Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Lars Lien Ankile
Kjartan Krange

Exploration of Forecasting Paradigms and a Generalized Forecasting Framework

Master's thesis in Industrial Economics and Technology Management
Supervisor: Sjur Westgaard
June 2022

NTNU
Norwegian University of
Science and Technology

Lars Lien Ankile
Kjartan Krange

Exploration of Forecasting Paradigms and a Generalized Forecasting Framework

Master's thesis in Industrial Economics and Technology Management
Supervisor: Sjur Westgaard
June 2022

Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Exploration of Forecasting Paradigms and a Generalized Forecasting Framework

Ankile, Lars L.[a], Krange, Kjartan[a]

[a]*Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology (NTNU), Alfred Getz vei 3, Gløshaugen, Trondheim, 7491, Trøndelag, Norway*

Abstract

First, this paper presents a total ordering of the theoretical lower bound loss of different forecasting paradigms in the following descending order: Model selection, Model combination, Non-parametric univariate models, and Non-parametric multivariate models.

Second, we create a generalized forecasting framework to test the above forecasting paradigms ex-ante. We implement the framework by creating a novel datacube consisting of daily stock prices and 100,000 quarterly reports from about 1600 global companies and several daily macro time series, all from 2000 to spring 2022. Lastly, we utilize the framework and show that modern multivariate time series approaches are powerful but domain-dependent. We demonstrate the domain-dependent accuracy by showing convincing results when predicting corporate bankruptcy risk, moderate results when predicting stock price volatility, and lacking results when finally predicting company market capitalization.

Given the domain-dependent convincing results and mostly unrealized theoretical lower bound loss of multivariate approaches, we hope to encourage further research on non-parametric, multi-signal approaches that leverage a wider array of available information.

Keywords: forecasting, time series, machine learning

Sammendrag

Først presenterer vi en total ordning av den teoretiske nedre grense for prediksjonsfeil for forskjellige prediksjonsparadigmer i følgende synkende rekkefølge: modellvalg, modellkombinasjon, ikke-parametriske univariate modeller og ikke-parametriske multivariate modeller.

Deretter lager vi et generalisert prediksjonsrammeverk for å teste de ovennevnte prediksjonsparadigmer på forhånd. Vi implementerer rammeverket ved å lage en original datakube bestående av daglige aksjekurser og 100 000 kvartalsrapporter fra omlag 1 600 globale selskaper og flere daglige makrotidsserier, fra 2000 til våren 2022. Til slutt bruker vi rammeverket og viser at moderne multivariate tidsseriemodeller er kraftige, men domeneavhengige. Vi demonstrerer den domeneavhengige nøyaktigheten gjennom overbevisende resultater for anslag av bedriftskonkursrisiko, moderate resultater ved prediksjon av aksjekursvolatilitet og manglende resultater når vi til slutt forutsier selskapets markedsverdi.

Gitt gode resultater i visse domener, samt fortsatt urealisert teoretisk høyeste nøyaktighet av multivariate modeller, håper vi å oppmuntre til videre forskning på ikke-parametriske multisignaltilnærminger som utnytter et bredere spekter av tilgjengelig informasjon.

Preface

With these words, written on the evening of 11.06.2022, we conclude our Master's Thesis after five years at the Norwegian University of Science and Technology.

First and foremost, it has been a wonderful experience becoming adults in the scenic environments around Gløshaugen and the Nidaros river. From the bottom of our hearts, we want to thank our friends, more numerous than we can count, for all the ravishing times we have shared and the hardships we have faced together. This is how lifelong friendships are made.

Secondly, we want to profess the joy we find in the world of science and learning. The following work is the product of countless hours spent pouring over books, lively discussions with peers, and nervous hours before examinations. We have fortified our belief that thinking for oneself is the highest virtue, best amplified by reading vigorously.

Lastly, we want to thank each other for our fruitful collaboration over the years. Debating partners, teachers for each other, but most of all for being good friends. This has been especially valuable when we have been very alone in this work, with virtually no advising from the department.

If we ever get kids, and you are one of those kids reading this, you are not yet as smart as we are right now... buhu ... go back to your books!

Lars Lien Ankile and Kjartan Krange 11.06.2022 21:41

Contents

0. Introduction

Time series modeling has historically played a vital role in applications such as climate modeling, biological sciences, medicine, supply chains, and finance (Lim and Zohren, 2021). As a result, researchers have proposed a plethora of different approaches, of which most have focused on parametric models (e.g., auto-regressive methods, exponential smoothing, structural time series models) (Lim and Zohren, 2021). Parametric models are models where one defines the functional form and fits the model by adjusting the function's parameters.

Using Neural Networks (NN) for forecasting is not new, and in fact dates at least back to 1964 (Zhang et al., 1998), where Hu (1964) uses an adaptive linear network forecast weather. Any big strides were not possible, however, until the backpropagation algorithm for optimizing non-linear NNs was introduced by Rumelhart et al. (1986) (see subsubsection 2.2.2 for a brief primer on backpropagation and gradient descent). From this point, and until today, there have been a plethora of increasingly complex applications of different models to forecasting (Zhang et al., 1998; Jiang, 2021; Gandhmal and Kumar, 2019).

As more data, computing power, and more complex models have become available, the field of forecasting has seen significant improvements (Makridakis et al., 2018a). Still, there is a high reliance on the classic econometric methods of old in practice.

The introduction of Deep Learning (DL) has upended a staggeringly wide array of fields, such as image classification (Krizhevsky et al., 2012), natural language processing (Brown et al., 2020), chess and go engines (Silver et al., 2018), protein folding (Senior et al., 2020), and nuclear fusion plasma control (Degrave et al., 2022). Non-parametric DL methods have bested the previous human heuristic approaches in all these fields–and the trend is only pointing in one direction.

Furthermore, also within the forecasting field, recent work shows the superiority of DL approaches that use models that rely less heavily on assumptions and allow for a more purely data-driven approach to solving the problems at hand (Ahmed et al., 2010; Oreshkin et al., 2019). DL methods can be interpreted as non-parametric in that their functional form is not defined on an assumption of the underlying problems data generating process. Furthermore, (Lee et al., 2017) proves that in the limit of infinite width, a deep neural network is a Gaussian process (GP), an example of a non-parametric model. Thus, we describe deep learning as non-parametric in this paper.

Forecasting methods can be univariate or multivariate, and we find the former most widely discussed in practice. Moreover, the fact that the most prominent forecasting competitions, M4 (Makridakis et al., 2018a) and the ongoing M6 competition (Makridakis, 2022), operate only on univariate time series further corroborates this finding. Since univariate methods are a subset of multivariate methods, this univariate paradigm likely restricts the upper limit forecasting accuracy.

However, there is some evidence showing parametric multivariate approaches struggle with outperforming their univariate counterparts (Fortin et al., 2022). On the other hand, recent research finds that non-parametric multivariate approaches can have superior performance to univariate approaches (Huang et al., 2021; Guiguet et al., 2018; Chaudhuri and Ghosh, 2016).

In finance, Atsalakis and Valavanis (2009) and Nti et al. (2020) find that technical analysis is the most prominent forecasting type, which only uses information held in lags of the dependent variable itself. At the same time, we know investors leverage several different sources of information, like financial statements and macro numbers, when investing, i.e., a multivariate approach. Moreover, the data on the effectiveness of technical analysis is dubious at best Nti et al. (2020); Nazário et al. (2017). As soon as transaction costs and spreads are included in the calculation, any performance tends to evaporate (Nazário et al., 2017). Furthermore, Geva and Zahavi (2014) finds that combining several data sources leads to superior predictive performance, but only for sufficiently complex models, i.e., Neural Networks.

As opposed to non-parametric DL methods, parametric models hard-code human judgment and domain knowledge assumptions into the model, potentially harmful restrictions on the expressive ability of the model. This paper will try to motivate the view that this is not the most effective way of forecasting in the modern data-driven era. Intuitively, Data Generating Processes (DGPs) of the natural world are often non-linear, and the goal of forecasting is precisely to approximate DGPs.

In Ankile and Krange (2022) we created the world-class ensembling forecasting algorithm on the M4 dataset (Makridakis et al., 2018a), and the third-lowest loss on the test set known to the authors. Still, we believe these results are limited by the assumptions of the ensembling approach. In this paper, our aim is to establish a theoretical lower bound loss of different forecasting methods as well and induce a shift towards multivariate forecasting approaches in the forecasting field. Furthermore, the aim of this work is to research the best-in-class non-parametric multivariate forecasting in theory and practice.

In particular, we will establish a total order of modeling power between different model paradigms. These include local and global models, parametric and non-parametric models, and univariate and multivariate models. We will empirically test this total order on a novel financial dataset introduced in this paper, that we term a datacube.

The rest of the paper is organized as follows. In Part , section 1 presents theory on forecasting paradigms and summarizes some relevant literature in the fields of bankruptcy, volatility, and stock price prediction, upon which we build our forecasting framework. Furthermore, in section 2 we lay out some technical foundation underlying the different techniques implemented. In part Part I, section 4 we present a total order of theoretical lower bound error for different model paradigms by providing best-case and average-case analysis. Next, Part II, section 5.3.1 describes the dataset and processing pipeline. Then, section 5.3.1 provides an overview of the setup of the forecasting framework and its constituent parts, evaluation, and analysis methods. section 6.4.2 presents the results from the three different experiments the framework is applied to. Lastly, in Part II, section 7.3.2, we discuss some problems with the framework and possible further work. Lastly, we conclude the paper in section 8.11.

Part
Context

1. Background

1.1. Background on Forecasting Paradigms

1.1.1. Parametric, Univariate Models

Traditional modeling procedures, introduced by Box and Jenkins (Box, 1970) in the 1970s, combine linear Auto-Regression (AR) and Moving Average (MA), e.g., the popular ARIMA model (Box, 1994) and the general Exponential Smoothing model (Brown, 2004), where a forecast is equal to its forecast predecessor and a scaled error based on the predecessor forecast. Parametric models also model variance, like the influential heteroscedastic models of ARCH (Engle, 1982) and GARCH model (Engle, 2001).

These methods have been workhorses in the literature and have performed very well compared to more complex methods for decades (Makridakis and Hibon, 2000). However, in recent years, more complex models have started to outperform the more straightforward methods (Makridakis et al., 2018a; Ahmed et al., 2010). The renowned M4 competition (Makridakis et al., 2018a) showed that complex approaches using Deep Learning (DL) and gradient descent have come to outperform the traditional statistical models significantly. This result was an important finding as all M competitions up to, and including, M3 found the opposite, i.e., simple methods outperform complex ones (Makridakis and Hibon, 2000).

1.1.2. Non-Parametric, Univariate Models

LeCun et al. (2015) point out that DL methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, and a vast array of other domains. Using DL to forecast the financial markets is nothing new. However, it is still not widely embraced in the literature, especially among researchers in the economic, financial, and econometric fields, as can be seen by perusing the latest publications in the relevant journals.

One of the first examples of using a DL sequence model, the Long Short-Term Memory network (LSTM), was made by Fischer and Krauss (2018). They find that a relatively simple LSTM model outperforms Feed-Forward Neural Nets (FNN) and Random Forests (RF).

1.1.3. Parametric, Multivariate Models

Researchers have proposed Multivariate generalizations of the ubiquitous univariate, parametric models, like the Vector Auto-Regressive model (Toda, 1991), the Multivariate Exponential Smoothing (Enns et al., 1982), and the Seasonal ARIMA with Exogenous variables (Vagropoulos et al., 2016). Advanced models include state representation, called state-space models, of which the regime switch model as presented in (Brooks, 2019) is an example. Furthermore, common trend modeling, as introduced by Stock and Watson (1988), tests for common cycles between time series, which again can be used in forecasting. In different papers, parametric multivariate models show both superior (Bagshaw, 1987) and inferior (Brandt and Bessler, 1984) to parametric univariate ones. These results are peculiar as most multivariate approaches always could set coefficients for the other time series lags to zero, thus being reduced to the corresponding univariate model. The cause of this discrepancy is likely a lack of modeling power in the parametric approaches and a lack of practitioner sophistication required in forecasting. Regardless, there is ample room for improvement.

1.1.4. Non-Parametric Multivariate Forecasting

Already in 1992, researchers Chakraborty et al. (1992) established the superiority of multivariate methods, Neural Networks in particular, for such forecasting. They find that their simple, feed-forward neural network outperforms the multivariate ARMA model by Tiao and Tsay (1989) an order of magnitude.

Most statistical methods model only linear relationships, which are computationally inexpensive, but often cannot accurately represent temporal variations (Chakraborty et al., 1992). The ability of non-parametric methods to non-linear model relationships, like Deep Neural Nets (DNN), is likely a driver of the superior performance of the DNN approach as compared to statistical modeling. See also Hornik et al. (1989) for a primer on DNNs' ability to model arbitrarily complex functions.

The prominent classical univariate AR time series model, and its extensions ARMA, ARIMA, and VAR, are highly utilized in practice. Still, they inevitably suffer from overfitting and high computational cost for high-dimensional inputs and series with long-term dependencies (Wan et al., 2019).

Several examples of more advanced multivariate models utilizing modern machine learning methods have appeared in the literature. Wan et al. (2019) used an NN architecture called a Multivariate Temporal Convolutional Neural Net (M-TCN), to forecast time series from the *Beijing PM2.5* and *ISO-NE* datasets (Wan et al., 2019). This architecture is an augmented version of the Temporal Convolutional Neural Net (TCN) proposed by Bai et al. (2018), which has shown great promise on several sequence modeling tasks.

Zhang et al. (2019) show that a Multi-task TCN outperforms separate TCNs per series for a dataset of different water quality measurements. Pantiskas et al. (2020) propose an architecture based on TCN and an Attention Mechanism (Vaswani et al., 2017), called Temporal Attention Convolutional Neural Net (TACN), to forecast an air quality dataset and water quality. They use the same water quality dataset as Zhang et al. (2019) and also use their Multi-task TCN as the baseline model for both datasets. The TACN model outperforms the baseline.

1.1.5. Assumption of Linear Variable Relationships

In many forecasting techniques correlation is used in the form of e.g. PCA (Stock and Watson, 2002), and clustering (Pawlikowski and Chorowska, 2020). We postulate that adhering to these forms of (often linear) relationships between variables is something that reduces the possibility of finding nonlinear correlation.

Creating a well-defined, dependent relationship between two variables that still show no linear correlation is trivial.

$$
\begin{aligned}
x &\sim (0, \sigma) \\
y(x) &= \begin{cases} x & \text{if}\, x \leq 0 \\ -x & \text{if}\, x > 0 \end{cases} \\
\implies &\mathbb{E}(COR(y, x)) = 0
\end{aligned}
$$

1.1.6. Global and Local Models

The *local* method within forecasting assumes that each time series of a set stems from a data generating process independent from other processes (Montero-Manso and Hyndman, 2021). This assumption implies that the best approach to fitting models to a group of time series is fitting one independent model to each series. The local approach to modeling is to for each time series in a set of series to fit a single model and makes a forecast, Montero-Manso and Hyndman (2021). This method is prone to overfitting and requires a significant emphasis on manually expressing information of y_t by adjusting for seasonality, short- and long-term trends, and explanatory variables in the case of the local multivariate model.

An alternative to the local approach is called the *global* approach, as presented in Salinas et al. (2020) (also termed cross-learning). Global models assume that the underlying data generating processes for a set of time series share properties and 'behave' similarly. Thus by fitting a model on more than a single time series at once, one can improve accuracy by cross-learning global properties. For instance, one can do this by fitting an ARIMA model across an entire set of time series before forecasting, thus reducing noise and learning global patterns in the data. Montero-Manso and Hyndman (2021) find that across some different example datasets like M3 monthly data (Makridakis and Hibon, 2000) a deep neural net global model approaches the accuracy of standard parametric methods such as theta, exponential smoothing, and ARIMA. Parametric global models perform significantly worse than their parametric local versions.

We believe this dichotomy and classification of properties might be flawed and present our thoughts on local and global methods in section 4.5.

1.1.7. Model Ensembling

Ensemble forecasting produces forecasts from a set of forecasting algorithms fitted to a given problem. In the M4 competition both the 2. and 3. places used a form of model ensembling (Makridakis et al., 2018a). There are two main ways of producing ensemble forecasts given a set of forecasting methods, $\mathcal{F}$ that all produce forecasts for the same Time Series (TS). (1) *Model selection* involves picking the most promising model $f \in \mathcal{F}$, based on limited information, that will have the highest accuracy. (2) *Model combination*, on the other hand, involves estimating weights for each forecast $\hat{\mathbf{p}}_{\mathcal{F}} = \{\hat{p}_f \,|\, \sum_{\mathcal{F}} \hat{p}_f = 1 \wedge \hat{p}_f \geq 0 \quad \forall f \in \mathcal{F}\}$, such that the accuracy of the weighted average is as high as possible. Generally, an aim is that the model combination outperforms the accuracy of the individual forecasts combined (Petropoulos et al., 2020).

Lemke and Gabrys (2010) found model combination improved the forecasting ability of the ensemble to a level above the best performance of the individual forecasting methods. As a result, the paper shows that model combination is able to outperform selection even on a meta-level.

The 'no-free-lunch' theorem establishes that no algorithm can outperform all other models or the random forecast when testing on all possible data (Wolpert, 1996). This theorem implies that without knowing anything about a problem, one cannot assume anything about the performance of an algorithm (Lemke and Gabrys, 2010). Of course, there will be specific problems for which one algorithm performs better than another in practice. Thus, Lemke and Gabrys (2010) shows that the above assumption can be relaxed by automatically extracting features from series that will function as a proxy for domain knowledge. Furthermore, Goodfellow et al. (2016) argues that the reason model combination works are that different models will not have residuals with perfect correlation for the test set. At the same time, model diversity alone must not be the only aspect of an accurate combination of methods—it is individual accuracy as well, according to Lemke and Gabrys (2010).

Wolpert (1996)s no-free-lunch theorem makes it evident that a single algorithm cannot perform better than the random forecast or all other models when the algorithm is tested on all possible data. Lemke and Gabrys (2010) shows that the theorem implies that without knowing anything about a problem it is impossible to deduce anything about the performance of an algorithm. In practice, it is trivially true that problems have algorithms that outperform other procedures. In this regard, Lemke and Gabrys (2010) relaxes the above assumption by extracting features from series to proxy as domain knowledge. Moreover, Goodfellow et al. (2016) argues that model combination works because different models have residuals that are not perfectly correlated on a test set. Model diversity alone cannot be the single matter when designing combination methods, individual accuracy matters as according to Lemke and Gabrys (2010)

1.2. Background on Financial Forecasting

1.2.1. Bankruptcy prediction

Bankruptcy prediction as in being able to predict defaults on loans before they happen in time can intuitively be of great economic value to firms and as a result society in general.

Following the Norwegian definition an entity is bankrupt when it is: 1) *Illiquid*, meaning that its liquid assets do not meet its short-term liabilities. 2) *Insolvent* that its total assets are less than its total liabilities (Brækhus, 2017). Although this formulation is based on Norwegian law and law practice, we believe reformulations of the above are globally what is recognized as a bankruptcy.

Correct prediction of 1) and 2) above, is of great interest to the economic sector, and especially for banks. Stein (2005) shows that acting on more accurate prediction models of defaults translates into more *profitable* actions. Thus proving the economic and social relevance of bankruptcy prediction.

Categorical bankruptcy prediction is by nature dependent upon a distribution of skewed labels which poses some challenges. Indeed for most standard real-world datasets of companies, there are fewer defaults than non-defaults. Logically, if a company's median lifetime is strictly greater than the frequency sample of the dataset, one would have more samples of non-defaults than defaults. Empirical proof of the above claim can for instance found in Kainth and Wahlstrøm (2021)s extensive dataset of Norwegian and Swedish privately held companies. 1.8% and 1.5% of their data are defined as bankrupt according to NGAAP and IFRS respectively. A recent example of a study of this problem in general called the *unbalanced data problem* can be found in (Krawczyk, 2016).

Lastly, there are intricacies in what time horizon one should use when forecasting bankruptcy. By increasing the forecasting window a company has a strictly lower chance of not going bankrupt. The Basel III regulatory framework defines that a one year should be used (on Banking Supervision, 2017) (thus is also what we opt for, see: subsection 6.2).

1.2.2. Volatility Prediction

There are several methods to say something about the volatility in the future, e.g., forecasting using models, calculating implied volatility from options (Mayhew, 1995), tracking the Volatility Index (VIX)[1], or analyzing the macro trends in the wider market. The focus of the present work will be on model-based forecasting.

In the financial markets, volatility is a significant number as it is the input to considerations such as risk management in investment, option pricing, and regulation of financial markets. Several large institutions trade options and swaps that depend on volatility for speculation or hedging, and the Basel Accord of 1996 introduced solidity requirements based on VaR, which in turn relies on volatility forecasting (Granger and Poon, 2001).

Furthermore, research interest saw an increase after the quantitative work of Engle (1982) with the Autoregressive and Conditional Heteroscedastic (ARCH) model, later augmented to Generalized ARCH (GARCH) by Bollerslev (1986). (We discuss GARCH in more detail in subsubsection 2.1.2). Granger and Poon (2001) find that out of 78 papers reviewed, 45 used some sort of variation of ARCH (the other 33 used implied volatility). This finding shows the popularity of ARCH models for volatility modeling from 1982 until ca. 2000.

Before the introduction of the ARCH family of models, simpler models pervaded, like the *Random Walk* model, *Historical Average*, *Simple Moving Average*, *Exponential Smoothing*, and *Exponentially Weighted Moving Average*, as well as certain extensions (like Taylor (2004)).

As Deep Learning has caught the attention of researchers in all fields, so has the interest in applying it to volatility forecasting increased. For example, Chen et al. (2017b) use Recurrent Neural Nets (RNN) with Gated Recurrent Units (GRU) to forecast volatility in the Chinese stock market. Another paper, Alenezy et al. (2021), uses a collection of non-linear transforms in combination with an adaptive network-based fuzzy inference system to outperform an ARIMA model in predicting volatility in the Saudi stock market. Harrison and Moore (2012) find, in a study comparing models of differing complexity in several European countries, that more complex models that allow for asymmetric volatility outperform the simpler ones with symmetry restrictions.

Despite the large strides forward within machine learning and the increasing adoption of such algorithms for, e.g., stock market forecasting, the volatility literature still seems centered around the AR-, ARCH-, and GARCH-families of models (Lim and Sek, 2013; Nonejad, 2017; Lin, 2018).

[1]Traded on the Chicago Board Options Exchange (INDEXCBOE: VIX)

1.2.3. Market Capitalization Prediction

The case of predicting stock prices is an interesting one. On the one hand, it must be possible, given the momentous success over many years of algorithmic trading houses like Renaissance Technologies, Citadel Securities, D.E. Shaw, Jane Street, and Two Sigma Securities. On the other hand, anyone who has discovered a profitable strategy will not be incentivized share the approach, as such strategies are only viable when information asymmetries exist. (If everyone trades the same strategy, the strategy seizes to work as everything is priced in when one has taken the position.) Thus, the academic literature on the topic will necessarily not represent the true status of the field, and will most likely be dominated by non-results.

There are many types of time series, where financial series (especially stock prices) are on the hard side of the spectrum of how predictable they are, especially for traditional statistical methods (Niaki and Hoseinzade, 2013). Some important attributes causing this is that they are:

- *non-linear*, i.e., there are often relationships between inputs and outputs that are not linear, which can cause many classical methods to break down (see subsubsection 1.1.5 for an example),
- *noisy*, i.e., the price is only partly decided by intrinsic value and underlying value drivers, and in large part (especially short-term) because of randomness and chaos,
- and *non-stationary*, i.e., stock prices tend to have non-constant mean and variance, but rather the opposite, which poses problems for methods that assume stationarity.

Less academically, and more intuitively: it is hard to beat all other agents in a market, especially when intelligent dollars grow faster than their counterpart. Keeping the above points in mind, researchers have highlighted some aspects of NNs that would motivate the use and continued exploration of their limits in financial forecasting going forward (Eakins and Stansell, 2003; Hornik et al., 1989; Lam, 2004), for example, they are

- *non-presumptive in data*, i.e., they do not assume any particular distribution on the underlying data,
- *non-presumptive in form*, i.e., they do not assume that the solution takes any particular functional form and only rely only on the data given during training,
- *nonlinear*, i.e., they can in theory model any kind of relationship $\mathbb{R}^n \to \mathbb{R}^m$ (see subsubsection 2.2.1 for more on NNs as universal function approximators),
- and *generalizing*, i.e., they have proven time and time again that they are able to extract relationships from inputs and outputs in the data it has seen before successfully applying them to novel data.

Despite having many desirable characteristics, they also have some important drawbacks (Hussain et al., 2008), for example, they are

- *sensitive to parameter choices*, i.e., the number of parameters that must be tuned just right makes the space of possible models enormous, and finding the right combination might be demanding,
- *sensitive to inputs*, i.e., they often require that inputs be properly scaled, and it can be difficult to decide decisively what features to include in the model,
- and *data hungry*, i.e., they require an immense amount of data to be able to generalize, which is not always feasible to either collect or store.

Regarding these limitations, several strategies have been devised to mitigate the effects of the problems, which will be discussed further in subsubsection 2.2.1. Here we mention that NNs can be suitable for both univariate and multivariate prediction (Tay and Cao, 2001). One early application of Neural Networks used in the financial markets was by Werbos (1988), where they used an NN to make time-series predictions in US gas markets, and convincingly outperformed more traditional least-squares methods (e.g., linear regression and Box-Jenkins).

The most common approach to stock market prediction is to use only raw prices and derivatives thereof (Gandhmal and Kumar, 2019; Jiang, 2021). For example, Jiang (2021) find in a systematic review of the stock market prediction literature that 36% of surveyed papers used historical stock prices only, while another 25% used stock prices and technical indicators (61% total). 6% of papers used stock prices, indicators, and macro data, while 2% included fundamental data at all. Since technical indicators are solely a transformation of the raw stock price, adding them to the dataset does not introduce new data, it can only introduce the data in a manner that might be easier for the model to utilize.

One problem of many of the approaches in the literature is that they do not fully calculate the profitability of the strategy, only metrics like accuracy, AUC, MAPE, or similar, which only quantifies if predictions are generally more or less accurate, but not if they are useful in the context of their time series.

2. Technical Theory

2.1. Baselines and Parametric Models

First and foremost, we use Naïve as a baseline in all three forecasting problems. In all timeseries forecasting beating Naïve is almost always an important benchmark. This paper, and most literature e.g. Brooks (2019); Makridakis et al. (2018a), define Naïve as $\hat{y}_t = y_{t-1}$.

When it is used in the context of bankruptcy prediction, there are examples where the strategy shows good results in terms of accuracy for instance in Moen (2020).

In the case of volatility forecasting, we suspect naïve is ought to do well, due to volatility clustering (Mandelbrot, 1997).

If y_t is market cap prediction, beating this benchmark (perhaps adjusted for risk-free return) is a holy grail. If we actually were able to do this, it is likely that this part of the paper would not have been added, for obvious monetary reasons.

Naïve is the most important baseline. Although its name is simple, in all three forecasting problems we present, it encodes much important information.

2.1.1. ARIMA

The ARIMA model (AutoRegressive Integrated Moving Average) model, is a monument in the time series forecasting literature. Its impact on time series forecasting and famousness is the reason we chose to add ARIMA models as a benchmark in this paper. This fact also makes our results more comparable to others. ARIMA is, for instance, used as a benchmark in Makridakis et al. (2018a).

An ARMA(n, m) process has the following form, as presented in Brooks (2019):

$$y_t = \mu + \phi_1 y_{t-1} + ...\phi_n y_{t-n} + \theta_1 u_{t-1} + ...\theta_m u_{t-m} + u_t. \tag{1}$$

If we difference y_t d times, and plug it into Equation 1 this would yield the ARIMA(n, d, m) process. The model has a well-behaved mean dependent upon ϕ, indeed it is easy to see that if $\phi > 1$ the model is growing out of control since shocks to the system are infinitely propagated. An AR process's properties are independent of the time at which the series is observed (stationary) if the following holds:

$$|z_i| > 1 \forall i \in \{0, ...m\} \wedge 1 - \phi_1 z_1 + ...\phi_n z_n = 0 \tag{2}$$

MA processes with non-infinite lags are trivially stationary.

If we restrict the max lag of n and m lags of AR and MA processes, there are 2^{n+m} possible lag configurations. In practice, standard implementations limit the search space to include all lags up to a given max lag. i.e. you only have to search over nm lags. One also needs some ex-ante metrics for what is deemed a good model, since overfitting can easily happen in such a large space of possible lags. As a solution to this problem various forms of information criteria can be selected, such as AIC, SBIC, HQIC (Brooks, 2019), without writing down any of these formulas here: in general information criteria weights the fall in the residual sum of squares against a penalty for adding more variables to an equation.

The ARIMAX model (ARIMA eXogenous) adds exogenous variables with one lag of each variable to Equation 1. Literature has different conclusions about its effectiveness (Peter and Silvia, 2012; Kongcharoen and Kruangpradit, 2013), both finding its forecasting accuracy lower and higher compared to that of a standard ARIMA model. When forecasting with exogenous variables one needs to make assumptions about its future values, this follows logically from the fact one does not have values for the exogenous values in the future. One can for instance use the naive prediction for the exogenous variables (feed information forward) or use some other prediction tool.

2.1.2. GARCH

The GARCH (Generalized Autoregressive Conditionally Heteroscedastic) model class, models volatility as non-constant and conditional upon previous own lags and error terms. A generalized example of the GARCH model presented by Brooks (2019) is the following, where y_t is modeled as an AR(1):

$$y_t = \mu + u_t, u_t \sim \mathcal{N}(0, \sigma_t^2), \tag{3}$$

$$\sigma_t^2 = \alpha_0 + \sum_i^q \alpha_i u_{t-i} + \sum_j^p \beta_j \sigma_{t-j}^2 \tag{4}$$

The idea of the GARCH model is that time series are often not homoscedastic in reality (Mandelbrot, 1997). Thus using errors and volatility lags can model the time-dependent variance of reality, as signified by the t in σ_t of Equation 3.

2.1.3. Logistic Regression

The Logistic Regression method is used to classify a binary target variable. In the below p is the target variable (probability that Y=1):

$$p = \frac{e^{a+bX}}{1 + e^{a+bX}},$$
$$\ln(\frac{p}{1-p}) = a + bX$$

Where one fits the model in the second line of the above equation, giving rise to the value of p, which can be interpreted as a probability of a label. This is given by X, thus the model returns a likelihood of a label given a particular set of observations (X).

2.2. Non-Parametric Methods and Deep Learning

4 of the following subsections (subsubsection 2.2.1, subsubsection 2.2.2, subsubsection 2.2.4, subsubsection 2.2.5) are adapted from Ankile and Krange (2022).

2.2.1. Basics of Neural Networks

An Artificial Neural Network (ANN), or simply Neural Network (NN), is a subset of machine learning methods that receive their name since they resemble brains in some ways. It is the notion of neurons being connected in an intricate. In our time series models, we mainly use a specific form of an NN, called a Temporal Convolutional Neural Network (TCN), which we discuss in more detail in subsubsection 2.2.9).

A NN performs non-linear transformations from input to output and can, in theory, approximate any real-valued function from one finite-dimensional space to another to any desired degree of accuracy. Provided there is at least one hidden layer, a non-linear squashing function is present, and sufficiently many hidden units are available (Hornik et al., 1989). A NN works by iteratively using matrix multiplication (linear transformation), importantly followed by a non-linear function (Goodfellow et al., 2016). See subsubsection 2.2.3 for more on activation functions.

Equation 5 shows a single non-linear transformation layer $\mathbf{L}$, where g is a non-linear activation function, $\mathbf{A}$ is a linear transformation, and $\mathbf{b}$ is the bias. There is a difference in the complexity of a linear, and non-linear model can learn, for instance, the XOR function, which is infeasible for a linear function but trivial for several non-linear models (Minsky and Papert, 2017).

$$\mathbf{L} = g(\mathbf{A}^T\mathbf{x} + \mathbf{b}) \tag{5}$$

One can depict a NN as a graph, and, more formally, it is the iterative application of the above function for an arbitrary number of steps, with an arbitrary number of parameters at each step (this flexibility makes the model susceptible to being over-parameterized, which we briefly discuss in subsubsection 2.2.5). For instance a neural network with one hidden layer could be written as $y_{dep} = L_{out} \circ L_h \circ L_{in}(x_{ind})$. The first layer, the input layer, is provided data, which is passed through the network, layer by layer. At the last layer, L_{out} creates the output with a fitting activation that is application-specific (e.g., classification or regression). The output then flows to a loss measure (see subsection 6.2), which updates the parameters in the network (often called weights), backward through the layers in a process called backpropagation (Goodfellow et al., 2016). Note that if one were to remove g from Equation 5, the complexity increase from iteratively adding layers would be removed as $\mathbf{A_1 A_2 ... A_n x} = \mathbf{Ax}$, results in a linear transformation, where $\mathbf{A_i}$ are all linear transformations.

A Fully-Connected Neural Network, as our weighting model is an example of, is an NN where all layers are fully connected, i.e., there is an edge between each node in one layer to all nodes in the next (nm connections between a layer with n and m nodes).

2.2.2. Gradient Descent

Gradient descent is a way of fitting NNs by reducing the value of $f(x)$ by moving a small negative direction of the derivative. Equation 6 shows how gradient descent is deduced as in Goodfellow et al. (2016). Since the relationship

$$f(x + \epsilon) \approx f(x) + \epsilon \frac{df}{dx}, \tag{6}$$

holds at sufficiently small ϵ, the derivative helps minimize a loss function as it shows what direction would cause a decrease in the loss (i.e., an increase in performance or accuracy). This observation leads to the relationship

$$\rightarrow f(x - \epsilon \cdot \text{sign}(\frac{df}{dx})) \leq f(x), \tag{7}$$

for ϵ small enough. Therefore, we can minimize the loss by moving in small increments in the opposite direction of the derivative. Cauchy et al. (1847) originally termed this process gradient descent.

Gradient descent is the primary step of backpropagation, where each weight is updated recursively backward through the model, layer by layer. Gradient descent can also be generalized for multi-input cases, where the aim would be to move in the direction of the steepest descent in the multidimensional space of the loss function. In the multi-variable case, we would get

$$\theta' = \theta - \epsilon \nabla_\theta \mathcal{L}(\theta), \tag{8}$$

where ϵ is the learning rate and $\nabla_\theta \mathcal{L}(\theta)$ is the vector containing all partial derivatives of the loss function $\mathcal{L}$ with respect to the model parameters θ (Goodfellow et al., 2016).

2.2.3. Activation functions

For gradient descent to work, all activation functions must be differentiable.

The *Rectified Linear Unit* (ReLU) is the most widely used non-linear activation function in modern deep learning (LeCun et al., 2015). It is simply a half-wave rectifier, which means that the function propagates values 0 or greater unchanged, but takes all negative inputs to 0. Se Equation 9 for the mathematical definition and Figure 1 for a visual representation of the function.

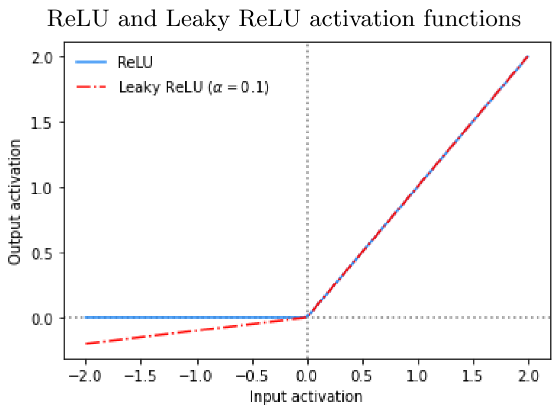


Figure 1: Visualization of the common activation functions Rectified Linear Unit (ReLU) and its extension, Leaky ReLU.

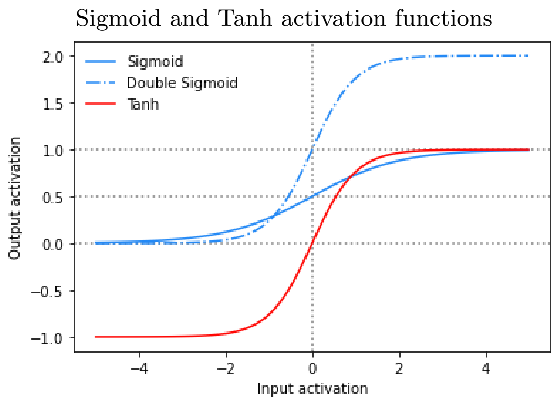


Figure 2: Visualization of the previously prevalent activation functions Sigmoid (in blue) and Tanh (red), as well as a variant of Sigmoid that is stretched to double height and used in experiments in section 5.3.1.

Previously, smoother activation functions, like the hyperbolic tangent, $\tanh x$, or sigmoid, $\frac{1}{1+e^{-x}}$, were in vogue (LeCun et al., 2015) (see Figure 2 for visualizations), but ReLU enables NNs to learn much faster than the aforementioned functions, especially as the networks get deep, and has naturally found wide adoption.

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{9}$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -\alpha x & \text{if } x < 0 \end{cases} \tag{10}$$

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{11}$$

2.2.4. Hyperparameter Tuning

Modern machine learning models have increased exponentially in complexity over the last couple of decades, and with it, an exponentially increasing space of hyperparameters to search over Li et al. (2017). Unfortunately, there are, in general, no methods that guarantee optimal hyperparameters except for a brute force search, for most hyperparameters Snoek et al. (2012). We present a discussion about how we can solve this vital issue in subsubsection 6.3.5.

2.2.5. Early Stopping

Large models with a sufficiently large number of parameters can suffer from overfitting, meaning that the model stops learning general relationships between input and output and instead learns to fit the noise of, or memorize, the training set. We can observe this phenomenon by a deviation in training and validation loss, exemplified in Figure 3. Goodfellow et al. (2016) argue that the best regularization technique for NNs is to stop the training at the point where the validation loss is at its lowest, known as *early stopping*. This regularization technique requires using some of the training data as a validation set, reducing the size of the training set. This limitation can be overcome by retraining on the entire training set using the exact specifications as found previously with the optimal hyperparameters and stopping point (Goodfellow et al., 2016).

2.2.6. Feed-Forward Neural Networks

Feed-Forward Neural Networks (FFNN), also called Multi-Layer Perceptrons (MLP) are the fundamental building blocks in deep learning (Goodfellow et al., 2016). The model is composed of successive layers of neurons, or nodes, that are connected to all neurons in the previous layer and all neurons in the next layer. Figure 5 shows a conceptual representation of a fully-connected network with 4 layers, where the leftmost layer serves as the input layer and the rightmost serves as the output layer. The 2 layers in the middle are known as hidden layers (Goodfellow et al., 2016).

The connection between successive layers is implemented as matrix multiplications, where an input vector of length n is multiplied with a matrix

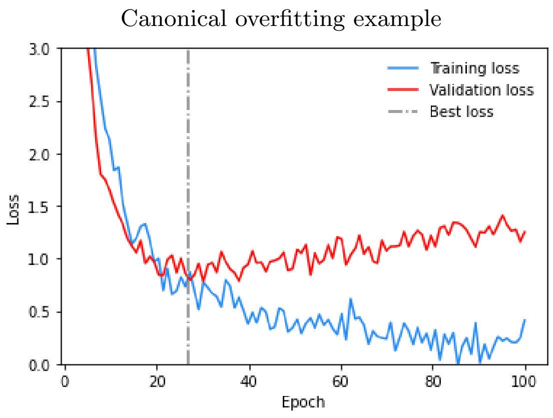


Figure 3: Illustration of the tell-tale signs of a model that has been overfitted. The defining factor is that the training loss falls (blue line) while the validation loss (red line) starts to increase. A common strategy is to save the weights at the currently best validation loss and restore those weights when training concludes (Goodfellow et al., 2016).

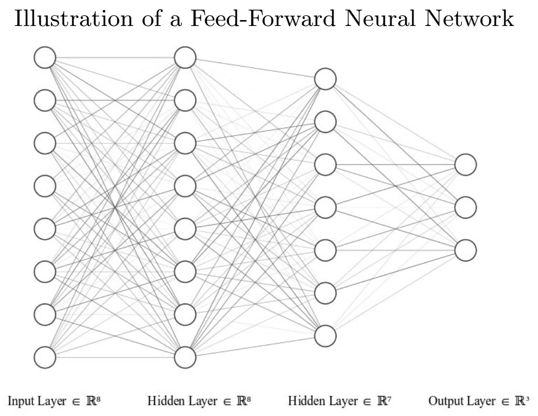


Figure 4: Schematic of a canonical Fully-connected, feed-forward neural network. The input travels from left to right, from the input layer, through two hidden layers, to three output nodes in the last layer.

of size $n \times m$, where m is the number of neurons in the subsequent layer. A bias term is added to the product before the result is passed to a non-linear activation function. This is necessary to allow the network to model non-linear functions, as discussed in subsubsection 2.2.2.

2.2.7. Convolutional Neural Networks

Convolutional Neural Networks (CNN) dates back to a paper by LeCun et al. (1989). This neural network architecture is specialized for data that elicits grid-topology (i.e., points residing closely to each other have higher importance than those far away) (Goodfellow et al., 2016). Examples are time-series data (1-dimensional), black and white images (2-dimensional), color images (3-dimensional, and video (4-dimensional).

For time-series data, successive data points are obviously more related than data separated by a year. Ditto for pixels in an image lying side-by-side or on separate edges. To model this knowledge in the neural network, one introduces a concept called weight sharing (Goodfellow et al., 2016). Weight sharing achieves two goals in the context of CNNs. First, the model can learn general representations of the data, without care for where spatially the features are located, i.e., a cat in an image looks similar if it is in the top right corner or the bottom left corner. Second, weight sharing allows the network to model more complex relationships while using fewer parameters than fully-connected equivalents.

Weight sharing happens through kernels of weights being convolved with the input in a sliding manner (see Figure 5). This way, the weights in the kernel are shared between all pixels in the input. Further, the weights can be used to find features in the inputs regardless of location, as wished. In short, the fundamental equation underlying convolutional neural networks is the discrete convolution (here for 2 dimensions) (Goodfellow et al., 2016),

$$
\begin{aligned}
S(i,j) &= (K * I)(i,j) \\
&= \sum_m \sum_n I(i-m, j-n) K(m,n),
\end{aligned}
$$

where I is the input image, K is the aforementioned kernel, and S is the output activation. This output activation is normally passed through a max-pooling layer (not covered) and a non-linear activation function, similarly to fully-connected networks (typically ReLU or related functions).

Bai et al. (2018) presents a generalization of Convolutional Neural Nets specifically for time series data modeling tasks.

2.2.8. Recurrent neural networks

Vanilla RNN

The fundamental idea of Recurrent Neural Networks (RNN) is similar to that of CNNs, except that instead of sharing parameters in space dimensions, it shares them in the time dimension. As such, RNNs are designed to be especially suited for

Illustration of the 2-dimensional convolution

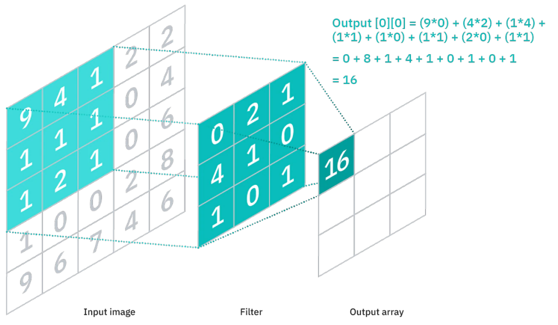

Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

Figure 5: A schematic representation of how the convolutional layer works in convolutional neural networks. The leftmost matrix is the input (e.g., image) and the 3-by-3 matrix is the kernel or filter. The operation involves sliding the kernel across the input image, element-wise multiplication of the two matrices, and storing the result in the output matrix to the right. One usually has several kernels for each layer and several layers for each model.

time series modeling tasks (Goodfellow et al., 2016). The canonical equations governing RNNs are as follows (Goodfellow et al., 2016).

$$\begin{aligned} a^{(t)} &= b + Wh^{(h-t)} + Ux^{(t)}, \\ h^{(t)} &= \tanh(a^{(t)}), \\ o^{(t)} &= c + Vh^{(t)}, \\ \hat{y}^{(t)} &= \text{softmax}(o^{(t)}) \end{aligned}$$

All the above variables are vectors and matrices (when superscript I omitted it implies time step (t)), where x is the input, a is the activation produced by multiplying the input with its weight matrix U and adding it to the previous time step's hidden state $h^{(t-1)}$ multiplied with its weight matrix W. b and c are bias terms, and o is the raw output, often transformed somehow before emitted as, e.g., with softmax for classification with multiple classes (e.g., word modeling).

Optimizing recurrent networks is not directly straightforward, and instead relies on a trick called unfolding (Goodfellow et al., 2016). Figure 6 shows a simple recurrent model and its unfolded parallel. When unfolding, one creates a version of the model that has a node for each time step one has given to the recurrent nodes, which are laterally connected instead of recurrently (see Figure 6). This results in an equivalent architecture to that of a feed-forward network, allowing one to apply standard backpropagation. This procedure is known as Backpropagation Through Time (BPTT) (Goodfellow et al., 2016).

Unfolding of Recurrent Networks

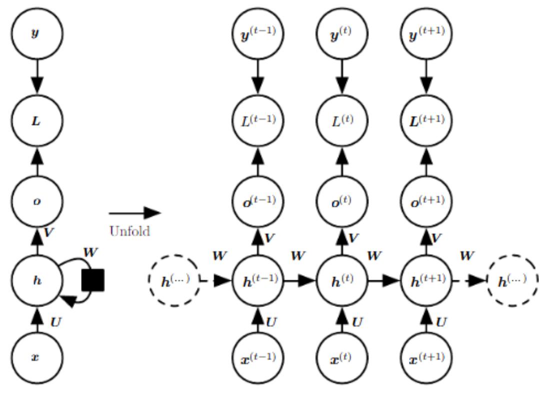


Figure 6: A very simplified model of a model with a recurrent connection to the left. This model combines each new observation received at x with the calculated activation from the previous time step, and can as such model temporal relationships. To the right, the figure shows how the model must be unfolded when one is performing optimization with backpropagation. In short, this means that one creates a copy of the model for each observation it received, and instead of it having a self-loop, it is connected to successive copies of itself. This constellation is equivalent to regular feed-forward neural networks and can be optimized as such. (Source: Goodfellow et al. (2016)).

One problem arises though when performing BPTT for long sequences. When optimizing, one calculates the gradients of the loss w.r.t. the model weights, which leverages the product rule of differentiation. For long sequences, the unrolling procedure creates very long chains of successive products, meaning that numbers that are either below or above 1, will either go exponentially to 0 or infinity, respectively. This is known as the problem of vanishing or exploding gradients (Goodfellow et al., 2016), and is addressed by LSTMs below.

LSTM

A Long Short-Term Memory network (LSTM) (Hochreiter and Schmidhuber, 1997) addresses the problem of vanishing or exploding gradients introduced when modeling long sequences with recurrent networks (Goodfellow et al., 2016). The gradients are kept more stable with the addition of a set of complex gates that allows the network to dynamically remember or forget information based on the current state of the network (Gers et al., 2000), which Figure 7 illustrates. LSTMs have proven very adept at modeling several different time series problems, e.g., speech recognition, machine trans-

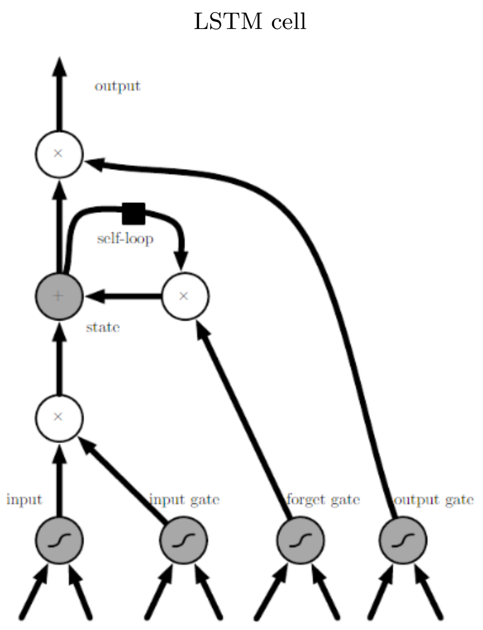


Figure 7: Block diagram of an LSTM cell showing the intricate network of gates, cells, self loops, and how they cooperate to enable networks composed of these cells to selectively store or remove information in the state. (Source: Goodfellow et al. (2016)).

lation, handwriting generation, and image captioning (Goodfellow et al., 2016). Other, simpler versions of LSTMs are also devised, but they have not received equally widespread adoption (Goodfellow et al., 2016).

2.2.9. Temporal Convolutional Neural Nets

Another, more recent solution to the problems of vanishing gradients in modeling of long time-series dependencies, is the Temporal Convolutional Neural Network (TCN), by Bai et al. (2018). This architecture combines ideas from both CNNs and RNNs, resulting in architecture with several desirable attributes. Similar to LSTMs, TCNs can model arbitrarily long dependencies in the input sequences. TCNs achieve this not through recurrent connections, but rather through a clever application of successive convolutional layers. Figure 8 shows an illustration of how the model works.

In short, TCNs combine layers of convolutions with increasing dilation in the kernels. Dilation is how much space there is between the weights in the kernel, i.e., with dilation 2, the weights in the kernel are multiplied by every other input data point. In the case of TCNs, dilation is leveraged in such a way that one doubles the dilation for each successive step, which enables the network to be connected to inputs arbitrarily far into the past while still covering all intermediate time steps. Again, see Figure 8 for a visual representation of this idea.

Bai et al. (2018) show empirically the effectiveness of TCNs over LSTMs on several time-series modeling tasks that LSTMs have traditionally been the best at. The authors also emphasize the ability to parallelize the training of TCNs to a higher degree than LSTMs because TCNs are not truly sequential like LSTMs. This ability allows for faster training with similar amounts of parameters.

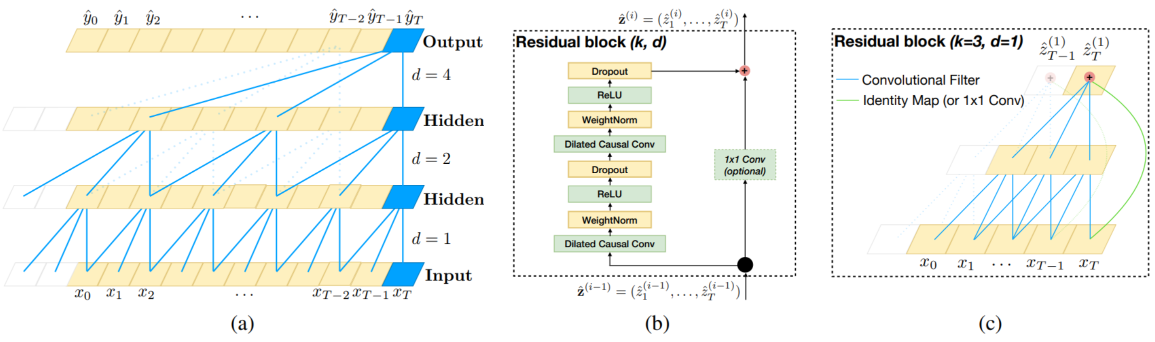
2.2.10. Dead neuron problem

An insidious problem in machine learning is the problem of dead neurons, which the authors of this paper also encountered. In short, the use of the ReLU activation introduces a problem. In Equation 12, one can see the derivative of ReLU. This takes the value zero when given negative input, which, in turn, when multiplied with the chain of derivatives during gradient descent, turns the whole chains to zero. When the derivatives are zero, one can no longer update the weights of the network, and the neurons that become zero are stuck at zero. One common solution is to use the related Leaky ReLU activation, which substitutes the 0 derivatives with a slightly negative one, which allows for weight updates, as shown in Equation 13.

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & \text{if}\, x \geq 0 \\ 0 & \text{if}\, x < 0 \end{cases} \tag{12}$$

$$\frac{d}{dx}\text{Leaky ReLU}(x) = \begin{cases} 1 & \text{if}\, x \geq 0 \\ -\alpha & \text{if}\, x < 0 \end{cases} \tag{13}$$

The architecture of the Temporal Convolutional Neural Network



$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T
(a)
$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \ldots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \ldots, \hat{z}_T^{(i-1)})$
(b)
Residual block (k=3, d=1)
$\hat{z}_{T-1}^{(1)}$ $\hat{z}_T^{(1)}$
Convolutional Filter
Identity Map (or 1x1 Conv)
x_0 x_1 ... x_{T-1} x_T
(c)

Figure 8: Schematic representation of the Temporal Convolutional Neural Networks as presented by Bai et al. (2018). (a) The leftmost figure shows how the receptive field can be regulated through layers of successive doubling of the stride in each 1D convolutional layer. Furthermore, the data is padded in such a way that the output at any given step is a combination of only past observations, so-called causal convolutions. (b) The middle figure shows how two and two stacks of convolution, ReLU, and dropout are grouped into what is called a residual block. The name stems from the residual connection that skips the convolutions to the right in the figure. (c) The rightmost figure shows that the convolutional layers within a residual block are of the same specification, i.e., have the same kernel size and stride.

3. Contributions

The contributions of this paper are twofold.

1. We present a total order of theoretical error bounds for forecasting paradigms and give an ex-ante and ex-post view of the methods' expressive abilities. We motivate that multivariate, non-parametric methods have the theoretical lowest bound loss of the models considered which provides room for performance gains in practice.
2. We provide a general forecasting framework, which can be readily adapted to solve different financial forecasting tasks. The approach is general in that it uses data cubes with dimensions of entities (companies), features, and time, which encapsulates most financial forecasting problems, and applies loss functions at the correct part of the model pipeline to greatly increase the flexibility of the range of financial modeling tasks it is able to solve.

Contribution 1. takes place in Part I. The theoretical lower bounds and expressive abilities of the models serve as the motivation to undertake the experimentation with the different forecasting paradigms in the general forecasting framework in Part II, fulfilling contribution 2.

To facilitate apt experimentation, we create a novel dataset of 104k quarterly fundamental reports, as well as daily stock data for 1846 companies, and several daily macro time series, all for the time period 2000-2022.

We use modern machine learning methods and compare them with statistical and naïve benchmarks to analyze the framework. We use this framework to attempt three distinct tasks:

1. We show better profitability than the benchmark for bankruptcy risk forecasting in terms of AUC and on-par results with another paper on the bankruptcy classification task. Using a profitability measure, exogenous models are found to be more profitable taking the asymmetric payoffs of loans into account.
2. Forecasting stock price volatility shows moderate results. Our models outperform the GARCH baseline and show some promising signs of learning market-wide variance dynamics but slightly underperform compared to the naïve baseline.
3. Forecasting market capitalization was, as expected, the least convincing of the results. This indicates that even using all quarterly reports of every listed company in the global oil and banking sector over 22 years, if pattern recognition is possible, it requires more resources than is available to us. Though no model outperformed the naïve benchmark, the multivariate, non-parametric models outperformed both the parametric models and the univariate, non-parametric model.

Part I

Theory on Forecasting Paradigms

	Parametric	Non-Parametric
Univariate	$f(y_t; c_f)$	$f(y_t; \theta)$
Multivariate	$f(y_t, X_t; c_f)$	$f(y_t, X_t; \theta)$

Table 1: Overview of the fourfold model classes. Somewhat abusing notation, the given $_f$ for each c in the Parametric column represents the assumption of the underlying DGPs functional form.

4. Forecasting Paradigm Total Ordering

This section constructs theoretical proofs of a total order of model selection, model combination, non-parametric univariate methods, and multivariate complex methods ex-post facto.

We begin by presenting formal definitions of the above. After that, we quantify the validity of the Hoeffding Inequality on a sizeable real-world time series dataset, i.e., the 48k monthly time series from the M4 Competition, displaying how the assumption of functional form can be erroneous.

We then construct a chain of total order ex-post-facto lower bound model losses. Simultaneously, we present empirical calculations indicating the validity of the corresponding ex-ante proposition.

4.1. Initial notation and definitions

With *functional form* we mean a combination of operators, parameters, and symbolic variables.

Let f be a function, c_f be parameters inferring a DGP, and θ be parameters relating to the learning algorithm and not a DGP itself. Let y_t be the dependent time series and X_t be m exogenous variables organized into a matrix of size $t \times m$. Then Table 1 shows the function in the tetrachotomy $(Univariate, Multivariate) \times (Parametric, Non-Parametric)$.

Table 1 shows the trivial difference of univariate and multivariate functions. Noted that we refer to arity and not output size in this definition of uni- and multivariate functions.

Furthermore, note that parametric univariate models make assumptions of the functional form through c_f, while the θ parameters do not make assumptions of f, including the data distribution.

This minor distinction is very important. We are reducing the forecasting space immensely by using parametric models, and imply that the functional form somehow imitates the underlying DGP of y_t.

In subsection 4.2, we will quantify how erroneous a simple assumption of the distribution of a set of y_t can be.

4.2. Quantifying the validity of the Hoeffding Inequality

For a particular time t and problem, an entity has an ex-post-facto view of previous times and an ex-ante view of the future. One can rewrite the famous Hoeffding (Hoeffding, 1963) Inequality in the following manner, where E is an error, $\mathcal{H}$ is the hypothesis class, and N is the size of an i.i.d dataset:

$$E^t_{ante} < E^t_{post} + \sqrt{\frac{\log|\mathcal{H}| + \log(\frac{2}{\delta})}{2N}} \tag{14}$$

With probability of at least $1 - \delta$

Equation 14 assumes that the underlying distribution is Sub-Gaussian. In Salinas et al. (2020) $E^t_{ante} = E_{out}$ and $E^t_{post} = E_{in}$ both are $\in [0, 1]$. How ever, this leaves the usability of the above formula rather slim: Indeed, in plenty real life situations assuming a Sub-Gaussian distribution is erroneous. Below we quantify how much reality deviates from this Sub-Gaussian assumption by applying Equation 14 to the M4 dataset (Makridakis et al., 2018a) consisting of 100,000 time series (TS).

By fitting an AR(1) model with a single coefficient, stored as a floating-point number of double precision (i.e., 64 bits), we imply $|\mathcal{H}| = 2^{64}$. For the entire set of time series, it is also straightforward to calculate E^t_{ante} and E^t_{post} by predicting the loss for the test set and the in-sample, respectively, as provided by Montero-Manso et al. (2020). In figure Figure 9 we divide the dataset into samples of size N with $\delta = 0.05$. As N increases, the Hoeffding equation fails to bound the error more frequently. As the rightmost addend decreases and

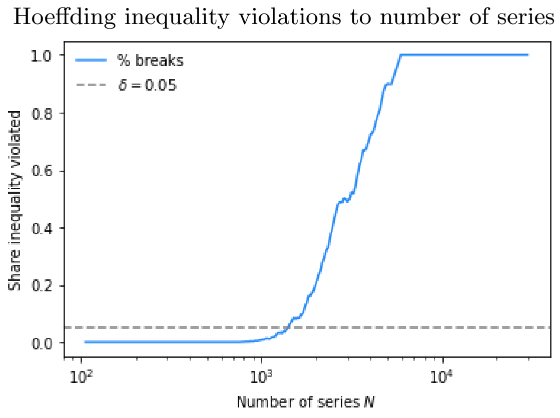


Figure 9: With $\delta = 0.05$, one expects the Hoeffding inequality (Equation 14) to hold in 95% of cases. However, for M4 monthly data, we observe that the inequality breaks down completely and is violated in 100% of cases, given N number of series large enough ($>\sim 4500$).

Equation 14 theoretically becomes more strict, reality in the form of E^t_{ante} does not comply with the bounds deduced from E^t_{post} and parameters.

Apart from the algebraic workings of Equation 14, why is the inequality so often broken in the practical example of the M4 monthly data? Trusting Hoeffding's mathematical ability, the error must stem from the assumptions of the equation. Precisely, the M4 monthly information is not Sub-Gaussian. In Figure 4.2, the kurtosis of the M4 series is around 967 times larger than a normal distribution with the same mean and standard deviation.

As seen in Figure 4.2, the tails of the real-world distribution are simply not represented well by the normal assumption. The number of Hoeffding violations is a direct consequence of the Sub-Gaussian assumption, the error seen in Figure 9 is thus a representation of how wrong normality assumptions can be in actuality. It is worth noting that the classic OLS model and its derivatives rely on such assumptions (often in residuals) for them to be BLUE (Best, Linear, Unbiased, Estimator). The BLUE property and its assumptions are established in the literature and a good discussion can for instance be found in Greene (2018).

4.3. The ex-post lower bound loss of model combination is less than or equal to model selection

Let $\mathcal{L}$ denote a loss function, i.e., a function that evaluates the performance of a function approximator. A relatively lower loss implies a relatively superior performance. $\mathcal{L}_L$ with the superscript L is the lower bound on the losses a class of function approximators can produce.

Ex-post

In mathematical notation the statement is:

$$\min_{f\in\mathcal{F}}(\mathcal{L}(y_{t+h}, f(y_t)) \geq \tag{15}$$

$$\min_{\mathbf{p}}(\mathcal{L}(y_{t+h}, \sum_{f\in\mathcal{F}} p_f \cdot f(y_t)) \tag{16}$$

$$\text{given} \quad (\sum_{f\in\mathcal{F}} p_f = 1) \wedge (p_f \geq 0 \quad \forall f \in \mathcal{F}) \tag{17}$$

Equation 15 shows our definition of model selection. Equation 16 shows our definition of model combination. It is also necessary to assume that neither approach has a rich deep neural network as part of its ensemble.

This is a trivial proof ex-post since for any f setting right hand side $p_f = 1 \implies f \in \mathcal{F}$ on the right hand side. In other words: model selection is a subset of model combination.

In Ankile and Krange (2022) we prove that ex-post model combination, as opposed to model selection, leads to a 10.4% reduction in optimal forecast MASE and a 5.5% reduction in optimal solution variance. This result is for the 100k univariate time series from the M4 Competition (Makridakis et al., 2018b).

This observed effect comes to fruition because ensemble combinations can reduce the bias introduced by one individual model by counteracting it with another model's bias in the opposite direction. Montero-Manso and Hyndman (2021) states that "The more accurate the individual models that enter the ensemble, the more accurate the results of the ensemble, so the target accuracy is that of the individual methods." This statement is correct for model selection but not for model combination, where adding a strictly worse single model might increase accuracy than a strictly better single model. This effect is exemplified in Figure 11.

Ex-ante

Although model selection has a higher loss than model combination ex-post, there is evidence that the same holds ex-ante.

Our DONUT combination model only predicts a model selection equivalent weight in $< 1\%$ of all its M4 forecasts (Ankile and Krange, 2022). Furthermore, DONUT outperforms FFORMA, the sec-

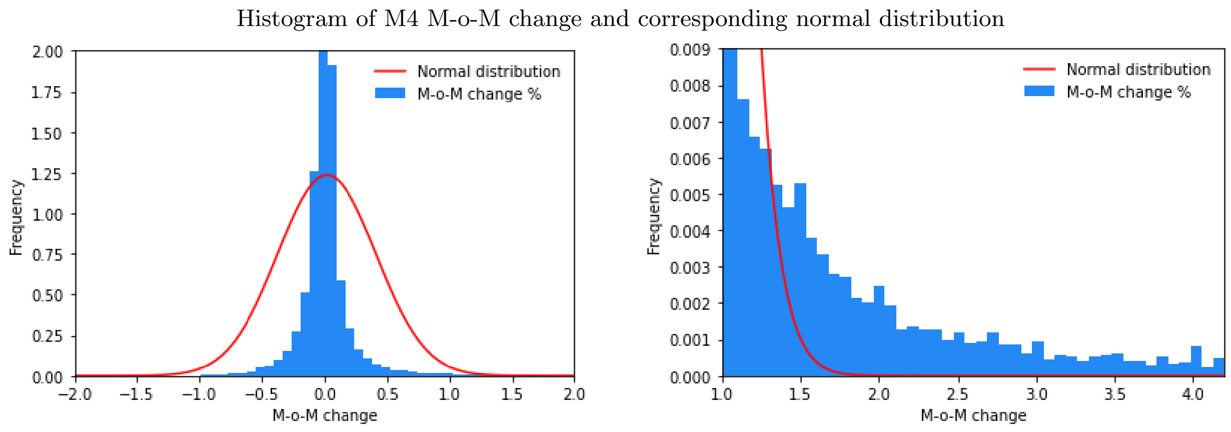


Figure 10: Histogram of M4 M-o-M and the normal distribution of the data mean and std. The kurtosis of the actual M-o-M change is 4798 >> 3 of the normal distribution.

ond placing M4 competitor (Montero-Manso et al., 2020), with a significant margin, further indicating model combination superiority over model selection.

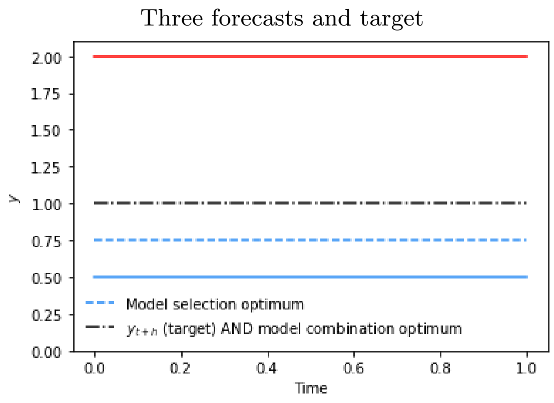


Figure 11: In this constructed example we have a target in dash-dotted black and an ensemble of 2 forecasts in solid red and blue. If we add the dashed blue line forecast to the ensemble the model selection approach lower bound loss decreased, how ever in both cases a model combination method has 0 lower bound loss

4.4. *The ex-post lower bound loss of non-parametric univariate methods is less than or equal to model combination*

Ex-post

In mathematical notation the statement is:

$$\mathcal{L}_L(y_{t+h}, f(y_t;\theta)) \leq \mathcal{L}_L(y_{t+h}, \sum_{f \in \mathcal{F}} p_f(entity(y_t)) \cdot f(y_t)) \text{ given (4)} \tag{18}$$

Montero-Manso and Hyndman (2021) argue that *“the property of the consistency of an estimator (or ‘universal approximator’ of neural networks) cannot be invoked as a rationale for the use of global models in the traditional setting (no single function exists which can approximate a set of time series in general).”* This is seemingly based on the notion of the *universal function approximator* from Hornik et al. (1989), as well as the *no free lunch-theorems* (NFL) of Wolpert and Macready (1997). This statement seems to disregard the caveat that the NFL theorem applies to the set of all possible optimization problems. At the same time, time-series forecasting probably could be regarded as one class of problems and hence can be solved by a single algorithm.

An existence proof of the above is the work done by Oreshkin et al. (2020). Here, the authors train their N-BEATS model (Oreshkin et al., 2019) on the FRED dataset, which contains 290 thousand US and international economic time series published by the Federal Reserve Bank of St. Louis. This model is then used to forecast the 100k time-series in the M4 Competition data set and achieve a similar result to Montero-Manso et al. (2020), the second-place winner of the competition. This result illustrates that transfer learning could leverage general patterns in time series and have the regularizing power (anti-overfitting).

Furthermore, we argue that choosing an ensemble approach is equivalent to making assumptions about the underlying Data Generation Process (DGP). In the latter case, the choice of $\mathcal{F}$ in a model ensemble will result in a lower bound loss as a function of the models it consists of and the time series at hand. However, a non-parametric model of sufficient size does not imply such a constraint.

Ex-ante

N-Beats, a model which only consists of NNs, can train on one set of time series (the *source* set) and then transfer this learning to other groups of series (the *target* set), out-competing state-of-the-art models on the target set (Oreshkin et al., 2020). For example, when N-beats used M4 as a source set, the algorithm beat the winner of the M3 target set (5.7% sMAPE improvement). This improvement suggests that even though no single function can approximate a set of time series in general, non-parametric univariate methods out-compete some of the best parametric models on a single series. We also infer that several different time series elicit the same types of behaviors, which is an argument for cross-learning.

4.5. The ex-post lower bound loss of non-parametric multivariate methods is less than or equal to Non-parametric univariate methods

Ex-post

In mathematical notation the statement is:

$$\mathcal{L}_L(y_{t+h}, f(y_t, X_t;\theta)) \leq \mathcal{L}_L(y_{t+h}, f(y_t;\theta)) \quad (19)$$

We believe an NN needs the correct information to approximate and distinguish between underlying DGPs. Our main postulation also witnessed in the excellent performance of recent multivariate approaches (see: subsubsection 1.1.4) is that adding more variables as input to a forecasting algorithm will increase the accuracy of predictions.

There is no guarantee that the values of a time series contain the most crucial information for bias reduction in a forecast. For instance, any equity investor will look at many variables before investing, e.g., market growth, revenue growth, costs, and interest rates, when the dependent variable is market capitalization.

Ex-ante Different variables can hold the information necessary to deduce a pattern (non-linear correlation) in a data set.

For a univariate approach, as our training dataset approaches infinity, the formula which solves for the NN's optimal forecast is given below, in Equation 20.

$$\begin{gathered}\lim_{N\to\infty} \mathcal{D}_N^{tr} \implies NN(y_t): \\ \min_{NN(y_t)} \sum_{i\in\mathcal{D}_N^{tr}} \mathcal{L}(y_{i,t+h}, NN(y_t))\end{gathered} \quad (20)$$

Empirically it is easy to prove cases where multivariate methods outperform univariate methods. We create a fake dataset where the signal is identical for all series and while ex-post continuation is one of three time series. More formally, we made $\mathcal{D}_N^{tr}$ and $\mathcal{D}_N^{val} \in \mathcal{D}_N = \{y_{1,t}, y_{2,t} \cdots y_{N,t}, y_{1,t+h}, y_{2,t+h} \cdots y_{N,t+h}\}$ where $y_{i,t} - \epsilon_i = y_{j,t} - \epsilon_j | \epsilon_i, \epsilon_j \sim (0, 0.1) \forall y_i \neq y_j \in \mathcal{D}_N$. Thus, a neural network training on the signal will converge towards Equation 20. With three possible values $F = \{f_1, f_2, f_3\}$ y_{t+h} can take in this setup: $\mathbb{P}[y_{i,t+h} = f_j | y_{i,t}] = \frac{1}{3} \forall i \in N \land \forall j \in |F|$.

We then add a variable x as part of the input to a neural net where $\mathbb{P}[y_{i,t+h} = f_j | x_i] = 1, \forall i \in N \exists j \in |F|$. Figure 12 shows the result of this training on the validation set. Thus, the net receives all information needed to predict the future perfectly through the newly introduced variables. We plot the loss curves for the two implementations of a model (with and without the exogenous variable x) in Figure 12. This experiment further reinforces the claims above regarding the lack of expressive ability in univariate models.

This constructed proof (Supported by computation) shows that there are multivariate methods that outperform univariate methods. However, does the opposite claim hold: Can univariate methods outperform multivariate ones? While this can hold in practice, we note that a multivariate model has the option to set all but one variable influence to zero producing a univariate model (In the in-

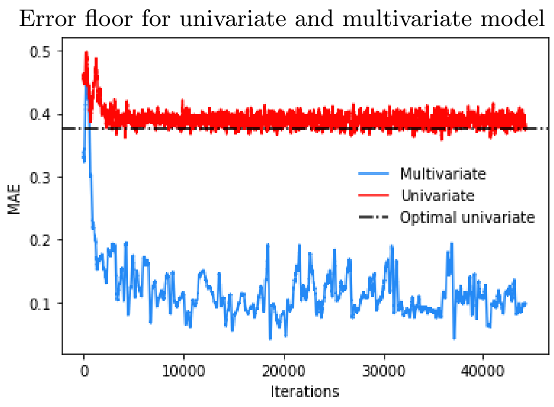


Figure 12: It is evident that the multivariate model quickly learns to use the highly relevant information in x, rapidly outperforming the univariate model. The univariate model converges to the black line which is the solution of Equation 20 where $\mathcal{L}(\cdot) = \text{MAE}(\cdot)$.

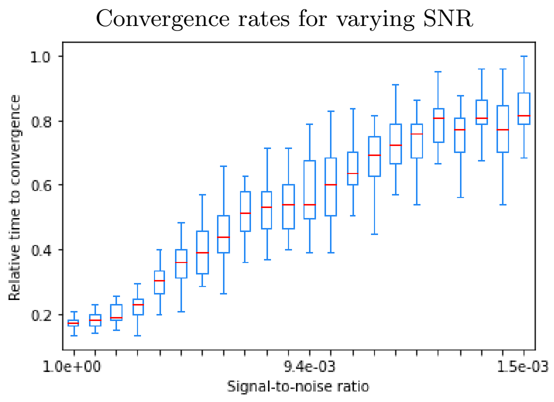


Figure 13: The leftmost column shows that when the number of lags used, $|\mathbb{A}|$, equals the number of exogenous variables, $|\mathbb{E}|$, the model finds a better solution than the optimal, univariate solution 100% of the time. However, as the number of lags increases, and the signal-to-noise ratio (SNR) with it, the time needed for the model to converge past the local minimum increases. The whiskers also indicate that the variance in convergence time increases with SNR

stance of an NN, set all input weights not related to one variable to zero). Thus the lower bound loss of a multivariate model cannot be higher than its univariate counterpart.

There is, however, a pitfall in the above line of argument. One needs to make sure that the information added to the model is appropriately weighted compared to the existing information. I.e., if one constructs a univariate, autoregressive model, and one were to add the set of exogenous variables $\mathbb{E}$ to the set of autoregressions $\mathbb{A}$, if $|\mathbb{A}| >> |\mathbb{E}|$, one runs the risk of the model disregarding the new, important information. Figure 13 exemplifies this problem. The figure shows how the model struggles increasingly to utilize the useful information as the ratio between the number of exogenous variables and lags decreases. Therefore, we argue that one must take care when constructing multivariate models to make sure the model of choice is enabled to learn the important relationships in the data.

Z. False Local-Global Dichotomy

(Montero-Manso and Hyndman, 2021) discuss the dichotomy between local and global models and analyze some properties of the approaches. However, we would argue that this dichotomy might be fundamentally meaningless and might rather depend on if globality stems from cross-learning done explicitly by machines or implicitly by humans.

Furthermore, the literature seems to confuse local models with global models. In most 'local' approaches, the meta-learner (i.e., the process defining, e.g., the functional form of a model) is a human being. This fact is in opposition to the arguments by Montero-Manso and Hyndman (2021), i.e., that local models are fitted only to the forecasting problem at hand, treated as a '[...] separate regression problem of finding a function to predict future values from the observed part of the series'. An example of this is the ARIMA class of models.

To elaborate on the above point, we think there is a flaw in the notion of a 'local' model. One such erroneous conclusion is classifying an ARIMA model fitted to a single series as a local model. An ARIMA model incorporates many assumptions and ideas that humans have arrived at through years of research. When fitting the model on a single y_t, the resulting model is a product of a human meta-learning system and fine-tuned on the time series at hand.

It is unlikely that any truly tabula rasa model would decide to fit an ARIMA model to time series. However, human researchers and the humans implementing the model have observed many time series, leading them to develop the idea of autoregression, integration, and moving averages. People then embed these ideas into the models. Furthermore, they use cross-learning concepts, such as seasonality and trend, to discern whether ARIMA is an appropriate model for the task at hand.

A local model can only be genuinely local (fitted only to the task at hand) iff we feed a tabula rasa model only a single time series used for predicting future values of the same series. The time series would have to be very long for this to make sense.

Part II

General Financial Forecasting Framework

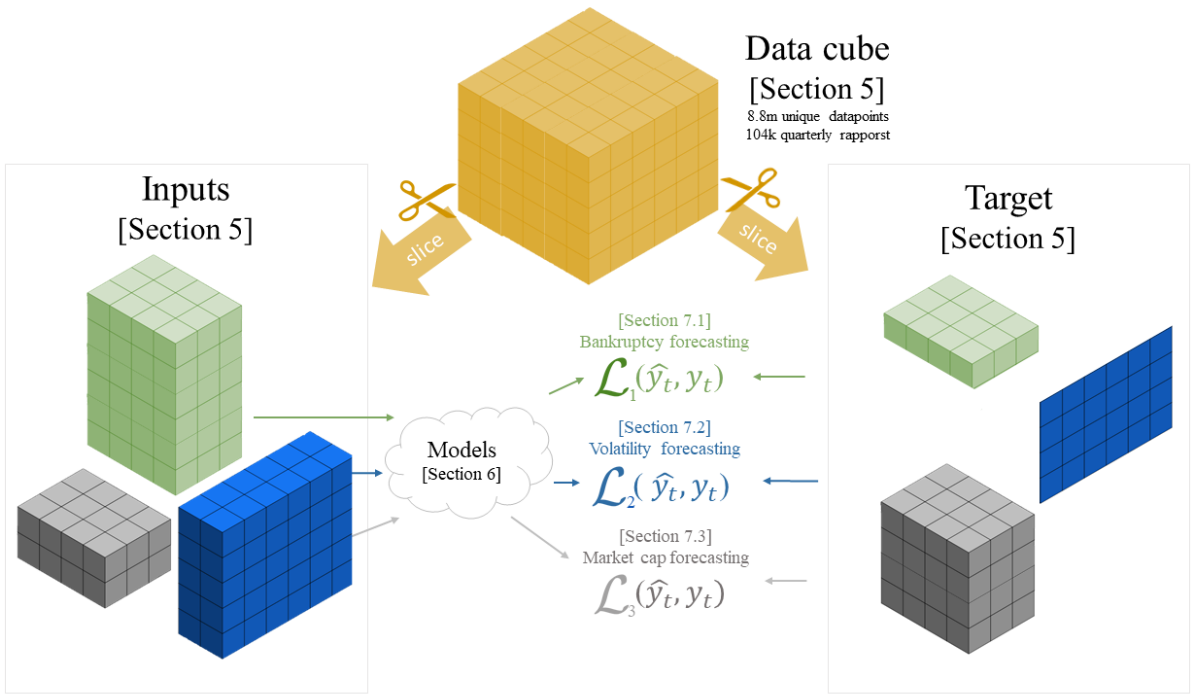


Figure 14: The overview of our general forecasting framework. The large datacube is sliced into input and targets of different combinations, which are then passed to neatly defined loss functions to quickly model many different forecasting scenarios. One can select from a plethora of models and data to explore many different problems fast in any combination of features, time, and companies. As an example, we select balance sheet data across all times for the target, and whatever inputs we want e.g. fundamentals, stock prices, and worldwide risk-free rates to model the bankruptcy classification task.

5. Dataset

5.1. Data overview

Our dataset is novel, compiled by the authors, consisting of a large quantity of fundamental and stock data and, to a lesser extent, macro time series and company metadata. The main series to predict is the stock prices (or market cap) of 1846 global oil and banking service companies, with data from January 2000 until April 2022. In addition, we have collected primary fundamental data from 104k quarterly reports over the 22 years. Finally, we have included several related macro financial and commodity price time series, e.g., interest rates and oil prices, used in the multivariate models. In total, there are 8.7M data points in the dataset.

We want to measure our forecasting problem's accuracy for forecasts around a month into the future. Therefore, with around four weeks per month and five trading days per week, we use a horizon of 20 periods, i.e., days in this case.

The last 20 data points for each time series will be put away for testing and comparing the different approaches. Here, we ensure that a specific point in time restricts where the training data stops and where the testing data starts. This restriction makes sure we under no circumstance have any data leakage, i.e., data from the future implicitly being available in the past.

We will also split the training set into training and validation sets used for tuning training parameters for the non-parametric models.

Using fundamental accounting data on a quarterly basis is not something that is frequent in financial studies using deep learning applications due to its low-frequency Jiang (2021). We tackle this by using a large amount of data with the Refinitiv Eikon database, and by structuring the data into a cube as seen inFigure 15. This way, a model many financial updates for a one-time era.

5.2. Missing data

An obstacle in this work is the number of missing values and nonsensical entries in the datasets we combine. Therefore, collecting and cleaning data in the best possible manner, given the state of the data, has constituted a considerable portion of the efforts required to produce this thesis.

Jiang (2021) argues that missing data in financial applications is not as severe as in other domains since it is often collected by trustworthy sources and that logging accurately is something that is part of the nature of market data. Still, he recommends using forward filling to match different frequencies of data, when this leads to missing gaps in the time dimension. Thus, we use forward filling actively in our data pipeline, representing 'the latest available information.

In the following, we present our data cleaning step by step and show how much the data cleaning process reduces $\mathcal{D}_T = \{\forall \text{data points} \neq NA\}$, or the amount of data from its original size.

To summarize, we remove companies that meet certain thresholds of NA data points, with an emphasis on removing data when in doubt to make sure we are left with a trustworthy and sensible dataset afterward. Figure 16 shows our data processing pipeline. The Sankey diagram is a visualization of the data cleaning process in detail below.

1. *Cleaning stock data*
   Our 'stock' data set contains all data relating to the market capitalization of firms.
   We apply two filters in the cleaning of stock data. We begin by removing missing values resulting from collecting dates pre-listing or post-delisting of a given ticker, i.e., we trim NA values at both ends of a time series. Secondly, we remove all tickers with chains of NA values that are longer than five days, as seen in Figure 18. We lose about 6% of companies and data points in the process.
2. *Cleaning fundamental data*
   Our 'fundamental' data set contains 12 columns comprising fields in all three financial statements (profit and loss, balance, cash flows) every quarter.
   We delete all companies in the fundamental dataset without a single entry in the 'announcement_date' column. We do this because we can not use information if we do not know when it entered the market (to safeguard us from 'leaking' information from the future into the past). We also remove odd quarterly numbers where revenue is negative, or EBITDA exceeds revenue in a quarter. Finally, we can all financials where EBITDA/Revenue exceeds -500%. Figure 16 shows all of the mentioned cleaning processes from the loss from Eikon fundamentals.

Conceptual representation of the datacube

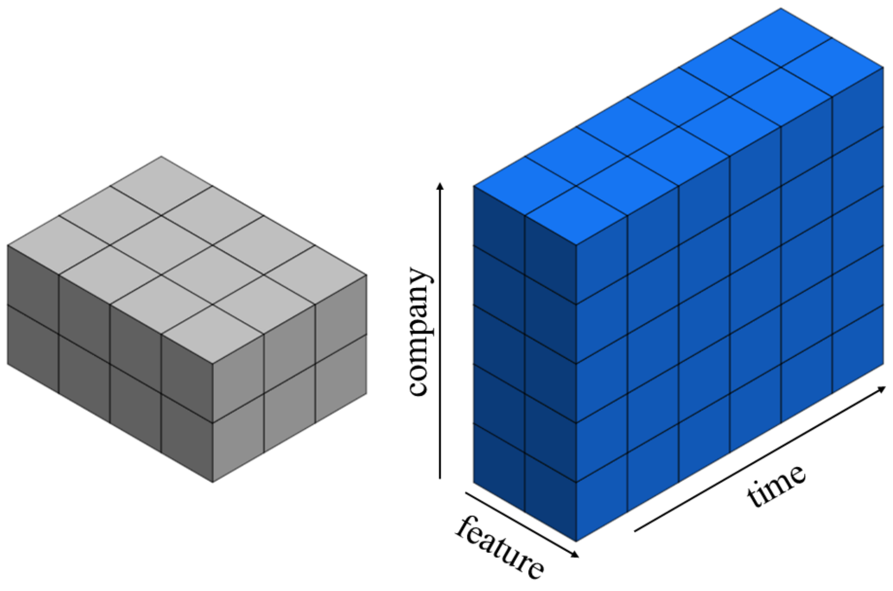
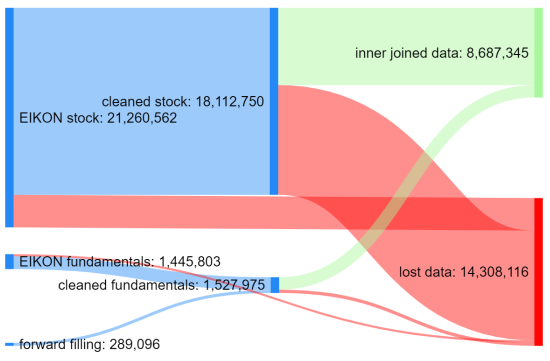


Figure 15: Overview of the data as it is used in a practice. The two colors represent two different problems, where the input shape is fitted to the data one wants to find trends for. In the case above, the grey problem has fewer time entries and companies, but more features compared to the blue case. These cubes are restructurings of a given $\mathbb{E}_{T-k,T}$, displayed on the previous page. Note that this implies that when we are later working with means and medians across distributions, we have to define what dimensions they are across. In our program, they are implemented as PyTorch tensors.

Sankey diagram showing data processing



inner joined data: 8,687,345
cleaned stock: 18,112,750
EIKON stock: 21,260,562
EIKON fundamentals: 1,445,803
cleaned fundamentals: 1,527,975
lost data: 14,308,116
forward filling: 289,096

Figure 16: Sankey diagram showing our data pipeline from the collection from Refinitiv Eikon database to our inner joined dataset across fundamental and stock data. The blue color is raw data as collected from Eikon. The red color is the data we lose due to our filtering process which seems noisy.

Share of entries per day from the year 2000

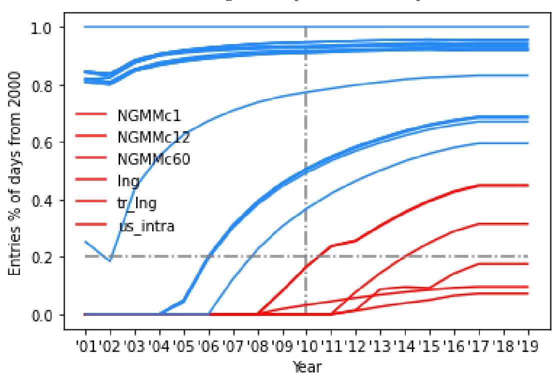


Figure 17: Red color indicates the time series we have removed due to too little signal before the year 2010. Blue indicates the macro series we keep. We drop all LNG series and the US intra-bank lending rates because of their high percentage of missing values. Our limit was at least 20% signal in 2010.

After the above cleaning process, we still have sporadic missing values. We forward fill all of these (i.e., replace a missing value with the entry before it), so a model always uses the latest available data.

3. *Keeping the intersection of the datasets*
   We only have stock prices for some companies, and we only have a few fundamentals for others. Combining the dataset through an inner join is necessary to ensure the resulting dataset is complete. This inner join leads to a loss of around 2 million data points, as seen in Figure 16.
4. *Cleaning macro data*
   A problem with the macro data time series is that the frequency of entries increases sharply with time. We reason that a signal cannot enter the dataset too late since a neural network is likely to build a forecasting algorithm independent of a signal that later enters the training loop. Thus we limit the signal-to-noise ratio to having reached 20% from 2000 to 2010. The y-axis gives the signal-to-noise ratio in the data from the beginning until the point on the x-axis. So for a given point on the x-axis, we can read off how much data is available to the model. As shown in red in Figure 17 all LNG series and the US intra-bank lending rates are dropped because of their low signal-to-noise ratio. The same reasoning holds for the Intercontinental Exchange Index German NCG Natural Gas Electronic Futures (NGMMcx=NGMM x month forward).
   As a side note, we find this positive correlation between year and amount of data a problem that is little discussed in the literature but is essential to address appropriately.

The collected metadata has few missing values and is trivially added to the inner company-specific data, increasing the total number of data points by around 1000 rows $\cdot$ 9 columns = 9000 data points.

Our final combined dataset has a size of 8.790M data points, 1846 companies, and 104,178 annual rapports, with an average of 8.8 entries per report (11.6 counting forward filled entries and added entries of posts as a percentage of revenue).

5.3. Training data

We have two objectives for our training data set: 1) We need the data to *maximize knowable signal*, i.e., where maximize signal means that we want to

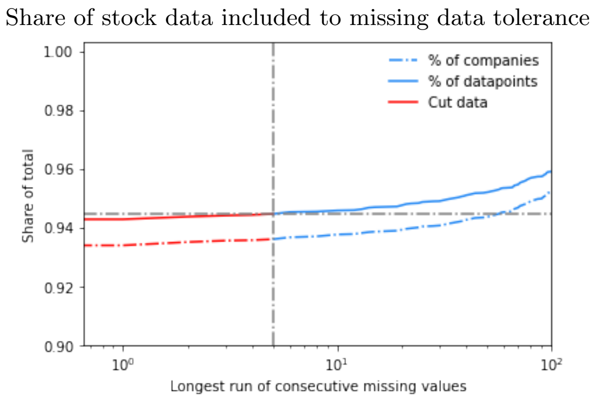


Figure 18: We allow 5 consecutive Na values as marked by the dash-dotted vertical line. Note that imposing 100 NA values would only lead to a 2 percentage point increase in companies and data points.

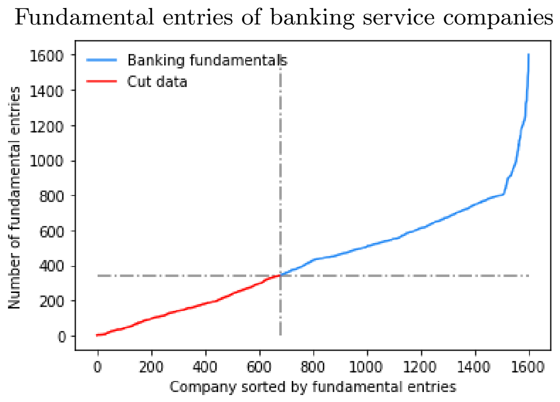


Figure 19: We select the 923 companies from the banking sector with the most entries of fundamental data since 2000. Red indicate the ∼700 companies which we drop due to this restriction.

extract as close to all signal that exists as possible and knowable means that no knowledge from the future leaks into the past. 2) It needs to be normalized for proper training.

Simultaneously satisfying these two objectives leads to some difficulties that are non-trivial to solve, which we discuss below.

We want our training environment to handle a T ='current time'. At time $T \leq \mathcal{T}$ = last time of train/test set, 2019/2022, respectively. We make available the k last macro and market capitalizations and the last p announced quarterly reports up to and including time T. We define the data we allow our models to use as $\mathbb{E}_T(k,p)$. Figure 22 shows an overview of the three final data frames we construct, which handle the two above objectives and are thus $= \mathbb{E}_T(k,p)$. We choose the $\mathbb{E}$ notation to allude to subsubsection 6.3.3 'Era' concept. this data segmentation's 'time window' approach. Note that the 'macro' data is implicitly defined as it is not dependent on k, p or T, which is also why the metadata frame is not part of the stock&fundamental data frame, as this approach would result in storing redundant information.

In our training, we divide history into rolling windows of $k = 240$ trading days (i.e., roughly a year back in time). Partitioning leaves us with a table of 240 stock prices. In $STOCK\&FUNDAMENTAL$ we include the last $p = 4$ quarterly reports of fundamental data (m post including posts from the balance, p&l, and cash flow statement) $\implies \mathbb{E}_T(240,4)$. We have $c = 19$ macro time series in $MACRO$, also with 240 lags from T. We have metadata in $META$ of all companies containing 4 company location metrics, gradually increasing in specificity, the founding year, and 5 sector codes also progressively more exact.

We normalize all fundamentals in the p&l and cash flow statements by dividing them by quarterly revenue. Next, we add columns dividing the absolute value numbers from the two above financial statements by the firm's market cap at T. Finally, we normalize all balance sheet numbers by total assets. These three data sets represent the fundamental data in $\mathbb{E}_T(240,4)$.

We add a column to our data frame of the last market capitalizations min-max normalized across all companies for a given $\mathbb{E}_{T-k,T}$. This normalization gives the neural network information on a given company's market cap relative to its peers within a training window.

Next, we add a column with the last market cap divided by the market cap of Apple Inc. as of 05.05.2022 (BIG_NUM in Figure 22). This normalization procedure allows the model to have information about firms' size in absolute terms over time. Furthermore, with this column, the companies' total size is deducible for the learning entity since we divide the columns of fundamental data by the firm's market cap at T).

Note that we do not divide by the max market cap of all our stocks, as this could lead to (possibly non-negligible) data leakage and could corrupt our results. We have found minimal discussion regarding correct normalization in multivariate time series forecasting literature. Nevertheless, normalizing according to principles 1) and 2) is something we believe is fundamental to proper multivariate forecasting and needs establishing best practices.

Then, we normalize each macro series in $MACRO$ by its maximal value across $\mathcal{T}$, rounded up to the nearest 100. This *could* have resulted in data leakage; however, we believe the upwards rounding to the nearest 100 restricts the information about the future we give the model, as there is a many-to-one mapping between actual future max value and normalization.

The next issue is for the $METADATA$ frame, which contains constant information about the companies to be categorized. Furthermore, we need to represent somehow the tree-like category structure of both the location and business sector values. Although one could encode the information of the categories as a binary string, this is not advisable as the model would have to learn to decode this representation before it can use the information within. Thus we opt for categorizing the using PyTorch embeddings.

In the normalization procedures, one must account for missing values (NAs). For example, if a single total assets value is missing for a single quarter, this would lead to missing balance sheet information for this quarter. E.g., if 'total assets' is missing, 'total liabilities' divided by 'total assets' is also undefined. This project's share of missing data makes it unreasonable to remove all NA values. We think missing data is a part of nature and models need to adapt. For instance, one could forward fill missing data, but there would still be problems with NA values if, e.g., the 'total liabilities' first appear year $T + n$. The above is possibly just one out of many ways NA values eternally permeate real-world data despite one's best effort to rid the world of it.

Furthermore, since a single NA value is enough to abend the training procedure, we need a robust way to handle them. Therefore, we chose to set missing values to 0. Then, we rely on the model learning that a 0 in the input is equivalent to missing data. We think this makes sense as it resembles the regularizing approach in NNs, known as *dropout*. This procedure randomly zeroes out inputs to help the model be more robust against noise and prevent overfitting.

In practice, our data is shaped in a cube format with dimensions (company, feature, time) as seen in Figure 15. Our neural networks are fed one company's data at a time and are given one entire table of features over time. This is a great structure for particular TCNs, however, it is also a generalization of the input structure and works for many other models as the cube can be sliced into the particular data one wants to work with. Slicing at feature and company could for instance be a peer valuation metric, or slicing at two features and a time could be a test for co-integration.

5.3.1. Bankruptcy data

The bankruptcy data is perhaps the data slicing that needs the most care to make right. Firstly in Figure 20, we can see that the number of quarterly rapports that by our definitions are classified as bankrupt is increasing. This is perhaps fine as all recordings seem to be correlating positively with time. However, it seems quite volatile. Thus, we in Figure 21 see the same number in terms of % bankruptcies per year. This number is much smaller than the 10% of the fundamentals-only view. This leads us to suspect that "bankrupt" companies report many times after we would have classified them as bankrupt. We handle this when predicting and fitting the models.

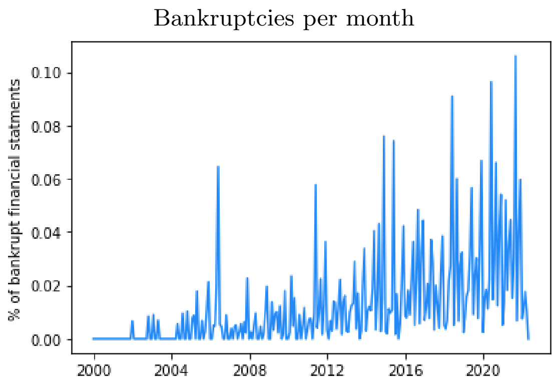


Figure 20: Display of the percentage of bankrupt fundamentals over time. It is increasing and quite volatile. Although there are some double counting as companies have multiple year internal statements which return bankrupt according to our definitions.

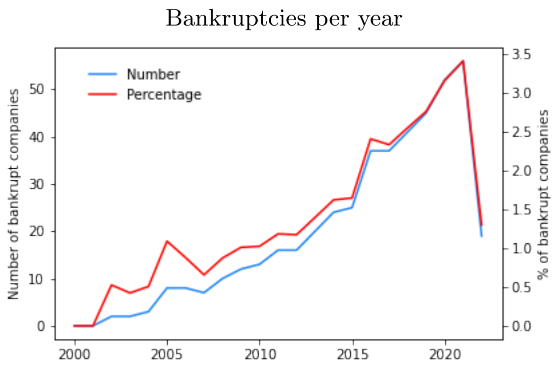


Figure 21: Display of the number and percentage of bankrupt companies per year. The last dip is likely explained by the fact that April 2022 is the end of our data.

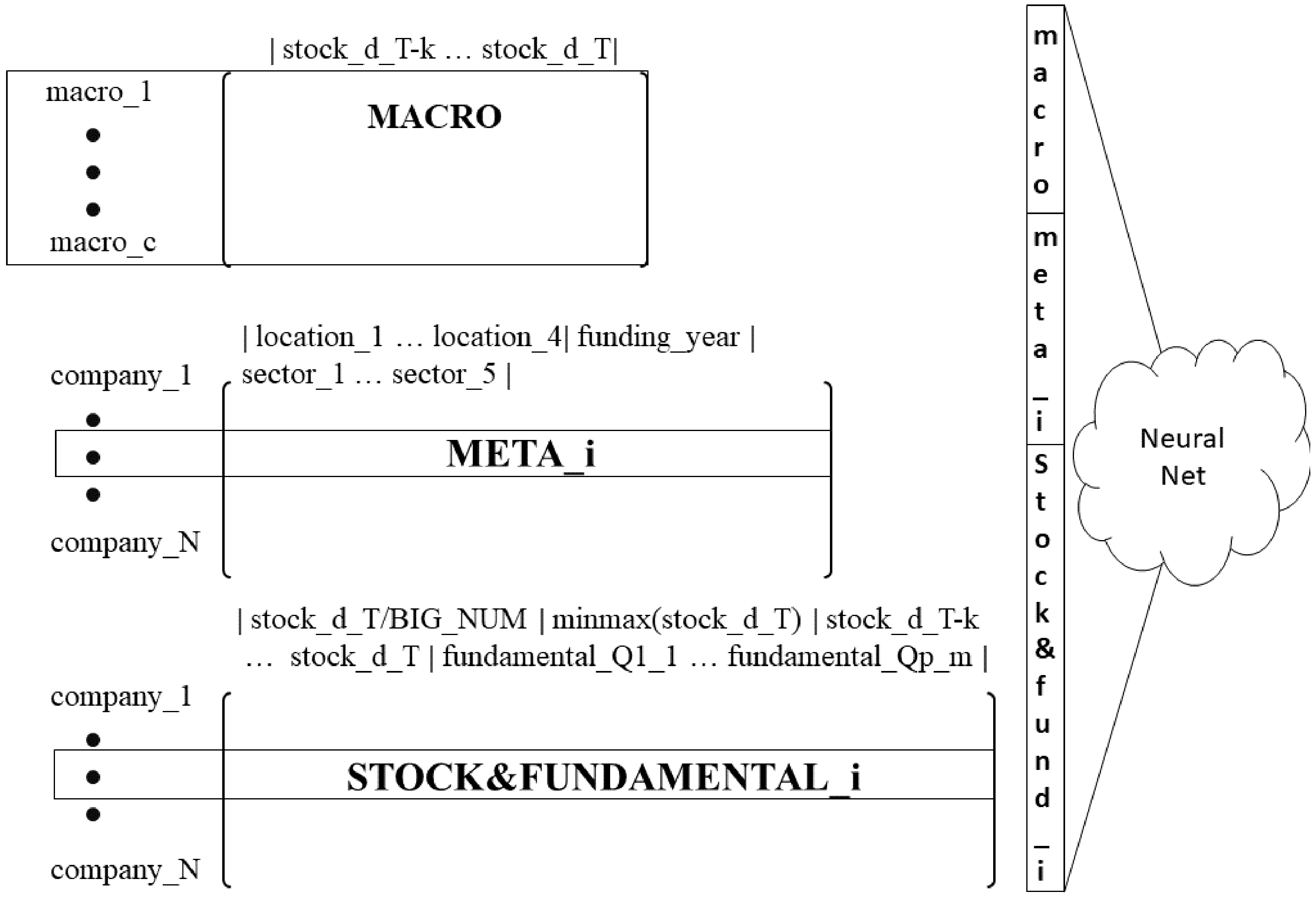

| stock_d_T-k … stock_d_T|
macro_1
macro_c
MACRO
| location_1 … location_4| funding_year | sector_1 … sector_5 |
company_1
company_N
META_i
| stock_d_T/BIG_NUM | minmax(stock_d_T) | stock_d_T-k … stock_d_T | fundamental_Q1_1 … fundamental_Qp_m |
company_1
company_N
STOCK&FUNDAMENTAL_i
macro
meta_i
Stock&fund_i
Neural Net

Figure 22: Overview of the data frames, for a given $\mathbb{E}_{T-k,T}$ the information in the each of the three boxes are concatenated for company i, creating input tensor for company i.

6. Empirical Methods

6.1. Models

This section provides information about our chosen models for each model paradigm we investigate. We have three forecasting problems we attempt to solve in this paper, and below we list the models we use for these categories. As a general baseline model to compare all models, we utilize the Naïve model (see section 2).

6.1.1. Parametric, Univariate Models

In the fundamental forecasting approach, we use Logistic Regression as a benchmark. The bankruptcy prediction problem is that of a classification problem, and Logistic regression is one of the most influential and standard models in this setting. We use scikit-learns implementation of the model, with the parameter 'class_weights' set to 'balanced'. The last parameter is to weigh the Bankruptcy weight as much as the non-bankruptcy prediction even though the latter is much more frequent in the dataset as discussed in subsubsection 1.2.1

For the variance prediction problem, we fit (68,092) simple GARCH(1,0,1) models, with an input window of 1 business year, predicting 1 month ahead of volatility measured in terms of standard deviation, in a rolling window fashion. We chose this partly because Brooks (2019) states that 'a GARCH (1,1) model will be sufficient to capture the volatility clustering in the data...'. Furthermore, there is no online Python implementation of Auto-GARCH we have found. Thus, we fit many models, one for each era in the test set over time.

For the market capitalization prediction, We use the ARIMA model to represent the parametric, univariate class. This model is widely used in the literature and has shown itself as a reliable model in many domains, including financial forecasting. In addition, these models also have direct analogs in the multivariate paradigm, to which we can compare performance. The specific implementation of ARIMA we have used is the AutoARIMA class from the open-source package for Python, pmdarima (Smith et al., 2017–). Notably, this is different from the rolling window methodology for the GARCH models. Where we refit the model for each training window, then use the estimated parameters to forecast the test set. This is a qualitative weighting of AutoARIMA's superiority to hard coding the GARCH lags.

6.1.2. Parametric, Multivariate Models

To facilitate apt comparisons between the univariate and multivariate approaches for the parametric paradigm, we implement an ARIMA model with exogenous variables (ARIMAX). The principle is the same as for ARIMA above, and we use the software implementation mentioned above, i.e., Smith et al. (2017–).

The vital difference is that, in addition to having coefficients for autoregressions and moving averages, it also has coefficients for the exogenous variables, one for each. These coefficients allow the model to adapt to changes in the environment in which the time series resides. For a drosophilic[2] example of this, see Figure 23.

See subsubsection 2.1.1 for a more in-depth explanation of the ARIMA models.

6.1.3. Non-Parametric, Univariate Models

We use LSTM and TCN models as described in section 2. They are fit to the same input as the target.

6.1.4. Non-Parametric, Multivariate Models

We use LSTM-X and TCN-X models as described in section 2 as our non-parametric multivariate models. These models do not have a restriction on the input space as a function of the output space.

6.2. Accuracy Measures

6.2.1. Time series regression measures

For our primary accuracy measure, we have opted to use the Mean Absolute Percentage Error, or MAPE for short. Armstrong and Collopy (1992) defines MAPE (essentially) as in Equation 21.

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{|y_t - \hat{y}_t|}{|y_t|} \cdot 100 \qquad (21)$$

MAPE is suitable for several reasons in the context of stock price and volatility prediction. First, because the number it produces is interpretable as the percentage one misses, it is easy to reason about. E.g., if a model has a MAPE loss of

[2]Meaning the smallest or simplest possible example that illustrates the effect.

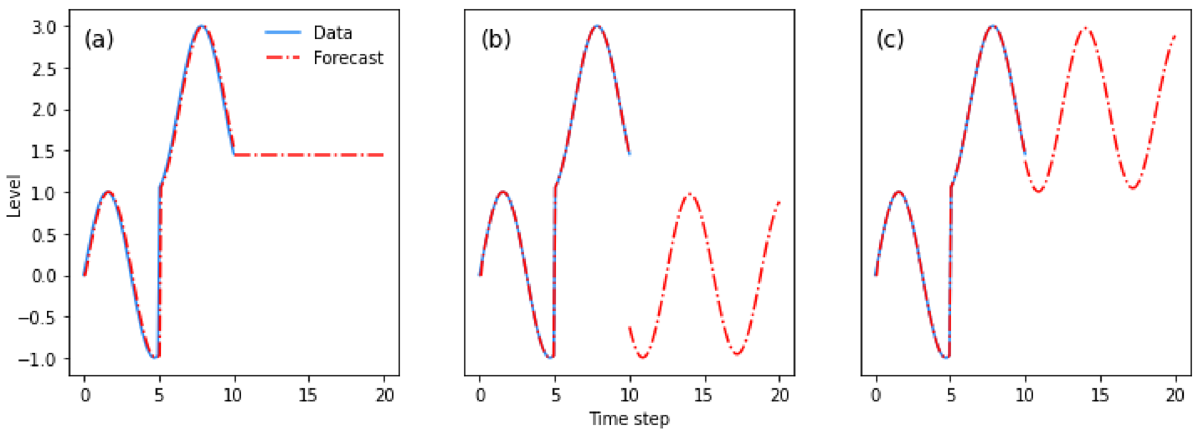


Figure 23: Illustration of how an ARIMA model with exogenous variables can adapt to changes in the environment when the same purely univariate ARIMA model breaks down. Panel (a) shows how a sine signal jumps at $x = 5$ and continues parallelly shifted. The best the ARIMA manages to do is degenerate into a naïve prediction. Panels (b) and (c), on the other hand, show how, when we include a single state-indicating variable, the model can adapt to the changing environment perfectly ((b) being low state and (c) high state).

0.05, it means that, on average, it misses the target value by 5%. Second, since errors are relative, one can straightforwardly compare loss values for different time series, despite market capitalization having vastly different magnitudes from company to company. Lastly, MAPE is widely used in the financial forecasting literature (Gandhmal and Kumar, 2019), facilitating cross-study comparisons.

We also utilize the related symmetric Mean Absolute Percentage Error (sMAPE), which is defined in Equation 22. This measure has many of the same attributes as MAPE, with the exception of it being more stable when the true future value is close to zero (since one also divides by the predicted value as well). On the other hand, it is somewhat biased in that it penalizes over-prediction and under-prediction differently (Chen et al., 2017a).

$$sMAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \cdot 200 \qquad (22)$$

Chen et al. (2017a) also presents more complex measures, like UMBRAE, which elicit several desirable attributes. We have still opted to use MAPE to allow for more straightforward and meaningful interpretation and comparison.

6.2.2. Bankruptcy accuracy measures

To predict bankruptcy we use the formal Norwegian definition of bankruptcy discussed in subsubsection 1.2.1. In short, we use Binary Cross Entropy implementation where the task is to for across the training time window, return true or false whether or not the target can be classified as defaulted, seen below:

$$\begin{aligned}
\text{Insolvency}(y_t) &= \mathbb{1}[y_{t,liab} > y_{t,assets}] \\
&\exists t \in \mathbb{E}^{val=1year} \\
\text{Illiqudity}(y_t) &= \mathbb{1}[y_{t,currentliab} > y_{t,currentassets}] \\
&\exists t \in \mathbb{E}^{val=1year} \\
\text{bankruptcy}(y_t) &= \text{Insolvency}(y_t) \wedge \text{Illiqudity}(y_t) \\
y_1 = \text{bankruptcy}(y_t) &= 1, y_0 = \text{bankruptcy}(y_t) = 0 \\
\mathcal{L}(\hat{y}_i, y_i) &= -w_i[y_i \log(\hat{y}_i) + (1 - y_i)log(1 - \hat{y}_i)] \\
&i \in \{0, 1\}
\end{aligned}$$

The w_i is the weights we to make the much less frequent bankruptcies contribute as much as the non-default labeling tasks, of which the algebra we do not provide here but is of course found in our Our GitHub repository.

The loss function is an approximation and does not truly measure actual filed bankruptcies for all companies through time. Firstly, we do not have this data as part of our dataset. And secondly, one main aim of this paper is to show that our methods are truly general and that easily can be adapted to new problems.

We use four different accuracy measures for measuring accuracy in the categorization task of default non-default for comparability with another paper on the topic (Moen, 2020). We do not go into great detail but provide their definitions and a short description of how to interpret the measures. The four accuracy measures are the following:

Accuracy:

Accuracy is the % of correctly predicted labels for a whole set of predictions and target labels. Higher accuracy is generally better.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{23}$$

The F-1 score: The F-1 score is the harmonic mean of Precision and Recall. By combining both these statistical measures the F-1 score can be described as offering a more nuanced metric compared to that of accuracy. A higher F-1 score is generally better. To define F-1 score we first define precision and recall:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \\ F1 - score &= \frac{2 * Precision * Recall}{Precision + Recall} \end{aligned}$$

The Brier Score: The Brier score is a measure of how far from the ground truth a set of probabilities for mutually exclusive events is. A lower Brier Score is better.

$$BrierScore = \frac{1}{N}\sum_{i=1}^{N}(prob_f(outcome_i) - outcome_i)^2$$

AUC (Area Under the ROC Curve):

The AUC is best described by a picture, it is the area under the ROC (Receiver Operating Characteristic) curve. It tracks the movement of the true positive rate, and false-positive rate for a set of predictions when increasing the limit for what is deemed a True prediction. An AUC score of 1 is perfect and 0.5 is what a set of 50% predictions will give in the limit. A higher AUC score is generally better.

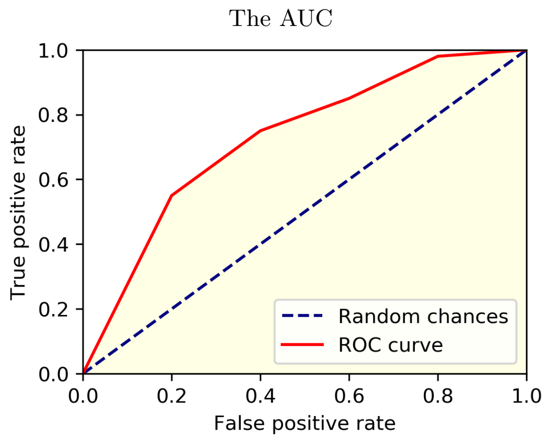


Figure 24: The AUC shows the true positive rate, and false positive rate for a set of predictions when increasing the limit for what is deemed a True prediction

Lastly, we create our own profitability measure as a function of a given ROC. Each step in the ROC corresponds to a confusion matrix. We again argue that each confusion matrix can be transformed into profits given a bank's corporate loan objectives of lending out money to non-defaulting companies. Each confusion matrix is transformed in the following way, where T is a threshold function given a probability and τ for returning true, and where we somewhat arbitrarily set the payback time to 10 years:

$$\begin{aligned} &profit(r, \tau, \text{model}) = \\ &\sum_{i}^{Comp} \Bigg(T(p_i(\text{model}), \tau) B_{iF} ((1 + r)^{10} - 1) \\ &- T(p_i(\text{model}), \tau) B_T 0.8, \Bigg) \\ &B_{iT} + B_{iF} = 1 \forall i \\ &B_{iT}, B_{iF} \in \{0, 1\} \end{aligned}$$

This is a simplification of the real world in many ways, however, the transformation into profit space is important to do for real-world applicability and analysis that matters.

6.3. Training Procedure

Our test set starts on January 1st, 2019. From there, the test set contains all months up to and including March 2022. Consequently, only data before January 1st, 2019 is allowed to be used in

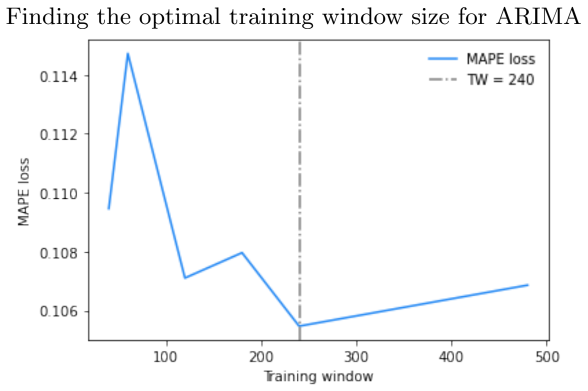


Figure 25: To empirically find the optimal window of previous data points to train for, we ran several tests on a validation set. We chose a random subset of 200 tickers. We evaluated the resulting performance of the fitted ARIMA models for those companies on the validation set for many training window sizes between 1 month and 2 years. 1 year, i.e., or 240 data points backward in time, came out as the strongest candidate.

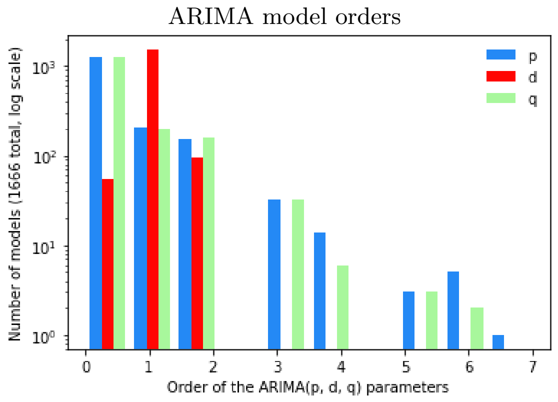


Figure 26: Overview over all ARIMA models and their orders for autoregressive, p, and moving average q lag terms, and number of times it differences, d.

training the models. To make the comparison fair, we freeze all model parameters after training completes, i.e., the models cannot fit more when predicting later months in the test set; they only get access to the latest values in the series on which to base their predictions.

6.3.1. Parametric, Univariate Models

For the ARIMA model, we started by using a subset of the data to investigate the effects of the Training Window (TW) on the performance of ARIMA. In particular, we explore how the accuracy of forecasts varies when the amount of data any single ARIMA model can fit on varies. For example, Figure 25 shows how MAPE changes with TW. We find that the optimal TW is 240 data points, i.e., 12 months, which seems reasonable given that the task is to forecast one month into the future.

To train the models, we pulled the market cap data for all companies and split it at 2019-01-01, our hard limit for what constitutes training data and what constitutes test data. Since we use 60 data points for training and 20 for evaluating, we included only the companies that could satisfy this data requirement. This filtering includes most companies.

Then, we fitted separate ARIMA models for each time series. In particular, we fitted AutoARIMA models from the pmdarima-package Smith et al. (2017–). This model automatically finds the best order fitting the ARIMA model according to Akaike Information Criterion (Akaike, 1973). We used all standard settings except disabling the seasonality orders and allowing AR and MA lag terms to take any values up to 10 (instead of 5). We believe including seasonality complicates our analyses unnecessarily and is unlikely to have any expressive power given the short forecast horizon. We refer to Smith et al. (2017–) for further details on implementation.

In Figure 26, we plot a histogram of the model orders of all 1 666 ARIMA(p,d, q) models fitted (on a log scale). We can observe that the vast majority of models select parameters to be either 0, 1, or 2. This is likely due to the use of information criteria in model selection, which penalizes larger models if the model fit is not more than proportionately improved. A very small subset of models have 5, 6, or 7 autoregressive lagged terms or moving average terms. From this, we are led to believe that the ARIMA models are having a hard time coming up with better models than some relatively naïve models.

All the 69k GARCH models were fitted in a rolling window fashion, month for month on the test set.

Lastly, we used the resulting models to produce forecasts for all the time series, which we subsequently compared to the actual values for January 2019 up to and including March 2022.

Although each ARIMA and GARCH model is relatively simple and fast to fit, since one has to fit separate models for each company, the total running time can be large. When using an Intel Core i7-10700 CPU @ 2.90GHz CPU, training took about 0.5 hours, and prediction for the full test set took about 1.5 hours for the ARIMA model. 6.1 hours was the total running time for the fitting and prediction of the GARCH on the same setup.

The fitting and prediction of the logistic regression models were more or less instant, running on the same CPU, and is not discussed further here.

6.3.2. Parametric, Multivariate Models

The procedure we utilized to generate results for our parametric multivariate models is very similar to the univariate case. The critical difference is the added complication of handling exogenous variables.

Initially, we include the exogenous variables by joining the 'market' cap data for each stock with the 'fundamental' data for that stock and the 'macro' data (which is the same for all companies). Since the periods differ between the sets (e.g., the 'market cap' data is daily and 'fundamentals' are quarterly), the 'market cap' dates serve as the base, and we insert the other sets into that base. This procedure creates gaps, especially for the quarterly 'fundamental' data. We fill these gaps forward, and only forward, such that we are utterly confident that no information leaks from the future into the past.

We restrict the models to train on 60 data points here (as in the univariate case), mainly to facilitate comparisons between univariate and multivariate ARIMA models. Furthermore, some experimentation bolsters that this restriction is best for the multivariate ARIMA.

We then pass this data to the same AutoARIMA model as above, as this model straightforwardly extends to the case with exogenous variables. However, there is a caveat because of the internal matrix algebra, which forces us to drop columns of data that are constant through time before we pass it to the procedure (invertible matrices must have linearly independent columns, and a constant term is indistinguishable from a constant variable). Constant columns occur most commonly in the 'fundamental' data set, as companies sometimes have delays in their reporting of quarterly reports or report several at a time when first listed. Therefore, not all ARIMAX models will train on the same subset of columns.

When predicting, there are also necessary considerations regarding the exogenous variables. The ARIMAX model requires values for the exogenous variables for all future time steps it is predicting to make predictions. Since these values are strictly unknowable at prediction time, the best we can do is take the latest observed value for all and propagate them forward (copy) for the coming month. These variables will then function to establish the overall level for the ARIMA model to predict within, similar to the simple example shown in Figure 23.

Lastly, we again compare the resulting forecasts with the ground truth and produce a Mean Absolute Percentage Error (MAPE) loss for the forecasts.

For ARIMAX models training took about 15 minutes, and prediction for the full test set took about 4.5 hours, on the same setup as in the last section.

6.3.3. The Era-Epoch Training Algorithm

This section introduces our Novel Era-Epoch Training Algorithm, which we use to fit our models.

The key idea is that we divide data into a set of *eras* of fixed length, which contain non-fixed epochs. For a time t_0 to $\mathcal{T}$ the algorithm divides all data into eras, $\mathbb{E}$, which is the set of all data y and X, the dependent and independent variables in this time frame. A model is then fit on this era's dataset, of which one entire run trough is called an *epoch*. The epochs are run until a stop-criteria based on the validation loss $loss^{Va}$ is reached, inducing an onwards movement to the next era since there is no more to gain from training on this Eras dataset. Importantly, by this procedure, the model learns what it can from each era but never predicts things backward in time, which we believe is unhelpful since it is already known. See Algorithm 1 for the pseudo-code describing this procedure in detail.

This procedure contrasts with the parametric model since we train models for windows bordering the test set. Therefore, these cannot incorporate knowledge about patterns from the past or other companies in the same segment.

The algorithm begins with generating all eras between the start and end dates defined (one era for each month) on Line 1 and initializing the model parameters to a vector of normally distributed random variables in Line 2.

In Line 10, we update the parameters of the model in the simplest way possible. We utilize more complex methods for parameter updates, most notably the Adam optimizer (Kingma and Ba, 2014). An in-depth discussion of the workings of these different optimizers is not appropriate here, and we would instead direct the reader to the code in the

Algorithm 1 Era-Epoch Training

```
1  $\mathbb{E} \leftarrow generate\_eras(t_0, \mathcal{T})$
2  $\boldsymbol{\theta} \leftarrow$ random vector $\sim \mathcal{N}(0,1)$

3  for $t = t_0$ to $\mathcal{T}$ by $h$ do
4    $y_{t+h}, y_t, X_t \leftarrow get\_data(\mathbb{E}, t)$
5    $y_{t+2h}, y_{t+h}, X_{t+h} \leftarrow get\_data(\mathbb{E}, t+h)$
6    for $epoch = 1$ to $max\_epoch$ do
7      $\hat{y}_{t+h} \leftarrow f(y_t, X_t; \boldsymbol{\theta})$
8      $loss^{Tr} \leftarrow \mathcal{L}(y_{t+h}, \hat{y}_{t+h})$
9      $\nabla_\theta \leftarrow \frac{\partial}{\partial \theta}(loss^{Tr})$
10     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \nabla_\theta$
11     $loss^{Va} \leftarrow \mathcal{L}(y_{t+2h}, f(y_{t+h}, X_{t+h}; \boldsymbol{\theta}))$

12     if $stop\_criteria(loss^{Va})$ then
13       End era by breaking inner for-loop
14     end if
15   end for
16 end for
```

GitHub repository and the respective papers describing the methods and the definitive textbook on deep learning, Goodfellow et al. (2016).

In this master thesis particular implementation of the Era-Epoch Training Algorithm, t_0 = December 31st, 2000. We take the first 12 months of input data to predict the next month of data, with the month after that serving as the validation data. This collection of months of training data is the first era, and it will contain several Epochs limited by the stopping criteria as a function of the validation loss. We let the era advance a month at a time thus having $h = 20$ (since there are about 20 trading days for each month). The raw data that our non-parametric models train on is the same as the parametric counterparts and is described meticulously in section 4.5.

6.3.4. Traditional training loop

We also use a traditional training loop for our models, where we sample random eras from the past and validate on future eras. Figure 27 shows that where one chooses to start to train a model can heavily influence what market conditions it learns from. This led us to also opt for this traditional approach, making the gradient decent of neural networks less dependent on starting conditions.

6.3.5. Hyperparameter search

One significant drawback of most machine learning methods is that they have several parameters governing their fitting algorithm, known as *hyperparameters*, and they are often relatively sensitive to small changes in the hyperparameters. Therefore, to find the optimal combination of parameters, we conduct a large-scale hyperparameter search. The hyper-parameters are a result of a thorough over the hyperparameter space, based on Bayesian search and Hyperband-pruning of runs, as described in Li et al. (2017). The online service wandb.ai provided the logic for orchestrating the search. The actual search parameters can be found as logs in of training in our Github repository.

In short, Bayesian search means that the algorithm starts out picking a combination of hyperparameters completely at random. Over time, the algorithm will update its prior probabilities of what hyperparameters are the best in this situation. Thus, it will over time hone in on the areas of the search space that are the most promising. This improves upon grid- and random search, because it is often not feasible to test out all combinations of parameters, and the algorithm, can quickly discard obviously bad neighborhoods of combinations.

Hyperbands are a strategy for deciding which experiments (a full training cycle for a given set of hyperparameters) to let run until completion and which to kill early in the interest of directing the resources towards the most effective causes. For the details on this algorithm, we direct the reader to the work by Li et al. (2017).

For both the Bayesian search and the Hyperband algorithms to work, it requires a metric with which to compare training runs. Usually, one would use a validation set that is a random sample of the data available for training. The nature of the time series complicates this situation. That which is representative behavior for a time series is expected to morph with time, as the market conditions change (economic booms and busts, interest rates, inflation, commodity prices, and a plethora of other observable and non-observable reasons). Therefore, the metric we use to compare candidate models must accurately represent different market conditions. We settled on a setup that mixes the losses for the next 12 months with the loss for the last 12 months of all the available training data. We normalize all losses by the naive loss to make each loss value more comparable across eras.

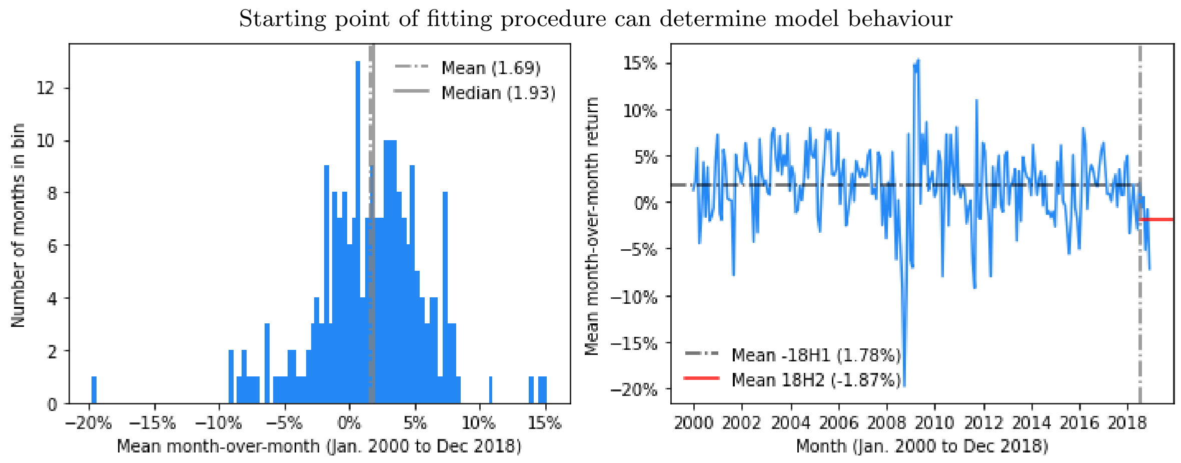


Figure 27: Where one chooses to fit a model determines the market and trends it is subject to. For instance, our net is training in a market with a positive average return before the last eras of the training loop, where the average return is bearish.

6.4. Feature importance

6.4.1. Permutation importance

We used input data permutation as a way of discovering feature importance. For panel data, there are standardized libraries for this, but we constructed our own cube feature importance for this project. In our data cube, we select a subset of features and times and scramble these for each company. One then sees how a prediction deviates from the baseline prediction when the input data is scrambled. One repeats the scramble multiple times to get a distribution of loss differences of which we can investigate the mean and standard deviation to make inferences about what input variables a model is using. If a selection of features and times scrambled leads to a large loss over many scrambles one can deduce that the model uses this information to create accurate predictions.

6.4.2. Gaussian importance

For the macroeconomic variables we are not able to permute them like in subsubsection 6.4.1, this is due to the fact that they are independent of companies and thus, are, dependent on model implementation fed into the model as features of the companies. Thus we create a gaussian importance test where we add gaussian noise to all examples of a macroeconomic series to determine their impact on model output, as a deviation from baseline in the same way as is done for the traditional permutation approach.

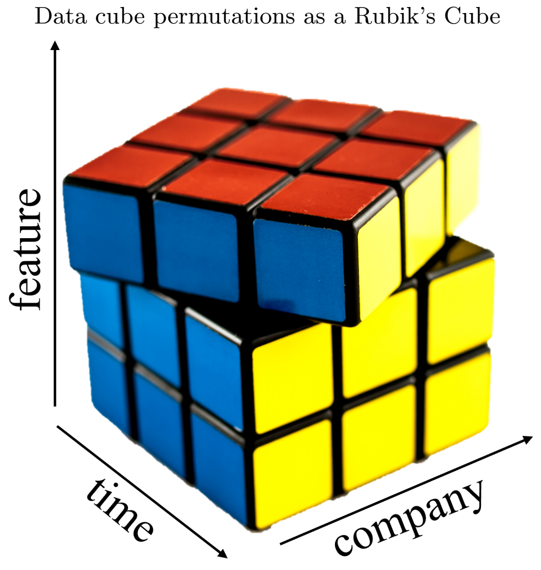


Figure 28: The permutations of our input data can somewhat analogously be viewed as a Rubik's Cube rotation. The rotation moves one feature time tuple to another company's input and logs how this impacts the loss.

7. Results

This section presents the results from applying our general forecasting approach to 3 different but related problems:

1. Bankruptcy risk forecasting
2. Volatility forecasting
3. Market capitalization forecasting

First, we present our results on bankruptcy prediction and analyses in subsection 6.4.2. Second, we forecast stock price volatility in subsection 6.4.2. Lastly, we apply the framework to market capitalization (stock price) forecasting in subsection 7.2.2. We compare the results in each part to relevant benchmarks. The framework shows good performance on the bankruptcy forecasting task, adequate results when forecasting variance, and does not beat the market, in the market capitalization forecasting task.

Table 2 shows the results from our experiments with predicting bankruptcy risk for all companies in our dataset. We also include results from a related work by Moen (2020) on a dataset of Norwegian companies for reference.

The table shows that the univariate TCN model out-competes all other models in terms of the four accuracy measures we use for bankruptcy prediction (accuracy, Brier, F1, and AUC, see subsubsection 6.2.2 for definitions). We also show that our exogenous models have a higher Area under the ROC Curve (AUC). Furthermore, we find that when put in the context of a bank considering loan applications, the AUC metric transforms into a more profitable prediction strategy.

We believe the test results for the fundamental category are the most convincing out of the 3 tested in this work. We test the fundamental prediction capabilities of the general forecasting framework in terms of a classification problem for bankruptcy. We show that our model performs similarly to statistical models on established accuracy measures. We also compare the results to those of Moen (2020). Please note that Moens results are on a different dataset, and thus the comparisons must be taken with a grain of salt. Nevertheless, we believe they are relevant, and our respective benchmarks show similar results.

We use the standard classification threshold of $\tau > 0.5$ to predict whether a company or its financial statements are predicted to go bankrupt within the following year. The predictions are scored according to the measures in subsubsection 6.2.2. We emphasize that this is just one fundamental prediction we chose to do, and augmenting the system to predict revenue growth, EBIT margin, refinancing, etc., are all achievable through picking appropriate output from the datacube and designing a loss function measuring the desired outcome.

In Figure 29 we show the confusion matrix for the category-specific accuracy for our TCN-X and LSTM-X models, the univariate TCN, and a Logistic Regression model (denoted LR). Fitted on the train set, it is evident that the neural networks have learned to generalize. Even though there is a great unbalance in the dataset, all three non-parametric models predict bankruptcies with a 70-77% internal accuracy. Most importantly the LR model dominates all other models' confusion matrices. We note, however, that this domination does *not* imply a higher AUC and accuracy. In fact, we find the opposite to be true, as discussed below.

Comparing TCN to TCN-X and LSTM-X, it seems more inclined to predict ´Not bankrupt', which results in more true negatives (negative being ´Not bankrupt', but also the most false negatives, which in the context of the labeling task is the most costly error (since a defaulted loan can incur bigger losses than a fully paid loan incur gains).

In Figure 30 we show the absolute number of predictions of the model and how they categorize in terms of actual and predicted. This is simply another view of the same information presented above. Still, it is evident that all models learn to predict 'not bankrupt' quite a lot, which is sensible, and is what a Naïve baseline would also do since the class of companies that are not going bankrupt in any given year is much larger than those that do. However, the fact that all models but TCN have a true-positive rate that is 3 times greater than their false-negative rate is a clear indication that the models have learned.

In Figure 31 we present our results (blue bars) compared to Moen (2020). We find that more advanced models can outperform simpler ones. Firstly, across accuracy, Brier score, F1-score, and AUC, our models are on par with Moen (2020). The 4 models we include from Moen (2020) (red bars) are the RNN (Recurrent Neural Network), LSTM, CB (CatBoost, gradient boosting method)[3], and Logistic Regression. The correlation between the scores across metrics from the two papers validates the overall validity of our approach. Most notably, our F1 score is quite a lot better at $\sim$0.40 for both our models, and AUC scores by Moen (2020) are higher than all of ours. Our best model in terms of all but AUC is the TCN. In fact, our TCN outcompetes all models except for CB and our LSTM-X in terms of AUC. Nevertheless, the differences could be caused by some intricacies in our respective bankruptcy definitions.

In Figure 32 we provide prediction frequency histograms for all our models. The leftmost bar shows the rate at which the model predicts that a company will almost certainly not go bankrupt within the next year for all companies in the test set. It is evident that the non-parametric models have a clear

[3]For info on CatBoost, see catboost.ai

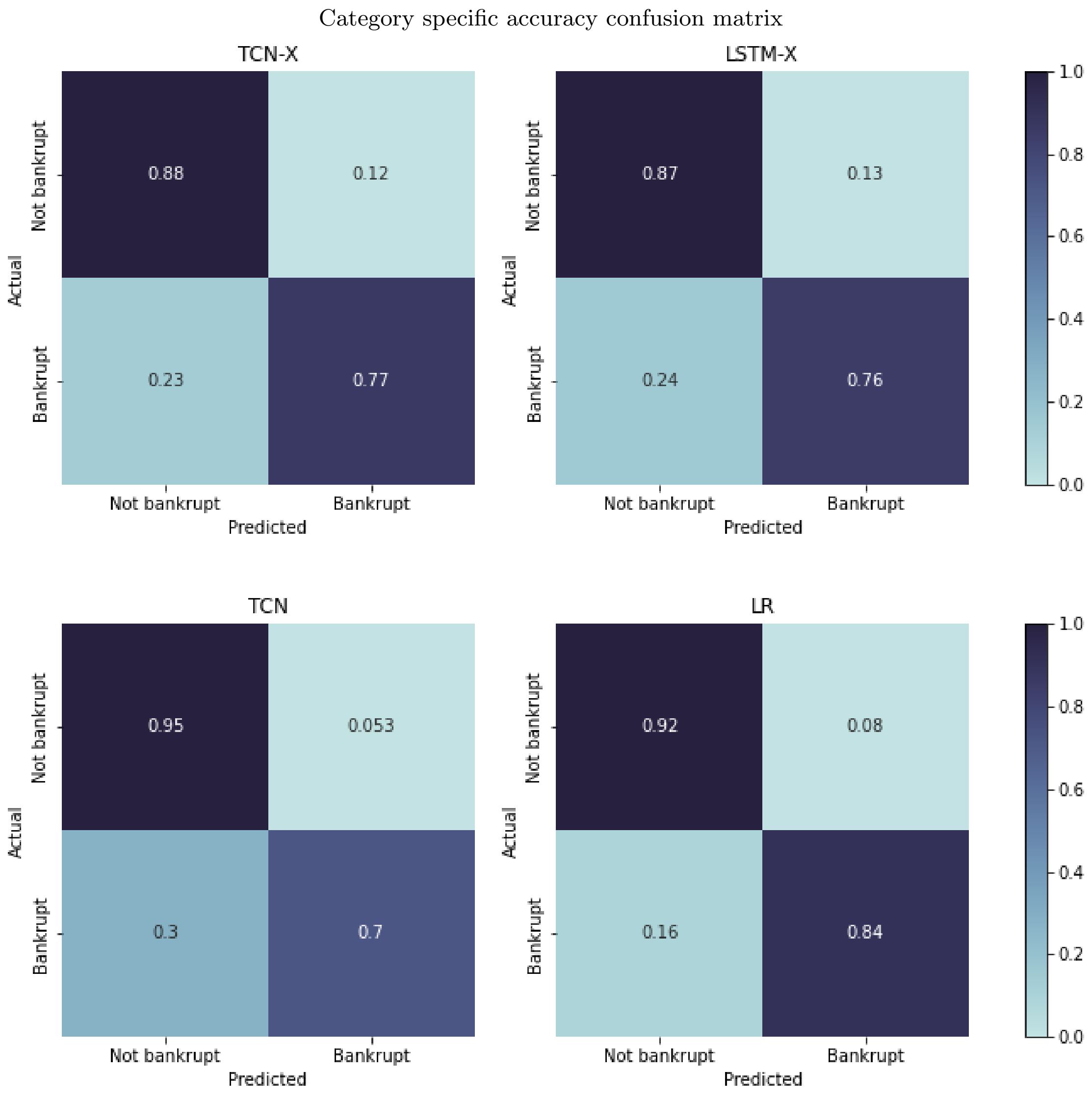


Figure 29: The confusion matrix for the % correct labeling within each category. The LR model is strictly dominating all other models in this confusion matrix. Note that all other models have a 70-77% correct prediction for the Bankrupt - Predicted Bankrupt category.

Bankruptcy prediction results

Model	Exogenous	Parametricity	Locality	Accuracy	Brier	F1-Score	AUC
TCN-X	Yes	Non-parametric	Global	0.8759	0.1241	0.4118	0.8224
LSTM-X	Yes	Non-parametric	Global	0.8705	0.1295	0.3965	0.8479
TCN	Yes	Non-parametric	Global	**0.9323**	**0.0677**	**0.5546**	0.8471
LR	Yes	Parametric	Global	0.9157	0.0843	0.5471	0.7262
RNN†	Yes	Non-parametric	Global	0.8238	0.1261	0.1023	0.8795
LSTM†	Yes	Non-parametric	Global	0.8230	0.1295	0.1008	0.8836
CB†	Yes	Non-parametric	Global	0.8144	0.1258	0.1019	**0.8895**
LR†	Yes	Parametric	Global	0.8146	0.1332	0.0961	0.8609

Table 2: Result summary from our models from the bankruptcy risk prediction task, as well as the models from Moen (2020), which are marked with †.

Absolute number confusion matrix

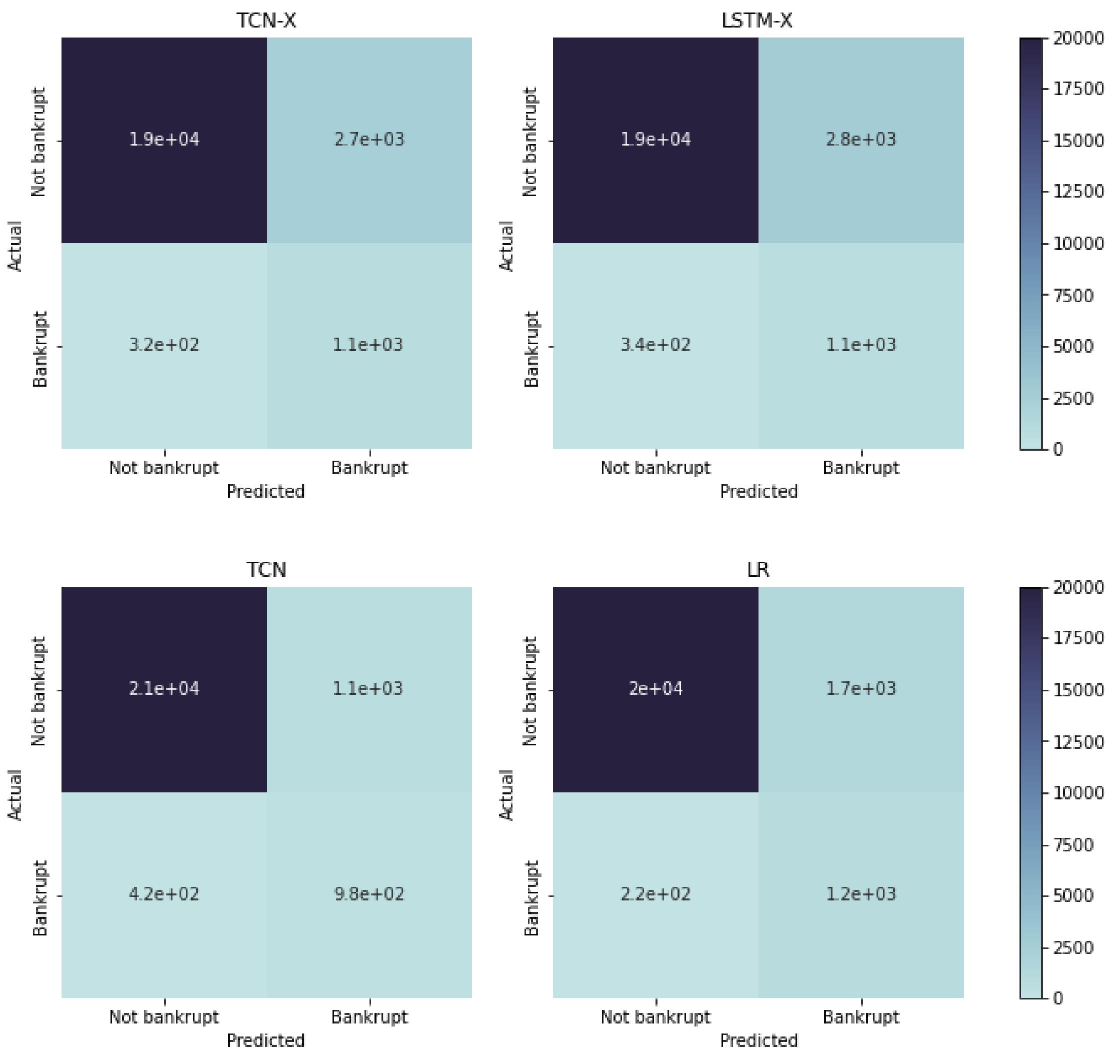


Figure 30: The confusion matrix for the absolute number of prediction in each category for four models. LR is strictly dominating all other models. The Bankrupt - Predicted Bankrupt is still ∼3 times larger than the Not Bankrupt - Bankrupt category for both non-parametric multivariate methods.

Accuracy measures for models compared with other paper

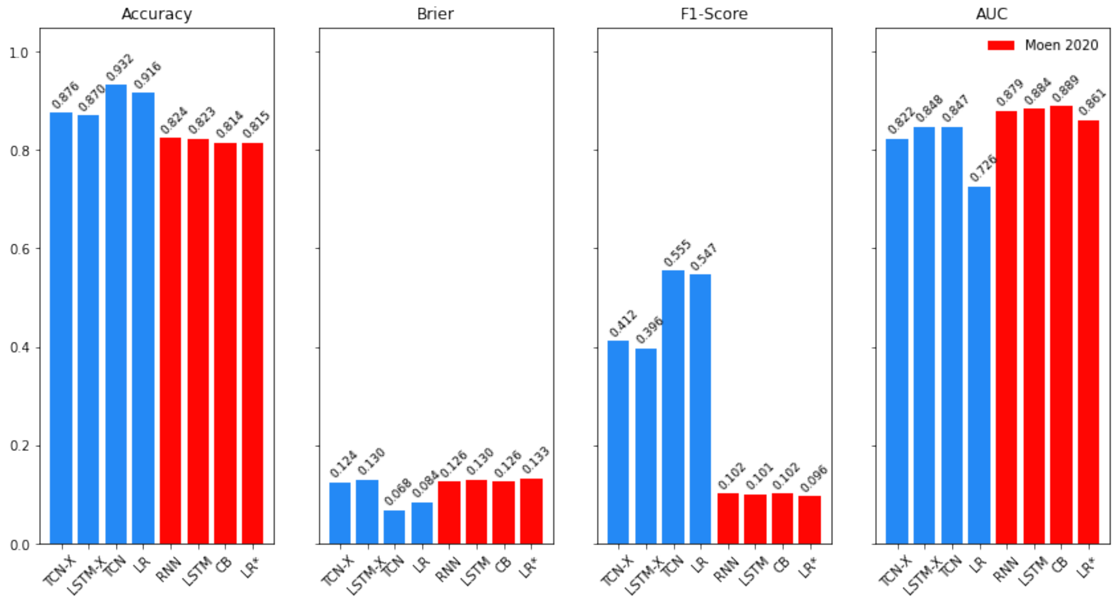


Figure 31: The bankruptcy version of our models seems on par with the models of Moen (2020) when the numbers are taken as is. Note that the dataset and notion of bankrupt is different in both papers.

tendency to predict ∼0% chance of bankruptcy with high conviction. This is interesting, as humans are familiar with a notion of confidence in prediction, and would indeed weigh this as very important when e.g. processing loan applications. In this light, the prediction weights of the LR model, are much more spread out and the model prediction is thus heavily dependent upon the threshold one sets. Although they are small the areas of each bar along the x-axis of the non-parametric models (confidence of bankruptcy) are clearly non-negligible, and the highest for our two TCN models. Thinking about an actual probability distribution we believe these plots are rational. In subsubsection 5.3.1 we show that around 2.5% of the fundamentals in the test window are categorized as bankrupt, this implies that a good model should have most of its area below the 50% mark on the x-axis.

Another way to see the confidence in the neural networks is by displaying AUC scores as seen in Figure 33. The following is what we deem one of the most compelling findings of all our analysis, and thus we want to emphasize it: *Both exogenous models are highly confident in most scenarios, this is not the case for any other model we tested.* The significant spike seen for both the blue and red colors shows that the exogenous models have a large set of predictions seemingly infinitesimally close to 0. Stated simply, it means that given more information about the market condition and more of a company's fundamental data, the neural network implementations get extremely confident in predicting non-defaults. This strong conviction is not the case for the TCN. Given only the previous assets, current assets, liabilities, and current liabilities (in this case, the combination of them is the univariate case), the TCN has a more even distribution of convictions, which means that the model hesitates more in predicting 0% chance of default. We believe Figure 33 and the above reflections are the most apparent ex-ante result for multivariate superiority presented in this paper.

The above conviction transforms the exogenous models to the most profitable when we pass the AUC through the profit function described in subsubsection 6.2.2. Intuitively, a firm conviction in financial markets can be transformed into more profits. Figure 34 displays the AUC passed through our profit function. The absolute maximum is at the ca. 58% threshold for the LR model. Interestingly, as we suspected from the leftmost part of the AUC curve, the exogenous conviction is rightfully transformed into more profits, measured in terms of the average value of a 10-year loan with no defaults. A

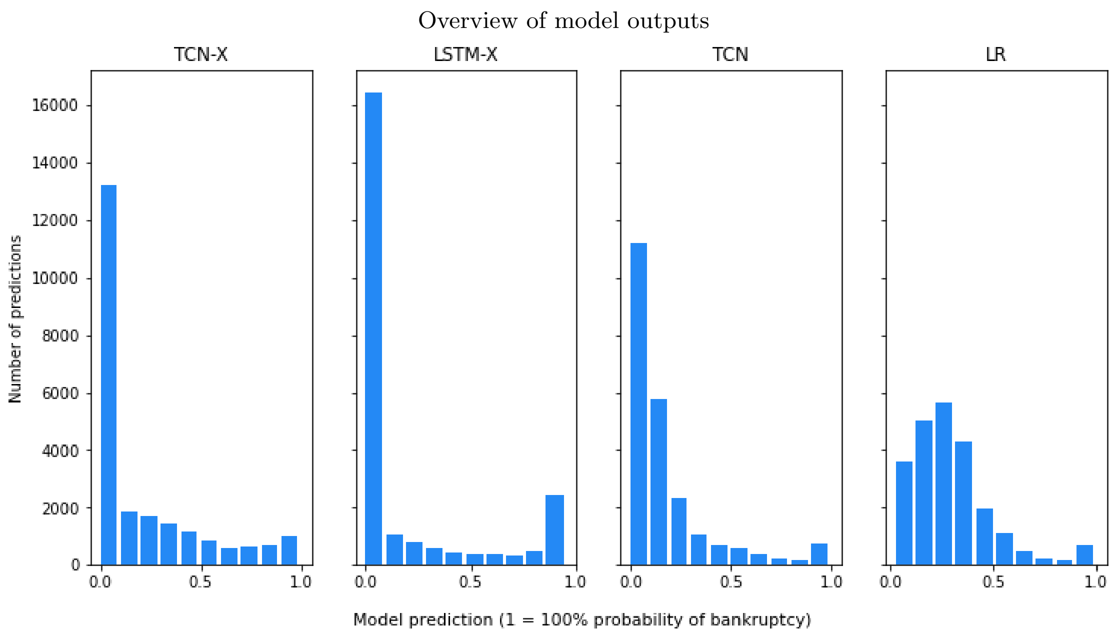


Figure 32: Both exogenous models has a clear tendency to predict 0.0 chance of bankruptcy. Since only ∼50 of the more than 1600 companies of the test set are classified as bankrupt this is to be expected by a good model.

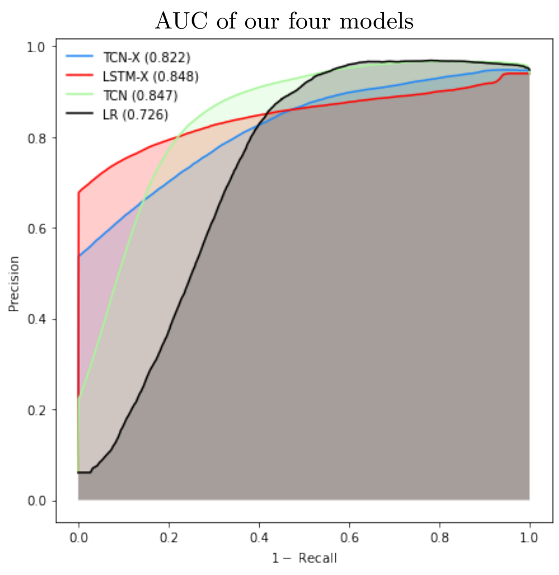


Figure 33: The AUC curve for the four models we have implemented. Our LSTM-X has the highest AUC. Note the big difference in high conviction for the X models given a true limit. The exogenous models are essentially 100% certain of non-bankruptcy for a great deal of companies.

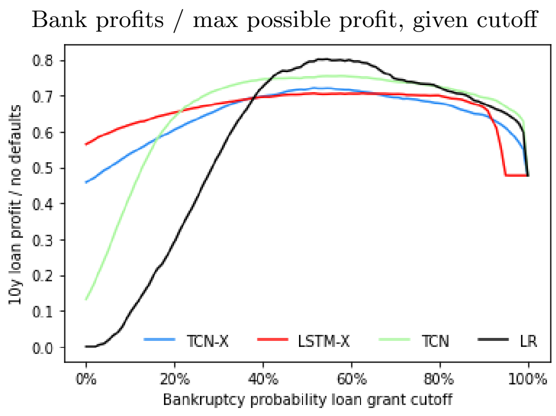


Figure 34: Show the profit transformation of the models AUC. When looking for strong conviction in default predictions LSTM-X is the most profitable. The max profit of all models is found at around 58% certainty for the LR model.

more typical investment metric is the ROIC which can be seen in Figure 36. Fittingly all models are equal when doing the Naïve forecast at 100% conviction of handing out a loan.

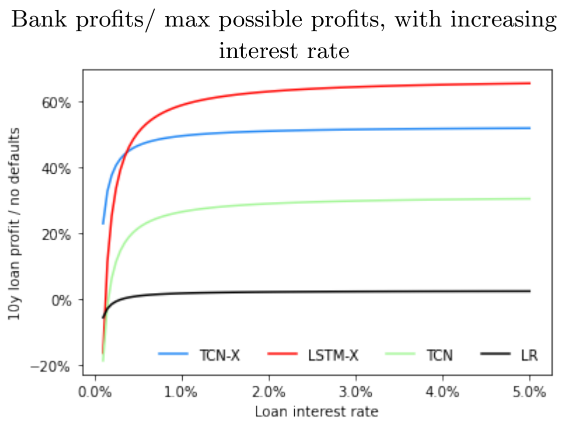


Figure 35: Setting $\tau = 0.05$, demanding high conviction for the bank to grant a loan the LSTM-X model is the most profitable for all values of loan interest rate. In reality a 5% chance of bankruptcy would demand a high rate of return (of which LSTM-X) is still the most profitable.

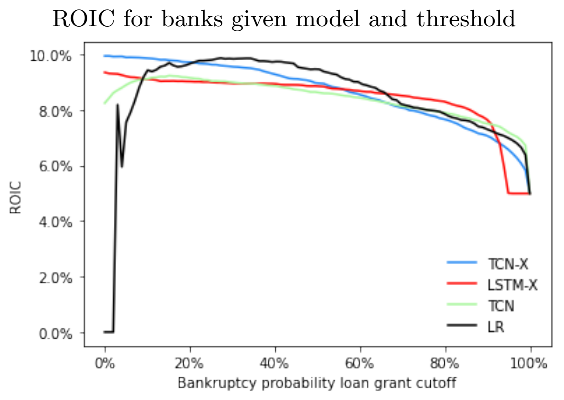


Figure 36: Given a 1% inflation adjusted interest margin a year, ROIC of the exogenous multivariate models is the highest in the strongest Non-bankrupt conviction cases. The absolute maximum ROIC is 9.9% at approximately 98% conviction of non-default for TCN-X.

This section presents results on the test set from January 2019 to April 2022 for parametric and non-parametric models trained on the training datacube (January 2000 to December 2018). We fit univariate TCNs, multivariate TCN-Xs, and multivariate LSTM-Xs on different variables, and 69k GARCH(1,0,1) models and the naïve strategy serve as the baseline.

The results of the experiments are in Table 3. Our TCN-X is the best non-naïve model, although naïve strictly dominates every other model we test.

7.2.1. Analysis of parametric models

We fit (68,092) simple GARCH(1,0,1) models, with an input window of 1 business year, predicting 1 month ahead of volatility measured with standard deviation. Notably, this is opposed to our methodology of the ARIMA fitting, where we fit data in a training window, then use the estimated parameters to forecast the test set. In addition, we refit the GARCH model because there is no online Python implementation of Auto-GARCH or a GARCH that allows for weight updates, which we have found. Thus, we fit many models for each era in the test set over time.

Starting with one forecast, Figure 37 shows the GARCH40 model prediction. The name refers to the 40 refitted GARCH(1,0,1) models used to produce forecasts for the entire test set (which consists of 40 months). The GARCH40 model's prediction's shape is similar to the actual AKER BP monthly volatility but is a lagged. The predictions lagging the observations is a natural consequence of how GARCH models work (reacting to past changes and assuming volatility clustering). Although we fit the model on a full year and the predictions are seemingly accurate, the loss seen in Table 3 increases since the model only captured the spike in March 2020 later when its effect is counterproductive (a sizeable positive deviation between actual and GARCH40 in the later month).

When looking at many predictions in Figure 38, it is evident that the GARCH40 models have some considerable spikes in predicted volatility. These spikes could be detrimental to the models' overall loss. For example, since the GARCH models seem to resemble the naive approach, if the standard deviation rapidly diminishes from one extreme month (which indeed seems to be the case in spring 2020), overall, it will lead to a higher loss as the extreme

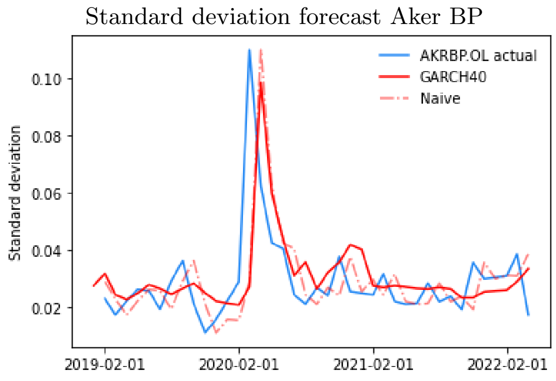


Figure 37: GARCH40: The GARCH(1,0,1) model rolling window fitted on Aker BP. The GARCH model is in similar shape to the actual standard deviation although is lagged.

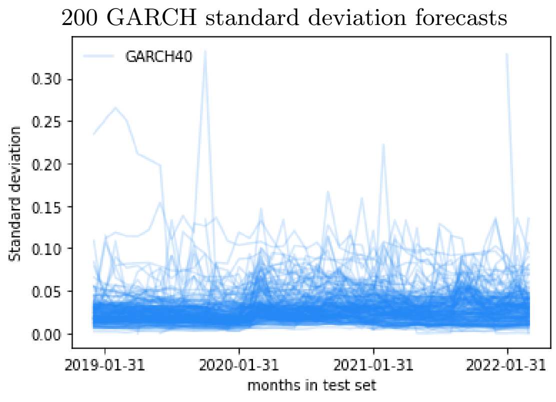


Figure 38: GARCH: The plot shows 1 month ahead day-meaned forecast of 200 randomly chosen time series. Each line is the prediction of 39 months in the test set, independently trained on 1 year of prior data. Some of the GARCH models are highly spiky and the time independent median volatility prediction is around $\sigma = 2.5\%$

event makes the model over-predict the standard deviation of the following month. Such a finding could be interesting to keep in mind when looking at the non-parametric models' results in the following section.

7.2.2. Analysis of non-parametric models

We create a TCN model termed 'TCN-MV' as we fit it only on the company's Market cap and the market VIX. The NN learns to use both features as seen in Figure 39. The market cap seems to be the feature of the two that is most important, as we very consistently get a 0.9 higher loss than the baseline (standard prediction without noise) when

Model	TW	h	Exogenous	Parametricity	Locality	Input scale	Target scale	Era mode	MAE	Gap to best	% from best	Best 99%
Naïve	20	20	No	Parametric	Local	None	Return	N/A	**1.44%**			**1.34%**
GARCH	240	20	No	Parametric	Local	None	Return	N/A	1.59%	0.15%	0.52%	1.46%
TCN	240	20	No	Non-parametric	Global	Min-max	Return	Random	1.53%	0.09%	0.30%	1.42%
TCN-MV (X)	240	20	Yes	Non-parametric	Global	Min-max	Return	Random	1.49%	**0.05%**	**0.17%**	1.38%
LSTM-MV (X)	240	20	Yes	Non-parametric	Global	Min-max	Return	Random	1.54%	0.10%	0.33%	1.43%

Table 3: Table overview of standard deviation prediction. Naïve is better than all other models. h is the forecasting horizon, and TW is the training window, i.e., the number of time steps into the past the model can see each time it forecasts h time steps into the future.

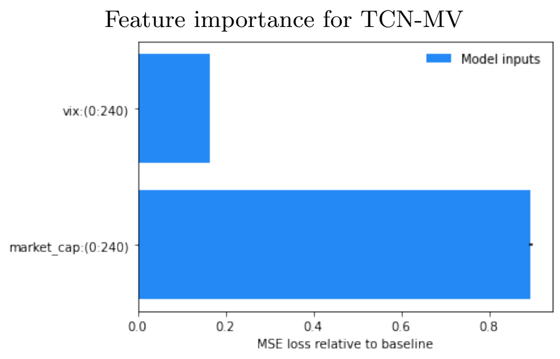


Figure 39: TCN-MV: The results from performing Gaussian importance with 100 iterations with the model and our data cube. The noise the Gaussian importance procedure uses is normalized with the median of each respective feature.

applying Gaussian importance calculation as described in subsubsection 6.4.2.

Continuing in understanding this TCN-X model, we show one prediction in Figure 40 on Aker BP, 1 month ahead forecasts on our test set. The prediction is relatively constant over time, with some spikiness around March 2020. In Figure 41, the spike is a prediction that affects all forecasts, which complies with preconceived notions as the market's overall volatility spiked at that time. It seems like the network has a pretty homoscedastic view of the variance, as we can see that the prediction is more or less constant with time.

This homoscedasticity can probably be explained through the NN being a global model, training and averaging its observations across all companies and time. This way of training stands in stark contrast to the local models (GARCH). The significant spike in the March 2020 prediction contradicts the homoscedasticity, which shows that the model has some notion of a macro event where it increases all its predictions, despite having never seen March 2020 data before.

The GARCH40 models looks better than the TCN-MV model but it is a fact that it has a higher

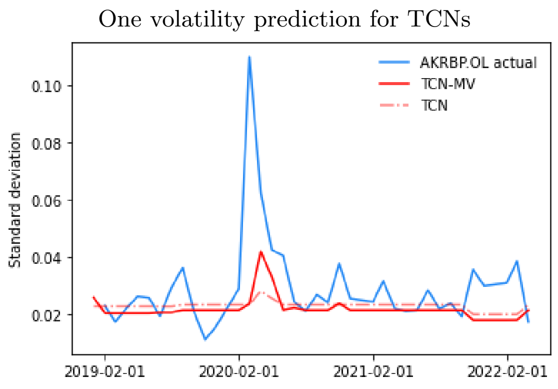


Figure 40: TCN: 1 month ahead prediction for AKER BP volatility over the test set. Note that the TCN predicts a variance level that is quite constant, although there are some spikes.

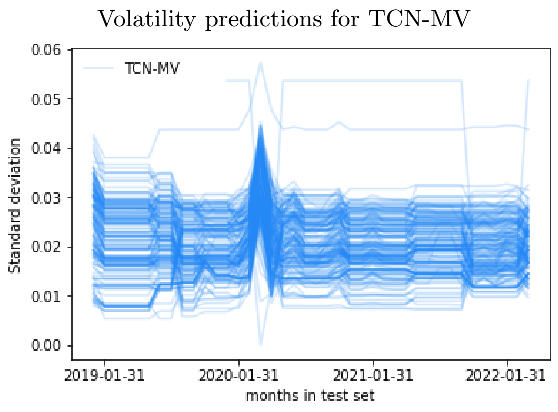


Figure 41: VIX&MCAP: 1 month ahead prediction for 200 random tickers. Note that the spike from Figure 40 is present for all tickers, though with different magnitudes.

Distribution of mean loss for TCN-MV and GARCH40

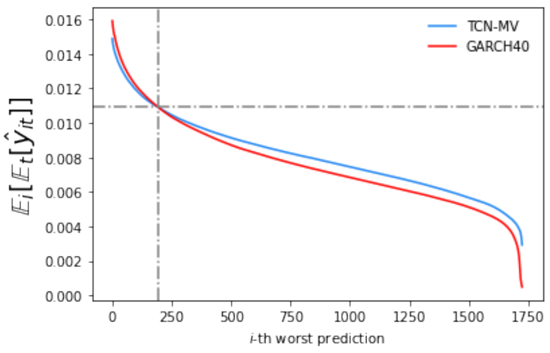


Figure 42: Removing the ∼200th worst companies from TCN-MV and GARCH40, makes GARCH superior in this specific case. In other words, the mean and median loss of TCN-MV is lower and higher respectively than that of GARCH40.

mean loss on the test set. This is because we are comparing mean, in fact, the GARCH40 model has a higher median score. In Figure 42, we show the losses of GARCH40 and TCN-MV if we were to remove the i-th worst company measured in MAE loss. We would need to remove 200 companies for GARCH40 to outperform TCN-MV. If one were to remove a company randomly from our distribution, with replacement, TCN-MV would outperform GARCH40 given enough removals – this is the mean. As presented in section 5.3.1, we train the NN to minimize *mean* loss. We postulate that we would have seen a higher variance and dependence on market capitalization (as seen in Figure 39) if we were to change the batch normalization procedure to that of a median instead of a mean over tickers.

In Figure 43 we show the mean across companies of the standard deviation in each forecast. I.e., how much the forecasts vary from a straight line or, to some extent, the homoscedastic assumption. As Figure 37 indicated, when taking the mean over companies, the inclusion of the VIX has made the TCN predictions vary more, as seen from the higher bar of the TCN-MV compared to TCN.

Standard deviation of volatility forecasts

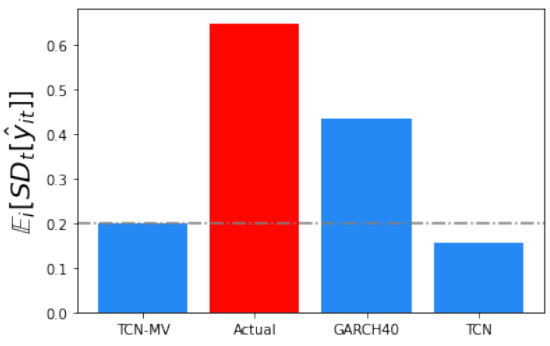


Figure 43: GARCH40 has a higher variance within each time series when averaged over companies. TCN-MV has a higher standard deviation in its forecast compared to TCN. Interestingly, the actual time series has the largest volatility of the selection, meaning all models underestimate volatility.

Table 4 presents the accuracy of the different models as measured by MAPE on our dataset with daily observations from 01.01.2019 to 20.04.2022. Strikingly, according to the MAPE measurement, Naïve outperformed ARIMA by a 1%-point margin or 14%. Using the martingale and efficient hypothesis of markets, Naïve is the holy grail to beat in any market. It keeps much more information about the current state of the economy than what the models of this setup have as inputs. Note that of the non-naïve models, the TCN-X model is the best performing model with a MAPE of 7.12%. Notably, ARIMA with exogenous variables has a colossal loss, far worse than any other model.

7.3.1. Analysis of parametric models

The standard ARIMA model performs only .57 percentage points worse than the best-performing TCN-X. Furthermore, it makes sense that such a simple model would not beat the average dollar on the market, i.e., Naïve. The selected models are highly parsimonious (having relatively few parameters), as displayed in Figure 26.

The large loss seen in Table 4 seems to be a consequence of overfitting training data. As seen in Figure 45, the EBIT number is likely to work great in-sample. When the model starts predicting far into the future, the relationship it found in-sample no longer holds. Then, small $\frac{\partial ebit}{\partial t}$ will have large effects on the prediction since the ARIMAX is a function that, to a large extent, is determined by this single variable. More specifically the ARIMAX model has $\frac{\partial f_{arimax}}{\partial ebit} << -2 \cdot 10^6$.

This tendency by ARIMAX does not seem to make much sense. Given a new company, one would expect the EBIT to correlate with the market cap. However, in the in-sample minimization context of the ARIMAX, it is perfect logic. The MSE minimization happens to fit in-sample with no regard to context. Thus, for the particular fitting having too large market cap weights and reducing them with negative EBIT weights is one way to minimize the ex-post loss. However, as is clear from Table 4, the same does not hold ex-ante on the test data, nor is it something a person with any sense of finance would think.

One interesting aspect of ARIMAX is the ability to look at the coefficients it fitted for the exogenous variables. Large coefficients (properly scaled to account for different magnitudes in different features) can indicate where it finds the most useful information. The results from the ARIMAX models seem dubious. Figure 44 and Figure 45 show some analysis of the ARIMAX models' internals. In short, the analysis consists of inspecting the coefficients the ARIMAX procedure has settled on as the best for the exogenous variables.

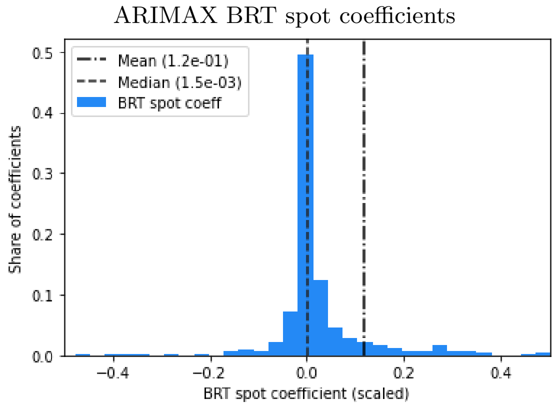


Figure 44: The normalized distribution of the coefficient for the Brent spot price for all ARIMAX models. The coefficients clusters around zero with both mean and median positive, but with a positive skew. We expect a positive bias, as the oil price is one of the most important factors for an oil company's profitability.

In Figure 45 we have plotted the mean coefficients across all companies, adjusted for company size and relative exogenous variable size. The mean of the absolute value of the coefficients seems to make sense until one adds the error bars. With the error bars, the mean values are no longer visible, as the variance in some cases is such that $\pm\sigma$ comprises a large area on both sides of 0.

In Figure 44 we look at the impact of the spot price of Brent oil on the companies. This figure paints a similar picture as Figure 45, with mean and median positive. However, here too, we see a significant variance around 0. This variance, again, makes it hard to argue that these results show a convincing relationship between company value and oil price, as any investor would believe, a priori.

One can observe in Table 4 that the loss achieved with ARIMAX is 8 orders of magnitude larger than all the other results. This colossal loss is because some time series elicits erratic behavior, which causes ARIMAX to break down for some time se-

Model	TW	h	Exogenous	Parametricity	Locality	Input scale	Target scale	Era mode	MAPE	Gap to best	% from best	Best 99%
Naïve	1	20	No	Parametric	Local	None	None	N/A	**6.95%**			**6.80%**
AutoARIMA	240	20	No	Parametric	Local	None	None	N/A	7.93%	0.99%	14.21%	7.44%
ARIMAX	240	20	Yes	Parametric	Local	None	None	N/A	$4.59{\times}10^{8}$%	$4.59{\times}10^{8}$%	$6.61{\times}10^{9}$%	11.17%
TCN	240	20	No	Non-parametric	Global	Min-max	Market cap	Random	7.71%	0.76%	10.98%	7.58%
TCN-X	240	20	Yes	Non-parametric	Global	Min-max	Market cap	Random	7.12%	**0.17%**	**2.45%**	6.97%
LSTM-X	240	20	Yes	Non-parametric	Global	Min-max	Market cap	Random	7.15%	0.20%	2.88%	7.00%

Table 4: Market cap problem: This table presents the best results achieved per model in terms of Mean Absolute Percentage Error (MAPE) and the models' distinguishing attributes. h is the forecasting horizon, and TW is the training window, i.e., the number of time steps into the past the model can see each time it forecasts h time steps into the future.

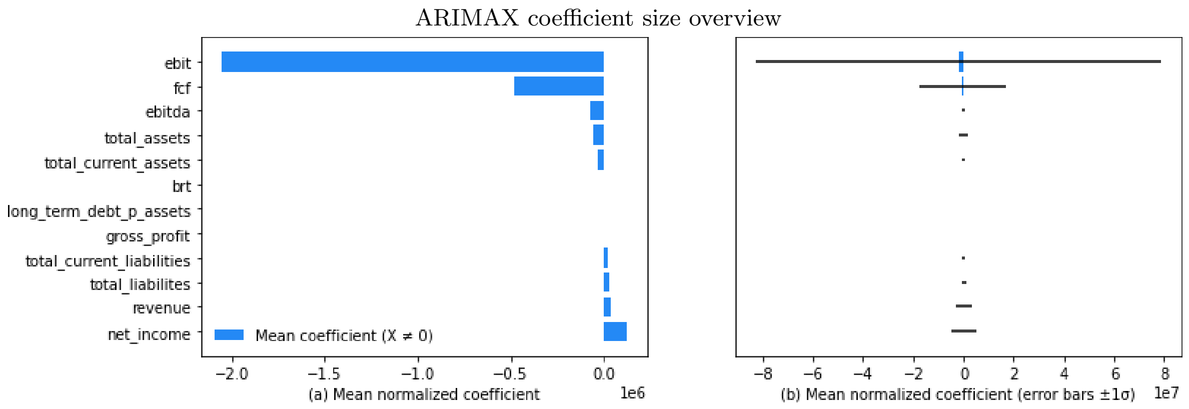


Figure 45: On the left, in panel (a), one can see the mean coefficient of each exogenous variable across all the companies. These values are adjusted for company size and relative coefficient size to make them comparable across companies and between variables. We see here that, on average, an increase in 'ebit' implies an increase in market value, and an increase in 'net_income' is the opposite. This finding seems to make no sense in isolation. Now consider panel (b) to the right, where we have added error bars of $\pm 1\sigma$. The size of the standard deviation completely dwarfs the absolute size of the bars, implying that the variance is enormous and the mean values meaningless.

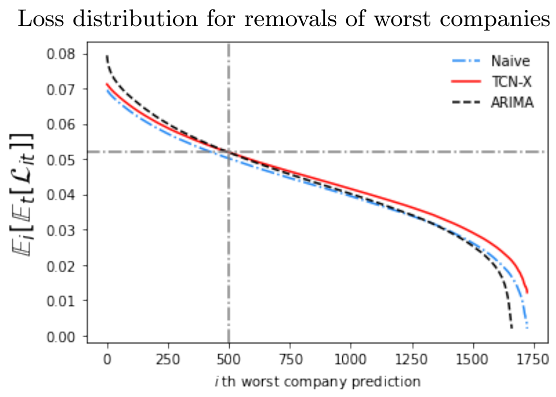


Figure 46: The company mean over time meaned forecasts loss. When we remove the predictions which contribute the most to loss it is evident that TCN-X outperforms ARIMA until the removal of the 500 worst company forecast.

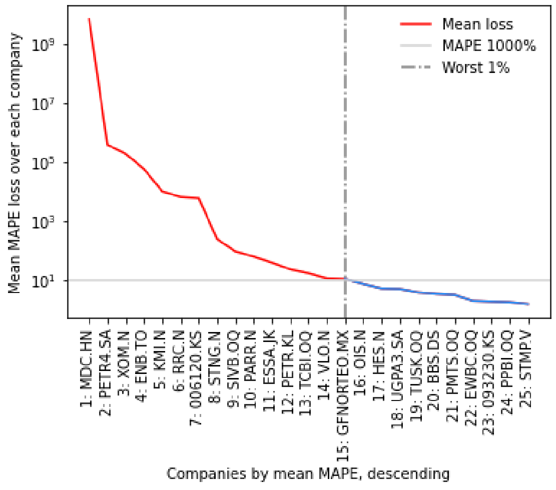


Figure 47: As with many things, the distribution of losses follows a power law curve (Clauset et al., 2009), as the loss produced for the worst-performing ticker is around 1000x worse than the next worse. As such, by removing the 1% of tickers that the model performs the worst for, the model's loss decreases to $\frac{1}{4\times10^7}$ the original value. The other models also elicit similar characteristics, albeit not nearly as extreme as ARIMAX.

ries completely. We find that this is an example of where a more 'stable´ method like ARIMAX potentially causes much more harm than more complex non-parametric methods in the worst case.

To compare the methods better, we investigated the nature of the breakdowns despite this total failure. As is often the case, we found that a minimal amount of time series contributes the vast majority of the error, see Figure 47. The rightmost column in Table 4 shows the MAPE loss for all methods when we removed the companies with the 1% worst mean test loss. A couple of interesting observations emerge. First, the worst-performing models have the most to gain from removing their worst time series. Second, AutoARIMA and TCN have their orders reversed compared to the loss for all series.

7.3.2. Analysis of non-parametric models

Figure 50 shows the feature importance of the inputs to the neural models, or perhaps, the lack thereof. The figure shows bars of $permutation(set(features), i)_{loss} - baseline_{loss} i \in \{1, ...50\}$. When the bar is positive, it means that when we permute the inputs, it leads to a worse prediction. It is evident that shuffling the market cap and three financial statements has such a low influence on the model prediction that we can deduce that the function the neural net is approximating is independent of these inputs. Surely, if a small oil company suddenly reported profits of the size of Saudi Aramco, investors would run to invest in them. This TCN-X model, on the other hand, does not mind and believes the market cap of Small Oil Corp to follow as before.

The reason is that the model has learned to act similar to the naïve forecast. Due to our normalization, the neural network learns to predict a vector that is close to all ones. This prediction is equivalent to naïve (predict mcap_{t-1} and divide by mcap_{t-1}). This result suggests that the prediction by this model is not a function of the inputs. We share some thoughts on why this might be the case in section 7.3.2 and more investigation is needed and encouraged by the wider forecasting community.

Importantly we believe the naïve forecast to be a local minimum with a relatively safe, low loss. Moreover, it is easy to approximate by NNs (set all weights to 0 and keep set last layer bias to 1d tensor of 1's). For our dataset, then, this suggests that if there is information in the $\mathbb{E}_{\mathcal{T}}$, the NNs are not able to find them. This inability could, for instance,

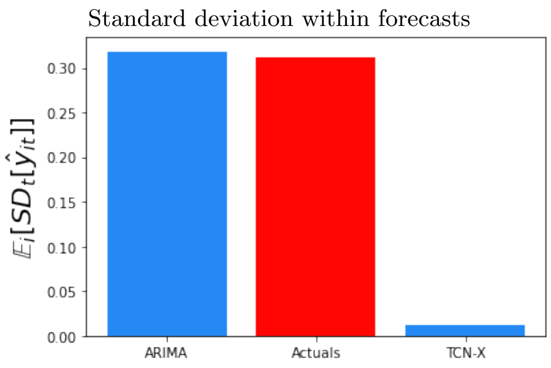


Figure 48: TCN-X has a notably lower standard deviation across time, then taking the mean over all companies. ARIMA has a standard deviation that is higher than that of actuals, which suggest it has overfitted to noise in the training data.

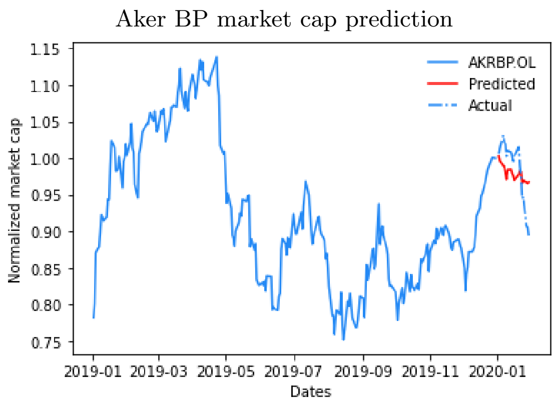


Figure 49: An example prediction by the TCN-X model for January 2020 for Aker BP. Note that this prediction is downwards trending, and independent of all inputs.

be explained by a requirement for even more data manifest in a lower loss, or the neural network representations are not big enough. There could also be signals the model could not find because of too much noise. Our experiments in Figure 13 show that NNs given too much noise in the input of a neural network will have increased time to convergence.

Figure 49 shows the forecasts of TCN-X on AKER BPs market capitalization. It's interesting to note that it is downwards trending. Given that markets tend to rise, it would seem more plausible that this is what the neural network would have learned.

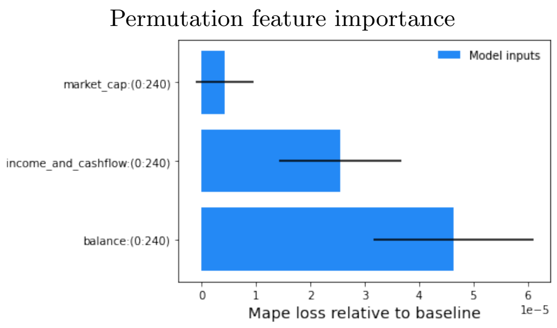


Figure 50: The results from 100 permutations of the market cap, income & cash flow statement and balance. Note that since the neural networks predictions shows close to no loss when input is permuted it implies that the function it is approximating is independent of the all inputs.

8. Problems and further works

8.1. Data quality

We used Refinitiv Eikon as our data source. Despite giving us access to large amounts of data, the data was generally of poor quality. The low quality lead to the authors spending obscene amounts of time pouring over the data, making sense of it, and cleaning what they could. This resulted in several companies being dropped from the dataset in the process. This poses two problems. First, This could introduce some bias in the data, as presumably, data quality is lower for some types of companies (e.g., small companies or companies in less developed economies). Second, the cleaning resulted in fewer data points, which can adversely impact machine learning methods that require enormous amounts of data to effectively generalize. Both of these issues could definitely be mitigated by more work in collecting and cleaning data.

Furthermore, there is a definite chance of there still being erroneous data in the data set. Most likely, it is not too prevalent, but some deep learning models can be very sensitive to extreme values and might be adversely affected.

8.2. Noise in data

Stock prices are naturally highly erratic and do not necessarily represent the intrinsic value of an enterprise. Employing techniques to de-noise the data could be beneficial in this case. Several methods have been proposed, recently, e.g., researchers have applied wavelet transforms (Bao et al., 2017; Liang et al., 2019) and kNN classifiers (Sun et al., 2017) to make the data more well-behaved.

8.3. Data normalization

We have taken care to prepare the data in a manner that both minimizes noise and maximizes signal. Especially in the context of neural networks, this is especially important, as they often expect values centered around 0 with unit variance. We think there are performance gains to be had in normalizing in different ways and augmenting the data through different transformations. There are also some pitfalls here. For instance, we normalized future values by the last period's newest registered market cap but this actually makes the naïve forecast completely independent of inputs. Most neural networks quickly became independent of inputs in this case. This is because the naïve prediction is market cap and thus a normalization by market cap returns a vector of ones. A rather non-trivial effect of a normalization that seemed like it made sense both in an economic and machine learning context.

8.4. Feature selection

We used a total of 37 features for our datacube. This is a relatively large number but in no way exhaustive in terms of what is available in data sources like Eikon or Bloomberg. Furthermore, it is likely that some of our features are pure noise in the context of the chosen forecasting problems and would be better left out. Therefore, further work could investigate the space of available inputs and the relevance in terms of signal to noise for different forecasting tasks.

8.5. Fundamental non-predictability

We are of the belief that different time series are of very different characters, where some are much more predictable than others. Areas where many actors are present and all actors both have a vested interest in the outcome and can affect the outcome will naturally be harder to forecast. This is because more eyes mean more information is already priced in. This represents our third experiment, predicting stock market movements, which also showed the most lacking results. On the other end of the spectrum, one finds things such as weather, traffic, and product demand, which can be forecasted with a much larger degree of confidence. These are fundamentally different time series, and forecasting them might also be strictly more useful for society. Therefore, we think that directing the forecasting efforts towards soluble problems with useful solutions is the most productive.

8.6. Model selection

We have sought to use some of the methods that have proven themselves to be reliably strong in the literature. Still, we humbly admit that there definitely are many approaches that would perform better that we did not try. This concerns the parametric as well as the non-parametric models. We encourage researchers to test the framework and datacube for different models and tasks to see if the theoretical optimum is within reach.

8.7. GARCH fitting

There are some subtleties with regards to the frequency of return inputs and output standard deviation measure. We found one paper on this Chen et al. (2015), where the return frequency can be different from that of the forecast horizon. Unfortunately, neither was any online package readily available to us nor did we have the resources to implement it ourselves. Thus we mean over each GARCH prediction 20 days ahead standard deviation forecast and let that mean equal a month ahead volatility prediction. Arguably, this averaging defeats the exact heteroscedastic modeling purpose of GARCH models.

8.8. ARIMAX's lacking interpretability

There are several reasons why the ARIMA model with exogenous variables (ARIMAX) we used to generate the results in section 6.4.2 had coefficients that seemingly did not bear any usable information, and that none of the parametric models could outperform the Naïve model. One important factor is that ARIMAX cannot model relationships that are not strictly linear in each separate variable. And, since it is highly likely that there are much more complex relationships than that, it is not surprising that the ARIMAX models break down. As an example, a highly indebted company will likely react much more negatively to a drop in the oil price than a company without leverage. One possible solution to this could be to test several transformations of the exogenous variables and apply lasso methods for variable selection.

8.9. Time handling

In our datacube, all data, except the metadata, is treated as having a daily frequency. This includes data from quarterly reports. This implies some level of data duplication along the time axis. In the age of big data, this is not a considerable hurdle. However, there could be constellations of the data of different frequencies that would be more amenable for learning for the models, and we would like to explore this further.

8.10. Loss measures

We have utilized tried and true loss functions in this work. However, there is a vast literature on different loss functions and accuracy measures that we could have explored further. Especially with optimizing the models, they can be sensitive to the particular loss function used to calculate the gradients. There could be gains to be had by a more intelligent choice of loss functions for optimization.

8.11. Computing power

We have based our training rig around the GPUs offered by Google Colab[4]. These are more powerful than what can be done on a laptop, but when running large sweeps of hyperparameter search for large models, one would benefit greatly from having access to much larger amounts of computing power.

[4]colab.research.google.com

9. Conclusion

First, we presented a total ordering of the theoretical lower bound loss of different forecasting paradigms in the following descending order: Model selection, Model combination, Non-parametric univariate models, and Non-parametric multivariate. This shows ex-ante and ex-post that non-parametric and multivariate methods are more expressive and thus have a decreased lower-bound loss compared to other models considered

Second, we create a generalized forecasting framework to test the above forecasting paradigms ex-ante. We implement the framework by creating a novel datacube consisting of daily stock prices and 100k quarterly reports from about 1600 global companies and several daily macro time series, all from 2000 to spring 2022. Lastly, we utilize the framework and show that modern multivariate time series approaches are powerful but domain-dependent. We demonstrate the domain-dependent accuracy by showing convincing results when predicting corporate bankruptcy risk, moderate results when predicting stock price volatility, and lacking results when finally predicting company market capitalization.

Given the domain-dependent convincing results and mostly unrealized theoretical lower bound loss of multivariate approaches, we hope to encourage further research on non-parametric, multi-signal approaches that leverage a wider array of available information.

Lastly, there are several areas where our approach could improve. For example, better data cleaning and augmentation could induce stronger learning. More data on companies from more industries could be added. It could also be highly relevant to apply more computing power and test out a wider array of models and parameters. We also think there might be many related problems to which our framework can be applied with exciting results.

References

Ahmed, N.K., Atiya, A.F., Gayar, N.E., El-Shishiny, H., 2010. An empirical comparison of machine learning models for time series forecasting. Econometric reviews 29, 594–621.

Akaike, H., 1973. Maximum likelihood identification of gaussian autoregressive moving average models. Biometrika 60, 255–265.

Alenezy, A.H., Ismail, M.T., Wadi, S.A., Tahir, M., Hamadneh, N.N., Jaber, J.J., Khan, W.A., 2021. Forecasting stock market volatility using hybrid of adaptive network of fuzzy inference system and wavelet functions. Journal of Mathematics 2021.

Ankile, L.L., Krange, K., 2022. The donut approach to ensemblecombination forecasting. arXiv preprint arXiv:2201.00426 .

Armstrong, J.S., Collopy, F., 1992. Error measures for generalizing about forecasting methods: Empirical comparisons. International journal of forecasting 8, 69–80.

Atsalakis, G.S., Valavanis, K.P., 2009. Surveying stock market forecasting techniques–part ii: Soft computing methods. Expert systems with applications 36, 5932–5941.

Bagshaw, M.L., 1987. Univariate and multivariate arima versus vector autoregression forecasting, working paper 87-06. Working papers of the Federal Reserve Bank of Cleveland .

Bai, S., Kolter, J.Z., Koltun, V., 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 .

on Banking Supervision, B.C., 2017. Basel iii: Finalising post-crisis reforms. Bank for International Settlements .

Bao, W., Yue, J., Rao, Y., 2017. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PloS one 12, e0180944.

Bollerslev, T., 1986. Generalized autoregressive conditional heteroskedasticity. Journal of econometrics 31, 307–327.

Box, G.E., 1970. Gm jenkins time series analysis: Forecasting and control. San Francisco, Holdan-Day .

Box, G.E., 1994. Time series analysis: forecasting and control. Technical Report. Princeton University.

Brandt, J.A., Bessler, D.A., 1984. Forecasting with vector autoregressions versus a univariate arima process: An empirical example with us hog prices. North Central Journal of Agricultural Economics , 29–36.

Brooks, C., 2019. Introductory Econometrics for Finance. 4 ed., Cambridge University Press. doi:10.1017/9781108524872.

Brown, R.G., 2004. Smoothing, forecasting and prediction of discrete time series. Courier Corporation.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020. Language models are few-shot learners. Advances in neural information processing systems 33, 1877–1901.

Brækhus, S., 2017. Insolvens. URL: `https://snl.no/insolvens`.

Cauchy, A., et al., 1847. Méthode générale pour la résolution des systemes d'équations simultanées. Comp. Rend. Sci. Paris 25, 536–538.

Chakraborty, K., Mehrotra, K., Mohan, C.K., Ranka, S., 1992. Forecasting the behavior of multivariate time series using neural networks. Neural networks 5, 961–970.

Chaudhuri, T.D., Ghosh, I., 2016. Artificial neural network and time series modeling based approach to forecasting the exchange rate in a multivariate framework. arXiv preprint arXiv:1607.02093 .

Chen, C., Twycross, J., Garibaldi, J.M., 2017a. A new accuracy measure based on bounded relative error for time series forecasting. PloS one 12, e0174202.

Chen, W., Zhang, Y., Yeo, C.K., Lau, C.T., Lee, B.S., 2017b. Stock market prediction using neural network through news on online social networks, in: 2017 international smart cities conference (ISC2), IEEE. pp. 1–6.

Chen, X., Ghysels, E., Wang, F., 2015. Hybrid-garch: A generic class of models for volatility predictions using high frequency data. Statistica Sinica , 759–786.

Clauset, A., Shalizi, C.R., Newman, M.E., 2009. Power-law distributions in empirical data. SIAM review 51, 661–703.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al., 2022. Magnetic control of tokamak plasmas through deep reinforcement learning. Nature 602, 414–419.

Eakins, S.G., Stansell, S.R., 2003. Can value-based stock selection criteria yield superior risk-adjusted returns: an application of neural networks. International Review of Financial Analysis 12, 83–97.

Engle, R., 2001. Garch 101: The use of arch/garch models in applied econometrics. Journal of economic perspectives 15, 157–168.

Engle, R.F., 1982. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. Econometrica: Journal of the econometric society , 987–1007.

Enns, P.G., Machak, J.A., Spivey, W.A., Wrobleski, W.J., 1982. Forecasting applications of an adaptive multiple exponential smoothing model. Management Science 28, 1035–1044.

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

Fortin, A.P., Simonato, J.G., Dionne, G., 2022. Forecasting expected shortfall: Should we use a multivariate model for stock market factors? International Journal of Forecasting .

Gandhmal, D.P., Kumar, K., 2019. Systematic analysis and review of stock market prediction techniques. Computer Science Review 34, 100190.

Gers, F.A., Schmidhuber, J., Cummins, F., 2000. Learning to forget: Continual prediction with lstm. Neural computation 12, 2451–2471.

Geva, T., Zahavi, J., 2014. Empirical evaluation of an automated intraday stock recommendation system incorporating both market data and textual news. Decision support systems 57, 212–223.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. MIT press.

Granger, C.W., Poon, S.H., 2001. Forecasting financial market volatility: A review. Available at SSRN 268866 .

Greene, W., 2018. Econometric Analysis. Pearson.

Guiguet, V., Baskiotis, N., Guigue, V., Gallinari, P., 2018. Context-aware forecasting for multivariate stationary time-series. Under review for ICLR 2019 conference .

Harrison, B., Moore, W., 2012. Forecasting stock market volatility in central and eastern european countries. Journal of forecasting 31, 490–503.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9, 1735–1780.

Hoeffding, W., 1963. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58, 13–30. URL: http://www.jstor.org/stable/2282952.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural networks 2, 359–366.

Hu, M.J.C., 1964. Application of the adaline system to weather forecasting. Ph.D. thesis. Department of Electrical Engineering, Stanford University.

Huang, Y., Capretz, L.F., Ho, D., 2021. Machine learning for stock prediction based on fundamental analysis, in: 2021 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE. pp. 01–10.

Hussain, A.J., Knowles, A., Lisboa, P.J., El-Deredy, W., 2008. Financial time series prediction using polynomial pipelined neural networks. Expert Systems with Applications 35, 1186–1199.

Jiang, W., 2021. Applications of deep learning in stock market prediction: recent progress. Expert Systems with Applications 184, 115537.

Kainth, A., Wahlstrøm, R.R., 2021. Do ifrs promote transparency? evidence from the bankruptcy prediction of privately held swedish and norwegian companies. Journal of Risk and Financial Management 14, 123.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Kongcharoen, C., Kruangpradit, T., 2013. Autoregressive integrated moving average with explanatory variable (arimax) model for thailand export, in: 33rd International Symposium on Forecasting, South Korea, pp. 1–8.

Krawczyk, B., 2016. Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence 5, 221–232.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25.

Lam, M., 2004. Neural network techniques for financial performance prediction: integrating fundamental and technical analysis. Decision support systems 37, 567–581.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. nature 521, 436–444.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. Neural computation 1, 541–551.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S.S., Pennington, J., Sohl-Dickstein, J., 2017. Deep neural networks as gaussian processes. arXiv preprint arXiv:1711.00165 .

Lemke, C., Gabrys, B., 2010. Meta-learning for time series forecasting and forecast combination. Neurocomputing 73, 2006–2016.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A., 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research 18, 6765–6816.

Liang, X., Ge, Z., Sun, L., He, M., Chen, H., 2019. Lstm with wavelet transform based data preprocessing for stock price prediction. Mathematical Problems in Engineering 2019.

Lim, B., Zohren, S., 2021. Time-series forecasting with deep learning: a survey. Philosophical Transactions of the Royal Society A 379, 20200209.

Lim, C.M., Sek, S.K., 2013. Comparing the performances of garch-type models in capturing the stock market volatility in malaysia. Procedia Economics and Finance 5, 478–487.

Lin, Z., 2018. Modelling and forecasting the stock market volatility of sse composite index using garch models. Future Generation Computer Systems 79, 960–972.

Makridakis, S., 2022. The m6 financial forecasting competition. URL: https://m6competition.com/.

Makridakis, S., Hibon, M., 2000. The m3-competition: results, conclusions and implications. International journal of forecasting 16, 451–476.

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2018a. The m4 competition: Results, findings, conclusion and way forward. International Journal of Forecasting 34, 802–808.

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2018b. The m4 competition: Results, findings, conclusion and way forward. International Journal of Forecasting 34, 802–808. URL: https://www.sciencedirect.com/science/article/pii/S0169207018300785, doi:https://doi.org/10.1016/j.ijforecast.2018.06.001.

Mandelbrot, B.B., 1997. The variation of certain speculative prices, in: Fractals and scaling in finance. Springer, pp. 371–418.

Mayhew, S., 1995. Implied volatility. Financial Analysts Journal 51, 8–20.

Minsky, M., Papert, S.A., 2017. Perceptrons: An introduction to computational geometry. MIT press.

Moen, P.A., 2020. Bankruptcy prediction for Norwegian enterprises using interpretable machine learning models with a novel timeseries problem formulation. Master's thesis. NTNU.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R.J., Talagala, T.S., 2020. Fforma: Feature-based forecast model averaging. International Journal of Forecasting 36, 86–92.

Montero-Manso, P., Hyndman, R.J., 2021. Principles and algorithms for forecasting groups of time series: Locality and globality. International Journal of Forecasting 37, 1632–1653. URL: https://www.sciencedirect.com/science/article/pii/S0169207021000558, doi:https://doi.org/10.1016/j.ijforecast.2021.03.004.

Nazário, R.T.F., e Silva, J.L., Sobreiro, V.A., Kimura, H., 2017. A literature review of technical analysis on stock markets. The Quarterly Review of Economics and Finance 66, 115–126.

Niaki, S.T.A., Hoseinzade, S., 2013. Forecasting s&p 500 index using artificial neural networks and design of experiments. Journal of Industrial Engineering International 9, 1–9.

Nonejad, N., 2017. Forecasting aggregate stock market volatility using financial and macroeconomic predictors: Which models forecast best, when and why? Journal of Empirical Finance 42, 131–154.

Nti, I.K., Adekoya, A.F., Weyori, B.A., 2020. A systematic review of fundamental and technical analysis of stock market predictions. Artificial Intelligence Review 53, 3007–3057.

Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y., 2019. N-beats: Neural basis expansion analysis for interpretable time series forecasting. arXiv preprint arXiv:1905.10437 .

Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y., 2020. Meta-learning framework with applications to zero-shot time-series forecasting. CoRR abs/2002.02887. URL: https://arxiv.org/abs/2002.02887, arXiv:2002.02887.

Pantiskas, L., Verstoep, K., Bal, H., 2020. Interpretable multivariate time series forecasting with temporal attention convolutional neural networks, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE. pp. 1687–1694.

Pawlikowski, M., Chorowska, A., 2020. Weighted ensemble of statistical models. International Journal of Forecasting 36, 93–97. URL: `https://www.sciencedirect.com/science/article/pii/S0169207019301190`, doi:`https://doi.org/10.1016/j.ijforecast.2019.03.019`. m4 Competition.

Peter, Ď., Silvia, P., 2012. Arima vs. arimax–which approach is better to analyze and forecast macroeconomic time series, in: Proceedings of 30th international conference mathematical methods in economics, pp. 136–140.

Petropoulos, F., Apiletti, D., Assimakopoulos, V., Babai, M.Z., Barrow, D.K., Taieb, S.B., Bergmeir, C., Bessa, R.J., Bijak, J., Boylan, J.E., et al., 2020. Forecasting: theory and practice. arXiv preprint arXiv:2012.03854 .

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. nature 323, 533–536.

Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T., 2020. Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting 36, 1181–1191. URL: `https://www.sciencedirect.com/science/article/pii/S0169207019301888`, doi:`https://doi.org/10.1016/j.ijforecast.2019.07.001`.

Senior, A.W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A.W., Bridgland, A., et al., 2020. Improved protein structure prediction using potentials from deep learning. Nature 577, 706–710.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al., 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 362, 1140–1144.

Smith, T.G., et al., 2017–. pmdarima: Arima estimators for Python. URL: `http://www.alkaline-ml.com/pmdarima`. [Online; accessed ¡today¿].

Snoek, J., Larochelle, H., Adams, R.P., 2012. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems 25.

Stein, R.M., 2005. The relationship between default prediction and lending profits: Integrating roc analysis and loan pricing. Journal of Banking & Finance 29, 1213–1236.

Stock, J.H., Watson, M.W., 1988. Testing for common trends. Journal of the American statistical Association 83, 1097–1107.

Stock, J.H., Watson, M.W., 2002. Forecasting using principal components from a large number of predictors. Journal of the American Statistical Association 97, 1167–1179. URL: `http://www.jstor.org/stable/3085839`.

Sun, H., Rong, W., Zhang, J., Liang, Q., Xiong, Z., 2017. Stacked denoising autoencoder based stock market trend prediction via k-nearest neighbour data selection, in: International Conference on Neural Information Processing, Springer. pp. 882–892.

Tay, F.E., Cao, L., 2001. Application of support vector machines in financial time series forecasting. omega 29, 309–317.

Taylor, J.W., 2004. Volatility forecasting with smooth transition exponential smoothing. International Journal of Forecasting 20, 273–286.

Tiao, G.C., Tsay, R.S., 1989. Model specification in multivariate time series. Journal of the Royal Statistical Society: Series B (Methodological) 51, 157–195.

Toda, H., 1991. Vector autoregression and causality. Ph.D. thesis. Yale University.

Vagropoulos, S.I., Chouliaras, G., Kardakos, E.G., Simoglou, C.K., Bakirtzis, A.G., 2016. Comparison of sarimax, sarima, modified sarima and ann-based models for short-term pv generation forecasting, in: 2016 IEEE International Energy Conference (ENERGYCON), IEEE. pp. 1–6.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.

Wan, R., Mei, S., Wang, J., Liu, M., Yang, F., 2019. Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. Electronics 8, 876.

Werbos, P.J., 1988. Generalization of backpropagation with application to a recurrent gas market model. Neural networks 1, 339–356.

Wolpert, D.H., 1996. The lack of a priori distinctions between learning algorithms. Neural computation 8, 1341–1390.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1, 67–82.

Zhang, G., Patuwo, B.E., Hu, M.Y., 1998. Forecasting with artificial neural networks:: The state of the art. International journal of forecasting 14, 35–62.

Zhang, Y.F., Thorburn, P.J., Fitch, P., 2019. Multi-task temporal convolutional network for predicting water quality sensor data, in: International Conference on Neural Information Processing, Springer. pp. 122–130.

Index	Name	Type	Frequency	Description
0	market_cap	Stock	Daily	Stock price
1	global_relative	Fundamental	Quarterly	Market cap normalized to all companies
2	peers_relative	Fundamental	Quarterly	Market cap normalized to all comapnies in current timeframe
3	revenue	Fundamental	Quarterly	Reported revenue
4	gross_profit	Fundamental	Quarterly	Reported gross profit
5	ebitda	Fundamental	Quarterly	EBITDA
6	ebit	Fundamental	Quarterly	EBIT
7	net_income	Fundamental	Quarterly	Net income
8	fcf	Fundamental	Quarterly	Free cashflow
9	total_current_assets	Fundamental	Quarterly	Total current assets
10	total_liabilities	Fundamental	Quarterly	Total liabilities
11	total_current_liabilities	Fundamental	Quarterly	Total current liabilities
12	long_term_debt_p_assets	Fundamental	Quarterly	Long term debt share of revenue
13	short_term_debt_p_assets	Fundamental	Quarterly	Short term debt share of revenue
14	gross_profit_p_revenue	Fundamental	Quarterly	Gross profit debt share of revenue
15	ebitda_p_revenue	Fundamental	Quarterly	EBITDAshare of revenue
16	ebit_p_revenue	Fundamental	Quarterly	EBITshare of revenue
17	net_income_p_revenue	Fundamental	Quarterly	Net income share of revenue
18	fcf_p_revenue	Fundamental	Quarterly	Free cashflow share of revenue
19	brt	Macro	Daily	Brent oil spot price
20	clc_1	Macro	Daily	Gas future 1 month
21	clc_12	Macro	Daily	Gas future 12 months
22	clc_60	Macro	Daily	Gas future 5 years
23	wtc_1	Macro	Daily	Light oil futures 1 month
24	wtc_12	Macro	Daily	Light oil futures 12 months
25	wtc_60	Macro	Daily	Light oil futures 5 years
26	wti_ref	Macro	Daily	West Texas oil reference index
27	dub_ref	Macro	Daily	Dublin oil reference index
28	cn_10y	Macro	Daily	Chinese 10 year interest rate
29	us_10y	Macro	Daily	US 10 year interest rate
30	de_10y	Macro	Daily	German 10 year interest rate
31	gb_10y	Macro	Daily	British 10 year interest rate
32	high_yield_bond	Macro	Daily	US treasury bond
33	vix	Macro	Daily	Volatility index
34	eur_fx	Macro	Daily	EUR to USD
35	gbp_fx	Macro	Daily	GBP to USD
36	cny_fx	Macro	Daily	CNY to USD

Table A.5: A summary of all features included in the datacube.

Appendix A. List of variables

Appendix B. Data feature visualization

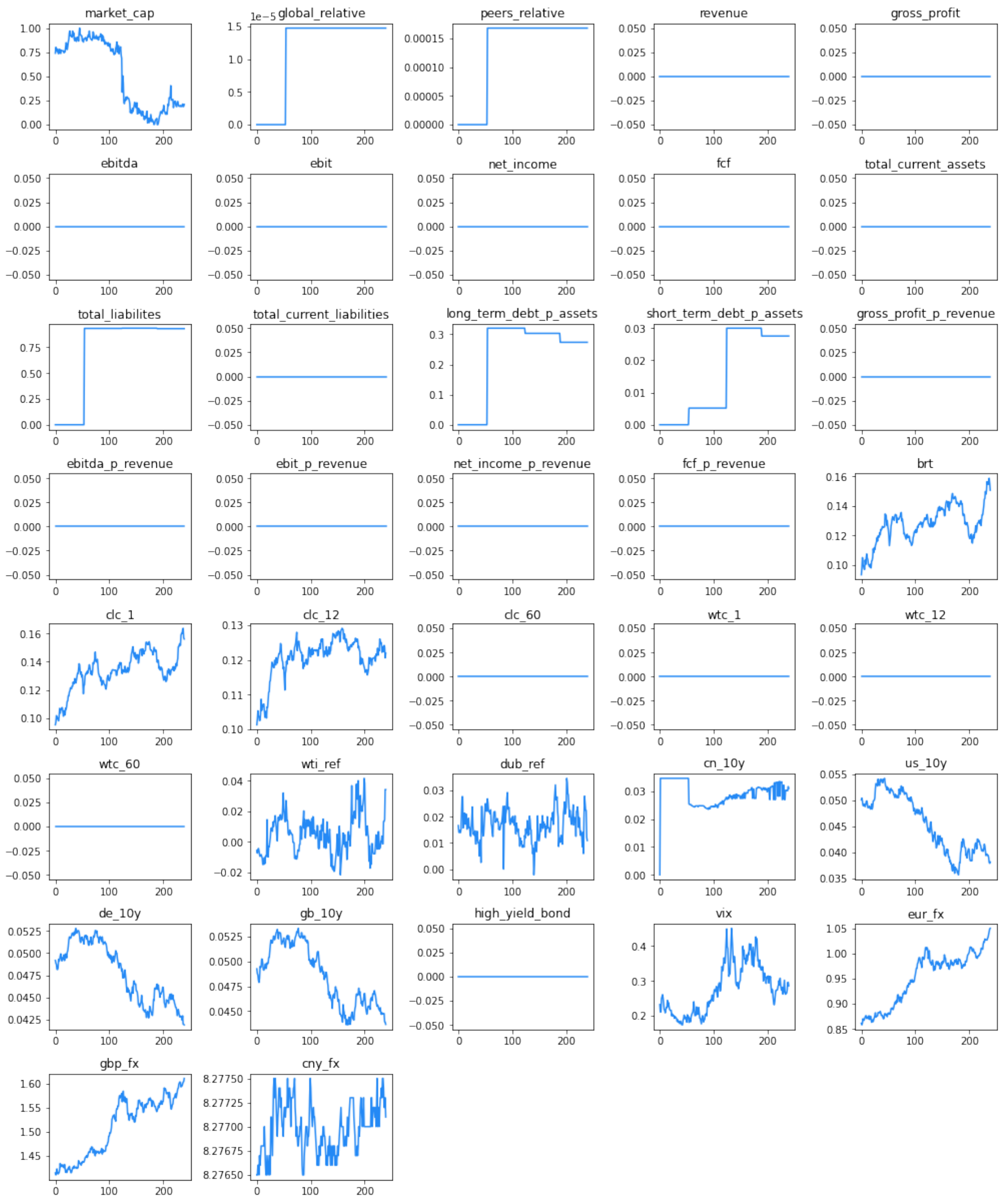


Figure B.51: Overview of the different features part 1.

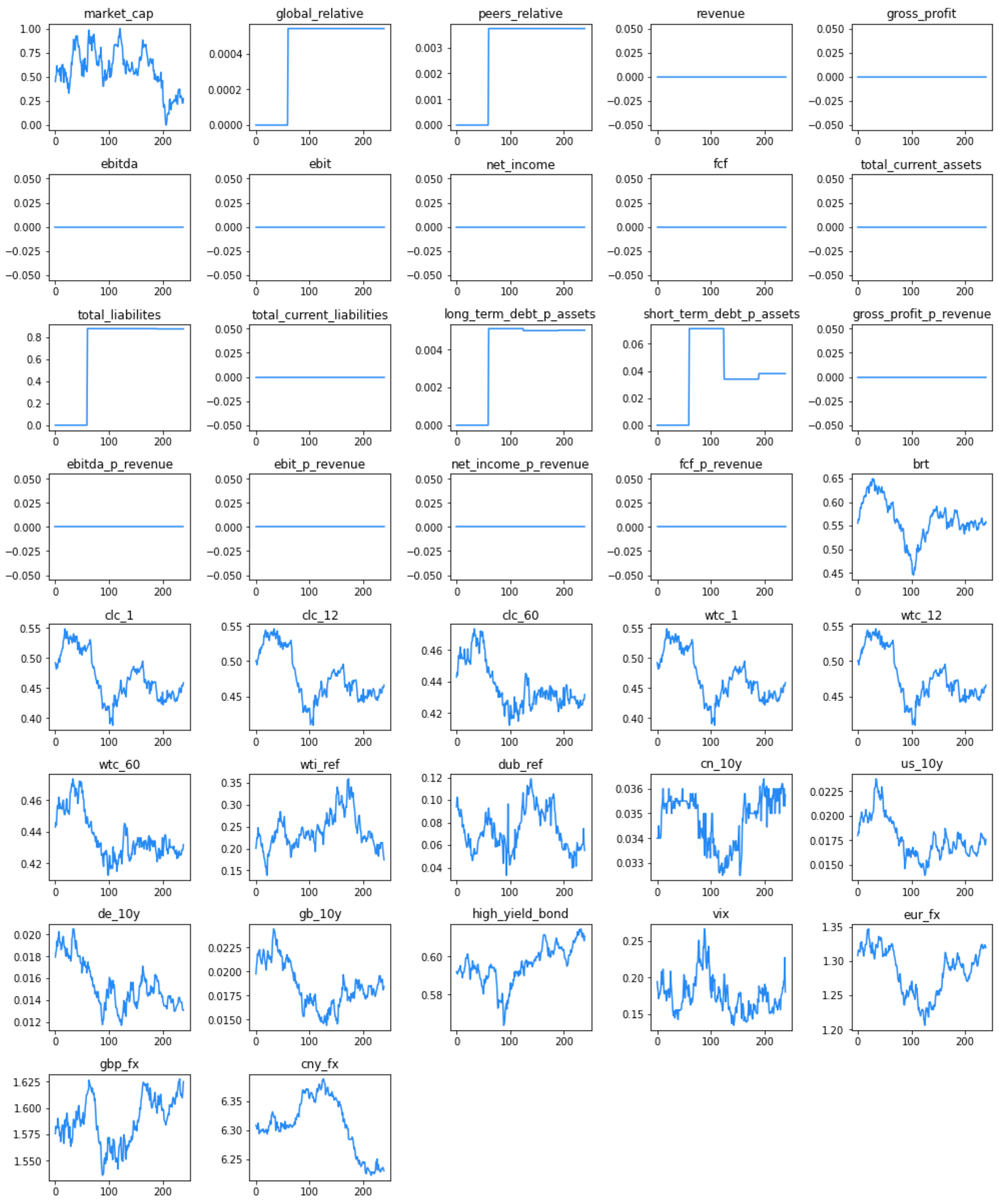


Figure B.52: Overview of the different features part 1.

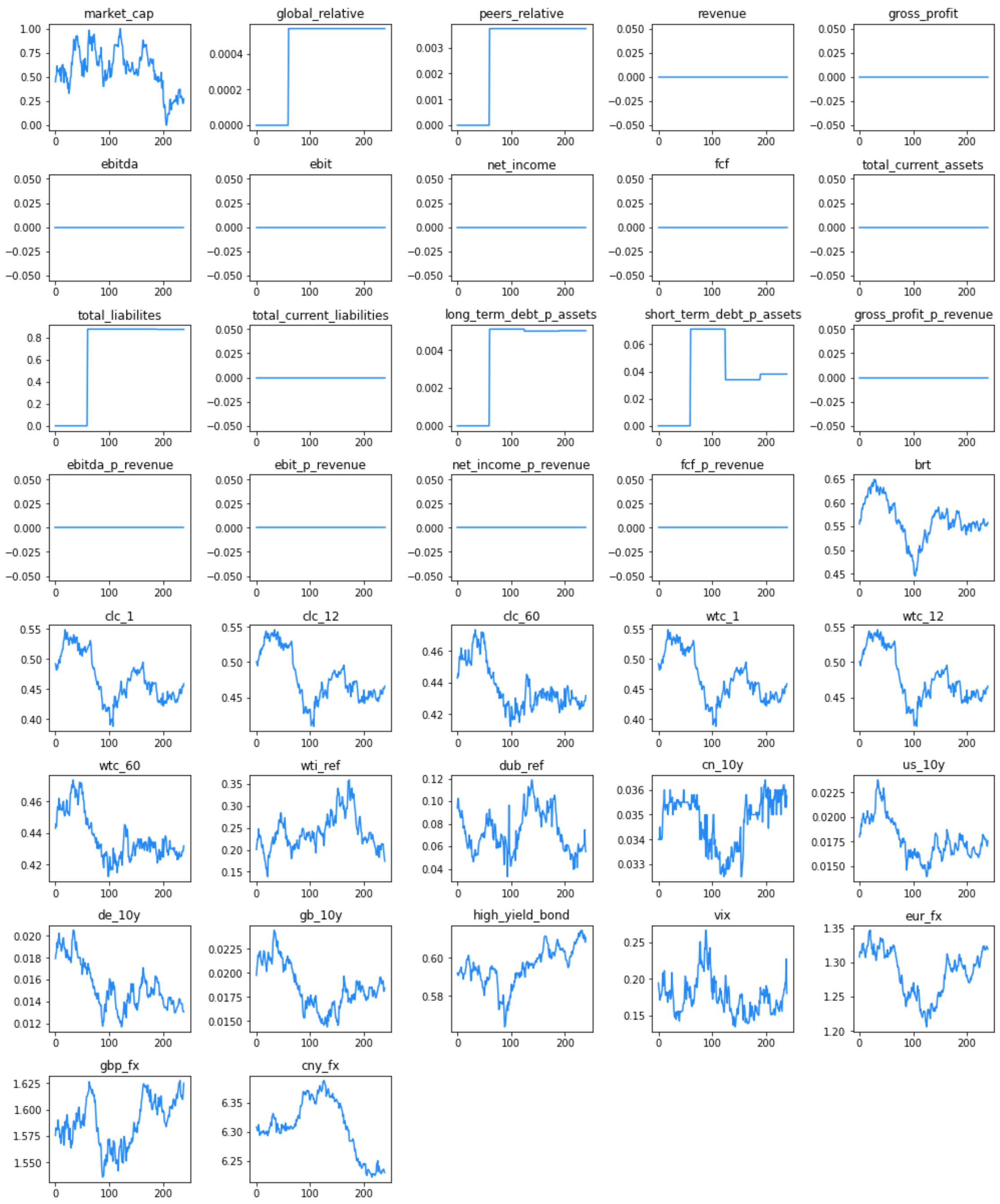


Figure B.53: Overview of the different features part 1.

Appendix C. Selected Relevant Python Implementations

Appendix C.1. Mean absolute percentage error

```
def mape_loss(target: torch.Tensor, y_pred: torch.Tensor) -> torch.Tensor:
    mask = (~target.isnan()) & target.isfinite()
    denom = mask.sum(dim=1)
    target[target != target] = 0
    l = ((
        ((y_pred - target).abs() / (target.abs() + 1e-6) * mask)
    ).sum(dim=1) / denom).mean()
    return l.clamp(min=0, max=10_000)
```

Appendix C.2. Standard deviation mean squared error

```
def std_loss_diff_mse(target: torch.Tensor, y_pred: torch.Tensor) -> torch.Tensor:
    # y_t/y_k-y_{t-1}/y_k => (y_t-y_{t-1})/y_k * y_k/y_{t-1} = (y_t-y_{t-1})/y_{t-1}
    target_ = target.diff() * (target[:, :-1] ** (-1))

    mask = (~target_.isnan()) & (target_.abs() <= 10)
    mask2 = mask.sum(dim=1, keepdim=True) >= 2

    target_[~mask] = 0

    denom = mask.sum(dim=1, keepdim=True)
    denom2 = mask2.sum(dim=0).item()

    l = ((torch.nan_to_num(
            (torch.sum(((target_ - torch.sum(target_, dim=1, keepdim=True) / denom)
            * mask) ** 2, dim=1, keepdim=True) / denom) ** (1 / 2) - y_pred)
            * mask2
        )** 2
    ).sum() / denom2

    return l
```

Appendix C.3. Binary cross entropy with missing data

```
def three_balance_to_bankrupt(three_targets):
    # feature order: total_current_assets/total_assets
    # total_liabilites/total_assets
    # total_current_liabilities/total_assets
    assets_to_liab = 1 / three_targets[:, 1, :]
    asset_prob = assets_to_liab < 1

    curr_assets_to_liab = three_targets[:, 0, :] / three_targets[:, 2, :]
    curr_asset_prob = curr_assets_to_liab < 1

    both_problems = curr_asset_prob * asset_prob

    target = torch.sum(both_problems, dim=1, keepdim=True) > 1
    return target
```

```
def cross_entropy_bankruptcy(
    three_targets: torch.Tensor, y_pred: torch.Tensor
) -> torch.Tensor:
    target = three_balance_to_bankrupt(three_targets)

    weights = 1 / (
        target.sum().div(len(target)) * target
        + (~target).sum().div(len(target)) * (~target)
    )  # sheeeeeshhhh

    weights /= weights.sum()

    return nn.functional.binary_cross_entropy(
        torch.sigmoid(y_pred), target.to(torch.float32), weight=weights
    )
```

Model	TW	h	Exogenous	Parametricity	Locality	Input scale	Target scale	Era mode	MAPE	Gap to best	% from best	Best 99%
Naïve	1	240	No	Parametric	Local	None	None	N/A	**28.97%**			**26.78%**
AutoARIMA	240	240	No	Parametric	Local	None	None	N/A	49.90%	20.93%	72.26%	37.99%
ARIMAX	240	240	Yes	Parametric	Local	None	None	N/A	$2.39{\times}10^9$%	$2.39{\times}10^9$%	$8.24{\times}10^9$%	75.15%
TCN-X	240	240	Yes	Non-parametric	Global	Min-max	Market cap	Random	36.92%	**7.95%**	**27.45%**	33.98%

Table D.6:

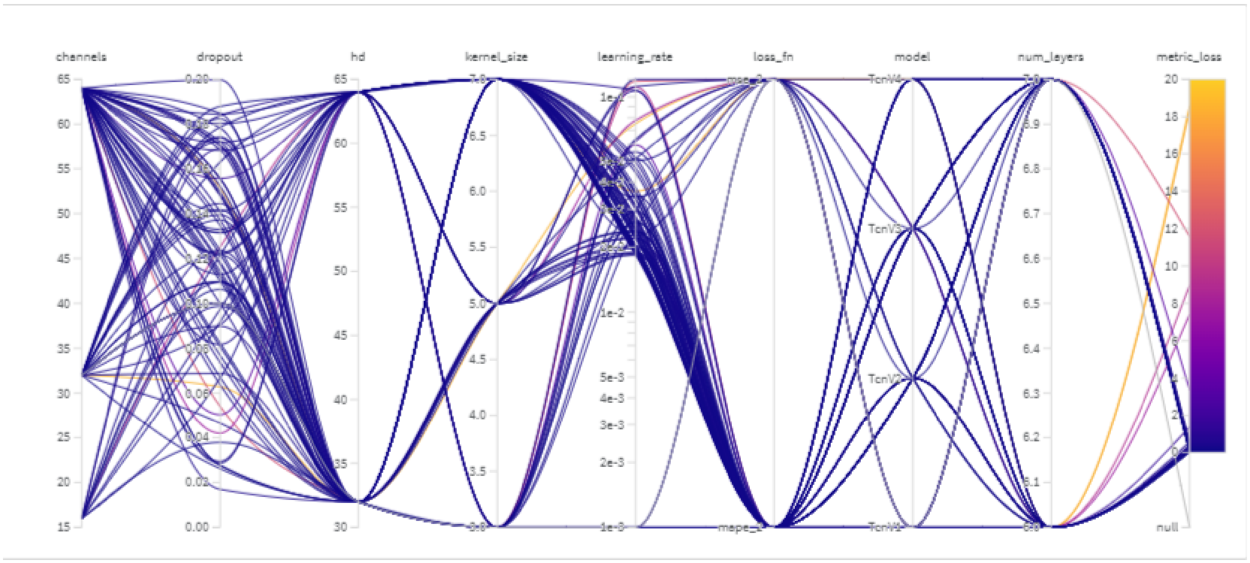


Figure E.54: Visualization of some of the hyperparameter combinations tried for market cap prediction and associated metric loss.

Appendix D. Results for longer time horizon (h = 204)

Appendix E. Hyperparameter search

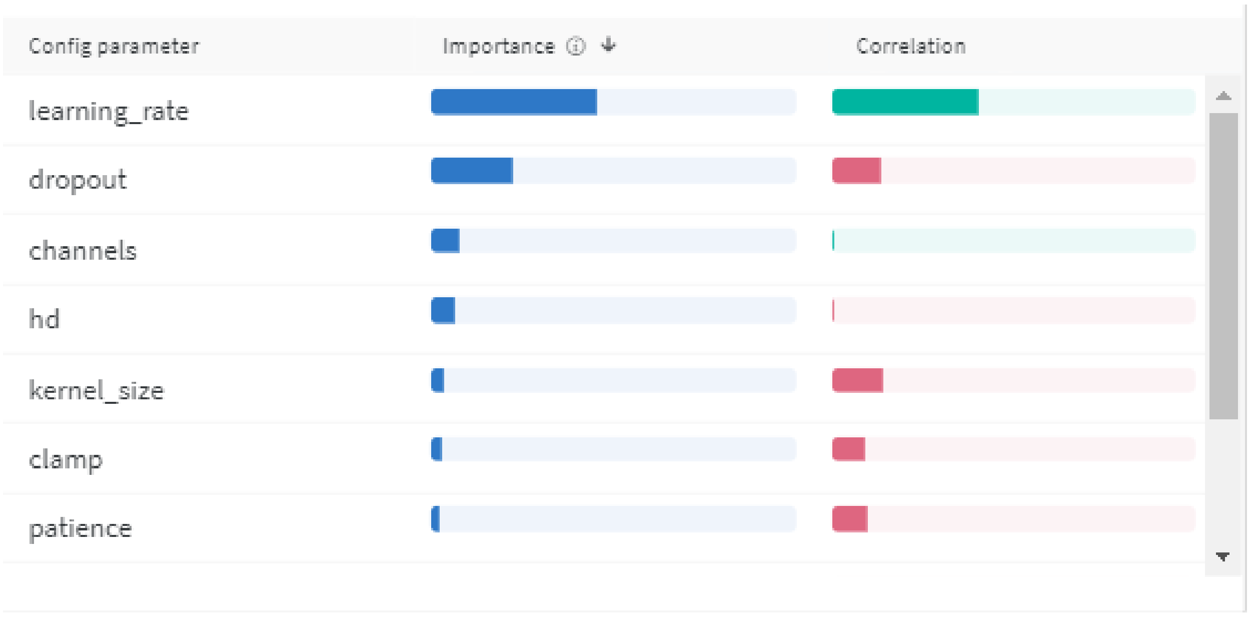

Config parameter | Importance | Correlation
learning_rate
dropout
channels
hd
kernel_size
clamp
patience

Figure E.55: Visualization of some of the hyperparameter combinations tried for market cap prediction and associated metric loss.

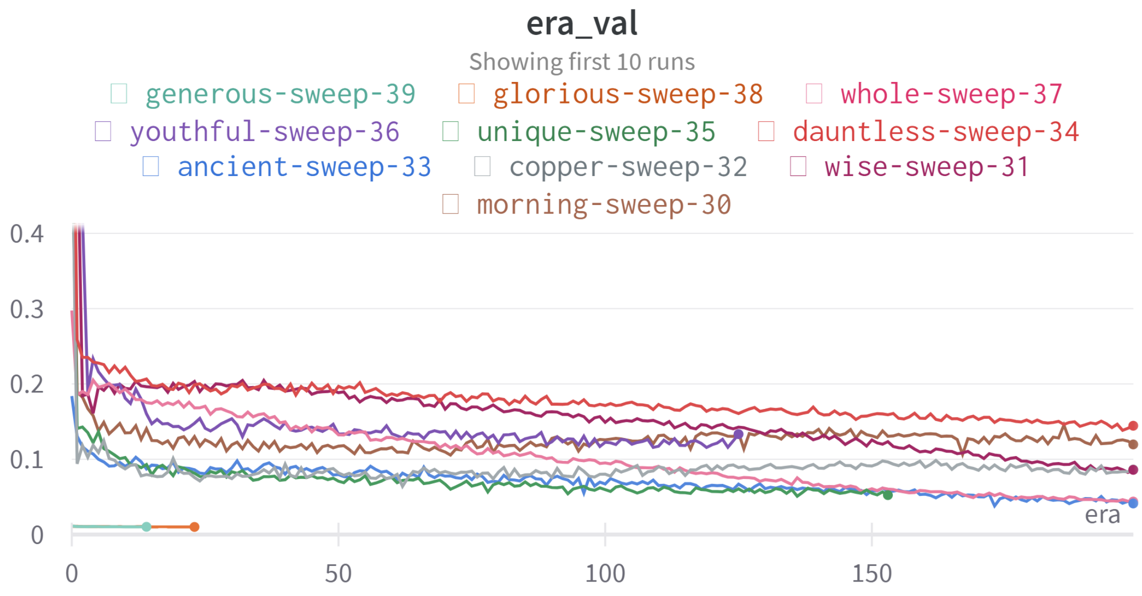


Figure E.56: Overview of a sweep for volatility prediction. The era validation is steadily decreasing but seems to plateau rather quickly.

Ankile & Krange

Exploration of Forecasting Paradigms and a Generalized Forecasting Framework

NTNU

Norwegian University of
Science and Technology