Kolesnikova, A. et al.

Working Paper

Can Deep Learning Predict Risky Retail Investors? A Case Study in Financial Risk Behavior Forecasting

IRTG 1792 Discussion Paper, No. 2019-023

Provided in Cooperation with:
Humboldt University Berlin, International Research Training Group 1792 "High Dimensional Nonstationary Time Series"

Suggested Citation: Kolesnikova, A. et al. (2019) : Can Deep Learning Predict Risky Retail Investors? A Case Study in Financial Risk Behavior Forecasting, IRTG 1792 Discussion Paper, No. 2019-023, Humboldt-Universität zu Berlin, International Research Training Group 1792 "High Dimensional Nonstationary Time Series", Berlin

This Version is available at:
https://hdl.handle.net/10419/230799

IRTG 1792 Discussion Paper 2019-023

Can Deep Learning Predict Risky Retail Investors? A Case Study in Financial Risk Behavior Forecasting

A. Kolesnikova [*]
Y. Yang [*2]
S. Lessmann [*]
T. Ma [*3]
M.-C. Sung [*3]
J.E.V. Johnson [*3]

[*] Humboldt-Universität zu Berlin, Germany
[*2] University College London, United Kingdom
[*3] University of Southampton, United Kingdom

This research was supported by the Deutsche Forschungsgesellschaft through the International Research Training Group 1792 "High Dimensional Nonstationary Time Series".

http://irtg1792.hu-berlin.de
ISSN 2568-5619

Can Deep Learning Predict Risky Retail Investors? A Case Study in Financial Risk Behavior Forecasting

A. Kolesnikova[a], Y. Yang[c], S. Lessmann[a], T. Ma[b], M.-C. Sung[b,*], J.E.V. Johnson[b]

[a]*School of Business and Economics, Humboldt-University of Berlin*
[b]*Southampton Business School, University of Southampton*
[c]*Department of Computer Science, University College London*

Abstract

The paper examines the potential of deep learning to produce decision support models from structured, tabular data. Considering the context of financial risk management, we develop a deep learning model for predicting whether individual spread traders are likely to secure profits from future trades. This embodies typical modeling challenges faced in risk and behavior forecasting. Conventional machine learning requires data that is representative of the feature-target relationship and relies on the often costly development, maintenance, and revision of handcrafted features. Consequently, modeling highly variable, heterogeneous patterns such as the behavior of traders is challenging. Deep learning promises a remedy. Learning hierarchical distributed representations of the raw data in an automatic manner (e.g. risk taking behavior), it uncovers generative features that determine the target (e.g., trader's profitability), avoids manual feature engineering, and is more robust toward change (e.g. dynamic market conditions). The results of employing a deep network for operational risk forecasting confirm the feature learning capability of deep learning, provide guidance on designing a suitable network architecture and demonstrate the superiority of deep learning over machine learning and rule-based benchmarks.

Keywords: risk management, retail finance, forecasting, deep learning

1. Introduction

The paper applies recently developed deep learning (DL) methods to forecast the behavior of retail investors in the spread-trading market. Market makers depend on accurate forecasts of traders' future success to manage financial risks. Through developing a DL-based forecasting model and confirming the profitability of a model-based hedging strategy, we provide evidence that characteristic features of DL generalize to the structured data sets commonly employed in retail finance and decision support.

DL methods operate in a stage-wise manner. For example, in a deep neural network (DNN), each layer receives an input from previous layers, learns a high-level representation of the input, and passes the representation (i.e., output) to a subsequent layer.

A popular example of the stage-wise approach is that of face recognition. To detect faces in an image, the first layers in a DNN learn low-level concepts such as lines and borders from raw pixels.

*Corresponding author
Email addresses: kolesnikova.alisa.n@gmail.com (A. Kolesnikova), yaodong.yang@outlook.com (Y. Yang), stefan.lessmann@hu-berlin.de (S. Lessmann), tiejun.ma@soton.ac.uk (T. Ma), M.SUNG@soton.ac.uk (M.-C. Sung), J.E.Johnson@soton.ac.uk (J.E.V. Johnson)

Deeper layers generalize lower layer outputs to more complex concepts such as squares and triangles that eventually form a face [1]. An analogous example in decision support could be corporate credit risk modeling. Bankruptcy prediction models estimate default probabilities on the basis of ratios of accounting variables (e.g., total assets/total liabilities) [2]. In a DL framework, such ratios represent a low level representation. Using the balance sheet as (raw) input, lower layers in a DNN can relate a variety of statement variables and calculate informative ratios in a data-driven manner. A higher level representation of the data could then include the trend in a financial ratio or inter-dependencies between ratio variables. The specific representation is calculated autonomously.

A hierarchical composition of representations of different complexities enable a DNN to learn abstract concepts such as that of a delinquent borrower. Representation learning also enhances the ability of a model to extract patterns that are not well represented in the training data, which is a problem for other data-driven models [3]. DL methods have delivered excellent results in applications such as computer vision, language processing, and many others [4]. This success has established the effectiveness of DL-based feature learning in applications that rely on unstructured data [5].

Applications of conventional - 'shallow' - machine learning (ML) are manifold. Marketing decision models, for example, support all stages of the customer life cycle including response modeling, cross-/up-selling [6], and churn prediction [7]. Financial institutions use ML to anticipate financial market developments [8], predict the solvency of corporate borrowers [9], or inform credit approval decisions [10]. Such applications rely on structured data such as past customer transactions, price developments, or loan repayments. It is not obvious that the success of DL in text mining, image recognition and other tasks that involve unstructured data generalizes to decision support applications where structured data prevails. Therefore, the objectives of the paper are to examine the effectiveness of DL in decision support, to test whether its feature learning ability generalizes to the structured data sets typically encountered in this field, and to offer guidance on how to setup a DL-based decision support model.

We pursue our objectives in a financial risk management context. Using data from the spread-trading market, we predict the profitability of individual traders. The modeling goal is to identify traders that pose a high risk to the market maker, and recommend a hedging policy that maximizes the marker maker's profits. Beyond the utility of such a policy for a spread-trading company, the trader risk prediction task represents challenges that are commonly encountered in ML-based decision support.

A first challenge is class imbalance. Adverse events such as borrower default or customer churn represent minorities in their populations, and this impedes ML [7]. A second challenge called concept drift arises in dynamic environments. ML models infer (*learn*) a functional relationship between subject characteristics (e.g., previous trades of a client) and a prediction target (e.g., trader profitability) from past data. Changes in the environment render this relationship more volatile and harder to infer. The curse-of-dimensionality is another modeling challenge. Corporate data warehouses provide a huge amount of information about modeling subjects (e.g., traders) and it is difficult to learn generalizable patterns in the presence of a large number of features [11]. Finally, the success of ML depends on the availability of informative features. Feature engineering is carried out manually by domain experts. Given high labor costs, a shortage of skilled analysts and the need to revise hand-crafted features in response to external

changes (e.g., in trader behavior), manual feature engineering decreases the efficiency of ML and becomes an impediment to ML-based decision support.

The common denominator of the modeling challenges is that they reduce the representativeness of the training data. Being a data-driven approach, ML suffers from reduced representativeness, which suggests that the challenges diminish the effectiveness and efficiency of data-driven ML models. Considering our application setting as an example, the representation learning ability of DL could help to identify a more generic representation of the trading profile of high-risk traders than that embodied in hand-crafted features. More generality in the inferred feature-target-relationship would offer higher robustness toward external variations in trading behavior; for example, variations introduced by changes in business cycle, market conditions, company operations, etc. Replacing the need for costly manual feature engineering would also raise the efficiency of model-based decision support.

Examining the degree to which DL remedies common modeling challenges in decision support, the paper makes the following contributions. First, it is one of the first studies to examine the effectiveness of DL in conjunction with structured, individual-level behavioral customer data. Predicting individual trader's risk taking behavior, we focus on retail finance, which is a pivotal application area for operations research [12] that, to our knowledge, no previous DL study has considered. Empirical results provide evidence that DL predicts substantially more accurately than ML methods. Second, we demonstrate the ability of DL to learn informative features from operational data in an automatic manner. Prior research has confirmed this ability for unstructured data [1]. We expand previous results to transactional and behavioral customer data. This finding is managerially meaningful because many enterprises employ structured data for decision support. Third, the paper contributes to financial risk taking forecasting practice in that it proposes a DNN-based approach to effectively manage risk and inform hedging decisions in a speculative financial market.

The DL methodology that we employ in the paper is not new. However, DL and its constituent concepts such as distributed representations are rarely explained in the language of business functions. Business users can benefit from an understanding of DL concepts to enable them to engage with data scientists and consultants on an informed basis. A better understanding might also lead to more appreciation of formal, mathematical models and help to overcome organizational inertia, which is a well-known impediment to fact-based decision support [13, 14]. Against this background, a final contribution of the paper is that it increases awareness of DL in business through evidencing its potential and providing a concrete recipe for how to set up, train, and implement a DNN-based decision support approach. To achieve this, we elaborate on the methodological underpinnings of DL and the decision model we devise for trader risk classification.

2. Related Work

The literature on DL is growing at high pace. A comprehensive overview of the state-of-the-art is available in recent surveys [1, 4, 5]. We focus the review of related literature to DL applications in finance. Table 1 analyzes corresponding studies along different dimensions related to the forecasting setting, underlying data, and neural network topology.

To clarify the selection of papers, we acknowledge that DL has other applications in finance beyond forecasting including index tracking [15] or modeling state dynamics in limit-order-book data [16]. Furthermore, DL has been applied to generate financial forecasts from textual data [17]. Table 1 does not include such studies as they do not concentrate on prediction or consider a different source of data.

Finally, one may argue that a recurrent neural network (RNN) is a DNN by definition, because recurrent cells exhibit temporal depth. With the rise of DL, gated RNNs such as LSTM (long short-term memory) gained popularity and are often characterized as DNNs [18]. This is not necessarily true for their predecessors, some of which have been used in finance [19]. Table 1 analyzes studies that used contemporary gated RNNs and omits those that use earlier types of RNNs.

Table 1 shows that the majority (roughly 60%) of previous studies forecast developments in financial markets, such as price movements [20], volatility [21] or market crashes [22]. Applications in risk analytics such as financial distress prediction [23] or credit scoring [24] are also popular. Considering the objectives of forecasting, columns two and three reveal that previous studies have not considered forecasting human behavior, which is the focus of this paper.

The type of input data represents a second difference between most previous studies and this paper. DNNs that forecast financial market prices typically receive lagged prices as inputs. For example, [20] and [18] use minute- and day-level price returns as inputs. By contrast, the risk modeling task we face consists of a dynamic regression problem with different types of predictor variables (see Section 5.1). The *feature* columns in Table 1 show that few prior studies mix numerical and discrete input variables.

A core feature of DNNs is the ability to automatically extract predictive features from the input data [25]. One objective of this paper is to confirm the feature learning capability in a risk management context. A substantial difference in the type of input data has implications for feature learning. It is not obvious that results observed in a time series setting generalize to a dynamic regression setting with diverse input variables. With respect to risk management, we observe from the column *profit simulation* in Table 1 that most previous work has not examined the economic implications of a DL-based risk management approach; [24] being an exception.

In addition to the application setting and input data, a third difference between most previous work and this study concerns the architecture of the DNN. Table 1 sketches the topology of previous networks in its three rightmost columns. Given our focus on forecasting studies, every network includes a supervised learning mechanism, meaning that weights in the network are trained through minimizing the empirical loss on the training data set [26]. This is typically implemented by means of a fully-connected output layer. This layer requires only one unit with a linear or softmax activation function to solve regression and classification problems, respectively. Table 1 shows that purely supervised learning networks prevail in previous work. From this observation, we conclude that more research into networks with supervised and unsupervised layers is desirable.

In total, nine studies consider unsupervised pre-training. The majority implement pre-training using a deep belief network. Long before pre-training was popularized, a seminal study proposed self-organizing maps for unsupervised time series pattern extraction [27]. Stacked denoising auto-encoders (SdA), the approach we use for feature learning, have received little attention. Evidence of their effectiveness in risk

analytics is originally provided in this paper.

In summary, the contribution of our work to literature emerges through a combination of characteristics concerning the forecasting setting, the data employed, and the way in which we devise and assess the DL-based forecasting model through using state-of-the-art approaches for network training and unsupervised pre-training and evaluation of the profitability of model-based hedging decisions.

The study closest related to our work is [24]. The authors estimate a DNN from a data set of over 3.5 billion loan-month observations with 272 variables relating to loan characteristics and local economic factors to support portfolio management. To that end, [24] model the transition probabilities of individual loans between states ranging from current over different delayed payment states to delinquency or foreclosure. Our study differs from [24] in terms of the application setting and DL methodology.

The DL models of [24] consists of feed-forward networks of up to 7 layers (and ensembles thereof). Deep feed-forward networks are a generalization of the three-layer networks widely used in previous work ([28]). The DNN architecture proposed here is different. It uses multiple layers of different types of units and relies on unsupervised pre-training to extract predictive features. Pre-training elements provide distinctive advantages and have been found effective in financial applications [15]. Consequently, we further advance the methodology of [24].

Concerning the application, the mortgage risk modeling setting of [24] as well as conventional credit scoring settings [12] differs substantially from trader risk prediction. A credit product can be considered a put option with the lender having the right to grant credit, but no obligation to do so. Credits may also be secured by collateral and, most importantly, it is possible to hedge risks while still earning money from commissions. However, we consider a spread-trading context where the market maker is *obliged* to accept orders from its clients. These orders are similar to futures contracts with an arbitrary strike date. In addition, unlike in the money lending business where a customer will be given a credit limit, in the spread trading market, informed traders or insiders can make unlimited profits from the market marker. Consequently, the market maker faces the risk of adverse selection. At the same time, the economics of the spread-trading market require the market maker to hedge risks very selectively (because hedging quickly reduces revenues to zero). Thus, our forecasting task is to identify those traders who pose a substantial risk to the market maker.

Table 1: Summary of Related Work on DL in Finance

	Area[1]	Subject[2]	Target	Time Series	Time Window	Observations	Features[3]		Horizon	Study Design	Data part.[4]	Profit sim.	Supervised Layers[5]	Unsup. Pretrain.[5]	Architecture[6]
[27]	MM	Exr	Direction	5	9/1973 - 5/1987	3,645	2	con	day	rolling window	100 / 30		RNN	SOM	x-7-2-o
[29]		Exr	Return	3	1976 - 2004	$< 1,000$	5	con	week	temporal split	0.7 / 0.3		FC	DBM	x-20-20-20-o
[21]		Ind	Volatility	1	10/2004 - 7/2015	2,682	25 (t3)	con	day	temporal split	0.7 / 0.3		LSTM		x-1-o
[30]		Fut	Direction	43	3/1991 - 9/2014	50	9895	con	5 min	rolling window	25000 / 12500	yes	FC		x-1000-100-100-100-o
[20]		Fut	Return	4	1/2014 - 9/2015	100	150	con	minute	rolling window	15000 / 5000	yes	RLRNN		x-128-128-128-20-o
[31]		Ind	Return	6	7/2008 - 9/2016	2,079	19 (t4)	con	day	rolling window	2y / 1q / 1q[7]	yes	LSTM	SdA	x-10-10-10-10-10-1-1-1-1-1-o
[32]		Sto	Better S&P500	∼500	1/1992 - 10/2015	380	31	con	day	rolling window	750 / 250	yes	FC		x-31-10-5-o
[33]		Co	WTI crude oil spot price	1	1/1986 - 5/2016	365	200	con	month	temporal split	0.80 / 0.20		FC	SdA	x-100-10-o
[18]		Sto	Better S&P500	∼500	1/1992 - 10/2015	380	1 (t240)	con	day	rolling window	750 / 250	yes	LSTM		x-25-o
[34]		Ind	Return	2	1/2000 - 7/2017	4,3	6 (t20)	con	day	temporal split	0.45 / 0.55	yes	LSTM+ LSTM+FC		x-(x1-5-3 — x2-4-2)-2-o
[22]		Ind	Crash	2	1/1996 - 12/2017	5,4	131	con	1 | 5 days	temporal split	0.66 / 0.33		FC		x-64-32-8-2-o
[35]		Ind	Volatility	1	1/2001 - 1/2017	3,963	6 (t22)	con	day	temporal split	0.68 / 0.32		LSTM+FC		x-10-4-2-5-o
[36]		Sto	Better S&P	300	1/1993 - 5/2015	6300	≤ 592	con	day	rolling window	504 / 126	yes	FC	DBN	x-148-74-o
[37]	RA	Ent	Insolvency	N.A.	2002 - 2006	1,2	30	con	Year	random split	800 / 400		FC	DBN	x-500-500-1000-o
[38]		Ent	Insolvency	N.A.	2001 - 2011	$\sim 83,000$	180	con	Year	rolling window	04.01.2001		FC	DBN	x-1000-1000-1000-o
[39]		Ent	Firm perf.	22	2000 - 2015	286	15	con	Year	temporal split	10. Mrz		FC	DBN	x-200-200-200-200-o
[40]		Ent	Rating	N.A.	1/2016 - 12/2016	661	11	mix	N.A.	cross-val.	10 fold		FC	DBN	Not specified
[24]		Mor	Default	N.A.	1/1995 - 6/2014	$3.5 * 10^9$	272	mix	month	temporal split	214 / 19	yes	FC		x-200-140-140-140-140-o
[23]		Ent	Insolvency	286	2016 - 2017	117,019	181	con	N.A.	random split	0.6/0.2/ 0.2		FC		x-50-50-1
[41]	FD	CC	Fraud	2	5/2015 - 5/2015	$1.65 * 10^7$	30 (t10)	mix	N.A.	temporal split	0.43/ 0.08/ 0.49		LSTM		x-100-100-100-o

[1] MM: financial market modeling, RA: risk analytics, FD: fraud detection

[2] Exr: exchange rate of a pair of currencies, Ind: financial market index, Fut: future contract, Sto: individual stock, Co: commodity, Ent: enterprise, Mor: mortgage, CC: credit card.

[3] Number of input features and their type using abbreviations con (continuous feature) and mix (continuous and categorical features). For studies that use LSTM networks we also report the length of a time-lagged input sequence using the notation (tl) where t means time and l is the number of lags. For example, [3] consider 25 features and feed the last three observations (days) of each feature into their LSTM.

[4] Partitioning of the data for model training, validation, and testing. Fractional numbers represent percentages with respect to the size of the data set while values greater zero depict absolute numbers of observations. The notation is training set size / validation set size / test set size. Not all studies use separate validation data. Then, the two values given in the column represent training set size / test set size.

[5] RNN: recurrent neural network, FC: fully-connected layer, LSTM: Long-short-term-memory, RLRNN: reinforcement learning RNN, SOM: self-organizing-map, SdA: stacked denoising auto-encoders, DBM: deep belief network.

[6] Symbols x and o represent the multivariate input and scalar output of the network. Numbers give the size of hidden layers. For studies that use pre-training, hidden layer sizes refer to units of the unsupervised layers (e.g., DBM, SOM, or SdA). Exceptions and special cases for complex topologies exists and we elaborate on these in the discussion of the table.

[7] The notation is slightly different from other studies. The authors use a rolling window evaluation to train, validate, and test their models using daily prices from two years, one quarter, and one quarter, respectively.

3. Risk Taking and Behavior Forecasting in the Spread-Trading Market

Spread trading is becoming increasingly significant. Forty per cent of the 1.2 trillion traded annually on the London Stock Exchange is equity derivative related and 25 per cent of this (120 billion) relates to spread trading [42].

Spread trading often refers to pairs trading of stocks or to trading spreads in the futures market [43]. However, our study focuses on the form of spread trading which relates to retail *contracts for difference (CFD)*. In this market, a retail investor and a market maker enter a contract related to a specified financial instrument (e.g. a share, commodity or an index) and at the end of the contract they exchange the difference between the closing and opening price of that financial instrument. Consequently, investors trade on the direction and magnitude of movements of a financial instrument. For example, a client might place a long order on the S&P500 with stake size \$10 per point. If the S&P500 rises by a particular *increment*, the client makes a profit of $\$10 * increment$; otherwise s/he loses this amount. The market maker continuously quotes bid and ask prices for marketable instruments. Unlike brokers, who help clients to trade with other investors, market makers buy or sell financial instruments from their own inventory. Provided clients meet a margin requirement, they can open and close positions at any time. The market maker is obliged to accept these orders and faces the risk of adverse selection.

Forecasting which traders pose the most risk (i.e. those who are likely to make the most profit) and deciding which risks to hedge into the main market is crucial for market makers. Informed traders might take advantage of inside information and leave the market maker with positions against a market rally. In theory, the potential loss of the market maker from one trade is unbounded. For example, IG Group, the largest retail financial services provider in UK, recently lost 30 million GBP due to deficient risk control and inflexible hedging strategies. As a result of similar problems, FXCM, the largest market maker on the global spot FX market, went bankrupt[1].

The spread between quoted bid and ask prices is the main source of revenue of the market maker. For liquid markets, such as those for the S&P500 or for the USD/GBP or EUR/GDP, the spread is greater in the spread trading market than in the underlying market. However, for less liquid financial instruments (e.g. the DAX or FTSE100 index) the spread is less than that offered in the underlying market. This later situation is often faced by spread trading firms when they need to place large volume transactions into the underlying market for less liquid financial instruments. If the market maker hedges a trade, they lose the potential profit from the spread whether or not the hedging was necessary. The market maker also faces transaction costs to hedge a position, including commission and the higher spreads in some markets when they seek to hedge large volumes. Therefore, designing a predictive classification model that distinguishes A-book clients (i.e. those who pose most risk to the market maker) from B-book clients (those who pose less risk) is vital. The market maker will hedge positions from A-book clients to protect against losses and will take the risk of the positions from B-book clients to increase profits. Typically, 90% of the total revenues come from B-book clients [44].

[1] See https://www.forbes.com/sites/steveschaefer/2015/01/16/swiss-bank-stunner-claims-victims-currency-broker-fxcm-bludgeoned/#7e94f5466de0

The decision task under study is whether to hedge an incoming trade. This task translates into a classification problem, which we address through developing a DNN to predict high risk (A-book) traders. Provided the DNN learns patterns from observed trading behavior that facilitate an accurate prediction of a trader's future successes, it can assist the market maker through recommending hedging decisions and enhancing risk management in daily operations. Figure 1 illustrates the DNN-enabled hedging strategy.

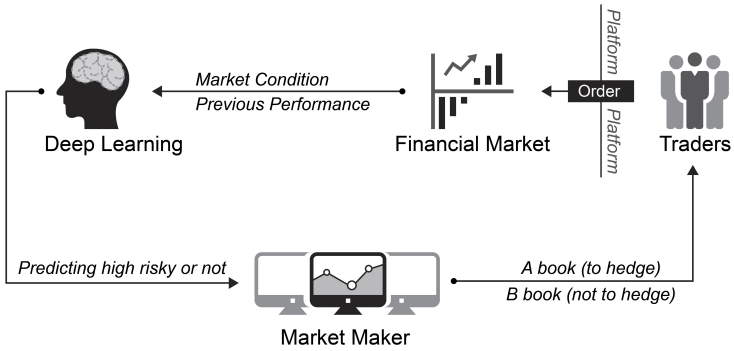

Figure 1: Workflow of how hedge strategy works for market makers

3.1. Trader Classification and Hedging Strategy

The definition of an A-book client is subjective and depends on the business strategy of the market maker. The company which provided the data prefers to remain anonymous (we refer to them hereafter as STX), but is a large player in the UK spread-trading market. From interviews with their front-desk dealers, who engage in day-to-day risk management, we found that STX at the time of the study, defined a client i to be a high risk trader if s/he secured a return greater than 5% from her previous 20 trades. The strategy of STX was to hedge the trades of these clients.

The deployed hedging strategy is dynamic, since STX determines the status of a client (A- or B-book) from the performance of their previous 20 trades. Therefore, client status can change due to a single trade. Accordingly, we frequently observe a situation where STX takes the risk of trade j of client i while hedging against trade $j+k$ of client i. In a speculative market, the overall return of a set of past trades can give misleading guidance to the future profitability of a trader. For example, a skilled trader, who follows a consistent strategy, shows high trading discipline, routinely uses and updates stop-loss limits, etc., can regularly lose money due to the randomness of the environment. Similarly, a poor trader, who violates all the above principles, occasionally makes a profit. This suggests that a trader's past performance is not necessarily a reliable signal of their true ability. Consequently, the goal of developing a client classification model is to generate a superior signal for hedging decisions by accounting for all other characteristics available in the data.

We develop a DNN to learn the latent nature of a trader from past trading data. The target concept, trader ability, is highly variable, corrupted by noise, and difficult to accommodate in a pre-defined, static set of trader characteristics. Therefore, it will be important for the DNN to distill, from transactional data, high-level distributed representations of the target concept, which capture the underlying generative

factors that explain variations in trading behavior. In this regard, success in trader classification will evidence the ability of DL to automatically extract informative features.

3.2. Trader Behavior Prediction and Decision Support

It is not obvious that representation learning is effective in risk management. Applications such as, credit scoring, churn prediction or trader classification involve the forecasting of human behavior. One would expect the maximal attainable accuracy in a behavior forecasting model to be less than in, e.g., face detection. For example, the prediction target is less clearly defined (e.g., STX used a 5% threshold but this is subjective) and the feature-target relationship is typically weak. Our trader behavior forecasting study aims to clarify the potential of representation learning and DL in decision support.

We argue that the prediction task is representative of a range of modeling challenges in decision support because it exhibits several characteristics that often occur in corporate applications of data-driven prediction models. More specifically, we face challenges that diminish the representativeness of the training data. First, in response to previous gains and losses and changes in the macro-environment, the behavior of individual traders can be variable, erratic and dynamic. Second, detailed, time-ordered information about individual traders, asset prices and their underlying fundamentals and broader indicators of market sentiment (e.g., economic growth) are readily available, which leads to high dimensionality. Third, the specific way in which variables relate to each other and govern traders' profits is complex, nonlinear, and likely to evolve over time. Automatic feature extraction, if successful, is a promising way to cope with these challenges. Fourth, the spread trading setting displays class imbalance in that only a few traders succeed in securing systematic positive returns above 5%, while the vast majority of clients lose money. Last, effective risk management requires accurate predictions at the level of an individual trader. Accuracy is a general requirement in predictive decision support.

4. Methodology

In view of the scarcity of DL applications in the risk analytics literature, we revisit principles of DL and detail how we configure the DNN to classify spread traders into A- or B-book clients. The online Appendix elaborates on these concepts and DNN training.

4.1. Principles of Deep Learning

DL aims at learning multiple levels of representations from data, where higher levels represent more abstract concepts. A deep architecture with multiple layers of abstraction and its ability to learn distributed representations provides several advantages over conventional *shallow* ML methods and we discuss these below.

The deep architecture. ML methods learn a functional relationship between variables, which characterize the relationship between an observation and a prediction target. High variability of this function complicates the ML approach and may lead to poor generalization. Sources of high variability include external shocks to the environment in which a decision model operates. Learning theory suggests that

to represent a functional relationship, a learning machine with depth k needs exponentially more computational units than a machine with depth $k+1$ [25]. The depth of commonly-used machine learning methods is as follows [3]: linear and logistics regression (depth 1); boosting and stacking ensembles: depth of base learner (depth +1, one extra layer for combining the votes from base learners); decision trees, one-hidden-layer artificial neural networks (ANNs), support vector machines (depth 2); the visual system in the human brain (depth 5-10, [45]).

The concept of depth explains a large number of empirical findings related to, for example, ANNs or support vector machines outperforming simple regression models or ensemble classifiers outperforming individual learners [46]. Increased depth allows these methods to implicitly learn an extra level of representation from data [3]. Additional levels facilitate generalization to new combinations of the features, which are less represented in the training data. Enlarged capacity also allows the learning machine to capture more variations in the target function, which discriminates classes accurately. Furthermore, the number of computational units a model can afford is severely restricted by the number of training examples. As a result, when there are variations of interest in the target function, shallow architectures need extreme complexity (large amounts of computational units) to fit the function. Consequently, they need exponentially more training examples than a model with greater depth [3].

Distributed Representations. DL methods learn distributed representations from data. An example of a distributed representation is principal component analysis (PCA). PCA re-orients a data set in the direction of the eigenvectors, which are ordered according to their contribution to explained variation. This is a distributed representation where the raw variables collaborate to generate a principle component. In predictive ML, principle components can replace the original variables. The functional relationship to learn is then that between the target variable and the principle components. This can simplify the learning task, increase predictive accuracy, and facilitate feature reduction [47]. However, ML methods learn local, non-distributed representations. Using the raw variables in a data set, they partition the input space into mutually exclusive regions. For example, support vector machines infer a decision boundary from the local training examples of adjacent classes that are closest to each other.

The goal of ML is to classify novel examples, which are not part of the training set. However, the training data may lack representativeness (e.g., because of a change in the environment). An advantage of distributed representations is that they are better able to accommodate new observations that the training data does not represent well. Consider our trader classification problem as an example: Traders exhibit different trading styles; they use different strategies, follow different stop-loss rules, etc. Assume traders are split into 5 different clusters, with traders in the same cluster sharing a trading style. Using a non-distributed representation, we need 5 different features to exclusively represent each cluster, $0 = 00000, 1 = 01000, ..., 4 = 00001$. A distributed representation requires only $\lceil \log_2 5 \rceil = 3$ features to model the clusters (as a binary code), $0 = 000, 1 = 001, ..., 4 = 100$. Using three distributed features, this representation can also accommodate a new type of trader (i.e., using trading strategies that have not been employed in the training sample): $5 = 101$. This exemplifies an advantage of distributed representations, namely that the number of patterns that the representation can distinguish grows exponentially

with the number of features. However, for non-distributed representations, this number grows, at best, linearly.

4.2. Building the Deep Neural Network

DL methods consist of multiple components with levels of non-linear transformations. A typical instance is a neural network with several hidden layers [32, 24]. Training a DNN requires solving a non-convex optimization problem, which is difficult because of the vanishing gradient problem [48]. Gradient vanishing prohibits propagating error information from the upper layer back to lower layers in the network, so that connection weights in lower layers cannot be adapted [49]. As a result, the optimization will often terminate in poor local minima. Remedies to this problem include unsupervised pre-training, parametric Rectifier Unit (ReLu), Xavier initialization, dropout, and batch normalization. We take advantage of these techniques to develop a trader classification DNN for risk management. Below, we introduce pre-training and dropout. Interested readers find a similar description of the other concepts in the online Appendix.

4.2.1. Unsupervised Pre-Training

The goal of pre-training is to find invariant, generative factors (i.e., distributed representations), which explain variations in the data and amplify those variations that are important for subsequent discrimination. Through a sequence of non-linear transformations, pre-training creates layers of inherent feature detectors without requiring data labels. This facilitates a local learning of connection weights. Avoiding a propagation of error information through multiple layers of the network, pre-training helps to overcome the vanishing gradient problem. Stacking multiple layers of progressively more sophisticated feature detectors, the DNN can be initialized to sensible starting values. After discovering a structural relationship in the data, one can then add a supervised learning technique (e.g., logistic regression) on top of the pre-trained network and tune parameters using back-propagation. Unsupervised pre-training, where the use of the target label is postponed until the fine-tuning stage, is especially useful in management decision support where class imbalance is a common problem [10].

Two classical implementations of pre-training are deep belief networks (DBN), which are pre-trained by restricted Boltzmann machine [50], and stacked denoising autoencoders (SdA), which are pre-trained by the autoencoder [48]. Both strategies minimize an approximation of the log-likelihood of a generative model and, accordingly, typically show similar performance [51, 52]. This, together with the fact that deep belief networks have already received some attention in the risk analytics literature (see Table 1), led us to use the framework of the stacked denoising autoencoder [52].

Denoising Autoencoder. The denoising autoencoder (dA) learns a distributed representation (namely the "code") from input samples. Suppose we have N samples and each sample has p features. Receiving an input $\mathbf{x} \in \mathbb{R}^p$, the learning process of a dA includes four steps:

Step 1: *Corruption.* The dA first corrupts the input $\mathbf{x}$. By sampling from the Binomial distribution $(n = N, p = p_q)$, (where the corruption rate q_p is a hyper parameter that needs tuning outside the

model), the dA randomly corrupts a subset of the observed samples and deliberately introduces noise. For example, if the input features a binary, corruption corresponds to flipping bits.

Step 2: *Encoder.* The dA deterministically maps the corrupted input $\widetilde{\mathbf{x}}$ into a higher-level representation (the code) $\mathbf{y} \in \mathbb{R}^k$. The encoding process is conducted via an ordinary one-hidden-layer neural network (the number of hidden units k is a hyper parameter that needs tuning outside the model). With weight matrix W, biases b, and encoding function $h(\cdot)$, e.g., *sigmoid* function, $\mathbf{y}$ is given as:

$$\mathbf{y} = h(W \cdot \widetilde{\mathbf{x}} + b) \tag{1}$$

Step 3: *Decoder.* The code $\mathbf{y}$ is mapped back by a decoder into the reconstruction $\mathbf{z}$ that has the same shape as the input $\mathbf{x}$. Given the code $\mathbf{y}$, $\mathbf{z}$ should be regarded as a prediction of $\mathbf{x}$. Such reconstruction represents a *denoising* process; it tries to reconstruct the input from a noisy (corrupted) version of it. Similar to the encoder, the decoder has the weight matrix $\widetilde{W}$, biases $\widetilde{b}$, and a decoding function $g(\cdot)$. The reconstruction $\mathbf{z}$ is:

$$\mathbf{z} = g(\widetilde{W} \cdot \mathbf{y} + \widetilde{b}) \tag{2}$$

Step 4: *Training.* Optimizing the parameters of dA (W, b, $\widetilde{W}$, $\widetilde{b}$) involves minimizing the reconstruction error $L_{(\mathbf{x},\mathbf{z})}$; achieved by letting the code $\mathbf{y}$ learn a distributed representation that captures the main factors of variation in $\mathbf{x}$. Theoretically, if we use the mean squared error ($L_{H(\mathbf{x},\mathbf{z})} = ||\mathbf{x} - \mathbf{z}||^2$) as the cost function and linear functions as both encoder $h(\cdot)$ and decoder functions $g(\cdot)$, the dA is equivalent to PCA; the k hidden units in code $\mathbf{y}$ represent the first k principal components of the data. The choice of cost function depends on the distributional assumptions of input $\mathbf{x}$. In this paper, we measure the reconstruction error by the cross entropy loss function, as most of our features are probabilities $\mathbf{x} \in [0,1]^p$. In addition, we incorporate an L2 penalty (also called weight decay [53]). This is equivalent to assuming a Gaussian prior over the weights and a common approach to encourage sparsity among weights and improve generalization. The regularization parameter λ captures the trade-off between reconstruction error and model complexity. The parameter needs tuning outside the model and offers a way to protect against overfitting. Higher values of λ penalize model complexity more heavily and, ceteris paribus, reduce the risk of overfitting. The final cost function is:

$$L(\mathbf{x},\mathbf{z}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{p}[x_{ik}\log z_{ik} + (1-x_{ik})\log(1-z_{ik})] + \lambda \left\|W\right\|_2 \tag{3}$$

Several solvers (e.g., stochastic gradient descent) are available to carry out the optimization.

$$\arg\min_{w,\widetilde{w},b,\widetilde{b}} L(\mathbf{x},\mathbf{z} \mid \Theta) \tag{4}$$

Step 5: *Stacking.* Once a dA has been trained, one can stack another dA on top. Layers are organized in a feed-forward manner. The second dA takes the encoded output of the previous dA (the code $\mathbf{y}$) as its new inputs $\mathbf{x}$. Each layer of dA is trained locally, finding its own optimal weights regardless of the other layers. Iteratively, a number of dAs can be stacked upon each other to construct a stacked denoising autoencoder (SdA). The encoding weights of each dA can then be treated as initializations for the network in the next step. Figure 2 illustrates the working flow of dA.

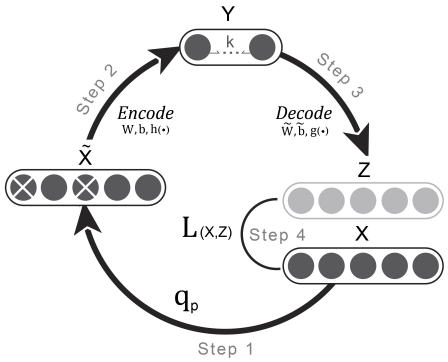

Figure 2: Architecture of denoising auto-encoder.

4.2.2. Supervised fine-tuning

The SdA can be trained in a feed-forward, layer-wise manner. To employ the network for prediction, network training continues with supervised fine-tuning that teaches the DNN which types of trading behaviors (in the form of distributed representations) identify A-book clients. To that end, we add a *softmax* regression on top of the SdA. This way, we solve a supervised learning problem using the distributed representation of the raw input as features (which the SdA output embodies), and a binary indicator variable as target, which indicates whether a trade should be hedged. Formally, with parameter weight W and bias b, the probability that a trade $\mathbf{x}$ belongs to class i is:

$$\begin{aligned} P(Y = i|x, W, b) &= \text{softmax}_i(Wx + b) \\ &= \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}} \end{aligned} \tag{5}$$

We employ the negative log-likelihood as cost function in supervised fine-tuning. Suppose $y^{(i)}$ is the true class for the input $x^{(i)}$, the cost function then states:

$$L(W, b, \mathbf{x}) = -\sum_{i=1}^{N} \log(P(Y = y^{(i)}|x^{(i)}, W, b)) \tag{6}$$

4.2.3. Protecting Against Overfitting Using Dropout Regularization

Neural networks are vulnerable to overfitting. To prohibit the DNN emphasizing idiosyncratic patterns of the training data and protect against overfitting, our DNN includes a dropout layer behind each hidden layer. Figure 3 depicts the concept of dropout. During DNN training, hidden layer neurons and their corresponding connection weights are removed from the network. This is done for each batch of training samples in an iteration. The gradients contributed by that batch of samples also bypass the dropped-out neurons during back-propagation (see the online Appendix for a detailed explanation of DNN training). The probability of a hidden neuron being dropped out follows a Bernoulli distribution with a given dropout rate.

A DNN trained with dropout mimics the behavior of an ensemble model. When calculating predictions, the DNN considers all hidden layer neurons but multiplies the connecting weights of each hidden

neuron by the expectation of the Bernoulli distribution. This way, although training a single DNN with N hidden neurons, the prediction of the DNN implicitly integrates predictions of 2^N candidate networks with different combinations of hidden neurons. More formally, dropout simulates a geometric model averaging process; each possible combination of hidden neurons is considered, which is the extreme case of bagging. Model combination is known to increase predictive accuracy [7, 46].

Dropout also acts as a regularizer in that it effectively removes random weights from training, which prevents hidden neurons from co-adapting to each other. Moreover, model averaging reduces variance, which, via the bias-variance decomposition, reduces forecast error. Controlling the complexity of a DNN, dropout helps to protect against overfitting. Theoretical details on dropout and how it prevents overfitting can be found in [54].

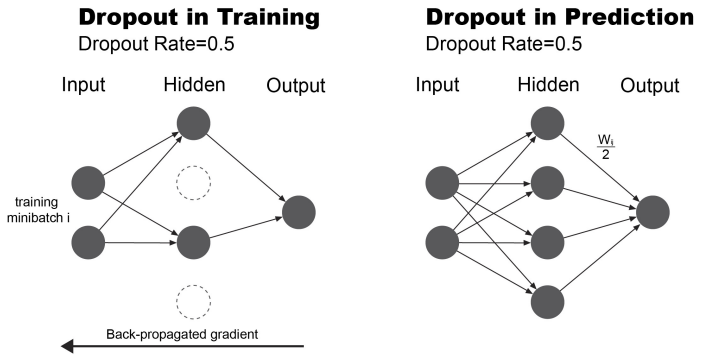

Figure 3: Principle of dropout in training and predicting.

Recall that we augment dropout through using an $L2-$penalty during SdA training to increase the robustness of the DNN toward overfitting. For the same reason, we make use of early-stopping.

4.2.4. Network Training and Configuration

Our DNN involves unsupervised pre-training using SdA. We tune weights in a layer-wiser manner and then fine-tune the DNN as a whole in a supervised way, with each hidden layer followed by a dropout layer. In addition, we use several other DL concepts to protect against overfitting and simplify the network training process including Xavier's initialization, batch normalization layers and using ReLU as the activation function. Previous work on DL has elaborated on these concepts and established their utility [26], and we detail them in the online Appendix. Figure 4 summarizes the overall architecture of the DNN employed for trader risk classification.

The parameters to determine in the pre-training stage are the weight matrix and bias in each dA (both the encoder and the decoder). The parameters in the supervised fine-tuning stage are the weight matrix and bias in each encoder of SdA and in the *softmax* regression. We use stochastic gradient descent with momentum and a decaying learning rate for DNN training. The online Appendix provides an explanation of these concepts and motivates our choices. In particular, Algorithm 1 in the online Appendix provides a fully-comprehensive description of network training using pseudo-code. Section 2 of the online Appendix also elaborates on our approach to decide on DNN hyper-parameters such as the

number of hidden layers in SdA, and how we tune these using random search [55].

The techniques we employ are available in DL software packages, which facilitate defining the topology of a DNN, provide routines for numerical optimization to train the DNN, and offer the functionality to apply a trained model for forecasting. We use the Python library *Theano*, which is a GPU-based library for scalable computing. The GPUs used for experiments were *Nvidia Tesla K20m* with 2496 cores and 5GB GDDR5 high bandwidth memory each. We observe this infrastructure to provide a 10-15 times improvement in speed over training a DNN using traditional CPUs for DNN training (which equates to reducing run-times from more than a week to 1-2 days). In appraising these figures it is important to note that i) large run-times result from the size of the data set, and that ii) training complex ML models may be as costly. For example, depending on the specific configuration, training a random forest classifier on the spread trading data can easily require more than 3 days.

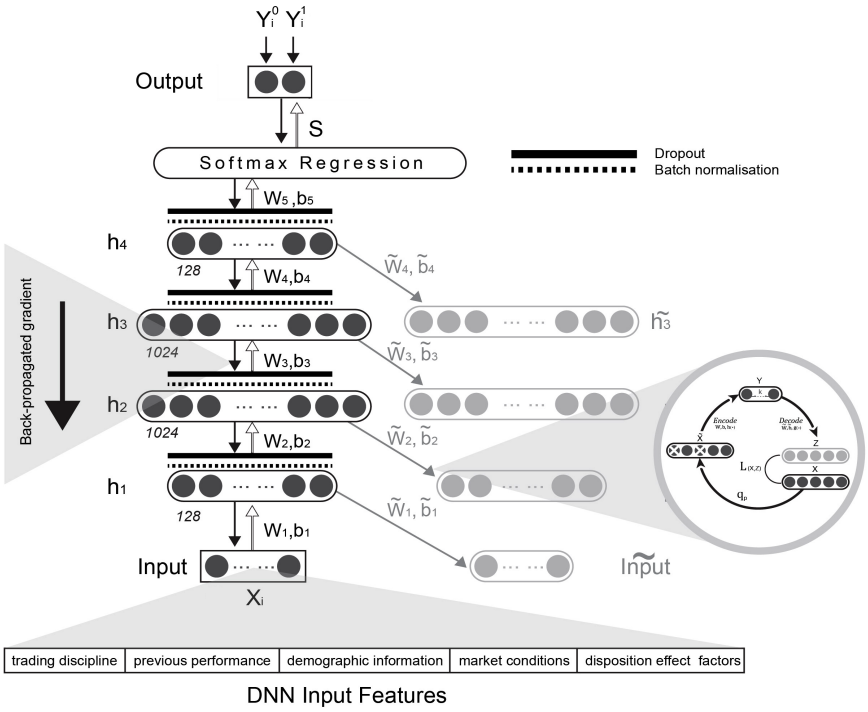

Figure 4: Topology of the deep network employed in this study. Stacked denoising auto-encoder with 4 hidden layers with $128, 1024, 1024, 128$ hidden units each. The output layer predicts class membership probabilities based on the output of the last dropout layer using the softmax function.

5. Experimental Design

The following sub-sections describe the spread-trading data set and elaborate on the definition of A-book clients, and introduce model evaluation criteria and benchmark classifiers.

5.1. Dataset and Target Label Definition

STX provided 11 years of real-life trading data for the period November 2003 to July 2014. Overall, the data includes the trades of $25,000$ active traders (over 30 million trades across 6064 different financial

instruments). To prepare the data for analysis, we replaced missing values using EM imputation and *Chebyshev*'s method for outlier treatment [11].

Supervised learning requires a labeled data set $D = \{y_j, x_j\}_{j=1...n}$, where x_j is a vector of features that characterize trade j, $n = 30$ million is the total number of trades in the data, and y_i denotes the target variable. However, data characterizing an individual trade is limited. Relating trades to their corresponding traders facilitates enriching the set of features by using information from previous trades $j - k$ to decide on trade j.

The decision task of STX is whether to hedge trade j. Therefore, we consider a binary target:

$$y_{ij} = \begin{cases} +1 \rightarrow \text{hedge} & , \text{ iif Return}_i \geq 5\% \\ -1 & , \text{ otherwise} \end{cases} \tag{7}$$

with (8)

$$\text{Return}_i = \frac{\sum_{20<j\leq 100} \text{P\&L}_{i,j}}{\sum_{20<j\leq 100} \text{Margin}_{i,j}}$$

where i, j index trader i and trade j, respectively, $P\&L$ is the profit and loss of trade j, and *Margin* is the amount of money required by the market maker in order to place the order, which normally equals the stake size times the margin requirement. To label trade j, we determine the status of trader i at the time of issuing that trade. We define trader i to be an A-book client if s/he secures a return above 5% from her next hundred trades subsequent to j. Recall that the 5% threshold mimics the current policy of STX. We also sustain the STX approach to hedge all trades from A-book clients. However, our method to define the client status and label their trades is forward looking whereas STX considers the past profits of trader i. Our target label definition is also dynamic in that the trader status can change with every trade. According to that definition, 6.43% of the trades in the data set come from A-book clients and should be hedged.

Of course, at the time when STX receives trade j, the future profits of trader i are unknown. Therefore, we develop a prediction model to forecast y_{ij} from the information the company can observe at the time when trade j is made. The feature vector x_{ij} includes demographic information of the client making trade j and information concerning the client's trading behavior for the 20 trades prior to trade j. The decision to consider the past 20 trades is based on the hedging policy of STX, which uses a rolling window of the 20 trades prior to trade j to decide on the status of the client.

5.2. Trader Characteristics and Feature Creation

We create variables for trader classification based on interviews with experienced members of the dealing desk of STX. A first round of interviews was aimed at identifying risk factors that domain experts deem indicative of good/bad traders. Based on corresponding results, we developed a semi-structured survey that was presented to seven members of the dealing desk in a second round of interviews. The survey asked participants to evaluate behavioral traits, which emerged from the first round, on a Likert Scale from 1-7, where values of 1 and 7 represent a strong indication for bad and good trading behavior, respectively. After completing the survey, we asked participants to suggest strategies they would apply if trading the FTSE100 index and a single stock from the FTSE100, respectively. This was to gather

ideas for novel factors not yet covered in the survey. The results of the interviews guided the feature engineering. A non-disclosure agreement with STX prohibits formally defining all features. However, the following description provides a comprehensive overview of the type of features and how they have been created. The features reflect the specific situation of STX. Risk analysts may find the following description useful to inform feature engineering in related applications. However, since our study focuses on the application of a DL methodology, it does not warrant claims related to generalizability of features. In general, features split into five groups. The first group comprises trader *demographics* such as age, country of origin, post code, employment status and salary group. Features of this group are nominal and enter ML models in the form a dummy codes. STX employs a range of socio- and micro-geographic data to cluster post codes. They follow a similar logic to cluster countries. [2]

Features of the second group capture the *past performance* of traders. We use aggregations such as the mean and standard deviation to calculate corresponding features over a rolling window of 20 previous trades relative to the focal trade. The choice of a window size of 20 follows STX's hedging policy at the time of the study. In addition to profitability, we compute a set of related performance indicators such as the average win rate, average number of points in profit, whether a client has been in profit, etc. We also consider the risk adjusted return (i.e., Sharpe ratio [56] and features related to the number and sizes of past withdrawals and deposits).

A third group of features describes traders' preferences related to *markets* and *channels*. For example, one feature simply counts the number of markets in which a trader invests while another encodes whether traders showed a strong preference for a specific market in their previous 20 trades. Using this information, we create features describing the most popular market cluster in a trader's full history and last 20 trades, respectively. The subgroup of channel preferences includes features that count the number of opening and closing trades made through the STX web front-end and mobile app, respectively, as well as ratios derived from these counts.

Results of the survey identified the *disposition effect* as a relevant factor to detect poor traders. The disposition effect [57] describes the phenomenon that investors tend to quickly sell trades that are in profit but are reluctant to sell trades in loss. Features of the fourth group strive to capture the disposition effect. We determine per trader the average amount and time s/he leaves winning and loosing positions open, and calculate their ratio. We also consider sums instead of averages and window lengths of the previous 20 and all previous trades.

Another discriminating factor that emerged from the interviews concerns *trading discipline*. Members of the dealing desk pointed out that good traders display a tendency to set manual limits (stop losses and profit levels) and when making profits to move these with the market. The fifth group of features captures signals concerning the consistency of a trader's strategy. The variation index of stake sizes exemplifies corresponding features. We also consider simpler features such as the standard deviation of

[2] STX has not revealed details of their cluster mechanisms to us. However, they assured us that the clustering does not employ any information of trader profits, which might otherwise introduce a look-ahead bias through leaking information from the prediction target to the features.

stake sizes and features that capture the frequency of trades as well as their variation. Other features in this group relate to the tendency of clients to trade within/outside of normal trading hours (e.g., number and share of corresponding trades), which we consider an indicator of traders' professionalism. The degree to which traders partially close trades may also signal expertise and hence traders posing a higher risk. Hence, we create a feature measuring the share of trades that have been closed in the previous 20 trades. The previous examples sketch the type of features we employ. Using operations such as varying window sizes, aggregation functions, creating dummy features through comparing a feature to a threshold (e.g., whether any of the last 20 trades has been closed using the mobile app), and considering bi-variate interactions, we obtain a collection of close to 100 features. An objective of the paper is to test whether the DNN can learn predictive higher-level features automatically. For example, the discussion on feature engineering suggests multicollinearity among features, which feature selection could remedy. However, a sub-goal following from our objective is to test how effectively the DNN automatically discards redundant and irrelevant features. Therefore, we do not perform feature selection.

5.3. Exploratory Data Analysis and Feature Importance

To shed light on how A-book and B-book clients differ across the features, we report results of an exploratory data analysis. Table 2 reports descriptive statistics for the ten most informative features for A-book and B-book clients, respectively. We select these features according to the Fisher score [7]. Features with the suffix 20 are calculated over a window of 20 past trades relative to a focal trade. For example, given a trade j (equivalent to one observation in the data set) from a trader i, we consider the $j-1, j-2, ...j-20$ trades of trader i and calculate their mean, standard deviation, etc. We use all available trades of a trader if s/he has less than 20 trades. In interpreting the results of Table 2 it is important to note that STX rescaled numeric features to the zero-one interval using min/max scaling. Rescaling is a common data preprocessing approach and ensures comparability of feature values. In addition, it helps to protect the confidentiality of the data.

Table 2 reveals that differences between the client groups in the means of variable values are small. This indicates that good and bad traders cluster in the behavioral space spanned by these features and that a classification of traders using these features will be challenging. To support this view, we estimate a logistic regression model on the training set using the features of Table 2 and observe a McFadden R^2 close to zero. Considering standard deviations, Table 2 suggests that the trading behavior of B-book clients is slightly more volatile compared to A-book clients, which supports findings from the interviews that good traders follow a consistent strategy. Table 2 also emphasizes the disposition effect as a potentially discriminating factor. Several of the top ten features aim at capturing the disposition effect through contrasting the duration with which traders keep winning versus losing positions. Last, the third and fourth moment of the feature distributions hint at some differences between good and bad traders. However, as shown by the failure of the logistic regression, translating these differences into a classification rule is difficult and may be impossible with a linear model.

To further inspect the origin of close to random performance of logistic regression (on all features) and to gain more insight into the feature-response relationship, we also examine feature importance using

Table 2: Descriptive Statistics of Top-Ten Features

Feature	Mean		Std.Dev.		Skew		Description
	A-book	B-book	A-book	B-book	A-book	B-book	
ProfitxDur20	0.325	0.332	0.172	0.178	0.994	0.962	Interaction of ProfitRate20 and DurationRate20
SharpeRatio20	0.443	0.446	0.081	0.085	1.097	1.131	Mean/st.dev. of returns
ProfitRate20	0.496	0.504	0.241	0.248	0.346	0.328	Average profit rate of client
WinTradeRate20	0.621	0.626	0.203	0.207	-0.203	-0.210	Clients average winning rate
AvgOpen	0.534	0.539	0.218	0.228	-0.345	-0.311	Average of the P&L among trader's first 20 trades
DurationRate20	0.319	0.322	0.119	0.121	-0.148	-0.161	Average time client leaves winning vs losing position open
PerFTSE20	0.251	0.244	0.357	0.353	1.151	1.197	Share of trades placed in the FTSE100
DurationRatio20	0.127	0.128	0.067	0.070	3.398	3.812	Mean trade duration (mins) / std.dev. trade duration
AvgShortSales20	0.487	0.482	0.269	0.274	-0.027	-0.018	Share of short positions
PassAvgReturn20	0.502	0.503	0.052	0.057	-0.295	0.065	Average return up to the last 20 trades

a random forest (RF) classifier. Feature importance scores extracted from tree-based ensemble classifiers are a popular way to quantify the relative impact of features on the response variable [11]. Figure 5 depicts the distribution of RF-based normalized importance scores for the first fifty features (ordered in terms of importance); the remainder being omitted to ensure readability. We highlight those features that have previously been identified as important by the Fisher-score.

Figure 5 reveals differences between the variance adjusted comparison of group means, which the Fisher-score embodies, and the RF-based ranking. For example, the strongest feature according to Table 2, *ProfitxDur20*, does not appear in Figure 5 and the highest rank that a feature of Table 2 achieves in Figure 5 is ten as observed for the feature capturing a trader's average over the last twenty trades prior to the decision point. Interestingly, this feature, *PassAvgReturn20*, is the one that STX use in their hedging policy.

RF generates importance scores through comparing (out-of-bag) classification performance on the original data and that data after corrupting one feature through adding random noise. The magnitude of the performance decrease captures the importance of the corrupted feature [11]. This implies that RF assesses the importance of one feature vis-a-vis all other features, whereas the Fisher-score assesses one feature at a time. Given the different mechanism to measure importance, some differences between the RF and Fisher-score ranking are to be expected. It is still surprising that the most important features of the latter receive relatively low ranks in Figure 5. An interpretation of this result is that it evidences intricate dependencies between the binary response and features, which the Fisher-score does not capture. This interpretation agrees with the poor performance of the logit model. As detailed in Section 6.1, the

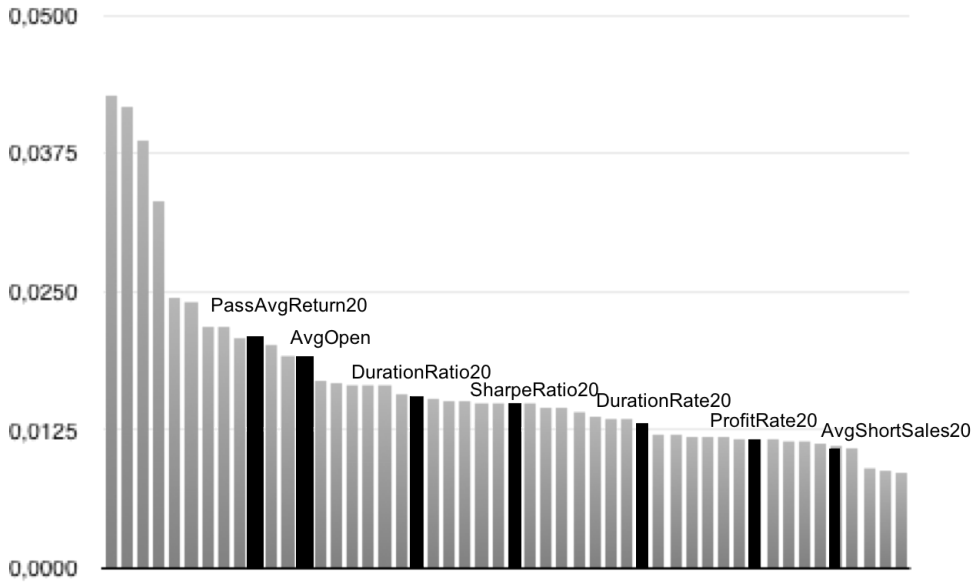

Figure 5: Normalized variable importance scores based on RF-classifier for the top 50 features. Dark color identifies features that also appear in the Fisher-score ranking (Table 2)

performance of the logit model improves but remains inferior to more expressive nonlinear classifiers after accounting for multicollinearity.

With respect to the complexity of the feature-response relationship, the distribution of importance scores in Figure 5 may be considered evidence of a set of three to four features being particularly strongly related to the response. We caution against this interpretation. The distributional shape is a consequence of the scaling of the y-axis, to ensure readability of the figure. The magnitude of importance scores is small, even for the left-most features. Therefore, importance differences between features (e.g., feature four and five) appear more substantial than they are. Recall that the scores capture the degree to which RF performance decreases if we corrupt one feature. Given the magnitude of importance scores, we interpret the results of Figure 5 as evidence of a low signal between the raw features and the future success of a trader. This emphasizes the trader classification task to be challenging. Even a powerful RF classifier, often observed to predict accurately [46, 7, 32], fails to identify strong dependencies among the raw features and the target. Low importance scores also question representativeness of the training data. This motivates our analysis whether a DNN, equipped with higher depth than RF, is able to learn more abstract, latent, features that enable predicting traders' future performance more accurately than conventional 'shallow' learners.

We complete the analysis of feature importance by aggregating importance scores across the main feature groups in Figure 6. The analysis offers insight as to the relative importance of different types of trader characteristics. The results displayed in Figure 6 agree with the views of STX dealing desk members. We find trader demographics and features in the markets & channels category to carry least weight, which reinforces the view that propensity for risk taking may be attributed to the competence and trading style rather than particular country of origin or gender. Past performance and trading discipline are most important for high risk trader identification, substantiating the claim that features

capturing the professional behavior of traders are of primary value for the task at hand.

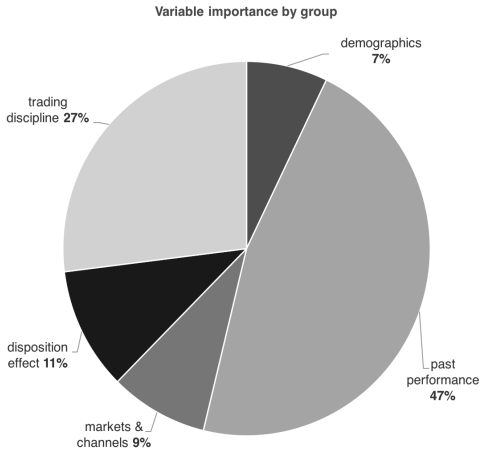

Figure 6: Analysis of group-level feature importance. The aggregation is performed by adding up the RF-based importance scores of all features belonging to the same group and normalizing group-level scores to sum to unity.

5.4. Data Organization, Evaluation Criteria and Benchmark Classifiers

Testing the predictive performance of ML models requires assessing the accuracy of their forecasts on hold-out data not used during training. Several strategies for data organization exists. We employ n-fold cross-validation, which involves randomly partitioning the data into n folds of approximately equal size, training a model on the union of $n-1$ of these folds, and assessing the performance of the resulting model through comparing actual classes to model-based class probability predictions on the remaining fold. Repeating model building and assessment n times increases the robustness of results compared to a single partitioning of the data into one training and one test set. We consider settings of $n = 10$ and $n = 5$ in subsequent comparisons. [3]

The client classification problem exhibits class imbalance and asymmetric error costs. Hedging a trade that eventually leaves the trader with a loss diminishes the profit margin of the market maker. Failing to hedge a high risk trade is far more severe and may leave the market maker with a very large loss. To reflect this asymmetry, we evaluate a classification model in terms of the profit or loss (P&L) that results from hedging trades according to model predictions.

The P&L assesses classification performance in that it is based on discrete class predictions. Taking cost asymmetry into account, it is a more suitable performance indicator than conventional metrics such as classification error, the F-measure, or others. However, to augment the P&L-based evaluation, we also assess classification models in terms of the area under a receiver-operating-characteristics curve (AUC). The AUC is equivalent to the Mann-Whitney-Wilcoxon U statistic. Considering a randomly chosen A-book client and a randomly chosen B-book client, the AUC approximates the probability that a classifier

[3]The computationally simpler train-test split setup was considered in preliminary experiments to identify suitable benchmark classifiers and examining the impact of class imbalance on these classifiers and the DNN. Interested readers find corresponding results in the online Appendix.

assigns a higher score to the A-book client [58]. In this interpretation, we use the term *score* to refer to the classifier-estimated probability of a client being a high risk trader. A notable feature of the AUC is that it captures the discriminatory ability of a classifier to rank order cases in the right order; for example, assigning higher (lower) probabilities to A-book (B-book) clients. The evaluation is independent from a classification threshold and the degree to which probabilistic predictions are well-calibrated [59]. Hence, the AUC assesses the model from a different angle than the P&L.

To compare the performance of our DNN to benchmarks, we select four ML classifiers as benchmarks, including logistic regression, ANNs, RF, and adaptive boosting. A comprehensive description of the classifiers is available in, e.g., [11]. We report the hyper-parameter settings that we consider during model selection in Section 2 of the online Appendix, where we also elaborate on hyper-parameter tuning.

6. Empirical Results

The empirical analysis compares the proposed DNN to benchmark classifiers and rule-based hedging strategies that embody domain knowledge.

6.1. Predictive Accuracy of the DNN and ML-based Benchmark Classifiers

We first present results concerning the predictive performance of different classifiers in Table 3. The AUC assesses models in terms of their ability to discriminate A- and B-book clients whereas P&L values capture the profitability of a model-based hedging policy. To ensure comparability across folds, we normalize the total P&L observed in one cross-validation fold by the number of traders in that fold. For example, a value of 296 GBP in the first fold in the base scenario where STX does not hedge against any trade indicates that the average client loses this amount of money from trading with STX, which is equivalent to the profit of STX.

Table 3 reveals variations in model performance across different folds, which emphasizes the merit of comparing models using cross-validation. Considering P&L, we observe the DNN to provide the best performance in seven out of ten folds. Accordingly, the DNN also achieves the highest average P&L and outperforms benchmarks by a sizeable margin. For example, the average P&L per trader (across folds) of the DNN is 2,931 GBP compared to 2,079 GBP of a hypothetical base setting in which STX would not hedge any trade. Compared to the second highest average P&L of 2,690, which comes from the logit model, the DNN provides a nine percent improvement. The P&L is informative for risk managers as it estimates the value of model-recommended hedging decisions. The AUC offers an additional perspective on model performance. Unlike P&L, which depends on the specific trades against which a model recommends hedging, the AUC emphasizes a model's discriminatory ability, that is whether it assigns higher risk scores to actual A-book clients. The AUC supports the appealing performance of the DNN. It achieves the highest performance in each fold and performs substantially better than the benchmarks in the comparison. For example, the second best benchmark in terms of the AUC is the RF classifier, which produces an average AUC of 0.720 c.f. an average AUC of 0.814 for the DNN).

Table 3: DNN Performance vs. Benchmarks in terms of P&L and the AUC

	Cross-validation folds										
	1	2	3	4	5	6	7	8	9	10	mean
	Average P&L per trader in GDP										
no hedging	2268	2130	1601	2122	2230	2536	1785	1938	1870	2306	2079
DNN	2245	2906	3490	2863	2245	3536	2679	3536	3014	2792	2931
Logit	3021	2281	2901	2707	3413	2643	2443	2358	2452	2679	2690
ANN	2619	2624	2207	2614	3002	2515	2594	2503	2691	2920	2629
RF	2745	2539	2255	2559	2574	3198	2624	2295	2578	2451	2582
AdaBoost	2402	2756	1869	1857	2435	2956	2215	2482	2672	2741	2439
SVM	1511	2656	1382	1278	1140	1796	835	3149	312	2402	1646
	Area Under Receiver-Operating Characteristics Curve (AUC)										
DNN	0.816	0.806	0.809	0.802	0.826	0.804	0.842	0.816	0.782	0.832	0.814
ANN	0.633	0.643	0.645	0.638	0.625	0.645	0.640	0.645	0.618	0.616	0.635
Logit	0.708	0.698	0.716	0.699	0.690	0.704	0.735	0.692	0.708	0.701	0.705
RF	0.714	0.734	0.726	0.728	0.736	0.721	0.710	0.717	0.684	0.730	0.720
AdaBoost	0.647	0.631	0.637	0.618	0.648	0.656	0.635	0.650	0.625	0.632	0.638
SVM	0,688	0,887	0,692	0,691	0,794	0,664	0,695	0,586	0,625	0,592	0,690

6.2. Antecedents of DNN Forecast Accuracy

Table 3 evidences the superiority of the DNN over ML-benchmarks for the specific data set used here. To examine the robustness of model performance, it is important to clarify the antecedents of DNN success. One characteristic feature of the DNN is its multilayered - deep - architecture. Previous research establishes a connection between the depth of a model and its expressive capacity [25], which suggests depth to be a determinant of predictive accuracy. Another characteristic that distinguishes the proposed DNN from ML benchmarks is its use of unsupervised pre-training. Aiding model training through finding more abstract, generative features, we expect predictive accuracy to benefit from pre-training. A third factor of interest is class imbalance. Skewed class distributions are a well-known impediment to classification and only seven percent of the traders in the data are A-book clients. Therefore, the fact that the DNN is more robust toward class imbalance than the ML benchmarks could also explain the results of Table 3. In the following, we examine the influence and importance of these three factors.

6.2.1. The Deep Architecture

The DNN generates predictions in the last layer, where the last layer output neuron receives the combined input from multiple previous layers of SdAs and translates these signals into class probability predictions using the softmax function. This network configuration is equivalent to running logistic regression on the output of the hidden layers. To shed light on the value of the deep architecture, we compare DNN predictions to predictions from an ordinary logistic regression with the original features

as covariates. The logistic model represents an approach which takes away the deep hidden layers from the DNN and only sustains the last layer. This is useful for appraising the merit of the distributed representations, which the deep hidden layers extract from the raw features.

Figure 7 displays the receiver-operating-characteristics (ROC) and a Precision-Recall (PR) curve for the DNN and a simple logit model. The plot emphasizes that the deep architecture substantially improves the network's discriminative ability. The performance of the logit model on raw features is almost random. The AUC value of 0.812 for the DNN suggests that performing the same regression on the high level representations, which the DNN learns from the raw features, facilitates a reliable detection of the positive class. Consequently, the DNN succeeds in extracting predictive features from the input data. In appraising Figure 7 it is important to note that the logit model is not meant to contribute a strong benchmark. As shown in Table 3, a regularized logit model with feature selection performs better than random. The purpose of Figure 7 is to evaluate the overall effect of the deep architecture compared to using the raw features as is, which motivates using the ordinary logit model for this comparison. The overall conclusion emerging from the analysis is that the deep architecture affects the predictive performance of the DNN.

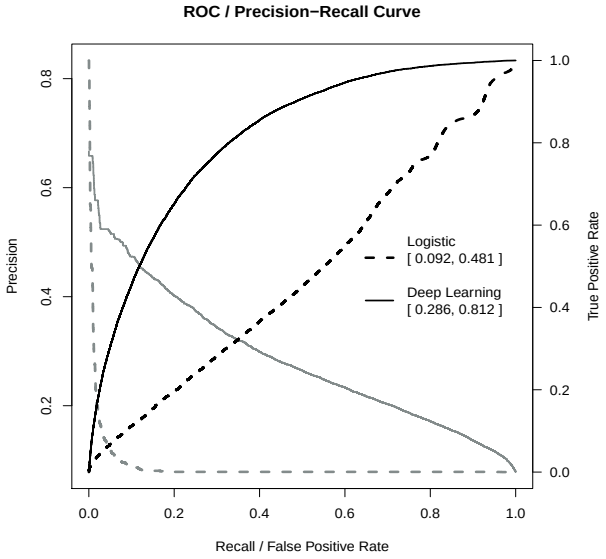

Figure 7: ROC (black), Precision-Recall Curve (grey) of deep learning and logistic regression. Results are based on a DNN model estimated from the first 70% of the data and applied to predict risk scores for the remaining 30% of trades. Curves depict model accuracy across these 30% trades.

6.2.2. Unsupervised Pre-Training

The proposed DNN uses unsupervised pre-training for representation learning and feature extraction. To confirm the merit of pre-training, we examine the discriminative strength of each neuron in the unsupervised pre-training stage. We aim to check whether DNN learns distributed representations that help differentiate A- and B-book clients from *unlabeled* data. To that end, Figure 8 provides the histograms of activation values for neurons in the first dA layer of the DNN. The histograms show that activation values tend to be less than 0.4 when receiving a trade from a B-book client. Trades from A-book clients

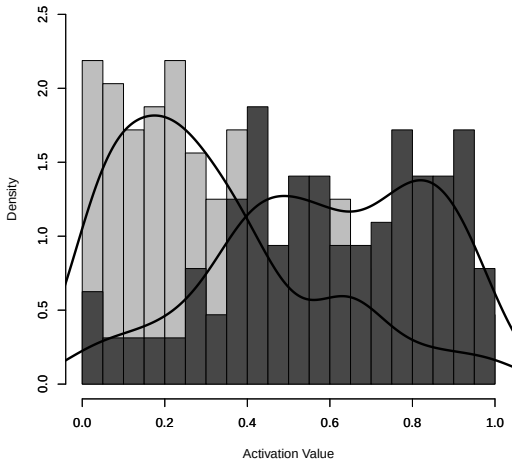

Figure 8: Histogram of activation values of neurons in the first dA layer for A-book (deep color) and B-Book (light color) client trades. The test set is re-sampled such that the ratio between high risk and normal traders is one.

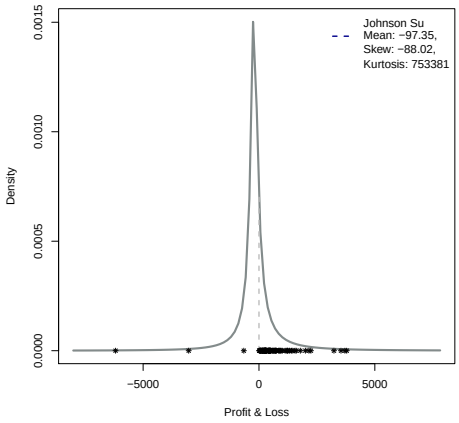

Figure 9: Top 100 stimuli of the best neuron from the test set

typically result in an activation value of 0.4 and above. While the magnitude of the activation values is irrelevant, the discrepancy of activation values for trades from different types of clients illustrates that - even with unlabeled data - the neurons in the first dA layer differentiate A- from B-book client trades. The intricate non-linear transformation between layers prohibit a replication of this analysis for higher layers because the relationship between activation values and input signals is no longer monotone. However, Figure 8 provides preliminary evidence that the spread trading data facilitates the extraction of higher-level generative features using pre-training.

To substantiate the analysis and gain more insight into the link between neuron activation values and trades from different types of traders, we examine whether trades that trigger high activation values in a neuron are indeed worth hedging. To that end, we first calculate the maximum and minimum activation values for every neuron of the first layer, and 20 equally spaced threshold values between these boundaries. Subsequent analysis is based on a single neuron. We chose the neuron and corresponding threshold that give the purest separation between A- and B-book client trades (see Figure 8) upon manual inspection. Using this neuron, we find the 100 trades in the test set that activate the neuron the most. Figure 9 plots these trades on the overall P&L distribution. The results illustrate that, with a few false negatives, 97% of the trades that maximally activate the neuron end in profit and leave the market maker with a loss. Hedging against these trades, as indicated by the neuron's activation levels, is economically sensible. Although the eventual hedging decisions are based on the prediction of the DNN as a whole, the single neuron analysis provides further evidence of unsupervised pre-training of SdA layers to extract patterns that are indicative of a trade's risk. This confirms that the DNN learns distributed representations from the input data, which eventually help to distinguish high risk traders from other clients.

6.2.3. Analysis of the Class Imbalance Effect

A growing body of literature on deep imbalanced learning indicates that DL models inherit vulnerability toward class imbalance from their ML ancestors [60]. However, it seems plausible that the DNN is more robust toward the adverse effect of imbalance than the ML benchmarks due to pre-training. Pre-training is carried out in an unsupervised manner. Therefore, class imbalance cannot occur. Figure 8 indicates that, without having access to class labels, pre-training has extracted patterns that help to differentiate high risk traders and B-book clients. Only the DNN has access to this information, which might give it an advantage over the ML benchmarks in the comparison of Table 3. To test this, we repeat the comparison after addressing class imbalance using the SMOTE (synthetic minority class oversampling technique) algorithm. SMOTE remedies class skew through creating artificial minority class examples in the neighborhood of actual minority class cases [61].

Table 3 in the online Appendix reports detailed results of classifiers after applying SMOTE. Given that oversampling increases the number of observations and, in turn, the time to train different learning algorithms, we reduce the number of cross-validation folds and estimate performance using 5-fold cross-validation. More specifically, we configure the SMOTE algorithm such that it produces artificial A-book clients until both classes are balanced. Figure 10 summarizes corresponding results through depicting the average cross-validation performance for each learning algorithm before and after applying SMOTE in terms of P&L and the AUC.

Figure 10 reveals that SMOTE consistently improves the predictive performance for all models. P&L and AUC are substantially higher after addressing class imbalance, which reemphasizes the adverse effect of the latter. We also observe that the margin with which the DNN outperforms ML benchmarks decreases. For example, the strongest benchmarks after oversampling in terms of P&L and the AUC are the logit and ANN benchmark, respectively. The DNN performs 6 (9) and 4 (13) percent better than these competitors, where numbers in brackets denote the corresponding percent performance improvement in the original (i.e., imbalanced) data. A first interpretation of this result is that Table 3 gives an optimistic picture of DNN performance. The accuracy gap between the DNN and the ML benchmarks is less than Table 3 suggests if ML benchmarks receive auxiliary tuning in the form of addressing class imbalance. In addition, Figure 10 also confirms the DNN to be more robust toward class imbalance. While benefiting from SMOTE, its ability to identify high risk traders accurately is less dependent on oversampling the minority class compared to the ML benchmarks. This agrees with results of Figure 8 concerning the merit of unsupervised pre-training.

6.3. Implications for Risk Management

A model-based hedging policy comprises hedging the trades of clients classified as A-book by the model and taking the risk of all other trades. To clarify the managerial value of the proposed DNN, we compare the P&L of a DNN-based hedging strategy against that of rule-based strategies. One rule-based approach is the current policy of STX, which involves hedging trades of clients who secured a return above five percent in their previous 20 trades. In addition, we develop three custom hedging heuristics. Our first policy, *Custom 1*, relies on the Sharpe Ratio and singles out traders who achieve a higher

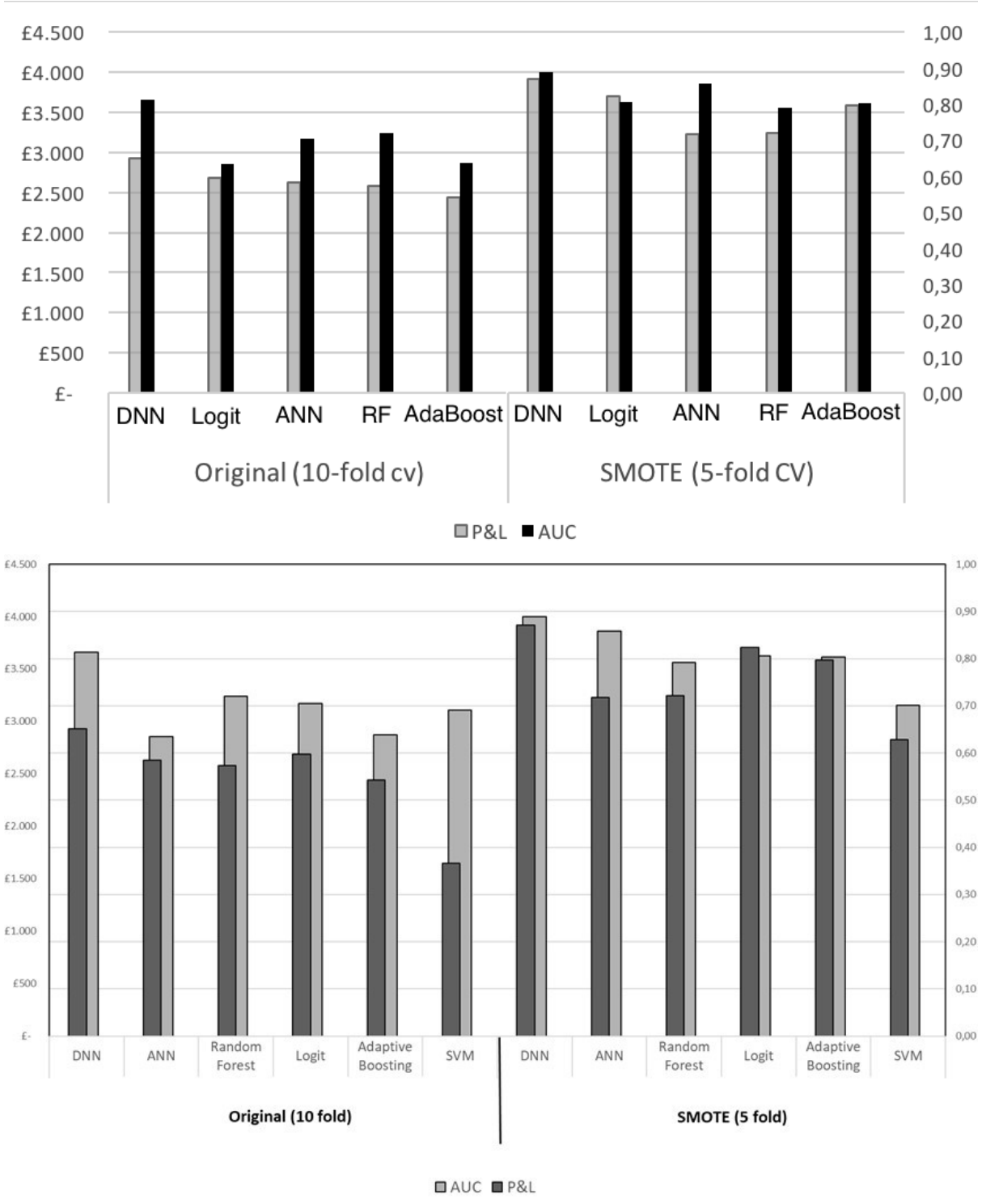

Figure 10: Cross-validation performance in terms of P&L before and after SMOTE.

than average Sharpe ratio in their past 20 trades. We suggest that securing risk-adjusted returns above the average indicates trader expertise. Since professionalism is only one reason for a successful trading history, *Custom 2* heuristic addresses another group of traders, which we characterize as overconfident. Such traders may display higher yields than other market participants and exhibit aggressive trading behavior, manifesting itself through bigger lot sizes, higher frequency and shorter time interval trades [62]. The Custom 2 heuristic thus considers the average trade duration and number of trades to deduce traders who may pose a greater risk. The third strategy, *Custom 3*, hedges trades from clients with a positive track record since trading with STX. The rationale is that traders who are unsuccessful in their early experiences might quit. Traders with a longer track record are either truly successful (and should be hedged against) or gamblers with a negative expected value (and should not be hedged against). Following this line of thinking, the most important risk STX is facing comes from *new* A-book clients. Comparisons to Custom 3 shed light on the ability of the DNN to identify such new A clients, as improvement over Custom 3 signals the DNN recognizing high risk traders that the track record-based logic of Custom 3 fails to capture.[4] We also consider an ensemble of the custom rule-based heuristics, constructed by means of majority voting.

Drawing on domain knowledge, the rule-based strategies adopt a deductive approach. To complement the analysis, we also add one inductive rule-based approach in the form of a classification tree. Trees are regarded as interpretable classifiers. However, the degree to which decision makers can understand trees depends on their depth. In the interest of interpretability, we consider a classification tree (ctree) with two levels.

Table 4: Average P&L per trader in GBP of the DNN-Based and Rule-Based Hedging Strategies

	Cross-validation folds										
	1	2	3	4	5	6	7	8	9	10	mean
no hedge	2.268	2.130	1.601	2.122	2.230	2.536	1.785	1.938	1.870	2.306	2.079
DNN	**2.245**	**2.906**	**3.490**	**2.863**	**2.245**	**3.536**	**2.679**	**3.536**	**3.014**	**2.792**	**2.931**
STX	2.014	2.229	2.155	2.238	1.878	1.913	1.872	2.190	2.937	1.811	2.124
custom 1	1.534	1.549	1.236	1.550	1.417	1.488	1.392	1.618	1.986	1.341	1.511
custom 2	1.969	1.831	1.486	1.440	2.299	1.724	2.076	1.594	1.594	2.046	1.806
custom 3	1.736	1.825	1.579	1.965	1.865	1.937	2.000	1.785	2.402	1.750	1.884
ensemble	1.869	1.950	1.530	2.151	1.897	1.852	1.784	1.656	2.480	1.693	1.886
ctree	2.163	1.544	2.799	2.154	2.089	2.299	1.972	2.051	2.215	1.974	2.126

Table 4 reveals the current policy of STX to be the most suitable deductive hedging strategy. The logic of Custom 1 - 3 draws on financial theory. However, each of the three approaches, as well as an ensemble of them, performs worse than a hypothetical baseline setting in which STX would not hedge any trade. Observing Custom 1 - 3 to perform worse than this baseline supports the view that the focal

[4]We are grateful to an anonymous reviewer who suggested the logic of the Custom 3 heuristic.

trader classification task represents a challenging problem. Following this line of reasoning, Table 4 also emphasizes the soundness of the STX policy.

Unlike the deductive STX approach, the tree-based heuristic learns from past data. More specifically, the tree uses three features for splitting the data: a trader's average P&L in their initial 20 trades with STX, the minimum number of minutes until closing a losing position in their last 20 trades, and the average Sharpe ratio over their last 20 trades. These features display similarity with the custom heuristics. For example, considering a trader's initial performance follows the logic of Custom 3 while account for risk-adjusted returns is similar to Custom 1. Finally, considering a trader's reaction toward losses, the tree uses one of the variables to capture the disposition effect. We observe the two-level tree to produce slightly larger P&L than the STX heuristic. This suggest that a trader's average past performance, embodied in the STX approach, approximates the more complex rule set of the tree with some accuracy. Although the criticality of accurate hedging in the spread trading market suggests a revision of the STX approach with a tree-based approach, another finding from Table 4 is that implementing a DNN-based hedging strategy enables STX to further improve P&L compared to its current policy and the other rule-based hedging strategies we consider. Compared to the STX heuristic, the DNN raises per trader profits by 2,931 - 2,124 = 807 GBP, which implies a substantial, managerially meaningful improvement when considering the total number of clients of STX. For example, the data set used here comprises roughly 25K active traders.

The STX heuristic represents an established business practice at the partner company and reflects many years of industry experience. Moreover, the heuristic is extremely fast to execute and completely transparent. The situation for the DNN is far different. Classifying incoming trades more accurately, a DNN-based hedging policy is more profitable than rule-based approaches. The main cost of accuracy and profitability improvements is the black-box character of the corresponding risk management system. The client classification rules from the DNN originate from automatically extracted distributed representations of high risk traders. The business logic encapsulated in these rules is not interpretable for decision-makers, which also prohibits testing the agreement of these rules with domain knowledge.

Improved performance of the DNN leaves risk managers with the task to decide whether performance improvements are large enough to compensate the opaqueness of DNN and associated disadvantages, such as a lack of justifiability, higher computational requirements, etc. In the case of STX, we expect the imperative to hedge trades accurately and the magnitude of the performance improvement observed on their data to justify the adoption of a sophisticated DNN-based hedging strategy. The same might be true for other the spread-trading companies, although these would first need to replicate the results of this study to confirm the effectiveness of the DNN. A detailed description of the DNN configuration in the paper and especially the online Appendix will hopefully simplify this task. A more strategic consideration is that reluctance to adopt a new technology such as a sophisticated DNN-based hedging policy might also harm the competitive position of STX if competitors deploy corresponding solutions and use them to offer better prices to retail investors. At the same time, we caution against an overly optimistic view toward advanced DL-based decision aids. The empirical results observed in this study come from a single data source, which, although large in size, reflects the peculiarities of the market

position and client structure of STX, and require a replication with different data in future research to raise confidence in the superiority of DL that we observe here.

Given that the main disadvantage that we associate with the DNN is opaqueness, we conclude this chapter with acknowledging that DL and other complex ML models are not incomprehensible per se. An approach called information fusion-based sensitivity analysis provides insight into the relationship between model inputs (i.e. features) and outputs (i.e., forecasts) in any type of ML-based prediction model, including DNNs [63]. Previous finance applications of this approach [64, 8] demonstrate how it enables interpreting black-box ML models.

7. Discussion

The empirical results suggest that the DNN approach outperforms rule-based and ML benchmarks. It identifies high risk traders more accurately than other classifiers and provides higher financial gains when used for hedging decisions.

Predicting traders' risk taking behavior and future profitability under dynamic market conditions is challenging. Traders differ in their characteristics and trading behavior, and both are likely to change over time. Identifying unskilled traders is especially difficult due to the high variation in both behavior (input) and performance (output). Compared to genuine good traders, it is harder to identify uniform trading patterns for poor traders. Interviews with STX's staff hint at skilled traders sharing certain characteristics such as the ability to capture market rallies, following a consistent strategy, setting and adjusting limits, etc. On the other hand, there are countless ways in which poor traders lose money, including ignoring any of the above rules. In the high dimensional behavioral space, the number of potential variations of poor traders is innumerable. This contradicts the prior assumption of ML methods that the distribution $P(label|features)$ is smooth and well represented in the training data. Consequently, conventional ML faces difficulties in profiling trading patterns. The deep architecture equips DNN with higher expressive capacity to store the large number of variations of trading behaviors. Complexity theory shows a function that can compactly be represented by an architecture of depth k to require an exponentially growing number of computational units to represent the same function with smaller depth [3]. This suggests that increased depth enables the DNN to profile new combinations of behavioral variations and generalize to new trading patterns less represented in the training data.

Furthermore, the chance of making profit in the spread-trading market is highly noisy. Even poor traders can, by luck, win money. In fact, Figure 11 reveals that most of the clients who trade with STX have a greater than 50% win/lose ratio. However, even though traders win money on more than 50% of their trades, Figure 11 shows that average losses exceed average winnings by a large margin. Therefore, it is often sensible to classify a trader as a B-client and refrain from hedging their trades, even if many of their previous trades ended in profit.

Although based on an economic rationale, input features relating to past risk-adjusted return, trading frequency, etc. do not facilitate an accurate discrimination of spurious from genuine good traders. This arises because several feature values may coincide. The entanglement of spurious and genuine good traders in the behavioral feature space of trader characteristics further complicates the trader

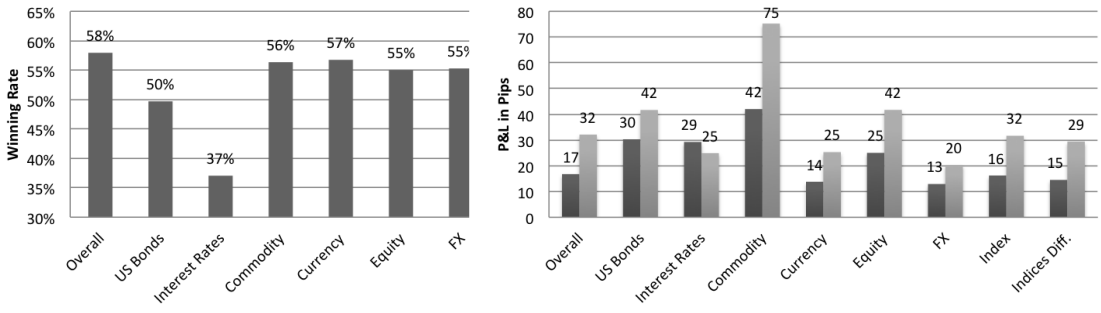

Figure 11: Retail traders' average winning ratio and average P&L points (profit in dark, loss in grey) on different categories of investments on the spread trading market.

classification problem and harms conventional ML methods. The DNN draws upon the raw features and creates sensible abstractions from these features that exhibit a stronger connection with the target.

A specific DNN component we employ for trader classification is unsupervised pre-training. Observed results confirm that pre-training enables the DNN to construct layers of feature detectors that capture underlying generative factors, which explain variations across different trading behaviors. Stacking multiple layers of progressively more sophisticated feature detectors, the DNN learns to disentangle these factors from the input distribution. Variations that are important for subsequent discrimination are amplified, while irrelevant information within the input data is suppressed [65]. We examine this ability in Figure 7, 8 and 9. After pre-training, the higher levels of the feature hierarchy store robust, informative, and generalizable representations that are less likely to be misled - and, thus, invariant to - the entangling of trading patterns in the input-space.

8. Conclusions

We set out to examine the effectiveness of DL in management support. Corresponding applications often involve developing normative decision models from structured data. We focus on financial risk taking behavior prediction and develop a DNN-based risk management system.

The results obtained throughout several experiments confirm the ability of DL, and the specific architecture of the DNN we propose, to extract informative features in an automatic manner. We also observe DNN-based predictions of trader behavior based on these features to be substantially more accurate than the forecasts of benchmark classifiers. Finally, our results demonstrate that improvements in forecast accuracy translate into sizable increases in operating profit. This confirms the ability of the proposed DNN to effectively support (hedging) decision making in this risk management case study.

Our findings pave a way to approach other behavior forecasting problems using DL. For example, direct marketers can increase the likelihood of consumers' responding to a promotion by studying clients' buying behaviors. Banks can enhance their risk control and make sensible credit approval decisions by analyzing clients' credit repayment behavior. E-commerce companies can dynamically adjust website layouts according to visitor preferences. These are only a few examples out of the vast space of tasks in decision support which generate large amounts of structured data and are routinely supported by ML.

We provide evidence that the methodology reported here offers potentially significant gains in forecasting accuracy. Reappraising these gains in the scope of other business applications is essential to confirm that the appealing performance of the DNN that we observe is not specific to this case study.

Acknowledgement

We thank the editor, Prof. Teunter, for his efforts in handling our paper and are thankful to three anonymous reviewers whose feedback has helped tremendously to improve earlier versions of the paper. We are especially grateful to J.C. Moreno Paredes for his invaluable help with data preparation.

ONLINE APPENDIX

The online appendix complements the paper through i) further elaborating on the training of the trader classification DNN, ii) documenting candidate hyper-parameter settings of the DNN and the ML benchmark classifiers and how we have tuned these in a model selection, and iii) providing additional empirical results. These results include empirical findings from pre-tests based on a static split sample design. We also report the distributions of the performance indicators that we have used in the comparison of the proposed DNN to benchmark ML classifiers. Finally, we provide additional results of a comparison of the DNN to alternative DL models.

1. Training the Proposed DNN

The DNN employed in the paper integrates multiple DL concepts. In summary, the DNN is first pre-trained via SdA, with Xavier's initialisation for weights and ReLU unit as the activation function, in a greedy layer-wiser manner. This is the unsupervised pre-tuning stage after which we fine-tune the DNN as a whole in a supervised way, with each hidden layer followed by batch normalisation and dropout. In the following, we detail the optimisation of the cost function, , which we did not detail in the main body of the paper for brevity; namely, the training of the parameters of the DNN in both the pre-training and fine-tuning stage as well as other DL concepts such as Xavier's initialization, ReLU, and batch normalization.

The parameters to train in the pre-training stage are the weight matrix and bias in each dA (both the encoder and the decoder), and the parameters to train in the supervised fine-tuning stage are the weight matrix and bias in each encoder of SdA and in the *softmax* regression. The rest of the parameters (e.g., the number of hidden layers in SdA, the number of hidden neurons in each hidden layer), are hyper-parameters that need adjusting on top of the training process. We elaborate on the treatment of such hyper-parameters below in Section 2.

1.1. Xavier's initialization

The solution to a non-convex optimization problem depends on the initial values of the weight parameters W. By default, SdA initialize weights randomly. If network weights are initialized too small (large), the signal shrinks (expands) as it passes through each layer until it becomes too tiny (massive)

to be useful. Xavier's initialization [66] guarantees that weights are sensibly initialized by ensuring that the variance of the input and output signals passed through the network remain the same.

$$Var(W_i) = \frac{2}{n_{in} + n_{out}} \tag{9}$$

where W_i is the weight matrix in layer i and n_{in} and n_{out} are the number of neurons feeding in and out.

1.2. ReLU

Using non-linear coding functions $h(\cdot)$ and $g(\cdot)$ enables a dA to discover intricate non-linear structures from the input data. It has become common practice to replace the *sigomid* function with a ReLu in the encoding part. The activation functions in dA are then:

$$h(x) = \quad \text{ReLU}(x) = \quad \max(0, x) \tag{10}$$

ReLU outperforms other non-linear transformation functions on a majority of ML tasks [67]. Setting half of the outputs to zero, it creates robust and sparse representations, which is beneficial for learning algorithms [68]. In addition, ReLU does not require any exponential computation, which substantially accelerates learning. Moreover, its derivative is a step function that provides the network with more non-linearities. These become paramount if stacking a multitude of dAs to build a DNN [69]. For example, ReLU does not suffer from the gradient explosion/vanishing problem [26].

1.3. Batch normalization

During DNN training, the distribution of each layer's inputs changes with updates of the parameters of the preceding layers. With greater depth, small changes to network parameters are amplified; thus, the layers need to keep adapting to the new distribution. Enforcing lower learning rates, this problem decelerates the training process. A batch normalization layer fixes the means and variances of layer outputs according to 11 [70]. It whitens each feature independently after it passes through an activation function in the hidden layer. Moreover, instead of using the whole training sample, the mean and variance in the whitening process are estimated in a batch-wise manner so that the training of layer parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ can be integrated into the original back-propagation algorithm

$$\hat{\mathbf{x}} = \boldsymbol{\gamma} \cdot \frac{\mathbf{x} - E[\mathbf{x}]}{\sqrt{\text{Var}[\mathbf{x}] + \epsilon}} + \boldsymbol{\beta} \tag{11}$$

1.4. Stochastic gradient descent

SGD has been shown to be an effective tool to train the DNN. It is an extension of ordinary gradient descent. In ordinary gradient descent, the model parameters θ are repeatedly updated by taking small steps downward on an error surface that is defined by an objective function $E(\theta)$; here it is the cost function $L_{(\theta,\mathbf{x})}$ as Equation 3 or 6). At each iteration e, the step size is set by the learning rate ε, and the direction of each step equals the back-propagated gradient of the objective function over the N entire training set:

$$\nabla E(\boldsymbol{\theta}, \mathbf{x}) = \frac{\partial L(\boldsymbol{\theta}, \mathbf{x})}{\partial \boldsymbol{\theta}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L(\boldsymbol{\theta}, x_i)}{\partial \boldsymbol{\theta}} \tag{12}$$

$$\begin{aligned} \boldsymbol{\theta}_e &= \boldsymbol{\theta}_{e-1} - \varepsilon \nabla E(\boldsymbol{\theta}_{e-1}, \mathbf{x}) \\ &= \boldsymbol{\theta}_{e-1} - \frac{\varepsilon}{N} \sum_{i=1}^{N} \frac{\partial L(\boldsymbol{\theta}_{e-1}, x_i)}{\partial \boldsymbol{\theta}_{e-1}} \end{aligned} \tag{13}$$

SGD works identically to ordinary gradient descent, except that in each iteration it only uses a "batch" (a subset m of N) of training samples in computing the gradient. The training samples are divided into multiple batches in advance. SGD iterates through different batches and updates the parameters until the value of the cost function stops decreasing (hitting the optimum).

$$\nabla E(\boldsymbol{\theta}, \mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(\boldsymbol{\theta}, x_i)}{\partial \boldsymbol{\theta}} \tag{14}$$

SGD speeds up the convergence because in every iteration when updating the values of the parameters, it is not necessary to run through the complete training set in order to update the parameters. It is "stochastic" because using batch approximations introduces noise in estimating the true gradient: the gradient over all training samples. Although such estimation is biased, it leads the DNN to skip over poor local minima that traditional GD would fall into. Meanwhile, it reduces variance of the gradients [71], which helps prevent the DNN from overfitting [72]. Most importantly, SGD makes better use of the memory allocation in computers since the training data in large scale ML task are usually too large to be fitted into the local memory.

1.5. Momentum

Momentum [73] has become a common trick in achieving the state-of-the-art performance. If the objective function has the pattern of a long shallow ravine leading to the optimum with steep walls on the sides (a deep "U" shape with optimum at the bottom), the SGD tends to oscillate across the optimum since the gradient will point down to steep sides rather than along the way direct towards the optimum. Deep architectures are shown to have similar patterns near the local minima [74], therefore, ordinary SGD can lead to slow convergence; particularly after the initial steep gains. The momentum technique is one way to pull the objective function along the shallow ravine. It speeds up the convergence, and reduces the risk of oscillating by incorporating the gradient information from previous steps. Mathematically, the momentum term $\mu \in (0, 1]$ determines to what extent previous gradients are combined into the current update. As a rule of thumb, μ is set to 0.5 in first epochs. After the training stabilises, it is increased to 0.9 or higher in later iterations. In epoch e, the model parameters θ are updated as Equation (15).

$$\begin{aligned} \boldsymbol{\theta}_e &= \boldsymbol{\theta}_{e-1} - \Delta \boldsymbol{\theta}_e \\ \Delta \boldsymbol{\theta}_e &= \mu_e \Delta \boldsymbol{\theta}_{e-1} + (1 - \mu_e) \varepsilon_e \nabla E(\boldsymbol{\theta}_{e-1}) \end{aligned} \tag{15}$$

1.6. Decaying learning rate

SGD is sensitive to the learning rate since it indicates how big a step should be taken in each update. The intuition in choosing the learning rate is that we should decrease it as the number of updates increases, otherwise the training process will just oscillate near the local minima. A naive implementation is that we decay the learning rate by a certain percent after each epoch (an epoch is one run that SGD iterates over all the batches of training data). There are other advanced methods, such as Adagrad [75] and Adadelta [76], that can dynamically set a learning rate.

1.7. Early stopping

Improving the DNN's fitness to the training data often comes at the cost of increased generalisation error; i.e the overfitting problem. In order to prevent overfitting, we stop the SGD procedures earlier, before the cost function converges to the true best minimum. Early stopping technique [77] offers guidance as to how many updates are allowed before the learner starts to overfit. Such guidance is based on measuring the model's real-time performance on a pre-allocated validation set. If the performance on the validation set stops improving for a long time and it exceeds the limit of patience (a self-defined parameter), then the training process will be stopped because it has probably met the local minima.

1.8. Implementation Details

We have implemented the simple benchmark classifiers (i.e., logistic regression, Naive bayes, decision tree) with the help of the *scikit-learn* library [78] in Python. The *Scikit-learn* library offers the option of weighted class samples. To take the unbalanced class issue into account, we tried different weight values, ranging from 1 to 15, and selected the one with the best performance for each algorithm. Other hyper-parameters in the library API were set at default, other than those specifically mentioned in Section 2 in the online Appendix. For more advanced ML algorithms (i.e., SVM, ensemble methods), we have employed a *big data* framework: Hadoop + Spark. ML algorithms with iterative calculations often prohibit researchers from performing large scale data analysis. For example, in the work of comparing ML algorithms in the problem of mortgage default prediction, even with $300,000$ training samples, SVM is regarded as computationally infeasible due to its $\mathrm{O}(N^3)$ complexity [79]. By using a special data abstraction called Resilient Distributed Dataset (RDD) and caching the RDD into memory, Spark offsets the weakness of low efficiency on running iterative jobs for a traditional Hadoop framework [80]. This makes machine learning on big data possible. We deployed the Hadoop + Spark architecture on Amazon EC2. Nineteen $m1.xlarge$ salves each with 4 cores and $12.6gb$ memory were used.

1.9. GPU Implementation

DNN has a massively parallel structure. Training DNNs heavily depends on matrix calculations which can be computed simultaneously. This makes the training task perfectly suitable for speeding up by graphics processing units (GPUs). A GPU has thousands of cores, and can thus support large scale of parallelisations. In the task of training DNNs, it can offer 20 times faster speeds compared to the CPUs [1]. With the aid of GPU, we were able to train huge DNNs (e.g. millions of parameters) in a timely manner.

Algorithm 1 Pseudo code for training DNN

Input: Training data set $(\mathbf{X}_{train}, \mathbf{Y}_{train})$, N samples with P features; Validation data set $(\mathbf{X}_{valid}, \mathbf{Y}_{valid})$, N' samples with P features. $\mathbf{X}_{train}$ and $\mathbf{X}_{valid}$ are normalised before input.

Input: Number of hidden layers: I, the i-th hidden layer with h^i neurons; Corruption rate in the i-th layer of dA: q_i; Activation functions in encoders or decoders: $h(\cdot), g(\cdot)$; Cost function in layer-wise pre-training or fine-tuning: $L_p(\cdot), L_f(\cdot)$

Input: Learning rate of pre-training or fine-tuning in the e-th epoch: $\varepsilon_{p\_e}, \varepsilon_{f\_e}$; Momentum in the e-th epoch: u_e; Batch size in SGD: B; Maximum number of epochs in pre-training or fine-tuning stage: N, M; Dropout rate in the i-th hidden layer: p_i

Output: DNN for later inference, e.g., making predictions on out-of-sample dataset

Xavier's Initialisation: i-th layer encoder / decoder bias and weights $[\mathbf{b}_i, \tilde{\mathbf{b}_i}] = 0$, $[\mathbf{W}_i, \tilde{\mathbf{W}}_i] \sim \text{Gaussian}(0, \frac{2}{h^{i-1} + h^i})$

// Layer-wise unsupervised pre-training via denoising autoencoder:

$\mathbf{x_0} = \mathbf{X}_{train}, \;\; e = 0$ // setting the input and the epoch counter

for $i \in (1, 2..., I)$ **do**

  // encoder's outputs from previous layer is used as the input for the subsquent layer

  $\mathbf{x_i} = h(\mathbf{W}_{i-1} \cdot \mathbf{x}_{i-1} + \mathbf{b}_{i-1})$

  while $e \leq N$ **do**

    for $j \in (1, 2..., \frac{N}{B})$ **do**

      // Corrupting the j-th input batch of data $\mathbf{x}_i^j$ by randomly knocking out samples

      $\hat{\mathbf{x}}_i^j \leftarrow \mathbf{x}_i^j * \textbf{Binomial}\,(n = B, p = q_i)$

      // Computing the reconstruction of $\tilde{\mathbf{x}}_i^j$ through the encoder and decoder:

      $\mathbf{z}_i^j = g(\tilde{\mathbf{W}}_i^e \cdot h(\mathbf{W}_i^e \cdot \hat{\mathbf{x}}_i^j + \boldsymbol{b}_i^e) + \tilde{\boldsymbol{b}_i^e})$

      // Computing the reconstruction error:

      $L_p(\hat{\mathbf{x}}_\mathbf{i}^\mathbf{j}, \mathbf{z}_\mathbf{i}^\mathbf{j} \mid \boldsymbol{\Theta}) = L_p(\hat{\mathbf{x}}_\mathbf{i}^\mathbf{j}, \mathbf{z}_\mathbf{i}^\mathbf{j} \mid \mathbf{W}_i^e, \mathbf{b}_i^e, \tilde{\mathbf{W}}_i^e, \tilde{\boldsymbol{b}_i^e})$

      // Computing the average gradient among a batch and update the parameters:

      $$\mathbf{W}_i^{e+1} \leftarrow \mathbf{W}_i^e - \frac{\varepsilon_{p\_e}}{B} \sum_{k=1}^{B} \frac{\partial L_p(\hat{x}_i^{j\_k}, z_i^{j\_k} | \boldsymbol{\Theta})}{\partial \mathbf{W}_i^e}, \mathbf{b}_i^{e+1} \leftarrow \mathbf{b}_i^e - \frac{\varepsilon_{p\_e}}{B} \sum_{k=1}^{B} \frac{\partial L_p(\hat{x}_i^{j\_k}, z_i^{j\_k} | \boldsymbol{\Theta})}{\partial \mathbf{b}_i^e}$$

      $$\tilde{\mathbf{W}}_i^{e+1} \leftarrow \tilde{\mathbf{W}}_i^e - \frac{\varepsilon_{p\_e}}{B} \sum_{k=1}^{B} \frac{\partial L_p(\hat{x}_i^{j\_k}, z_i^{j\_k} | \boldsymbol{\Theta})}{\partial \tilde{\mathbf{W}}_i^e}, \tilde{\mathbf{b}}_i^{e+1} \leftarrow \tilde{\mathbf{b}}_i^e - \frac{\varepsilon_{p\_e}}{B} \sum_{k=1}^{B} \frac{\partial L_p(\hat{x}_i^{j\_k}, z_i^{j\_k} | \boldsymbol{\Theta})}{\partial \tilde{\mathbf{b}}_i^e}$$

    end for

    if $L_p(\mathbf{X}_{valid} \mid \boldsymbol{\Theta})$ meets the early stopping condition **then**

      Save the weights and bias of the encoder in each layer

      break

    end if

    e += 1

  end while

end for

// Supervised fine-tunning the whole network:

Initialise parameters in the batch normalisation layer: $\boldsymbol{\gamma}_i \sim \text{Uniform}(-h^i, h^i), \boldsymbol{\beta}_i = 0$

Xavier's Initialisation: softmax layer weight $\hat{\mathbf{W}} \sim \text{Gaussian}(0, \frac{2}{h^i + 2})$, bias $\hat{\mathbf{b}} = 0$

$\mathbf{x_0} = \mathbf{X}_{train}, \;\; e = 0$ // resetting the input and the epoch counter

while $e \leq M$ **do**

  for $j \in (1, 2..., \frac{N}{B})$ **do**

    // Feed forward to compute the outputs for each batch of training data:

    for $i \in (1, 2..., I)$ **do**

      $\mathbf{x}_i^j = h(\mathbf{W}_i^e \cdot \mathbf{x}_{i-1}^j + \mathbf{b}_i^e)$ // inherit the weights and bias from pretrained encoder

      $\hat{\mathbf{x}}_i^j = \boldsymbol{\gamma}_i^e \cdot \frac{\mathbf{x}_i^j - E[\mathbf{x}_i^j]}{\sqrt{\text{Var}[\mathbf{x}_i^j] + \epsilon}} + \boldsymbol{\beta}_i^e$ // batch normalisation layer

      // Drop out the neurons with their corresponding weights and outputs:

      $\hat{\mathbf{x}}_i^j \leftarrow \hat{\mathbf{x}}_i^j * \textbf{Binomial}\,(n = h^i, p = p_i)$

    end for

    $\mathbf{Y}_{predict}^j = \text{softmax}(\hat{\mathbf{W}}^e \cdot \hat{\mathbf{x}}_I^j + \hat{\mathbf{b}^e})$ // softmax layer outputs the predictions

    // Computing the cost function:

    $L_f(\mathbf{Y}_{predict}^j, \mathbf{Y}_{train}^j \mid \boldsymbol{\Theta}) = L_f(\mathbf{Y}_{predict}^j, \mathbf{Y}_{train}^j \mid \mathbf{W}_{1\sim I}^e, \mathbf{b}_{1\sim I}^e, \hat{\mathbf{W}}^e, \hat{\boldsymbol{b}^e}, \boldsymbol{\gamma}_{1\sim I}^e, \boldsymbol{\beta}_{1\sim I}^e)$

    // Computing the gradient and update the parameters of the softmax layer

    $\boldsymbol{\theta} : \{\hat{\mathbf{W}}, \hat{\mathbf{b}}\}$ // parameters to update

    $\Delta\boldsymbol{\theta}^{e+1} = \mu_e \Delta\boldsymbol{\theta}^e + (1 - \mu_e)\frac{\varepsilon_{f\_e}}{B} \sum_{k=1}^{B} \frac{\partial L_f(\mathbf{Y}_{predict}^{j\_k}, \mathbf{Y}_{train}^{j\_k} | \boldsymbol{\Theta})}{\partial \boldsymbol{\theta}^e}$ // with momentum

    $\boldsymbol{\theta}^{e+1} \leftarrow \boldsymbol{\theta}^e - \Delta\boldsymbol{\theta}^{e+1}$

    // Propagating back gradients and update intermediate layers

    for $i' \in (l, l-1, ..., 1)$ **do**

      $\boldsymbol{\theta} : \{\mathbf{W}_{i'}, \mathbf{b}_{i'}, \boldsymbol{\gamma}_{i'}, \boldsymbol{\beta}_{i'}\}$ // parameters of hidden and batch normalisation layers

      // Applying chain rules

      $\Delta\boldsymbol{\theta}^{e+1} = \mu_e \Delta\boldsymbol{\theta}^e + (1-\mu_e)\frac{\varepsilon_{f\_e}}{B} \sum_{k=1}^{B} \left\{ \frac{\partial L_f(\mathbf{Y}_{predict}^{j\_k}, \mathbf{Y}_{train}^{j\_k} | \boldsymbol{\Theta})}{\partial \hat{\mathbf{x}}_l^j} \cdot \prod_{i^*=l}^{i'+1} \left\{ \frac{\partial \hat{\mathbf{x}}_{i^*}^j}{\hat{\mathbf{x}}_{i^*-1}^j} \right\} \cdot \frac{\partial \hat{\mathbf{x}}_{i'}^j}{\partial \boldsymbol{\theta}^e} \right\} \boldsymbol{\theta}^{e+1} \leftarrow \boldsymbol{\theta}^e - \Delta\boldsymbol{\theta}^{e+1}$

    end for

  end for

  if $L_p(\mathbf{Y}_{predict}^j, \mathbf{Y}_{train}^j \mid \boldsymbol{\Theta})$ meets the early stopping condition **then**

    Save the DNN

    break

  end if

  e += 1

end while

2. Hyper-Parameter Tuning for the DNN and Benchmark ML Classifiers

The (predictive) performance of a learning algorithm depends on the setting of algorithmic hyper-parameters. Tuning hyper-parameters in a model selection stage is important to ensure that an algorithm performs well on a given data set [46]. We tune the proposed DNN and ML benchmark classifiers in such a way that we reserve a fraction of 20% of the training set as a validation data to assess candidate models with different hyper-parameter settings. We then use the model with the best hyper-parameter configuration in terms of validation set performance to generate risk predictions for the trades in the test set. Given that we employ n-fold cross-validation, the model training and evaluation occurs n times. In theory, this suggests that the selection of suitable hyper-parameters should also be undertaken n times; once for each loop of cross-validation. Given the computational effort associated with training advanced learning algorithms on a large data set and the number of alternative hyper-parameter settings, repeating model selection n times is computationally infeasible. Therefore, we perform model selection only once in the first iteration of cross-validation. In subsequent iterations, we retrain the learning algorithms using the hyper-parameter specification identified as most suitable in the first cross-validation iteration.

We acknowledge that our model selection approach suffers the limitation that it uses only a relatively small amount of data to tune hyper-parameters. More specifically, when setting $n = 10$ in cross-validation, training sets comprise roughly 90% of the available data out of which 20% are used as validation set. Therefore, the single validation partition on which we assess the predictive performance of candidate hyper-parameter settings is roughly 18% of the full data set. While computational considerations render model selection in every round of cross-validation infeasible, we suggest that our approach is also suitable. Our motivation for this view is twofold. First, 18% of the full data set are still a sizeable amount of data in absolute terms, as we work with a large data set including about 30 million trades. Second, the DNN classifier exhibits the largest number of different hyper-parameters and hyper-parameter candidate settings in the comparison. DNNs are also considered sensitive with respect to hyper-parameter choices, which suggest that model selection is particularly important for DNNs. Consequently, reducing the amount of hyper-parameter tuning in our approach compared to a full model selection in every cross-validation iteration provides a conservative evaluation of the ability of the DNN. If the available resources facilitates a more comprehensive tuning of hyper-parameters, it is plausible to expect the DNN to benefit the most from such additional tuning.

The range of candidate settings that we consider for each learning algorithm is based on previous literature, while accounting for both the large size of the data set and computational feasibility. More specifically, we draw inspiration from previous classifier comparisons [46, 81, 79, 7] to identify candidate settings for the ML benchmarks. For the DNN, we follow the recommendations of [55] and consider the candidate hyper-parameter settings he proposes. We also follow the advice of [55] to not tune DNN hyper-parameters using grid-search, which would involve a full-enumerative search across all combinations of hyper-parameter candidate settings that is computationally intractable, but to use random search. Other than using random search, the tuning process for the DNN is the same as that for the ML benchmarks.

Table 5 and Table 6 report the candidate hyper-parameter settings that we consider during the tuning of the DNN and the ML benchmarks, respectively.

Table 5: Candidate Hyper-Parameter Settings for the Proposed DNN

Attribution	Hyper-parameter	Range Selected
DNN Topology	Number of hidden layers (dAs)	[2, 3, 4, 5, 6]
	Number of hidden units in each hidden layer[a]	[32, 64, 128, 256, 512, 1024, 2048]
	Weight decay regularizer λ in dA	[10^{-2}, 10^{-3}, 10^{-4}, 10^{-2}, 10^{-3}, 10^{-5}]
	Corruption rate in each hidden layer[b]	[0.2, 0.3, 0.4, 0.5]
SGD Training	Learning rate in pre-training	[1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}]
	Learning rate in fine-tuning	[1.5, 1, 0.5, 0.1, 0.05, 0.01]
	Learning rate decay[c]	[0.99, 0.995, 0.999]
	Number of samples in minibatch	[10, 20, 30, 50, 100, 150, 200]
	Momentum interval[d]	[200, 500, 800]
	Momentum start[d]	0.5
	Momentum end[d]	[0.9, 0.99]
	Number of epochs in pre-training	[30, 50, 70]
	Maximum number of epochs in fine-tuning[e]	[500, 1000, 1500]
Dropout	Dropout rate in each layer[f]	0.5

Notes: **a**) The number of neurons in the middle layers is selected larger than the the number of hidden units in the first and top layers, e.g., 4 hidden layers: [64, 256, 512, 32]. **b**) The corruption rate is set in an increasing way, e.g., 4 hidden layers: [0.2, 0.3, 0.4, 0.5]. **c**) In the pre-training stage, the learning rate is fixed. In the fine-tuning stage, the learning rate decays by a given percent in each epoch. **d**) For epoch $e \in [0, \textit{momentum interval}]$, the momentum μ_e increases linearly from *momentum start* to *momentum end*. After that, it stays at *momentum end*. **e**) In the fine-tuning stage, we also use early stopping. Be aware that the training process can stop before the current epoch reaches the maximum number. **f**) The dropout rate is the same for all dropout layers.

Table 6: Candidate Hyper-Parameter Settings for ML Benchmark Classifiers

Algorithm	**Hyper-Parameter**	**Candidate Settings**
Logistic Regression	form of regularization	none, L2, L1, forward selection
	regularizer	$\{ 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3} \}$
Artificial Neural Network	number of hidden units	2, 8, 31, 64, 128
Random Forest	number of trees	32, 64, 128, 256, 512, 1024, 2048
	max. depth	1, 2, 4, 8, 20
	random subspace	2, 5, 10, 15, 31
Adaptive boosting	number of trees	32, 64, 128, 256, 512, 1024, 2048
	max. depth	1, 2, 4, 8

3. Comparison of DNN to Benchmark ML Classifiers After Addressing Class Imbalance Using SMOTE

Table 7 reports empirical results concerning the comparison of the DNN to the ML-based benchmark classifiers after addressing class imbalance using SMOTE. We estimate predictive performance in terms of the P&L resulting from hedging trades according to model predictions, the AUC and the F1-measure. The results indicate that all classifiers benefit from SMOTE. Compared to Table 3 in the main part of the paper, values of P&L and predictive performance are consistently higher. This confirms that the skewed distributions of A-book and B-book clients in the focal data set has adversely affected all classification models.

Table 7: DNN Performance vs. Benchmarks After Addressing Class Skew Using SMOTE

folds	1	2	3	4	5	mean
Average P&L per trader in GBP						
DNN	3.944,7	4.169,1	3.937,7	3.745,1	3.791,9	3.917,7
Logit	3.721,4	3.612,7	3.696,1	4.047,3	3.453,9	3.706,3
ANN	3.242,6	3.077,5	3.514,7	3.142,9	3.170,1	3.229,6
RF	3.363,6	2.911,5	3.578,1	3.110,7	3.264,0	3.245,6
AdaBoost	3.651,9	3.381,9	3.553,6	3.762,8	3.588,0	3.587,6
Area under the Receiver-Operating-Characteristics Curve (AUC)						
DNN	0,902	0,903	0,875	0,879	0,882	0,889
Logit	0,799	0,811	0,811	0,802	0,805	0,806
ANN	0,858	0,861	0,859	0,843	0,869	0,858
RF	0,798	0,787	0,799	0,780	0,796	0,792
AdaBoost	0,809	0,798	0,807	0,794	0,808	0,803

4. Auxiliary Empirical Results from Preliminary Tests

The following subsections augment the empirical results presented in the main part of the paper. While results in the main paper are based on cross-validation to ensure robustness, several preliminary experiments were undertaken using a simpler, computationally less demanding split-sampling approach. This approach involved partitioning the data sequentially into a training set (70%) for developing predictive models and a test set (30%) for assessing their accuracy. Given reduced computational requirements, the preliminary experiments could also involve a larger number of ML algorithms and considered support vector machines (SVMs), C5.0 decision trees, and the Naive Bayes classifier (NB) as additional benchmarks to the DNN.

Trades from November 2003 to April 2013 entered the training set, whereas trades from May 2013 to July 2014 served as the hold-out test set. According to the definition of the prediction target, 6.1% (7.2%) of the trades in the training (test) set came from A-book clients and should be hedged. To address

the class imbalance in preliminary experiments, we bootstrapped the test set with different ratios of class '+1' trades ranging from 0.05, 0.1, ... 0.5, and drew 1000 bootstrap samples for each ratio. This approach enabled examining the performance of the DNN across different scenarios with varying degrees of class skew and increased robustness because it implied repeating the out-of-sample evaluation 11000 times on different (bootstrapped) test sets.

4.1. Pre-test results of DNN vs. ML Benchmarks

4.1.1. Aggregated Classifier Performance

Table 8 summarizes the performance of the DNN and benchmark ML classifiers across bootstrap samples with different ratios of high risk traders. The percentage figures in the header of Table 8 give the ratio of A-book clients per bootstrap. For example, if the ratio is 0.3, the bootstrap sample on which we compare ML models includes 30% A-book clients. Their trades represent the positive class (with $y_i = 1$) against which the market maker should hedge. This implies that the ratios shown in Table 8 do not represent the prior probability of the positive class in the data, which depends on the total number of *trades* across all high risk traders.

The setting *default* represents the true ratio of high risk traders in the test set, which is 7.2%. For each ratio, Table 8 reports the average performance of a model over 1000 bootstrapped test sets. An underscore highlights the best model per performance indicator. We assess models in terms of the P&L that emerges from STX taking hedging decisions according to model recommendations. In addition, we consider the AUC and the F-measure. To check whether performance differences between classifiers are significant, we apply the nonparametric Friedman test together with multiple pairwise comparisons [82]. The last column of Table 8 reports the p-values of the pairwise comparisons (after adjustment using Holm's procedure), where the DNN is the control classifier. A low p-value indicates that DNN performs significantly better than the corresponding benchmark classifier [82].

Table 8: Performance Comparison of the DNN to Benchmark ML Classifiers

Metrics	*Classifiers*	Bootstrap with different percentage of high risk clients											Friedman χ^2_7	*Holm's* p-value
		0.05	default	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50		
P&L (million)	Deep learning	22.50	18.30	14.92	7.43	-0.25	-7.84	-16.07	-23.28	-30.85	-38.70	-45.83	60.5 **(.0000)**	**(0.0208)**
	SVM	21.32	17.39	13.97	5.02	-3.52	-11.82	-20.93	-29.63	-38.37	-47.63	-56.67		**(.0000)**
	ANN	21.55	16.11	11.28	1.52	-8.17	-18.25	-28.58	-38.39	-47.95	-58.16	-68.2		**(.0000)**
	Logistic	21.51	15.97	11.64	1.95	-7.89	-17.43	-28.02	-37.49	-47.42	-57.26	-67.28		**(.0000)**
	C5.0 Tree	21.36	16	11.43	2.08	-7.45	-16.75	-27.01	−36.26	-45.72	-55.6	-64.61		**(.0001)**
	RF	20.68	16.32	11.88	3.40	-5.22	-8.44	-19.53	-28.65	-37.44	-49.08	-57.44		**(.0006)**
	AdaBoost	21.97	17.1	13.07	4.4	-4.21	-12.91	-22.36	-30.8	-39.55	-48.53	-56.95		**(.0000)**
	Naive bayes	17.53	12.2	10.29	1.83	-9.69	-18.7	-28.47	-37.27	-46.04	-55.46	-64.08		
AUC	Deep learning	0.813	0.812	0.812	0.812	0.812	0.812	0.812	0.812	0.812	0.812	0.812	80.1 **(.0000)**	**(.0000)**
	SVM	0.679	0.679	0.68	0.68	0.68	0.68	0.68	0.68	0.68	0.68	0.681		**(.0000)**
	ANN	0.791	0.791	0.791	0.791	0.791	0.791	0.791	0.792	0.791	0.791	0.791		**(.0000)**
	Logistic	0.489	0.489	0.489	0.489	0.489	0.489	0.489	0.489	0.489	0.489	0.489		**(.0000)**
	C5.0 Tree	0.744	0.744	0.745	0.745	0.745	0.744	0.744	0.744	0.745	0.745	0.745		**(.0000)**
	RF	0.762	0.762	0.762	0.762	0.762	0.761	0.762	0.762	0.762	0.762	0.762		**(.0000)**
	AdaBoost	0.801	0.801	0.801	0.801	0.801	0.801	0.801	0.801	0.801	0.801	0.801		**(.0000)**
	Naive bayes	0.483	0.482	0.482	0.482	0.482	0.482	0.482	0.482	0.482	0.482	0.482		
F-measure	Deep learning	0.248	0.282	0.298	0.318	0.331	0.338	0.343	0.347	0.35	0.352	0.354	75.5 **(.0000)**	**(.0000)**
	SVM	0.19	0.24	0.265	0.296	0.307	0.307	0.303	0.291	0.281	0.27	0.26		**(.0000)**
	ANN	0.024	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.025		**(.0000)**
	Logistic	0.016	0.016	0.017	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.016		**(.0000)**
	C5.0 Tree	0.064	0.065	0.066	0.066	0.067	0.067	0.067	0.067	0.067	0.067	0.067		**(.0000)**
	Random forest	0.169	0.239	0.204	0.258	0.310	0.322	0.320	0.307	0.309	0.304	0.341		**(.0000)**
	Adaptive boosting	0.163	0.172	0.176	0.182	0.184	0.186	0.187	0.188	0.188	0.189	0.189		**(.0000)**
	Naive bayes	0.045	0.061	0.071	0.088	0.1	0.108	0.115	0.121	0.125	0.129	0.132		

Table 8 reveals that the DNN outperforms the benchmarks by a substantial margin. Its superiority is consistent across different performance indicators and ratios of high risk traders. In each of the $4 * 11 = 44$ settings, the DNN gives the largest profit and the most accurate class predictions. Similarly, in experimental settings where the market maker loses money due to a synthetically increased amount of high risk traders through bootstrapping (e.g., 20% or more), the DNN helps to mitigate the loss to a large extent. Formally, the observed results facilitate rejecting the null hypothesis of the Friedman test for each performance measure. In the following post-hoc comparisons between the DNN and benchmarks, we can also reject the null hypothesis of equal performance and conclude that DNN performs significantly better than each benchmark.

4.1.2. Distribution of Classifier Performance

Previous results depict the average performance of different classification models across different ratios of A-book clients and performance indicators. We compute these averages across 1,000 bootstrap samples per A-book client ratio. The boxplots provided below further extend the results of Table 8 through reporting the distribution of model performance across the 1,000 bootstrap samples.

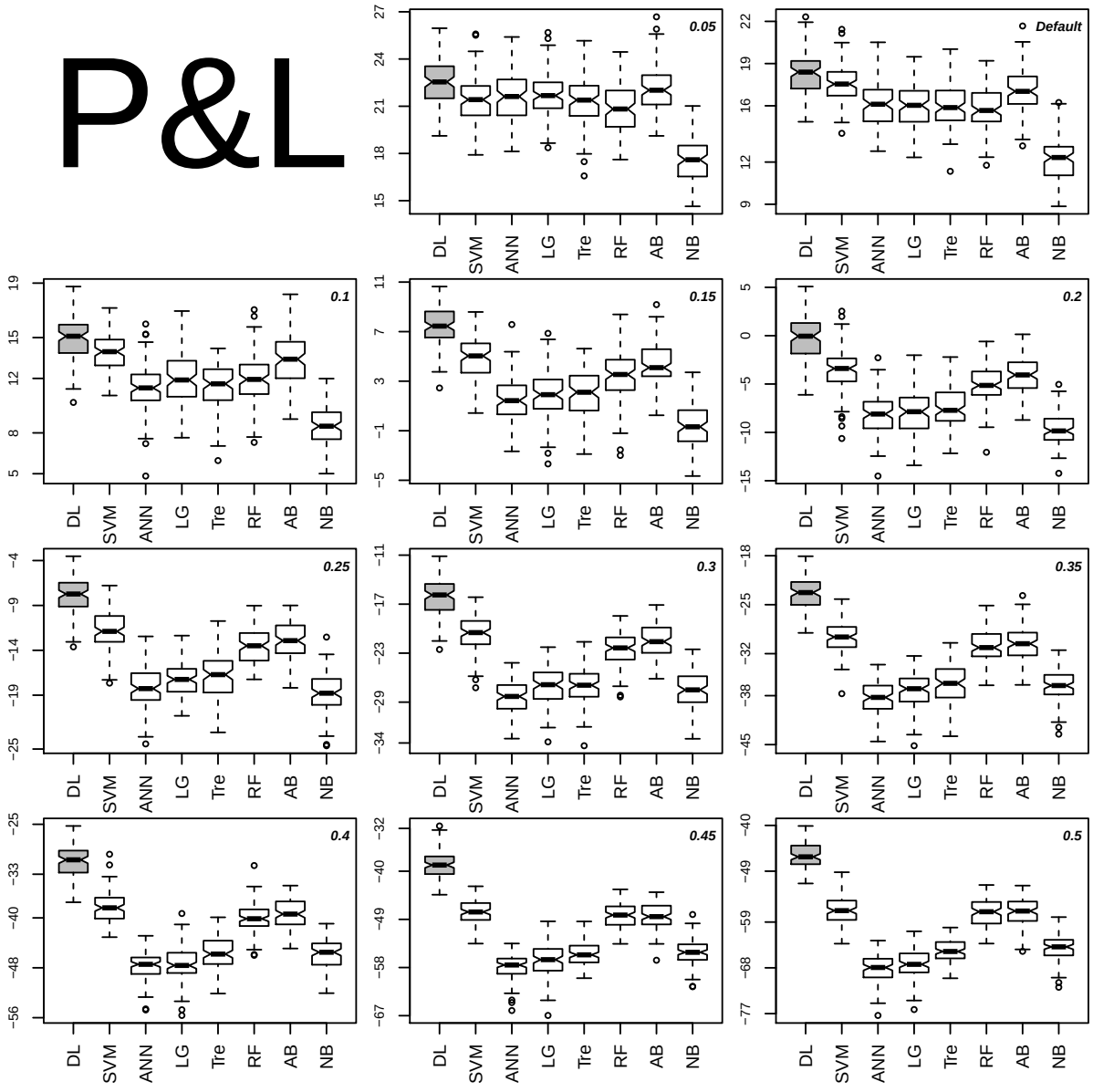
P&L
Default
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
DL
SVM
ANN
LG
Tre
RF
AB
NB

Figure 12

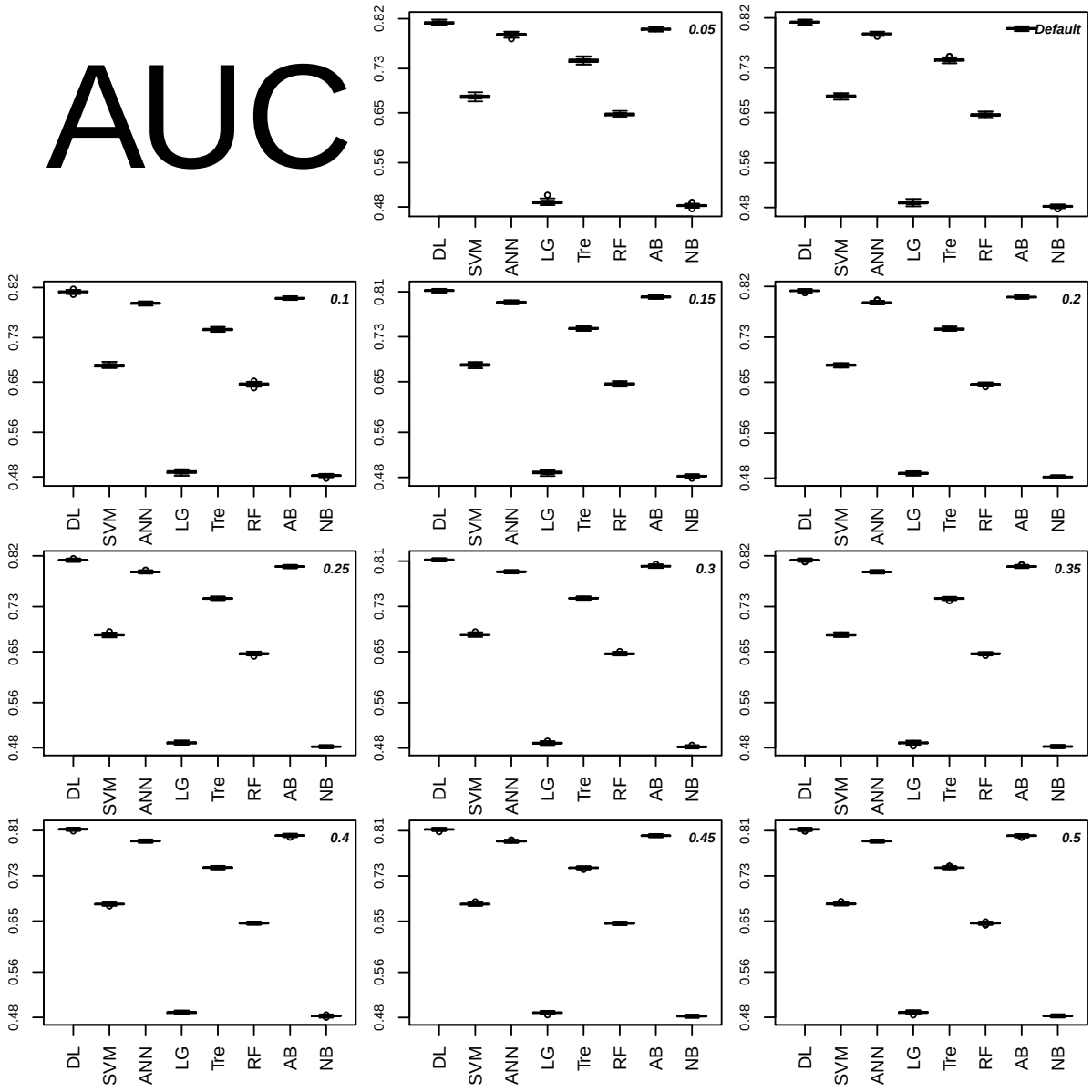
AUC
Default
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
DL
SVM
ANN
LG
Tre
RF
AB
NB
0.48
0.56
0.65
0.73
0.81
0.82

Figure 13

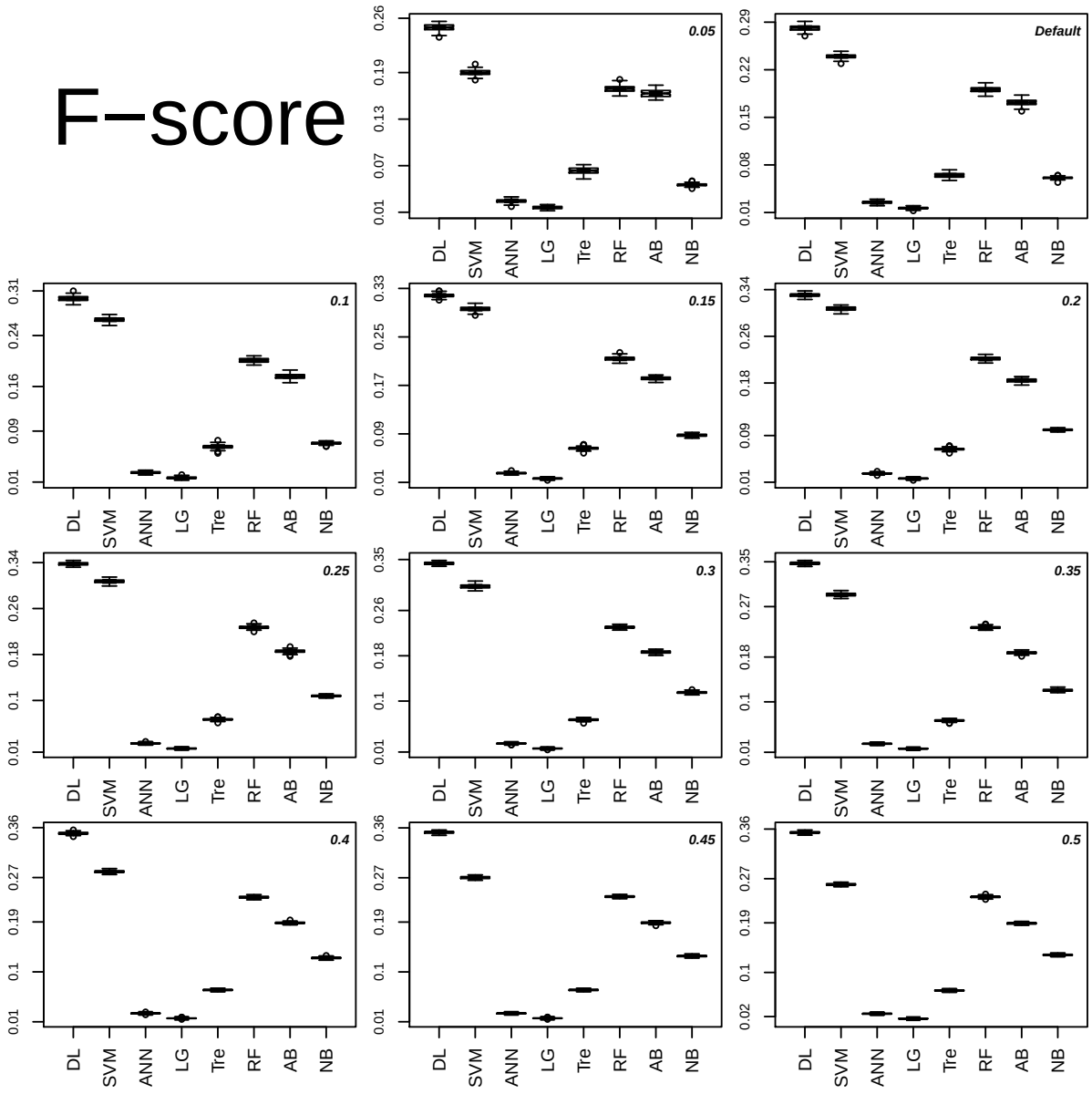

Figure 14

4.1.3. Comparison of the DNN to other DL Models

We explain in the main part of the paper that the ability to learn high level distributed representations from input data is not specific to the DNN we propose here. Prior literature credits the whole family of DL methods for this feature [1]. Therefore, we experiment with other popular DL methods and compare their performance in trader risk behavior forecasting to that of the proposed DNN. Corresponding results shed light on the effectiveness of the proposed DNN relative to other DL benchmarks and contribute additional insight to what extent unsupervised pre-training as well as other architectural choices we have made contribute to the performance of our DNN.

Considering encouraging results in the area of mortgage default prediction [24], the first DL benchmark we consider consists of a deep feed-forward network (DFNN) with more than one hidden layer.

We also consider a convolutional neural network (CNN). While prior work has, to our best knowledge, not considered CNNs for risk analytics, CNNs have shown excellent results in other domains [5], which indicates that they represent a useful benchmark. Finally, using the sequence of trades per trader as a time-ordered input, we compare the proposed DNN to a recurrent neural network with long short-term memory cells (LSTM) [18]. Using the same performance indicators and statistical tests as in the comparison to ML benchmarks (Table 3 in the main paper), we report the results of the three DL methods together with those of our DNN in Table 9.

The overall conclusion from Table 9 is that the other DL benchmarks do not perform as well as the proposed model. Our DNN consistently achieves the best performance across evaluation criteria and class ratios. Therefore, Table 9 supports the proposed DNN and its underlying topological choices (see Figure 3 in the main paper). In particular, none of the three DL benchmarks employs unsupervised pre-training. Therefore, the superior performance of the proposed DNN may be taken as evidence for the suitability of unsupervised pre-training.

However, we caution against over-emphasizing results of Table 9. DL methods are complex and require careful tuning to unfold their full potential. This paper focuses on one particular type of DNN and its potential to support decision-making in risk management. Performing a fully-comprehensive benchmark of several alternative complex DL models is beyond the scope of the paper. Accordingly, we do not claim superiority of the proposed DNN to deep, feed-forward networks, CNNs and other DL methods in general, and acknowledge that more elaborate tuning of corresponding approaches may give performance comparable to the DNN we employ.

Table 9: Comparison of Proposed DNN Against Other Deep Learning Benchmarks

Metrics	*Classifiers*	Bootstrap with different percentage of high risk clients											Friedman χ^2_7	*Holm's* p-value
		0.05	default	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50		
P&L (million)	Proposed DNN	22.50	18.30	14.92	7.43	-0.25	-7.84	-16.07	-23.28	-30.85	-38.70	-45.83	56.5 **(.0000)**	**(.0000)**
	DFNN	19.54	16.43	12.53	5.42	-2.34	-8.90	-18.43	-25.65	-32.54	-40.55	-52.21		**(.0000)**
	CNN	20.53	14.43	12.43	5.01	-4.53	-9.42	-19.43	-30.43	-35.62	-47.33	-54.45		**(.0000)**
	LSTM	21.09	15.01	14.2	3.54	-5.42	-9.21	-17.54	-26.75	-34.53	-45.76	-50.54		
AUC	Proposed DNN	0.813	0.812	0.812	0.812	0.812	0.812	0.812	0.812	0.812	0.812	0.812	78.3 **(.0000)**	**(.0000)**
	FDNN	0.704	0.704	0.704	0.604	0.703	0.704	0.704	0.704	0.705	0.704	0.704		**(.0000)**
	CNN	0.793	0.793	0.793	0.793	0.791	0.793	0.793	0.793	0.793	0.793	0.793		**(.0000)**
	LSTM	0.745	0.745	0.745	0.745	0.746	0.745	0.743	0.744	0.745	0.745	0.745		
F-Score	Proposed DNN	0.248	0.282	0.298	0.318	0.331	0.338	0.343	0.347	0.35	0.352	0.354	69.5 **(.0000)**	**(.0000)**
	DFNN	0.11	0.204	0.242	0.284	0.302	0.303	0.305	0.340	0.335	0.301	0.330		**(.0000)**
	CNN	0.094	0.032	0.199	0.242	0.303	0.306	0.312	0.302	0.329	0.339	0.302		**(.0000)**
	LSTM	0.081	0.225	0.205	0.209	0.321	0.312	0.309	0.301	0.321	0.305	0.329		

Notes: We tune the hyper-parameters of the three DL benchmarks using grid-search in the same manner as the ML benchmarks. The hyper-parameters and search spaces we consider are as follows. *DFNN*: no. of hidden layers [2, 3, 4], no. of neurons per hidden layer [50; 200]. *CNN* no. of convolutional layers [2, 3, 4], filter size 3, max. pooling size = 2. *LSTM* no. of hidden layer [2, 3]. FDNN and CNNs use activation functions of type ReLu in the hidden layers.

5. Robustness of the DNN With Respect to Random Weight Initialization

We train the DNN using stochastic gradient descent. Starting the minimization of the loss function from randomly initialized weights using Xavier initialization (see above), gradient descent will deliver different solutions depending on the random initial weights. Therefore, the performance of the DNN may vary with random initial weights. To examine the robustness of the DNN with respect to the initial weights, we repeat the initialization ten times, develop a DNN model on the training set, and assess its performance in terms of the AUC on the test set. Table 10 reports corresponding results and suggests that the dependence of the DNN with respect to initial weights is not substantial.

Table 10: Robustness of DNN Performance With Respect to Initial Weights

Seed	AUC
1	0.843
12	0.857
123	0.853
1234	0.845
12345	0.855
123456	0.846
1234567	0.852
12345678	0.850
123456789	0.831
1234567890	0.852
Mean	**0.847**

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[2] R. Geng, I. Bose, X. Chen, Prediction of financial distress: An empirical study of listed chinese companies using data mining, European Journal of Operational Research 241 (1) (2015) 236–247.

[3] Y. Bengio, Learning deep architectures for AI, Foundations and trends® in Machine Learning 2 (1) (2009) 1–127.

[4] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117.

[5] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26.

[6] Z.-Y. Chen, Z.-P. Fan, M. Sun, A multi-kernel support tensor machine for classification with multitype multiway data and an application to cross-selling recommendations, European Journal of Operational Research 255 (1) (2016) 110–120.

[7] W. Verbeke, K. Dejaeger, D. Martens, J. Hur, B. Baesens, New insights into churn prediction in the telecommunication sector: A profit driven data mining approach, European Journal of Operational Research 218 (1) (2012) 211–229.

[8] A. Oztekin, R. Kizilaslan, S. Freund, A. Iseri, A data analytic approach to forecasting daily stock returns in an emerging market, European Journal of Operational Research 253 (3) (2016) 697–710.

[9] P. du Jardin, A two-stage classification technique for bankruptcy prediction, European Journal of Operational Research 254 (1) (2016) 236–252.

[10] G. Paleologo, A. Elisseeff, G. Antonini, Subagging for credit scoring models, European Journal of Operational Research 201 (2) (2010) 490–499.

[11] T. Hastie, R. Tibshirani, J. H. Friedman, The Elements of Statistical Learning, 2nd Edition, Springer, New York, 2009.

[12] J. N. Crook, D. B. Edelman, L. C. Thomas, Recent developments in consumer credit risk assessment, European Journal of Operational Research 183 (3) (2007) 1447–1465.

[13] G. L. Lilien, Bridging the academicpractitioner divide in marketing decision models, Journal of Marketing 75 (4) (2011) 196–210.

[14] C. Hsinchun, R. H. L. Chiang, V. C. Storey, Business intelligence and analytics: From big data to big impact, MIS Quarterly 36 (4) (2012) 1165–1188.

[15] J. B. Heaton, N. G. Polson, J. H. Witte, Deep learning for finance: deep portfolios, Applied Stochastic Models in Business and Industry 33 (1) (2017) 3–12, asmb.2209.

[16] J. Sirignano, Deep learning for limit order books, CoRR abs/1601.01987.
URL `https://arxiv.org/abs/1601.01987`

[17] M. Kraus, S. Feuerriegel, Decision support from financial disclosures with deep neural networks and transfer learning, Decision Support Systems 104 (2017) 38–48.

[18] T. Fischer, C. Krauss, Deep learning with long short-term memory networks for financial market predictions, European Journal of Operational Research 270 (2) (2018) 654–669.

[19] N. Huck, Pairs selection and outranking: An application to the S&P 100 index, European Journal of Operational Research 196 (2) (2009) 819–825.

[20] Y. Deng, F. Bao, Y. Kong, Z. Ren, Q. Dai, Deep direct reinforcement learning for financial signal representation and trading, IEEE Transactions on Neural Networks and Learning Systems 28 (3) (2017) 653–664.

[21] R. Xiong, E. P. Nichols, Y. Shen, Deep learning stock volatility with google domestic trends, CoRR arXiv:1512.04916v3.

[22] S. P. Chatzis, V. Siakoulis, A. Petropoulos, E. Stavroulakis, N. Vlachogiannakis, Forecasting stock market crisis events using deep and statistical machine learning techniques, Expert Systems with Applications 112 (2018) 353–371.

[23] P. M. Addo, D. Guegan, B. Hassani, Credit risk analysis using machine and deep learning, Risks 6 (2) (2018) 1–20.

[24] J. A. Sirignano, A. Sadhwani, K. Giesecke, Deep learning for mortgage risk (2016).
URL `https://people.stanford.edu/giesecke/`

[25] G. F. Montufar, R. Pascanu, K. Cho, Y. Bengio, On the number of linear regions of deep neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 2924–2932.

[26] Y. Bengio, I. Goodfellow, A. Courville, Deep learning, MIT Press, 2016.

[27] C. L. Giles, S. Lawrence, A. C. Tsoi, Noisy time series prediction using recurrent neural networks and grammatical inference, Machine Learning 44 (1) (2001) 161–183.

[28] A. Oztekin, R. Kizilaslan, S. Freund, A. Iseri, A data analytic approach to forecasting daily stock returns in an emerging market, European Journal of Operational Research 253 (3) (2016) 697–710.

[29] F. Shen, J. Chao, J. Zhao, Forecasting exchange rate using deep belief networks and conjugate gradient method, Neurocomputing 167 (2015) 243–253.

[30] M. Dixon, D. Klabjan, J. H. Bang, Classification-based financial markets prediction using deep neural networks, Algorithmic Finance 6 (3-4) (2017) 67–77.

[31] W. Bao, J. Yue, Y. Rao, A deep learning framework for financial time series using stacked autoencoders and long-short term memory, PLOS ONE 12 (7).

[32] C. Krauss, X. A. Do, N. Huck, Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500, European Journal of Operational Research 259 (2) (2017) 689–702.

[33] Y. Zhao, J. Li, L. Yu, A deep learning ensemble approach for crude oil price forecasting, Energy Economics 66 (2017) 9–16.

[34] Y. Baek, H. Y. Kim, Modaugnet: A new forecasting framework for stock market index value with an overfitting prevention lstm module and a prediction lstm module, Expert Systems with Applications 113 (2018) 457–480.

[35] H. Y. Kim, C. H. Won, Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models, Expert Systems with Applications 103 (2018) 25–37.

[36] N. Huck, Large data sets and machine learning: Applications to statistical arbitrage, European Journal of Operational Research 278 (1) (2019) 330–342.

[37] B. Ribeiro, N. Lopes, Deep belief networks for financial prediction, in: B.-L. Lu, L. Zhang, J. Kwok (Eds.), Proccedings of the International Conference on Neural Information Processing (ICONIP'2011), Neural Information Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 766–773.

[38] S. H. Yeh, C. J. Wang, M. F. Tsai, Deep belief networks for predicting corporate defaults, in: Proceedings of the 24th Wireless and Optical Communication Conference (WOCC), IEEE Computer Society, 2015, pp. 159–163.

[39] J. Lee, D. Jang, S. Park, Deep learning-based corporate performance prediction model considering technical capability, Sustainability 9 (6) (2017) 899–911.

[40] C. Luo, D. Wu, D. Wu, A deep learning approach for credit scoring using credit default swaps, Engineering Applications of Artificial Intelligence 65 (2017) 465–470.

[41] J. Jurgovsky, M. Granitzer, K. Ziegler, S. Calabretto, P.-E. Portier, L. He-Guelton, O. Caelen, Sequence classification for credit-card fraud detection, Expert Systems with Applications 100 (2018) 234–245.

[42] C. Brady, R. Ramyar, White paper on spread betting, Lond. Cass Bus. Sch.

[43] N. Huck, Pairs trading and outranking: The multi-step-ahead forecasting case, European Journal of Operational Research 207 (3) (2010) 1702–1716.

[44] M. Pryor, The Financial Spread Betting Handbook 2e: A Guide to Making Money Trading Spread Bets, Harriman House Limited, 2011.

[45] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, T. Poggio, A quantitative theory of immediate visual recognition, Progress in brain research 165 (2007) 33–56.

[46] S. Lessmann, B. Baesens, H.-V. Seow, L. C. Thomas, Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research, European Journal of Operational Research 247 (1) (2015) 124–136.

[47] A. Ula, O. T. Yldz, E. Alpaydn, Eigenclassifiers for combining correlated classifiers, Information Sciences 187 (0) (2012) 109–120.

[48] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, Advances in neural information processing systems 19 (2007) 153.

[49] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks, The Journal of Machine Learning Research 10 (2009) 1–40.

[50] G. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554.

[51] Y. Bengio, O. Delalleau, Justifying and generalizing contrastive divergence, Neural Computation 21 (6) (2009) 1601–1621.

[52] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th International Conference on Machine learning, ACM, 2008, pp. 1096–1103.

[53] A. Krogh, J. A. Hertz, A Simple Weight Decay Can Improve Generalization, Morgan Kaufman, 1992, pp. 950–957.

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

[55] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 437–478.

[56] K. Dowd, Adjusting for risk: An improved Sharpe ratio, International Review of Economics & Finance 9 (3) (2000) 209 – 222.

[57] M. Weber, C. F. Camerer, The disposition effect in securities trading: An experimental analysis, Journal of Economic Behavior & Organization 33 (2) (1998) 167–184.

[58] D. J. Hand, Measuring classifier performance: Acoherent alternative to the area under the roc curve, Machine Learning 77 (1) (2009) 103–123.

[59] A. Beque, K. Coussement, R. Gayler, S. Lessmann, Approaches for credit scorecard calibration: An empirical analysis, Knowledge-Based Systems 134 (15) (2017) 213–227.

[60] J. M. Johnson, T. M. Khoshgoftaar, Survey on deep learning with class imbalance, Journal of Big Data 6 (1) (2019) 27. doi:10.1186/s40537-019-0192-5.
URL https://doi.org/10.1186/s40537-019-0192-5

[61] H. He, E. A. Garcia, Learning from imbalanced data, Knowledge and Data Engineering, IEEE Transactions on 21 (9) (2009) 1263–1284.

[62] A. V. Benos, Aggressiveness and survival of overconfident traders, Journal of Financial Markets 1 (3) (1998) 353 – 383.

[63] A. Oztekin, D. Delen, A. Turkyilmaz, S. Zaim, A machine learning-based usability evaluation method for elearning systems, Decision Support Systems 56 (2013) 63–73.

[64] C. Sevim, A. Oztekin, O. Bali, S. Gumus, E. Guresen, Developing an early warning system to predict currency crises, European Journal of Operational Research 237 (3) (2014) 1095–1104.

[65] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning?, The Journal of Machine Learning Research 11 (2010) 625–660.

[66] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[67] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.

[68] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier networks, in: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume, Vol. 15, 2011, pp. 315–323.

[69] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.

[70] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.

[71] W. A. Gardner, Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique, Signal Processing 6 (2) (1984) 113–133.

[72] O. Bousquet, L. Bottou, The tradeoffs of large scale learning, in: Advances in neural information processing systems, 2008, pp. 161–168.

[73] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 1139–1147.

[74] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, Neural Networks, IEEE Transactions on 5 (2) (1994) 157–166.

[75] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, The Journal of Machine Learning Research 12 (2011) 2121–2159.

[76] M. D. Zeiler, Adadelta: An adaptive learning rate method, arXiv preprint arXiv:1212.5701.

[77] L. Prechelt, Automatic early stopping using cross validation: quantifying the criteria, Neural Networks 11 (4) (1998) 761–767.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[79] T. Fitzpatrick, C. Mues, An empirical comparison of classification algorithms for mortgage default prediction: evidence from a distressed mortgage market, European Journal of Operational Research 249 (2) (2016) 427–439.

[80] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2012, pp. 2–2.

[81] S. Finlay, Multiple classifier architectures and their application to credit risk assessment, European Journal of Operational Research 210 (2) (2011) 368–378.

[82] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Information Sciences 180 (10) (2010) 2044–2064.

IRTG 1792 Discussion Paper Series 2019

For a complete list of Discussion Papers published, please visit http://irtg1792.hu-berlin.de.

001 "Cooling Measures and Housing Wealth: Evidence from Singapore" by Wolfgang Karl Härdle, Rainer Schulz, Taojun Xie, January 2019.

002 "Information Arrival, News Sentiment, Volatilities and Jumps of Intraday Returns" by Ya Qian, Jun Tu, Wolfgang Karl Härdle, January 2019.

003 "Estimating low sampling frequency risk measure by high-frequency data" by Niels Wesselhöfft, Wolfgang K. Härdle, January 2019.

004 "Constrained Kelly portfolios under alpha-stable laws" by Niels Wesselhöfft, Wolfgang K. Härdle, January 2019.

005 "Usage Continuance in Software-as-a-Service" by Elias Baumann, Jana Kern, Stefan Lessmann, February 2019.

006 "Adaptive Nonparametric Community Detection" by Larisa Adamyan, Kirill Efimov, Vladimir Spokoiny, February 2019.

007 "Localizing Multivariate CAViaR" by Yegor Klochkov, Wolfgang K. Härdle, Xiu Xu, March 2019.

008 "Forex Exchange Rate Forecasting Using Deep Recurrent Neural Networks" by Alexander J. Dautel, Wolfgang K. Härdle, Stefan Lessmann, Hsin-Vonn Seow, March 2019.

009 "Dynamic Network Perspective of Cryptocurrencies" by Li Guo, Yubo Tao, Wolfgang K. Härdle, April 2019.

010 "Understanding the Role of Housing in Inequality and Social Mobility" by Yang Tang, Xinwen Ni, April 2019.

011 "The role of medical expenses in the saving decision of elderly: a life cycle model" by Xinwen Ni, April 2019.

012 "Voting for Health Insurance Policy: the U.S. versus Europe" by Xinwen Ni, April 2019.

013 "Inference of Break-Points in High-Dimensional Time Series" by Likai Chen, Weining Wang, Wei Biao Wu, May 2019.

014 "Forecasting in Blockchain-based Local Energy Markets" by Michael Kostmann, Wolfgang K. Härdle, June 2019.

015 "Media-expressed tone, Option Characteristics, and Stock Return Predictability" by Cathy Yi-Hsuan Chen, Matthias R. Fengler, Wolfgang K. Härdle, Yanchu Liu, June 2019.

016 "What makes cryptocurrencies special? Investor sentiment and return predictability during the bubble" by Cathy Yi-Hsuan Chen, Roméo Després, Li Guo, Thomas Renault, June 2019.

017 "Portmanteau Test and Simultaneous Inference for Serial Covariances" by Han Xiao, Wei Biao Wu, July 2019.

018 "Phenotypic convergence of cryptocurrencies" by Daniel Traian Pele, Niels Wesselhöfft, Wolfgang K. Härdle, Michalis Kolossiatis, Yannis Yatracos, July 2019.

019 "Modelling Systemic Risk Using Neural Network Quantile Regression" by Georg Keilbar, Weining Wang, July 2019.

IRTG 1792, Spandauer Strasse 1, D-10178 Berlin
http://irtg1792.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the IRTG 1792.

IRTG 1792 Discussion Paper Series 2019

For a complete list of Discussion Papers published, please visit http://irtg1792.hu-berlin.de.

020 "Rise of the Machines? Intraday High-Frequency Trading Patterns of Cryptocurrencies" by Alla A. Petukhina, Raphael C. G. Reule, Wolfgang Karl Härdle, July 2019.

021 "FRM Financial Risk Meter" by Andrija Mihoci, Michael Althof, Cathy Yi-Hsuan Chen, Wolfgang Karl Härdle, July 2019.

022 "A Machine Learning Approach Towards Startup Success Prediction" by Cemre Ünal, Ioana Ceasu, September 2019.

023 "Can Deep Learning Predict Risky Retail Investors? A Case Study in Financial Risk Behavior Forecasting" by A. Kolesnikova, Y. Yang, S. Lessmann, T. Ma, M.-C. Sung, J.E.V. Johnson, September 2019.

IRTG 1792, Spandauer Strasse 1, D-10178 Berlin
http://irtg1792.hu-berlin.de

This research was supported by the Deutsche Forschungsgemeinschaft through the IRTG 1792.