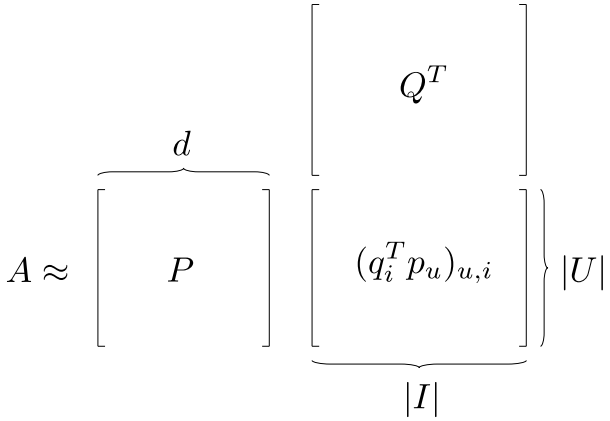


Figure 1.7: An illustration of the principle of matrix factorization in a recommendation framework.

A variation of matrix factorization, called *nonnegative matrix factorization* (NMF) (Paatero and Tapper, 1994; Hoyer, 2004) considers constraining the latent representations of users and items to nonnegative vectors. The motivation is twofold. First, in the implicit feedback setting, the matrix A is nonnegative itself, and it consequently makes sense to express it with matrices sharing the same constraint. Second, the latent factors can be understood as a decomposition of the final recommendation score, and a decomposition of an element in its latent components is generally expressed as a summation, from which an explanation can be derived — e.g., in (Hoyer, 2004) with the decomposition of face images in their parts (mouth, nose, ...) found using nonnegative matrix factorization. However, in a financial setting, latent factors are not explicit and cannot be conceptualized into meaningful components of a recommendation score. For that reason, the rationale behind NMF is usually dealt with a classic matrix factorization where scores are constrained to a given range, e.g., using the *sigmoid* function $\sigma(x) = 1/(1+e^{-x})$ to map x_{ui} to $[0; 1]$ in the implicit feedback setting.

Classic matrix factorization learns latent representations $p_u, \forall u \in U$, $q_i, \forall i \in I$ by minimizing the squared distance of the reconstruction PQ^T to the adjacency matrix A, computed on the set of observed scores κ (Koren et al., 2009), as

$$\min_{p_*, q_*} \sum_{(u,i) \in \kappa} \left(a_{ui} - q_i^T p_u\right)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{1.7}$$

with λ an hyperparameter controlling the level of L_2-norm regularization. The parameters of this model can be learnt either with stochastic gradient descent (see Appendix A) or with alternating

least squares (Hu et al., 2008; Pilászy et al., 2010; Takács and Tikk, 2012). A probabilistic interpretation of this model (Mnih and Salakhutdinov, 2008) shows that this objective function is equivalent to a Gaussian prior assumption on the latent factors.

This formulation of matrix factorization is particularly relevant for explicit feedback. In implicit feedback however, all unobserved events are considered negative, and positive ones are implied — a (u, i) pair considered positive does not assuredly correspond to a true positive event, as explained in the incipit of this section. To counter these assumptions, Hu et al. (2008) introduce a notion of *confidence* weighing in the above equation. Keeping r_{ui} as the number of observed interactions of the (u, i) pair in the considered dataset, the authors introduce two definitions of confidence:

- *Linear:* $c_{ui} = 1 + \alpha r_{ui}$
- *Logarithmic:* $c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon)$

where α, ϵ are hyperparameters of the algorithm. We empirically found logarithmic confidences to provide better results, a performance that we attribute to logarithms flattening the highly heterogeneous users' and items' activity we observe in our datasets (see Section 1.2.2). Defining $p_{ui} := \mathbb{1}_{r_{ui}>0} = a_{ui}$, the implicit feedback objective is written

$$\min_{p_*, q_*} \sum_{u,i} c_{ui} \left(p_{ui} - q_i^T p_u\right)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2). \tag{1.8}$$

Note that the summation is now performed on all possible (u, i) pairs. Most often, the total number of pairs makes stochastic gradient descent impractical, and these objectives are optimized using alternating least squares. In our case, however, the total number of pairs of a few millions — tens of millions, in the worst case — still allows for gradient descent optimizations.

A related approach is *Bayesian Personalized Ranking* (BPR) (Rendle et al., 2009), an optimization method specifically designed for implicit feedback datasets. The goal, as initially exposed, is to provide users with a ranked list of items. BPR abandons the probability approach for x_{ui} used in the above algorithms to focus on a scoring approach. The underlying idea of BPR is to give a higher score to observed (u, i) couples than unobserved ones. To do so, the authors formalize a dataset $D = \{(u, i, j)|i \in I_u^+ \wedge j \in I \backslash I_u^+\}$ where I_u^+ is the set of items which have a positive event with user u in the considered data. D is consequently formed of all possible triplets containing a positive (u, i) pair and an associated negative item j. The BPR objective is then defined as maximizing the quantity

$$L_{BPR} = \sum_{(u,i,j)\in D} \ln \sigma(x_{uij}) - \lambda \|\Theta\|^2 \tag{1.9}$$

where x_{uij} is the score associated to the (u, i, j) triplet, defined such that $p(i >_u j|\Theta) := x_{uij}(\Theta)$ with $>_u$ the total ranking associated to u — i.e., the ranking corresponding to the preferences of u in terms of items, see Rendle et al. (2009, Section 3.1)—, and λ is an hyperparameter controlling the level of regularization. As D grows exponentially with the number of users and items, this objective is approximated using negative sampling (Mikolov et al., 2013). Following matrix factorization ideas, we define $x_{uij} = x_{ui} - x_{uj}$, with $x_{ui} = q_i^T p_u$ where p_u, q_i are latent factors defined as previously. The BPR objective is a relaxation of the ROC AUC score (see Section 1.4), a ranking metric also used in the context of binary classification. BPR combined with latent factors is, at the time of writing this thesis, one of the most successful and widespread algorithms to solve implicit feedback recommendation.

In specific contexts, Average Precision (AP) and the related mean Average Precision (mAP) are more reliable metrics than ROC AUC to score recommender systems — see Section 1.4 for an in-depth explanation of AP, mAP and a comparison of AP and AUC. Optimizing a relaxation

of mAP is proposed in (Eban et al., 2017; Henderson and Ferrari, 2016). Extending the work of Chapelle and Wu (2010), Shi et al. (2012) propose a relaxation of mAP trained in a way similar to BPR. They derive a smoothed approximation of user-side mAP as

$$L_{mAP} = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{\sum_{i=1}^{|I|} a_{ui}} \sum_{i=1}^{|I|} a_{ui} \sigma(x_{ui}) \sum_{j=1}^{|I|} a_{uj} \sigma(x_{uij}) \tag{1.10}$$

with σ, x_{ui}, x_{uij} defined as previously. This term corresponds to a smoothed approximation of the interpretation of mAP as an average of precisions at L, detailed in Section 1.4. Using properties of average precision, Shi et al. (2012) also derive a sampling technique to approximate the summations in a fast way. Still, as appealing as it seems to directly optimize mAP, this appears too computation-intensive to be of practical use.

Neural network-based models

In recent years, neural networks have made their appearance in the field of recommender systems. We cover here some interesting neural architectures and approaches that extend the above algorithms.

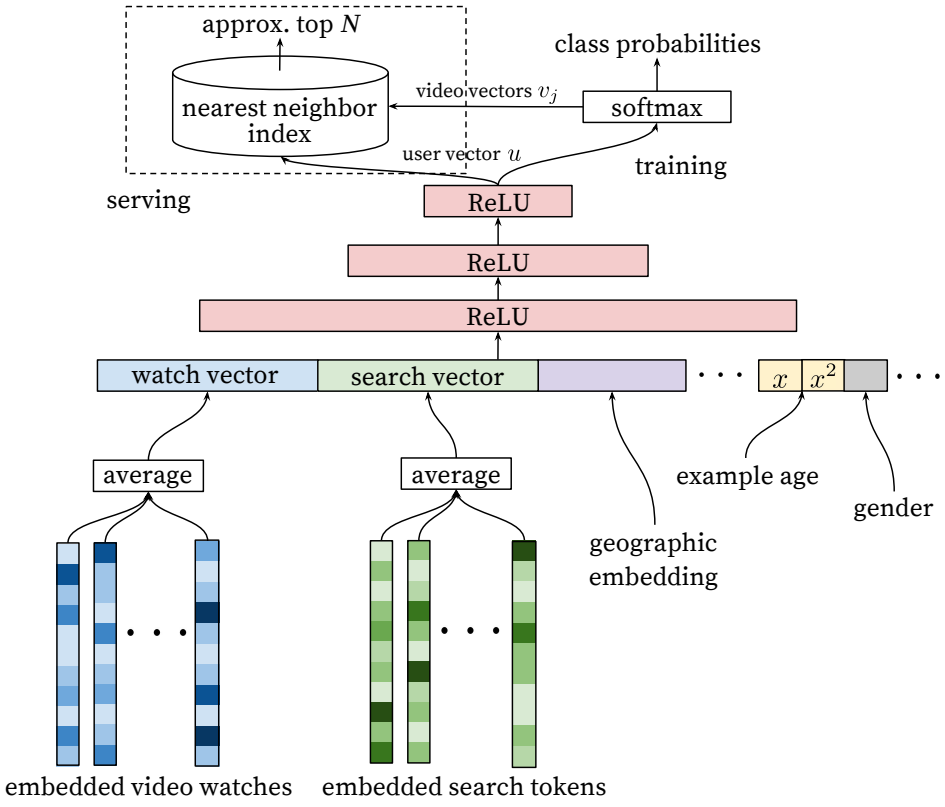


Figure 1.8: Candidate generation network of the YouTube recommender system. It is composed of a feed-forward neural network, receiving as inputs a series of features related to the behavior of the user on the platform, and features related to the *age* of the training sample. The network outputs an embedding of the considered user, which is mapped to N nearest neighbors at serving time to obtain the set of videos that are scored by the ranking network. Illustration taken from Covington et al. (2016).

With millions of users and billions of videos, YouTube is a particularly relevant field of application for recommender systems. When a user is connected to the YouTube platform, a

customized set of videos appears on the home page. Covington et al. (2016) explain that these recommendations are made using a two-parts recommender system, using two distinct neural networks. The first one is a *candidate generation* network, which role is to select a set of personalized videos for a given user among the total corpus of YouTube videos, and the second one is a *ranking* network, which role is to rank selected videos to promote content that engages users. To promote engagement, the ranking network is trained on expected watch time instead of click-through-rate.

The recommendation task is split into two separate tasks because of the particularly high number of items to recommend — there are billions of candidate YouTube videos. We focus here on the candidate generation network, illustrated in Fig. 1.8, as it corresponds more closely to our setting. This network is trained on cross-entropy, using as classes the total corpus of YouTube videos considered. The probability of a class is determined using the softmax function, such that for a given (u, i) couple, with $u \in U$ a user and $i \in I$ a video, the corresponding probability is given by $e^{\langle x_u, x_i \rangle} / \sum_{j \in I} e^{\langle x_u, x_j \rangle}$ where the x_is are trainable embeddings of the videos and x_u corresponds to the output of the network. Interestingly, the purpose of this architecture is to output user embeddings. The network only maintains embeddings of videos, search tokens and the considered categorical variables, and assimilates users as the average embedding of their watch history in its input. Doing so allows for providing recommendations to new users very easily, as no embedding is directly attached to the users. It is also worth noting that even though YouTube provides ways for users to give explicit feedback (thumbs up/down, subscribing...), authors use *implicit feedback* signals to build positive events. Implicit feedback allows for more positives in general, and, in particular, for more positives in the long tail of YouTube videos where users provide in general far less feedback explicitly, as seen in Section 1.2.2.

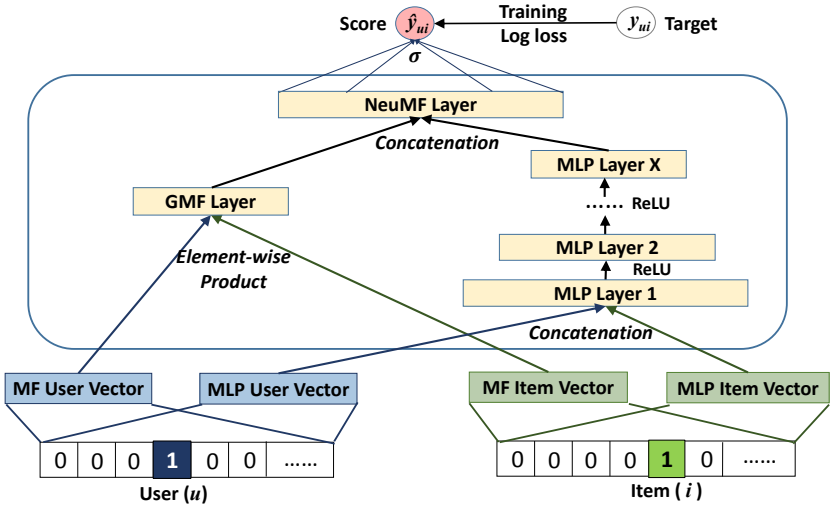


Figure 1.9: The NeuMF architecture, which combines matrix factorization with a multilayer perceptron to perform recommendation. The outputs of both models are concatenated and fed to a logistic regression model trained with cross-entropy. Illustration taken from He et al. (2017).

An architecture directly related to matrix factorization is the one presented in He et al. (2017), which builds on the work presented in Cheng et al. (2016). He et al. (2017) introduce a dual architecture called NeuMF, illustrated in Fig. 1.9. The core idea is to augment the capacities of a classic matrix factorization with the feature extraction capabilities of deep learning without harming the results of any of the two in any way. To that extent, NeuMF maintains two sets of user and item embeddings. One set is devoted to the matrix factorization layer, which computes their element-wise product $x_u \odot x_i$, and the other is used as inputs to a feed-forward multilayer

perceptron with ReLU activations (Nair and Hinton, 2010). The outputs of the matrix factorization layer and the multilayer perceptron are then concatenated as $[h_{MF}, (1-\alpha)h_{MLP}]$, where α is a trade-off hyperparameter. The concatenation is then fed to a logistic regression model trained with cross-entropy.

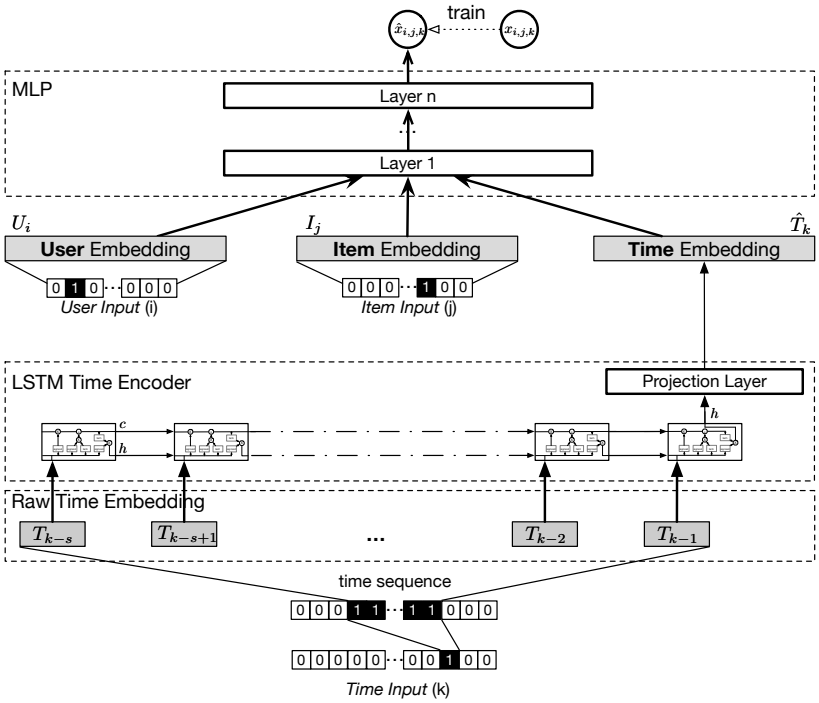


Figure 1.10: A neural tensor factorization architecture. Users and embeddings are treated as previously, whereas time embeddings are generated using an LSTM encoder. Illustration taken from Wu et al. (2018).

Last, one can use neural network architectures to perform tensor factorization, as done in (Wu et al., 2018). Considering a temporal context with $t \in [\![1; T]\!]$, a tensor factorization model decomposes a three-dimensional tensor into latent d-dimensional representations of users x_u, items x_i, and time x_t. Elements of the original tensor are then approximated by $\sum_{l=1}^{d} x_{u,l} x_{i,l} x_{t,l}$, and the network is trained with euclidean distance between original elements and their approximation. The model is illustrated in Fig. 1.10. Embeddings of users and items are used as previously. In this architecture, each time-step is embedded. These embeddings serve as inputs to an LSTM encoder, which uses the s previous time-steps $k \in [\![t-s; t-1]\!]$ to compute the current time-step embedding x_t. Note that tensor factorization approaches such as this one can only make predictions for time-steps seen during training — they are unable to *extrapolate*, i.e., perform predictions in the future.

Note, however, that doubts were cast on some neural network approaches. Dacrema et al. (2019) claim in a recent report that most results of neural network recommenders are not reproducible. Consequently, deep learning in recommender systems must be taken with a grain of salt, and carefully examined against algorithms that proved their efficiency on multiple benchmarks, such as the one presented in Chapter 2.

Resource redistribution models

Resource redistribution models were introduced in (Zhou et al., 2007). The underlying idea is to perform a weighted one-mode projection, i.e., projecting the bipartite graph on a user graph by connecting users that interacted with the same items (or resp. for items with users), in a way that preserves the information encoded in the bipartite graph. These weights W can then be used to make personalized recommendations — e.g., considering a vector $f_u = (a_{ui})_{i \in I} \in \mathbb{R}^{|I|}$ and a weight matrix corresponding to the one-mode projection of the bipartite graph on the items graph, personal recommendations for user u can be obtained as $W f_u$.

Zhou et al. (2010) uses this principle to derive two models, called HeatS and ProbS for *heat-spreading* and *probabilistic spreading*, respectively aiming at diversity and accuracy of recommendations, and propose a hybrid model that combines the qualities of the two. The resource-distribution strategies for both these models is illustrated in Fig. 1.11. The corresponding weight matrices $W^H, W^P \in \mathbb{R}^{|I| \times |I|}$ are written

$$W^H_{ij} = \frac{1}{k_i} \sum_{u=1}^{|U|} \frac{a_{ui} a_{uj}}{k_u} \tag{1.11}$$

$$W^P_{ij} = \frac{1}{k_j} \sum_{u=1}^{|U|} \frac{a_{ui} a_{uj}}{k_u}, \tag{1.12}$$

where k_* represent the degree of the corresponding node in the bipartite graph, and a_{ui} is defined as previously. We see here that $W^H = (W^P)^T$. The hybrid method, called SPHY in (Zeng et al., 2014) for similarity-preferential hybrid, is obtained with

$$W^{SPHY}_{ij} = \frac{1}{k_i^{1-\lambda} k_j^{\lambda}} \sum_{u=1}^{|U|} \frac{a_{ui} a_{uj}}{k_u}, \tag{1.13}$$

with λ an hyperparameter to tune.

Zeng et al. (2014) extend these models by proposing to scale the weight matrices with a hyperparameter θ as $(W^{SPHY})^{\theta}$ in a one-mode projection on the users graph. The role of θ is to enhance or suppress the effects of similar users. Consequently, its value depends on the sparsity of the one-mode network: in sparse cases, one would decrease the weight of similar users with $\theta < 1$ so that rarely seen users have weights closer to more popular ones, and consequently be recommended nonetheless.

Contrarily to most publications in recommender systems, these algorithms stress the importance of diversity in recommendations and provide a direct way to obtain more diverse ones, along with metrics monitoring this diversity. Moreover, as these algorithms only rely on direct operations on the adjacency matrices, they are easily implemented and provide results faster than any of the above algorithms.

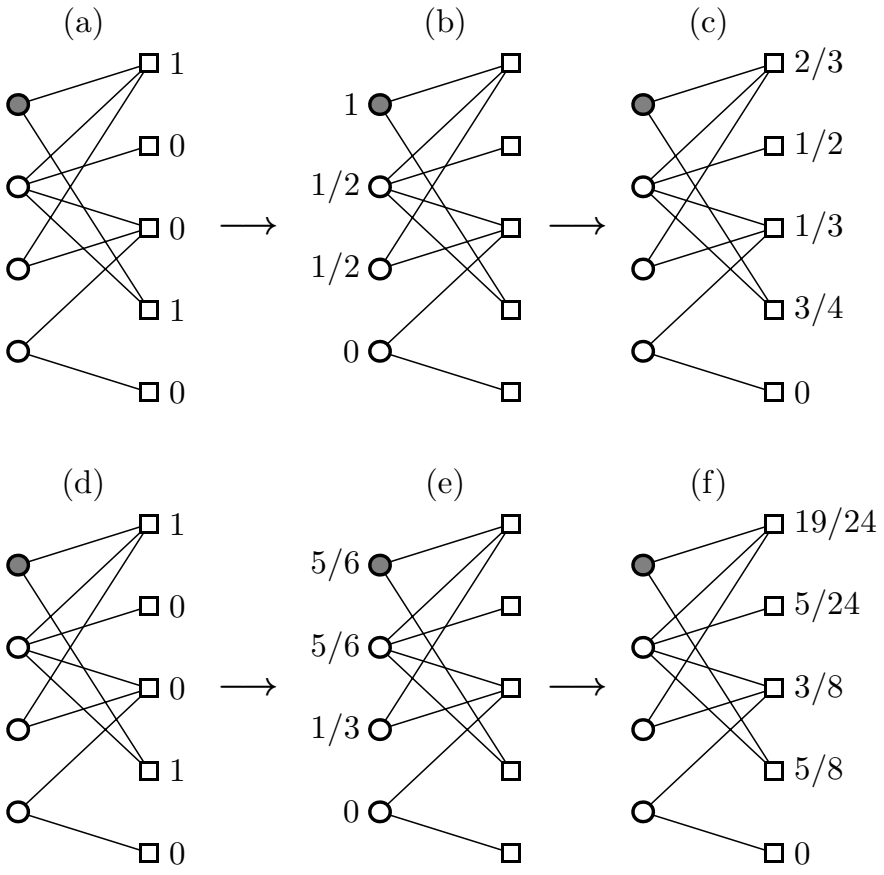


Figure 1.11: HeatS, which process is illustrated in *(a,b,c)*, redistributes resource via an averaging procedure where users receive a level of resource equal to the mean amount possessed by their neighboring items, and objects then receiving back the mean of their neighboring users' resource levels. ProbS, in *(d,e,f)*, evenly distributes among neighboring users the initial resource and then evenly redistributes it back to those users' neighboring items. Illustration taken from Zhou et al. (2010).

1.3.4 Other approaches

For the interested reader, we mention here a few more approaches that caught our attention but that were not tried extensively in our setting and may be worth further investigations.

For now, we mainly derived algorithms that can either be classified under the supervised or unsupervised learning paradigms (Murphy, 2012, Chapter 1). The problem of recommendation can, however, also be stated under the *reinforcement learning* paradigm, by considering agents receiving rewards for making good recommendations. To that extent, contextual-bandits for recommendation have received some attention (Li et al., 2010; Wang et al., 2016). Zheng et al. (2018) adapted the Deep Q-Learning algorithm (Mnih et al., 2013) to news recommendation. These approaches are appealing in the sense that they are able to adapt to the current needs of the users directly. However, these algorithms need to interact with the final user to be used directly, and this cannot be done in our business setup.

From now on, we used methods that extract information from the bipartite graph structure of the model, but some recommender systems directly use this structure to make predictions. Eksombatchai et al. (2018) use random walks on a weighted bipartite graph to perform recommendations in real-time and is used in production at Pinterest. Wang et al. (2019) use

graph neural networks to enhance classic matrix factorization strategies with the information of higher-order connectivity from the bipartite graph.

1.3.5 The cold-start problem

Cold-start refers to the fact that one needs to gather enough information about a new user or item to include it in the predictions of a recommender system (Bobadilla et al., 2013). Notably, in matrix factorization-related approaches, the set of users and items on which predictions can be made is determined by the sets on which the adjacency matrix was built, and cannot be changed unless retraining the algorithm.

In most cases, recommender systems suffer from the *user cold-start* — an e-commerce website is not able to onboard a new client in its recommender system before gathering enough information about its shopping patterns. In the CIB business, however, onboarding new clients is not that common — the financial setting suffers more from *item cold-start*. As a matter of fact, financial products such as bonds and options have a limited lifespan: new products are constantly emitted to replace expiring ones. For a matrix factorization-like algorithm, however, these two products are distinct and could not be assimilated. Bobadilla et al. (2013) provide multiple ways in which user and item cold-start can be tackled. We also provide in the next chapters ways to battle it in general, and more especially in the case of item cold-start.

1.4 Evaluating financial recommender systems

Evaluating recommender systems can be done in many ways, depending on how the problem is framed. We describe below the set of metrics we use for the remainder of this thesis.

1.4.1 Deriving symmetrized mean average precision

The problem of retrieving probabilities of interest $\hat{p}$ introduced in Section 1.3.2 can be understood as a classification task - our target is a discrete value in $\{0, \dots, C-1\}$ with C the number of classes that we consider. If we are only interested in the binary interest of our clients, $C = 2$; if we want to introduce buy and/or sell interests, we use $C = 4$ to represent no interest, buy interest, sell interest and interest into buying *and* selling the considered asset in the considered window of time.

In such a classification setting, metrics based on the *confusion matrix*, see Fig. 1.12, are used to assess the performance of algorithms. In a binary setting, denoting the prediction of the monitored algorithm for a particular sample $\hat{y}$, the confusion matrix compares the predicted $\hat{y}$ with the true labels y, distinguishing between four different cases.

		True class	
		0	1
Predicted class	0	True Negatives (TN)	False Negatives (FN)
	1	False Positives (FP)	True Positives (TP)

Figure 1.12: The confusion matrix outline.

This confusion matrix can then be summed up through a collection of metrics such as *precision*, which corresponds to the proportion of correct predictions that were made, *recall*, also called *true positive rate*, which corresponds to the proportion of the real targets that were retrieved, *false positive rate*, which corresponds to the proportion of negative events predicted as positive, i.e., the proportion of "false alarms"... Formulas for these quantities are given in Table 1.1.

Table 1.1: Examples of metrics describing the confusion matrix.

Precision (P)	$\frac{TP}{TP+FP}$
Recall (R or TPR)	$\frac{TP}{TP+FN}$
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$
True Negative Rate (TNR)	$\frac{TN}{FP+TN}$
False Discovery Rate (FDR)	$\frac{FP}{FP+TP}$

However, a comparison criterion must be unique: one algorithm could beat its competitors on a given metric but not on the others, leading to an ambiguous choice of preferred algorithm. As we would like to compare our algorithms with a combination of confusion matrix-based metrics nonetheless, we have to rely on summary metrics. The most common ones are F-scores, defined as

$$F_\beta = (1+\beta^2) \cdot \frac{P \cdot R}{(\beta^2 \cdot P) + R}. \tag{1.14}$$

F_1 score is the most common one and corresponds to the harmonic mean between precision and recall. Less classic options include F_2 scores, which weigh more recall than precision, and $F_{0.5}$ score which weigh more precision than recall. Consequently, the choice of β entirely depends on the use case we want to assess.

We examined here the case of binary classification tasks. In a multiclass setting, it is usual to derive scores for each class in a one-versus-all setup, identical to a binary one, and average found scores using a given strategy. The most common averaging strategies are *micro-* and *macro-*averaging. For classes $c \in \{0, \dots, C-1\}$, macro-averaging is defined as, with the example of precision P,

$$P_{macro} = \frac{1}{C} \sum_c P_c, \tag{1.15}$$

where P_c is the precision of class c obtained considering the one-versus-all setup. Micro-averaging is defined as, with the example of precision,

$$P_{micro} = \frac{\sum_c \mathrm{TP}_c}{\sum_c \mathrm{TP}_c + \sum_c \mathrm{FP}_c}. \tag{1.16}$$

As appears in these formulas, macro-averaging treats all classes equally, whereas micro-averaging takes into consideration the relative weight of all classes. In the financial setup, it is often the case that the considered classes are *ill-balanced*, i.e., that the distribution of classes among samples is not uniform at all. In our future interest prediction setting, it appears that a client will most of the time not be interested in a given product, leading to an over-representation of the "no interest" class, i.e., to severe imbalance. The "positive interest" class is the one that truly matters: its performance must consequently not be drowned under the performance of the "no interest" one. For that reason, when required, we choose macro-averaging procedures to give equal weights to all considered classes.

However, most algorithms do not directly lead to the class prediction $\hat{y}$. In a binary setting, classification algorithms learn the probability of the positive class $\hat{p} = p_\theta(y = 1|x)$, and we have to introduce a threshold τ such that $\hat{y} = 1$ if $\hat{p} > \tau$ and 0 else: the above metrics are

consequently highly dependent on this threshold τ. A better summary metrics strategy to avoid this dependence is to look at *area under curves*. In binary classification tasks, the **area under the receiver operating characteristic curve**, abbreviated as AUC ROC or AUC for short, is often used. The ROC curve is defined as $f(\text{FPR}) = \text{TPR}$, where each point corresponds to a given threshold τ. The AUC has a remarkable statistical property (Fawcett, 2006):

Proposition 1.4.1. *The AUC score is equivalent to the probability that the scored algorithm attributes a higher score to a randomly chosen positive sample than to a randomly chosen negative one.*

Proof. Let us call X_1 a random variable describing the scores of true positives with distribution p_1, X_0 a random variable describing the scores of true negatives with distribution p_0 and assume their independence. AUC score is defined as the area under the TPR-FPR curve. Considering scores in $\mathbb{R}$ — if we use as scores the probability of the positive class $p_\theta(y = 1|x)$, this range is restricted to $[0; 1]$ —, we can write AUC as the following:

$$AUC = \int_{+\infty}^{-\infty} P(X_1 > \tau) dP(X_0 > \tau), \tag{1.17}$$

where $P(X_1 > \tau)$ and $P(X_0 > \tau)$ respectively correspond to TPR and FPR, and τ is the threshold. Note that the integral is defined from $+\infty$ to $-\infty$: as FPR corresponds to the x-axis, we scan thresholds from highest to lowest — $P(X_0 > +\infty) = 0$ corresponds to the left-hand side of the x-axis.

Using p_1 and p_0, we get:

$$AUC = \int_{+\infty}^{-\infty} \int_{\tau}^{+\infty} p_1(x) dx (-p_0(\tau) d\tau) \tag{1.18}$$

$$AUC = \int_{-\infty}^{+\infty} \int_{\tau}^{+\infty} p_1(x) dx p_0(\tau) d\tau \tag{1.19}$$

But the last integral can be re-written $AUC = P(X_1 > X_0)$, hence the property. □

From this property, we can see that AUC scores of 0.5 correspond to random classifiers, and a good classifier consequently has an AUC greater than 0.5. In cases of severe imbalance, however, it appears that AUC ROC is ill-suited to monitor performance. As this metric uses FPR, in case of severe imbalance $FP \gg TN$ — a small variation in the number of false positives will therefore not impact the FPR, consequently biasing AUC towards high scores. This bias leads to the wrong belief that the monitored model is performing well in cases of severe imbalance. The **average precision score** (AP) corrects this issue. AP corresponds to the area under the $f(R) = P$ curve, as known as the precision-recall curve. An extensive study of the relationship between AP and AUC ROC can be found in (Davis and Goadrich, 2006), and an intuitive comparison of AUC ROC and AP can be found in Section 1.4.2. AP score can also be interpreted as the average of the precisions of top-L predictions, noted P@L, for all L corresponding to the ranks of true events in the full recommendation list. For both these metrics, the closer the score to 1, the better the algorithm.

However, recommender systems are not usual classification tasks, and we need to tailor these metrics to correspond more closely to our ultimate goal of delivering every day relevant recommendations to our clients. Recommender systems can be thought of as information retrieval systems: a recommender system provides ranked lists of potentially interesting items to a user,

who willingly — or not — queried it. An overview of useful metrics for information retrieval can be found in (Manning et al., 2010). We stick here with AP for its above-cited properties, using a slight variation of it. Instead of AP, we monitor the performance of our models using **mean average precision** (mAP), defined as

$$\text{mAP} = \frac{1}{|Q|} \sum_{q \in Q} AP(q), \tag{1.20}$$

where q is a *query*. We define queries in two ways:

- **User-side.** A query corresponds to the recommendation list of a given *user* at a given date.
- **Item-side.** A query corresponds to the recommendation list of a given *item* at a given date.

The item-side query is introduced to obtain the item perspective usually not monitored in recommender systems. This perspective is particularly relevant for corporate banks — to control her risk, a market maker needs to control her positions, i.e., make sure she does not hold a given position for too long. It is consequently of strategic interest to find relevant clients for a given asset, and item queries allow for monitoring the corresponding performance. Moreover, salespeople usually monitor the provided recommendations by looking at the predicted scores for a particular client or asset — we consequently need our models to provide useful rankings at the client and asset scales, and that is what user- and item-side mAP scores evaluate. Note that mean AUC scores can be defined in the same way, as

$$\text{mAUC} = \frac{1}{|Q|} \sum_{q \in Q} AUC(q). \tag{1.21}$$

Finally, as we want to describe the performance of our algorithms with one score only, we introduce the **symmetrized mAP**, defined as the harmonic mean between the user- and item-side mAP. From now on, this score is used to monitor the performance of our algorithms — when applicable. As seen in Section 1.3.2, content-filtering approaches may rely on cross-features for which we might lack data, preventing us from computing them at any time for all couples. In such cases, we favor using AP on the overall task instead.

1.4.2 A closer look at area under curve metrics

To understand the behavioral differences between AUC and AP, we study a couple of examples to get a better intuition at how they score particular recommendations. We consider here recommendation lists of size 10000 to mimic our actual user-side setting, that we score with both AUC and AP.

Consider a user having two interests at a given date, and first examine a case where these interests were ranked 10-th and 100-th.

- Following the probabilistic interpretation of AUC, in that case we can compute an AUC of $0.5 * ((1 - \frac{9}{9999}) + (1 - \frac{98}{9998})) \approx 0.995$. The score consequently highlights what would be an excellent recommendation.
- Using the P@L interpretation of AP, we compute a score of $0.5 * (1/10 + 2/100) \approx 0.06$. The score consequently highlights what would be a very poor recommendation.

Now, consider the extreme case where the two interests are ranked first and last. We get the following:

- With the probabilistic interpretation, we compute an AUC of $0.5 * (1 + (1 - 1)) = 0.5$. It corresponds to the score of a random recommendation.
- With the P@L interpretation, we compute an AP of $0.5 * (1 + 2/10000) = 0.5001$, the score of a quite good recommendation.

From these examples, we see that AUC scores do not differentiate predictions relatively to their position in the recommendation list. This behavior leads to a good score in the first example, and a bad one in the second. A recommender system with good AUC is consequently a recommender that is good on a broad scale. Contrarily, AP weighs heavier the top of the recommendation list due to its usage of P@L. It follows that in the first example, the AP score is quite low due to the first true event appearing in the 10-th position only. But in the second example, the relatively good AP score is entirely driven by the fact that the first prediction was indeed correct.

Consequently, AP is more adapted in cases where we matter most about the first recommendations that the system does. In our business case, that is the reason why we favor AP: considering that recommendation lists are sales' calls plannings, a salesperson needs to get good returns on her calls fast, as their time (and patience) is limited. There are no correct metrics to use in the absolute — there are metrics more adapted to a given situation.

1.4.3 Monitoring diversity

Qualitative metrics can also be used to monitor our recommender systems. The precision of recommender systems is, of course, of great importance. Still, a system only targeting precision might fall into the trap of always recommending the same items, thereby a self-sustained loop — a client who is only recommended already seen items keeps treating them with us, and maybe finds another counterparty for other items that may be of interest for her. Moreover, we also want to recommend items in the long tail (see Section 1.2.2). For these reasons, monitoring the *diversity* of our systems following the ideas introduced in Zhou et al. (2010) and continued in Zeng et al. (2014) is also helpful.

We introduce two qualitative metrics called **diversity** and **similarity**. Diversity $D(L)$ is defined as

$$D(L) = \frac{1}{L} \sum_{i \in I} \sum_{u \in U} k_i \mathbb{1}_{i \in R_{1:L,u}} \tag{1.22}$$

with $R_{1:L,u}$ the top-L ranked items for user u and k_i the degree of item i as computed on the training dataset, the degree corresponding to the number of unique users connected to item i in the bipartite graph corresponding to the training dataset. $D(L)$ corresponds to the average degree of an item present in the top-L recommendation lists of all our users at a given date — the lower the metric, the more diverse the recommendations. Similarity $S(L)$ is defined as

$$S(L) = \frac{2}{|U|(|U| - 1)} \sum_{u > v} \frac{C_{uv}(L)}{L} \tag{1.23}$$

with $C_{uv}(L)$ the number of identical items in top-L recommendation lists of users u and v. $S(L)$ corresponds to the average number of identical items in the top-L recommendation lists for all pairs of users, at a given date. The lower the metric, the less similar the recommendation lists. L depends here on the asset class and has to be chosen in accordance with the users of the recommender system. Both of these metrics are introduced on the user-side and could be computed on the item-side as well. As the purpose of these metrics is only qualitative, it is here more relevant to stick with un-symmetrized values to keep their interpretation intact.

These metrics are monitored along with symmetrized mAP in our further experiments, when applicable. In cases of approximately equal symmetrized mAP performances, we favor algorithms obtaining better diversity and similarity scores.

1.5 The challenges of the financial world

In this chapter, we gave a relatively broad overview of the problem we want to solve, what data we have at our disposal to solve it and algorithms that can solve it. In doing so, we met challenges among which the heterogeneity of clients and assets (Section 1.2.2) and the item cold-start problem (Section 1.3.5). We explore here, haphazardly, additional challenges that we face in future financial interest prediction and that we will, partly, tackle in the remainder of this work.

We already mentioned in Section 1.1 that the RFQ information is imperfect. We only know a part of clients' interests, either the ones expressed on a platform where BNPP CIB operates or the ones told to salespeople. The final decision of the client is not always known, due to the rules of D2C platforms seen in Section 1.1.1. We have to keep this fact in mind when constructing our algorithms — what appears to us as no interest events could be real interests that we did not monitor. Moreover, the financial setup brings about two challenges: the *buy-sell asymmetry*, and the potentially *cyclic interests*. The asymmetry corresponds to the fact that a client can only sell a product she already possesses — such a piece of information can be obtained for some of BNP Paribas CIB clients from their portfolio snapshots, but does not concern the majority of them. The second challenge differentiates the financial setup from a classic e-commerce recommendation problem. In finance, buying a given product does not prevent at all from re-buying it in the future, a behavior that would be far rarer, e.g., for an Amazon customer. We consequently have to keep all of the past interests of a client in her potential future interests. There is even a tendency for a client to trade what she has already traded in the past: studying the corporate bonds RFQ database, in 2019, on average 28% of clients' RFQs are made on products they already traded before. If we restrict to the 500 most active clients of 2019, this figure rises to 58%, meaning that active clients have a high tendency to replicate their past behaviors.

Another challenge comes from the specificities of asset classes: there cannot be a single, cross-asset model. Each asset class has its own characteristics and features, as exposed in Section 1.1.2. We also observe different behaviors depending on the asset class — professional investors do not trade equity, options, and bonds in the same ways and have proper strategies for each of these asset classes. Moreover, in the same asset class, investors might even react differently to the same input signal: in equity, for instance, a price decrease does not mean the same for trend followers and mean-reverting traders. A global, cross-asset model is consequently not a good option, and we also need ways to adapt to the potentially many different trading strategies that investors have in each asset class, i.e., their behavioral heterogeneity.

Yet another challenge, usually disregarded in the recommender systems literature, is the importance of time, omnipresent in finance. Financial indicators continuously evolve, and with them the interests of clients: consequently, time must be taken into account to provide relevant recommendations. This evolution of interests with time is related to what is known as *concept drift* in the recommender systems literature. Tsymbal (2004) identifies concept drift as coming either from a change in the underlying data distribution or a change in the target concept. Drift can also be sudden or gradual. Financial time series are highly non-stationary — consequently, the primary source of concept drift we have to face is a gradual or abrupt change in the underlying data distribution. This challenge is, to the best of our knowledge, not addressed sufficiently by literature as it conceptually differs from the usual trends or seasonalities.

Additional challenges come from the way our future interest prediction problem is framed. A standard assumption in the recommender systems literature is that we can directly interact with the final users of the system. However, as seen in Section 1.1, the recipients of the provided recommendations are not directly the clients of the bank, but the salespeople with whom clients are engaged in business discussions. We consequently cannot resort to A/B testing strategies or bandit algorithms: sales take a risk when calling a client on behalf of the bank, and need to be able to trust the system. For a sales to follow a recommendation, predictions must be *understandable*. Giving insights into why recommendations are made helps sales gain confidence in the system, and provides them with ways to introduce the trade idea to their clients. This falls in the field of *interpretability*, where many different proposals have been made to try and understand the predictions made by black-box algorithms such as neural networks or gradient tree boosting algorithms. A popular method is LIME (Ribeiro et al., 2016), which learns an interpretable, linear model around the prediction to be interpreted. For neural networks, methods such as DeepLIFT (Shrikumar et al., 2017), which backpropagates the output to be interpreted to the input features, have been proposed. Lundberg and Lee (2017) propose to unify the field of interpretability using the game theory notion of Shapley values (Shapley, 1953). Finally, Kim et al. (2017) propose to interpret outputs using user-defined concepts instead of input features, which may not always be meaningful to a non-specialist.

On a more practical note, our predictions must also be *reliable* and *user-friendly*. Scores must be good indicators of the confidence we have that the interest might happen so as not to lose time on unlikely interests. Moreover, scores must also be easily readable by our users: for instance, if we take as score the probability of interest in a given product, it will most often be very low — a more meaningful score here could be the daily variation of predicted probability. A scoring scale, e.g., between 0 and 5, could also be instituted following sales' wishes and needs. Consequently, we see that sales have to be fully integrated into the design of the recommender system's specifications to provide them with meaningful help and enhance their proactivity. Among these technical considerations, we also find legal aspects: as in most businesses, there are compliance challenges related to the design of such an algorithm. Thus, great care must be taken into the data sources we use, and the results we disclose to our clients.

Chapter 2

First results on real-world RFQ data

Chapter 1 introduces the future interest prediction task to a great extent and provides insights into suitable algorithms for solving this task. The goal of Chapter 2 is to further these insights through the application of classic recommender systems to the future interest prediction problem using a real-world RFQ dataset from BNP Paribas CIB. In this chapter, Section 2.1 reviews strategies allowing to handle multi-class outputs with models designed for binary ones and consequently bridge the gap between content-filtering and collaborative filtering algorithms, and Section 2.2 presents a benchmark of classic, seminal algorithms introduced in Section 1.3.

2.1 From binary to oriented interests

In Section 1.1, we introduced the problem of future interest prediction as predicting whether a client is interested in buying and/or selling a given financial asset at a given time. However, most collaborative filtering algorithms cannot handle *per se* a multi-class classification setting such as this one and only deal with binary interests. However, collaborative filtering and, more particularly, matrix factorization-like algorithms are particularly interesting in that they allow for *a posteriori* analysis of latent representations of users and items, a feature we do not retrieve in content-filtering strategies. Consequently, this section explores two ideas that allow bridging the gap between binary and multi-class prediction settings, enabling the usage of matrix factorization-like algorithms for the future interest prediction task.

A simple strategy for dealing with multi-class prediction is, when applicable, to split the prediction task into multiple tasks. As a matter of fact, our future interest prediction task is composed of two *nested* prediction tasks. The first one aims to find whether a client has an interest in a given product, and the second one seeks to retrieve the nature of this interest, i.e., whether it is an interest in buying and/or selling the considered product. Consequently, in such a setting, it is possible to use two algorithms to solve each of these tasks: a matrix factorization-like algorithm would deal with the interest/no interest prediction task, and a second algorithm with the interest's nature retrieval. A natural choice for this second algorithm would be a content-filtering one using appropriate, handcrafted features, trained on a cross-entropy objective. Another simple idea that deals with any multi-class prediction setting is to split the prediction task into as many tasks as there are classes, each task being a "one-versus-all" retrieval of a given class. A matrix factorization-like algorithm can then be used for each of these tasks.

However, the "one-versus-all" strategy does not exploit the potential correlations between the different classes of the prediction task. Exploiting these correlations can only be done with algorithms that directly address the multi-class framing of the prediction problem. We introduced in Section 1.3.3 the tensor factorization idea for tackling temporal contexts. The same idea

can be applied to multi-class classification, by considering three-dimensional adjacency tensors $A \in \mathbb{R}^{|U|\times|I|\times C}$, with C the number of classes. Using a tensor factorization model using latent representations of users, items and classes respectively denoted $P \in \mathbb{R}^{|U|\times d}$, $Q \in \mathbb{R}^{|I|\times d}$ and $K \in \mathbb{R}^{C\times d}$ with d the latent dimension, the score of the c class of a given (u, i) couple is given by

$$x_{ui}^{c} = \sum_{l=1}^{d} p_{u,l} q_{i,l} k_{c,l}. \tag{2.1}$$

This algorithm can then be trained using a cross-entropy entropy objective, as

$$l = \sum_{c} y_{ui}^{c} \log \sigma \left(x_{ui}^{c} \right), \tag{2.2}$$

with $y_{ui}^{c} := a_{u,i,c}$ and σ the sigmoid function.

In this thesis, we focus on the development of *ad hoc* algorithmic designs aimed at solving specific challenges of the financial sector, as will be seen in Chapters 3 and 4. Consequently, the binary-to-multi-class problem considered in this section appears as a technicality, and will not be evoked further. From now on, we use the most natural setting for the considered algorithms, be it binary or multi-class.

2.2 A benchmark of classic algorithms

This section presents a benchmark of algorithms presented in Section 1.3 on a G10 bonds RFQ dataset from BNP Paribas CIB which is more extensively studied in Chapter 4. Our dataset comprises hundreds of clients and thousands of bonds and ranges from 01/08/2018 to 30/09/2019. We consider as training set the events from 01/08/2018 to 31/07/2019, as validation set the events from 01/08/2019 to 31/08/2019 and as test set the events from 01/09/2019 to 30/09/2019. First, we examine the methodology and hyperparameters considered for algorithms of four different model classes. Second, we present, compare, and analyze the results obtained with these algorithms on all the metrics introduced in Section 1.4. All models are scored on a perimeter composed of all the couples $U \times I = \{(u, i)|u \in U, i \in I\}$, with U (resp. I) the set of all users (resp. items) observed in the training set, for all the dates of the scored split of data. Qualitative metrics were all computed using $L = 20$, where L corresponds to the number of top recommended items or users, depending on the chosen side (see Section 1.4.3).

2.2.1 Historical models

We first consider historical models, as introduced in Section 1.3.1. In this benchmark, we use two historical models: a naive one that we note "Histo", and a temporal one that we note "λ-Histo". λ-Histo differs from the Histo model in that events are penalized according to their observation date: the further away in the past the observation of a (u, i) couple, the lower its weight in the score of the (u, i) couple.

The scores are computed using the training set only, using for λ-Histo a temporal penalty inspired from the one introduced in Section 1.3.1, defined as $e^{-\lambda(T-t)}$, where T is the final day of the training set and t is the observation day of the event. These scores are then re-used to perform recommendations on all dates of the validation and test sets.

Figure 2.1 shows the evolution of the validation performance, defined by the symmetrized mAP score, with the λ parameter. This parameter can be interpreted in terms of *half-life* — the range of values considered corresponds to half-lives ranging from roughly two years to a day. The best validation performance is obtained for a value of $\lambda \approx 0.02$, corresponding to a half-life

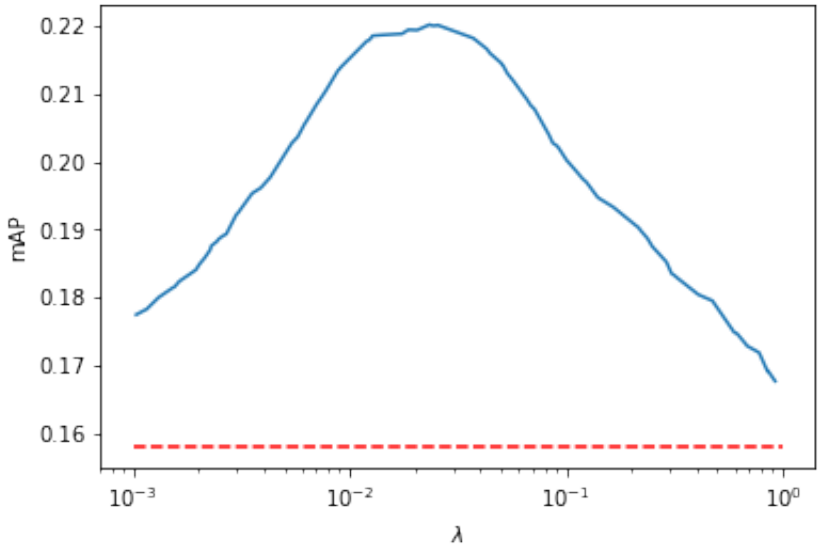


Figure 2.1: Evolution of validation symmetrized mAP score with λ. The optimal value is found for $\lambda \approx 0.02$, corresponding to a half-life of roughly a month. The horizontal, red dotted line corresponds to the performance of the unpenalized Histo model.

of roughly a month. Interestingly, on the range of λ values considered, all instances of λ-Histo outperform the classic Histo model, showing the superiority of temporal approaches.

2.2.2 Resource redistribution models

We now focus on the resource redistribution models, as seen in Section 1.3.3, and more particularly on the hybrid SPHY model introduced in Zeng et al. (2014). This algorithm performs recommendations through transformations of the adjacency matrix into user similarity matrices and is controlled by two hyperparameters — λ, that controls the hybridization between models aiming at diversity and precision of recommendations, and θ, that controls the reliance on similar users. $\lambda = 0$ corresponds to the model aiming at diversity of recommendations, and $\lambda = 1$ to the model aiming at precision of recommendations. Using $\theta > 1$ enhances the reliance on similar users for the recommendations of a particular user, and using $\theta < 1$ lowers that reliance. We search for the optimal λ and θ values with an exhaustive grid search, considering λ values in the $[0; 1]$ range and θ values in the $[0.1; 3]$ range.

Figure 2.2 illustrates the validation symmetrized mAP performance of an exhaustive grid of λ and θ values in a heatmap. We see that the best performances are found in the bottom-right corner of the heatmap, i.e., for $\lambda > 0.5$ and $\theta > 1$. The optimal value is found for $\lambda = 0.85$ and $\theta = 2.2$. Best performances are found for λ values that favor the model targeting precision of recommendations in the hybridization, corresponding to the model's expected behavior. Interestingly, the optimal performance is obtained for a hybrid model, with $\lambda = 0.85$: introducing a moderate amount of diversity in the provided recommendations consequently enhances model performances. Best performances are also found for θ values above 1, i.e., for models that give more weight to similar users in the scoring process.

Figure 2.3 illustrates the validation diversity metric performance on the same exhaustive grid. Diversity, defined in Section 1.4, corresponds to the mean degree of top-20 recommended items on the user-side(Fig. 2.3**(a)**), and to the mean degree of top-20 recommended users on the item-side (Fig. 2.3**(b)**). A low diversity value consequently corresponds to more diverse recommendation lists: a low mean degree means that the recommendation lists comprise less popular

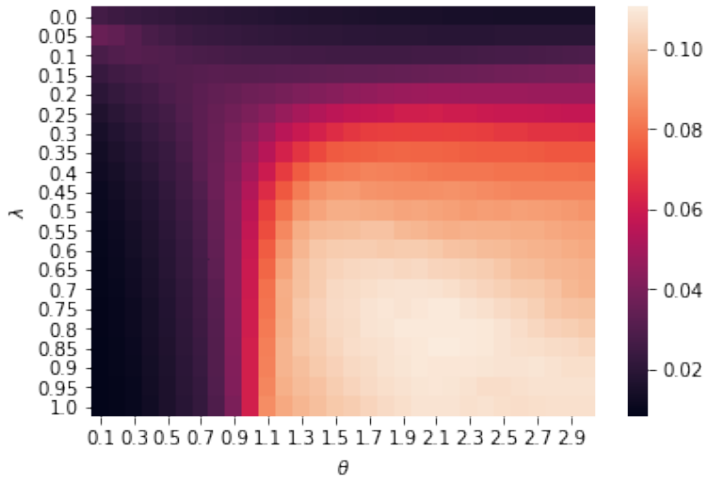
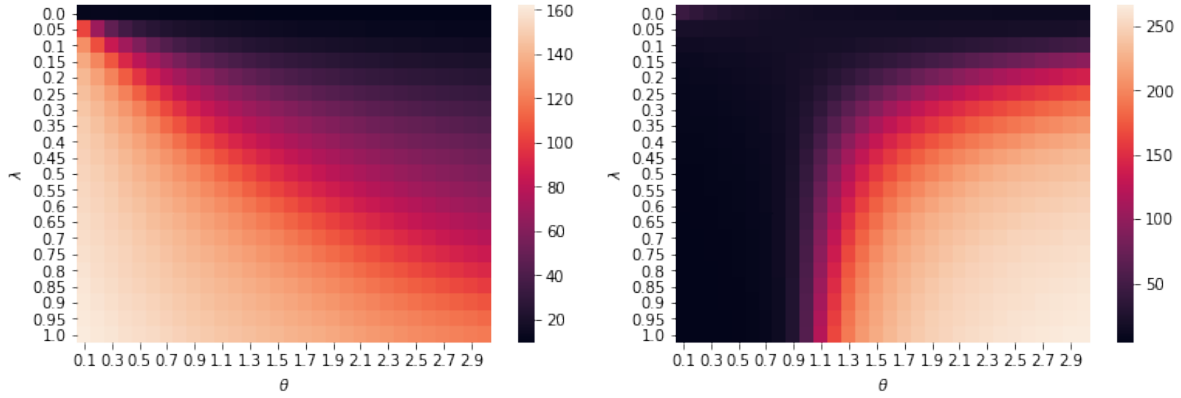


Figure 2.2: Heatmap visualization of the evolution of validation symmetrized mAP score with the λ and θ parameters of the SPHY model. The optimal value is found for $\lambda = 0.85$ and $\theta = 2.2$.

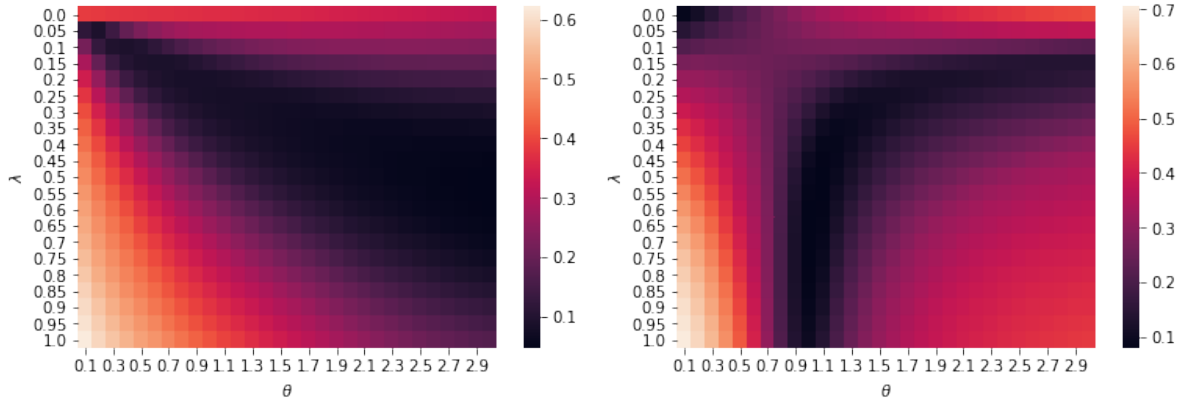


(a) **(b)**

Figure 2.3: Heatmap visualization of the evolution of validation diversity metrics with the λ and θ parameters of the SPHY model, computed using top-L lists with $L = 20$. **(a)** Validation user-side diversity. **(b)** Validation item-side diversity.

items or users. Interestingly, the two graphs have particularly distinct profiles. In Fig. 2.3**(a)**, we see that the most diverse models in terms of recommended items are obtained for low λ values, corresponding to the expected behavior of the model. In Fig. 2.3**(b)**, we see that the most diverse models in terms of recommended users are obtained for low λ values, as expected, but also for low θ values, i.e., for $\theta < 1$. Indeed, models with $\theta < 1$ lower the weight of similar users in the scoring process, leading to less differentiated users, and consequently more diversity in the users recommended for a given item.

Finally, Fig. 2.4 illustrates the validation similarity metric performance on the same exhaustive grid. Similarity, also defined in Section 1.4, corresponds to the mean overlap of top-20 recommendation lists of users on the user-side, illustrated in Fig. 2.4**(a)**, and of items on the item-side, illustrated in Fig. 2.4**(b)**. We also see two distinct profiles here. In Fig. 2.4**(a)**, we see that the most dissimilar recommendation lists are obtained for λ values in the $[0.05; 0.7]$ range, evolving accordingly with the θ values: as the hybrid model tends more and more towards precision of recommendations, a higher reliance on similar users is required to get dissimilar



(a) **(b)**

Figure 2.4: Heatmap visualization of the evolution of validation similarity metrics with the λ and θ parameters of the SPHY model, computed using top-L lists with $L = 20$. **(a)** Validation user-side similarity. **(b)** Validation item-side similarity.

recommendation lists for all users. Indeed, polarizing users around their most similar peers appear to specialize their recommendation lists, and consequently lead to dissimilar lists overall. In Fig. 2.4**(b)**, we see that the most dissimilar items' recommendation lists are obtained either for low values of λ and θ or, more interestingly — and counterintuitively —, for values of $\theta \approx 1$ and $\lambda > 0.5$. Slightly less dissimilar lists are also obtained with values of $\lambda \approx 0.2$ and $\theta > 2$, a behavior already encountered on the user-side in Fig. 2.4**(a)**.

2.2.3 Matrix factorization models

We examine in this benchmark two matrix factorization approaches based on the implicit feedback and BPR objectives, introduced in Section 1.3.3, that we respectively call *MF - Implicit* and *MF - BPR*. For both models, hyperparameters are found using a combination of a hundred trials of random search and hand fine-tuning. The MF - Implicit algorithm with best validation symmetrized mAP performance used an embedding size $d = 512$, batches of size 256, a L_2 penalty weight of $4e^{-7}$, a learning rate of $7e^{-4}$ and a logarithmic confidence with parameters $\alpha = 4$ and $\epsilon = 10$. The confidence weights are illustrated in Fig. 2.5, which shows the evolution of weights with the number of occurrences r — in the considered dataset, the maximal value of r is 223, and confidence weights are consequently found in the $[1; 14]$ range.

The MF - BPR algorithm with best validation symmetrized mAP performance used embeddings of size $d = 1024$, batches of size 256, a L_2 penalty weight of $2e^{-7}$ and a learning rate of $9e^{-5}$. Both algorithms were trained with the Adam optimizer (Kingma and Ba, 2015), with early stopping and a patience of 50.

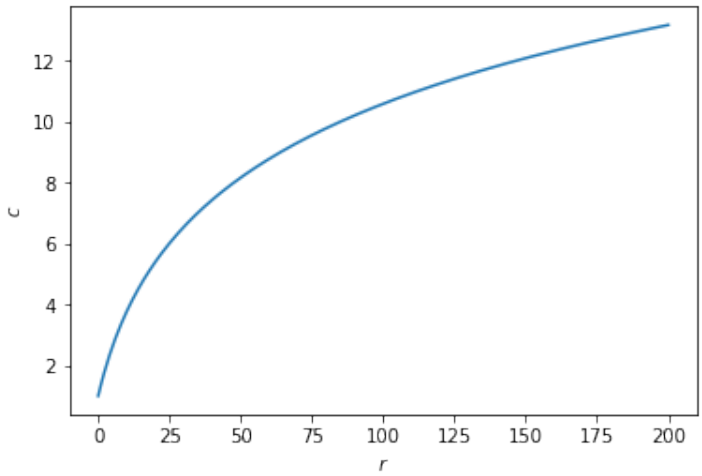


Figure 2.5: Evolution of confidence weights c with the number of occurrences of a given couple r, with a logarithmic weighing using $\alpha = 4$, $\epsilon = 10$.

2.2.4 Gradient tree-boosting models

Finally, we also include in this benchmark a content-filtering approach based on a gradient tree boosting model implementation, LightGBM (Ke et al., 2017). To train this algorithm, we rely on handcrafted, proprietary features describing the state of a given *(date, user, item)* triplet and that can be split into three groups:

- **Activity features.** These features correspond to frequencies of activity, aggregated at various levels: activity of the considered user, of the considered user on the considered item, of the considered item, of the sector of activity of the considered user,... computed with running means of various window sizes.
- **Market features.** These features correspond to the market data related to the considered item, e.g., its price, yield, spread,... computed with running means of various window sizes and transformed into z-scores.
- **Categorical features.** These features correspond to categorical, static information related to the considered triplet, e.g., the sector of activity of the considered user, the day of the week, the issuer of the item,...

However, activity features describing the activity of a particular *(user, item)* couple require to observe events for that couple to be computed. Consequently, for this experiment, we regard as negative events at a given date all the *(user, item)* couples for which we observed at least one positive event in the past six months and that are not positive at this date. To manage bonds' illiquidity, we increase the proportion of positive events in this perimeter by considering that positive *(date, user, item)* triplets remain positive for the following four business days. It follows that the *(date, user, item)* perimeter used to train our LightGBM instances is only composed of historical events. Consequently, this algorithm cannot perform predictions for *(user, item)* couples that were not observed in the recent past at a given date, as activity features cannot be computed for such couples. The LightGBM algorithm is trained on this dataset with a binary cross-entropy objective.

To compare LightGBM instances to the collaborative filtering approaches presented above, we keep as scoring perimeter the couples formed by all users and items observed during the training period on all the dates of the split to be scored. If a *(date, user, item)* triplet is present in the LightGBM dataset, it is attributed the score corresponding to the predicted probability of positive event; if not, it is attributed a score of 0. This trick allows for comparing all our models,

but biases LightGBM average precision scores towards higher values. This can be explained by the fact that couples obtaining a 0 score with this methodology are, overall, less likely to be positive, and composed of less popular users and items. Consequently, for a given user (resp. item), historical items (resp. users) will appear on top of the recommendation lists, leading to overall better average precision scores. Consequently, LightGBM results presented in Section 2.2.5 should be taken with a grain of salt.

Hyperparameters for this algorithm are optimized using a combination of a hundred trials of random search and hand-fine tuning. The algorithm with the best validation symmetrized mAP performance used trees with a maximal depth of 20, a maximal number of leaves of 256 and a minimum of 100 samples per leaf, a maximum of 500 trees, trained with a learning rate of 0.02, a subsampling of 90% of the training data every 5 trees and early stopped with a patience of 50.

2.2.5 Overall benchmark

We present in this section the results obtained for all models considered in this benchmark on all the metrics introduced in Section 1.4. Table 2.1 shows the symmetrized mAP scores on validation and test sets for all models. We see that the best model in terms of symmetrized mAP is LightGBM, as expected from the bias introduced in the scoring methodology and outlined in Section 2.2.4. Interestingly, the second-best algorithm is the λ-Histo model — including a temporal penalty in a basic historical model consequently provides a strong baseline model.

Table 2.1: Comparison of classic algorithms on symmetrized mAP scores, computed on the validation and test sets. Metrics expressed in percentage.

Algorithm	Valid mAP	Test mAP
Histo	15.80	13.33
λ-Histo	22.01	17.13
SPHY	11.06	9.94
MF - Implicit	13.87	11.61
MF - BPR	16.31	14.01
LightGBM	**31.42**	**29.52**

Let us now examine the overall AUC and AP metrics, and the decomposition of test symmetrized mAP into user- and item-side mAP scores, as seen in Table 2.2. Expectedly, the best model in terms of overall AUC is the MF - BPR algorithm, as this model is trained on a relaxation of the AUC metric. We also remark that historical models rank higher among models in terms of AP than AUC — historical couples are more likely to appear on top of recommendation lists as investors tend to replicate a consequent portion of their trades, as seen in Section 1.5. As AP puts a heavier weight on the top of recommendation lists, as explained in Section 1.4.2, securing one good recommendation on the top of the list ensures strong AP performances. The same phenomenon is exacerbated in the LightGBM experiment due to its restricted scoring perimeter. Moreover, user-side mAP scores are always lower than item-side ones. Therefore, for the RFQ prediction problem, ranking clients for financial assets is an easier task than ranking assets for clients.

Finally, Table 2.3 presents the qualitative metrics results of all the considered algorithms. The most diverse algorithm appears to be LightGBM, closely followed by λ-Histo. Interestingly, LightGBM provided very similar recommendation lists on both the user and item sides — the construction of the LightGBM dataset results in restricted pools of scored users and items but does not restrain from recommending rather "novel" users and items. A surprising result is that the most dissimilar recommendation lists were obtained with the λ-Histo model, closely followed

Table 2.2: Comparison of classic algorithms on overall AUC and Average Precision scores, and user- and item-side mAP scores. Metrics computed on the test set, and expressed in percentage.

Algorithm	Overall AUC	Overall AP	User-side mAP	Item-side mAP
Histo	84.80	3.72	10.38	18.61
λ-Histo	85.0	4.98	14.69	20.85
SPHY	90.46	1.61	7.42	15.06
MF - Implicit	91.61	2.12	9.29	15.46
MF - BPR	**92.81**	2.78	10.86	19.70
LightGBM	74.90	**14.99**	**26.63**	**33.12**

by the Histo model. Consequently, it appears that users, and reciprocally items, "specialize" on a given set of products, and that item popularity is a relative notion among users.

Table 2.3: Comparison of classic algorithms on qualitative metrics — diversity (D) and similarity (S) metrics obtained on the test set. Degrees taken into account for diversities are computed on the training set. Metrics computed using $L = 20$.

Algorithm	D_{user}	D_{item}	S_{user}	S_{item}
Histo	91.75	183.28	8.28%	16.24%
λ-Histo	85.29	166.94	**7.37%**	**13.06%**
SPHY	117.86	250.86	16.95%	38.95%
MF - Implicit	105.01	212.91	10.35%	22.07%
MF - BPR	100.24	183.89	9.76%	16.37%
LightGBM	**81.10**	**161.44**	69.93%	78.92%

2.3 Concluding remarks

The benchmark presented in this chapter makes it possible to better understand the different models and metrics introduced in Chapter 1, and provides further insights into the challenges of future interest predictions. Notably, the benchmark shows that beating a good baseline is a hard task. In this benchmark's experiment, the simple Histo and λ-Histo models obtain very competitive performances on almost all considered metrics, and λ-Histo is the second-best model as determined by validation mAP performance, beating all the collaborative filtering approaches considered. The low performance of collaborative filtering approaches can be attributed to the non-stationarity of our data, compensated for with the temporal penalty in the λ-Histo model. An in-depth study of this phenomenon and how collaborative filtering approaches can be adapted to tackle it is conducted in Chapter 4.

Moreover, we also encountered the relative difficulty of comparing models using different scoring approaches, such as LightGBM and the collaborative filtering algorithms considered in this benchmark. As these two approaches do not natively use the same scoring perimeters, reconciling them is inevitably done at the detriment of one of them. A competitor to LightGBM, using the same scoring methodology, is introduced in Chapter 3.

Chapter 3

A supervised clustering approach to future interests prediction

The work exposed in this chapter has been orally presented at the 24^{th} Workshop of Economics with Heterogeneous Interacting Agents and was selected for an oral presentation at the 13^{th} Financial Risks International Forum organized by the Institut Louis Bachelier. It has been selected for publication in the Algorithmic Finance journal under the title "Deep Prediction of Investor Interest: a Supervised Clustering Approach".

This chapter proposes a neural network architecture for the future interest prediction problem that performs both investor clustering and modeling at the same time, thereby introducing a *supervised clustering* approach. The relevance of this architecture for the clustering and modeling tasks is controlled in an experiment on a synthetic scenario, and the performance of the algorithm is monitored on two real-world databases, a publicly available dataset about the position of investors in the Spanish stock market and a proprietary RFQ dataset from BNP Paribas Corporate and Institutional Banking already presented in Section 1.2.

In this chapter, we examine the heterogeneity of investors and introduce a mathematical modeling of this heterogeneity (see Section 3.1), we introduce the above-mentioned neural network architecture designed to tackle heterogeneous clusters of investors (see Section 3.2), we experiment this architecture on the synthetic and real-world datasets (see Section 3.3), we present the path of ideas that led to the introduced design (see Section 3.4) and we elaborate on ways to extend the current version (see Section 3.5).

3.1 Heterogeneous clusters of investors

Predicting future investor activity is a challenging problem in finance, as already stressed in Chapter 1. Two of the challenges that make this problem particularly difficult are the substantial heterogeneity of both investors and assets (see Section 1.2), compounded by the non-stationary nature of markets and investors and the limited time over which predictions are relevant. This chapter explicitly tackles the first difficulty; the second one will be further explored in Chapter 4.

3.1.1 Translating heterogeneity

The heterogeneity of investors translates into a heterogeneity of investment strategies (Tumminello et al., 2012; Musciotto et al., 2018): for the same set of information, e.g., financial and past activity indicators, investors can take totally different actions. Take for instance the case of an asset whose price has just decreased: some investors will buy it because they have positive long-term price increase expectations and thus are happy to be able to buy this asset at a discount; reversely, some other investors will interpret the recent price decrease as indicative of the future trend or risk and refrain from buying it. We consequently expect a collection of heterogeneous behaviors to emerge in our RFQ datasets.

It has been shown that *ad hoc* methods are surprisingly efficient at clustering investors according to their trades in a single asset (Tumminello et al., 2012). Besides, clusters of investors determined for several assets separately have a substantial overlap (Baltakys et al., 2018), which shows that one may be able to cluster investors for more than a few assets at a time. The activity of a given cluster may also systematically depend on some clusters' previous activity, which can then be used to predict the investment flow of investors (Challet et al., 2018). Here, we leverage deep learning to train a single neural network on all the investors and all the assets of a given market and give temporal predictions for each investor, following the content filtering strategy exposed in Chapter 1, Section 1.3.2. Our goal is to leverage these clusters of investors, expected to follow distinct investment strategies, to enhance predictions of single investors.

Formally, in our setting, we call strategy a mapping f from current information x to the expression of interest to buy and/or sell a given asset, encoded by a categorical variable y: $f : x \mapsto y$. We call here

$$\mathcal{D} = \{f_k : x \mapsto y\}_k \tag{3.1}$$

the set of all the investment strategies that an investor may follow. Unsupervised clustering methods suggest that the number of different strategies that describe investors' decisions is finite (Musciotto et al., 2018). We therefore expect our dataset to have a finite number K of clusters of investors, each following a given investment strategy f_k. Consequently, we expect $\mathcal{D}$ to be such that $|\mathcal{D}| = K$, i.e.

$$\mathcal{D} = \{f_k : x \mapsto y\}_{k=1,\cdots,K}. \tag{3.2}$$

Alternatively, $\mathcal{D}$ can be thought of as the set of distinguishable strategies, which may be smaller than the total number of strategies and, therefore, be considered an effective set of strategies. Consequently, a suitable algorithm to meet our goal needs to infer the set of investment strategies $\mathcal{D}$.

3.1.2 Non-universality of investors

A simple experiment shows how investors differ. We first transform BNP Paribas CIB bonds' *Request for Quotation* (RFQ) database, along with market data and categorical information related to the investors and bonds, into a dataset of custom-made, proprietary features describing the investors' interactions with all the bonds under consideration, as seen in Section 2.2.4. This dataset is built so that each row can be mapped to a triplet *(date, investor, financial product)*. This structure allows us, for a given investor and at a given date, to provide probabilities of interest in buying and/or selling a given financial product in a given time frame. As the final decision following an RFQ is not always known, we consider the RFQ itself as the signal of interest in a product. Consequently, we consider a given day to be a *positive event* for a given investor and a given financial product on the day when the investor signaled his interest in that product and the following four business days. The reason is twofold: first because bonds are by essence illiquid financial products and second because this increases the proportion of positive events.

At each date, negative events are randomly sampled in the *(Investor, Financial Product)* pairs that were observed as positive events in the past and that are not positive at this date. Using this dataset, we conduct an experiment to illustrate the *non-universality* of investors, i.e., the fact that investors have distinct investment strategies. The methodology of this experiment is reminiscent of the one used in Sirignano and Cont (2019) to study the universality of equity limit order books.

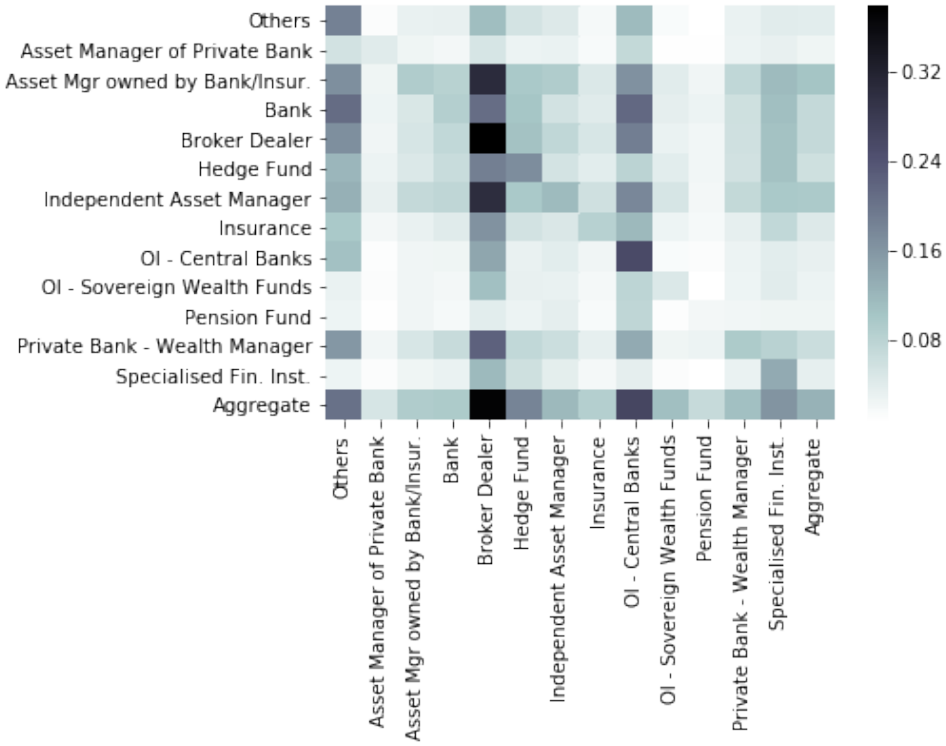


Figure 3.1: Universality matrix of investors' strategies: the y-axis shows investors' sector used to train a gradient tree boosting model while the x-axis shows investors' sector on which predictions are made using the model indicated on y-axis. Scores are average precision, macro-averaged over classes and expressed in percentages.

We use a dataset constructed as described above with five months of bonds' RFQ data. We split this dataset into many subsets according to the investors' business sector, e.g., one of these

subsets contains investors coming from the Insurance sector only. We consider here only the sectors with a sufficient amount of data samples to train and test a model. The remaining sectors are grouped under the *Others* flag. Note that this flag is mainly composed of Corporate sectors, such as car industry, media, technology, telecommunications... For each sector, some of the latest data is held out, and a LightGBM model (Ke et al., 2017) is trained on the remaining data. This model is then used for prediction on the held-out data of the model's underlying sector, and for all the other sectors as well. For comparison purposes, an aggregated model using all sectors at once is also trained and tested in the same way.

Because classes are unbalanced, we compute the average precision score of the obtained results, as advised by Davis and Goadrich (2006), macro-averaged over all the classes (see Section 1.4.1), which yields the *universality matrix* shown in Fig. 3.1. The y-axis labels the sector used for training, and the x-axis is the section on which the predictions are made.

We observe that some sectors are inherently difficult to predict, even when calibrated on their data only — this is the case for Asset Managers of Private Banks and Pension Funds. On the contrary, some sectors seem to be relatively easy to predict, e.g., Broker Dealers and, to some extent, Central Banks. Overall, we note that there is always some degree of variability of the scores obtained by a given model — no universal model gives good predictions for all the sectors of activity. Thus follows the *non-universality of clients*. In addition, it is worth noting that the aggregated model obtained better performance in some sectors than the models trained on these sectors' data only. As a consequence, a suitable grouping of sectors would improve predictions for some sectors. This observation is in agreement with the above K-investment strategies hypothesis.

Following on these hypotheses, the algorithm introduced in this chapter leverages deep learning both to uncover the structure of similarity between investors, namely the K clusters, or strategies, and to make relevant predictions on each inferred cluster. The advantage of deep learning lies in that it allows solving both tasks at once and thereby unveils the structure of investors that most closely corresponds to their trading behavior in a self-consistent way.

3.1.3 Related work

The work conducted in this chapter finds its roots in mixture-of-experts research, which began with Jacobs et al. (1991), from which we keep the essential elements that drive the structure presented in Section 3.2, and more particularly the gating and expert blocks. A rather exhaustive history of the research performed on this subject can be found in Yuksel et al. (2012).

The main inspiration for our work is Shazeer et al. (2017), which, although falling within the *conditional computation* framework, presented the first adaptation of mixture of experts for deep learning models. We build on this work to come up with a novel structure designed to solve the particular problem presented in Section 3.1. As far as we know, the approach we propose is new. We use an additional loss term to improve the learning of the strategies, reminiscent of the one introduced in Liu and Yao (1999).

3.2 Introducing Experts Network

We introduce here a new algorithm called the *Experts Network* (ExNet). The ExNet is purposely designed to be able to capture the hypotheses formulated in Section 3.1, i.e., to capture a finite, unknown number K of distinct investment strategies $\mathcal{D} = \{f_k : x \mapsto y\}_{k=1,\cdots,K}$.

3.2.1 Architecture of the network

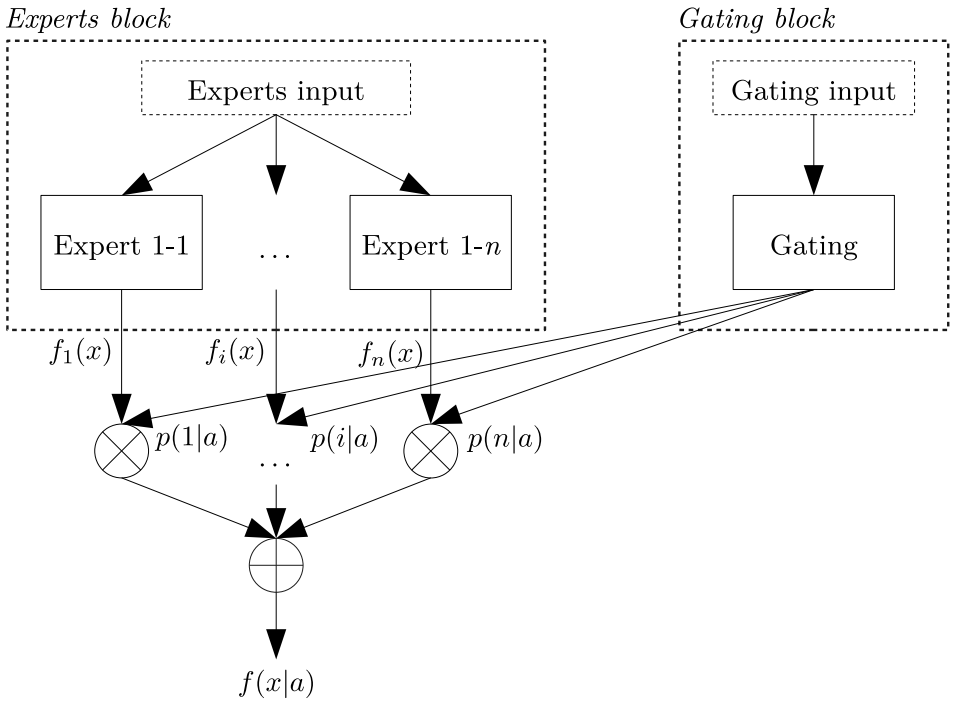


Figure 3.2: Global architecture of the Experts Network. The network is composed of two blocks: an expert block, comprising n independent neural networks, and a gating block, consisting of an independent neural network whose role is to allocate investors to the different experts. These blocks receive different input data, indexed by the same *(Date, Client, Financial Product)* triplet.

The structure of an ExNet, illustrated in Fig. 3.2, comprises two main parts: a **gating block** and an **experts block**. Their purposes are the following:

- The **gating block** is an independent neural network whose role is to learn how to dispatch investors to n *experts* defined below. This block receives a distinct, categorical input, the *gating input*, corresponding to an encoding of the investors and such that the i-th gating input sample is related to the same investor than the i-th experts' input sample. Its output consists of a vector of size n, which contains the probabilities that the input should be allocated to the n experts, computed by a softmax activation.
- The **experts block** is made of n independent sub-networks, called *experts*. Each expert receives as input the same data, the *experts' input*, corresponding to the features used to solve the classification or regression task at hand, e.g., in our case, the features outlined in Section 3.1: for a given row, the intensity of the investor's interest in the financial asset considered, the total number of RFQ done by the investor, the price and the volatility of the asset... As investors are dispatched to the experts through the gating block, each expert will learn a mapping $f : x \mapsto y$ that most closely corresponds to the actions of its attributed investors. Consequently, the role of an expert is to retrieve a given f_k,

corresponding to one of the K underlying clusters of investors we hypothesized.

The outputs of these two blocks are combined through

$$f(x|a) = \sum_{i=1}^{n} p(i|a) f_i(x), \tag{3.3}$$

where a denotes the investor related to data sample x and $p(i|a)$ is the probability that investor a is assigned to expert i. Our goal is that K experts learn to specialize in K clusters. As K is unknown, retrieving all clusters requires that $n \geq K$, i.e., n should be "large enough". We will show below that high values of n do not impact the network ability to retrieve the K clusters; using large n values therefore ensures that the $n \geq K$ condition is respected and only impacts computational efficiency. The described architecture corresponds in fact to a *meta*-architecture. The architecture of the experts is still to be chosen, and indeed any neural network architecture could be used. For simplicity and computational ease, we use here rather small feed-forward neural networks for the experts, all with the same architecture, but one could easily use experts of different architectures to represent a more heterogeneous space of strategies.

Both blocks are trained simultaneously using gradient descent and backpropagation, with a loss corresponding to the task at hand, be it a regression or classification task, and computed using the final output of the network only, $f(x|a)$. One of the most important features of this network lies in the fact that the two blocks do not receive the same input data. We saw previously that the gating block receives as input an encoding of the investors. As this input is not time-dependent, the gating block of the network can be used *a posteriori* to analyze how investors are dispatched to experts with a single pass of all investors' encodings through this block alone, thereby unveiling the underlying structure of investors interacting in the considered market.

For a given investor a, the gating block computes attribution probabilities of investor a to each expert

$$p(x|a) = Softmax\left(W_{experts} * x\right), \tag{3.4}$$

where x is a trainable d-dimensional embedding of the investor a, $W_{experts}$ is a trainable $n \times d$-dimensional matrix where the i-th row corresponds to the embedding of the corresponding expert, and we define $Softmax(x)_k = e^{x_k} / \sum_i e^{x_i}$.

3.2.2 Disambiguation of investors' experts mapping

The ExNet architecture is similar to an ensemble of independent neural networks, where the gating block of the network gives the weighing of the average. We empirically noticed that ExNets might assign equal weights to all experts for all investors without additional penalization. To avoid this behavior, and help each investor follow a single expert, we introduce an additional loss term

$$L_{\text{entropy}} = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \sum_{j=1}^{n} p(j|a_i) \log p(j|a_i), \tag{3.5}$$

where $\mathcal{B}$ is the current batch of data considered, n is the number of experts, and $p(j|a_i)$ is the attribution of investor a_i to the j-th expert. This loss term corresponds exactly to the entropy of the probability distribution over experts of a given investor. Minimizing this loss term will therefore encourage distributions over experts to peak on one expert only.

3.2.3 Helping experts specialize

Without a suitable additional loss term, the network has a tendency to let a few experts learn the same investment strategy, which also leads to more ambiguous mappings from investors to experts. Thus, to help the network finding different investment strategies and to increase its discrimination power regarding investors, we add a *specialization loss* term, which involves cross-experts correlations, weighted accordingly to their relative attribution probabilities. It is written as:

$$L_{\text{spec}} = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} w_{i,j} \overline{\rho}_{i,j} \tag{3.6}$$

$$\text{with } w_{i,j} = \frac{\overline{p}_i \overline{p}_j}{\sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \overline{p}_i \overline{p}_j} \text{ if } i \neq j, 0 \text{ else,} \tag{3.7}$$

$$\text{and } \overline{p}_i = \frac{\sum_{a \in A} p_i^a}{\sum_{a \in A} \mathbb{1}_{p_i^a > 0}}. \tag{3.8}$$

Here, $i, j \in \{1, \ldots, n\}$, $\overline{\rho}_{i,j}$ is the batch-wise correlation between experts i and j outputs, averaged over the output dimension, and $\overline{p}_i$ is the batch-wise mean attribution probability to expert i, with p_i^a the attribution probability of investor a to expert i, computed on the current batch of investors that counted this expert in their top L only. The intuition behind this weight is that we want to avoid correlation between experts that were confidently selected by investors, i.e., to make sure that the experts that matter do not replicate the same investment strategy. As the number of the investors clustered around a given expert should not matter in this weighing, we only account for the nonnegative probabilities for all the considered investors in the weights, computed using relative mean probabilities $\overline{p}_i$. In some experiments, it was found useful to rescale L_{spec} from $[-1; 1]$ to $[0; 1]$.

This additional loss term is reminiscent of Liu and Yao (1999). As a matter of fact, in ensembles of machine learning models, negatively correlated models are expected to perform better than positively correlated ones. A better performance can also be expected from the experts of an ExNet, as negatively correlated experts better span the space of investment strategies. As the number of very distinct strategies grow, we can expect to find strategies that more closely match the ones investors follow in the considered market, or the basis functions on which investment strategies can be decomposed.

3.2.4 From gating to classification

Up to this point, we only discussed gating inputs related to investors. However, as seen above, being able to retrieve the structure of attribution of inputs to experts only requires to use categorical data as input to the gating part of the network. In fact, gating can be performed on whatever is thought to be suitable — for instance, it is reasonable to think that bond investors have different investment strategies depending on the bonds' grades or on the sector of activity of the bond issuers. Higher-level details about investors could also be considered, e.g., because investment strategies may depend on factors such as the sector of activity of the investor, i.e., whether it is a hedge fund, a central bank, or an asset manager, or the region of the investor. The investor dimension could even be totally forgotten, and the gating performed on asset-related categories only. Gating allows one to retrieve the underlying structure of interactions of a given category or set of categories. Consequently, one can purposely set categories to study how they relate to the problem one wants to study. However, this may impact the model's performance, as chosen categories do not necessarily have distinct decision rules.

Note also that the initialization of weights in the gating network has a major impact on the algorithm's future performance. To find relevant clusters, i.e., clusters that are composed of

unequivocally attributed categories and that correspond to the original clusters expected in the dataset, categories need to be able to explore many different clusters' configurations before the exploitation of one relevant configuration. Consequently, the gating block must be initialized so that all the expert weights are fairly evenly initially distributed to allow for this exploration. In our implementation, we use a random normal initialization scheme for the d-dimensional embeddings of the categories and of the experts.

3.2.5 Limitations of the approach

Our approach allows us to treat well a known, fixed base of investors. However, it cannot efficiently deal with new investors or, at a higher level, new categories, as seen in Section 3.2.4, as embeddings for these new types of elements would need to be trained from scratch. To cope with such situations, we recommend using sets of fixed categories to describe the evolving ones. For instance, instead of performing gating on investors directly, one can use investors' categories such as sector, region,... that are already present in the dataset and on which we can train embeddings. Doing so improves the robustness of our approach to unseen categories. Note that this limitation corresponds to the cold-start problem, already encountered in Section 1.3.5 and further explored in Section 4.4.3.

3.3 Experiments

Before testing the ExNet architecture on real data, we first check its ability to recover a known strategy set, to attribute investors correctly to strategies, and finally to classify the interest of investors on synthetic data. We then show how our methodology compares with other algorithms on two different datasets: a dataset open-sourced[1] as part of the experiments presented in Gutiérrez-Roig et al. (2019), and a BNP Paribas CIB dataset. Source code for the experiments on synthetic data and the open-source dataset is provided and can be found at `https://github.com/BptBrr/deep_prediction`.

3.3.1 Synthetic data

Generating the dataset

Taking a cue from BNP Paribas CIB bonds' RFQ database (cf. Section 1.2.2), we define three clusters of investors, each having a distinct investment strategy, which we label as "high-activity", "low-activity" and "medium-activity". Each cluster contains a different proportion of investors, and each investor within a cluster has the same activity frequency: the "high-activity" cluster accounts for roughly 70% of the dataset samples while containing roughly 10% of the total number of investors. The "low-activity" cluster accounts for roughly 10% of the samples while containing roughly 50% of the total number of investors. The "medium-activity" cluster accounts for the remaining number of samples and investors. In all the clusters, we assume that investors are equally active.

We model the state of investors as a binary classification task, with a set of p features, denoted by $X \in \mathbb{R}^p$, and a binary output Y representing the fact that an investor is interested or not in the considered asset. Investor a belonging to cluster c follows the decision rule given by

$$Y_a = (\sigma(w_a^T X) > U) \in \{0, 1\} \tag{3.9}$$

, where

$$w_a = w_c + b_a \in \mathbb{R}^p, \tag{3.10}$$

[1] Data available at `https://zenodo.org/record/2573031`

$w_c \in \mathbb{R}^p$ being the cluster weights and $b_a \sim \mathcal{N}(0, \alpha) \in \mathbb{R}$ an investor-specific bias, $X_i \sim \mathcal{N}(0, 1)$ for $i = 1, \cdots, p$, U is distributed according to the uniform distribution on $[0, 1]$, and σ is the logistic function.

The experiment is conducted using a generated dataset of $100,000$ samples, 500 investors and $p = 5$ features. This dataset is split into train/validation/test sets, corresponding to 70/20/10% of the whole dataset. α is set to 0.5, and the cluster weights are taken as follows:

- High-activity cluster: $w_{high} = (5, 5, 0, 0, -5)$
- Low-activity cluster: $w_{low} = (-5, 0, 5, 0, 5)$
- Medium-activity cluster: $w_{medium} = (-5, 0, 5, 5, 0)$

These weights are chosen so that the correlation between the low- and medium-activity clusters is positive, but both are negatively correlated with the high-activity cluster. In this way, we build a structure of clusters, core decision rules, and correlation patterns that are sufficiently challenging to demonstrate the usefulness of our approach.

Results

We examine the performance of our proposed algorithm, ExNet, against a benchmark algorithm, LightGBM (Ke et al., 2017). LightGBM is a widespread implementation of gradient boosting, as shown, for example, by the percentage of top Kaggle submissions that use it. The LightGBM is fed with the experts' input and an encoding of the corresponding investors, used as a categorical feature in the LightGBM algorithm. For comparison purposes, experiments are also performed on a LightGBM model fed with experts' input and an encoding of the investors' underlying clusters, i.e., whether the investor belongs to the high-, low- or medium-activity cluster, called *LGBM-Cluster*.

ExNets are trained using the cross-entropy loss since we want to solve a classification problem. The network is optimized using Nadam (Dozat, 2016), a variation of the Adam optimizer (Kingma and Ba, 2015) using Nesterov's Accelerated Gradient (Nesterov, 1983), reintroduced in the deep learning framework by Sutskever et al. (2013), and Lookahead (Zhang et al., 2019). For comparison purposes, experiments are also performed on a multi-layer perceptron model fed with the experts' inputs concatenated with a trainable embedding of the investors, called *Embed-MLP* — consequently, this model differs from a one-expert ExNet in that such an ExNet would not use an embedding of the investor to perform its predictions. All neural network models presented here used the rectified linear unit, $\text{ReLU}(x) = \max(x, 0)$, as activation function (Nair and Hinton, 2010).

LightGBM, ExNet and Embed-MLP results are shown in Table 3.1. They were obtained using a combination of random search (Bergstra and Bengio, 2012) and manual fine-tuning. LightGBM-Cluster results used the hyperparameters found for LightGBM. These results correspond to the model achieving the best validation accuracy over all our tests. The LightGBM and LightGBM-Cluster shown had 64 leaves, a minimum of 32 samples per leaf, a maximum depth of 10, a learning rate of 0.005, and a subsample ratio of 25% with a frequency of 2. ExNet-Opt, the ExNet which achieved the best validation accuracy, used 16 experts with three hidden layers of sizes 64, 128 and 64, a dropout rate (Srivastava et al., 2014) of 40%, loss weights $\lambda_{spec} = 7e^{-3}$ and $\lambda_{entropy} = 1e^{-3}$, a batch size of 1024, and a learning rate of $7e-3$. The Embed-MLP model shown used two hidden layers of size 128 and 64, a dropout rate of 15%, an embedding size $d = 64$, a batch size of 64, and a learning rate of $4.2e^{-5}$.

To study the influence of the number of experts on the performance of ExNets, we call *ExNet-n* an ExNet algorithm with n experts and vary n. These ExNets used experts with no hidden layers, batch-normalized inputs (Ioffe and Szegedy, 2015), and an investor embedding of size

$d = 64$. They were trained for 200 epochs using early stopping with a patience of 20. These experiments were carried out with a learning rate of e^{-3} and a batch size of 64, which led to satisfactory solutions in all tested configurations. In other words, we only vary n to be able to disentangle the influence of n for an overall reasonably good choice of other hyperparameters. Only the weights attributed to the specialization and entropy losses, λ_{spec} and λ_{entropy}, were allowed to change across experiments.

Table 3.1: Experimental results on synthetic data: accuracy of predictions, expressed in percentage, on train, validation and test sets, and on subsets of the test set corresponding to the original clusters to be retrieved.

Algorithm	Train Acc.	Val Acc.	Test Acc.	High Acc.	Medium Acc.	Low Acc.
LGBM	**96.38**	92.05	92.41	92.85	90.47	92.34
LGBM-Cluster	93.89	92.33	92.94	93.03	92.54	**93.89**
Embed-MLP	93.87	92.88	93.19	93.20	93.14	92.24
ExNet-Opt	93.57	**92.99**	**93.47**	**93.56**	93.09	93.17
ExNet-1	74.86	74.56	74.56	80.39	48.67	38.72
ExNet-2	90.73	90.59	90.86	91.66	87.32	82.71
ExNet-3	92.73	92.50	93.06	92.97	93.47	**93.89**
ExNet-10	92.91	92.66	93.16	93.12	93.36	**93.89**
ExNet-100	92.71	92.55	93.04	92.96	**93.41**	**93.89**
Perfect model	93.62	93.51	93.71	93.75	93.52	94.82

Table 3.1 contains the results for all the tested implementations. As the binary classification considered here is balanced, we use accuracy as the evaluation metric. This table reports results on train, validation, and test splits of the dataset, and a view of the test results on the three different clusters generated. As the generation process provides us with the probabilities of positive events, it is also possible to compute metrics for a model that would output these probabilities, denoted here as *perfect model*, which sets the mark of what good predictive performance is in this experiment.

We see here that the LGBM implementation fails to retrieve the different clusters completely. LGBM focused on the high-activity cluster and mixed the two others, which led to poorer predictions for both of these clusters, particularly for the medium-activity one. In comparison, LGBM-Cluster performed significantly better on the medium- and low-activity clusters. Embed-MLP better captured the structure of the problem but appears to mix the medium- and low-activity clusters as well — the performance on the low-activity cluster is the lowest of the three and the medium- and low-activity clusters' decision functions are similar —,albeit getting a better predictive performance. ExNet-Opt, found with random search, captured well all clusters and obtained the best overall performances.

Moreover, the ExNet-n experiment shows how the algorithm behaves as n increases. ExNet-1 successfully captured the largest cluster in terms of samples, i.e., the high-activity one, partly ignoring the two others, and therefore obtained poor overall performance. ExNet-2 behaved as the LGBM experiment, retrieving the high-activity cluster and mixing the remaining two. ExNet-3 perfectly retrieved the three clusters, as expected. Even better, the three clusters were also perfectly retrieved with ExNet-10 and ExNet-100: this is because the ExNet algorithm, thanks to the additional specialization loss, is not sensitive to the number of experts even if $n \gg K$, as long as there are enough of them. Thus, when $n \geq K$, the ExNet can retrieve the K initial clusters and predict the interests of these clusters satisfactorily.

Further analysis of specialization

The previous results show that as long as $n \geq K$, the ExNet algorithm can efficiently capture the investment strategies corresponding to the underlying investor clusters. One still needs to check that the attribution to experts is working well, i.e., that the investors are mapped to a single, unique expert. To this end, we retrieved from the gating block the attribution probabilities to the n experts of all the investors *a posteriori*. For comparison, we also analyze the investors' embeddings of Embed-MLP. The comparison of the final embeddings of ExNet-Opt and the ones trained in the Embed-MLP algorithm is shown in Fig. 3.3.

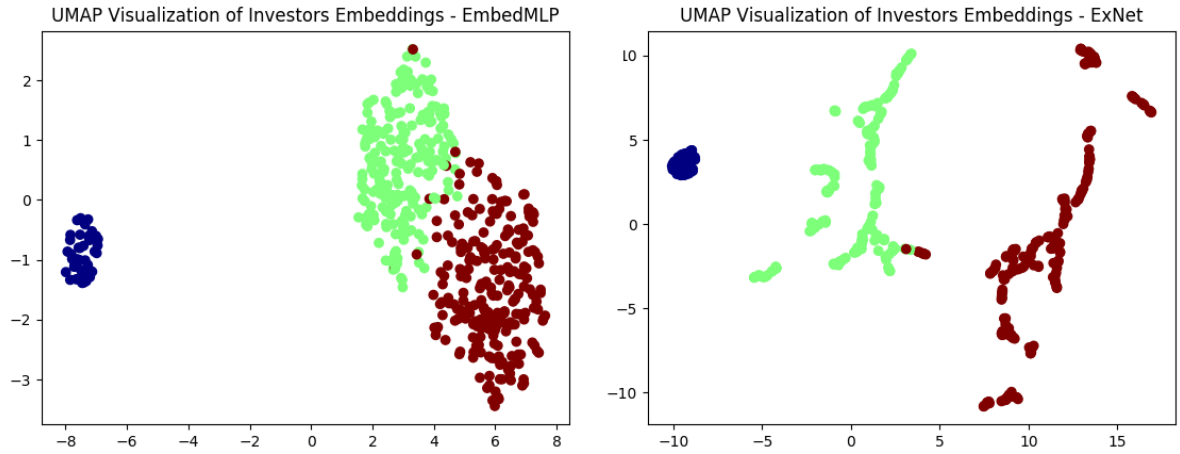


Figure 3.3: UMAP visualization of investors embeddings for both Embed-MLP (left) and ExNet (right) algorithms. Colors correspond to investors' original clusters: high-activity is shown in blue, medium-activity in green and low-activity in red.

To visualize embeddings, we use here the UMAP algorithm (McInnes et al., 2018), which is particularly relevant as it seeks to preserve the topological structure of the embeddings' data manifold in a lower-dimensional space, thus keeping vectors that are close in the original space close in the embedding space, and making inter-cluster distances meaningful in the two-dimensional plot.

Consequently, the two-dimensional map given by UMAP is a helpful tool for understanding how investors relate to each other according to the deep learning method considered. In these plots, high-activity investors are shown in blue, low-activity investors in red and medium-activity investors in green. We can see in Fig. 3.3 that the Embed-MLP algorithm did not make an apparent distinction between the low- and medium-activity clusters, contrarily to the ExNet, which separated these two categories except for a few low-activity investors mixed in the medium-activity cluster. The ExNet algorithm was, therefore, completely able to retrieve the original clusters.

The attribution probabilities to the different experts of ExNet-Opt are shown in Fig. 3.4. We see in this figure that the attribution structure of this ExNet instance is quite noisy, with three different behaviors discernable. The first group of investors corresponds to the low-activity cluster, the second group to the medium-activity cluster, and the last one to the high-activity cluster. Here, attributions are very noisy and investors of a same cluster are not uniquely mapped to an expert.

However, it is possible to achieve a more satisfactory experts attribution, as one can see in Fig. 3.5 with the plots of ExNet-100. The more precise attribution comes from the fact that the ExNet-100 instance used a higher level of $\lambda_{entropy}$ than ExNet-Opt — at the expense of some performance, we can obtain far cleaner attributions to the experts. We see here on the left

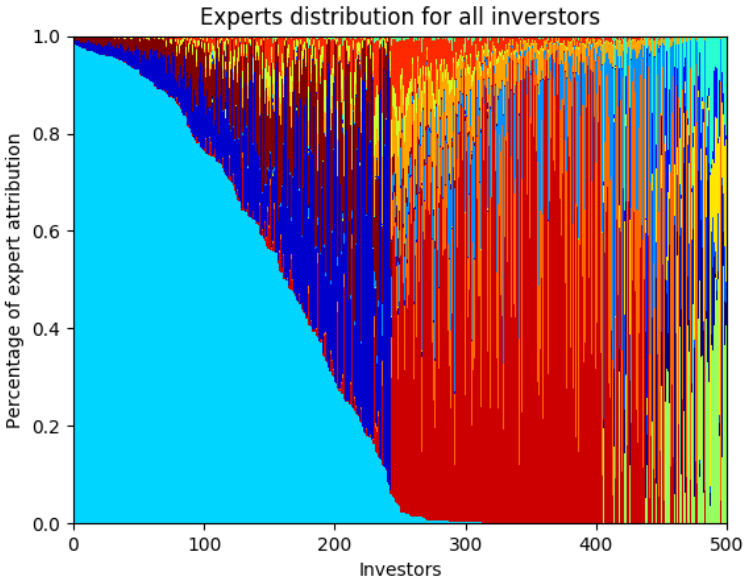


Figure 3.4: Distribution over experts of all investors for ExNet-Opt, obtained with random search. Each column shows the attribution probabilities of a given investor, where colors represent experts.

plot that all investors were attributed almost entirely to one expert only, with for each of the corresponding experts mean attribution probabilities $\overline{p} > 99\%$, even with an initial number of experts of 100, i.e., the $n \gg K$ setting. One can directly see on the UMAP plot three well-defined, monochromatic clusters. We can also see here that a low-activity investor got mixed in the medium-activity cluster, and that two separated low-activity clusters appear — these separated clusters originate from the fact that some low- and medium-activity investors were marginally attributed to the expert corresponding to the other cluster, as appearing on the experts' distribution plot.

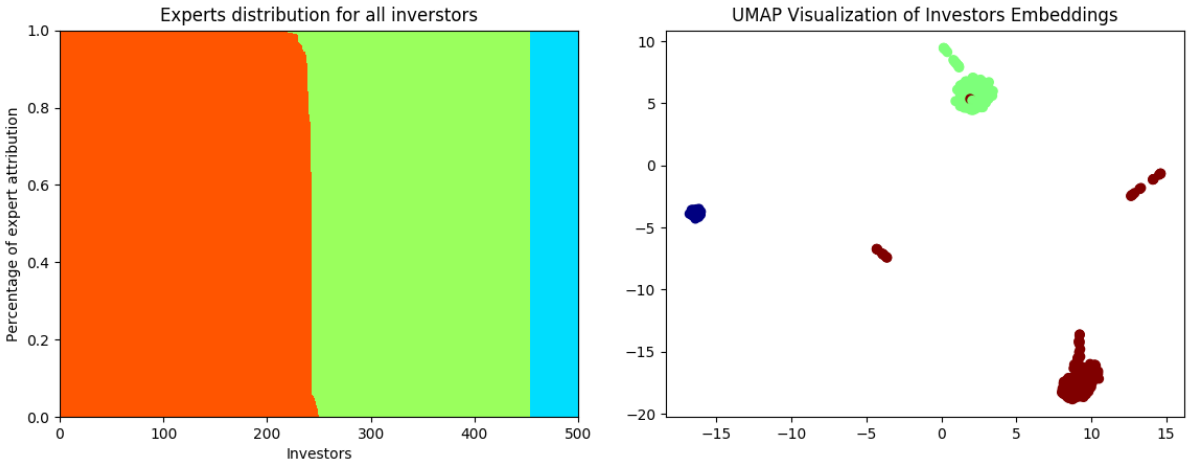


Figure 3.5: Distribution over experts of all investors and UMAP visualization of investors embeddings for the ExNet-100 algorithm. Each column of the left plot shows the attribution probabilities of a given investor, where colors represent experts. Colors on the right plot correspond to investors' original clusters: high-activity is shown in blue, medium-activity in green and low-activity in red.

The ExNet-100 therefore solved the problem that we originally defined, obtaining excellent predictive performance on the three original clusters and uniquely mapping investors to one expert only, thereby explicitly uncovering the initial structure of the investors, a feature that an algorithm such as Embed-MLP is unable to perform.

3.3.2 IBEX data

Constructing the dataset

This experiment uses a real-world, publicly available dataset published as part of Gutiérrez-Roig et al. (2019) (`https://zenodo.org/record/2573031`) which contains data about a few hundred private investors trading 8 Spanish equities from the IBEX index, from January 2000 to October 2007. For a given stock and each day and each investor, the dataset gives the end-of-the-day position, the open, close, maximum, and minimum prices of the stock as well as the traded volume.

We focus here on the stock of the Spanish telecommunication company Telefónica, TEF, as it is the stock with the largest number of trades. Using this data, we try to predict, at each date, whether an investor will be interested in buying TEF or not. An investor is considered to have an interest into buying TEF when

$$\Delta p_t^a = p_t^a - p_{t-1}^a > 0, \tag{3.11}$$

where p_t^a is the position of investor a at time t. We only consider here the buy interest, as the sell interest of private investors can be driven by exogenous factors that cannot be modeled, such as a liquidity shortage of an investor, whereas the buy interest of an investor depends, to some extent, on market conditions. Thus, we face a binary classification problem which is highly unbalanced: on average, a buy event occurs with a frequency of 2.7%.

We consider a temporal split of our data in three parts: training data is taken from January 2000 to December 2005, validation data from January 2006 to December 2006, and test data from January 2007 to October 2007. We restrict our investor perimeter to investors that bought TEF more than 20 times during the training period. We build two kinds of features:

- **Position features.** Position is shifted such that at date t corresponds p_{t-1}, and is normalized for each investor using statistics computed on the training set. This normalized, shifted position is used as a feature, along with moving averages of it with windows of 1 month, 3 months, 6 months and 1 year.
- **Technical analysis features.** We compute all the features available in the `ta` package (Padial, 2018), which are grouped under five categories: Volume, Volatility, Trend, Momentum, and Others features. As most of these features use close price information, we shift them such that features at a date t only use the information available up to $t-1$.

We are left with 308 rather active investors and 63 features composing six different groups.

Results

ExNet and LightGBM are both trained using a combination of random search (Bergstra and Bengio, 2012) and hand fine-tuning. Because of the class imbalance of the dataset, the ExNet is trained using the focal loss (Lin et al., 2017), an adaptive re-weighting of the cross-entropy loss introduced in Section 1.3.2. Other popular techniques to handle class imbalance involve under-sampling the majority class and/or oversampling the minority one, such as SMOTE (Chawla et al., 2002). The γ parameter of this loss is randomly searched, as any other hyperparameter of the network. We also use the baseline buy activity rate of each investor in the training period as a benchmark.

Table 3.2: Experimental results on IBEX data: average precision scores, expressed in percentage.

Algorithm	Train	Val	Test
Historical	9.68	4.55	2.49
LGBM	**22.22**	7.53	5.35
ExNet-4	18.37	**8.63**	**6.45**

The LightGBM shown in Table 3.2 used 16 leaves with a minimum of 128 samples per leaf, a maximum depth of 4, a learning rate of 0.002, a subsample ratio of 35% with a frequence of 5, a sampling of 85% of columns per tree, with a patience of 50 for a maximum number of trees of 5000. The ExNet shown used 4 experts with two hidden layers of size 32 and 32 with a dropout ratio of 50%, embeddings of size $d = 32$, an input dropout of 10%, $\lambda_{spec} = 7.7e^{-4}$ and $\lambda_{entropy} = 4.2e^{-2}$, a focal loss of parameter $\gamma = 2.5$, a batch size of 1024, a learning rate of $7.8e^{-4}$ and was trained using Nadam and Lookahead, with an early stopping of patience 20. As can be seen on this table, both algorithms beat the historical baseline, and the ExNet achieved overall better test performance. While the precision of LightGBM is better in the training set, it is inferior to that of ExNet in the validation set, a sign that ExNet is less prone to overfitting than LightGBM.

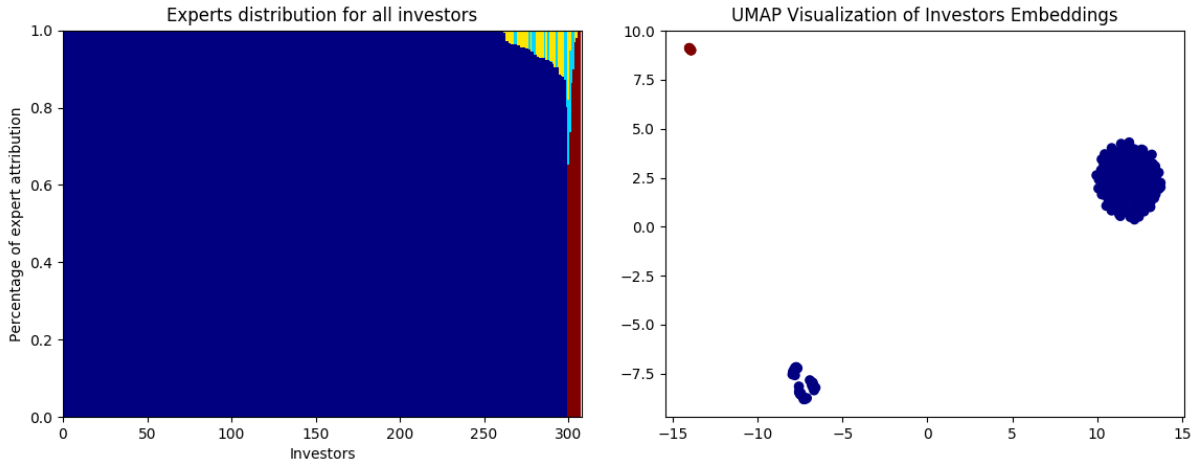


Figure 3.6: Distribution over experts of all investors and UMAP visualization of investors embeddings for the ExNet algorithm on the TEF dataset. Each column of the left plot shows the attribution probabilities of a given investor, where colors represent experts — same colors are used on the right plot. We call here cluster 1 the blue cluster on the right, cluster 2 the blue cluster on the left and cluster 3 the red cluster on the top-left corner.

Figure 3.6 gives a more in-depth view of the results obtained by the ExNet algorithm. Three distinct behaviors appear in the left plot. Some of the investors were entirely attributed to the

blue expert (Cluster 1), some investors used a combination of the blue expert and two others (Cluster 2) and some used combinations of the light blue and red experts (Cluster 3). These three clusters are remarkably spaced in the UMAP plot on the right. Therefore, it appears that the ExNet retrieved three distinct behavioral patterns from the investors interacting on the TEF stock, leading to overall better performance than the LightGBM, who was not able to capture them, as the experiments performed in Section 3.3.1 show.

Experts analysis

We saw in Section 3.3.2 that the ExNet algorithm retrieved three different clusters. Let us investigate in more detail what these clusters correspond to. First, the investors' typical trading frequency attributed to each of these three clusters is clearly different. The investors that were mainly attributed to the blue expert in Fig. 3.6, corresponding to cluster 1 on the UMAP plot, can be understood as "low-activity" investors, trading on average 2.1% of the time. Cluster 2 can be understood as medium-activity investors, buying on average 5.5% of the days; cluster 3 is made of high-activity investors buying on average 13.9% of the time. The ExNet therefore retrieved particularly well three distinct behaviors, corresponding to three different activity patterns.

To get a better understanding of these three clusters, we can try to assess these clusters' sensitivity to the features used in the model. We use here *permutation importance*, a popular method in machine learning, whose principle was described for a similar method in Breiman (2001). This method replaces a given feature by permutations of all its values in the inputs and assesses how the model's performance evolves in that setting. We applied this methodology to the six groups of features: we performed the shuffle a hundred times and averaged the corresponding performance variations. For each of the three clusters, we pick the investor who traded the most frequently and apply permutation importance to characterize the cluster's behavior. Results are reported in Table 3.3.

Table 3.3: Percentage of average precision variation when perturbing features of the group given in the first column for the three clusters appearing in Fig. 3.6, using permutation importance. Cluster 1 corresponds to the cluster of low-activity investors, cluster 2 to the medium-activity ones and cluster 3 to the high-activity ones.

Feature group	Cluster 1	Cluster 2	Cluster 3
Position	**-37.4%**	**-43.1%**	**-23.7%**
Volume	**-18.6%**	+10.6%	**-19.9%**
Volatility	**-22.8%**	-4.5%	-2.1%
Trend	**-9%**	-2.4%	-3.7%
Momentum	+1.7%	+4.3%	**-13.5%**
Others	-0.7%	+8.3%	+0.2%

We see that the three groups have different sensibilities to the groups of features that we use in this model. While all clusters are particularly sensitive to position features, the respective sensitivity of groups to the other features vary: leaving aside cluster 2 that only looks sensitive to position, cluster 1 is also sensitive to volume, volatility, and trend, whereas cluster 3 is also sensitive to volume and momentum. Consequently, the clusters encode not only the activity rate, but also the type of information that a strategy needs, and by extension the family of the strategies used by investors, thereby validating the intuition that underpins the ExNet algorithm.

3.3.3 BNPP CIB data

The previous experiments proved the ability of the network to retrieve the structure of investors with a finite set of fixed investment strategies, and the usefulness of our approach on a real-world dataset. We now give an overview of the results we obtain on the BNPP CIB bonds' RFQ dataset specified in Section 3.1 for the non-universality of clients' study.

As a reminder, assets considered are corporate bonds. The data used ranges from early 2017 to the end of 2018 with temporal train/val/test splits and is made of custom proprietary features using clients-related, assets-related, and trades-related data, as seen in Section 2.2.4. Our goal is, at a given day, to predict the interest of a given investor into buying and/or selling a given bond; each row of the dataset is therefore indexed by a triplet *(date, investor, bond)*. Targets are constructed as previously explained in Section 3.1. In the experiment, we consider 1422 different investors interacting around a total of more than 20000 distinct bonds.

The left-hand plot of Fig. 3.7 shows the distribution over experts for all the 1422 considered investors. We see three different patterns appearing: one which used the brown expert only, another one the green expert only and a composite one. These patterns lead to four clusters on the right-hand plot. In this plot, as previously done in the IBEX experiment, each point corresponds to an investor, whose color is determined by the expert to which she is mainly attributed, with colors matching the ones of the left-hand plot. We empirically remark that these clusters all have different activities: the larger brown cluster is twice more active than the green one, the two smaller clusters having in-between average activities. The ExNet therefore retrieved distinct behavioral patterns, confirmed by a global rescaled specialization loss below 0.5, hence negatively correlated experts.

Obtaining finer clusters could be achieved in multiple ways. A higher-level category could be used as gating input: instead of encoding investors directly, one could encode their sector of activity, in the fashion of the non-universality of clients experiment. With an encoding of the investors, training an ExNet on a restricted set of investors corresponding to one of the retrieved clusters only would also lead to a finer clustering of the investors — a two-stage gating process could even directly lead to it and will be part of further investigations on the ExNet algorithm. Note however that these maps (and ExNets) are built from measures of simultaneous distance, hence, do not exploit lead-lag relationships — how ExNets could be adapted to a temporal setting to retrieve lead-lag relationships will be worthy of future investigations as well.

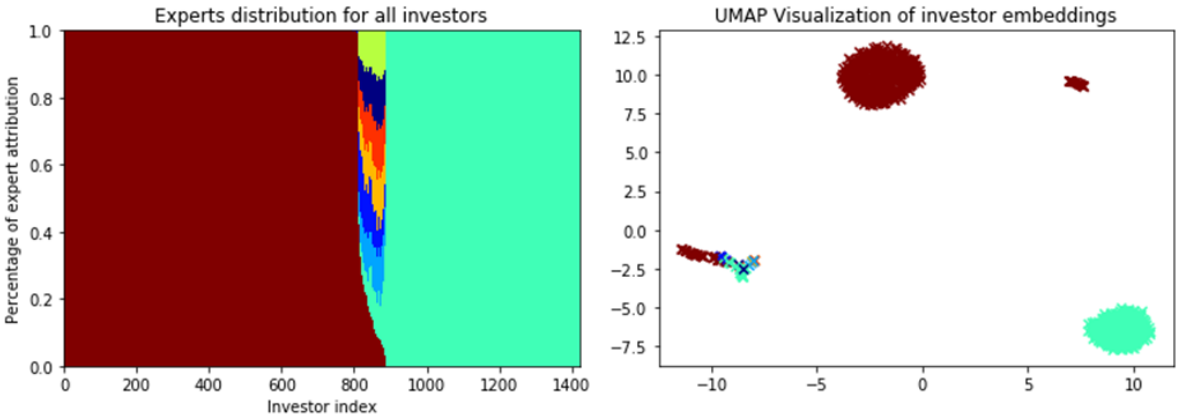


Figure 3.7: Distribution over experts of all investors and UMAP visualization of investors embeddings for the ExNet algorithm on the BNPP CIB Bonds' RFQ dataset. Each column of the left plot shows the attribution probabilities of a given investor, where colors represent experts — same colors are used on the right plot.

On a global scale, these plots help us understand how investors relate to each other. Therefore, one can use them to obtain a better understanding of BNP Paribas CIB business and how BNP Paribas CIB clients' behave on a given market through a thorough analysis of the learned experts.

3.4 A collection of unfortunate ideas

Designing the ExNet architecture presented above took time, efforts, and a consequent number of dead ends. Throughout the development of this algorithm, and more generally, for all the work presented in this thesis, Occam's razor served as a guiding principle: when competing algorithms led to similar results, we always favored the simpler one. This section presents, along with milestone designs that led to the actual one, a compendium of potentially promising ideas that led to mild disappointments, and consequently suffered the razor blade.

3.4.1 About architecture

Although the network presented in Section 3.2 is inspired to a great extent by the work of Shazeer et al. (2017), a lot of architectural variations were tried to obtain more satisfactory results. We examine here two of the most promising ideas we tested.

Representing the shared features of investment strategies

The ExNet design corresponds to the intuition that on a given market, investors tend to form clusters corresponding to particular investment strategies. Investment strategies, however, might share features — at the very least, their shared goal of making money. This core behavior underpinning all potential investment strategies can be explicitly represented as another independent neural network, receiving as input the same features experts do.

The corresponding architecture is depicted in Fig. 3.8. We see that an "average" block, using the same input as experts, is systematically added to the recombination of experts, leading to an output

$$f(x|a) = f_{\text{avg}}(x) + \sum_{i=1}^{n} f_i(x)p(i|a). \tag{3.12}$$

In such a setting, experts $f_i(x)$ learn what amounts to *residuals* of the final output, in a manner close to the first step of a boosting algorithm (Hastie et al., 2009, Chapter 10), or more recently to a residual block (He et al., 2016).

This architecture was proven successful in learning the output but failed to cluster investors to experts correctly, and was consequently abandoned to designs that succeeded at both tasks.

A gating strategy built for exploration

The initial design of the ExNet's gating block was a lot closer to the architecture presented in (Shazeer et al., 2017). The idea is that, for a given investor a, the gating block performs the following computation :

$$p(x|a) = Softmax\left(KeepTopL\left(W_{gating}x + \epsilon * \sqrt{Softplus(W_{noise}x)}\right)\right) \tag{3.13}$$

where x is a trainable d-dimensional embedding of the investor a, W_{noise}, W_{gating} are trainable weights of the network, $\epsilon \sim \mathcal{N}(0,1)$, the softmax function defined as before and with the softplus function defined as $Softplus(x) = \log(1 + \exp x)$. The $KeepTopL$ function, introduced

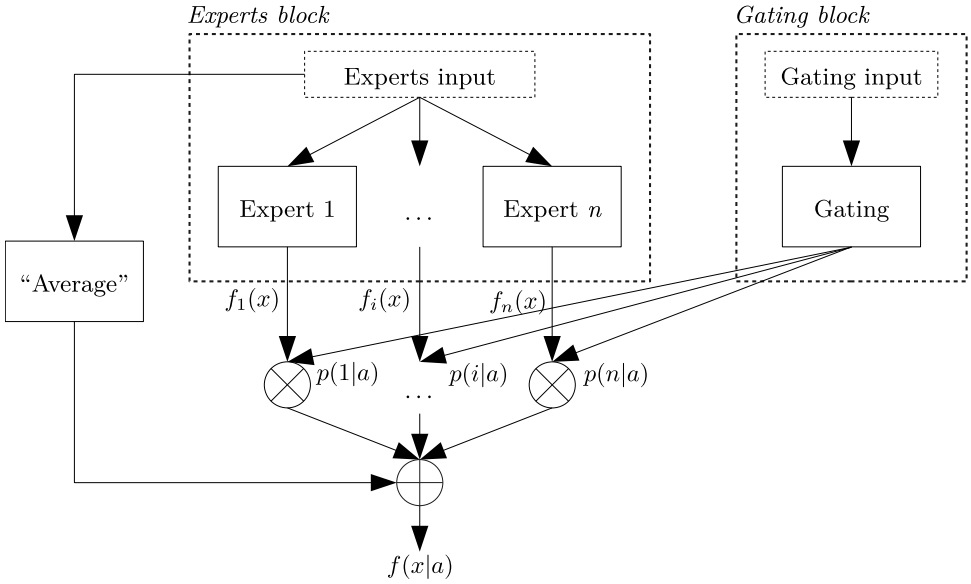


Figure 3.8: Global architecture of the Average Network. The average block learns the core behavior underpinning all investment strategies, and experts learn residuals describing how these strategies differ from the core.

in (Shazeer et al., 2017), is designed to restrict the scope of the gating process by transforming its input so that only the top L values are considered. This function is written:

$$KeepTopL(x, L)_i = \begin{cases} x_i & \text{if } x_i \text{ in top } L \text{ values of } x; \\ -\infty & \text{else.} \end{cases} \tag{3.14}$$

$KeepTopL$ adds sparsity to the network, with $L < n$ and usually set such that $L \ll n$. As we want experts to specialize in the investment strategy of a particular cluster of investors, the $KeepTopL$ function drives the network away from ensembling strategies, i.e., a setting where all investors are dispatched equiprobably amongst all experts, as an investor is restricted to using at most L experts. However, setting $L = 1$ is problematic as a given investor will not sufficiently try out new expert configurations. Consequently, L must be tuned along with the other hyperparameters of the network.

Noise is introduced in the gating procedure through the ϵ term along with the W_{noise} weight matrix to enhance the exploration of expert configurations by a given investor. Since W_{noise} is trainable and shared by all investors' embeddings, it allows for a given investor to set the confidence it has in its attribution to given experts, and change it if not appropriate by setting high levels of noise for irrelevant experts. Borrowing from the reinforcement learning vocabulary, we can say that L enhances *exploitation* of relevant experts, whereas W_{noise} enhances *exploration* of experts attributions.

This gating architecture is the first one that led to successes in both prediction and clustering tasks. However, the design introduces a new hyperparameter, L, which plays a crucial role in the algorithm's potential success. The discontinuity induced by the $KeepTopL$ function makes initialization of the network critical — for a given investor, experts not selected at the beginning of training have a high probability not to be used at all, even though the noise term is supposedly able to counterbalance this effect. L consequently needs to be large enough to cover most experts in the beginning, which contradicts its purpose. The current design, through the

entropy loss, induces sparsity without enforcing it: it is a smoother, more straightforward way to perform the same task that we consequently favored.

3.4.2 About regularization

The regularization terms presented in Sections 3.2.2 and 3.2.3 have known a series of evolutions following the architectural ones seen above. We explore here the evolution of the specialization and sparsity terms that led to their current versions.

Evolution of specialization term

The first version of the specialization term was inspired from the total variation measure (Berestycki, 2016, Section 1.2). Total variation measures the distance between two probability distributions. In a setting where experts output class probabilities, as is the case in the future interests prediction setting, total variation can enforce the differentiation of experts outputs by ensuring large distances between probability distributions. As we want to maximize total variation and as it is bounded by 1, the corresponding loss term we first used was written

$$L_{\text{spec}} = 1 - \frac{1}{n^2|\mathcal{B}|} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{x \in \mathcal{B}} \sum_{c=1}^{C} |f_i^c(x) - f_j^c(x)|, \tag{3.15}$$

where n is the number of experts, $\mathcal{B}$ a batch of data, C the number of output classes and $f_i^c(x)$ the output of expert i for class $c \in [\![1; C]\!]$. This term did not lead to satisfactory results: the L_1 penalization of the difference of the outputs here drives them such that one gets close to 1 and the other close to 0. We deemed that such a penalty term harmed the optimization of the prediction loss, and moved towards smoother versions of this penalty.

A smoother way to enforce differentiation of experts is to penalize their correlations, as exposed in Section 3.2.3. As we want to avoid similar experts, the idea is to penalize correlated ones — the corresponding term was written

$$L_{\text{spec}} = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \max\left(\overline{\rho}_{i,j}, 0\right), \tag{3.16}$$

where all parameters are defined as in Section 3.2.3. We see that this term differs from the current one in two ways: the maximum function and the weighing of the individual terms. The maximum served as a way only to penalize positively correlated experts. Although this perfectly avoids replication, it misses the fact that slightly correlated experts are expected to span the space of investment strategies better. The averaging here is also problematic: a good specialization loss could be obtained by driving unused experts to be negatively correlated. The weights used in the current version correct this by only accounting for used experts, which allows for small clusters to still matter in the specialization penalty.

Evolution of parsimony term

The parsimony term currently used is an entropy loss, as seen in Section 3.2.2. Before using entropy, we tried to enforce sparsity by acting on the variance of the probability distributions outputted by the gating block — the highest possible variance of a probability distribution is attained with a probability of 1 for a class, and 0 everywhere, which is the sparse result we wish to obtain. The corresponding penalty was written:

$$L_{\text{parsimony}} = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \left(\frac{1}{n\sigma_x^2} \right), \tag{3.17}$$

where $\mathcal{B}$ is a batch of data, n the number of experts and σ_x the standard deviation of the gating output of the investor corresponding to sample x. However, the inverse used in this loss term is problematic as the incurred penalty is not linear with regard to the underlying sparsity: distributions far from being sparse would incur a high penalty, whereas distributions closer to it would not matter much anymore in the term. Consequently, distributions trained with such a penalty would not attain the level of sparsity we wished for.

The parsimony term then disappeared with the introduction of the $KeepTopL$ function seen in Section 3.4.1: sparsity was then a direct result of the L hyperparameter, and we consequently abandoned the idea of enforcing it through a penalty term. The term finally reappeared in its current entropic form, which allowed dispensing with the noised gating architecture.

3.4.3 About training

A few tricks were also tried to enhance the training of the network, as it was particularly difficult in the first iterations of the methodology to converge to an ExNet that would be both good at prediction and clustering. Although these tricks disappeared in the current version of the network, we mention them here for the record.

Using pre-computed embeddings

Training a relevant clustering of investors from scratch is a difficult task. A reasonable idea is to initialize these embeddings with meaningful values to help the network learning embeddings leading to clusters that serve the prediction task. We tried using the embeddings of a matrix factorization algorithm trained on the same predictive task, in the fashion of the implicit version of matrix factorization introduced in (Hu et al., 2008) and explained in Section 1.3.3. However, this initialization did not prove useful in helping either the prediction or the clustering tasks and led to similar final results for both.

Alternating tasks training

ExNets solve two distinct tasks at once: a clustering task and a prediction task, both potentially subject to overfitting the training data. Classic regularization techniques will prevent overfitting for the prediction task, but might not be sufficient for the clustering one.

For this reason, we introduce an alternating training schedule. The idea is to split the training data into two parts, both respecting investors' proportions, i.e., an $x\%$ split of the dataset should contain $x\%$ of each investor's data. This ensures that both splits use the same set of investors and that their relative appearance proportions are maintained in both splits. Training is then done according to the following schedule:

1. Train the whole network for an epoch of a given split to *warm* the network *up*.
2. Loop until convergence:

- Train the gating block of the network only on a given split;
- Train the experts block of the network only on the other split.

The goal of this schedule was to improve the generalization of expert repartitions we obtain from the network. Training the network in such a way, however, did not prove useful whatsoever for improving out-of-sample performance on either task.

3.5 Further topics in Experts Networks

Although the ExNet algorithm proved its usefulness, as seen in Section 3.3, there is always room for improvement. We introduce here a couple of enhancements of the algorithm that have shown promise.

3.5.1 Boosting ExNets

Before introducing ExNets in the BNPP CIB RFQ prediction framework, leading algorithms were all gradient tree-boosting models such as XGBoost (Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017). The idea behind gradient boosting is to enhance the capacity of a simple, basis function by adding an ensemble of such functions (Hastie et al., 2009, Chapter 10). Neural networks are consequently not the perfect candidate for boosting, as these models are designed to capture complexity through depth.

Instead of directly plugging a neural network as basis function of a gradient boosting model, it is, however, possible to translate the core ideas behind boosting to a neural network architecture. We consequently designed such an architecture to boost the performance of an ExNet — the global architecture is depicted in Fig. 3.9. Experts blocks can here be thought of as the basis function of the boosting strategy. A single gating network is shared across all boosting stages to allow for a posteriori analysis of the obtained clustering. The output of stage k is written

$$f_k(x|a) = \eta f_{k-1}(x|a) + \sum_i f_{k,i}(x) + p(i|a), \tag{3.18}$$

and all stages incur a loss l_k on their outputs. The whole network is trained in an end-to-end fashion using stochastic gradient descent and backpropagation.

This network has the expected behavior: during training, each boosting stage has lower loss values than its direct predecessor. However, the overall gain of performance over a classic ExNet of this approach is marginal at the expense of far higher computation time. We consequently still favor the original ExNet architecture.

The approach taken for this network is reminiscent of residual networks (He et al., 2016). A residual network is made of residual blocks performing the computation $y = \mathcal{F}(x, \{W_i\}) + x$, where x is the block input, y the block output, $\mathcal{F}$ an operator and W_i, b_i the parameters of the operator — He et al. (2016) use $\mathcal{F} = W_2 f(W_1 x)$, with f an activation function. The initial goal of residual networks is to allow for network growth by tackling the vanishing gradient problems through skip-connections using the identity function. These connections allow for a better flow of the gradient throughout the network. Huang et al. (2018) explore the connection between residual networks and gradient boosting theory, through a BoostResNet model that boosts over features, instead of the classical boosting over outputs. Instead of using boosting for the neural network, a way to enhance the results of the ExNet could be to use residual networks as experts in the network. As residual networks can be interpreted as ensembles of relatively shallow networks (Veit et al., 2016), such a network would be an ensemble... of ensembles.

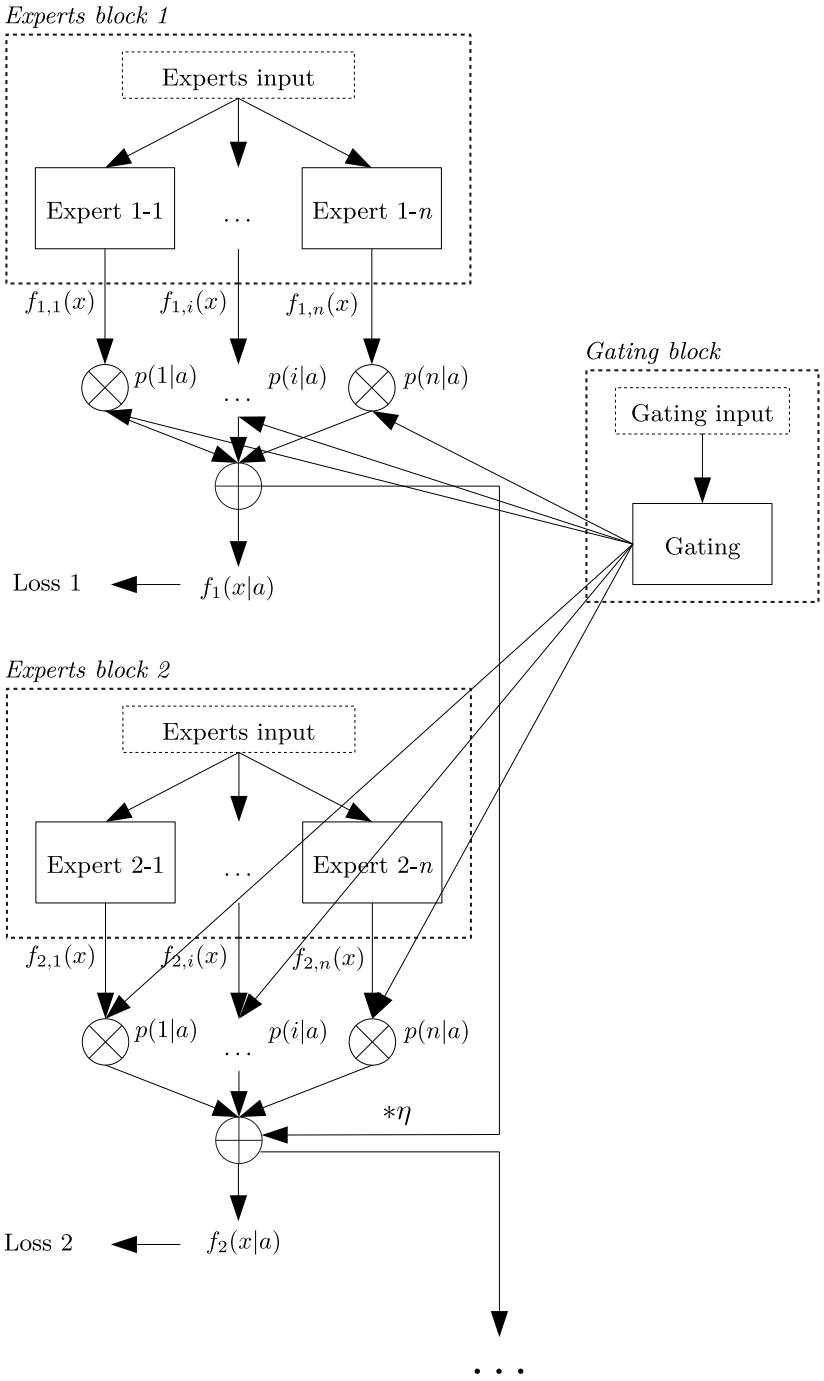


Figure 3.9: Global architecture of a boosted ExNet. A given stage k of boosting is computed using previous stages as $f_k(x|a) = \eta f_{k-1}(x|a) + \sum_i f_{k,i}(x) + p(i|a)$.

3.5.2 Towards a new gating strategy

The gating block of ExNets revolves around the idea of a one-to-one mapping between investors and experts. In the current version of the algorithm, this mapping is encouraged by an entropy loss, as seen in Section 3.2.2. However, the one-to-one mapping could be directly embedded in the network architecture — for a given investor, we could sample the corresponding expert using the categorical distribution defined by this investor's gating output.

Sampling is, however, a non-differentiable operation: we already encountered this problem

with the *KeepTopL* function in Section 3.4.1. The Gumbel-Softmax distribution, introduced simultaneously by Jang et al. (2017) and Maddison et al. (2017), solves this issue by replacing the non-differentiable sample with a differentiable one. The trick is to draw samples $y \in \Delta^{n-1}$, with Δ^{n-1} the $n-1$-simplex and n the number of experts, with y_i defined as

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{n} \exp((\log(\pi_j) + g_j)/\tau)}. \tag{3.19}$$

Here, $g_1, \ldots, g_n$ are i.i.d. samples drawn from a Gumbel(0,1) distribution — Gumbel(0,1) are easily drawn with $g = -\log(-\log(u))$ where $u \sim \mathcal{U}([0,1])$ —, $\pi_1, \ldots, \pi_n$ are the allocation probabilities for the considered investors as obtained with the gating block of the ExNet, and τ is a parameter controlling the temperature of the softmax distribution. The Gumbel-Softmax has the nice property that when $\tau \longrightarrow 0$, the distribution converges to a categorical one. Consequently, smoothly annealing τ during training allows for a quasi-categorical distribution with differentiable samples all along, solving the issues raised in Section 3.4.1. Guo et al. (2019) present an adaptation of this trick in the context of transfer learning.

Gumbel-Softmax has been adapted to ExNets but is still in its infancy for now. It has shown first promising results, leading to relevant yet imperfect one-to-one mappings between investors and experts, but it appears to be very sensitive to the annealing schedule of the τ parameter. The development of a schedule leading to satisfying results will be the subject of further research.

3.6 Concluding remarks

In this chapter, we introduced a novel algorithm, ExNet, based on the financial intuition that in a given market, investors may act differently when exposed to the same signals and form clusters around a finite number of investment strategies. This algorithm performs both prediction, be it regression or classification, and clustering at the same time. The fact that these operations are trained simultaneously leads to a clustering that most closely serves the prediction task and a prediction improved by the clustering. Moreover, one can use this clustering *a posteriori*, independently, to examine how individual agents behave and interact with each other. To help the clustering process, we introduced two additional loss terms that penalize the correlation between the inferred investment strategies and the entropy of the investors' allocations to experts. Thanks to an experiment with simulated data, we proved the usefulness of our approach, and we discussed how the ExNet algorithm performs on an open-source dataset of Spanish stock market data and data from BNP Paribas CIB.

This chapter extended the original article by providing insights into the development of the algorithm, including the difficulties encountered, the failed attempts, and potential leads for further improvements. Further research on ExNets will include how such architectures could be extended and staged and how they could be adapted to retrieve lead-lag relationships in a given market. Finally, it is worth noting that the ExNet architecture can be applied wherever one expects agents to use a finite number of decision patterns, e.g., in e-shopping or movie opinion databases (Bennett et al., 2007).

Chapter 4

Towards time-dependent recommendations

The work exposed in this chapter has been selected as a short paper to the 14th ACM Conference on Recommender Systems, RecSys2020, under the title "History-Augmented Collaborative Filtering for Financial Recommendations".

Chapter 3 tackled the challenge of investors' behavioral heterogeneity. In this chapter, we propose a novel collaborative filtering algorithm that captures the temporal context of a user-item interaction through the user and item recent histories, thereby tackling the non-stationarity of investors' interests with a custom neural network architecture. The performance and properties of the algorithm are monitored in a series of experiments on a G10 bond request for quotation proprietary database from BNP Paribas CIB.

This chapter introduces the context and reach of this work (see Section 4.1), explains in great extent the neural network architecture proposed to meet the targeted goals (see Section 4.2), presents experiments proving the usefulness of the said architecture (see Section 4.3) and elaborates on ways to further the proposed work, along with details about its development and a new use case (see Section 4.4).

4.1 Tackling financial challenges

In this section, we briefly review some of the notions introduced in Chapter 1, expose the challenges we want to tackle with our recommender systems and present recommender systems research related to these challenges.

4.1.1 Context and objectives

When a client wants to trade a financial product, she either requests prices on an electronic platform where many different market makers operate in a process called a *request for quotation* (RFQ), or contact a salesperson of a bank. Reciprocally, salespeople can also directly contact clients and suggest relevant trade ideas to them, e.g., financial products held by the bank and on which it might offer a better price than its competitors. Section 1.1.1 already elaborated on the crucial importance of proactive salespeople for the bank. To support salespeople's proactivity, we can provide them with an RFQ recommender system purposefully designed to assist them in their daily tasks. Consequently, our goal is to design a recommender system that suits the particularities of the financial world. The RFQ recommendation problem is an imperfect information and an implicit feedback problem: we only observe the electronic RFQs performed

on the platforms on which BNP Paribas provides services, and we do not explicitly observe clients' opinions about the products they request. Implicit feedback is a classic recommender system setup already addressed by the research community, e.g., in (Hu et al., 2008). The financial environment, however, brings about specific issues that require attention. To that end, the algorithm we introduce here has three main aims:

- **To incorporate time.** In a classic e-commerce environment, leaving aside popularity and seasonal effects, recommendations provided at a given date may remain relevant for a couple of months, since users' shopping tastes do not markedly evolve with time. In the financial world, a user is interested in a financial product at a given date not only because of the product's intrinsic characteristics but also because of the current market conditions. Time is consequently crucial for RFQ prediction and should be taken into account in a manner that allows for future predictions.
- **To obtain dynamic embeddings.** Being able to capture how clients (resp. financial products) relate to each other and how this relationship evolves with time is of great interest for business, as it allows getting a deeper understanding of the market. One of our goals is consequently to keep the global architecture of matrix factorization algorithms where the recommendation score of a *(client, product)* couple is given by the scalar product of their latent representations.
- **To symmetrize users and items.** Classic recommender systems focus on the client-side. However, the product-side is also of interest in the context of a corporate bank. To control her risk, a market maker needs to control her positions, i.e., make sure she does not hold a given position for too long. Consequently, coming up with the relevant client for a given asset is also important, and the *symmetry of clients and assets* will be integrated here in both our algorithms and evaluation strategies.

We tackle these goals with a context-aware recommender system that only uses client-product interactions as its signal source. Notably, the context considered here is temporal and will be inferred from clients' and products' past behaviors. The terms clients and users (resp. financial products/assets and items) will be used interchangeably to match the vocabulary used in the recommender systems literature.

4.1.2 Related work

Time-dependent collaborative filtering is an active area of research that has known many developments over the years (Shi et al., 2014). Introducing time in matrix factorization was done in Koren (2009b) with the *timeSVD++* algorithm. Ding and Li (2005) propose to downweigh past samples in a memory-based collaborative filtering algorithm. Tensor factorization algorithms can also handle time by considering the user-item co-occurrence matrix as a three-dimensional tensor, where the additional dimension corresponds to time, e.g., in Xiong et al. (2010) and Wu et al. (2018), the latter using neural network architectures. However, algorithms relying on matrix or tensor factorization do not allow for making predictions in the future. Enhancing latent factor models with Kalman filters as in Sarkar et al. (2007) or with LSTM cells (Hochreiter and Schmidhuber, 1997) to capture the dynamics of rating vectors as in Wu et al. (2017) alleviates this issue. Using historical data to introduce dynamics in a neural network recommender system was done in Covington et al. (2016), where the authors assimilate YouTube users to their viewing histories.

The algorithm introduced in this paper is, to some extent, reminiscent of graph neural networks (Hamilton et al., 2017) and their adaptation to recommendation (Wang et al., 2019), where one would stop at first-order connectivity. Using time to enhance recommendations with random walks on bipartite graphs was explored in Xiang et al. (2010). How graph neural networks behave with dynamic bipartite graphs is to the best of our knowledge yet to be discovered and

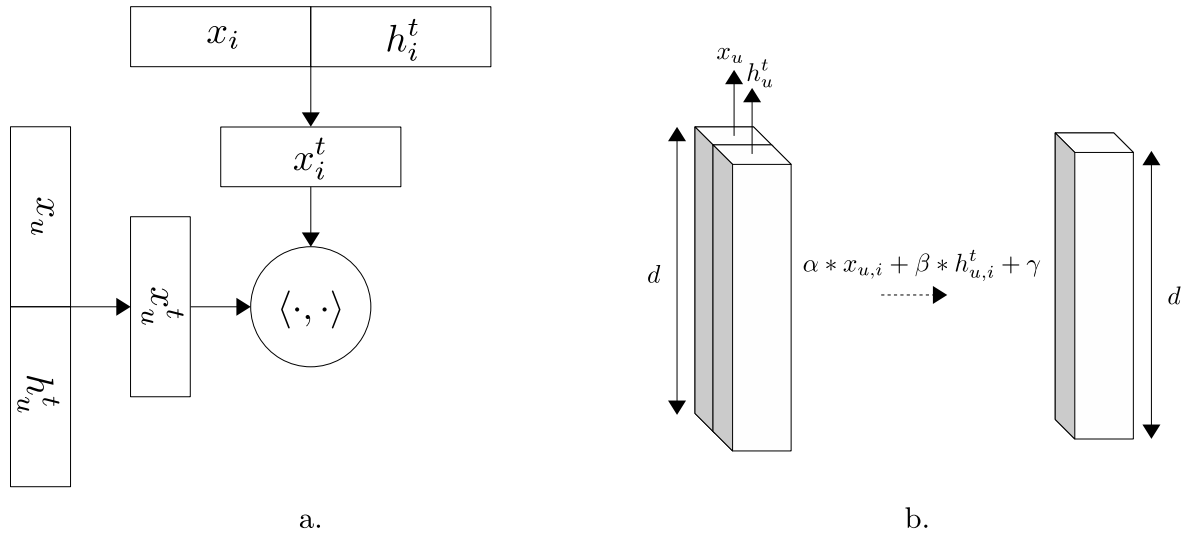
could lead to an extension of this work. A first attempt at financial products recommendation was made in Wright et al. (2018), with the particular example of corporate bonds.

4.2 Enhancing recommendations with histories

We introduce a neural network architecture that aims at producing recommendations through dynamic embeddings of users and items, that we call *History-augmented Collaborative Filtering* (HCF).

4.2.1 Some definitions

Let U be the set of all users and I the set of all items. Let us note $x_u \in \mathbb{R}^d$ for $u \in U$, $x_i \in \mathbb{R}^d$ for $i \in I$ the d-dimensional embeddings of all the users and items we consider. Let $t \in [\![0\ ; \infty]\!]$ be a discrete timestep, taken as days in this work. For a given u, at a given t, we define $\mathbf{h_u^t}$ as the items' history of u, i.e. the set of items found to be of interest to u in the past — we respectively define $\mathbf{h_i^t}$ as item i users' history. We use here as histories the last n events that happened strictly before t to either user u or item i. If at a given t we observe $0 \leq n' < n$ previous events, histories are only formed of those n' events. By doing so, we place users on the same event scale — high-activity users will have histories spanning a couple of days, whereas low-activity ones will span a couple of months (resp. for items). Histories formed of items/users of interest in a past window of fixed size were also tried but led to inferior performance than the previous history definition.



a. b.

Figure 4.1: **a.** Global architecture of the HCF network, composed of two symmetric user and item blocks. **b.** Illustration of the application of a one-dimensional convolution of kernel size 1 along the embedding dimension axis of our inputs x_u and h_u^t for a user $u \in U$, where α, β, γ are the parameters of the convolution filter and $l \in [\![1; d]\!]$. The same computation holds for items $i \in I$ respectively.

4.2.2 Architecture of the network

Figure 4.1.a shows the global architecture of the HCF network. It is composed of two symmetric blocks — a *user block* and an *item block*. The core components of these blocks are one-dimensional convolutions of kernel size 1 which compute dynamic embeddings of users (resp. items). The user block of HCF uses the embeddings of users x_u and the corresponding histories' average embeddings

$$h_u^t = \frac{1}{|\mathbf{h_u^t}|} \sum_{i \in \mathbf{h_u^t}} x_i \tag{4.1}$$

at time t as inputs, and respectively for items. If $\mathbf{h_u^t}$ is empty, we use $h_u^t = 0$. Convolutions are performed along the embedding dimension axis, considering user and history embeddings as channels (see Fig. 4.1.b). Convolutions were chosen because we empirically found that all architectures performing computations involving both $x_{u,l}$ and $h_{u,l'}^t$ with $l \neq l' \in [\![1; d]\!]$ the l-th component of the embedding systematically led to poorer performance than a linear component-wise combination of x_u and h_u^t. One-dimensional convolutions of kernel size 1 along the embedding size axis can be seen as a variation of the linear component-wise combination with shared parameters across all components, thus allowing for network depth.

4.2.3 Optimizing HCF

The network is trained using the *bayesian personalized ranking* (BPR) loss (Rendle et al., 2009), a surrogate of the ROC AUC score (Manning et al., 2010) already introduced in Section 1.3.3. It is defined in (Rendle et al., 2009) as

$$L_{BPR} = - \sum_{(u,i,j) \in D} \ln \sigma(x_{uij}), \tag{4.2}$$

where

$$D = \left\{ (u, i, j) | i \in I_u^+ \wedge j \in I \backslash I_u^+ \right\} \tag{4.3}$$

with I_u^+ the subset of items that were of interest for user u in the considered dataset, and $x_{uij} = x_{ui} - x_{uj}$, with x_{ui} the score of the (u, i) couple. σ is the sigmoid function, defined as $\sigma(x) = 1/(1 + e^{-x})$. The underlying idea is to rank items of interest for a given u higher than items of no interest for that u, and D corresponds to the set of all such possible pairs for all users appearing in the considered dataset. D grows exponentially with the number of users and items considered. It is consequently usual to approximate L_{BPR} with negative sampling (Mikolov et al., 2013).

In our proposed methodology, scores become time-dependent. Data samples are therefore not seen as couples, but as triplets (t, u, i). To enforce user-item symmetry in the sampling strategy, we define the sets

$$D_u^t = \left\{ (t, u, i, j) | i \in I_u^{t,+} \wedge j \in I \backslash \mathbf{h_u^t} \right\} \tag{4.4}$$

$$D_i^t = \left\{ (t, u, v, i) | u \in U_i^{t,+} \wedge v \in U \backslash \mathbf{h_i^t} \right\} \tag{4.5}$$

with $I_u^{t,+}$ the subset of items that had a positive interaction with user u at time t, and $U_i^{t,+}$ the subset of users that had a positive interaction with item i at time t. For a given positive triplet (t, u, i), we sample either a corresponding negative one in D_u^t or D_i^t with equal probability. Note that considering samples as triplets adds a sampling direction, as a couple that was active at a time t may no longer be active at other times $t + t'$ or $t - t''$, $t', t'' > 0$. Such sampling strategies will be more extensively studied in further iterations of this work.

4.2.4 Going further

HCF retains a matrix factorization-like architecture to allow for using dynamic embeddings of users and items *a posteriori*, e.g., for clustering, visualization, analysis of the evolution of relationships with time,... Note that the network outlined in this article only uses the user-item interactions signal. More complex features could be used in the user (resp. item) block, as long as they only depend on users (resp. items). Both blocks would then require encoders to map user and item embeddings to the same space.

Using features could also alleviate the well-known cold-start problem (see (Koren et al., 2009), or Section 1.3.5). This problem can arise for both users and items and is mainly targeted at users in the literature. However, new clients are not frequent in the CIB business. The item cold-start problem is a more important challenge to address in a financial recommender system, as new financial products constantly appear in the markets. With the outlined methodology, such products could not be scored directly. Nevertheless, some financial assets, and more particularly bonds, can be defined by the collection of their characteristics: embedding these characteristics instead of the bonds themselves would allow for forming scores for any bond at any time. This approach is explored further in Section 4.4.3.

4.3 Experiments

We conduct a series of experiments to understand the behavior of the proposed HCF algorithm on a proprietary database of RFQs on governmental bonds from the G10 countries. This database accounts for hundreds of clients and thousands of bonds and ranges from 08/01/2018 to 09/30/2019. We examine here the performance of our proposal in comparison to benchmark algorithms in two experiments. Benchmark algorithms, chosen for their relative respect of the aims outlined in Section 4.1, are a historical baseline and two matrix factorization algorithms trained with objectives corresponding to (Hu et al., 2008) and (Rendle et al., 2009), that we respectively call *MF - implicit* and *MF - BPR*. Further details about these models can be found in Section 1.3 and in Section 2.2. We adopt for MF - BPR the symmetrical sampling strategy outlined in Section 4.2. The historical baseline scores a *(user, item)* couple with their observed number of interactions during the training period considered.

In this section, the performance of our models is evaluated using mean average precision (mAP) (Manning et al., 2010), defined as

$$\text{mAP} = 1/|Q| * \sum_{q \in Q} \text{AP}(q) \tag{4.6}$$

where Q is the set of *queries* to the recommender system, and $\text{AP}(q)$ is the average precision score of a given query q. To monitor the performance of our algorithms on both the user- and item-sides, we define two sets of queries over which averaging:

- ***User-side queries.*** Queries correspond to the recommendation list formulated every day for all users;
- ***Item-side queries.*** Queries correspond to the recommendation list formulated every day for all items.

These two query sets lead to user- and item-side mAPs that we summarize with a harmonic mean in a symmetrized mAP score used to monitor all the following experiments, as

$$\text{mAP}_{sym} = \frac{2 * \text{mAP}_u * \text{mAP}_i}{\text{mAP}_u + \text{mAP}_i} \ . \tag{4.7}$$

See Section 1.4 for further details on this score. The user and item perimeters considered for the computation of the mAP scores are all the potential couples formed of all the users and items encountered during the training period[1]

4.3.1 Evolution of forward performance with training window size

This experiment aims at showing how benchmark algorithms and our HCF proposal behave with regards to stationarity issues. We already advocated the importance of time in the financial setup — taking the user side, clients behavior is *non-stationary*, owing to the non-stationarity of the financial markets themselves but also externalities such as punctual needs for liquidity, regulatory requirements, ... In machine learning, this translates into the fact that the utility of past data decreases with time. However, machine learning and more particularly deep learning algorithms work best when provided with large datasets (LeCun et al., 2015): there is an apparent trade-off between non-stationarity and the need for more training data. Our goal in this experiment is to show that introducing time in the algorithm reduces this trade-off.

To prove this, we examine the evolution of forward performance as the training window size grows. We split the G10 bonds RFQ dataset into three contiguous parts — a train part that ranges from up to 08/01/2018 to 07/31/2019 (up to one year), a validation part from 08/01/2019 to 08/30/2019 (one month) and a test part from 09/01/2019 to 09/30/2019 (one month). Validation is kept temporally separated from the training period to avoid signal leakages (De Prado, 2018). For all the considered algorithms, we train an instance of each on many training window sizes ranging from a week to a year using carefully hand-tuned hyperparameters and early stopping, monitoring validation symmetrized mAP.

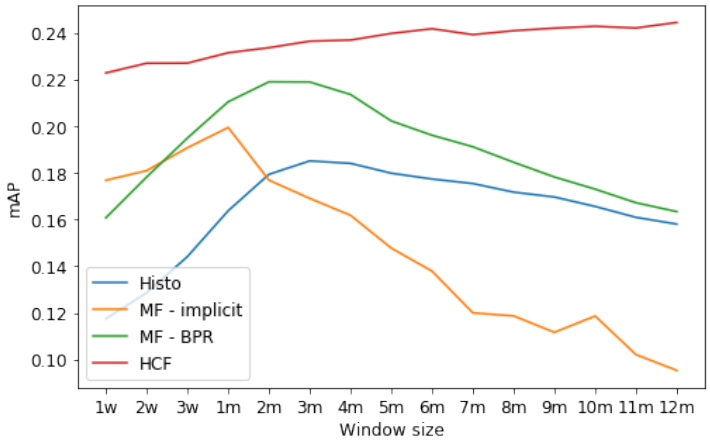


Figure 4.2: Evolution of validation symmetrized mAP with training window size. Whereas benchmark algorithms seem to have an optimal training window size, our HCF proposal keeps improving with the training window size.

We see in Fig. 4.2 that all benchmark algorithms present a bell-shaped curve. They attain a peak after which their performance only degrades as we feed these models more data, a behavior which corroborates the non-stationarity vs. amount of data trade-off. On the contrary, HCF

[1]This scoring perimeter proved to be the fairest with regard to all considered models, as a model is consequently scored only on what it can score and nothing else. Fixing the perimeter for all algorithms to all the couples encountered in the maximal training window and attributing the lowest possible score to otherwise unscorable couples only lowers the mAP scores of candidate algorithms that cannot obtain satisfactory results on such window sizes, as the added couples are not frequently observed as positive events. This phenomenon is observed in Section 2.2.5.

only gets better with training data size. Notably, HCF 12m, which obtained best validation performance, used $n = 20$ and blocks with two hidden layers and ReLU activations.

To show that these bell-shaped curves are not an artifact of the hyperparameters chosen in the previous experiment, we conduct a systematic hyperparameter search for multiple training window sizes using a combination of a hundred trials of random search (Bergstra and Bengio, 2012) and hand-tuning. The optimal sets of hyperparameters for each considered window size are then used as in the previous experiment to obtain graphs of their validation mAP scores against the training window size. Results are shown in Figure 4.3. MF - BPR 6m and 12m optimal hyperparameters happened to coincide, and their results are consequently shown on the same graph.

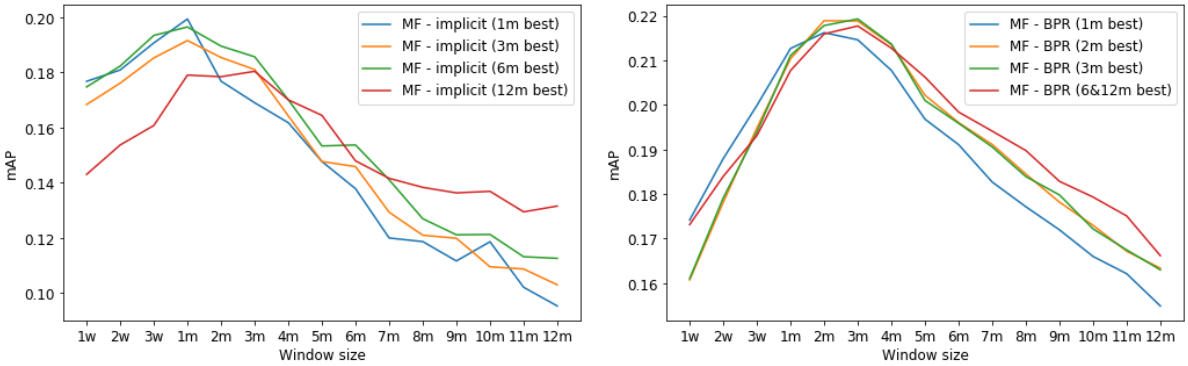


Figure 4.3: Evolution of validation symmetrized mAP with training window size. *Left:* Optimal MF - implicit for the 1m, 3m, 6m and 12m windows. A consensus arises around the 1m window. *Right:* Optimal MF - BPR for the 1m, 2m, 3m, 6m and 12m windows. A consensus arises around the 2m-3m windows, slightly favoring the 2m window.

We see here that for both MF - implicit and MF - BPR, hyperparameters optimized for many different window sizes seem to agree on optimal window size, respectively around one month and two months, with slight variations around these peaks. Consequently, bell shapes are inherent to these algorithms, which proves their non-stationarity vs. data size trade-off.

To obtain test performances that are not penalized by the discontinuity of the training and test windows, we retrain all these algorithms with the best hyperparameters and window size found for the validation period on dates directly preceding the test period, using the same number of training epochs as before. The performances of all the considered algorithms are reported in Table 4.1.

Table 4.1: Window size study — symmetrized mAP scores, in percentage.

Algorithm	Window	Valid mAP	Test mAP
Historical	3m	18.51	16.86
MF - implicit	1m	19.94	19.24
MF - BPR	2m	21.89	20.05
HCF	12m	**24.43**	**25.02**

We see here that our HCF algorithm, augmented with the temporal context, obtained better performances on both validation and test periods than the static BPR and implicit MF algorithms. Consequently, introducing time is essential to obtain better performances in the financial setup that we consider.

4.3.2 Evolution of forward performance with time

It follows from the experiment conducted in Section 4.3.1 that our benchmark algorithms cannot make proper use of large amounts of past data and have to use short training window sizes to get good performances compared to historical models. Moreover, the results of these algorithms are less stable in time than HCF results. Fig. 4.4 visualizes the results reported on Table 4.1 on a daily basis for all the considered algorithms.

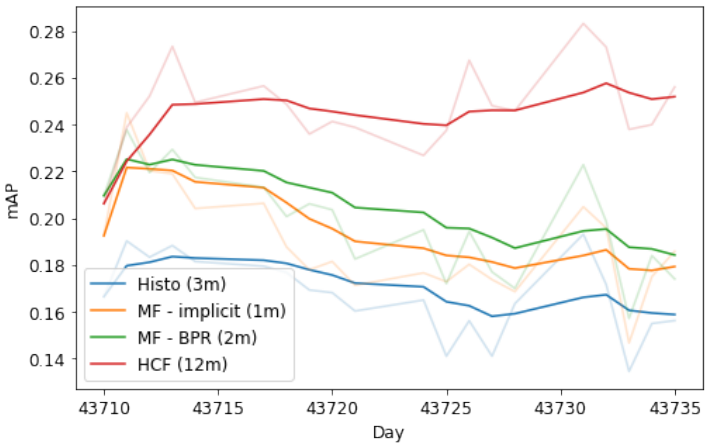


Figure 4.4: Daily evolution of symmetrized mAP during the test period. Light shades correspond to the true daily symmetrized mAP values, and dark ones to exponentially weighted averages ($\alpha = 0.2$) of these values.

We see a downward performance trend for all the benchmark algorithms — the further away from the training period, the lower the daily symmetrized mAP. On the contrary, HCF has stable results over the whole test period: introducing temporal context through user and item histories hinders the non-stationarity effects on the model's forward performances.

The results from Section 4.3.1 and the observed downward performance trend consequently suggest that benchmark models need very frequent retraining to remain relevant regarding the future interests of users (resp. items). A simple way to improve their results is to retrain these models daily with a fixed, sliding window size w — predictions for each day of the testing period are made using a model trained on the w previous days. Each model uses here the number of epochs and hyperparameters determined as best on the validation period in the previous experiment. Results of these *sliding* models are shown in Table 4.2, where HCF results correspond to the previous ones.

Table 4.2: Sliding study — symmetrized mAP scores, expressed in percentage.

Algorithm	Window	Test mAP
Historical (sliding)	3m	20.32
MF - implicit (sliding)	1m	24.27
MF - BPR (sliding)	2m	24.46
HCF	12m	**25.02**

We see that both implicit and BPR matrix factorizations significantly improved their results compared to their static versions from Table 4.1, but are still below the results of HCF trained on 12 months. Consequently, our HCF proposal is inherently more stable than our benchmark algorithms and captures time in a more efficient manner than their daily retrained versions.

4.3.3 Evolution of forward performance with history size

To conclude our experiments, we examine how HCF's performance evolves as we change the number of events n considered in history. Histories of size $n = 20$ were found to be optimal in validation in the experiment of Section 4.3.1. Our goal in this experiment is to examine how the performance of HCF evolves with this history size, all other hyperparameters being equal. The HCF algorithm was trained here in the same setup as the one leading to the test performances from Table 4.1 with history size as the only free hyperparameter, investigated in the $n \in [\![5; 50]\!]$ range. Results are shown in Figure 4.5.

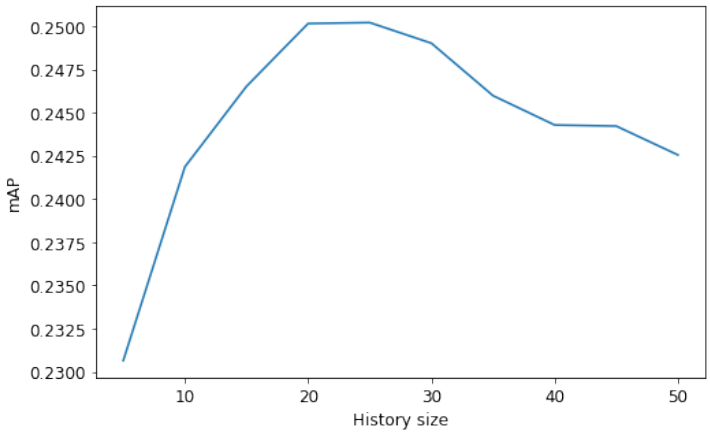


Figure 4.5: Evolution of HCF test performance with history size, all other hyperparameters kept fixed.

It appears from this graph that, all other hyperparameters being equal, there is an optimal history size of around $n = 20$, which matches the optimal one found in the experiment of Section 4.3.1. This result shows that HCF uses histories as a rather recent summary of the behavior of a user/item. However, depending on the user or item, "recent" does not necessarily mean the same thing — for low-activity users, $n = 20$ can span weeks, whereas it does not even span a day for the high-activity ones. Consequently, it would appear that personalizing n such that we use a distinct n_*, $* \in \{u, i\}$, for all users and items could help improve the results of HCF. How this personalization could be done will be the subject of further research.

4.4 Further topics in history-augmented collaborative filtering

The previous sections introduced the HCF algorithm and presented experiments proving its utility for the problem of future interest prediction. In this section, we examine supplementary material and experiments that extend and complete the notions previously introduced in this chapter.

4.4.1 Sampling strategies

Negative samplers for BPR have been the subject of numerous research attempts. Notably, Ding et al. (2019) propose to use a different, restricted set of items from which to sample for every user and demonstrate that sampling negatives from the whole item space is not necessary to achieve good performance. Zhang et al. (2013) introduce a dynamic negative sampler that samples with more probability negative items ranked higher than positive ones — a related idea is presented in Section 5.2.1. Section 4.2.3 introduces a symmetrized, history-aware sampling strategy for attributing negative samples to the observed positive ones. Along with the development of HCF, we tried many different variations of the sampling strategy. We present five different strategies in this section, all following the user-item symmetry of Section 4.2.3:

- *Naive sampling.* The naive strategy uniformly draws negative samples from the entire user (resp. item) pool, giving no consideration whatsoever to positives.
- *Weighted sampling.* The weighted strategy extends naive sampling by attributing sampling weights to users (resp. items) corresponding to their observed popularity in the training set. The idea is to compare positives to "relevant" negatives, where relevance is defined by popularity.
- *History-aware sampling.* The history-aware strategy, presented in Section 4.2.3, uniformly draws negative samples from the set of items (resp. users) that were not observed in the current history of the considered user (resp. item). The idea is to restrain negatives from being drawn among recent positives.
- *Backward sampling.* The backward sampling strategy uniformly draws negative samples from the pool of users (resp. items) that were positive in the past n days. Taking the user-side, the idea is to compare a positive (t, u, i) triplet with a negative (t, u, j) triplet that is very likely to be of interest as well, and consequently focus on the fact that user u (deliberately or not) chose i instead of j, a positive-negative pair that we deem to bear more information than a naive one.
- *Temporal sampling.* As HCF considers (t, u, i) triplets, time is a sampling direction that we can also consider among the user and item directions. The temporal sampling strategy uniformly draws a past date $t' < t$ for a positive (t, u, i) triplet, and consider as negative the (t', u, i) triplet. The idea is to compare a couple (u, i) at times when it was positive and negative and therefore where historical contexts were different: it focuses on the fact that the couple was positive at this particular t and (most likely) not at t', resulting in an expectedly more informative pairing than a naive one.

As of now, the sampling strategies that were observed to be the most efficient are the naive and history-aware ones. Even more, the backward and temporal strategies failed to converge, as well as the weighted strategy, depending on the chosen weighing scheme. A potential explanation for this non-convergence is that BPR optimizes a relaxation of the AUC score, a score which weighs equally all potential pairings of positive and negative events (see Section 1.4.2). The failing strategies deplete the pool of potential negatives up to a point where, in the temporal strategy, no negative user or item is shown to a given positive triplet: only a small part of all potential pairs is considered, leading to poor convergence. However, these sampling strategies try to focus on more informative triplets to pair with positive ones. Consequently, we could expect a combination of multiple sampling strategies to perform better than a given strategy alone. Such combinations will be the subject of further research on HCF.

4.4.2 Evolution of scores with time

To examine the behavior of the HCF algorithm, we can monitor the evolution of scores on a more microscopic way than through global performance metrics such as symmetrized mAP. The density of scores outputted by the HCF 12m model leading to the test performance shown in Section 4.3.1 is shown in Fig. 4.6. We see that predicted scores seem to follow a normal distribution, with outlying scores going up to 5.6. In the methodology introduced in 4.2, there is no architectural choice constraining the admissible range of scores. This could be easily corrected with a final score outputted by

$$f(\langle x_u^t, x_i^t \rangle) \tag{4.8}$$

with f a squashing function such as the sigmoid or hyperbolic tangent functions.

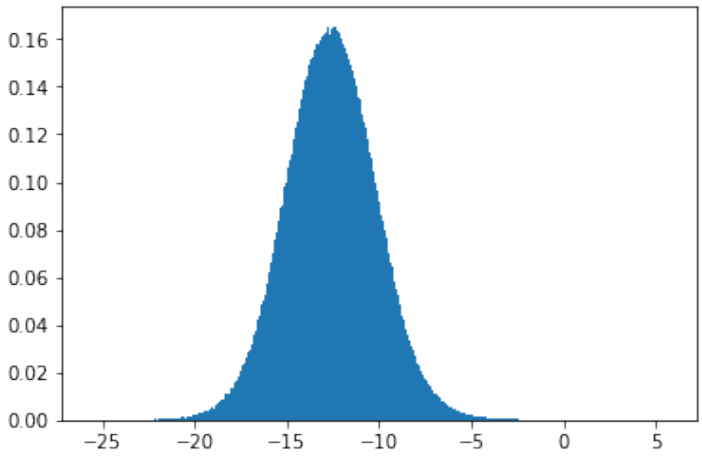
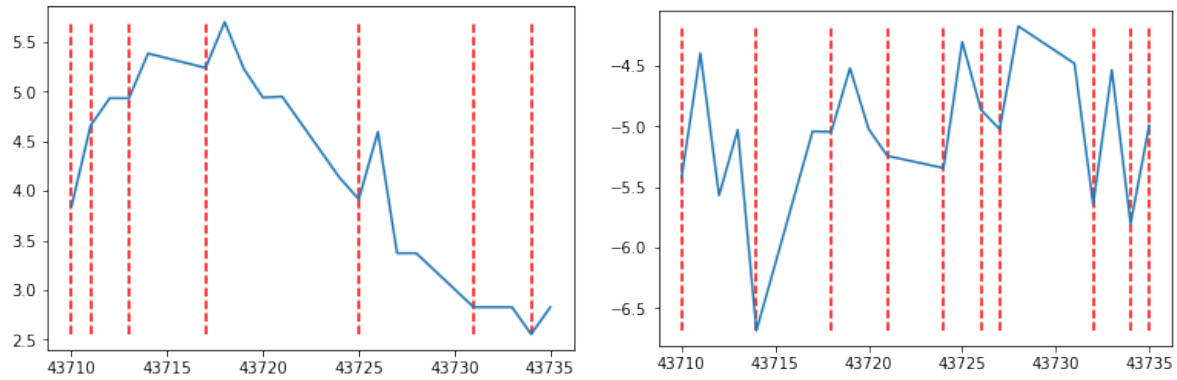


Figure 4.6: Density of scores. On the considered test period, predicted scores range from −25.6 to 5.6.

Let us now look at the particular scores we obtain for given *(user, item)* couples over time. Figure 4.7 examines the case of a medium-activity user and a high-activity one on the same item, and Figure 4.8 the case of a low-activity user on two distinct items. The first thing we can note in Fig. 4.7 is that the scoring scale of a particular user changes a lot — in Fig. 4.7**(a)** scores range between 2.5 and 5.6 on the considered period, and in Fig. 4.7**(b)** scores range between −6.5 and −4.3, even though the user considered in **(b)** is reputedly more active. Consequently, it seems that scores are not comparable for two particular users in the proposed methodology — a characteristic that the use of a squashing function for scores can correct. Moreover, we see in Fig. 4.7**(a)** that scores increase every time the user declares an interest in the considered product — integrating the product in the user's history consequently helps driving scores up for that particular product, a characteristic we expected from our methodology. The same only partly holds for the high-activity user of Fig. 4.7**(b)**. As histories are restricted to the n last interests of a user, a highly-active one will have a fast history turnover, hence a potential fast drop in interest for the considered item. Finally, we see that even though the scored item is the same for both users, the medium-activity user's score decreases over time, whereas the high-activity one seems to be mean-reverting.

Figure 4.8 illustrates the evolution of scores with time for a same low-activity client on two different items. In Fig. 4.8**(a)**, we see that the score of the item increases considerably after the reported trade. The score then drops a few days after and stabilizes at a higher value than before the trade. If Fig. 4.7**(b)** explored the case of a frequent history turnover, this figure illustrates the opposite: the drop in score can be imputed to a trade on another item, and the



(a) **(b)**

Figure 4.7: Evolution of scores with time. Dates are in Excel format, and dotted vertical lines represent true events during the considered period on the considered *(user, item)* couple. Here, the same item is considered with two distinct users. **(a)** This figure illustrates the case of a medium-activity client. We see that the score increases every time the item is traded, and that it globally decreases over time. **(b)** This figure illustrates the case of a high-activity client. We see that the score most often increases when the item is traded, and can decrease after a trade, a behavior we impute to the frequent history turnover of high-activity clients. On a global scale, the score has a mean-reverting behavior.

stable behavior to a quasi-frozen history. The same behavior can also be seen in Fig. 4.8**(b)**. However, the score seems to be more unsteady — this behavior can be imputed to the fact that the considered item is more active than the previous one, and consequently has a faster-evolving embedding due to its faster history turnover.

These four examples allow us to delve further into the microscopic behavior of the HCF algorithm. Notably, we see the influence of histories on the predicted scores — including items (resp. users) in the users' (resp. items') histories drives scores higher for these items. Consequently, these graphs validate the methodology pursued with HCF: enhancing static embeddings with histories lead to dynamic scorings that evolve accordingly with histories' contents.

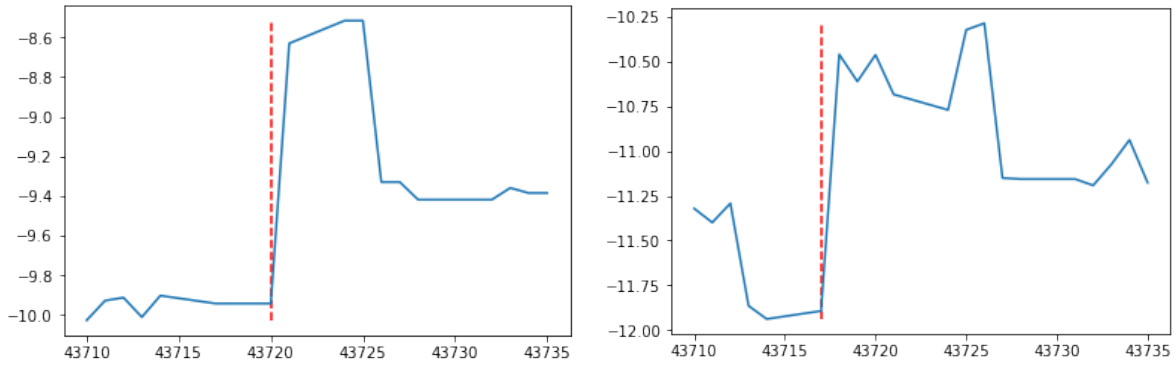
4.4.3 Solving cold-start

Section 4.2.4 provided a first insight into ways to solve the cold-start problem with the proposed methodology. In this section, we develop in greater extent how to solve cold-start and present results on the dataset studied in the experiments of Section 4.3. Note that this section focuses on item cold-start, but the presented methodology is also applicable to user cold-start.

Our goal is to replace the static embedding used to represent items with an embedding obtained through numerical and categorical features describing the state of the item at time t. Consequently, the dynamics in the network might now come from both the item latent representations, e.g., through market-related numerical features, and the usage of histories. Calling $f_{num,*}$ and $f_{cat,*}$ respectively the numerical features and the embeddings associated to the categorical features related to $* \in \{u, i\}$ and keeping the notations introduced in Section 4.2, we now use

$$x_* = W_*([f_{num,*}, f_{cat,*}]) \tag{4.9}$$

as embedding for $*$, where $[\cdot, \cdot]$ is the concatenation operator and $W_* \in \mathbb{R}^{N \times d}$ is a projection matrix, with $N = (|f_{num,*}| + \sum_l \dim(f_{cat,*,l}))$ is the size of the featurized representation, d the target embedding size and $textdim(f_{cat,*,l})$ the dimension of the l-th category's embedding.



(a) **(b)**

Figure 4.8: Evolution of scores with time. Dates are in Excel format, and dotted vertical lines represent true events during the considered period on the considered *(user, item)* couple. Here, the same user is considered with two different items. **(a)** This figure illustrates the case of a low-activity client on a given item. The score increases considerably after the trade, and stabilizes to a higher value than before the trade a few days later. **(b)** This figure illustrates the case of a low-activity client on a more active item than in **(a)**. The score increases after the trade as well, and is more unsteady due to the history turnover of the scored item.

Note that this formulation does not prevent from using a static embedding directly related to the identity of $*$, as an encoding of the identity could be included in the categorical features describing $*$. All embeddings of users, items, and their histories are computed in this manner.

Let us consider a simple version of this "featurized" HCF algorithm. As item cold-start is the main concern in our setting, we keep for users the previously introduced static embeddings. In this version, items are represented by a collection of numerical and categorical features:

- **Numerical features.** We use as numerical features the annualized time to maturity and time since issuance, and the current coupon rate of the bond.
- **Categorical features.** We use as categorical features embeddings of the bond's ticker, the emission currency, the coupon type, the bond's rating, and the corresponding bucket of issued amount. Sizes of these embeddings are hyperparameters of the network.

We train this algorithm on the same training and validation ranges and with the same hyperparameters than the 12m HCF of Section 4.3.1, and obtain test performance in the same manner. However, due to technical difficulties related to the ways we associate features to the historical elements, we resort here to the naive sampling strategy described in Section 4.4.1. Results are shown in Table 4.3. We see that the performance of the featurized version of HCF is slightly below the "classic" one in this setting, but still above the performance of the sliding algorithms introduced in Section 4.3.2. We only use here a small number of features to characterize bonds — adding features related to market data would most certainly help enhance performances.

Interestingly, we remarked that validation loss was more unsteady in the featurized version than the classic one when training the algorithm. As we use a restricted set of features to characterize items, the representation of a negatively sampled item might be close to a positive one, leading to ambiguous positive-negative comparisons. Consequently, a richer set of features would lead to improved performances and a more stable training.

Moreover, these two versions of HCF can be compared using the qualitative metrics introduced in Section 1.4.3. The results are shown in Table 4.4. For a reminder, diversity measures the mean degree of the objects present in the top-L recommendation lists of the considered side

Table 4.3: Comparison of classic and featurized versions of HCF — symmetrized mAP scores, in percentage.

Algorithm	Valid mAP	Test mAP
HCF	**24.43**	**25.02**
featurized HCF	23.92	24.90

— i.e., D_{user} measures the mean degree of the top-L *items* recommended — and similarity measures the overlap between the top-L recommendation lists of the considered side — i.e., S_{item} measures the average percentage of common *users* between the top-L recommendation lists of items. We see in Table 4.4 that the featurized HCF has more diverse recommendations and less similar recommendation lists on both the user and item sides. Consequently, using "featurized" representations of items improves the diversity of the provided recommendations.

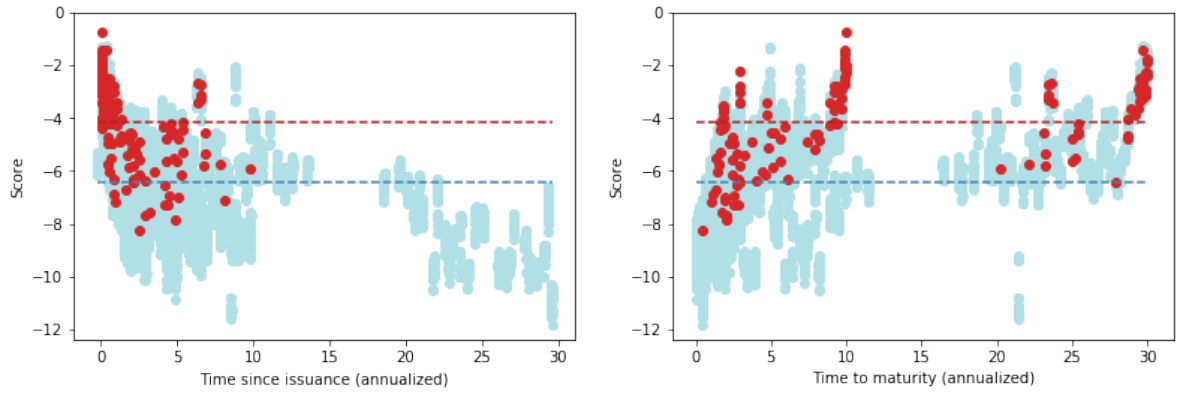
Table 4.4: Comparison of classic and featurized versions of HCF — diversity (D) and similarity (S) metrics obtained on the test set. Degrees taken into account for diversities are computed on the training set. Metrics computed using $L = 20$.

Algorithm	D_{user}	D_{item}	S_{user}	S_{item}
HCF	94.7	211.5	17.2%	21.0%
featurized HCF	**85.3**	**195.8**	**11.3%**	**16.9%**

Using features to represent items paves the way to new potential representations of scores, extending the work presented in Section 4.4.2. Figure 4.9 illustrates the evolution of scores for a particular user with the values of two numerical features: time since issuance, and time to maturity. Plotted scores correspond to the test period, defined in Section 4.3.1 as ranging from 09/01/2019 to 09/30/2019. The chosen user is the most active one during this period, and items plotted in this figure are American Treasury bonds — we see that a gap appears on both figures, originating from the fact that Treasury bonds are emitted every five years. We see that both Fig. 4.9**(a)** and Fig. 4.9**(b)** present an S-shaped score evolution: the most attractive bonds are the newly emitted ones, and close-to-maturity bonds are the least appealing for the considered user. We see that the predicted scores place positive events on average above negative ones, as shown by the dotted lines representing the attributed average scores to positive and negative events. Interestingly, the considered user showed interest in less than 10-years-old bonds only but spanned almost entirely the range of maturities.

If the plots from the classic version of HCF shown in 4.4.2 help us validate the expected behavior of the algorithms, the featurized version helps us uncover more information about the behavior of particular investors on the markets. Consequently, featurized HCF has two main advantages over its classic version. First, featurized HCF solves the cold-start problem — using features to represent users and items allows for computing scores for any couple at any time, provided that these new users and items can be defined in terms of the features used to train the HCF model. Second, featurized HCF allows for richer a posteriori analyses of scores than the ones permitted with the classic version and is thus of greater interest to understand the business.

On a final note, let us elaborate on *user cold-start*. We already advocated in Section 4.2.4 that the user cold-start is not a relevant problem in finance, as corporate banks such as BNP Paribas deal with a rather fixed base of investors. Moreover, if we wanted to tackle user cold-start, fully "featurized" user representations would not solve the problem entirely as a unique representation of users as features, at the exclusion of the user's identity, is difficultly achievable. Indeed, the potential features we can use to represent users include, e.g., numerical data about invested amounts, size of the user's portfolio, the user's sector of activity, the user's region... which



(a) **(b)**

Figure 4.9: Evolution of American Treasury bonds scores during the test period for the most active user with two different features. Figure **(a)** plots the evolution of score with the annualized elapsed time since issuance, and Figure **(b)** plots the evolution of score with the annualized remaining time to maturity. Red dots correspond to positive events on the considered period, and blue ones to negative events. The red and blue horizontal dotted lines respectively represent the average score of positive and negative events. Both figures are S-shaped, showing that newly emitted bonds are the most attractive and close-to-maturity ones the least attractive. On average, positive events were scored higher than negative ones.

are not sufficient to differentiate all users. However, these features can help us hinder the effect of the clients' heterogeneity (see Section 1.2.2) on the network's training. Using the notation introduced above, we define user embeddings as $x_u = W_u([f_{num,u}, f_{cat,u}])$, with the users' identities included among others in the considered categories of $f_{cat,u}$.

Such user embeddings share many parameters: embeddings corresponding to a class of a given category, e.g., European investors, will be trained every time a European investor appears in training, and all users share the W_u weights projecting the embedding to the targeted size. Contrarily to an algorithm which would only embed users' identities, these featurized user representations have shared elements across users. A low-activity client, which embedding will only be trained when the client is encountered during training, would consequently obtain a more informative representation with the featurized version. Proving that low-activity users get improved performance from such a strategy is yet to be done and will be the subject of further research.

4.4.4 Application to primary markets of structured products

Section 1.1.2 explores bonds and options, and gives insights into their potential interests to investors. However, as appealing as these financial products can be, they might not suffice to meet the specific needs of some investors. When investors want to invest in products tailored to their current situation, they have to rely on *structured products*. Structured products are flexible, customizable investments that are usually defined as the combination of a risk-free asset, such as a zero-coupon or governmental bond, and a risky asset, such as an exotic option. As structured products are customized to the needs of a given investor, they are particularly illiquid products in secondary markets. Therefore, we focus on the primary market of structured products and aim at anticipating clients' needs in that particular market. This section explores the problem of structured products primary recommendation and outlines a methodology for solving it.

According to Bloomberg, the structured products market weighed seven trillion dollars in 2019 (Bloomberg Professional Services, 2019). There are many different kinds of structured products — two popular ones are the following:

- **Reverse Convertible.** Reverse Convertibles are financial products linked to a particular stock or basket of stocks that are particularly relevant in situations where an investor expects a stable or low increase in the underlying. They deliver regular, high coupons and a payout at maturity that depends on the closing values of the underlying — if the underlying never closes below a given barrier, the payout corresponds to the initial payment (the *principal*); otherwise, the payout corresponds to the principal discounted by the actual performance of the underlying. Reverse convertibles are typically short-term investments, with a maturity of one year.
- **Autocall.** Autocalls are financial products linked to a particular stock or basket of stocks that are particularly relevant in situations where an investor expects a stable or low increase in the underlying but are more restrictive than reverse convertibles and consequently less expensive. An autocall delivers regular, high coupons and a payout that depends on the closing values of the underlying — if the underlying closes above a predetermined barrier, the principal is immediately paid and the investor will not receive further coupons; if the underlying never closes above a second given barrier, the payout corresponds to the principal, and otherwise the payout corresponds to the principal discounted by the actual performance of the underlying. Autocalls are typically medium-term investments that span a few years.

These two examples give insights into the customizable characteristics of structured products, from the underlying stock or basket of stocks to the specific barriers required for autocalls and reverse convertibles. Our goal is to design a model that could help structured products' salespeople better serve their clients by providing them with relevant structured product ideas, i.e., characteristics that match their current needs. More specifically, the chosen model should be able to score any collection of characteristics for any particular investor to help salespeople navigate potentially interesting ideas for their clients. To pursue this goal, we have at our disposal data corresponding to the structured products that BNP Paribas CIB clients bought, and at what date they bought them. Contrarily to the previously seen situations, this data does not come from D2C platforms (see Section 1.1.1), but from the *Over The Counter* (OTC) trading activity of the bank.

As structured products are customized to match the interests of their investors at a particular time, there is no notion of "unique product" on which we can rely, such as in bonds. When a client invests in a structured product, she invests in a collection of characteristics that suits her needs at that time. A specific set of characteristics can still be tagged as a unique product to allow for our previous methodologies, but it is highly doubtful that these characteristics will be of interest to another client. Consequently, historical baseline algorithms that rely solely on *(user, item)* couple frequencies for their predictions are not as relevant as previously to solve the structured products' primary recommendation problem. More broadly, only algorithms that do not suffer from the cold-start problem can be used to efficiently solve this problem.

The featurized HCF algorithm introduced in Section 4.4.3 is consequently one of the few proposals that both match the challenges of structured products' recommendation and the targeted goals, and allows for taking time into account in a satisfying way, as seen in the experiments of Section 4.3 and extended in Section 4.4.2. Using a set of proprietary features to describe the characteristics of the structured products, the product's underlying and its market conditions, we can train a featurized HCF model with the naive sampling strategy from Section 4.4.1 and contiguous splits of data, following the methodology outlined in Section 4.3.1. Note that it is also possible to use content-filtering approaches, as seen in Section 1.3.2, to solve this problem,

using the same kind of features as the ones outlined above for HCF. Such algorithms could be trained on a cross-entropy objective, with negative events sampled as in the HCF methodology — the imbalance ratio should be controlled to obtain relevant probabilities.

As a given structured product will only appear a couple of times in our data, scoring the algorithms becomes particularly tricky: the methodologies explained in Section 1.4 and 4.3 are not applicable anymore, as restraining the scoring perimeter to the users and items observed during training would not capture the actual trades that happened during the testing period. Consequently, we change the scoring perimeter previously defined as all potential *(user, item)* couples formed of users and items appearing in the training period to a perimeter defined by the couples formed of all users seen during the training period and all the products that were traded during the past month. This provides a perimeter of relevant items to which compare the scores predicted for the positive ones. As previously, we can use the symmetrized mAP score for evaluation. Table 4.5 shows first results of three algorithms on the primary structured products prediction problems: a historical baseline obtained as suggested above by considering sets of characteristics as our unique products, a "featurized" MF trained with the BPR loss, and a "featurized" HCF model — see Section 4.4.3 for more details. These algorithms use a training set that can range from 02/01/2019 to 12/31/2019, a validation set that ranges from 01/01/2020 to 01/31/2020 and a test set that ranges from 02/01/2020 to 29/02/2020. We use for these algorithms the window sizes the closest possible to the optimal ones found in Section 4.3.1, and the MF and HCF algorithms use the same values for their common hyperparameters.

Table 4.5: Comparison of historical, featurized MF and HCF algorithms on the primary structured products prediction problem. Symmetrized mAP scores, expressed in percentages.

Algorithm	Valid mAP	Test mAP
Histo - 3m	10.96	8.75
featurized MF (BPR) - 2m	13.67	11.93
featurized HCF - 11m	**34.33**	**32.75**

We see that the featurized HCF obtained, by far, the best validation and test performances. Note that the featurized HCF algorithm only differs from featurized MF by its usage of histories. Consequently, it appears that the introduction of time in the primary prediction problem is crucial, validating again the approach taken for HCF. Further research effort will be devoted to improving our models for the primary structured product prediction problem.

4.5 Concluding remarks

This chapter introduces a novel HCF algorithm, a time-aware recommender system that uses user and item histories to capture the dynamics of the user-item interactions, and that provides dynamic recommendations that can be used for future predictions. In the context of financial G10 bonds RFQ recommendations, we show that for classic matrix factorization algorithms, a trade-off exists between the non-stationarity of users' and items' behaviors and the size of the datasets that we use for training. This trade-off is overcome with history-augmented embeddings. Moreover, such embeddings outperform sliding versions of classic matrix factorization algorithms and prove to be more stable predictors of the future interests of the users and items.

This chapter also supplements the HCF algorithm with its featurized extension, provides insights into potential sampling strategies, studies predicted scores, and presents a use-case for which the featurized version of HCF is particularly relevant. Further research on the HCF subject will include how history sizes could be personalized, how sampling strategies could be combined and how we could tackle clients' heterogeneity more efficiently with featurized user representations.

Chapter 5

Further topics in financial recommender systems

Developing the two algorithms presented in chapters 3 and 4 required countless experiments, as can be seen in both chapters from the many iterations these algorithms have known. However, iterating on the models introduced in the previous chapters is not the only work that has been done during the elaboration of this thesis. Consequently, this chapter collects, haphazardly, the other experiments we conducted and ideas we developed and that do not deserve their own chapter.

This chapter introduces statistically validated networks and their application to BNP Paribas CIB datasets (see Section 5.1), elaborates on minor improvements that help previously introduced algorithms achieve better performance (see Section 5.2), presents an experiment trying to capture lead-lag relationships using neural networks (see 5.3) and paves the way for a model bridging the ideas introduced in chapters 3 and 4 (see Section 5.4).

5.1 Mapping investors' behaviors with statistically validated networks

Clustering clients can be done in many ways. Provided with user profiles, a wide range of clustering algorithms can be used to group lookalike clients — classic ways include K-means (Hastie et al., 2009, Chapter 13) and DBSCAN (Ester et al., 1996). In this section, we consider a rather unusual graph mining method that detects regular, common patterns among clients and that leads to a behavioral-based clustering of clients, reminiscent of the work presented in Chapter 3, and apply it to the corporate bonds RFQ datasets of BNP Paribas CIB. The method, called *statistically validated networks*, is described in Tumminello et al. (2011) and has been successfully applied to a financial dataset in Tumminello et al. (2012).

The financial dataset studied in Tumminello et al. (2012) corresponds to the trading behavior of individual investors on the Nokia stock. As bonds are less liquid products than stocks, the data related to a given bond is far sparser than it is the case for a given stock such as Nokia. Consequently, the methodology outlined in Tumminello et al. (2012) cannot be directly transposed to our bonds RFQ dataset. To use this methodology, we consider the interactions clients have with the bond market as a whole by considering all bonds as one.

The idea behind statistically validated networks is to connect clients according to their RFQ patterns. Let us consider a bipartite graph $G = (C, T, E)$ with C the set of clients, T the set of considered trading days, and E the set of edges connecting them. Clients can be connected

to a trading day by three different kinds of links, corresponding to their observed behavior on that day — a "buy" link if the client mostly requests bonds' prices for the buy direction on that day, a "sell" link if the client mostly requests bonds' prices for the sell direction on that day, or a "buy-sell" link if the client requests prices for both directions on that day in a balanced way. Links are determined using

$$r(i,t) = \frac{V_b(i,t) - V_s(i,t)}{V_b(i,t) + V_s(i,t)}, \tag{5.1}$$

where $i \in C$ represents a client, $t \in T$ a trading day, and V_b and V_s are the aggregated requested amount of buy and sell movements for client i on day t. We consider that the link between i and t is a buy link if $r(i,t) > \rho$, a sell link if $r(i,t) < -\rho$ and a buy-sell link if $-\rho < r(i,t) < \rho$ with $V_b(i,t) > 0$ and $V_s(i,t) > 0$. If client i is linked to day t with a buy link, we say that i in in a *buy state* during that day, and respectively for all other link types. Following Tumminello et al. (2012), the threshold value is taken to be $\rho = 0.01$. Note that taking $\rho = 0$ is equivalent to considering buy and sell links only.

We now want to project this bipartite graph into a one-mode graph that connects clients according to their RFQ patterns. A first idea is to consider a graph $G' = (C, E')$ where $(i,j) \in E'$ if i and j both made a move on the same day, e.g., if for a given day t, client i is in a buy state and client j in a sell state, i and j are connected in G' with a buy-sell link. However, this approach rapidly fails: as co-occurrence of states might happen for totally unrelated reasons, the corresponding graph would be fully connected with a high probability. Consequently, we have to correct for potentially random co-occurrences of states. To do so, the idea introduced in Tumminello et al. (2011) is to use a statistical test to check whether co-occurrence of states between two given clients is random. We note $G_{SVN} = (C, E_{SVN})$ the corresponding graph.

The statistical test is the following. We consider as null hypothesis that co-occurrence of states P and Q for clients i and j is random, and want to test whether this hypothesis holds. Let N_P be the number of days client i spent in state P, N_Q the number of days client j spent in state Q, $N_{P,Q}$ the number of days when we observe the co-occurrence of state P for client i and state Q for client j and T the total number of trading days we consider — for ease of notation, we assimilate the set of trading days to its number of elements. Under the null hypothesis, the probability of observing X co-occurrences of the given states for the two clients is given by the *hypergeometric distribution*, that we note $H(X|T, N_P, N_Q)$. Consequently, the p-value corresponding to the described statistical test is given by

$$p(N_{P,Q}) = 1 - \sum_{X=0}^{N_{P,Q}-1} H(X|T, N_P, N_Q) \tag{5.2}$$

As this test is performed for all possible client pairs and all possible combinations of states, we rely on multiple hypotheses testing to conclude on rejections of the null. Classic methods to account for multiple hypotheses are the Bonferroni correction (Bland and Altman, 1995) and the Benjamini-Hochberg procedure (Benjamini and Hochberg, 1995). The Bonferroni correction aims at controlling the *family-wise error rate* (FWER), defined as $\text{FWER} = \mathbb{P}(\text{FP} \geq 1)$, with FP the number of false positives. Bonferroni ensures that $\text{FWER} \leq \alpha$, with α the chosen significance level of the test, by rejecting an hypothesis H_i if its corresponding p-value p_i is such that $p_i \leq \frac{\alpha}{m}$, with m the total number of hypotheses considered.

We focus here on the Benjamini-Hochberg procedure, which is a more conservative way to account for multiple hypotheses. This procedure aims at controlling the *false discovery rate* (FDR), defined as the expectation of the *false discovery proportion* (FDP), written $\text{FDP} = \text{FN}/\max(R, 1)$, with FN the number of false negatives and R the total number of rejections. The Benjamini-Hochberg procedure attempts to control the FDR at level α. The procedure works as the following:

1. Order hypotheses in ascending order with regard to their associated p-values, $p_{(1)}, ..., p_{(m)}$, with $(\cdot)$ denoting the *order statistics* of the p-values and a total of m hypotheses to test;
2. Find the Benjamini-Hochberg cutoff defined as $\hat{k} = \max\left\{i : p_{(i)} \leq \frac{i\alpha}{m}\right\}$;
3. Either reject hypotheses $H_{(1)}, ..., H_{(\hat{k})}$ or perform no rejections if $\hat{k}$ is not defined.

We apply this procedure with a significance level of $\alpha = 0.01$, following the values experimented in Tumminello et al. (2011) and Tumminello et al. (2012). Counting now all undirected potential links between two clients, there are in total $m = 9|C|(|C| - 1)/2$ different hypotheses that we need to test.

The validated links, i.e., the links for which the corresponding p-value was rejected, form a *statistically validated network* of clients. Multiple types of links can link two clients, as we test for each client pair a total of nine different links — let us call these links *single*. If multiple types of links are validated for a given client couple, we say these clients are linked with a *multi-link*. For instance, if a "buy-buy" and a "sell-sell" link connect two clients, we consider them connected by a "buy-buy/sell-sell" link, translating the fact that these clients tend to buy or sell at the same moments. Tumminello et al. (2011) and Tumminello et al. (2012) note that only some kinds of multi-links tend to appear — Tumminello et al. (2012) evoke the prevalence of the "buy-buy/sell-sell" and "buy-sell/sell-buy" links, thereby showing positive and negative correlations in the clients' trading behaviors.

The statistically validated network obtained on the corporate bonds RFQ dataset is shown in Fig. 5.1. The corresponding graph was obtained using $\rho = 0$, to avoid detection of buy-sell links in the bipartite graphs — as these links were very infrequent, the SVN methodology did not validate any such links during our experiments, thereby potentially masking the validation of buy-sell, buy-buy or sell-sell links in the one-mode projection. When removing buy-sell links from the bipartite graph, the total number of hypotheses to test becomes $m = 4|C|(|C| - 1)/2$. An analysis of this result validates the methodology *a posteriori*. Indeed, we see in Fig. 5.1 that clients of a same sector of activity tend to be linked. Official institutions, a sector predominantly composed of central banks, are the most linked clients, and we see that official institutions are linked together with buy-buy links and linked to other banks by buy-sell links, corresponding to the fact that central banks buy bonds from banks to maintain stable inflation rates, a mechanism known as *quantitative easing*. Moreover, we see that asset managers are often paired together and that a few banks are linked by "buy-buy/sell-sell" multi-links shown on the graph as "bb-ss" links, showing a positive correlation between their RFQ patterns.

Clusters can be extracted from this network with algorithms such as Infomap, introduced in Rosvall and Bergstrom (2008). The Infomap algorithm forms clusters using random walks on a weighted graph G_{SVN}, where weights are given by the number of single links between a clients' pair. However, such clusters could not be used *per se* to get a better understanding of our client pool on the whole. From the few thousands of clients present in our corporate bonds RFQ dataset, we see that less than a hundred of them end up being validated in the final network. Even when considering the interactions of clients with the bonds' market overall, our data is still very sparse to obtain a relevant clustering of our clients' pool. However, the obtained clustering could be used as a feature of a predictive model, such as the ones introduced in chapters 3 and 4. Note that instead of deriving a statistically validated network of clients, the same method could be applied to bonds — as analysis of obtained results becomes more difficult in this setting, the final clusters obtained could be used in a model.

Consequently, this section shows that statistically validated networks are a powerful data mining tool to extract information from clients' co-occurring patterns. However, the illiquidity of the

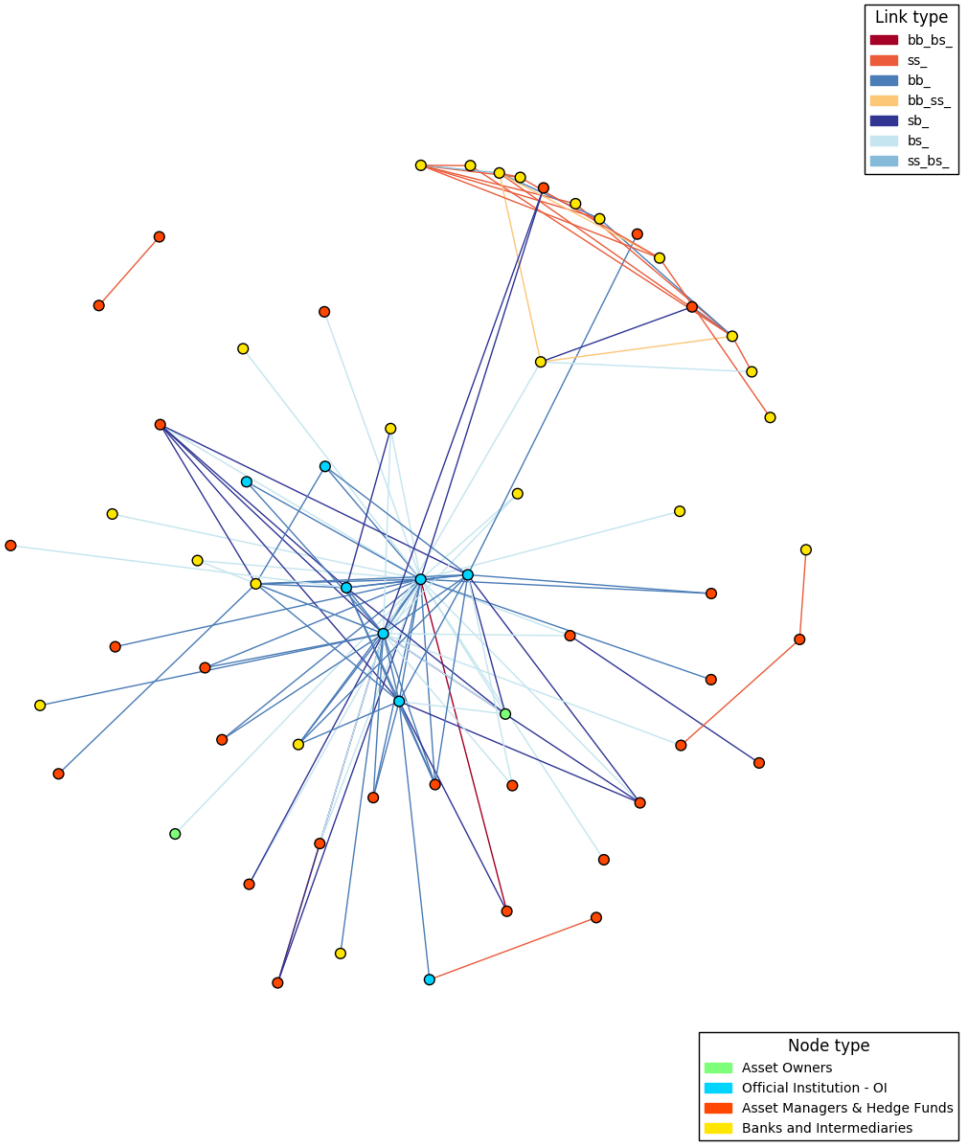


Figure 5.1: Statistically Validated Network obtained on the corporate bonds RFQ data from BNP Paribas CIB. Each dot corresponds to a client, where node color represents the client's sector of activity. We see seven types of links happening, three of them being multi-links. Clients of a same sector of activity tend to be paired together, and we see that official institutions — mainly central banks — are linked together with buy-buy links and linked to other banks with buy-sell links, a phenomenon we attribute to the mechanism of *quantitative easing*.

bonds' market, compared, e.g., to the stock market, hinders the quality of obtained results. The obtained SVNs only cover a small portion of our clients' pool, making their results difficult to use in practice. The clustering obtained with SVNs can be used as a feature of a predictive model — corresponding classes will, however, be very ill-balanced, which could lead to mixed results.

5.2 A collection of small enhancements

During the development of the algorithms presented in chapters 3 and 4, a lot of small improvements have been tried to enhance the performance of our predictive models. In this section, we present three of these most successful variations that we deem not worth their own research studies.

5.2.1 A focal version of Bayesian Personalized Ranking

Chapter 4 extensively studied and applied the BPR objective function of Rendle et al. (2009). In this chapter, we proposed ways to augment BPR through different sampling strategies, namely with a symmetrized, history-aware sampling (see Sections 4.2.3, 4.4.1). The BPR objective could also be enhanced with the ideas underpinning the focal loss (Lin et al., 2017) (see Section 1.3.2).

The focal loss is a variation of cross-entropy that weighs hard-to-classify samples more, as

$$l_{focal}(x) = -\sum_{i=1}^{C}(1-\hat{y}_i(x))^{\gamma} y_i(x) \log \hat{y}_i(x) \ , \tag{5.3}$$

with C the number of classes considered in the classification problem, (x, y) a given sample where y is a C-dimensional one-hot encoding of the class corresponding to sample x, $\hat{y}$ a predictor and γ a hyperparameter controlling the adaptive weighing. The term $(1-\hat{y}_i(x))^{\gamma}$ emphasizes the loss on the wrongly classified samples, or classified with low confidence. This idea can be easily adapted to the BPR objective, as

$$l_{BPR,focal} = (1-\sigma(x_{uij}))^{\gamma} \ln \sigma(x_{uij}), \tag{5.4}$$

where σ denotes the sigmoid function, x_{uij} is the score associated to the (u, i, j) triplet where (u, i) is a positive couple and j a negative item, and γ is an hyperparameter controlling the adaptive weighing. The BPR objective is constructed such that $\sigma(x_{uij})$ represents $\mathbb{P}(i >_u j)$ (Rendle et al., 2009), with $>_u$ the total ranking defined by user u, i.e., the order defined by the interests of u. Consequently, the $(1-\sigma(x_{uij}))^{\gamma}$ term focuses the loss on samples for which the model puts negative items higher than positive ones in its ranking. Note that setting $\gamma = 0$ leads to the classic BPR objective.

We experiment this new loss on the HCF model, using the hyperparameters and window sizes of the HCF 12m model of Section 4.3.1 and the same experimental setup for obtaining validation and test performances. Results are shown in Table 5.1.

Table 5.1: Comparison of classic and focal versions of HCF — symmetrized mAP scores, in percentage.

Algorithm	Valid mAP	Test mAP
HCF	24.43	25.02
focal HCF - $\gamma = 1$	**24.68**	**25.44**
focal HCF - $\gamma = 2$	24.53	25.31
focal HCF - $\gamma = 3$	24.39	25.16

We see that the best model with regards to validation performance is an HCF model trained with the focal BPR objective using $\gamma = 1$. Note here that a more thorough search for γ in the $]0; 2]$ range would be necessary to find the optimal γ value. We see that the validation

performance falls below the classic version for $\gamma = 3$ — with too high γ values, the low weight of triplets (u, i, j) such that $\mathbb{P}(i >_u j) \approx 0.5$, i.e., triplets for which the network is indecisive, hinders the network performance. The $\gamma = 1$ model achieved a test performance marginally above classic HCF, with a +1.7% increase in symmetrized mAP score. Consequently, our focal BPR objective proposal helps achieve overall better performances by focusing the BPR objective on hard-to-rank positive-negative pairs.

5.2.2 Validation strategies in non-stationary settings

K-fold cross-validation is a widespread technique for estimating the prediction error of a machine learning model (Hastie et al., 2009, Chapter 7.10). The idea is to split available data into K equally-sized parts, to train a model on $K-1$ of these parts and test it on the held-out last one. Noting $\kappa : \{1, \dots, n\} \longrightarrow \{1, \dots, K\}$ the partition function splitting the samples $\{(x_i, y_i)\}_{i=1,\dots,n}$ into K folds, the cross-validation estimate of the prediction error is given by

$$\mathrm{CV}(\hat{f}) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{-\kappa(i)}(x_i)), \tag{5.5}$$

with $\hat{f}$ the predictor, L the considered evaluation function, and $\hat{f}^{-\kappa(i)}$ a predictor trained on all K folds except for the one where sample i lies. K-fold cross validation allows for obtaining good estimates of the prediction error when there is not sufficient data to use a separate, held-out validation set.

However, in the financial context, De Prado (2018, Chapter 7) advocates that K-fold validation is prone to *signal leakages*. Indeed, in non-stationary settings samples cannot be considered i.i.d. anymore: when using temporal folds of data, on the boundaries between train and validation splits, it is often the case that $(x_t, y_t) \approx (x_{t'}, y_{t'})$ for t' close enough to t. De Prado (2018, Chapter 7) consequently proposes a "purged" K-fold cross-validation procedure, illustrated in Fig. 5.2. The procedure, similar to what is known in the literature as $h\nu$-block cross-validation (Racine, 2000), follows the idea of classic K-fold cross-validation but avoids leakages by discarding samples "close" to the validation block, i.e., in a h-sized window around the validation block. A complete overview of cross-validation strategies suitable for time series prediction can be found in Bergmeir and Benítez (2012).

However, when plenty of data is at our disposal, there is no need to use such cross-validation strategies: using a held-out, large validation set is sufficient to monitor the generalization performance of a model. In such cases, the common practice is to use contiguous splits of data ordered as training, validation, and test splits, as was done in the experiments of Chapter 3 and 4. Such a validation strategy emulates the real-world environment of the model, which will be used for predicting events *after* the training period. There are, however, two observations we can consider when choosing a validation strategy:

- We expect the performance of a model on the validation set to be a good predictor of the model's generalization performance. Consequently, the most important characteristic we should aim at when searching for an appropriate validation set is the correlation between the validation and test performances.
- In non-stationary environments, the further away from the training period the predictions, the worse their performance. Consequently, an appropriate validation strategy would ideally let samples temporally close to the testing period in the training set.

Following these observations, we try a validation strategy that uses contiguous splits of data ordered as validation, training, and test splits. Using the G10 bonds RFQ dataset (see Section 4.3 for more details), we train 200 instances of a LightGBM (Ke et al., 2017) with randomly

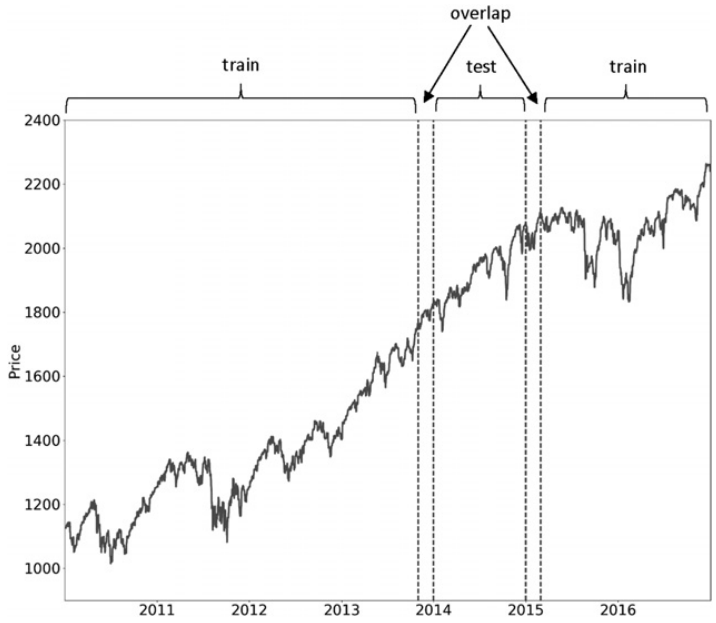


Figure 5.2: A split of purged K-fold cross-validation, a.k.a., $h\nu$-block cross-validation. To remove the dependence effects on the validation block, data samples before and after the block are discarded from the training set. Illustration taken from De Prado (2018, Chapter 7).

sampled hyperparameters in two different configurations: contiguous splits in the training, validation and test order and in the validation, training and test order, respectively illustrated in Fig. 5.3**(a)** and **(b)**. We see that both experiments obtain relatively similar results. The correlation between validation and test performances is good in both cases, the R^2 score being in favor of experiment **(b)**. Moreover, the best model in terms of validation performance obtains a marginally better test performance in the "validation, training" setting than in the "training, validation" one.

Consequently, in our RFQ datasets, it seems that the validation, training, test order is an appropriate validation strategy candidate. As it allows for better immediate model performance, the training period being the closest possible to the testing one, we favor this splits' ordering. Note, however, that contiguity of the validation and training sets is still necessary. Using a validation set too far away in the past leads to poor correlation between validation and test performances — another hint at the non-stationary behavior of the RFQ datasets.

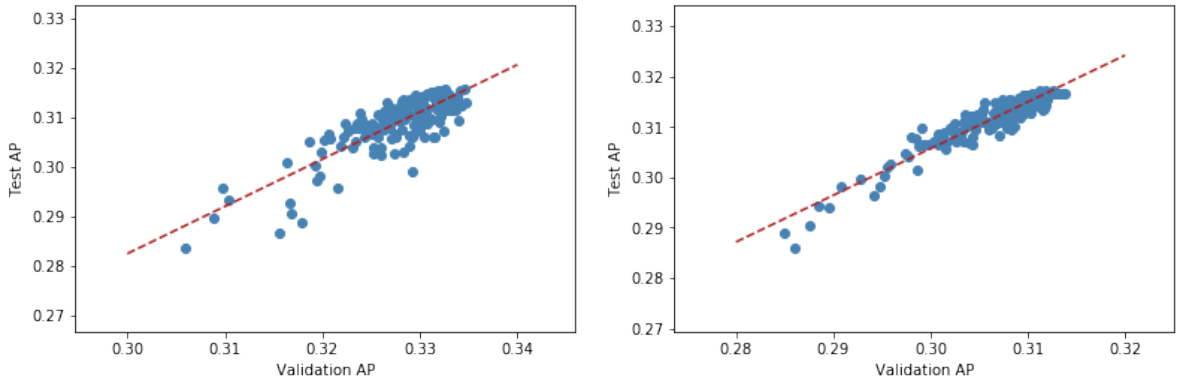
5.2.3 Ensembling with an entropic stacking strategy

When using fast-to-calibrate models such as gradient tree boosting algorithms, it is usual to test large random grids of potential hyperparameter combinations to find an optimal model for the task at hand. However, instead of only using the optimal model as defined by the performance on a held-out validation set, we could try to combine all trained models to improve the performance of the optimal one. This idea is known as *stacking* in the literature, an idea initiated in Wolpert (1992), and that has known many further developments.

We want here to linearly combine multiple instances of a given model, as

$$\hat{f} = \sum_i \alpha_i \hat{f}_i, \tag{5.6}$$

where $\hat{f}$ denotes the linear combination, $\hat{f}_i$ is a pre-trained instance of a given model, and the set of weights $\{\alpha_i\}_i$ are such that $\forall i, \alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. The non-negativity constraint



(a) **(b)**

Figure 5.3: Overall average precision test performance as a function of overall average precision validation performance. Blue dots represent instances of the LightGBM algorithm trained on the G10 bonds RFQ dataset, each dot corresponding to randomly sampled hyperparameters. Red dashed line represents a linear fit of these data points. In both experiments, correlation between validation and test performances is good. **(a)** Correlation between validation and test performances with a validation set positioned right after the training set. R^2 score: 0.69. **(b)** Correlation between validation and test performances with a validation set positioned right before the training set. R^2 score: 0.89.

on the α_i was first introduced in Breiman (1996). Note that this formulation is particularly reminiscent of the ExNet methodology, introduced in Chapter 3.

Let us consider here that the model instances are trained as usually exposed in this thesis, i.e., using a training, validation, and test splitting of the available data. Our process is to train the $\{\alpha_i\}_i$ on the validation set only to avoid overfitting the training split, until convergence of the considered loss — e.g., in a classification setting, we train these weights using a cross-entropy objective computed on $(\hat{f}(x_i), y_i)$ for i in the validation set. The set of $\{\alpha_i\}_i$ weights can be understood as a probability distribution along the instances considered in the stacked ensemble. To obtain meaningful weight repartitions, it is usual to regularize the α_i weights, e.g., using an L_1 penalty to select relevant instances to be used in the combination. In this work, we follow the spirit introduced with ExNets in Section 3.2.2 and penalize the weights with entropy, as

$$L_{\text{entropy}} = -\sum_i \alpha_i \log(\alpha_i). \tag{5.7}$$

Let us examine the outlined methodology in an experiment on a foreign exchange options RFM dataset — see Section 1.1 for more details. Here, we want to combine the top-30 LightGBM (Ke et al., 2017) instances in terms of validation loss in a stacked ensemble. The model converges after roughly 40 epochs. Weight repartition is illustrated in Fig. 5.4. Interestingly, we see that the top model was not significantly used here and that the weights span the top-20. The combination uses in total 6 instances with a weight $\alpha_i \geq 0.05$.

Figure 5.5 examines the correlation between the considered instances. In Fig. 5.5**(a)**, we see that the instances are overall largely positively correlated, with correlations above 98% for all model pairs. However, if we focus on the instances selected with $\alpha_i \geq 0.05$, we see that the correlations of the selected models are among the lowest, except for two specific instances. This behavior was anticipated, as we expect dissimilar models to perform better when combined than similar ones.

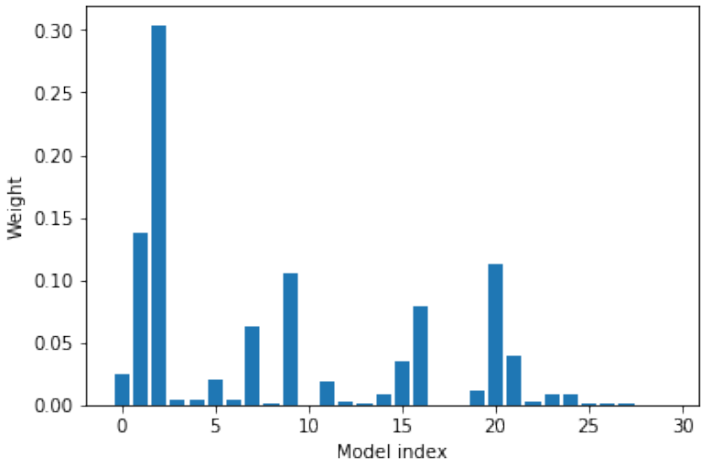


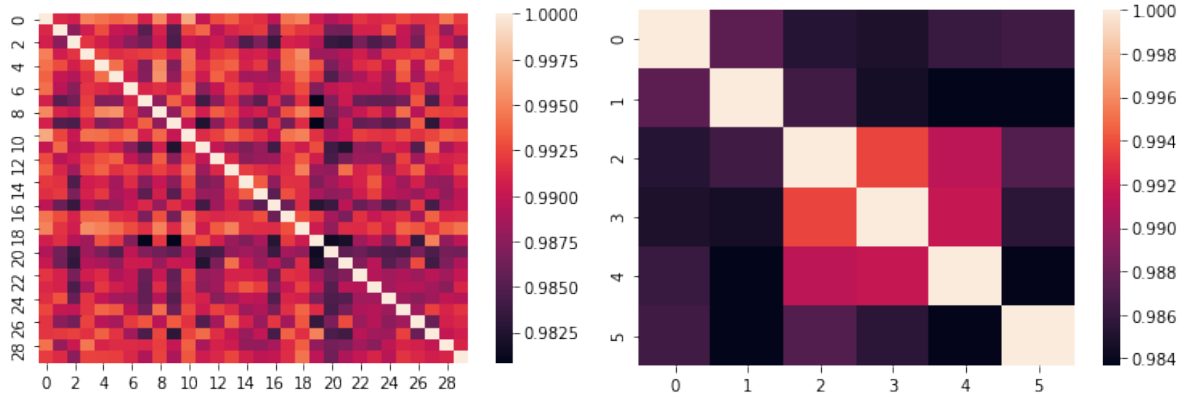
Figure 5.4: Weight repartition along the considered instances. Model indexes are ordered by performance on the validation set. We see that weights span the top-20, and that the best model was not significantly selected in the combination.

On a final note, the stacked ensemble obtained better performance on both validation and test splits than the single optimal model. Consequently, we see that stacked models yield better results than a single, well-tuned instance of a model by leveraging all models obtained during the hyperparameter optimization process. Moreover, this experiment only considered the top-30 models. Including a larger number of models would allow discovering smaller correlations between models — following on the above experiment, better results are expected when using less correlated models. Consequently, further iterations of this work will consider a more thorough panel of potential instances to combine.

5.3 First steps towards lead-lag detection

In finance, the notion of *lead-lag* refers to the fact that the activity of an investor on a given market might trigger the activity of other investors at a later date — more generally, it refers to the correlation of two time series, one being *lagged* with regards to the other. Lead-lag relationships are particularly helpful for understanding the endogenous portion of financial markets (Hardiman et al., 2013). Statistically validated networks, introduced in Section 5.1, have been successfully adapted and applied to the detection of lead-lag effects in diverse financial markets (Curme et al., 2015; Challet et al., 2018).

This section introduces a batch of ideas to detect lead-lag using neural networks, from lead-lag modeling to a network specifically designed to detect it. Although not fully mature, we hope these ideas are a step towards functional lead-lag detection modules in neural network architectures.



(a) **(b)**

Figure 5.5: Heatmap visualization of the Pearson correlations between the outputs $\hat{f}_i$ of the considered LightGBM instances. **(a)** Correlations of all top-30 instances. All instances are largely positively correlated. **(b)** Correlations of instances with $\alpha_i \geq 0.05$. We see that these correlations are among the lowest correlations obtained over all possible instance pairs.

5.3.1 Simulating lead-lag

Following ideas from the ExNet experiments on synthetic data (see Section 3.3.1), we design an experiment to prove that our methodology is able to retrieve lead-lag relationships. In lead-lag situations, there are what we call in this work *leaders* and *followers*. We consider a given pool of investors U, in which we find leaders $l \in L$ and followers $f \in F$, with $L, F \subset U$ and $L \cup F = U$. Note that we only consider a situation where leaders act on their own, and cannot follow anyone. The underlying idea is that followers mimic the behavior of one or more leaders with a delay — a *lag* — τ_f, that depends on the considered follower. In terms of inputs and outputs, for a follower to obtain the same output with a given lag, the follower should mimic her leader's input on the lagged day.

We represent this in a manner similar to Section 3.3.1. Again, we model the problem as a binary classification task using a set of p features, noted $X \in \mathbb{R}^p$, that we map to a binary output noted Y. X samples are drawn from a normal distribution $\mathcal{N}(0, 1)$. The behavior of leaders is described by the following equation:

$$\mathbb{P}(y_t^l | x_t^l) = \sigma\left(w^T x_t^l\right), \tag{5.8}$$

where w are the decision weights, x_t^l the features describing the state of leader l at time t and y_t^l represents the decision of leader l at time t. The behavior of followers, more intricate, is described by:

$$\mathbb{P}(y_t^f | x_t^f, \{x_{t-\tau_f}^l\}_{l \in L_f}) = \sigma\left(w^T\left(\beta x_t^f + (1-\beta)\frac{1}{|L_f|}\sum_{l \in L_f} x_{t-\tau_f}^l\right)\right), \tag{5.9}$$

where w are the decision weights *shared with leaders*, x_t^f the features describing the state of follower f at time t, y_t^f represents the decision of follower f at time t, L_f is the set of leaders that f follows, $\tau_f \in \{1, 2, ...\} := \Delta$ is the lag of follower f and β is an hyperparameter controlling the effect of leaders on followers. For a given follower, her decision is driven at $\beta\%$ by her own features, and at $(1-\beta)\%$ by the features of her leader(s), observed at the lag τ_f corresponding to her.

The final decision of a given investor is then determined with

$$Y_t^* = (\mathbb{P}(y_t^*) > U) \in \{0, 1\}, \ * \in \{l, f\}, \tag{5.10}$$

where U is uniformly drawn in $[0; 1]$. As our interest is in retrieving who follows who, we use a common decision rule for all investors not to add more complexity on top of an already difficult problem. The number of leaders followed L_f and the lags τ_f are randomly drawn for all followers. In this section, we consider a simple case where $p = 5$ features are used, $\forall f, \ |L_f| = 1$ and $\tau_f = 1$, and we vary the value of β to determine our ability to retrieve leaders and followers in a pool of 100 clients examined during 1000 time-steps. As in Section 3.3.1, decision weights are given by $w = (5, 5, 0, 0, -5)$. With such weights, binary outputs are balanced.

5.3.2 A trial architecture using attention

We introduce here an architecture that mostly relies on the *attention mechanism*. Attention, popularized by Transformer networks (Vaswani et al., 2017), is a deep learning concept particularly important in natural language processing. As of now, Transformer networks are the core component of all state-of-the-art language models, from GPT-2 (Radford et al., 2019) to BERT (Devlin et al., 2019) and their numerous improvements and variations (Dai et al., 2019; Yang et al., 2019; Liu et al., 2019; Clark et al., 2020). In NLP, the idea of attention is to enrich the representation of a given word using the words that surrounds it in a particular sentence, thereby *contextualizing* words' representations. To detect lead-lag relationships, we try here to adapt this idea to enrich the features of a given investor with the features — lagged or not — of other investors.

To do so, we introduce two sets of attention weights, $\{\alpha_{cl,u} \in \mathbb{R}^{|U|}\}_{u \in U}$ and $\{\alpha_{lag,u} \in \mathbb{R}^{|\Delta|+1}\}_{u \in U}$, respectively the attention weights of investors to investors, i.e., potential leaders, and investors to lags — the +1 refers to the attention given to the current time-step. For a given investor, at a given time-step t, the idea is to visualize data as a "cube" $x_t \in \mathbb{R}^{|U| \times p \times (|\Delta|+1)}$ and reduce this volume, identical for all investors at a given time-step, into a personalized sample using the attention weights corresponding to the considered investor, as illustrated in Fig. 5.6. Doing this for all investors, we obtain a design matrix $\tilde{X}_t$ that can be used as the input of a basic feed-forward neural network, trained with cross-entropy to retrieve targets y_t. Due to the computational requirements of this volume reduction, a batch of data is composed of the features of all investors for a given date.

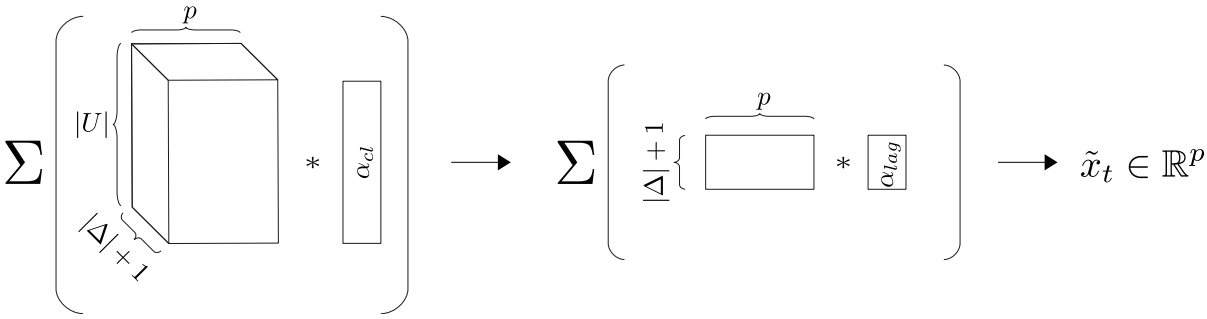


Figure 5.6: Example of data volume reduction for a particular investor using investor and lag attentions. The original three-dimensional tensor is reduced to a sample $\tilde{x}_t \in \mathbb{R}^p$ through two weighted averages, where weights are given by the investor and lag attentions of the considered investor.

Taking inspiration from Vaswani et al. (2017), we define attention weights using three sets of embeddings, all sharing the same dimension d:

- **Investor query embeddings.** These embeddings are used to query the attention weights corresponding to a particular investor, be it investor-investor or investor-lag attentions.
- **Investor key embeddings.** These embeddings, used in conjunction with investor query embeddings, give the queried investor compatibility to the considered key investor. Investor query and key embeddings introduce asymmetry: lead-lag relationships are oriented, and the corresponding attention weights matrix consequently cannot be symmetric.
- **Lag key embeddings.** These embeddings, used in conjunction with investor query embeddings, give the queried investor compatibility to the considered key lag.

The compatibility between the query and key embeddings is determined using

$$\{\alpha_{*,u}\}_{u\in U} = \left\{\sigma\left(\langle q_u K_*^T\rangle\right)\right\}_{u\in U}\ , \tag{5.11}$$

with $* \in \{cl, lag\}$, σ the sigmoid function, $q_u \in \mathbb{R}^d$ the query embedding of investor u and $K_* \in \mathbb{R}^{d_* \times d}$ the d_* considered key embeddings. Note that different query embeddings could be used to match lag keys: we use the same query embeddings for both attention modules to share parameters across two intricately linked attention-retrieval tasks, as the attention an investor has to pay to another investor is only valid for a given lag.

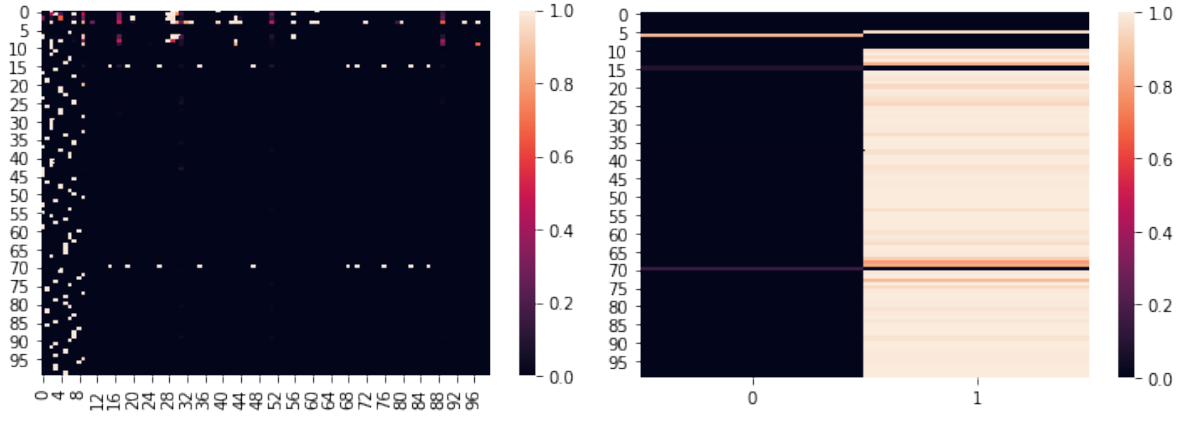
All attention weights are modulated by the sigmoid function, so that $\alpha_{*,u} \in [0;1], \forall * \in \{cl, lag\}$, $\forall u \in U$ — a given investor either has to pay attention to a particular investor (resp. lag), or not. As we want investors to focus on a few leaders at most only, we enforce sparsity of the investor-investor attention weights with an entropic penalty similar to the one introduced in Section 3.2.2 that drives attention weights to either 0 or 1.

5.3.3 First experimental results

We experiment the methodology outlined in Section 5.3.2 on two synthetic datasets, created according to 5.3.1 with respectively $\beta = 0$ and $\beta = 0.2$, β corresponding to the reliance of a follower, in percentage, on her own features. The two datasets contain 1000 dates, 800 of them being used for training and 200 as a validation set used for early stopping.

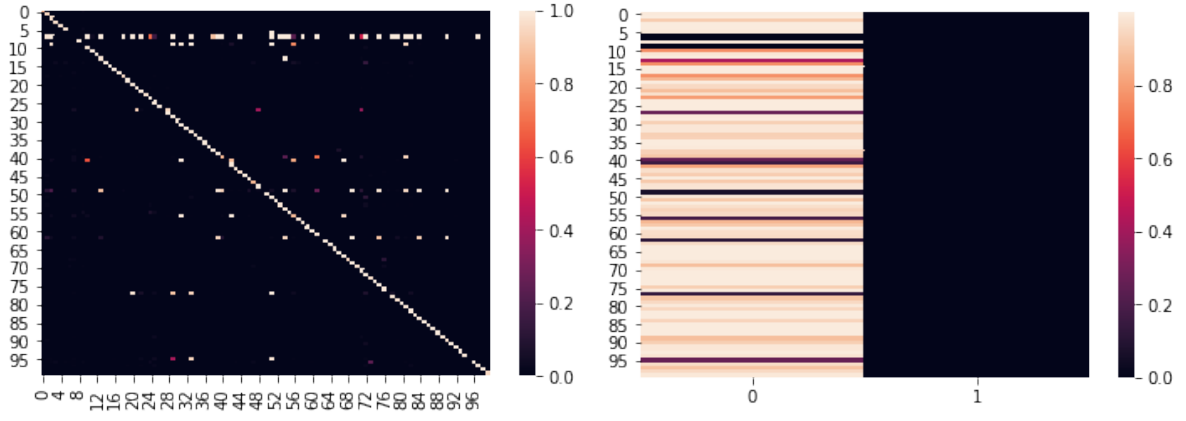
Let us first examine the $\beta = 0$ case, illustrated in Fig. 5.7. In this setting, a follower's decision function is entirely given by her leader's features. We see in Fig. 5.7**(a)** that the network was able to retrieve the leaders of almost all followers, but failed to learn that leaders only follow themselves, except for one. Interestingly, the attribution patterns of the two followers that did not retrieve their leader is the same: these two followers have the same original leader and managed to retrieve the leader's features using the features of other followers. The investor-lag attention weights shown in Fig. 5.7**(b)** confirm this. On the leader side, we see that the leader that learned to follow herself also learned to pay attention to a lag of 0, i.e., the current time-step. Most leaders have 0 attention weights on both lags — the network completely failed to learn the behavior of these leaders, and only provided them with 0 features. On the follower side, we see that almost all followers learned to pay attention to the lag of 1. The two followers that did not are the ones that recomposed the features of their leader with followers — we see here that they pay attention to a lag of 0, with a weight we can interpret as a scaling factor which counters the summation of many investors.

Let us now examine the $\beta = 0.2$ experiment, illustrated in Fig. 5.8. In this setting, the decision function of followers depends on both their features and the features of their leader. We see in Fig. 5.8**(a)** that with a few exceptions, almost all investors learnt to pay attention to themselves. However, no follower learned to pay attention to a leader — the network completely failed to retrieve lead-lag relationships in this setting. This behavior is confirmed in Fig. 5.8**(b)**, where we see that no weight was given for anyone to lags of 1.



(a) **(b)**

Figure 5.7: Heatmap visualization of investor-investor and investor-lag attention weights for the $\beta = 0$ experiment. **(a)** Investor-investor attention weights. Followers almost all retrieved their leader, but leaders did not learn to follow themselves. **(b)** Investor-lag attention weights. Only one leader learnt to pay attention to the right lag, and most leaders do not pay attention to any lag, and consequently gets $\tilde{x_t} = 0$, $\forall t$. Followers mostly learnt to pay attention to the right lag.



(a) **(b)**

Figure 5.8: Heatmap visualization of investor-investor and investor-lag attention weights for the $\beta = 0.2$ experiment. **(a)** Investor-investor attention weights. All investors learnt to follow themselves, but followers did not retrieve their leaders at all. **(b)** Investor-lag attention weights. We see that all investors learnt to focus on lag 0, and totally discarded the lag of 1.

We see from these two experiments that our proposal is able to retrieve lead-lag relationships in the $\beta = 0$ setting — when most investors only act according to a lagged set of their leader's features, the network is able to detect them, at the expense of the leaders who only followed themselves. In the $\beta = 0.2$ setting, we see the opposite phenomenon — as everyone needs to follow herself, the network completely discards the lead-lag relationships in the dataset. Of the many variations of the attention weights formulas that were tried, the current one gave the most promising results, although still clearly insufficient. There are a few ways we could try to improve this model:

- It follows from these two experiments that the proposed architecture has trouble paying attention to both each investor and their potential leaders. The *self-attention* of investors to themselves could be enforced in the model by choosing only to use the attention module

to detect lead-lag and not self-oriented relationships, i.e., by adding the features of a given investor at a given time-step to the result of the attention module depicted in Fig. 5.6. However, such a model would be entirely designed to mimic the mathematical formulation of the problem, consequently introducing bias in the process.

- In the current methodology, we enforce sparsity by penalizing the entropy of the attention weights in the same way as in Section 3.2.2. However, this penalty enforces to get either 0 or 1 but does not prevent many 1 from happening. This penalty term could, therefore, be coupled to a L_0 penalty, as is done in Guo et al. (2019) with a term

$$l_* = \frac{1}{|U|} \sum_{u \in U} \left(\max \left(\sum_i \alpha_{*,u,i} - k_*, 0 \right) \right)^2, \tag{5.12}$$

with $* \in \{cl, lag\}$, $\alpha_{*,u,i}$ the i-th term of the attention weight vector $\alpha_{*,u}$ and k_* the maximal number of admissible weights used for $*$. Such a penalty would help correct the behavior shown for the two outlying followers of the $\beta = 0$ experiment shown in Fig. 5.7.

- A problem of the current methodology could be that the network cannot really explore different lag or leaders' configurations before having to exploit its findings. Exploration of different configurations could be enhanced with the Gumbel-Softmax sampling strategy outlined in Section 3.5.2.

These three ideas could improve the current results and hopefully lead to satisfactory lead-lag relationships retrieval on synthetic data. If the first results of this attempt at lead-lag detection using neural network architectures are promising, further research is required to get a fully functional model that could work on both synthetic and real-world data.

5.4 Towards history-augmented experts

Chapters 3 and 4 introduced algorithms respectively aiming at taking into account the heterogeneity of investment strategies and at introducing time in future interests' prediction using clients' and assets' trading histories. These two goals are not orthogonal, and could both participate in the success of a financial recommender system. In this section, we explore to a greater extent how the ideas underpinning the ExNet and HCF algorithms could be merged into a new *history-augmented experts network* algorithm.

The ExNet algorithm revolves around the idea that not all investors follow the same decision function, and that the set of potential decision functions should be made explicit in the network architecture to uncover post-training the structure of relationships between investors' behaviors. In classic matrix factorization algorithms such as the ones introduced in Section 1.3.3, the decision function corresponds to the scalar product between user and item latent representations. Consequently, the user and item embeddings encode features deemed useful by the model during training for interests' prediction. As all users (resp. items) have their own embeddings, the model can learn to encode different features for each. Consequently, matrix factorization algorithms, to a certain extent, already personalize decision functions — the structure explicitly uncovered in ExNets with the gating block is tantamount to the result we would obtain by using clustering methods directly on the users' embeddings of a matrix factorization algorithm.

However, the history-enhancement step of the HCF algorithm introduces a level of complexity that might not be relevant to share between all users or items. Taking the user-side, the composition of a user's embedding and the embedding of her recent item history should, e.g., be different for users in search of novelty and users that rely heavily on their past behavior. Consequently, the convolution module, which takes care of the transformation of a static user embedding into a time-aware user embedding, could be considered as an *expert* of the HCF user

block. Moreover, this remark also holds on the item-side: for a popular item, less care might be taken to its history than for a less popular one that heavily relies on a small user pool. A proposal architecture using experts to span the space of compositions into dynamic embeddings is illustrated in Fig. 5.9. This network architecture is similar to the ExNet architecture and adapted to the user-item symmetry introduced in HCF.

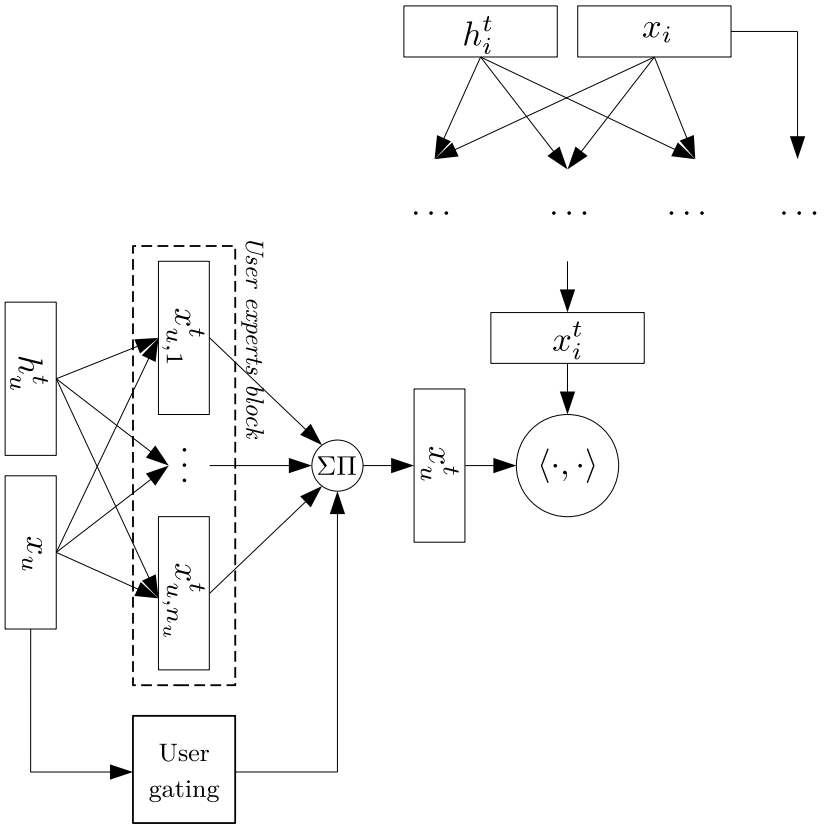


Figure 5.9: A proposal neural network architecture merging the ideas underpinning the ExNet and HCF algorithms. The network is composed of a user and item blocks, as HCF, and each block is decomposed into n_* experts, as ExNets. A gating block outputs the attribution weights of users (resp. items) to their experts. As in ExNets, these weights $p(\cdot|*)$, $* \in \{u, i\}$, are such that $p(e|*) \geq 0 \forall e \in \{1, \ldots, n_*\}$ and $\sum_{e=1}^{n_*} p(e|*)$.

The network uses two user and item blocks decomposed into respectively n_u and n_i experts. Each block retrieves the final embeddings x_u^t and x_i^t using the embeddings computed at each expert's level, as

$$x_*^t = \sum_{e=1}^{n_*} p(e|*) x_{*,e}^t, \tag{5.13}$$

where $* \in \{u, i\}$, n_* denotes the number of experts used on the $*$-side, $p(e|*)$ is the attribution weight of $*$ to expert e as outputted by the gating block of the $*$-side and $x_{*,e}^t$ is the dynamic embedding outputted by expert e. As we expect the challenges of experts' disambiguation and specialization (see Sections 3.2.2 and 3.2.3) to be present as well in this setting, the entropy and specialization loss terms should also be used.

Note that this proposal architecture uses the static embeddings of users and items as gating inputs. To match the dynamic aspects introduced with HCF, the gating blocks could use the

same convolution modules as the experts of this network and become time-dependent. This architecture is also fully compatible with the "featurized" version of HCF introduced in Section 4.4.3. Whether this proposal architecture and its featurized variation are superior to both the ExNet and HCF architectures or not is yet to be proven, and will be the subject of further research.

Conclusions and perspectives

Recommender systems must be adapted to the context of a corporate and institutional bank to provide relevant predictions. In this thesis, we prove the benefits of deep learning methodologies for the future interest prediction problem with *ad hoc* algorithms specifically designed to tackle challenges of the financial world.

With the Experts Network (ExNet) algorithm, we address the heterogeneity of investors' behaviors in financial markets. This custom neural network architecture obtains competitive results with respect to gradient tree boosting algorithms, which are particularly successful approaches in tabular data contexts. We demonstrate that the ExNet algorithm is able to retrieve clusters of investors showing different trading frequencies, and, more interestingly, different trading strategies. Moreover, the network outputs an *a posteriori* map of investors explicitly unveiling their behavioral similarities in the considered market. The History-augmented Collaborative Filtering (HCF) algorithm solves the problem of the non-stationarity of investors' behaviors. We show that this bespoke architecture obtains overall better results than competitor matrix factorization-based algorithms, but also that the results of HCF remain stable in time without having to regularly re-train the algorithm.

The ExNet and HCF algorithms used to solve these challenges are novel neural network architectures that respectively define and improve the state-of-the-art for these specific tasks. The performance of these algorithms can even be further enhanced using overall improvements such as the focal BPR loss or our temporal validation strategy in a non-stationary context.

Still, various research directions can be pursued to continue and extend the work of this thesis. ExNets can be robustified by using the Gumbel-Softmax strategy in their gating block. ExNets could also be staged — the presented architecture is only composed of one gating stage, but more complex gating structures could be tried to find finer-grained clusterings of investors. Moreover, ExNets detect correlations between investors' behaviors: they could consequently be leveraged in the lead-lag detection research. The approach taken for the HCF algorithm can also be further improved. Mainly, what appears to be the most relevant lead is to personalize the history sizes for users and items. Indeed, a more accurate representation of the users' and items' underlying dynamics could be obtained with a customized history size, depending, for instance, on users' and items' activity. Moreover, the introduction of time leads to a new sampling direction for BPR optimization that is not fully leveraged yet.

The algorithm introduced in Chapter 5, blending the underlying ideas of the ExNet and HCF algorithms, is also yet to be implemented and duly experimented. The architecture, conceptually simple, is however computationally very demanding — the following years will most certainly provide us with the appropriate tools for training such architectures in a reasonable amount of time. Finally, the future interest predictions provided by our deep learning algorithms should be made more directly interpretable to help salespeople better present the recommendations we provide to the clients of the bank.

I sincerely hope that the research presented in this thesis will be helpful to both academic researchers wanting to know more about the application of recommender systems to the financial world, and to industrial researchers aiming at improving the results of their future interest prediction algorithms.

Conclusions et perspectives

Les systèmes de recommandation doivent être adaptés au contexte d'une banque d'investissement pour fournir des prédictions pertinentes. Dans cette thèse, nous faisons la preuve de l'utilité des méthodologies d'apprentissage profond pour le problème de prédiction des intérêts futurs à l'aide d'algorithmes *ad hoc* spécifiquement conçus pour faire face aux défis du monde financier.

Avec l'algorithme de réseau d'experts (ExNet), nous adressons le problème de l'hétérogénéité de comportement des investisseurs sur les marchés financiers. Cette architecture de réseau de neurones nouvelle obtient des résultats compétitifs vis-à-vis d'approches de *gradient boosting* utilisant des arbres de classification, approches particulièrement fructueuses dans le contexte de données tabulaires. Nous démontrons que l'algorithme ExNet est à même de retrouver des groupes d'investisseurs suivant leurs fréquences d'investissement, mais aussi suivant leurs stratégies d'investissement. De plus, le réseau fournit un graphe *a posteriori* d'investisseurs dévoilant de façon explicite leurs similarités comportementales sur le marché considéré. L'algorithme de filtrage collaboratif augmenté par historiques (HCF) résout le problème de la non-stationnarité des comportements des investisseurs. Nous montrons que cette architecture, conçue pour ce problème, obtient de meilleures performances que des approches concurrentes aux spécifications proches, mais également que les résultats obtenus avec HCF sont stables dans le temps, évitant ainsi d'avoir à ré-entraîner l'algorithme de façon régulière.

Les algorithmes ExNet et HCF utilisés pour résoudre ces défis sont des architectures de réseau de neurones nouvelles qui respectivement définissent et améliorent l'état de l'art pour ces tâches spécifiques. Par ailleurs, les résultats obtenus avec ces algorithmes peuvent encore être améliorés à l'aide de modifications globales telles que la version focale de la fonction de coût de *Bayesian Personalized Ranking* (BPR), i.e., classement bayésien personnalisé, ou notre stratégie de validation temporelle pour contextes non-stationnaires.

De nombreuses directions de recherche peuvent cependant encore être poursuivies pour continuer et étendre le travail présenté dans cette thèse. Les ExNets peuvent être rendus plus robustes par l'utilisation de la distribution Gumbel-Softmax dans le bloc de *gating*, i.e., de répartition des investisseurs considérés. La répartition faite par les ExNets peut également se faire sur plusieurs niveaux — l'architecture présentée ne comporte qu'un niveau de répartition, mais des structures plus complexes, à plusieurs niveaux de granularité, pourraient mener à des partitionnements plus fins des investisseurs. Par ailleurs, les ExNets détectent les corrélations entre les comportements des investisseurs: ils pourraient donc être exploités dans une approche de détection de comportements meneur-suiveur sur les marchés. L'approche considérée pour l'algorithme HCF peut également être améliorée de plusieurs façons. En particulier, la personnalisation des tailles d'historique utilisées pour les clients et produits apparaît comme l'une des meilleures pistes d'amélioration potentielles. En effet, une représentation plus précise des dynamiques des clients et produits pourrait être obtenue via une taille d'historique adaptée à chaque, et dépendant, par exemple, de leur activité. De plus, l'introduction du temps dans l'algorithme ajoute une direction potentielle de tirage aléatoire pour l'optimisation via BPR qui

n'a jusqu'ici pas été pleinement prise en compte.

L'algorithme introduit au chapitre 5 mariant les idées des algorithmes ExNet et HCF est également encore à implémenter et tester comme il se doit. L'architecture, conceptuellement simple, demande cependant de grandes ressources de calcul — les prochaines années fourniront assurément des outils plus appropriés pour entraîner de telles architectures en un temps raisonnable. Enfin, les prédictions d'intérêts futurs fournies par nos algorithmes d'apprentissage profond devraient être rendues plus directement interprétables pour faciliter la présentation de nos recommandations par les vendeurs aux clients de la banque.

J'espère sincèrement que la recherche présentée dans cette thèse pourra être utile à la fois aux chercheurs académiques souhaitant en connaître davantage sur l'application des systèmes de recommandation au monde financier ainsi qu'aux chercheurs du secteur privé cherchant à améliorer les résultats de leurs algorithmes de prédiction des intérêts futurs.

Bibliography

K. Baltakys, J. Kanniainen, and F. Emmert-Streib. Multilayer aggregation with statistical validation: Application to investor networks. *Scientific Reports*, 8(1):8198, 2018.

J. Basilico and T. Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first International Conference on Machine Learning*, page 9, 2004.

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995.

J. Bennett, S. Lanning, et al. The Netflix Prize. In *Proceedings of KDD Cup and Workshop*, volume 2007, page 35. New York, NY, USA., 2007.

N. Berestycki. Mixing times of markov chains: Techniques and examples. *Latin American Journal of Probability and Mathematical Statistics (ALEA)*, 2016.

C. Bergmeir and J. M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

L. Bernardi, J. Kamps, J. Kiseleva, and M. J. Müller. The continuous cold start problem in e-commerce recommender systems. 2015.

F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

M. J. Bland and D. G. Altman. Multiple significance tests: the Bonferroni method. *British Medical Journal*, 310(6973):170, 1995.

Bloomberg. Bloomberg Professional Services. `https://www.bloomberg.com/professional/`, 2020. Last accessed on 04/01/2020.

Bloomberg Professional Services. Sure time to grasp the potential of structured products. `https://www.bloomberg.com/professional/blog/sure-time-to-grasp-the-potential-of-structured-products/`, 2019. Last accessed on 06/01/2020.

J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-based Systems*, 46:109–132, 2013.

L. Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

D. Challet, R. Chicheportiche, M. Lallouache, and S. Kassibrakis. Statistically validated lead-lag networks and inventory prediction in the foreign exchange market. *Advances in Complex Systems*, 21(08):1850019, 2018.

O. Chapelle and M. Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information Retrieval*, 13(3):216–235, 2010.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and P. W. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the twenty-second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the first Workshop on Deep Learning for Recommender Systems*, pages 7–10, 2016.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. *International Conference on Learning Representations*, 2020.

P. Covington, J. Adams, and E. Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the tenth ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

C. Curme, M. Tumminello, R. N. Mantegna, E. H. Stanley, and D. Y. Kenett. Emergence of statistically validated financial intraday lead-lag relationships. *Quantitative Finance*, 15(8): 1375–1386, 2015.

M. F. Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the thirteenth ACM Conference on Recommender Systems*, pages 101–109, 2019.

Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the fifty-seventh Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the twenty-third International Conference on Machine Learning*, pages 233–240. ACM, 2006.

M. L. De Prado. *Advances in financial machine learning*, chapter 7. John Wiley & Sons, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

J. Ding, G. Yu, X. He, F. Feng, Y. Li, and D. Jin. Sampler design for bayesian personalized ranking by leveraging view data. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

Y. Ding and X. Li. Time weight collaborative filtering. In *Proceedings of the fourteenth ACM International Conference on Information and Knowledge Management*, pages 485–492, 2005.

P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

T. Dozat. Incorporating Nesterov momentum into Adam. 2016.

E. Eban, M. Schain, A. Mackey, A. Gordon, R. Rifkin, and G. Elidan. Scalable learning of non-decomposable objectives. In *Artificial Intelligence and Statistics*, pages 832–840, 2017.

C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference*, pages 1775–1784, 2018.

M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

F. J. Fabozzi. *Bond Markets, Analysis and Strategies — 8th Edition*. Prentice Hall, 2012.

H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

J.-D. Fermanian, O. Guéant, and J. Pu. The behavior of dealers and clients on the european corporate bond market: the case of multi-dealer-to-client platforms. *Market Microstructure and Liquidity*, 2, 2016.

P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

T. Galanti, L. Wolf, and T. Hazan. A theoretical framework for deep transfer learning. *Information and Inference: A Journal of the IMA*, 5(2):159–209, 2016.

D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

O. Guéant, C.-A. Lehalle, and J. Fernandez-Tapia. Dealing with the inventory risk: a solution to the market making problem. *Mathematics and Financial Economics*, 7(4):477–507, 2013.

Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris. SpotTune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019.

M. Gutiérrez-Roig, J. Borge-Holthoefer, A. Arenas, and J. Perelló. Mapping individual behavior in financial markets: synchronization and anticipation. *EPJ Data Science*, 8(1):10, 2019.

M. Haldar, M. Abdool, P. Ramanathan, T. Xu, S. Yang, H. Duan, Q. Zhang, N. Barrow-Williams, B. C. Turnbull, B. M. Collins, et al. Applying deep learning to AirBnB search. In *Proceedings of the twenty-fifth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1927–1935, 2019.

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

S. J. Hardiman, N. Bercot, and J.-P. Bouchaud. Critical reflexivity in financial markets: a Hawkes process analysis. *The European Physical Journal B*, 86(10):442, 2013.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182, 2017.

X. He, Z. He, X. Du, and T.-S. Chua. Adversarial personalized ranking for recommendation. In *The forty-first International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 355–364, 2018.

P. Henderson and V. Ferrari. End-to-end training of object class detectors for mean average precision. In *Asian Conference on Computer Vision*, pages 198–213. Springer, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

K. Hornik, M. Stinchcombe, H. White, et al. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5(Nov):1457–1469, 2004.

Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.

F. Huang, J. Ash, J. Langford, and R. Schapire. Learning deep ResNet blocks sequentially using boosting theory. In *Proceedings of the thirty-fifth International Conference on Machine Learning*, 2018.

J. C. Hull. *Options, futures and other derivatives — 9th Edition.* Pearson Education, 2014.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

ISIN. ISIN Organization. `https://www.isin.org/`, 2020. Last accessed on 04/03/2020.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-Softmax. *International Conference on Learning Representations*, 2017.

Kaggle Inc. Kaggle. `https://www.kaggle.com/`, 2020. Last accessed on 04/23/2020.

A. Karpathy. Stanford CS231n Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/convolutional-networks/`, 2016. Accessed: 2020-05-11.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). *arXiv preprint arXiv:1711.11279*, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–434, 2008.

Y. Koren. The BellKor solution to the Netflix grand prize. *Netflix Prize Documentation*, 81 (2009):1–10, 2009a.

Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the fifteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 447–456, 2009b.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the nineteenth International Conference on World Wide Web*, pages 661–670, 2010.

T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988, 2017.

Y. Liu and X. Yao. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(6): 716–725, 1999.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations*, 2017.

C. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 16, chapter 8.4, pages 100–103. Cambridge University Press, 2010.

L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

A. Mnih and R. R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2008.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2013.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.

F. Musciotto, L. Marotta, J. Piilo, and R. N. Mantegna. Long-term ecology of investors in a financial market. *Palgrave Communications*, 4(1):92, 2018.

V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

E. Negre. *Information and Recommender Systems*. John Wiley & Sons, 2015.

Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

D. L. Padial. Technical Analysis Library using Pandas. `https://github.com/bukosabino/ta`, 2018.

Y.-J. Park and A. Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, pages 11–18, 2008.

I. Pilászy, D. Zibriczky, and D. Tikk. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM Conference on Recommender Systems*, pages 71–78, 2010.

J. Racine. Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of Econometrics*, 99(1):39–61, 2000.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the twenty-second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

P. Sarkar, S. M. Siddiqi, and G. J. Gordon. A latent space approach to dynamic embedding of co-occurrence data. In *Artificial Intelligence and Statistics*, pages 420–427, 2007.

L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28): 307–317, 1953.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *International Conference on Learning Representations*, 2017.

Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver. TFMAP: optimizing MAP for top-n context-aware recommendation. In *Proceedings of the thiry-fifth International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 155–164, 2012.

Y. Shi, M. Larson, and A. Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)*, 47 (1):1–45, 2014.

A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *Proceedings of the thirty-fourth International Conference on Machine Learning*, pages 3145–3153. JMLR. org, 2017.

J. Sirignano and R. Cont. Universal features of price formation in financial markets: Perspectives from deep learning. *Quantitative Finance*, 19(9):1449–1459, 2019.

D. Sorokina and E. Cantu-Paz. Amazon search: The joy of ranking products. In *Proceedings of the thiry-ninth International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 459–460, 2016.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15 (1):1929–1958, 2014.

X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, 2009.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM Conference on Recommender Systems*, pages 83–90, 2012.

The European Commission. *Markets in Financial Instruments Directive, II — Scopes and Definitions*. 2014.

A. Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2):58, 2004.

M. Tumminello, S. Micciche, F. Lillo, J. Piilo, and R. N. Mantegna. Statistically validated networks in bipartite complex systems. *PloS one*, 6(3), 2011.

M. Tumminello, F. Lillo, J. Piilo, and R. N. Mantegna. Identification of clusters of investors from their real trading activity in a financial market. *New Journal of Physics*, 14(1):013041, 2012.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

H. Wang, Q. Wu, and H. Wang. Learning hidden features for contextual bandits. In *Proceedings of the twenty-fifth ACM International Conference on Information and Knowledge Management*, pages 1633–1642, 2016.

X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua. Neural graph collaborative filtering. In *Proceedings of the fourty-second International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 165–174, 2019.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

D. Wright, L. Capriotti, and J. Lee. Machine learning and corporate bond trading. *Algorithmic Finance*, 7(3-4):105–110, 2018.

C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing. Recurrent recommender networks. In *Proceedings of the tenth ACM International Conference on Web Search and Data Mining*, pages 495–503, 2017.

X. Wu, B. Shi, Y. Dong, C. Huang, and N. Chawla. Neural tensor factorization. *arXiv preprint arXiv:1802.04416*, 2018.

L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the sixteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 723–732, 2010.

L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 211–222. SIAM, 2010.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, pages 5754–5764, 2019.

S. E. Yuksel, J. N. Wilson, and P. D. Gader. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193, 2012.

A. Zeng, A. Vidmer, M. Medo, and Y.-C. Zhang. Information filtering by similarity-preferential diffusion processes. *EPL (Europhysics Letters)*, 105(5):58002, 2014.

M. R. Zhang, J. Lucas, G. Hinton, and J. Ba. Lookahead Optimizer: k steps forward, 1 step back. *arXiv preprint arXiv:1907.08610*, 2019.

W. Zhang, T. Chen, J. Wang, and Y. Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the thirty-sixth International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 785–788, 2013.

G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176, 2018.

T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang. Bipartite network projection and personal recommendation. *Physical Review E*, 76(4):046115, 2007.

T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

List of Figures

List of Tables

Glossary

CIB. Corporate and Institutional Bank.
Sales. Financial lingo referring to salespeople.
D2C. Dealer-to-Client. Refers to electronic platforms where clients can stream prices for many financial assets as given by market makers.
RFQ. Request For Quotation. A process in which a client declares her interest into buying/selling a given amount of a given financial asset.
RFM. Request For Market. A process in which a client declares her interest in a given financial asset.
IOI. Indication of Interest. Indication recorded by sales that a given client might have an interest for a given financial asset.
G10. Refers to the G10 group of countries, composed of Germany, Belgium, Canada, United States of America, France, Italy, Japan, Netherlands, United Kingdom, Sweden, and Switzerland.
NLP. Natural Language Processing. Refers to a subfield of machine learning interested into the understanding of human language.
MLP. Multi-Layer Perceptron. Refers to a feed-forward, fully-connected neural network. See Appendix A for more information.
LGBM. Refers to the LightGBM algorithm (Ke et al., 2017).
LSTM. Long-Short Term Memory. A specific type of recurrent neural networks introduced in Hochreiter and Schmidhuber (1997).
ExNet. Experts Network. Refers to a novel neural network architecture introduced in this manuscript in Chapter 3.
HCF. History-augmented Collaborative Filtering. Refers to a novel neural network architecture introduced in this manuscript in Chapter 4.
MF. Matrix Factorization. A collaborative filtering algorithm outlined in Section 1.3.3.
NMF. Nonnegative Matrix Factorization. A collaborative filtering algorithm outlined in Section 1.3.3.
BPR. Bayesian Personalized Ranking. A loss function used in collaborative filtering, and introduced in Section 1.3.3.
UMAP. Uniform Manifold Approximation and Projection. A dimension reduction algorithm introduced in McInnes et al. (2018).
AUC ROC. Area Under the Receiver Operating Characteristic Curve. Also encountered as AUC, or ROC AUC. A binary classification metric.
AP. Average Precision. A binary classification metric.
mAP. Mean Average Precision. In this work, used for short of symmetrized mean average precision, a metric introduced in Section 1.4.

Appendix A

Basics of deep learning

The purpose of this appendix is to delve into the mathematics onto which deep learning lies. The structure of this appendix is as follows:

- Introducing the general principles on which the field of deep learning relies;
- Providing the reader with the mathematical tools required for training neural networks, i.e., the gradient descent and backpropagation algorithms;
- Thoroughly presenting the concepts behind the two classic neural network architectures of multi-layer perceptrons and convolutional neural networks.

For ease of understanding, we focus on a supervised learning framework throughout this appendix.

A.1 General principles of neural networks

This section introduces general principles of neural networks, from the historical perceptron model to multi-layer perceptrons and presents heuristics and ideas to guide the design of neural network architectures.

A.1.1 Perceptrons

Neural networks are a very popular toolbox to tackle supervised learning problems, from natural language processing tasks to computer vision ones. To understand their popularity, we trace back their history, relying mainly on the outline given by Nielsen (2015). In 1957, Frank Rosenblatt introduced the *perceptron*, a model broadly mimicking the functioning of a biological neuron: a perceptron receives inputs and then makes the decision to fire or not fire based on these inputs. The model of the perceptron is illustrated in Fig. A.1.

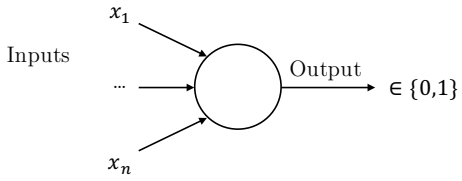


Figure A.1: Global outline of the perceptron model. A perceptron takes inputs $x_1, ..., x_n$ and outputs either 0 or 1, mimicking the behavior of a biological neuron.

The perceptron takes inputs $x_1, ..., x_n$ and outputs either 0 or 1, corresponding to the firing of

a biological neuron. Firing is determined using a set of *weights* $\{w_1, ..., w_n\}$ and a *bias* denoted b, according to

$$output = \begin{cases} 0 & \text{if } w \cdot x + b \leqslant 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{A.1}$$

By carefully tuning the weights and bias, the perceptron attributes to a given set of inputs a value of either 0 or 1. Fig. A.2 illustrates a simple perceptron based on the decision rule $3x - 2y + 1 > 0$.

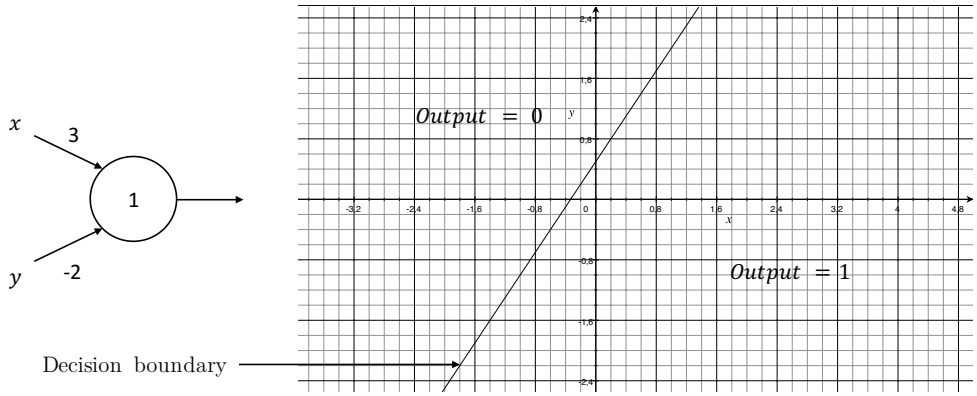


Figure A.2: A simple perceptron and its $\mathbb{R}^2$ representation. This perceptron corresponds to the decision rule $3x - 2y + 1 > 0$.

This example highlights the fact that a perceptron can be seen as a **linear classifier**, and that the corresponding decision boundary is given by the equation

$$\langle w, x \rangle + b = 0. \tag{A.2}$$

On one side of this boundary, the output of the perceptron is 1; on the other side, including the boundary itself, the output is 0. In $\mathbb{R}^n$, the decision boundary corresponds to a hyperplane. Following on this, we see that perceptrons are able to solve linearly separable problems. But perceptrons can be easily connected together by considering the outputs of some as the inputs of others, thereby creating a *network* of perceptrons — a *neural network*. The hope is then that such a network is able to solve more difficult problems than linearly separable ones.

Ler us consider the *NAND perceptron* to go further on this idea. Recall that the NAND gate, which takes as input two binary variables x_1 and x_2 and outputs x_1 NAND x_2, i.e., the *not and* logic operation, is a *universal logic gate*. This means that from the NAND gate, we are able to construct any other boolean function, and thereby performing any calculation. Consequently, if we are capable of mimicking the NAND gate with a perceptron, a neural network can theoretically compute any function.

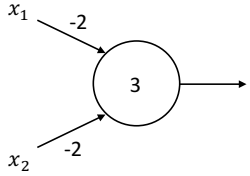


Figure A.3: The NAND perceptron, defined by the equation $-2x_1 - 2x_2 + 3 > 0$. The only pair (x_1, x_2) which gives 0 as output is the pair $(1, 1)$, corresponding to the *not and* logic operation.

The perceptron depicted in Fig. A.3 is equivalent to the NAND logic operator. The only pair (x_1, x_2) which gives 0 as output is the pair $(1, 1)$. Perceptrons have thus the computational universality property of the NAND gate, provided that all weights and biases are set in advance.

However, perceptrons are neither a simple ersatz of NAND gates nor another way to design logic circuits. The main interest of perceptrons and neural networks is their *learning property*. Instead of manually designing a network to solve a given problem, a neural network can learn by itself how to adjust and tune its weights and biases for the problem at hand. Learning is permitted by the gradient descent and backpropagation algorithms, that will be further explored in this appendix.

A.1.2 Multi-layer perceptrons

The properties of perceptrons outlined in Section A.1.1 can be seen as a first motivation to "go deep" and design large neural networks, although we lack theoretical guarantees for now. Let us focus first on a prerequisite for the learning property to be valid.

Perceptrons have binary outputs, as previously seen. However, it is very doubtful that any learning can happen in such a setting. Assume that we have a learning process able to carefully tune the weights and biases of a neural network — if the weights and biases of a given perceptron in the network are such that $w \cdot x + b \approx 0$, even a small change in either w or b is very likely to make the perceptron reverse its decision, and thereby completely change the decision of the whole network. Consequently, we need to introduce the concept of **activation functions**. Instead of binary outputs, consider now that neuron outputs are computed as $f(w \cdot x + b)$, where f is a given non-linear function. f needs to be non-linear, otherwise a neural network of such units would reduce to a single layer of neurons, since the set of linear functions is closed under composition. The differential of the activation function shows us that this new output is such that

$$df = \sum_j \frac{\partial f}{\partial w_j} dw_j + \frac{\partial f}{\partial b} db. \tag{A.3}$$

Consequently, a small change in weights or bias causes a small change in the output, proportionally to the partial derivative of the changed quantity. Historically, the first activation function considered was the sigmoid function $\sigma(x) := (1 + e^{-x})^{-1}$. Since a perceptron is a neuron which activation function is the Heaviside step function, a *sigmoid neuron* performs a smooth, continuous approximation of the perceptron. Classic activation functions include functions with a shape approximating the Heaviside function such as the sigmoid and hyperbolic tangent functions, or functions with interesting derivatives, such as the ReLU function defined by $f(x) = \max(0, x)$ which gradient is 1 for all positive inputs.

We will henceforth focus our study on neural networks composed of such neurons. These networks are called, by extension, *multi-layer perceptrons*. Fig. A.4 illustrates this concept with the example of a four-layer perceptron. Each neuron composing the network is linked to the others, such that each layer takes as inputs the outputs of the previous layer.

In fact, a theoretical guarantee ensures that multi-layer perceptrons are universal approximators. This guarantee can be stated as the following:

Theorem A.1.1. *For any measurable function mapping a finite-dimensional space to another finite-dimensional space, there exists a neural network with one hidden layer which can approximate this function to any desired degree of accuracy. This holds regardless of the activation*

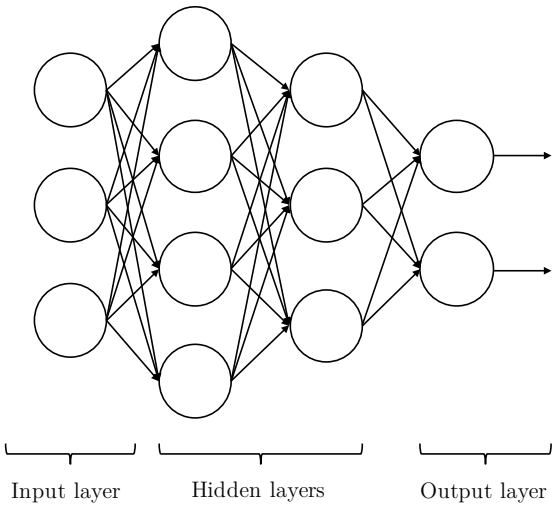


Figure A.4: An example of multi-layer perceptron. This multi-layer perceptron is composed of four layers. Intermediary layers are called *hidden*.

function used.

This theorem guarantees that neural networks with one hidden layer are universal function approximators. Proofs of such results are quite technical, and can be found for instance in Hornik et al. (1989) for the case of continuous activation functions and *squashing* functions, which are functions respectively tending to 0 and 1 as their input tends towards $-\infty$ and $+\infty$. As reassuring as this result can be, it only concerns relatively shallow neural networks. In that sense, the need to go deep does not rely on theory, but merely on practice and experimentation.

A.1.3 Designing deep neural networks

Let us now derive a mathematical formulation of neural networks, following a framework used, among others, in Galanti et al. (2016). A neural network architecture is a set (V, E, f), where V denotes the set of neurons, E the set of directed edges, and f the activation function. We represent the weights and biaises of the network with a weight function $w : E \to \mathbb{R}$ — the biases are included with the assumption that each layer includes a "bias" neuron, linked to all the neurons of the following layer only. Following on this, we can classify neural networks according to their architecture (V, E, f), defining the class $\mathcal{C}_{(V,E,f)}$ as the set of all neural networks which architecture is (V, E, f).

Neurons are organized in disjoint layers $L_0, ..., L_N$, such that $V = \cup_{i=0}^{N} L_i$. The number of neurons in a given layer, $|L_i|$, is called the *size* of the layer L_i. Using the graph terminology, we talk about *fully-connected layers* when, $\forall i \in \{0, ...N-1\}$, all neurons in layer L_i are connected to all neurons in layer L_{i+1}. A *feed-forward neural network* is a neural network which graph contains no directed cycles. Using these notations, we can therefore state that the class of multi-layer perceptrons is the set of all classes $\mathcal{C}_{(V,E,f)}$ such that the resulting network is feed-forward, with fully-connected layers.

We call a neural network *deep* when it has more than one hidden layer, i.e. when $N > 2$. Otherwise, we call it *shallow*. One of the main problems of deep learning is the design of such networks, i.e., choosing the appropriate neural network class $\mathcal{C}_{(V,E,f)}$ to solve a given problem. The sizes of the input and output layers depend entirely on the problem at hand, and are consequently easily found. For instance, let us consider that we want to build a network detecting the presence of cars and planes in an image. The input layer would take images —

assuming that we are working with 128*128 RGB images, the size of the input, i.e., of the L_0 layer, would be of $128 * 128 * 3 = 49152$. The output layer would consist of two neurons, each giving the probability that there is, say, a car for the first one, and a plane for the second one.

What is however more problematic is the design of the hidden layers. How many layers should we use, and with how many neurons in each? There is, as of now, no mathematical theory that can guide us towards what would be considered a "good" or "efficient" design. We have to rely on heuristics, which are often only useful for given sets of problems. The number of hidden layers and neurons in them are a part of the *hyperparameters* of the network, parameters that we have to tune and engineer carefully in order to get the network to solve the problem we want.

A classic heuristic is to adopt a *diamond* shape in the design of hidden layers. Starting with the input layer, we use several hidden layers with a growing number of neurons, and then we decrease this number so that it comes back to the size of the output layer. When doing so, we immerse the problem we want to solve in a higher-dimensional space, where it is supposedly more easily solved, and we then go back smoothly to the output space dimension. A common idea is also to use powers of 2 as hidden layer sizes. Doing this allows ones to see more clearly how the network reacts when we increase or reduce the number of neurons in a given layer: it is doubtful that adding a few neurons only would have an effect on the efficiency of a network, and this rule consequently guides the choice of layer sizes. Such ideas and heuristics, commonly used by practitioners, can be found for instance in Bengio (2012).

A.2 Stochastic gradient descent

In *deep learning*, learning is usually performed using two algorithms, one of which being the *stochastic gradient descent* (SGD) or one of its numerous variations. In this section, we introduce the gradient descent algorithm from which SGD is derived, and then briefly mention a couple useful variations of SGD.

A.2.1 Defining the optimization problem

An optimization problem is composed of three distinct parts, as stated in Domingos (2012). These three parts are the following:

- **The model.** The model part of the problem consists of the class of neural networks we consider and the data at our disposal. For example, a given class $\mathcal{C}_{(V,E,f)}$ in the class of multi-layer perceptrons. We suppose that the training data consists of *i.i.d.* samples of an unknown probability distribution, e.g., $(x_i, y_i)_{i=1...n} \sim_{i.i.d.} \mu$. A
- **The loss function.** The *loss function* is the function that we need to optimize using our model. It is a function of the weights and biases of the network that we write $\mathcal{L}(w, b)$. A classic example of loss function is the mean squared error (MSE), which in our case has the following form, writing a_i the output of the network for an input x_i:

$$\mathcal{L}(w, b) = \sum_{i=1}^{n} \|y_i - a_i\|_2^2. \tag{A.4}$$

  For ease of notation, we introduce $v = (w, b)$ such that $\mathcal{L}$ is now a function of v, denoted $\mathcal{L}(v)$.
- **The optimization method.** Also denoted the optimizer in literature, the optimization method is the algorithm we use to search in the considered class of neural networks the one whose weights minimize the loss function using training data. Gradient descent and its variations are examples of such methods, and are the ones we focus on in this appendix.

A.2.2 Deriving the SGD algorithm

Let us now consider optimization methods, and more particularly gradient descent. The gradient descent algorithm is based on the following functional analysis fact.

Proposition A.2.1. *Assume that $\mathcal{L}(v)$ is defined and differentiable in the neighbourhood of a point z. $\mathcal{L}$ decreases fastest in the opposite direction of its gradient $\nabla\mathcal{L}(z)$.*

Using this idea, starting from a given point z_0 and supposing that the loss function has all required regularity, we can construct a series of points $z_1, ...z_n$ such that

$$z_{i+1} = z_i - \eta\nabla\mathcal{L}(z_i) \ \forall i \in \mathbb{N}, \tag{A.5}$$

with, for η small enough, the fact that

$$\mathcal{L}(z_0) \geqslant \mathcal{L}(z_1) \geqslant ... \geqslant \mathcal{L}(z_n). \tag{A.6}$$

We stop when the difference in loss between two steps becomes very small, typically smaller than an $\epsilon > 0$ fixed beforehand. We therefore have an algorithm allowing us to compute a series of point which converges towards a minimum of the loss function. Note that the minimum we find using this algorithm may only be a local minimum of the loss. η is called the **learning rate**, and can be changed at every step of our algorithm.

Assume now that the loss function is **separable**, i.e. that it can be broken down into pieces that only depend on one sample of the training set $\{(x_i, y_i)\}_{i=1...n}$. The loss can then be written

$$\mathcal{L}(v) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i(v). \tag{A.7}$$

Following on this, the gradient $\nabla\mathcal{L}(v)$ can be decomposed in the same manner. But when we compute the sequence of points in the gradient descent algorithm, we would then have to calculate, at each step, $\frac{1}{n}\sum_{i=1}^{n}\nabla\mathcal{L}_i(v)$, which can be computationally very expensive when the training set is large, as is typically required for training deep neural networks.

To overcome this issue, we introduce the idea of stochastic gradient descent. At every step, instead of using all the available training data to calculate the gradient, we only calculate it using a small subset of the training examples picked uniformly at random, that we call a *mini-batch*. The gradient computed on that mini-batch of data approximates the true gradient,

$$dfrac1m\sum_{i\in B}\nabla\mathcal{L}_i(z_j) \approx \nabla\mathcal{L}(z_j), \tag{A.8}$$

where B is a subset of $\{1, ..., n\}$ picked uniformly at random and such that $|B| = m$. Note that this approximator is unbiased. We then update z using this approximated gradient and iterate this process, picking subsets of the remaining training examples until all are used. When the pool of training examples if depleted, we say that we completed an *epoch*. Algorithm 1

summarizes these steps.

Algorithm 1: The stochastic gradient descent algorithm.

Data: Training set $\{(x_i, y_i)\}_{i=1...n} \sim^{i.i.d.} \mu$
Result: A local minimum of the loss function $\mathcal{L}$
Initialize $v = (w, b)$, set the learning rate η;
for $l = 1, ...$ ***epochs*** **do**
  Set the training set to be the whole data;
  for $i = 1...n/m$ **do**
    Pick uniformly at random m training examples from the training set;
    Calculate the approximate gradient $\frac{1}{m}\sum_{j\in B}\nabla\mathcal{L}_j(v)$;
    Update v following $v := v - \eta\frac{1}{m}\sum_{j\in B}\nabla\mathcal{L}_j(v_i)$;
    Subtract chosen examples from the training set;

The mini-batch size m, the number of epochs and the learning rate η are all hyperparameters of the optimization problem, along with the hyperparameters of the neural network architecture.

A.2.3 A couple variations of SGD

The previous section introduced the classic SGD algorithm, but there exist also many other extensions and variants of the underlying idea. We briefly discuss in this section two of its most classic extensions, the *momentum method* and *RMSProp*. A recap of these two methods can also be found in the supplements of Mnih et al. (2016).

The momentum method was first introduced in Rumelhart et al. (1986). This method uses a slightly different updating step, remembering at each step the difference between the previous value of z, z_{i-1}, and the current value z_i; denoting $\Delta_i z = z_i - z_{i-1}$, the momentum update rule is defined as the following, where the gradient is approximated using a mini-batch :

$$z_{i+1} = z_i - \eta\nabla\mathcal{L}(z_i) + \alpha\Delta_i z. \tag{A.9}$$

This update rule is designed following basic physics principles. If we consider that our point z represents a ball on the "hilly space" that is our loss function $\mathcal{L}$, the momentum update has a simple physics analogy. The gradient part of the update corresponds to the force applied to the ball, and the delta part to a viscous force governed by the α coefficient, a new hyperparameter of the problem. A typical value of α is $\alpha = 0.9$. We observe that when using this rule, z is updated along its previous direction, thereby preventing the oscillations one can observe with classic SGD and that are due to the mini-batch approximation.

RMSProp, which stands for *Root Mean Square Propagation*, relies on a different idea. RMSProp uses an adaptative learning rate, which is found at each step using the previous gradients of the loss.

At each step, RMSProp keeps a moving mean of the squared gradients,

$$v(z_i) := \nu v(z_{i-1}) + (1-\nu)(\nabla\mathcal{L}(z_i))^2, \tag{A.10}$$

where ν is called the *forgetting factor*. We then use this mean to adapt the learning rate to the

current step through the update rule defined by

$$z_{i+1} = z_i - \frac{\eta}{\sqrt{v(z_i)}} \nabla \mathcal{L}(z_i). \tag{A.11}$$

All gradients used here are mini-batches approximations. This method, unpublished, was proposed by G. Hinton and T. Tieleman in 2012. Hinton suggests to take $\nu = 0.9$. This method was experimentally shown to adapt the learning rate in an efficient way, thereby accelerating convergence of stochastic gradient descent. At the time of writing this appendix, the most widespread optimizer is Adam (Kingma and Ba, 2015), which efficiently combines the ideas of momentum and RMSProp optimizers.

A.3 Backpropagation algorithm

SGD provides a way to update the weights and biases of a neural network using gradients, but assumes that we can effectively compute these gradients. In deep learning, gradients are computed using the backpropagation algorithm.

Let us first define the notations we use to refer to a neural network. Assuming a neural network of a given class $\mathcal{C}_{(V,E,f)}$ with L layers, for $l = 1, ..., L$, we denote as $w^l_{j,k}$ the weight for the connection of neuron k in layer $(l-1)$ to neuron j in layer l, and b^l_j for the bias for neuron j in layer l. For each layer, we can therefore define a *weight matrix* W^l and a *bias vector* $\mathbf{b}^l$. We also introduce $\mathbf{a}^l$, the *activation vector* of layer l, which components are defined such that

$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right). \tag{A.12}$$

Using these notations, the computations performed by a neural network can be summarized in the following vectorized form:

$$\mathbf{a}^l = f\left(W^l \mathbf{a}^{l-1} + \mathbf{b}^l\right). \tag{A.13}$$

Let us finally introduce the *weighted input* $\mathbf{z}^l$, such that

$$\mathbf{z}^l := W^l \mathbf{a}^{l-1} + \mathbf{b}^l, \tag{A.14}$$

therefore verifying $\mathbf{a}^l = f(\mathbf{z}^l)$.

Backpropagation provides us with a way to compute the partial derivatives of the loss function with regard to any weight $w^l_{j,k}$, i.e., $\frac{\partial \mathcal{L}}{\partial w^l_{j,k}}$, and any bias b^l_j, i.e., $\frac{\partial \mathcal{L}}{\partial b^l_j}$, of our network. Backpropagation relies on two assumptions on the loss function $\mathcal{L}$. The first one is the separability hypothesis that was already required for SGD. The second one is that the partial losses obtained through separability can be written as functions of the outputs of the neural network only, i.e., such that $\mathcal{L}_i = \mathcal{L}_i(a^L)$ — MSE, introduced in Section A.2.1, is an example of such a loss. For ease of notation, working from now on on such losses $\mathcal{L}_i$, we drop the subscript i.

Let us note

$$\delta^l_j := \frac{\partial \mathcal{L}}{\partial z^l_j} \tag{A.15}$$

the *error* of neuron j in layer l, and δ^l the associated vector. The backpropagation algorithm gives us a way to calculate these errors, and to relate them to the partial derivatives of interest $\frac{\partial \mathcal{L}}{\partial w^l_{j,k}}$ and $\frac{\partial \mathcal{L}}{\partial b^l_j}$ that we need for SGD. The algorithm relies on four equations, stated in the following proposition.

Proposition A.3.1. *Using the framework defined above, we have the following equations:*

$$\delta^L = \nabla_{a^L}\mathcal{L} \odot f'(z^L), \tag{A.16}$$

$$\delta^l = ((W^{l+1})^T\delta^{l+1}) \odot f'(z^l) \tag{A.17}$$

$$\frac{\partial \mathcal{L}}{\partial b^l_j} = \delta^l_j \tag{A.18}$$

$$\frac{\partial \mathcal{L}}{\partial w^l_{j,k}} = a^{l-1}_k \delta^l_j \tag{A.19}$$

where $\odot$ designates the Hadamard product, i.e. the component-wise vector product.

Proof. All these equations rely on the chain rule of derivation.

A.16 : Recall that $\delta^L_j = \frac{\partial \mathcal{L}}{\partial z^L_j}$. Applying the chain rule, this equation becomes $\delta^L_j = \sum_k \frac{\partial \mathcal{L}}{\partial a^L_k}\frac{\partial a^L_k}{\partial z^L_j}$. But, recalling the definition of a^L, $\frac{\partial a^L_k}{\partial z^L_j}$ is nil for all $k \neq j$, and thus we get $\delta^L_j = \frac{\partial \mathcal{L}}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j}$. Hence, using that $a^L_j = f(z^L_j)$, we obtain that

$$\delta^L_j = \frac{\partial \mathcal{L}}{\partial a^L_j} f'(z^L_j),$$

and hereby the result.

A.17 : We have that $\delta^l_j = \frac{\partial \mathcal{L}}{\partial z^l_j}$. Using the chain rule, we obtain

$$\begin{aligned}\delta^l_j &= \sum_k \frac{\partial \mathcal{L}}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j} \\ &= \sum_k \frac{\partial z^{l+1}_k}{\partial z^l_j}\delta^{l+1}_k,\end{aligned}$$

plugging in the definition of δ. We also know that $z^{l+1}_k = \sum_j w^{l+1}_{k,j}a^l_j + b^{l+1}_k = \sum_j w^{l+1}_{k,j} f(z^l_j) + b^{l+1}_k$. We thus obtain, differentiating this expression by z^l_j,

$$\frac{\partial z^{l+1}_k}{\partial z^l_j} = w^{l+1}_{k,j} f'(z^l_j).$$

Plugging this expression in the previous sum, we find that

$$\begin{aligned}\delta^l_j &= \sum_k w^{l+1}_{k,j}\delta^{l+1}_k f'(z^l_j) \\ &= ((W^{l+1})^T\delta^{l+1})_j f'(z^l_j),\end{aligned}$$

thus the result.

A.18 : Starting again from the fact that $\delta^l_j = \frac{\partial \mathcal{L}}{\partial z^l_j}$ and using the chain rule, we obtain

$$\begin{aligned}\delta^l_j &= \sum_k \frac{\partial \mathcal{L}}{\partial b^l_k}\frac{\partial b^l_k}{\partial z^l_j} \\ &= \sum_k \frac{\partial \mathcal{L}}{\partial b^l_k}\mathbb{1}\{j = k\}\end{aligned}$$

since $b_j^l = z_j^l - \sum_k w_{j,k}^l a_k^{l-1}$, hence the result.

A.19 : Starting now from $\frac{\partial \mathcal{L}}{\partial w_{j,k}^l}$ and using the chain rule, we obtain

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{j,k}^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{j,k}^l} \\ &= \delta_j^l a_k^{l-1},\end{aligned}$$

since $z_j^l = \sum_k w_{j,k}^l a_k^{l-1} + b_j^l$. We therefore have proven all the equations. □

These four equations provide us with a way to compute the output error δ^L, and then propagate it in our network in a backwards motion to find the gradients we need to update the weights and biases. We can therefore write the learning procedure, our optimizer, as the following, combining backpropagation and stochastic gradient descent. This pseudocode is inspired by (Nielsen, 2015).

1. Input a set of i.i.d. training examples $\{(x_i, y_i)\}_{i=1...n}$.

2. For each training example i, set the activation input $a^{i,1}$, and execute the following steps:

- **a. Feedforward :** for each layer $l = 2, \ldots, L$, compute the weighted input $z^{i,l} = W^l a^{i,l-1} + b^l$ and the corresponding activation $a^{i,l} = f(z^{i,l})$.
- **b. Output error :** Compute the output error $\delta^{i,L}$, using equation **A.16**.
- **c. Backpropagation of the errors:** for each layer $l = L-1, ..., 2$, compute the errors $\delta^{i,l}$ using equation **A.17**.

3. Stochastic Gradient Descent : for each layer $l = L, ..., 2$, update the weights and biases using mini-batches B, following the (classic) SGD defined in the previous section where at each step the update rule becomes

- **a.** $W^l = W^l - \frac{\eta}{m} \sum_{i \in B} \delta^{i,l} (a^{i,l-1})^T$ (Equation **A.19**)
- **b.** $b^l = b^l - \frac{\eta}{m} \sum_{i \in B} \delta^{i,l}$ (Equation **A.18**)

Using this optimizer, we can find in the network class $\mathcal{C}_{(V,E,f)}$ the set of weights and biases minimizing the loss function $\mathcal{L}$ on the training dataset. We now have all the required elements of our optimization problem, and can therefore *learn* from the training data an efficient network solving the problem at hand.

A.4 Convolutional neural networks

Let us close this appendix with the broad principles and functioning of *convolutional neural networks* (CNN), also encountered in this thesis in Chapter 4. This section is inspired from LeCun et al. (2015) and Karpathy (2016).

Convolutional neural networks can be understood as a kind of feed-forward neural networks specifically designed to process data in the form of multiple arrays, according to LeCun et al. (2015). For instance, a color image is composed of three two-dimensional arrays — one for each RGB channel —, language and time series can be seen as one-dimensional arrays, or audio spectrograms as two-dimensional arrays — one dimension for time, and one for frequency. CNNs are state-of-the-art algorithms in the fields of computer vision and speech recognition, their popularity having constantly grown since their massive success in the ImageNet competition (Russakovsky et al., 2015).

The image classification example presented in Section A.1.3 highlights that classic multi-layer perceptrons do not scale well to computer vision tasks. For images as small as 128*128, the size of the input layer is of more than 49000. Consequently, following the heuristics introduced in A.1.3 to design the network architecture, an appropriate deep neural network would need to be deep *and* wide, resulting in longer computation times and potential overfitting due to the particularly high number of parameters. Moreover, images have inherent properties that multi-layer perceptrons do not take into account — CNNs are designed in that purpose, following inspiration from the human visual cortex.

A CNN is most commonly built using three different kind of layers: *convolutional layers*, *pooling layers* and *fully-connected layers*. The goal of here is to provide intuition about their functioning — computational details are left to references such as Goodfellow et al. (2016).

Convolutional layers are constituted of a set of learnable neurons called *filters*. Filters are spatially smaller than their input, i.e., in terms of height and width, but share the same depth. The spatial size of a convolutional filter is called its *receptive field*. Considering the example of a filter of size 3*3 used on a 128*128*3 input image, the filter successively computes

$$\sum_{l=1}^{3}\sum_{k=1}^{3}\sum_{d=1}^{3} x_{i+l,j+k,d}a_{l,k,d} + b \tag{A.20}$$

where $\{a_{l,k,d}\}_{l,k,d}$ and b are the 10 trainable parameters of the filter. This filter is applied to all (i, j) positions of the input image to form a two-dimensional *activation map* that stores the result of the convolution product between the filter and the corresponding portion of the input. A convolutional layer is usually composed of multiple filters, which activation maps are stacked together to form the output volume of the layer. Figure A.5 presents an example of convolution layer with 5 filters on a 32*32*3 image. This figure illustrates the *local connectivity* of convolutional filters: a given slice of the output activation volume is connected to a small portion of the input volume only, significantly differing from the fully-connected setting of the previously studied multi-layer perceptrons. Consequently, convolutional filters provide an efficient way to detect local spatial correlations in the inputs, such as edges and corners.

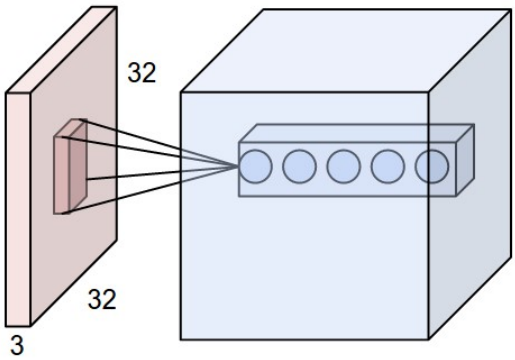


Figure A.5: An example of convolutional layer using with 5 filters, applied to a 32*32*3 image. Illustration taken from Karpathy (2016).

The output volume dimension is determined by the hyperparameters of the convolutional layer. These hyperparameters are the following:

- **Depth.** The depth of the output volume corresponds to the number of filters used in the convolution layer, and consequently controls the number of neurons connected to the same region of the input volume.

- **Stride.** The stride of the layer controls the way filters are applied over the input. Applying filters to all (i, j) positions of the input correspond to a stride of 1, but depending on filter sizes, practitioners can choose to apply filters every s pixels — that s corresponds to the stride hyperparamter.
- **Padding.** The padding controls the output volume spatial size by adding zeros to the borders of the input image. In literature, a *valid padding* corresponds to not adding zeros on the borders, and a *same padding* to adding zeros such that the output volume has the same spatial size than its input. Concretely, by applying 3*3 filters on a 128*128 image, for the 127-th and 128-th positions of the x- and y-axis there is not enough pixels to compute the convolution product — in this situation, padding the input image using two "borders" of zeros allows to retrieve 128*128 activation maps in output.

In a convolution layer, the number of weights and biases is given by $f \times n^2 \times d$, with f the number of filters, n the filter size and d the input depth. A depth slice of the output volume of a convolution layer has been obtained using the same parameters — this is called the *parameter sharing scheme*. This scheme is related to the property of *translation invariance* — as a given filter searches for a specific pattern in its input, constraining a slice depth to the same filter makes it search for the feature in the exact same way in the whole input.

Pooling layers reduce the size of their input following a non-linear *pooling* function. The most common pooling layer is the *max-pooling*. The max-pooling layer partitions each depth slice of the input into non-overlapping rectangles, and outputs the maximum of each of these regions. Pooling layers allow to gradually reduce the spatial size of the input, thereby reducing the number of parameters needed for computations and hence preventing overfitting.

Fully-connected layers correspond to the classic layers introduced in Section A.1.2. These layers are usually found at the end of computer vision networks and perform the classification or regression task at hand using the features extracted by the convolution layers of the network.

These types of layers are usually interwoven with ReLU activations, defined as $f(x) = \max(0, x)$, and all together form what we call (deep) convolutional neural networks. An example of such network is shown in Fig. A.6. In this figure, information flows bottom-up, from the picture of a Samoyed dog to its decomposition into features learnt by the network and a classification task performed by a last fully-connected layer. Among all 1000 classes of the ImageNet classification task, the right class is attributed 16% of probability.

In this thesis, we make use of one-dimensional convolution layers in Chapter 4. The computations performed with one-dimensional convolution layers follow the same idea than two-dimensional ones: for a given time series $\{x_t\}_t$ and a filter of size n, a one-dimensional convolution computes

$$\sum_{l=1}^{n} \sum_{d=1}^{m} x_{t+l,d} a_{l,d} + b, \tag{A.21}$$

with m the number of features per time-step t comprised in x_t.

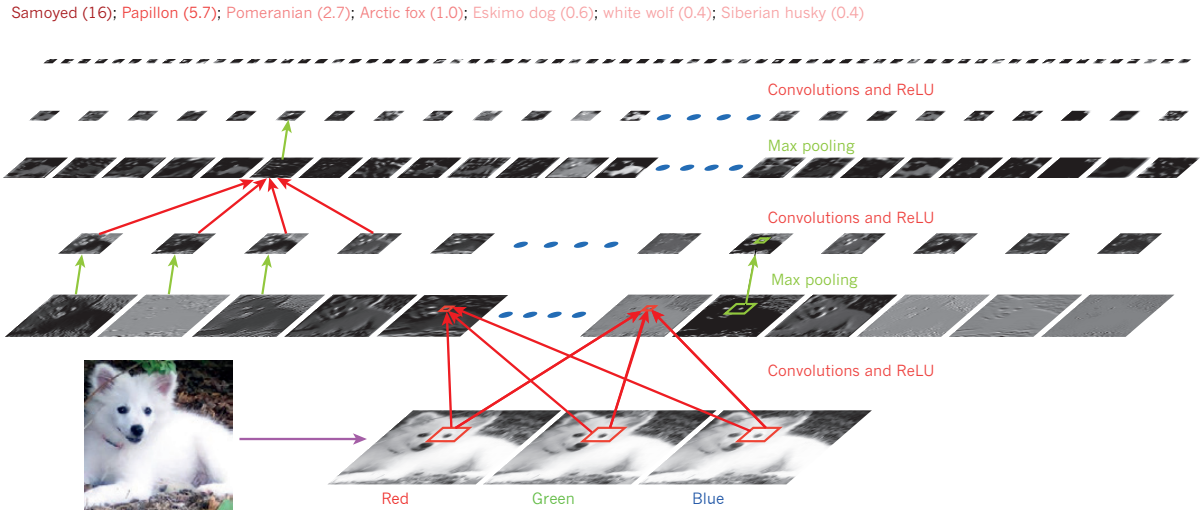


Figure A.6: An example of convolutional neural network, applied to the ImageNet 1000-classes classification task. Illustration taken from LeCun et al. (2015).

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Titre : Apprentissage statistique pour la recommandation de produits financiers

Mots clés : apprentissage statistique, finance, systèmes de recommandation, clustering supervisé, systèmes de recommandation dépendants du temps, apprentissage profond

Résumé : L'anticipation des besoins des clients est cruciale pour toute entreprise — c'est particulièrement vrai des banques d'investissement telles que BNP Paribas Corporate and Institutional Banking au vu de leur rôle dans les marchés financiers. Cette thèse s'intéresse au problème de la prédiction des intérêts futurs des clients sur les marchés financiers, et met plus particulièrement l'accent sur le développement d'algorithmes ad hoc conçus pour résoudre des problématiques spécifiques au monde financier.

Ce manuscrit se compose de cinq chapitres, répartis comme suit :

- Le chapitre 1 expose le problème de la prédiction des intérêts futurs des clients sur les marchés financiers. Le but de ce chapitre est de fournir aux lecteurs toutes les clés nécessaires à la bonne compréhension du reste de cette thèse. Ces clés sont divisées en trois parties : une mise en lumière des jeux de données à notre disposition pour la résolution du problème de prédiction des intérêts futurs et de leurs caractéristiques, une vue d'ensemble, non exhaustive, des algorithmes pouvant être utilisés pour la résolution de ce problème, et la mise au point de métriques permettant d'évaluer la performance de ces algorithmes sur nos jeux de données. Ce chapitre se clôt sur les défis que l'on peut rencontrer lors de la conception d'algorithmes permettant de résoudre le problème de la prédiction des intérêts futurs en finance, défis qui seront, en partie, résolus dans les chapitres suivants ;
- Le chapitre 2 compare une partie des algorithmes introduits dans le chapitre 1 sur un jeu de données provenant de BNP Paribas CIB, et met en avant les difficultés rencontrées pour la comparaison d'algorithmes de nature différente sur un même jeu de données, ainsi que quelques pistes permettant de surmonter ces difficultés. Ce comparatif met en pratique des algorithmes de recommandation classiques uniquement envisagés d'un point de vue théorique au chapitre précédent, et permet d'acquérir une compréhension plus fine des différentes métriques introduites au chapitre 1 au travers de l'analyse des résultats de ces algorithmes ;
- Le chapitre 3 introduit un nouvel algorithme, *Experts Network*, i.e., réseau d'experts, conçu pour résoudre le problème de l'hétérogénéité de comportement des investisseurs d'un marché donné au travers d'une architecture de réseau de neurones originale, inspirée de la recherche sur les mélanges d'experts. Dans ce chapitre, cette nouvelle méthodologie est utilisée sur trois jeux de données distincts : un jeu de données synthétique, un jeu de données en libre accès, et un jeu de données provenant de BNP Paribas CIB. Ce chapitre présente aussi en plus grand détail la genèse de l'algorithme et fournit des pistes pour l'améliorer ;
- Le chapitre 4 introduit lui aussi un nouvel algorithme, appelé *History-augmented collaborative filtering*, i.e., filtrage collaboratif augmenté par historiques, qui proposes d'augmenter les approches de factorisation matricielle classiques à l'aide des historiques d'interaction des clients et produits considérés. Ce chapitre poursuit l'étude du jeu de données étudié au chapitre 2 et étend l'algorithme introduit avec de nombreuses idées. Plus précisément, ce chapitre adapte l'algorithme de façon à permettre de résoudre le problème du *cold start*, i.e., l'incapacité d'un système de recommandation à fournir des prédictions pour de nouveaux utilisateurs, ainsi qu'un nouveau cas d'application sur lequel cette adaptation est essayée ;
- Le chapitre 5 met en lumière une collection d'idées et d'algorithmes, fructueux ou non, qui ont été essayés au cours de cette thèse. Ce chapitre se clôt sur un nouvel algorithme mariant les idées des algorithmes introduits aux chapitres 3 et 4.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY | ÉCOLE DOCTORALE Interfaces: matériaux, systèmes, usages

Title : Machine learning for financial products recommendation

Keywords : machine learning, finance, recommender systems, supervised clustering, time-aware recommender systems, deep learning

Abstract : Anticipating clients' needs is crucial to any business — this is particularly true for corporate and institutional banks such as BNP Paribas Corporate and Institutional Banking due to their role in the financial markets. This thesis addresses the problem of future interests prediction in the financial context and focuses on the development of ad hoc algorithms designed for solving specific financial challenges.

This manuscript is composed of five chapters:

- Chapter 1 introduces the problem of future interests prediction in the financial world. The goal of this chapter is to provide the reader with all the keys necessary to understand the remainder of this thesis. These keys are divided into three parts: a presentation of the datasets we have at our disposal to solve the future interests prediction problem and their characteristics, an overview of the candidate algorithms to solve this problem, and the development of metrics to monitor the performance of these algorithms on our datasets. This chapter finishes with some of the challenges that we face when designing algorithms to solve the future interests problem in finance, challenges that will be partly addressed in the following chapters;
- Chapter 2 proposes a benchmark of some of the algorithms introduced in Chapter 1 on a real-word dataset from BNP Paribas CIB, along with a development on the difficulties encountered for comparing different algorithmic approaches on a same dataset and on ways to tackle them. This benchmark puts in practice classic recommendation algorithms that were considered on a theoretical point of view in the preceding chapter, and provides further intuition on the analysis of the metrics introduced in Chapter 1;
- Chapter 3 introduces a new algorithm, called *Experts Network*, that is designed to solve the problem of behavioral heterogeneity of investors on a given financial market using a custom-built neural network architecture inspired from mixture-of-experts research. In this chapter, the introduced methodology is experimented on three datasets: a synthetic dataset, an open-source one and a real-world dataset from BNP Paribas CIB. The chapter provides further insights into the development of the methodology and ways to extend it;
- Chapter 4 also introduces a new algorithm, called *History-augmented Collaborative Filtering*, that proposes to augment classic matrix factorization approaches with the information of users and items' interaction histories. This chapter provides further experiments on the dataset used in Chapter 2, and extends the presented methodology with various ideas. Notably, this chapter exposes an adaptation of the methodology to solve the cold-start problem and applies it to a new dataset;
- Chapter 5 brings to light a collection of ideas and algorithms, successful or not, that were experimented during the development of this thesis. This chapter finishes on a new algorithm that blends the methodologies introduced in Chapters 3 and 4.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France