Machine Learning for Time Series Prediction in the Financial Industry

Arturo Buitrago Méndez

Master's thesis

Machine Learning for Time Series Prediction in the Financial Industry

Arturo Buitrago Méndez

September 30, 2019

Chair for Data Processing
Technische Universität München

Arturo Buitrago Méndez. *Machine Learning for Time Series Prediction in the Financial Industry.* Master's thesis, Technische Universität München, Munich, Germany, 2019.

Supervised by Prof. Dr.-Ing. Klaus Diepold and Matthias Kissel; submitted on September 30, 2019 to the Department of Electrical and Computer Engineering of the Technische Universität München.

Chair for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de/`.

Abstract

Time-series prediction of the performance of retail businesses is an area of interest for every agent along the card payment ecosystem and its value chain. Accurate forecasting of merchants' transactions can be used to evaluate the business' health and provide decision makers with valuable insights. Machine learning for time-series prediction has awoken a significant amount of interest; recent research in the field has been promising but its replicability and the validity of its results has been thrown into question. This work provides a basis by which to generate value from existing datasets, uses reproducible techniques and presents robust and valid results.

This work sets out to test whether machine learning methods can predict the transactions of small to medium-sized merchants more accurately than statistical methods — the de facto standard. It also seeks to identify the best possible predictor for the selected task. The author first conducted a wide-ranging literature research and created a list of 21 predictors and implemented them. I also defined statistical benchmarks and measurements by which to compare them against each other. The predictors were submitted to extensive cross-validation and their predictive performance was evaluated. The five highest scoring algorithms after this stage were then submitted to an exhaustive hyperparameter grid search to achieve more precise point predictions. The top predictor was clearly identified.

Eleven machine learning predictors outperformed all statistical benchmarks; no neural network-based approaches cleared this bar. K-nearest neighbor regression was found to most accurately predict merchants' transactions. Gradient tree boosting was deemed the best result to form the basis for a future prediction system due to its robustness, scalability and the availability of libraries dedicated to gradient boosting approaches.

Contents

1 Introduction

1.1 Technical Background

Time series are discrete, temporally ordered sequences of measurements y_t separated by time intervals Δt [3]. These values may be sampled directly, as in the measurement of a sensor, or result from the aggregation of multiple values, as is the case with the sales total of a retail product for a given year. Much of time series analysis is concerned with forecasting — i.e. the prediction of future values of y_t. Time series forecasting is prevalent in a variety of fields: governmental monetary and fiscal policy are often governed by growth forecasts, whereas weather forecasts tell us whether it makes sense to carry an umbrella for the day.

Time series prediction has traditionally been dominated by linear statistical methods such as the auto-regressive (AR) and moving average (MA) models like the ones proposed by Box and Jenkins in their seminal book "Time Series Analysis: Forecasting and Control" [12]. These approaches often model the seasonality and linearity of trends separately after finding a representation of the data that is stationary. For a more detailed description of linear methods, please refer to chapter 2.

Machine learning models have been proposed for the analysis and prediction of time-series as early as the 1960s [44]. They are especially promising for nonlinear phenomena, since they do not assume the linearity of the processes underlying the time series [88]. Despite some skepticism from more traditional researchers (see [59] for a dismissive view particularly on neural networks), the empirical accuracy of machine learning approaches has been well documented [10]. Most approaches still lag behind traditional linear time series approaches when looking at results spanning multiple different time-series [54]. Performance depends, however, on the characteristics of the data series to be analyzed [67]. This observation dovetails nicely with standard machine learning concepts such as hyperparameter tuning and feature engineering.

In this work, I aim to apply machine learning time-series prediction methods to a dataset of financial indicators of small to medium-sized businesses. I have at my disposal significant quantities of data on daily transactions volumes, at the granularity of single purchases.

1.2 Motivation for Selected Use Case

E-commerce has enjoyed steady growth and today accounts for just over 10% of global retail sales. It is expected to reach 13.7% by the end of 2019 [26]. This of course means that the remaining 86.3% will happen face to face, merchant to shopper. In fact, the popularity of brick-and-mortar shops has increased, up 10% in 2018 compared to the previous year [68]. For clarity's sake I will be referring to these businesses as “merchants" throughout this work.

These brick-and-mortar merchants are driving the rise of cashless payments. Traditionally, this has meant the adoption of card payment schemes, such as debit and credit cards — increasingly it also includes purely digital payment services such as Apple Pay or Alipay [16]. The adoption is fueled by benefits including significant cost reduction, the ability to maintain reliable records of transactions and better customer relationships. This phenomenon is especially important for customer-facing small and medium-sized enterprises (SMEs) [24].

So-called payment service providers (PSPs) offer the means by which businesses can accept card payments at all — they often are the ones to deal with the merchant directly . This category increasingly includes providers of software solutions for the “Point of Sale", which replace traditional cash registers. All of these actors are especially incentivized to maximize the number of payments that go through their system from each merchant — since most rely on fees from individual purchases [62]. The amount relies most critically on two factors. The first one is the reliability of the system on-site: the card reader, connection to the payment network and use by the merchant's employees must all work in unison. Secondly, the business must be doing well on its own; a failing business will not generate many transactions, cashless or otherwise.

Predicting the future “health” of these merchants by the information present in its previous transactions would yield great benefits for both parties involved. Predicting transaction volume essentially amounts to demand forecasting on the PSP's system, an integral part of standard business operations management [59]. It would enable the scheduling of rising transaction load, either through organic merchant growth or times of higher trade volume (e.g. Christmas in most of the Western hemisphere). Forecasting would also enable proactive customer support to the merchant when doing unseasonably poorly or even give hints as to anomalies in a store's card payment setup.

As mentioned in section 1.1, machine learning promises to provide a robust model for prediction that makes no assumptions on the underlying stochastic characteristics of the available data. My motivation in this work is thus to explore the feasibility of forecasting merchant health, particularly through machine learning methods. I also want to evaluate them rigorously in regards to accuracy and scalability.

1.3 Problem Statement

In order to focus my research efforts, I present below the following research questions, which will be answered by the end of this work.

1. Can machine learning methods provide more accurate predictions than statistical or naive methods?
2. Which machine learning methods can I profitably apply to my time-series data?
3. Which method yields the best trade-off between accurate prediction and feasibility of implementation?

The first question contemplates the possibility of transaction volumes following what amounts to random walks, a basic question for any attempt at forecasting [59]. The random-walk hypothesis posits that stock prices follow a random-walk process and thus cannot be predicted exclusively from its price history [23] — I must evaluate whether this is the case for my chosen dataset, see chapter 3.

Question two attempts to narrow the universe of possible predictors by discarding the ones that are evidently ineffective. I thus clear space to more deeply evaluate and fine-tune the methods which clear this initial hurdle. Chapter 5 discards multiple approaches that are deemed insufficient.

The answer to the final question points to the main contribution of this work. The answer will take into account benchmarking with established linear methods and other implemented predictors, as well as a comparison of scalability and complexity. Chapter 6 recaps this final result.

1.4 Thesis Contribution

This thesis aims to create a scientifically rigorous and reproducible process for the evaluation of a breadth of time-series prediction techniques. No assumptions are made about the suitability of the data for prediction or the ability of one algorithm to model the data correctly — reinforcing the validity of the outcomes. This research will enable future researchers to formulate more sophisticated experiments and models for use in the financial sector. Future work can use my wide-ranging approach as a heuristic for the a priori selection of forecasting methods: most poorly performing methods can be safely discarded. I also identify methods which may be promising but require a significant investment to arrive at a useful configuration — the case for most neural network-based approaches.

The results presented herein provide the basis for a predictor system for the transactions of small to mid-sized merchants, from which businesses can derive useful insights. I suggest tangible next steps for the implementation of such a system and identify ways to arrive at actionable suggestions from simple point forecasts. Engineers tasked with carrying out this work can confidently use this research as a theoretical basis and a roadmap.

1.5 Outline

Chapter 2 contains the literature research, which gives a broad overview of techniques for time-series prediction, including all techniques implemented in this work. Chapter 3 explores the available dataset, the units that compose it and its seasonality. The details of the implementation of the different predictors, including elements such as cross-validation, data preprocessing and benchmarks are included in chapter 4. The experimental results are presented and discussed in chapter 5. Finally, I summarize my results and explore possible next steps for research in chapter 6.

2 State of the Art

As stated briefly in chapter 1, there are two main approaches to time-series prediction and statistical modeling in general. As discussed by Professor Leo Breiman in his widely-discussed paper, “Statistical Modeling: The Two Cultures" [14], two main schools of thought exist: the “data modeling” culture and the “algorithmic modeling” culture. This distinction is borne out by my research into the topic, with prediction methods fitting neatly into one or the other, or as a combination of both.

The first approach, corresponding to statistical or stochastic prediction methods as described and summarized in section 2.1, assumes the available data follows a given stochastic distribution. Prediction then consists of finding the model which best fits this unknown distribution and drawing future observations from it [12]. This view has traditionally predominated in the forecasting domain [10] [14]. Due to their nature, methods that employ this strategy often have a much lower computational complexity than their counterparts [54].

The second approach, summarized in section 2.2, treats the data as a black box — it does not even attempt to model the stochastic distribution explicitly. “Algorithmic modeling” is rather concerned with using predictive accuracy as a heuristic to adapt the chosen algorithm to the data [14]. Breiman initially considered this approach to be marginal: he estimated about 2% of statisticians identified with this school of thought. The revival of interest in neural networks and machine learning in general seems to have changed that: hundreds of papers suggesting new ML approaches are available [38][85][88].

Professor Spyros Makridakis, well known because of his work in the forecasting domain and organizing the so-called M-Competitions [40] [55], considers that the results published in many of these papers have very limited applicability due to a lack of rigor [54]. He particularly considers the use of few time-series, short forecasting horizons and a lack of benchmarks for comparison. In this work, I have attempted to address those concerns. For more details on the implementation of benchmarks and comparable measurements, refer to chapter 4. For the sake of uniformity, the literature associated with the utilized benchmarks is explored below.

In the following chapter, I attempt to summarize the field of time-series forecasting, with an emphasis on machine learning-based methods (see section 2.2). Statistical prediction methods are also summarized and described, focusing mostly on well established and documented methods rather than bleeding edge research (section 2.1).

Not all methods presented here were chosen for implementation and evaluation:

some were deemed squarely out of scope. Those that were chosen are summarized in table 2.1 at the end of the chapter. For more details on decisions regarding implementation of the machine learning predictors or benchmarks, please refer to chapter 4.

2.1 Statistical Prediction Methods

Statistical prediction methods are the de facto standard in much of the financial literature. First I introduce the immensely influential work by Box and Jenkins in subsection 2.1.1 and then describe the smoothing method family in subsection 2.1.2.

2.1.1 Box-Jenkins approach

A great part of statistical prediction is based on the Box-Jenkins approach, described by George Box and Gwilym Jenkins and previously mentioned in chapter 1 [12]. The method roughly consists of trying to identify auto-regressive integrated moving average (ARIMA) processes which best fit the data available — placing the method squarely within Breiman's "data modeling" culture [45]. ARIMA processes encompass a large family of methods, including moving average (MA), auto-regressive (AR) and their combination, aptly abbreviated as ARMA. ARMA models can only describe stationary processes, whose mean and variance do not change over time.

AR processes of order p (written as $AR(p)$) are linear processes that follow the following structure:

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + \phi_2 \tilde{z}_{t-2} + ... + \phi_p \tilde{z}_{t-p} + a_t \tag{2.1}$$

where ϕ_i are the weight parameters for each stochastic process $\tilde{z}_i$ and a_t is an independent linear variable often modeled as Gaussian white noise [12]. In this notation, $\tilde{z} = z_t - \mu$ represents the deviation of the process from its mean. In short, an $AR(p)$ process can be described by a linear combination of its past p values.

Similarly, a **MA** process of order q may be written as

$$\tilde{z}_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q} \tag{2.2}$$

where θ_i are weight parameters corresponding to lagged values of a Gaussian white noise variable a_{t-i} [12].

Both of the models described above can be used to fit the data and extract future values. In order to increase their descriptive capability, consider the combination of both an **ARMA** process of order p, q or $ARMA(p, q)$, which includes both terms:

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + ... + \phi_p \tilde{z}_{t-p} + a_t - \theta_1 a_{t-1} - ... - \theta_q a_{t-q} \tag{2.3}$$

In order to simplify the above equations, a backshift operator B and a backward difference operator ∇ are defined:

$$Bz_t = z_{t-1} \implies B^m z_t = z_{t-m} \tag{2.4}$$

$$\nabla z_t = z_t - z_{t-1} \implies \nabla = (1 - B) \tag{2.5}$$

With these, an ARMA process can be succinctly described by

$$\phi(B)\tilde{z}_t = \theta(B)a_t \tag{2.6}$$

where $\phi(B)$ and $\theta(B)$ are polynomial operators in B of degrees p and q such that e.g. $\phi(B) = 1 - \phi_1 B - ... - \phi_p B^q$.

ARIMA processes extend the descriptive capabilities of ARMA into nonstationary processes and can be described with a further parameter d indicating the dth difference of a stationary series [12]. An $ARIMA(p, d, q)$ process can be written as

$$\phi(B)\nabla^d z_t = \theta(B)a_t \tag{2.7}$$

Initially, exploring the solution space of an ARIMA model required vast amounts of specialist knowledge to correctly assess stationarity, evaluating correlations and partial autocorrelations, etc. Nowadays much of this process can be effectively automated using different information criteria as an effective heuristic [45]. Of these, Akaike's Information Criterion [4] is the most common.

2.1.2 Smoothing Methods

The name for "smoothing" methods comes from their perceived property to separate the underlying time series from any noise surrounding it — effectively making the predicted values appear smoother when plotted [59]. Much like MA methods, smoothing methods take the previously observed values into account when forecasting the immediate value that follows them. Unlike MA methods, however, it places value on more recent ones by using an exponentially growing discount factor $\alpha < 1$:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t \tag{2.8}$$

with $\hat{y}_j$ as the forecast at $t = j$ [71]. The smoothed prediction is then a weighted sum of the last prediction corrected with the actual observed value. This smoothing method is called **simple exponential smoothing** and it was first described by Brown in 1957 [15]. Its exponential nature is evident when the last forecast is substituted according to equation 2.8:

$$\hat{y}_{t+1} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + (1-\alpha)^2\hat{y}_{t-1} \tag{2.9}$$

Building on Brown's work, other smoothing methods gained popularity, such as **Holt's linear exponential smoothing**, proposed by Holt in 1957 [43]. This method uses two smoothing constants (α and β) and expresses their relationship three of linear equations:

$$\hat{y}_{t+h|t} = l_t + b_t h \tag{2.10a}$$

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.10b}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \tag{2.10c}$$

with $\hat{y}_{t+h|t}$ being the prediction at time $t+h$ given the data points up to time t [71]. An extension of this method, often called the **Holt-Winters' method** was proposed to take into account seasonal patterns in the data [86].

A third approach, the **damped trend method** [32], seeks to dampen the effect of trends further by introducing a third factor ϕ to Holt's linear equations [32]:

$$\hat{y}_{t+h|t} = l_t + b_t(\phi + \phi^2 + ... + \phi^h) \tag{2.11a}$$

$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + \phi b_{t-1}) \tag{2.11b}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)\phi b_{t-1} \tag{2.11c}$$

Note how Holt's method is identical to the dampened trend method for $\phi = 1$.

The approach used by both exponential smoothing and ARIMA methods (cf. subsection 2.1.1) is very similar. Indeed, both have overlapping definitions, with simple exponential smoothing being an ARIMA model with a specific autoregressive operator [1]: the first-order exponential smoother as described in equation 2.8 is equivalent to an $ARIMA(0,1,1)$ process. Consequently, smoothing methods can be deployed to handle "data exhibiting a range of features, including very little trend, strong trend, no seasonality, a seasonal pattern that stays constant, and a seasonal pattern with increasing variation as the level of the series increases" [71] with little to no human input.

For simple, Holt and damped exponential smoothing, I use the `ets` method in the `forecast` package for R [48]. For more information on benchmarks, see section 4.1.

I would also like to highlight an aditional statistical forecasting approach proposed in order to compete in the M3 competition [40], the **Theta model** [5] — this method is relevant because of its use as a benchmark in this work (see chapter 4). The Theta-model works by decomposing seasonally-adjusted series into two components: short and long-term. After this artificial decomposition, two separate predictions are made using linear regression and exponential smoothing respectively. The two forecasts are then combined and re-seasonalized for the final prediction.

The field of statistical prediction methods is vast and storied, particularly when it comes to econometrics. More complex versions of the algorithms presented above intend to account for specific use cases or data characteristics. Such is the case for autoregressive conditional heteroskedasticity, ARCH, its generalized version GARCH [9] and its numerous variations, which are employed in more volatile time-series [59]. As stated earlier in this chapter, this thesis does not focus on the bleeding edge of statistical prediction research, but rather on helping machine learning methods catch up with the flexibility, robustness and predictive capability of the former. Readers interested in more material on statistical prediction methods are advised to refer to Rob J. Hyndman's work, which is particularly accessible [46].

2.2 Machine Learning-based Prediction Methods

This section gives an overview of most machine learning prediction methods, including a short discussion of where I draw the line between machine learning and other more "traditional" methods in subsection 2.2.1. The approaches themselves are separated into supervised learning (2.2.2) and neural networks (2.2.3).

2.2.1 What constitutes machine learning?

Scikit-learn, one of the most popular and widespread machine learning libraries for Python [66] contains implementations for more than fifteen generalized linear model predictors. These all work under the same assumption — the target function to be regressed to follows the following structure:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p \tag{2.12}$$

where $\hat{y}$ is a function of x_i input variables and w_i are coefficients. The regression task then consists in finding the best coefficients to fit the available data. This makes it conceptually identical to e.g. finding the weight parameters ϕ_i for a general AR process as described in equation 2.1. Where then should one draw the line between a statistical method and one that's machine learning-based?

For neural network-based approaches (c.f. subsection 2.2.3), this distinction is fairly straightforward: the models consist of layers of distinct units whose weights have been "trained" according to the input data — the weights themselves are not mapped to a particular mathematical model of the data and their interpretability is extremely limited [35]. They are set apart by their architecture alone.

For traditional supervised learning approaches and particularly the linear case described in equation 2.12, I make the distinction based on the process used to arrive at the coefficients. In a machine learning workflow for time series prediction, the error ϵ, i.e. the residual error between my predictions and the true response is used iteratively to steer the weights towards minimizing ϵ [60]. Weights are often randomly initialized and there are few restrictions to the values that the weights can take. Compare this to the rigid structure present in the Hyndman-Khandakar algorithm implemented by R's `auto.arima` [48]. Here, repeated unit root tests and limited model fittings are used to find the lowest AIC present in the limited space considered.

This definition explains the inclusion of methods as "traditional" as ordinary least squares into the machine learning category; not by virtue of their conception or history but rather the process by which they are implemented.

2.2.2 Traditional Supervised Learning Approaches

Formulating time-series prediction as a supervised learning problem is straightforward: "input-output pairs", or a "training set" are presented to a model which learns the function mapping input to output. To gauge the accuracy achieved, I then present it with a "validation set" composed exclusively of inputs and compare it with their corresponding outputs, which were not previously fed to the model [75]. All of the models presented below operate under this principle. For the predictors, I use the implementations found in the python package `scikit-learn` [66]. Approaches based on neural networks are covered in the following subsection (cf. 2.2.3).

Ordinary least squares fits a linear model that can be described by

$$y_t = w_0 + w_1 x_{1,t} + ... + w_k x_{k,t} + \epsilon_t \tag{2.13}$$

where x_i are the various predictor variables or features, y_t the forecast variable, ϵ_t the error term and w_i the weight coefficients [59] [47]. The model is fit by minimizing the residual sum of squares between the predicted and observed predictor variable instances [66]. The regression can be described as a minimization problem:

$$\min_w ||\mathbf{X}w - y||_2^2 \tag{2.14}$$

with **X** as the matrix of predictor variables.

Ridge regression, also called regularized least squares, weight decay or also Tikhonov regularization, attempts to give preference to smaller coefficients by adding a regularization term [70]:

$$\min_w ||\mathbf{X}w - y||_2^2 + \lambda ||w||_2^2 \tag{2.15}$$

with λ as a regularization factor [66].

Regularization is often used to avoid overly complex functions that overfit to the training data [60] — the effect of a higher regularization factor are illustrated in figure

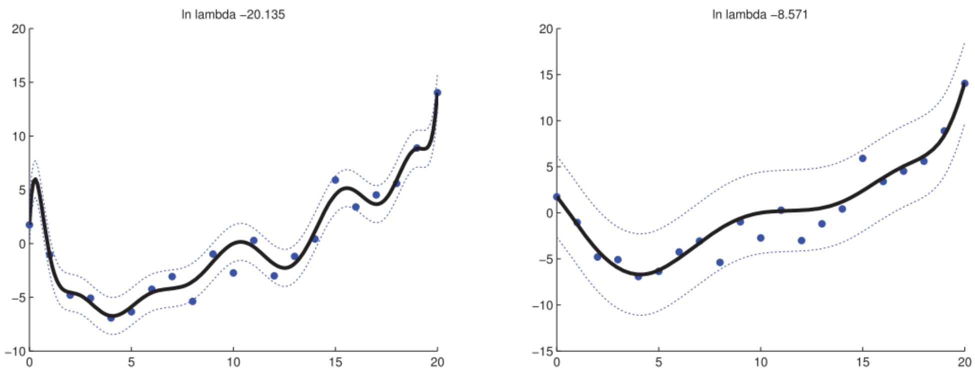


Figure 2.1: Illustration of regularization's effect on regression: an increasing regularization factor results in smoother curves [60]

2.1. **Lasso** also utilizes regularization and selects more aggressively for models with coefficients equal to zero [83]. Because of its preference for sparsity, it is often used in domains with very large datasets, where simpler models yield more scalable results. The **elastic net** method generalizes both of the previous approaches into a single model with a joint penalty term for weight regularization [30].

Least Angle Regression or LARS intends to expand on the performance gains present in elastic net. The algorithm begins by fitting a linear model with only one nonzero coefficient — the one most correlated with the forecast variable. It then iteratively adds the remaining coefficient with the highest correlation until all coefficients are taken into account or the model's predictive capability does not improve [60]. **LassoLARS** applies the same approach to a Lasso regularization method (see above) [25].

Orthogonal matching pursuit (OMP) is a variation of a previously postulated "matching pursuit algorithm". It employs a similar approach to LARS and its derivates: a greedy algorithm selects for the solution with the highest correlation of a limited number of coefficients and increasingly scales up the solution space [63]. It purports to converge with fewer iterations and computation cost than similar methods.

Bayesian regression, part of the wider field of Bayesian learning, opts for modeling uncertainty as a random variable. The approach starts from a preliminary model in which the output is Gaussian-distributed [61]:

$$p(y|\mathbf{X}, w, \alpha) = \mathcal{N}(y|\mathbf{X}w, \alpha) \tag{2.16}$$

This distribution is then updated with every new value observed. Note how the regularization factor α is also modeled as a further random variable — the fitting of the model also includes “fitting” the most effective regularization [8]. A Bayesian approach can also be used for model selection, classification and other applications. Interested readers should refer to Prof. Radford Neal’s work on the subject for an accessible introduction [61].

Under the same framework, **Gaussian Processes** marries a Bayesian approach with kernel functions (see below). One can consider Gaussian processes as a series of jointly Gaussian-distributed random variables and make the assumption that the function that is being regressed to can be described by such a process. Since one is dealing with a distribution over functions (e.g. the mean and covariance functions), optimization happens over the function-space instead of a traditional vector space. Kernel functions can be passed to the Gaussian process to act as the initial covariance for the prior distribution [69].

Stochastic Gradient Descent (SGD) is a simplification of traditional gradient descent, which is particularly well known for its uses in neural networks. Gradient descent uses the gradient of a vector field to iteratively find local minima. SGD creates an approximation of the gradient by drawing a “single randomly picked example” instead of the entirety of the data and progressively adjusts the weights and learning rate η according to the result [11].

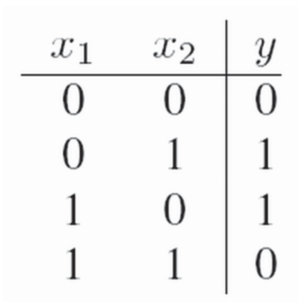
Perceptrons are very simple linear models that form the basis for early neural networks — they played an important role during the infancy of machine learning [72] [58]. They can still be used as efficient and simple linear models. The mathematical formulation is simple:

$$y(\mathbf{X}) = f\big(\sum_i (w_i x_i)\big) \tag{2.17a}$$

$$f(b) = \begin{cases} +1 & b \geq 0, \\ -1 & b < 0 \end{cases} \tag{2.17b}$$

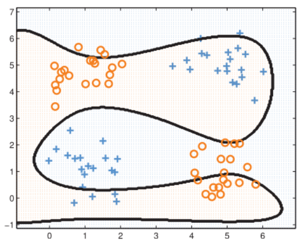
where f is the activation function. Weights are updated with every iteration depending on the forecast error [60]. They are often used in complex networks consisting of multiple layers of interconnected Perceptrons or “neurons”. Such “neural networks”, including a simple network of multiple Perceptrons is described further ahead (subsection 2.2.3).

A common problem in machine learning is the representation of the studied phenomena in feature vectors that are fed to the learning algorithm. The exercise becomes increasingly problematic as the complexity of the input data grows. A very popular approach sidesteps the issue altogether by using so-called kernel functions. These functions measure the distance between data points without the need for preprocessing into feature vectors. Many existing algorithms can be “kernelized” by employing such a kernel function in lieu of the inner product [60]. The algorithm can then find a solution in the higher dimensional space into which the kernel function has mapped the input. The so-called “kernel trick” can provide immediate solutions to problems that are not tractable for a traditional classifier, like the “XOR problem”, a function that is not linearly separable (see fig. 2.2). As an exemplary linear regressor using the kernel trick, I take a **kernelized ridge regressor**, c.f. equation 2.15.

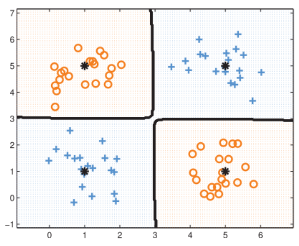


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

(a) XOR function



(b) Linear logistic classifier



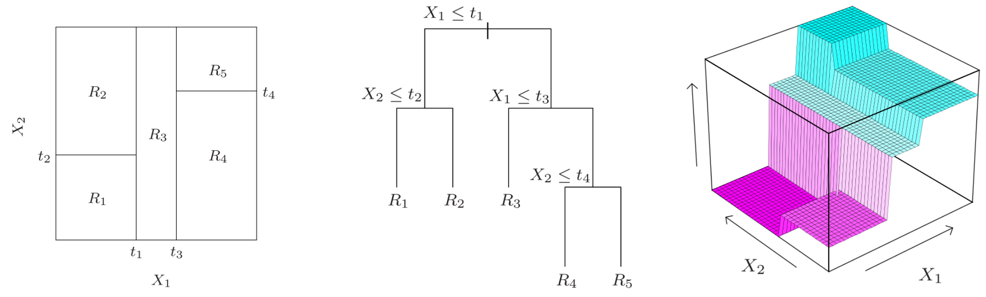
(c) Kernelized classifier

Figure 2.2: The kernel trick can be used to easily solve the classical XOR problem [60]

Building on kernel functions, the **Support Vector Machine** (SVM) has been successfully applied to a variety of tasks, including classification and regression in financial time-series [87]. An SVM combines a the kernel trick with a specific loss function to create a sparse solution which depends only on a subset of the training data, the “support vectors”. SVMs are a well-studied approach and have much capacity for optimization, e.g. tuning regularization parameters, nudging the machine into convergence through monitoring, etc. A wide-ranging introduction into the wider field of SVMs can be found in the work by Smola et al. [77].

Nearest Neighbor approaches are particularly intuitive: given an input, the closest-matching training samples are found and a label (discrete or continuous) is predicted. There is a wealth of distance metrics and criteria by which to assign the label. This approach has evident scalability issues — a comparison with the entire dataset is computationally intensive, but they can be assuaged through e.g. tree-like data structures [34]. The default implementation uses a k-Neighbors approach, in which the nearest k observations are the only ones taken into consideration for regression.

Decision Trees are a particular popular approach due to their interpretability, resistance to irrelevant predictor variables and flexibility in the face of continuous or discrete data; they mostly suffer of comparative inaccuracy [39]. To create a tree, the data is paritioned with a series of recursive splits in the feature space. This process continues until a specific depth is reached, defined a priori (explicitly or implicitly) by using a regression cost function. The results of such a split can be seen in figure 2.3.



(a) Partition of a two-dimensional feature space **(b)** Interpretation of partition as a binary tree **(c)** Prediction surfaces of the resulting partition

Figure 2.3: Trivial example of a decision tree [39]

Boosting was originally postulated for classification problems but can be easily expanded to include regression. Boosting is a simple additive model: many "weak" predictors, only slightly superior to random guessing, are combined to produce a much more accurate ensemble model [39]. This often means quickly training each of the weak predictors with a slightly modified version of the input data and training weights for each of the regressors. In my work, I use one of the most popular boosting algorithms, **AdaBoost** [76]. I encourage interested readers to read Schapire and Freund's original text, which manages to explain their algorithm in very succinct and intuitive terms.

A boosting approach can be easily combined with decision trees as the weak predictors. A handful of years after the formulation of boosting algorithms, researchers proved the equivalency of boosting algorithms with the minimization of an "edge function" which relies on an exponential loss function [13]. With this insight, the principle of gradient descent can be applied to boosting algorithms directly — this resulted in gradient boosting. From this family of approaches I take one of the most common: **Gradient Tree Boosting** (GTB) [31]. For GTB, decision trees of a specific depth are used as weak regressors whose predictions are as close to possible to the negative gradient of a previously-defined loss function [39].

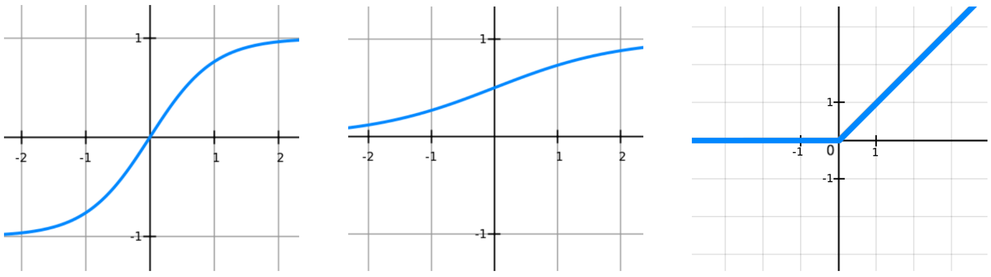
It should be by now evident that there is an almost inexhaustible wealth of possibilities for the combination and ensemble of different predictive approaches. Instead of trying to absolutely exhaust all such possible avenues, I have focused on the most accessible and well-established approaches, as available in one of the most popular machine learning libraries.

2.2.3 Neural Network-based Approaches

Neural Networks are deceivingly straightforward computational models: they are simply networks of artificial neurons whose function was inspired in real biological neurons. The simplest form of these neurons is the Perceptron, described in more detail in subsection 2.2.2. Neural networks and their derivatives have undergone various cycles of hype and disillusionment, resulting in two of what are often called "AI winters" [75]. Neural networks have been experiencing a vigorous revival in both capability and public interest, mostly due to advances in hardware (e.g. GPU acceleration) and an explosion in data availability and volume. They have been particularly adept at solving perceptual problems, including many image processing and recognition applications [20].

From a practical point of view, neural networks are simple nonlinear predictors whose behavior is determined by a number of moving parts: "layers" of neurons that combine to a model, loss functions to use as feedback for learning, and optimizers, which govern how learning progresses [64]. "Fitting" a neural network model happens by finding the optimal values for the unknown parameters present in the model (cf. equation 2.17). After calculating the loss between the predicted value and the reference output with the loss function, the network is submitted to an algorithm such as back-propagation to adjust these weights. Back propagation traditionally uses gradient descent and the chain rule to adjust the weights along the entire model, starting from the "end" of the model — effectively "propagating" the gradient back to the input neurons [39]. I decided to leave out a detailed description of loss functions or optimizers from this section and instead focus on the neurons and the networks they can create.

There are hundreds of papers dealing with neural network-based approaches to time-series prediction. Evaluating them all, or indeed a representative amount, would not only be impossible due to time constraints, but also inadvisable. As eloquently stated by Makridakis et al. [54], a large amount of these papers lack the necessary scientific rigor to be taken as a serious approach. In order to fairly evaluate neural network approaches, I decided to focus on the major architectures available and not in individual implementations custom-made for one or two time-series.



(a) Hyperbolic tangent (tanh) activation function

(b) Sigmoid activation function

(c) Rectified Linear (ReLU) activation function

Figure 2.4: Examples of different activation functions in artificial neurons [64]

I have also tried to answer some of the concerns raised — for more, please refer to chapter 4.

For the testing of the different neural network approaches, Sci-kit learn is very limited in both breadth and liberty of design. In order to have access to a more granular and powerful tool, I implemented my neural networks with the python library Keras [21]. Keras is capable of running on top of most major lower-level neural network libraries and offers GPU support, vital when handling the much higher number of computations necessary for this family of predictors.

As mentioned before, traditional neurons still represent a powerful tool. They have evolved since the days of the Perceptron however; artificial neurons now use a variety of activation functions and not exclusively a step function as detailed in equation 2.17. Various sample activation functions are illustrated in figure 2.4. The most basic network model configuration is the **feed-forward neural network**, also called the **multilayer Perceptron**. These networks are characterized by fully interconnected layers and a forward flow of information: there is no feedback connection to reintroduce or retain the output of the network (backpropagation gradients excluded) [35]. An exemplary feed-forward network is shown in figure 2.5.

Feed-forward networks can be augmented in a variety of ways, particularly to reduce the noise of the data or to more clearly extract predictive features. Such is the case for **Restricted Boltzmann Machines** (RBMs) [78]. RBMs are often used as building blocks for larger networks or as separate networks altogether. They classically have the structure of a simple 2-layer neural network: one input layer and one hidden layer — albeit with undirected connections instead of directed ones as is the case for feed-forward networks (see fig. 2.6). Consequently, neurons often use “contrastive divergence” as an update rule [41]. With training, the hidden nodes learn to reflect abstract structures present in the data. There are many ways to leverage the learned patterns; one of the most popular ones is to take the learned weights from the individual units in the hidden layer and transfer them to another

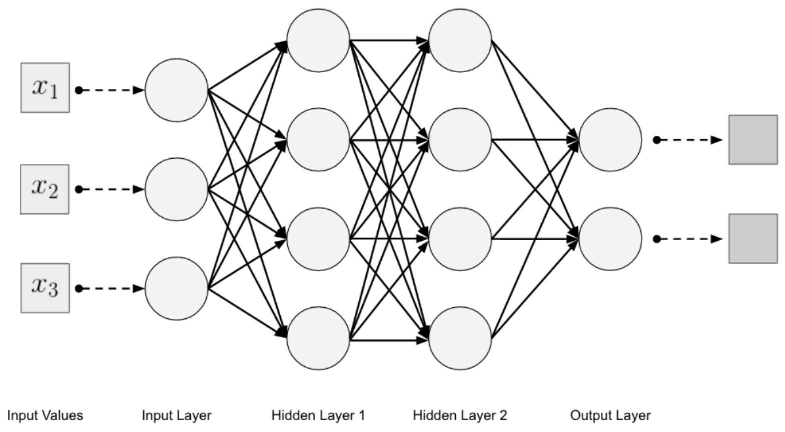


Figure 2.5: Exemplary structure of a feed-forward neural network [64]

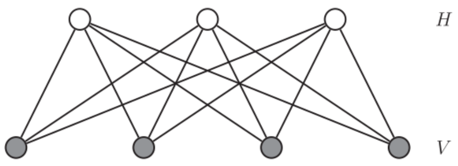


Figure 2.6: Exemplary structure of a Restricted Boltzmann Machine with a hidden layer H and a visible layer V. [60]

network, e.g. a feed-forward network. The resulting “pretrained” structure can then be used as a normal predictor [50].

Another type of feed-forward network component is the **autoencoder**, a network often employed to filter noisy data. It attempts to do so by reproducing the input after passing through a neuron bottleneck, i.e. a hidden layer with fewer units (see fig. 2.7). The bottleneck layer necessarily has a reduced representation capability compared to the output and input layers, “compressing” the data — forcing the network to prioritize which aspects remain [35]. Such units are then concatenated with a standard feed-forward layer and processed as usual [52] to generate a prediction. I focus only on approaches that concatenate the first part of the autoencoder instead of embedding it in the middle of a larger structure.

Despite the predictive power that traditional feed-forward neural networks have proven over many applications, they are not designed for the processing of sequential data, i.e. a series of values which can be interpreted as a single value that changes over time. **Recurrent neural networks** (RNNs), on the other hand, are explicitly created for this purpose [74]. RNNs are also feed-forward networks, but

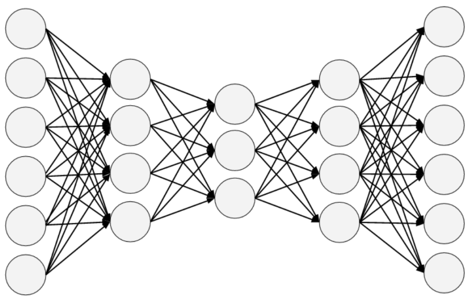


Figure 2.7: Exemplary structure of an autoencoder network [64]

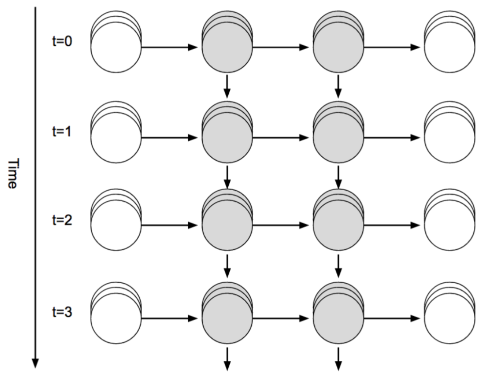


Figure 2.8: Flow of information across the network and time in a recurrent neural network [64]

they differ in the connections between neurons — feedback loops are incorporated in order to model the temporal dimension. The simplest of these connections is to feed the output of each neuron back to itself as an input. The information thus flows not only "forward" towards the output, but also along the time axis [35]. For a visualization of this principle, refer to figure 2.8. Note that the figure shows a single "flattened" feed-forward network, with two hidden layers.

It is also possible for RNNs to have more complex temporal feedback loops to try to better model the recurrent nature of the data. The most widespread unit used today is the **Long Short-Term Memory** (LSTM), a complex neuron which incorporates different "gate units" to regulate the data that gets aggregated into the neuron's activation function [42]. "Forget", input and output gates all have their own activation functions and associated weights. A detailed breakdown of the weight computation inside the unit is not relevant at this point, but it is important to note how the number of values that must be calculated during training is much higher than in a normal network.

Gated Recurrent Units (GRUs) attempt to address this explosion in computational complexity while still maintaining a mechanism for temporal relationships to be modeled [19]. GRUs have a single gating unit that controls both “remembering” and “forgetting”, making networks of GRUs much faster to train [35].

2.3 Ensemble methods

As in the Theta model (cf. subsection 2.1.2), any model can be arbitrarily combined with any number of other approaches, typically in a linear combination [59]. The weights associated with each approach could be optimized in hopes of finding the minimum error variance, as suggested by Bates and Granger [6]. They may also be combined naively, by just calculating the average between all present predictors. That is the case for the **combined smoothing method** proposed as a benchmark by Makridakis in his M4-Competition [53]. The method, identified by the shorthand “S-H-D” is composed of the arithmetic mean of simple, Holt’s and Dampened exponential smoothing (see subsection 2.1.2).

These ensemble methods are not bound by distinctions between statistical or machine learning methods. Indeed, there is much promise in combining the two: neural networks can be employed to find the right coefficients for e.g. smoothing algorithms — combining the insights generated by a costly neural network with the robustness of statistical methods. This is the case for Slawek Smyl’s work, overall winner of the M4 competition [55]. His approach combines a bespoke RNN made up of LSTM blocks that optimize smoothing and seasonality parameters. The end result is a combination of its component predictors [79]. I deemed the reimplementation of such a complex system squarely out of scope for the present work.

List of evaluated prediction methods		
Method Name	Reference	Benchmark
Statistical Methods		
ARIMA	[12]	☒
Simple Exponential Smoothing	[15]	☒
Holt's Exponential Smoothing	[43]	☒
Dampened Exponential Smoothing	[32]	☒
Combined Smoothing (S+H+D)	[54]	☒
Theta method	[5]	☒
Machine Learning Methods — Supervised Learning		
Ordinary Least Squares	[59]	☐
Regularized Least Squares ('Ridge')	[70]	☐
Lasso	[83]	☐
Elastic Net	[89]	☐
Least Angle Regression (LARS)	[25]	☐
LARS Lasso	[25]	☐
Orthogonal Matching Pursuit (OMP)	[73]	☐
Bayesian Regression	[61]	☐
Support Vector Machine (SVM)	[77]	☐
Linear SVM using Stochastic Gradient Descent (SGD)	[11]	☐
Gaussian Process Regressor	[29]	☐
Kernelized Ridge Regressor	[60]	☐
K-Nearest Neighbors (KNN)	[34]	☐
Decision Trees	[11]	☐
Adaboost	[76]	☐
Gradient Tree Boosting (GTB)	[39]	☐
Machine Learning Methods — Neural Networks		
Multi-layer Perceptron (MLP)	[58]	☒
Autoencoder + MLP	[52]	☐
Recurrent Neural Network (RNN)	[74]	☒
Long Short-Term Memory Network (LSTM)	[42]	☐
Gated Recurrent Unit Network (GRU)	[22]	☐

Table 2.1: List of evaluated prediction methods

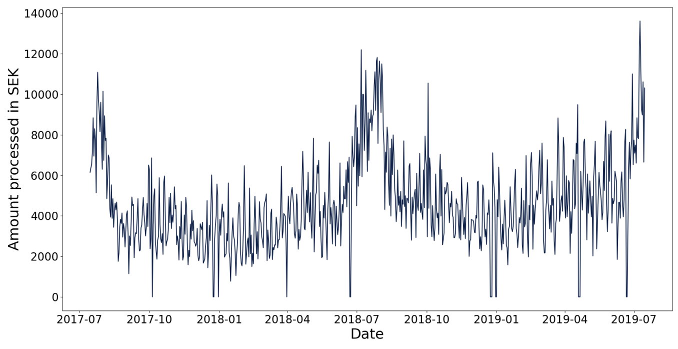
3 Description of Available Data

3.1 Description

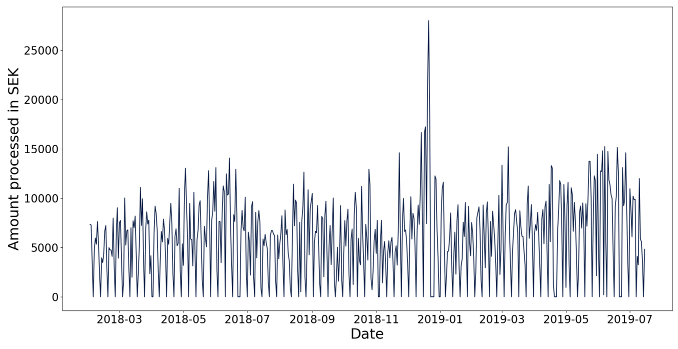
The dataset for this thesis was provided by my partner company, a medium-sized technology provider in the card payment industry. The company handles their clients' transaction details for payments in markets across Europe, Africa and North America. In order to work with a manageable amount of data, I selected the transaction history of a cashier system operating primarily in the Swedish market. This cashier system services small to medium-sized shopper-facing businesses or “merchants” (i.e. restaurants,furniture stores, coffee shops, etc.). The client in question was chosen primarily due to location: around 90% of Swedes polled use cards for everyday purchases [82] — meaning the captured information for any given merchant represents a large percentage of the vendor's total transactions. The data available to me spans a maximum of two years for regulatory and industry compliance purposes: date ranges lie between 2017 and 2019 for most businesses. The granularity present is of single transactions: each sale at a shop serviced by the client will result in a data point if the shopper pays with a credit or debit card.

As mentioned in chapter 1, I consider each individual merchant's data separately for the sake of interpretability and actionable insights. The available data included information for 659 individual businesses. As the cashier system constantly incorporates new vendors into its service, many of these merchants only had transaction history available for a few months. In order to work with manageable amounts of information while still having enough to be able to generalize the results of my work, I extracted the twenty merchants with the highest numbers of individual transactions. For comparability purposes,I resampled the data available into single-day bins. The average number of days available for analysis for the top twenty used was 605 days out of a possible 732. Seven of the top merchants had the maximum amount possible and the minimum in the top 20 was 432 days. A small visual sampling of these 20 vendors are presented in figure 3.1. All values displayed represent Swedish crowns (SEK). For identity protection purposes, the names of the individual businesses have been removed from the present work.

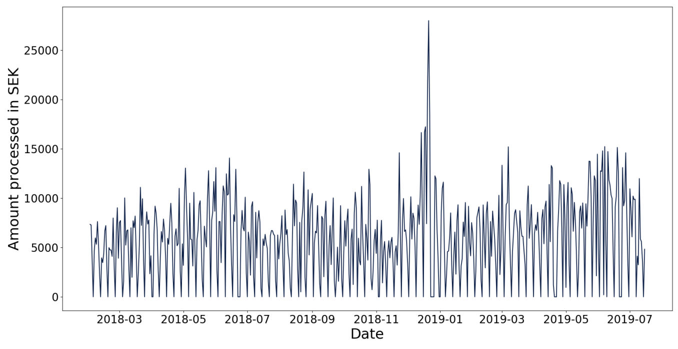
The nature of the data can be intuitively interpreted when visualized: all merchants have different business cycles, average transaction volumes and general behavior. The merchant in subfigure 3.1a for instance, has very recognizable peaks around late summer in all years observed — one can imagine a particularly popular café with a terrace. The business visualized in subfigure 3.1b probably took a couple



(a) Merchant with yearly (summer) cycles



(b) Merchant showing a small inactivity period, probably corresponding to summer vacations



(c) Merchant showing a periodic inactivity — the business closes on Sundays

Figure 3.1: Examples of different merchants showing patterns proper of shopper-facing businesses

of weeks during July of 2018 off for a summer vacation, effectively flatlining their processed transactions during this period. The merchant in subfigure 3.1c, on the other hand, exhibits behavior typical of a store closing on Sundays for a day of rest.

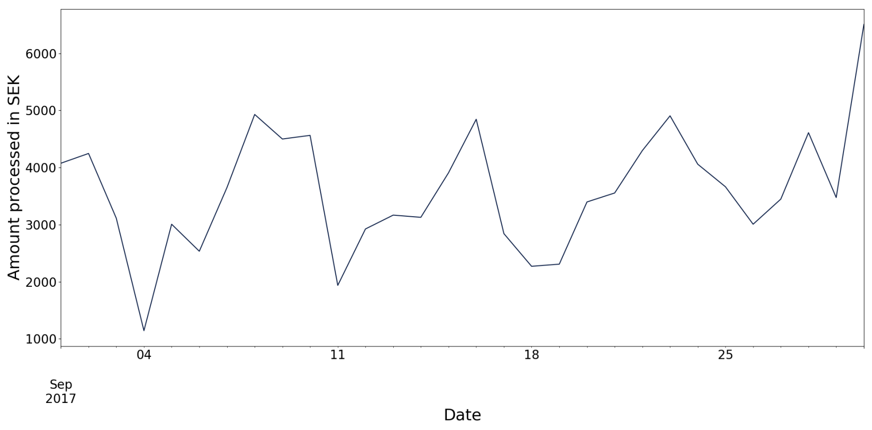
3.2 Seasonality

Seasonality is of particular interest in my dataset — cycles can be exploited for accurate forecasting [60][56]. I have already observed the yearly cycle present in subfigure 3.1a; note also the distinct peak during holiday season for subfigure 3.1b. These cycles reflect human consumption patterns and they exist not only in yearly intervals, but also in smaller time frames. To illustrate this, take a sample month from the same dataset, September 2017 (see figure 3.2).

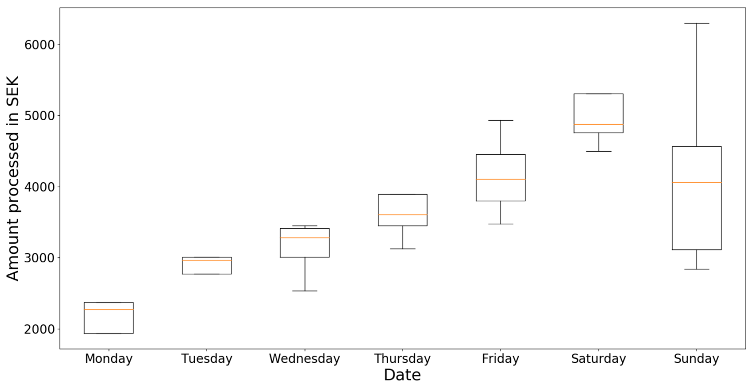
The weekly cycles are immediately evident when plotting the monthly course: there are distinct peaks and valleys every seven days. There is a particularly high consumption on the last day of the month, which could correlate with traditional paydays for salaried workers. Binning the different transaction amounts depending on the day of the week corroborates my observation about a peak and a valley (cf. subfigure 3.2b): people spend by far the least on Mondays, spending most of their money during the weekend. The highest peak in subfigure 3.2a corresponds not only with the end of the month: the 30th of September of 2017 fell on a Saturday.

These observations are not meant to be taken as generalizations for my entire dataset; as previously stated, the businesses themselves are too different from each other. The existence of such marked cycles does, however, inform my choices regarding feature creation, explored later on in section 4.3.

Finally, very marked seasonality can also be detrimental — one of the top twenty merchants was an exclusively seasonal business (see fig. 3.3). This business evidently presents cycles that repeat between activity bursts during the winter months. The logical information handling process in this case would be to remove the chunks of the data that hold no information: the months between May and October are effectively superfluous. This would, however, require individual preprocessing exclusively for this case. Since one of my guiding principles is finding a predictor that works "out-of-the-box" for as many merchants as possible, I elect to exclude this particular dataset. To maintain the same number of merchants for my model evaluation, the next possible merchant in terms of transaction numbers is taken.



(a) Transactions for the month of September 2017



(b) Breakdown of transaction volume by day of the week

Figure 3.2: Business cycles for a sample month

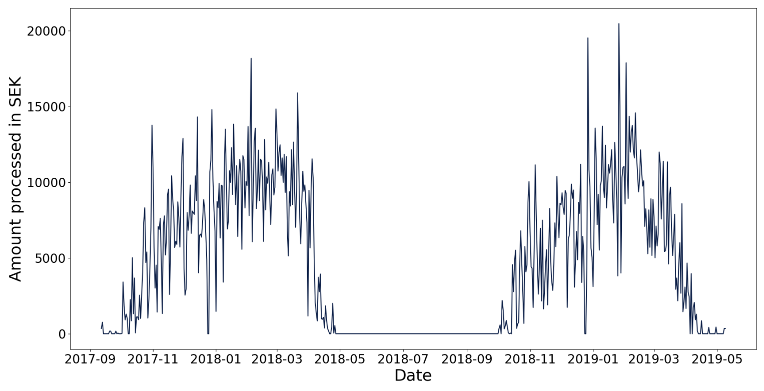


Figure 3.3: Merchant with business activity exclusively in the winter months

4 Implementation of Time-Series Predictors

As mentioned in previous chapters, many publications' efforts in the field of time-series prediction through machine learning suffer from limited applicability [54]. Three main reasons are highlighted: limited statistical significance caused by using only “a few or even a single time-series", short-term forecasting horizons, often one-step and no use of benchmarks to compare predictive accuracy. my research project attempts to address these three concerns in order to ensure that the results are widely applicable and can generate valuable insights for multiple agents along the value chain in the card payment industry.

In order to allay the first set of concerns, I test my models with twenty different time-series with more than 600 data points on average for each series, as detailed in chapter 3. All merchants have different patterns, cycles and magnitudes of transaction values. Additionally, I use exhaustive cross-validation along the entirety of the series as detailed ahead in section 4.2. Regarding limited forecasting horizons, forecasts are generated a full 28 steps ahead, corresponding to four weeks — significantly more than the 14-day predictions used in the M4 competition for daily values. Finally, I compare all of my predictors with a set of benchmarks and measurements from the field of time-series prediction using a mix of statistical and machine learning methods (cf. section 4.1).

When it comes to my predictors, I use an iterative process to select the best-performing technique — the corresponding sections are detailed below. I begin with a list of 21 predictors spanning linear, nonlinear and neural network machine learning methods; see section 4.4. These are evaluated with my gauntlet of twenty datasets with consistent feature creation and data scaling, detailed in section 4.3. After this initial evaluation, the resulting top five predictors in terms of accuracy then undergo hyperparameter tuning in order to achieve the best possible performance from each individual method as described in 4.5. After these these two evaluation stages, the absolute best predictor can be identified. The entire process is summarized visually in figure 4.1.

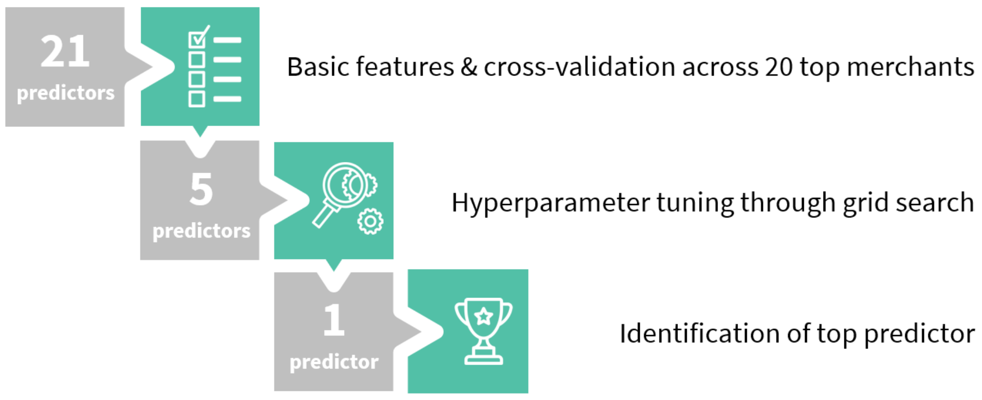

21 predictors
Basic features & cross-validation across 20 top merchants
5 predictors
Hyperparameter tuning through grid search
1 predictor
Identification of top predictor

Figure 4.1: Overview of my iterative process for the identification of the top predictor for my selected application

4.1 Benchmarks

According to my research, the only recent and applicable benchmarks for time-series prediction available in the literature are the competitions led by Professor Spyros Makridakis, also known as the M-Competitions [2]. The latest competition, dubbed “M4” by virtue of being the fourth such event, was held in 2018 [55]. In it, entrants ran their predictors through 100000 different time series from across different fields and using different forecasting horizons to determine which one could best handle such a wide range of data. The overall winner, as mentioned in section 2.3, was a hybrid machine learning and statistical method approach proposed by Slawek Smyl [79]. I am not interested in subjecting my predictors to the same gauntlet of tests: I am dealing with a much more specialized used case than in the competition. What I am interested in, however, is the benchmarks by which the results were measured.

Ten benchmarks were provided by the organizers. They included both naive, statistical and machine learning based approaches [54] (compare with table 2.1). Of the approaches shown listen in the table, only three have not been discussed in chapter 2: “Naive1”, “Naive2” and “sNaive”. Naive approaches are the simplest forecasting techniques. As defined by the M4 organizers, Naive 1 is simply the last value observed — this approach is the best possible predictor when dealing with true random walks. Seasonal naive is the last observed value in the last period, with the period being equal to the forecasting horizon. In my case this means the value of the series four weeks before. Finally, Naive 2 employs classical multiplicative decomposition as defined by the R `forecast` library to determine seasonality [48]. After that, the predicted value is the same value one period or “season” before [53].

Benchmarks one through eight are implemented in R using the `forecast` library. They are integrated into my Python predictors by way of the `rpy2` package [33].

The remaining two are implemented in Python: the multilayer perceptron uses scikit learn [66], while the RNN uses Keras [21], in a similar implementation to the ones I used. The organizers of the competition have made the repository containing these scripts available through Github.

4.1.1 Measurements

The M4 competition organizers propose three measurements for comparing the accuracy of the different predictors:

- Mean absolute scaled error (MASE)
- Mean absolute percentage error (SMAPE)
- Overall weighted average (OWA)

MASE, described at the time of its proposal as “the best available measure of forecast accuracy” by its authors, is based on scaled errors [49]:

$$MASE = \frac{1}{h}\frac{\sum_{t=1}^{h}|Y_t - \hat{Y}_t|}{\frac{1}{n-m}\sum_{t=m+1}^{n}|Y_t - Y_{t-m}|} \tag{4.1}$$

SMAPE was initially formulated as a response to most other accuracy measurements being "asymmetric", meaning they put “a heavier penalty on negative errors than on positive errors", a wholy arbitray distinction [40]. The measurement was first used in the M3 competition and was selected again for use in its successor, despite some critical voices [47].

$$SMAPE = \frac{1}{h}\sum_{t=1}^{h}\frac{2|Y_t - \hat{Y}_t|}{|Y_t| + |\hat{Y}_t|} \tag{4.2}$$

For both equations, Y_t is the value of the time series at point t, $\hat{Y}_t$ the estimated forecast, h the forecasting horizon and m the frequency of the data, which for my daily-sampled values equates to 1.

The OWA is computed taking into account both the SMAPE and MASE: both measurements are divided by the corresponding scores achieved by the Naive2 benchmark. The resulting relative SMAPE and relative MASE are then averaged to calculate the OWA. Note that while the SMAPE and MASE are computed for each fold and merchant dataset separately, the OWA calculation happens only once with the averaged values, per the stipulations of the M4 Competition.

4.2 Cross-Validation

One of the main areas of concern for a machine learning model is overfitting. Cross-validation (CV) is one of the most widespread techniques for model evaluation in regards to how well it generalizes over new data [80]. As eloquently explained in work by Hastie et al [39], it estimates the extra-sample error when the predictor is applied to a sample independent from the joint distribution of the training set features and variables. K-Fold CV, for instance, divides all available data into k “folds" or partitions of equal size, each of which can then be alternatively designated as the validation set (see figure 4.2), with the rest serving as the training set.

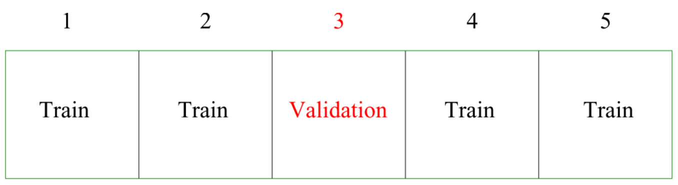


Figure 4.2: K-fold Cross-Validation for K = 5 [39]

One of the assumptions behind CV approaches such as K-fold is that observations in the data set are drawn from an independent and identically distributed (IID) process, i.e. the order of the samples is irrelevant [51]. This is not the case for time-series; quite to the contrary, one of the main phenomena I am modeling is the time-dependency of the data. While there is research to suggest that ordinary CV processes can still be applied to time-series data, it requires using models that are purely autoregressive [7].

It is also common practice to eliminate any “leakage" of information between the validation and the training set [51], so as not to achieve spuriously high scores when testing the model. In the case of time-series prediction, this means making sure that predictions about the future are only based on information gleaned from the past.

Both of these concepts coalesce into a special kind of cross-validation for time-series applications: starting from a defined training set size, consecutive training sets incorporate progressively larger time frames and are a subset of the previously generated sets. The validation set correspondingly moves along the time-axis but its size typically stays the same [47], see figure 4.3. The final measurements for any given predictor are the average across all cross-validation folds.

It is possible to fix the size of the training set instead of having it increase with time, especially since for most applications, newer data is more relevant for a prediction. Since I am not dealing with prohibitive amounts of data — the maximum of two years sums up to 730 data points — I let the window grow indefinitely. Restricting the size of the training window may become more relevant when implementing the system in a production setting.

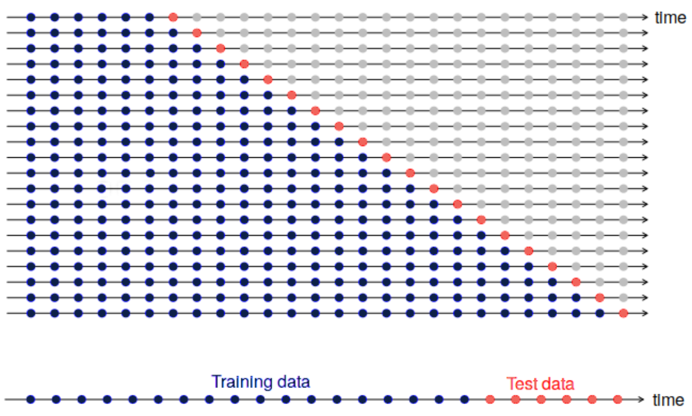


Figure 4.3: Time-Series Cross-Validation [47]

I have implemented my own system for creating cross-validation folds. Since the length of the datasets available is variable (cf. chapter 3), I define a minimum training size and forecast horizon and create the maximum amount of folds that are possible given those two numbers. Both of the numbers are somewhat arbitrary; in order to maintain interpretability, I define a minimum training size $N_{train,min} = 56\ days$ and a forecasting horizon of $fh = 28\ days$, corresponding to 8 and 4 weeks respectively.

4.3 Feature Engineering and Data Scaling

I am interested in boosting the predictive powers of my initial set of predictors through two elements of standard machine learning workflow: feature engineering and data normalization. These practices can be compared with common procedures in traditional statistical time-series prediction such as decomposition, deseasonalization or differentiation.

4.3.1 Feature engineering

Feature engineering allows for additional representational capability for my predictors [11]. External information can be especially useful for improving forecasting accuracy — adding precipitation data to one previously only containing temperature will greatly boost the predictive capability for a weather forecast system, for example. Unfortunately, I purposefully want to avoid assumptions about the business details of the merchants in my dataset to maintain generalization power. The inclusion of any additional data in the early stages would be speculative at best.

The features I create come from the data itself and my observation of its general structure and seasonality as explored in chapter 3; I simply explicitly model it for ease of access. First, the day of the week, day of the month and month are encoded as categorical features through one-hot encoding: for each categorical variable, n binary values are defined where n is the number of possible values for the given category – similarly to other research in the field [65]. Continuously valued historical features are also added for the transaction volume that took place 7, 14, 21 and 28 days ago, corresponding to weekly and monthly cycles. Finally, the value from the same last year is added to represent yearly cycles including holidays, which are fixed in the calendar. An overview of the features that form the input data for each machine learning predictor is presented in table 4.1.

Feature name	Feature type
Date	Continuous
Last observed transaction volume	Continuous
Historical values (7, 14, 21, 28, 365 days)	Continuous
Day of the week	Categorical (one-hot encoded)
Day of the month	Categorical (one-hot encoded)
Month of the year	Categorical (one-hot encoded)

Table 4.1: List of input features generated from time-series

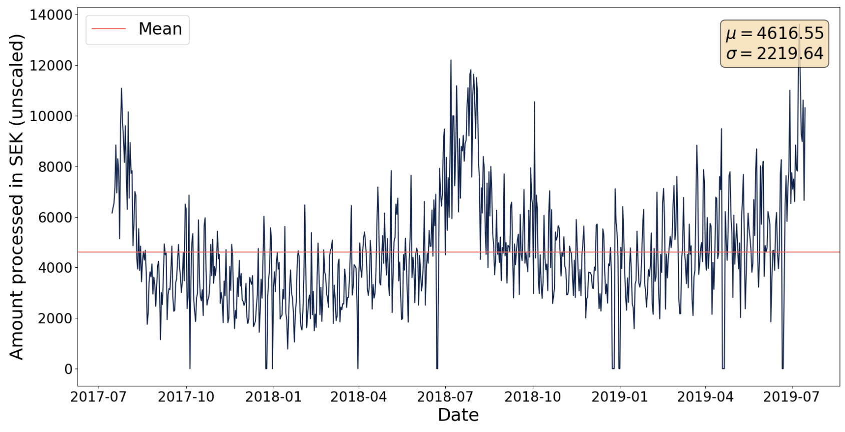
4.3.2 Data scaling

Data scaling is another standard machine learning method employed as a preprocessing step. Keeping input features on the same scale prevents them from converging at vastly different speeds during the execution of algorithms such as gradient descent; it is also essentially a requirement for some predictors [64]. Standardization specifically is the most common: it centers the input data to have a mean $\mu = 0$ and a variance $\sigma = 1$. The practice is so widespread that some machine learning libraries standardize all data by default — H2O for instance [37].

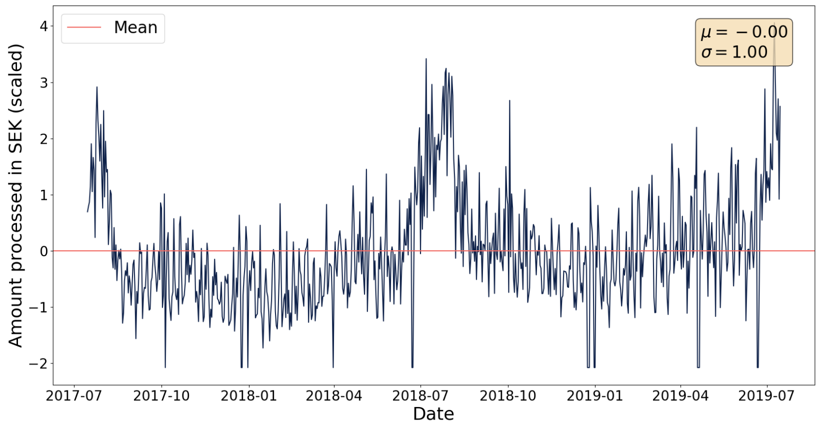
The effects of scaling are illustrated in figure 4.4. Note that despite the change in mean and variance, the patterns shown in the historical transaction data do not change. The line marking the mean daily transaction value does not move relative to other values, only relative to the y-axis. Shown values lose interpretability — negative transaction values have no analog with the real world. Unscaled targets ensure that final predictions maintain their validity.

4.4 Predictors

As stated in previous chapters, I use two main python-based libraries for my predictors: scikit-learn [66] for non-neural network approaches and Keras [21] otherwise. The only exception to this rule is the Multi-layer Perceptron; this is due to the MLP



(a) Sample merchant before data scaling



(b) Sample merchant after data scaling

Figure 4.4: Effect of standardization on a sample merchant's transactions. The vertical line marks the mean daily transaction value.

used in the M4 Competition as a benchmark being implemented in scikit-learn using specific parameters [54]. Note that by virtue of their purpose, the benchmark methods exclusively use the values of the time-series as input, meaning I cannot use my generated features as input. For this reason, I evaluate the MLP separately as a predictor using the proposed features, as well as keeping the version proposed in the M4 Competition as a benchmark.

Scikit-learn provides ready-made predictors — the relationship between the methods as described in chapter 2 and their corresponding classes is detailed in table

4.2. In digital versions of this work, it also includes a direct link to their source code. For the initial selection stage, I use the default parameters suggested by the library.

Prediction method	scikit-learn class
Ordinary Least Squares	`LinearRegression`
Regularized Least Squares (‘Ridge’)	`Ridge`
Lasso	`Lasso`
Elastic Net	`ElasticNet`
Least Angle Regression (LARS)	`Lars`
LARS Lasso	`LassoLars`
Orthogonal Matching Pursuit (OMP)	`OrthogonalMatchingPursuit`
Bayesian Regression	`BayesianRidge`
Stochastic Gradient Descent (SGD)	`SGDRegressor`
Random Sample Consensus (RANSAC)	`RANSACRegressor`
Kernelized Ridge Regressor	`KernelRidge`
Support Vector Machine (SVM)	`SVR`
K-Nearest Neighbors (KNN)	`NearestNeighbors`
Decision Trees	`DecisionTreeRegressor`
Adaboost	`AdaboostRegressor`
Gradient Tree Boosting	`GradientBoostingRegressor`
Multi-layer Perceptron	`MLPRegressor`

Table 4.2: List of predictors which use scikit-learn as their implementation library

Neural networks and their associated methods by their very nature do not often have pre-built implementations. Their components and behaviors are well optimized and understood but the configuration of the network (i.e. number of neurons, layers, etc.) varies wildly by application. I have based my configuration on existing research in the field, minimizing whenever possible the amount of arbitrary decisions taken when building the networks.

As stated previously, my MLP predictor is based on the one proposed as a benchmark for the M4 Competition — the major difference lies in the hidden layer: I scale up the number of neurons to accommodate for the input features detailed in section 4.3. The model further deviates from the defaults in `MLPRegressor` by using an identity activation function, using a maximum number of iterations of 100 and using an adaptive learning rate.

I utilize a similar approach with the simple recurrent neural network (RNN), taking the general structure from the one proposed by Makridakis et al [54] — expanded to take into consideration the generated input features. It features a fully connected RNN model provided by the `SimpleRNN` Keras method. I use linear activation functions without a bias, include no dropout and using an orthogonal kernel initializer. A densely-connected layer with a linear activation yields the final forecast variable.

For both LSTM and GRU-based networks, the underlying structure is based on a paper by Fischer et al [28]. The input layers are as large as made necessary by the input features, followed by a hidden layer made up of either of the recurrent neuron models, with a dropout value of 10%. The output densely-connected layer suggested is made up of one neuron with a $tanh$ activation function instead of two with softmax activation functions due to the intent to predict a continuous variable instead of two probabilities.

For the encoder-decoder network, I take the model provided by Lv et al [52] and [19]. I use two layers of LSTM blocks — comparable to Lv et al's stacked autoencoder layers — with as many units as there are input features. A final densely-connected layer with a $tanh$ activation function outputs the forecast variable.

As proposed by Fischer et al [28], criteria for early stopping is added to each of the Keras-based neural networks to efficiently manage computational resources. Using the `EarlyStopping` method, whenever the development of the forecast errors for my network stalls for a patience period of 10 iterations, the model assumes the local minimum has been found and cancels extraneous training epochs.

4.5 Hyperparameter Tuning

Most machine learning approaches essentially optimize within the solution space of variable coefficients within the model — linear weight coefficients for linear predictors (cf. equation 2.12) or neuron weights for neural networks (cf. equation 2.17). Most models, however, do not optimize for the parameters that underpin those models: regularization values for Ridge regression, number of layers in a neural network, activation functions, etc. These parameters are often called hyperparameters (not to be confused with the more classical Bayesian use of the term); optimizing them has become an important part of the machine learning workflow [84].

I opt for a simple grid search approach for the optimization of predictors' hyperparameters: I define a priori the search space for the different possible variables and retrain the model possible combination of values. The new model is trained over all the available datasets and their corresponding cross-validation folds to produce a final average OWA score. The search space is examined closely to determine if the search space should be expanded and make sure I have found the optimal local minima for the OWA score.

Running such an optimization for twenty-one approaches would be unfeasible: it would be an enormous waste of resources, both in terms of time and computational power. Forgoing such an effort, on the other hand, would be negligent. I opt instead for an iterative approach in which only the five most successful prediction methods undergo calculations for hyperparameter comparison. The precise nature of such a search depends entirely on the resulting top predictors — an in-depth description of my process in this section is thus impossible: tuning will be revisited in chapter 5.

5 Analysis and Discussion of Results

As shortly mentioned in chapter 4, results were generated iteratively in a two-step process (cf. figure 4.1). The results of the extraction of the top five basic predictors across the selected merchants are described in section 5.1. Those five predictors then undergo hyperparameter tuning to find the best possible predictor; the results are summarized in section 5.2. In section 5.3, I discuss the final identification of the top predictor and explain the reasons for my choice. Finally, I explore possible applications of external data to further boost the predictive performance of models in section 5.4.

5.1 Top basic predictors

After the first round of predictor evaluation using the merchant dataset, I calculated SMAPE, MASE and OWA for each of them as outlined in section 4.1.1. The results are shown in ascending order of OWA in table 5.1. The table includes the results for all benchmarks, which are marked as such in the 'B' column. Note how 'Recurrent Neural Network (RNN)' and 'Multilayer Perceptron (MLP)' have a benchmark and non-benchmark implementation as described in section 4.4. The five best performing predictors (highlighted at the top) in order of their OWA are:

1. Elastic Net
2. K-Nearest Neighbors (KNN)
3. Orthogonal Matching Pursuit (OMP)
4. Bayesian Regression
5. Gradient Tree Boosting (GTB)

In total I have eleven machine-learning predictors performing better than the best statistical prediction benchmark (Holt's exponential smoothing, cf. equation 2.10). This solidly confirms the hypothesis that machine learning predictors can outperform their statistical brethren in the chosen application, seemingly contradicting the results of the M4 competition. Out of the best performing eleven approaches, six are linear approaches — Elastic Net, OMP, Bayesian Regression, LARS Lasso, 'Ridge' and Lasso. The remaining five are nonlinear and non-neural network-based approaches: KNN, GTB, Adaboost, SGD and SVM.

Predictor	B	SMAPE	MASE	OWA
Elastic Net	☐	0.687974	0.783469	0.690933
K-Nearest Neighbors (KNN)	☐	0.700944	0.771044	0.693738
Orthogonal Matching Pursuit (OMP)	☐	0.698594	0.802952	0.704376
Bayesian Regression	☐	0.701045	0.832600	0.716932
Gradient Tree Boosting (GTB)	☐	0.711953	0.834818	0.724051
AdaBoost	☐	0.692842	0.912523	0.742243
LARS Lasso	☐	0.749876	0.908155	0.773469
Linear SVM using SDG	☐	0.753958	0.905661	0.774884
Regularized Least Squares (‘Ridge’)	☐	0.764505	0.936966	0.792728
Support Vector Machine (SVM)	☐	0.747994	1.004355	0.808542
Lasso	☐	0.777544	0.973224	0.813870
Holt’s Exponential Smoothing	☒	0.740176	1.166140	0.864843
ARIMA	☒	0.752141	1.150395	0.865821
Combined Smoothing (S+H+D)	☒	0.743846	1.165242	0.866620
Simple Exponential Smoothing	☒	0.747080	1.166810	0.869073
Theta Method	☒	0.749603	1.167188	0.870669
Dampened Exponential Smoothing	☒	0.749377	1.167600	0.870694
Autoencoder + MLP	☐	1.014880	0.783846	0.879465
Decision Trees	☐	0.867117	1.060967	0.898468
Multilayer Perceptron (MLP)	☒	0.778066	1.216986	0.905789
Recurrent Neural Network (RNN)	☒	0.792998	1.231913	0.920004
Gated Recurrent Unit Network (GRU)	☐	1.105685	0.947965	0.993478
Naive2	☒	0.867630	1.330322	1.000000
sNaive	☒	0.867630	1.330322	1.000000
Naive	☒	0.867630	1.330322	1.000000
LSTM Network	☐	1.172546	0.932932	1.026359
Kernelized Ridge Regressor	☐	1.544584	1.966455	1.629206
Gaussian Process Regressor	☐	1.985689	1.787963	1.816321
Recurrent Neural Network (RNN)	☐	1.110724	53.647110	20.80329
Multilayer Perceptron (MLP)	☐	1.200850	89.98625	34.51325
Ordinary Least Squares (OLS)	☐	1.085204	996778.9	374638.8
Least Angle Regression (LARS)	☐	1.259683	9.607e+42	3.611e+42

Table 5.1: Results of the evaluation of the initial 21 predictors, including benchmarks and ordered by ascending value of OWA

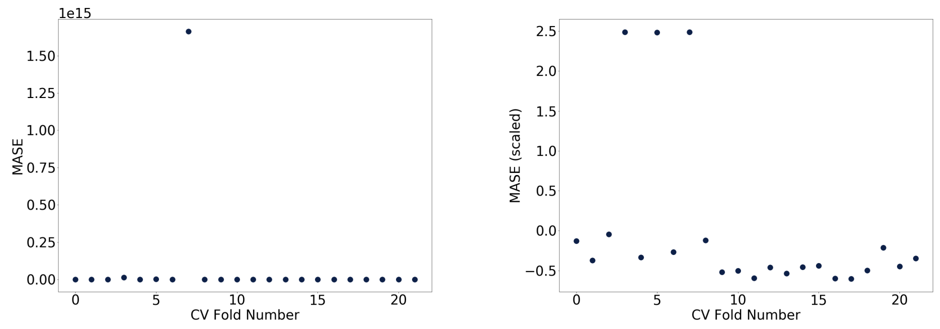
Performance of neural networks

The relatively poor performance of neural networks deserves a closer inspection. The best-performing approach was the Autoencoder-based method using hidden LSTM units, which managed to outperform both neural network benchmarks and the naive approaches, but fell short of the smoothing and ARIMA algorithms. The M4 Competition's results are a clear indicator that highly-complex neural networks can yield particularly good results in the field of time-series prediction: Smyl et al's work [79] performed much better than any other such method. This notion is supported by a wealth of papers using different manners of neural networks in their research, including the ones on which I based the structure of my own implementations (see section 4.4), despite concerns raised by Makridakis et al [54] regarding the validity of those results.

I consider that fine-tuning the architecture, hyperparameters and input of any of neural networks to achieve better results is eminently possible — indeed it is what I would have done had any of them landed within the best performing predictors. This would, however, go against my overall research goals: I want to survey the entirety of machine learning approaches to find the best model, not reverse engineer a hypothetically better-performing network architecture. The merit of such an endeavour in terms of resource efficiency is also questionable. To illustrate this point, I measured the training time for the top predictors and my simple LSTM network on a local machine. The single-threaded process was run on a Ubuntu 18.04.3 system with an Intel Core i5-4670K CPU with a base frequency of 3.40 GHz. I measured the training time for each of the initial 20 merchants, taking the entire time series into consideration. The results are shown in table 5.2. Even the longest-running top predictor does not reach an average training time of 3% of what a simple LSTM model requires. While there are many methods to streamline the development of a deep learning network — especially GPU acceleration and outsourcing of computing to a cloud service — I deem these efforts clearly out of the scope of my current work.

Predictor	Mean Training Time in ms
K-Nearest Neighbors(KNN)	0.871
Elastic Net	2.121
Orthogonal Matching Pursuit (OMP)	5.477
Bayesian Regression	6.028
Gradient Tree Boosting (GTB)	141.921
LSTM	5046.269

Table 5.2: Average training times for my top predictors and two neural network predictors



(a) A single outlier skews MASE values completely

(b) After scaling, three CV folds are discovered as the culprits

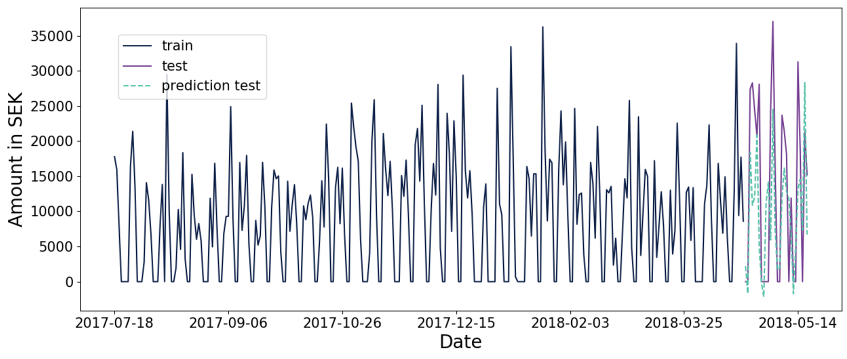
Figure 5.1: Values of MASE across CV folds in an example merchant, before and after scaling

Exploding MASE values

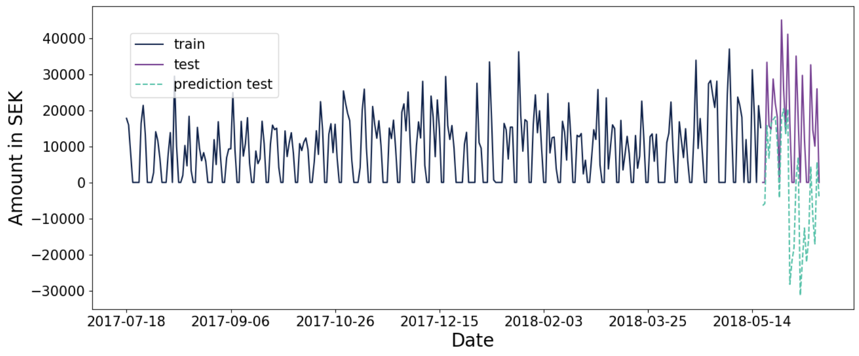
Another irregularity in my results deserves attention. The progression of SMAPE is relatively stable: the values fluctuate between 0.687974 and 1.985689, mostly in small increments. MASE — and by extention OWA — exhibits very different behavior in the four worst-performing predictors: RNN, MLP, OLS and LARS. LARS shows particularly egregious values for MASE, many orders of magnitude beyond its Lasso version for example. How can this behavior be explained? Looking at the individual MASE values, I notice they are consistently high across merchants, but highly volatile across CV folds. The behavior is shown in figure 5.1 for an example merchant.

In subfigure 5.1a, one outlier stands out among the rest, so much so that it dwarfs other unreasonable MASE values. A Box-Cox transformation scaling on the MASE data, as shown in subfigure 5.1b, reveals that a further two values are the culprits for this particular merchant's exploding MASE. Comparing two generated predictions from consecutive folds throws this into a starker contrast, see figure 5.2. Most folds behave like subfigure 5.2a, with MASE values within acceptable bounds, but folds such as the one in subfigure 5.2b skew the results towards values well in the trillions.

One could argue that such a behavior raises doubts about the suitability of MASE as a useful measurement for forecast accuracy. Exploding MASE values only affect a small handful of predictors, however — the most unstable ones. Predictions such as the one shown in subfigure 5.2b are completely nonsensical; models that produce such results are accordingly heavily punished. Predictors such as OLS, RNN or MLP show similar behavior, in much smaller scales.



(a) The fold preceding the largest outlier, MASE = 1.370340



(b) Largest outlier, MASE = 1.7×10^{15}

Figure 5.2: Two consecutive CV folds yield diametrically opposed predictions — and drastically different MASE values

5.2 Results of Hyperparameter Tuning

After the preliminary selection of the 5 best-performing predictors, I optimize over their hyperparameters as described in 4.5. As previously mentioned, this is the stage in which I define exactly which hyperparameters are optimized over. Since all of the top predictors are implemented in the `scikit-learn` library, I limit myself to the input parameters defined therein. The final list of values used for each of the top predictors' grid search is shown in table 5.3.

The values themselves were also derived using a small iterative process when necessary: base values were initially chosen and the resulting scores analyzed to corroborate whether a local minimum had been found within the vicinity of the initial default values. Whenever values were missing, they were added to the search. The criteria for comparing different combinations of parameters was the OWA, as

Parameter name	Values
Bayesian Ridge	
α_1	$10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}$
α_2	$10^{-7}, 10^{-6}, 10^{-5}$
λ_1	$10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}$
λ_2	$10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}$
Elastic Net	
α	$0.01, 0.1, 1, 100$
l_1 ratio	$0.1, 0.2, ..., 0.9, 1.0$
K-Nearest Neighbors (KNN)	
Number of neighbors	$3, 5, 7, 11, 15, 17, 19$
Weights	‘uniform’, ‘distance’
p	$1, 2$
Algorithm	‘ball_tree’, ‘kd_tree’, ‘brute’
Gradient Tree Boosting (GTB)	
Loss	‘ls’, ‘lad’, ‘huber’, ‘quantile’
Learning rate	$0.001, 0.01, 0.1, 1$
Criterion	“friedman_mse’, ‘mse’, ‘mae’
Maximum tree depth	$2, 3, 4, 5, 6$
Orthogonal Matching Pursuit(OMP)	
Non-zero coefficients	$1, 2, 3, 4, 5, 6, 7, 8$

Table 5.3: List of hyperparameters used for each of the top predictors’ grid search

in the rest of my work. An illustration of this process is shown exemplarily in figure 5.3. A very clear minimum can be seen for a learning rate of 0.1 and a ‘lad’ loss, which stands for least absolute deviation. The learning rate does not need to be adjusted beyond the given boundaries since any deviation from 0.1 results in an OWA increase.

The hyperparameters in table 5.3 are not the only parameters available for tuning for most of the top predictors’ methods. All of them include further values that can be adjusted for computational performance and speed of prediction without affecting the actual accuracy of said prediction — e.g. presorting data, ‘warm starts’, early stopping tolerance, etc. For OMP, Elastic Net, Bayesian Ridge and KNN there are no further parameters that can substantially affect the prediction accuracy of the predictors. Gradient Tree Boosting by its very nature is highly customizable — I have selected the parameters I deemed the most important for the search.

For each possible combination of hyperparameters, I run the same number of instances of the predictor as for the default version. This means I rerun every CV fold for each of the twenty top merchants. For GBT for instance, this results in a

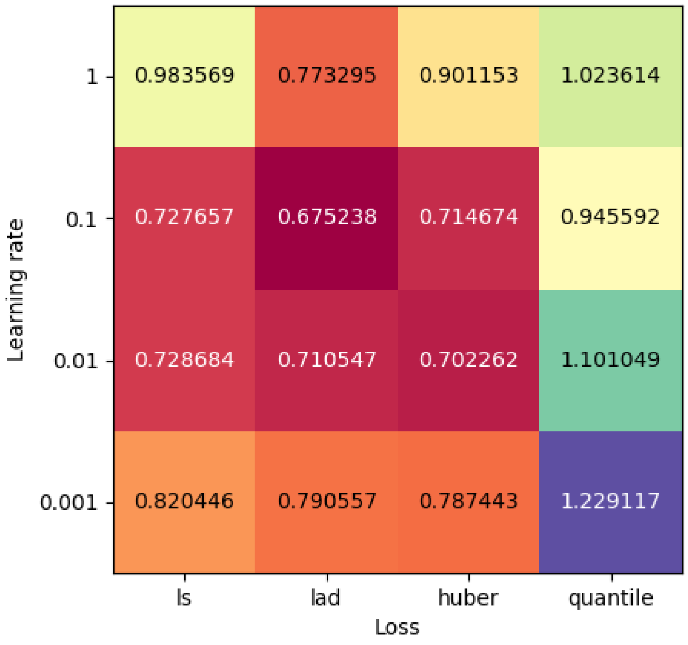


Figure 5.3: A local minimum is found in the 'loss' and 'learning rate' plane for GBT

further 240 individual predictors that undergo the same process as the initial 21 predictors. The results of this process are summarized in table 5.4, in ascending order of their OWA scores.

Grid search manages to achieve lower OWA for all of the top predictors, with particular gains being made in the algorithms with the most parameters available for exploration: KNN and GTB. GTB is particularly remarkable for leapfrogging three of the other predictors and landing as a close second to KNN.

5.3 Identifying the top predictor

Once the two previous evaluation stages have been cleared, I have a strong basis by which to select the best predictor for time-series prediction in the card payment industry. All of the top predictors have shown to be accurate and robust over both different merchants and different segments of their dataset. As shortly illustrated in table 5.2, all of their training times are in the same order of magnitude. If the only selection criteria are the experiments conducted in this work, the choice is clear: K-Nearest Neighbors achieves the best overall weighted average at the end of the selection process. If however, my goal is to select the best predictor for a

Final Hyperparameters	SMAPE	MASE	OWA
K-Nearest Neighbors (KNN)			
Algorithm = ‘brute’, number of neighbors = 13, $p = 1$, weights = ‘distance’	0.700944	0.771044	0.671867
Gradient Tree Boosting (GTB)			
Criterion = ‘mse’, learning rate = 0.1, loss = ‘lad’, maximum tree depth = 3	0.711953	0.834818	0.675151
Elastic Net			
α = 10, l_1 ratio = 0.9	0.687974	0.783469	0.684170
Orthogonal Matching Pursuit(OMP)			
Non-zero coefficients = 5	0.698594	0.802952	0.704376
Bayesian Ridge			
$\alpha_1 = 10^{-5}$,$\alpha_2 = 10^{-5}$, $\lambda_1 = 10^{-6}$,$\lambda_2 = 10^{-5}$	0.701045	0.832600	0.716702

Table 5.4: Results after hyperparameter tuning for my top predictors

system used in real business environments that can generate actionable insights for merchants regularly, the calculation changes substantially.

The top predictors are separated by slim margins: OWA values exhibit a range of around 0.045 between the first place KNN and the worst performing Bayesian Ridge. It is unlikely I can extract much more prediction accuracy from my current library and data input setup; an attempt is made further ahead in section 5.4. If accuracy gains yield diminishing returns, resource utilization and ease of deployment become a larger factor. I should also consider the existence of additional implementations or the availability of literature on the subject.

In this regard, Gradient Tree Boosting stands head and shoulders above the competition. Its capacity for more fruitful fine-tuning was already suggested in section 5.2, but it is the existence of the XGBoost library [18] that is the most substantive factor. XGBoost is a gradient boosting library that includes extensive support for GBT and offers packages for Python, JVM, R and Ruby among others. It also natively supports parallelization and deployment into virtual and distributed environments. The library’s outstanding results, especially when dealing with structured data have been noted in multiple publications [20] [57] [65]. The ease of scaling for GBT with XGBoost compares very favorably to KNN: nearest neighbor approaches retain information for all possible datapoints — a large impediment when dealing with larger databases. Efforts exist to assuage this problem [17], but they are not nearly as well regarded and maintained as XGBoost.

Taking all of the previous points into consideration, I believe that **Gradient Tree Boosting** is the best possible predictor for time-series prediction in the financial sector.

5.4 Possible external sources of data

As previously mentioned, this section is more speculative in nature and not formally part of the selection process for the top predictor. Up to now I have exclusively used information present within the input data: daily transaction volumes, lagged values, various forms of representing dates, etc. The consumer-facing nature of the available dataset, however, lets me speculate about the kind of external data that might affect shopping and consumption patterns. A café with a terrace will have very different transaction volumes if it is unseasonably cold outside or if a particular Saturday falls on a bank holiday. Since I have not developed a rigorous system to select for features, I present the results of adding such data as an addendum: they might form the basis for future efforts towards increasing the predictive capabilities of a system in a production setting. It is worth noting, as was mentioned in chapter 3, that all of the data comes from small to medium-sized merchants in the Swedish market.

Features and data sources

The first feature I add to the input data is historical temperature levels. Since I am working with anonymized data for each of the merchants, I do not have exact locations for any of the businesses provided in the dataset. Luckily, almost half of Sweden's population is concentrated in three cities: Stockholm, Gothenburg and Malmö. I thus add historical temperature means for the three cities provided by the Swedish Meteorological and Hydrological Institute [81]. I run the top 5 predictors as a benchmark to find out which combination of the cities temperature results in lower OWA values: using exclusively Stockholm's temperature proves to be the most fruitful. This may point towards a higher concentration of available businesses in the area around the Swedish capital. The resulting feature is added to the input data as a continuously valued vector. Secondly, I add a categorical feature for whether any given day is a holiday in Sweden. This information was extracted from Google's open holiday API [36]. The added feature vector simply marks with a 1 every day that is a holiday, the rest are kept at 0.

After evaluating the top predictors in the usual manner, I observed that each independent feature slightly increased the predictive accuracy of the models. The best gains were achieved when utilizing both sources in the input data. The corresponding results are summarized in table 5.5. As speculated in the last chapter, the gains

Predictor	SMAPE	MASE	OWA
K-Nearest Neighbors (KNN)	0.676191	0.727024	0.662928
Gradient Tree Boosting (GTB)	0.685379	0.720477	0.665762
Elastic Net	0.685512	0.763385	0.681966
Orthogonal Matching Pursuit (OMP)	0.699702	0.801672	0.704533
BayesianRidge	0.699414	0.817401	0.710279

Table 5.5: Results of the addition of two external sources of data for the top 5 predictors, ordered by ascending value of OWA

from additional features are noticeable but not groundbreaking — the order of the top predictors does not change.

KNN sheds around 0.0089 in its OWA score, while GTB closes the distance with a 0.0094 difference. The values are of course almost negligible, but I can ascertain that there is further space for improvement when adding external values to my input feature vectors. The addition of external information and the reimplementation of gradient tree boosting through XGBoost are undoubtedly the next steps towards an accurate and scalable prediction system.

6 Conclusion

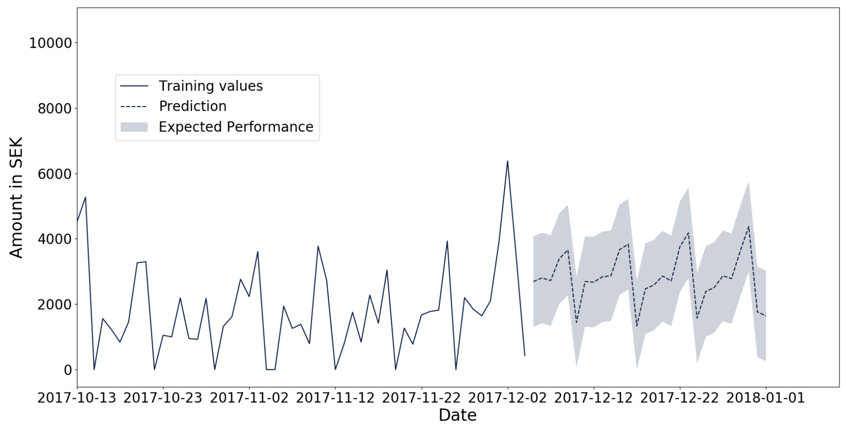
In this thesis, I set out to answer three main questions regarding the viability of machine learning-based prediction in the financial industry. In order to answer them, I conducted a wide-ranging literature research and implemented twenty-one predictors through well established machine learning libraries. I set up benchmark predictors, particularly from traditional statistical methods, as well as measurements as recommended by the best available time-series competition. I then submitted the predictors to a rigorous evaluation and cross-validation process through the data of a variety of mid-sized and small merchants in the Swedish market. I established that a number of machine learning methods can outperform their statistical counterparts consistently. In an attempt to find the most accurate predictor possible, I fine-tuned the top five predictors through a hyperparameter grid search. I also speculatively explored the addition of external data to further enhance predictive capabilities. In the end I identified both the most accurate predictor according to my experiments and the best predictor to implement a working prediction system with.

The most accurate predictor after the optimization process was K-Nearest Neighbors, with Gradient Tree Boosting coming in at a close second. Due to concerns about the scalability of KNN for larger datasets, as well as the existence of dedicated libraries for gradient boosting, I deem Gradient Tree Boosting the best available prediction method for the chosen use case. Notably, no neural network outperformed all statistical prediction methods, casting doubt on the value of the significant time and computational investment needed to develop and train such algorithms. Adding additional data sources can lead to more accurate results, albeit only slightly.

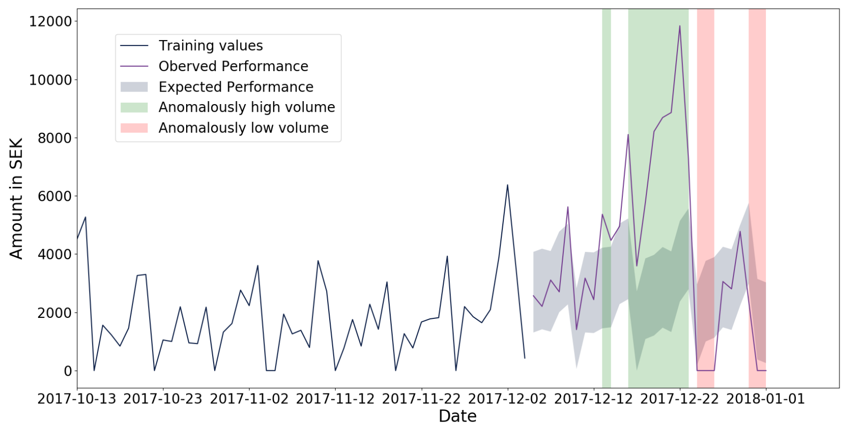
While I can generate acceptably accurate predictions for the transaction volumes of merchants, extracting actionable insights from these point forecasts is challenging. In section 6.1, I propose the basics of a system that can bridge this gap and give merchants an idea of what the health of their business is. Finally, I propose avenues of future research in section 6.2.

6.1 Towards actionable insights for merchants

Point forecasts always generate a certain error term — unless the time-series investigated is by its own nature trivial to predict. Merchants seeking to extract insights from such a forecast would be hard-pressed to do so: how much deviation is normal, how much is alarming? For this reason, forecasts are often generated together with prediction intervals, which are the space within I expect the predicted variable to lie with a previously defined probability [47]. In this section I would like to sketch the basics of an approach that uses such a simple prediction interval to intuitively highlight unusual behavior in merchant transaction values, see figure 6.1. In this example I have purposefully chosen a predictor that has not yet observed behavior patterns for the Christmas season, which is often very anomalous.



(a) Prediction intervals are created from a point forecast



(b) Deviations from the prediction interval are highlighted

Figure 6.1: The upper and lower bounds of a prediction interval can be used to mark unexpected volumes of transactions for a merchant

I generate a prediction interval by first calculating the standard deviation of the residuals observed during training. I then multiply this value with the desired coverage probability, e.g. 80%. The upper and lower bounds of the prediction interval are then calculated in the following manner:

$$\hat{y}_{T+h|T} \pm c\hat{\sigma} \tag{6.1}$$

where $\hat{y}_{T+h|T}$ is the prediction at time $T+h$ with T being the last observed value, c being the coverage probability and σ the observed residual variance [47]. An example is shown in subfigure 6.1a, with both the generated point forecast and prediction interval marked. Merchants can thus expect at least 80% of their transaction values to fall within the marked intervals. Deviation from this range, either positive or negative, can be used as a heuristic for the health of the business. Subfigure 6.1b marks both cases. It is plausible to imagine a business owner corroborating that their big Christmas sale went better than predicted and that the following slump is simply a product of taking more time off towards the end of the year. Too many days in a row of underperformance would deserve close inspection from management. This approach can be combined with other forecasts for e.g. monthly revenue.

While generating prediction intervals can bridge the gap between prediction and a suggested course of action, the design choices must be tested and validated. There are multiple ways to calculate prediction intervals — more volatile businesses may benefit from wider ranges, while larger merchants may prefer the opposite. The number of successive days of underperformance that should result in alarm or the relationship between daily transaction values and long-term revenue are open points. All of these decisions should be validated with real businesses to be able to generate valuable, transferable insights.

6.2 Future Work

I have already explored the possible next steps for a working prediction system in a production setting in previous sections and chapters. In regards to the goals presented in this work, many avenues of future research are possible.

In this work, every single merchant was handled separately: every predictor was trained exclusively on one time-series — without explicit information about the merchant itself, I could not guarantee that any two merchants would have similar patterns. Investigating the ability to generalize for all businesses within a given sector, i.e. all clothing retailers, all restaurants, etc., would be sure to yield interesting results. If most knowledge is transferable from one merchant to the other in this scenario, predictions could even be generated for new merchants without them needing to provide their historical transaction data. The same approach could be used to compare different regions: merchants in Scandinavian markets are likely to share many similarities between each other.

There is evidently a lot of work to be done to streamline predictor models for better computational performance. Reimplementation in a dedicated library and parallelization, have been mentioned multiple times already. Evaluating the importance of different features, as well as finding better performing features could form the basis for valuable research into methods for feature selection in the financial sector. While I have speculatively explored only a few of the most obvious possible external sources of data, macroeconomic indicators might yield better predictors for small business performance [27]. Many of these indices are openly published and widely accessible.

Finally, neural networks have a complex track record when dealing with time-series prediction. There are a wealth of research papers published every year touting the results of using a variety of neural network approaches for prediction, despite limited reproducibility of results and even questionable validity [54]. Indeed, my own neural network implementations did not fare better than their statistical counterparts. A more promising approach may be found in hybrid methods, such as the winner of the M4 Competition [79], which used complex networks to derive hyperparameters for smoothing approaches instead of creating predictions directly. Neural networks have powered breakthroughs in fields ranging from games to image and speech recognition — declining computational costs and increased availability of data may yet enable similar leaps for the field of time-series prediction.

List of Figures

List of Tables

Bibliography

1. B. Abraham and J. Ledolter. *Statistical Methods for Forecasting*, volume 147 of *Wiley Series in Probability and Statistics*. John Wiley & Sons, Inc., Hoboken, NJ, USA, sep 2006. ISBN 9780470316610. doi:10.2307/2981583. URL `http://doi.wiley.com/10.1002/9780470316610`.

2. N.K. Ahmed, A.F. Atiya, N. El Gayar, and H. El-Shishiny. An empirical comparison of machine learning models for time series forecasting. In *Econometric Reviews*, 29(5), pp. 594–621, 2010. ISSN 07474938. doi:10.1080/07474938.2010.481556.

3. K.P. Ajoy and P. Dobrivoje. *Computational Intelligence in Time Series Forecasting - Theory and Engineering Applications*. Springer, 2014. ISBN 9780874216561. doi:10.1007/s13398-014-0173-7.2.

4. H. Akaike. A new look at the statistical model identification. In *IEEE Transactions on Automatic ControlAkaike, H. (1974). A new look at the statistical model identification. IEEE Transactions on Automatic Control, 19(6), 716–723. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1100705*, 19(6), pp. 716–723, 1974. URL `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp={&}arnumber=1100705`.

5. V. Assimakopoulos and K. Nikolopoulos. The theta model: A decomposition approach to forecasting. In *International Journal of Forecasting*, 16(4), pp. 521–530, 2000. ISSN 01692070. doi:10.1016/S0169-2070(00)00066-2.

6. J.M. Bates and C.W.J. Granger. The Combination of Forecasts. In *Operational Research Quarterly*, 20(4), p. 451, 1969. ISSN 14732858. doi:10.2307/3008764. URL `https://www.jstor.org/stable/3008764?origin=crossref`.

7. C. Bergmeir, R.J. Hyndman, and B. Koo. Validity of Cross-Validation for Evaluating Time Series Prediction. In *Elsevier*, 120(April), 2017.

8. C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics*. Springer-Verlag, Berlin, 2006. ISBN 0387310738. URL `https://www.springer.com/de/book/9780387310732?referer=www.springer.de`.

9. T. Bollerslev. Generalized Autoregressive Conditional Heteroskedasticity. In *Journal of Econometrics*, 31, pp. 307–327, 1986. ISSN 03044076. doi:10.1016/0304-4076(86)90063-1. URL `https://pdfs.semanticscholar.org/7da8/bfa5295375c1141d797e80065a599153c19d.pdf`.

10. G. Bontempi, S. Ben Taieb, and Y.A. Le Borgne. Machine Learning Strategies for Time Series Forecasting. In *International Series in Operations Research and Management Science*, volume 186, pp. 62–77. 2013. doi:10.1007/978-3-642-36318-4_3.

11. L. Bottou. *Feature Engineering*. Technical Report, Princeton University, Princeton, NJ, 2010.

12. G.E. Box, G.M. Jenkins, and G.C. Reinsel. *Time series analysis: Forecasting and control: Fourth edition*. Wiley Series in Probability and Statistics. Wiley, jun 2013. ISBN 9781118619193. doi:10.1002/9781118619193. URL `https://onlinelibrary.wiley.com/doi/book/10.1002/9781118619193`.

13. L. Breiman. Arcing the edge. In *Statistics*, 4, pp. 1–14, 1997.

14. L. Breiman. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). In *Statistical Science*, 16(3), pp. 199–231, 2002. ISSN 0883-4237. doi:10.1214/ss/1009213726.

15. R.G. Brown. Exponential smoothing for predicting demand. In *Operations Research*, volume 5, p. 145. INST OPERATIONS RESEARCH MANAGEMENT SCIENCES 901 ELKRIDGE LANDING RD, STE˜. . . , 1957.

16. Capgemini and BNP Paribas. *World Payment Report 2018*. Technical Report, 2018.

17. G. Chatzigeorgakidis, S. Karagiorgou, S. Athanasiou, and S. Skiadopoulos. FML-kNN: scalable machine learning on Big Data using k-nearest neighbor joins. In *Journal of Big Data*, 5(1), 2018. ISSN 21961115. doi:10.1186/s40537-018-0115-x. URL `https://doi.org/10.1186/s40537-018-0115-x`.

18. T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, pp. 785–794. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4232-2. doi:10.1145/2939672.2939785. URL `http://doi.acm.org/10.1145/2939672.2939785`.

19. K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 2014.

20. F. Chollet. *Deep Learning with Python*. 1st edition. 2018. ISBN 9781617294433.

21. F. Chollet and Others. Keras. \url{https://keras.io}, 2015.

22. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. In Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, pp. 1–9, 2014. URL `http://arxiv.org/abs/1412.3555`.

23. P.H. Cootner. *The random character of stock market prices*. M.I.T. Press, 1964. URL `https://books.google.de/books?id=7XGvQgAACAAJ`.

24. Deloitte Australia. In SME Digital Payments: New opportunities to optimise, 2017. URL `https://www2.deloitte.com/content/dam/Deloitte/au/Documents/financial-services/deloitte-au-fs-zenith-payments-landscape-report-final-220517.pdf`.

25. B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least Angle Regression. In *The Annals of Statistics*, 32(2), pp. 1–44, 2003.

26. EMarketer. E-commerce share of total global retail sales from 2015 to 2021. In *Statista*, p. 2021, 2019. URL `https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/`.

27. J. Everett and J. Watson. Small Business Failure and External Risk Factors. In *Small Business Economics*, 11(4), pp. 371–390, 1998. ISSN 1573-0913. doi:10.1023/A:1008065527282. URL `https://doi.org/10.1023/A:1008065527282`.

28. T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. In *European Journal of Operational Research*, 270(2), pp. 654–669, 2018. ISSN 03772217. doi:10.1016/j.ejor.2017.11.054. URL `https://doi.org/10.1016/j.ejor.2017.11.054`.

29. M.a. Fischler and R.C. Bolles. Random Sample Paradigm for Model Consensus: A Apphcatlons to Image Fitting with Analysis and Automated Cartography. In *Communications of the ACM*, 24(6), pp. 381–395, 1981. ISSN 00010782. doi:10.1145/358669.358692.

30. J. Friedman, T. Hastie, and R. Tibshirani. Regularization Paths for Generalized Linear Models via Coordinate Descent. In *Journal of Statistical Software*, 33(1), pp. 1–3, 2010.

31. J.H. Friedman. Greedy function approximation: A gradient boosting machine. In *The Annals of Statistics*, 29(5), pp. 1189–1232, oct 2001. ISSN 0090-5364. doi:10.1214/aos/1013203451. URL `https://www.jstor.org/stable/2699986http://projecteuclid.org/euclid.aos/1013203451`.

32. E.S. Gardner and E. McKenzie. Forecasting Trends in Time Series. In *Management Science*, 31(10), pp. 1237–1246, 1985. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2631713`.

33. L. Gautier and R. contributors. rpy2 - R in Python. 2008. URL `https://rpy2.github.io/`.

34. J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *Computer*, 17, pp. 513–520, 2004. URL `http://eprints.pascal-network.org/archive/00001570/`.

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

36. Google LLC. Calendar API | Google Developers. 2019. URL `https://developers.google.com/calendar`.

37. H2O.ai. Python Interface for H2O. sep 2019. URL `https://github.com/h2oai/h2o-3`.

38. C. Hamzaçebi, D. Akay, and F. Kutay. Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting. In *Expert Systems with Applications*, 36(2 PART 2), pp. 3839–3844, 2009. ISSN 09574174. doi:10.1016/j.eswa.2008.02.042.

39. T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Second edi edition. Springer, aug 2017. URL `http://www.tandfonline.com/doi/abs/10.1198/tech.2003.s770`.

40. M. Hibon and S. Makridakis. The M3-Competition: results, conclusions and implications ‘. In *International Journal of Forecasting*, 16, pp. 451–476, 2000.

41. G.E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. In *Neural computation*, 18(7), pp. 1527–54, jul 2006.

42. S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. In *Neural Computation*, 9(8), pp. 1735–1780, 1997. ISSN 08997667. doi:10.1162/neco.1997.9.8.1735. URL `http://www7.informatik.tu-muenchen.de/{~}hochreit{%}0Ahttp://www.idsia.ch/{~}juergen`.

43. C.C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. In *International Journal of Forecasting*, 20(1), pp. 5–10, 2004. ISSN 0169-2070. doi:https://doi.org/10.1016/j.ijforecast.2003.09.015. URL `http://www.sciencedirect.com/science/article/pii/S0169207003001134`.

44. M.J.C. Hu and H.E. Root. An Adaptive Data Processing System for Weather Forecasting. In *Journal of Applied Meteorology*, 3(5), pp. 513–523, oct 2002. ISSN 0021-8952. doi:10.1175/1520-0450(1964)003<0513:aadpsf>2.0.co;2. URL `http://journals.ametsoc.org/doi/abs/10.1175/1520-0450{%}281964{%}29003{%}3C0513{%}3AAADPSF{%}3E2.0.CO{%}3B2`.

45. R.J. Hyndman. Box-Jenkins Modeling. In H. Daellenbach and R. Flood (eds.), *Informed Student Guide to Management Science*, number May, 1st edition, pp. 1–2. London, 2002. doi:10.4135/9781412950589.n80. URL `https://robjhyndman.com/papers/BoxJenkins.pdf`.

46. R.J. Hyndman. Publications Page. aug 2019. URL `https://robjhyndman.com/publications`.

47. R.J. Hyndman and G. Athanasopoulos. In Forecasting : Principles and Practice, 2018.

48. R.J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. In *Journal of Statistical Software*, 26(3), pp. 1–22, 2008. URL `http://www.jstatsoft.org/article/view/v027i03`.

49. R.J. Hyndman and A.B. Koehler. and Business Statistics Another Look at Measures of Forecast Accuracy Another look at measures of forecast accuracy. In *International journal of forecasting*, 22(November), pp. 679–688, 2005. ISSN 01692070. doi:10.1016/j.ijforecast.2006.03.001. URL `http://www.sciencedirect.com/science/article/pii/S0169207006000239{%}5Cnhttp://core.ac.uk/download/pdf/6340761.pdf`.

50. T. Kuremoto, S. Kimura, K. Kobayashi, and M. Obayashi. Time series forecasting using a deep belief network with restricted Boltzmann machines. In *Neurocomputing*, 137(July), pp. 47–56, 2014. ISSN 18728286. doi:10.1016/j.neucom.2013.03.047. URL `http://dx.doi.org/10.1016/j.neucom.2013.03.047`.

51. M. Lopez de Prado. *Advances in Financial Machine Learning*. Wiley, Hoboken, NJ, 2018. ISBN 9781119482086. doi:10.1007/s10551-015-2769-z.For.

52. Y. Lv, Y. Duan, W. Kang, Z. Li, and F.Y. Wang. Traffic Flow Prediction with Big Data: A Deep Learning Approach. In *IEEE Transactions on Intelligent Transportation Systems*, 16(2), pp. 865–873, 2015. ISSN 15249050. doi: 10.1109/TITS.2014.2345663.

53. S. Makridakis, E. Spiliotis, and V. Assimakopoulos. In Competitor's Guide: Prizes and Rules Contents, pp. 1–7, 2018. URL `https://www.m4.unic.ac.cy/wp-content/uploads/2018/03/M4-Competitors-Guide.pdf`.

54. S. Makridakis, E. Spiliotis, and V. Assimakopoulos. Statistical and Machine Learning forecasting methods: Concerns and ways forward. In *PLoS ONE*, 13(3), pp. 1–26, 2018. ISSN 19326203. doi:10.1371/journal.pone.0194889.

55. S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M4 Competition: Results, findings, conclusion and way forward. In *International Journal of Forecasting*, 34(4), pp. 802–808, 2018. ISSN 01692070. doi:10.1016/j.ijforecast.2018.06.001. URL `https://doi.org/10.1016/j.ijforecast.2018.06.001`.

56. F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, and J. Riquelme. A Survey on Data Mining Techniques Applied to Electricity-Related Time Series Forecasting. In *Energies*, 8(11), pp. 13162–13193, nov 2015. ISSN 1996-1073. doi:10.3390/en81112361. URL `http://www.mdpi.com/1996-1073/8/11/12361`.

57. P. Mehta, M. Bukov, C.H. Wang, A.G.R. Day, C. Richardson, C.K. Fisher, and D.J. Schwab. In A high-bias, low-variance introduction to Machine Learning for physicists, 2018. URL `http://arxiv.org/abs/1803.08823`.

58. M. Minsky and S. Papert. Perceptron: an introduction to computational geometry. 1969.

59. D.C. Montgomery, C.L. Jennings, and M. Kulahci. *Time-Series — Analysis, Model, and Forecasting*. Second edi edition. Wiley, 2016. ISBN 978-3-319-28723-2. doi:10.1142/9789814723855_0025. URL `http://link.springer.com/10.1007/978-3-319-28725-6`.

60. K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012. doi:10.1007/springerreference.

61. R.M. Neal. *Bayesian Learning for Neural Networks*. Ph.D. thesis, University of Toronto, 1996. doi:10.1007/978-1-4612-0745-0. URL `http://link.springer.com/10.1007/978-1-4612-0745-0`.

62. Oliver Wyman. In EU Retail and SME Payments - state of the industry, p. 68, 2016. URL `http://www.oliverwyman.com/our-expertise/insights/2016/nov/eu-retail-and-sme-payments.html`.

63. Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Conference Record of the Asilomar Conference on Signals, Systems & Computers*, 1, pp. 40–44, 1993. ISSN 10586393.

64. J. Patterson and A. Gibson. *Deep Learning: A Practicioner's Approach*. 2nd edition. O'Reilly & Associates, 2017. ISBN 9781491914250. URL `http://www.ieice.org/{~}prmu/jpn/ieice/ieice2017-1.pdf`.

65. B. Pavlyshenko. Machine-Learning Models for Sales Time Series Forecasting. In *Data*, 4(1), p. 15, 2019. doi:10.3390/data4010015.

66. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in {P}ython. In *Journal of Machine Learning Research*, 12, pp. 2825–2830, 2011.

67. F. Petropoulos, S. Makridakis, V. Assimakopoulos, and K. Nikolopoulos. 'Horses for Courses' in demand forecasting. In *European Journal of Operational Research*, 237(1), pp. 152–163, 2014. ISSN 03772217. doi:10.1016/j.ejor.2014.02.036.

68. PricewaterhouseCoopers. *Global Consumer Insights Survey 2018 - From mall to mobile: Adjusting to new consumer habits*. Technical Report, 2018. URL `https://www.pwc.com/gx/en/retail-consumer/assets/consumer-habits-global-consumer-insights-survey.pdf`.

69. C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*, volume 11. MIT Press, Cambridge, MA, 2006. ISBN 026218253X. doi:10.1142/S0129065704001899. URL `papers://8da4c6f7-0d59-4a65-81cc-71a14ebde18a/Paper/p544`.

70. R.M. Rifkin and R.A. Lippert. *Notes on Regularized Least-Squares*. Technical Report, Massachusetts Institute of Technolgy, Cambridge, 2007.

71. J.K.O. Rob J. Hyndman, Anne B. Koehler and R.D. Snyder. *Springer Series in Statistics Forecasting with Exponential Smoothing*. 2008. ISBN 978-3-540-71918-2.

72. F. Rosenblatt. *The Percepton: A Perceicing and Recognizing Automaton*. Technical Report, Cornell Aeronautical Laboratory, Inc., Buffalo, N.Y., 1957.

73. R. Rubinstein, M. Zibulevsky, and M. Elad. Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit. In *CS Technion*, pp. 1–15, 2008. doi:10.1.1.182.9978. URL `http://cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2008/CS/CS-2008-08.revised.pdf`.

74. D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. In *Nature*, 323(14), pp. 533–536, 1986.

75. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd edition. Prentice Hall, apr 2009. ISBN 978-0-13-604259-4.

76. R.E. Schapire and Y. Freund. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Journal of Computer and System Sciences*, 55, pp. 119–139, 1997. ISSN 00344885. doi:10.1088/0034-4885/55/7/004.

77. A.J. Smola and B. Schöllkopf. A tutorial on support vector regression. In *Statistics and Computing*, 14, pp. 199–222, 2004. ISSN 0960-3174. doi:10.1023/B:STCO.0000035301.49549.88.

78. P. Smolensky. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. Technical Report 667, University of Colorado, Boulder, Colorado, 1986. URL `https://pdfs.semanticscholar.org/3ceb/e856001031cfd22438b9f0c2cd6a29136b27.pdf?{_}ga=1.15022902.1038306691.1479690262`.

79. S. Smyl, J. Ranganathan, and A. Pasqua. M4 Forecasting Competition: Introducing a New Hybrid ES-RNN Model. 2018. URL `https://eng.uber.com/m4-forecasting-competition/`.

80. M. Stone. Cross-Validatory Choice and Assessment of Statistical Predictions. In *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2), pp. 111–147, 1974. ISSN 00359246. URL `http://www.jstor.org/stable/2984809`.

81. Sveriges Meteorologiska och Hydrologiska Institut. Ladda ner meteorologiska observationer | SMHI. 2019. URL `https://www.smhi.se/data/meteorologi/ladda-ner-meteorologiska-observationer/`.

82. SVERIGES RIKSBANK. Payment patterns in Sweden 2018. In *Sveriges Riksbank*, (May), pp. 1–10, 2018. URL `https://www.riksbank.se/globalassets/media/statistik/betalningsstatistik/2018/payments-patterns-in-sweden-2018.pdf`.

83. R. Tibshirani. Regression Shrinkage and Selection via the Lasso. In *Proceedings of the American Society of International Law at its annual meeting*, 58(1), pp. 267–288, feb 1996. URL `https://www.cambridge.org/core/product/identifier/S0272503700054525/type/journal{_}article`.

84. J.N. Van Rijn and F. Hutter. Hyperparameter importance across datasets. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2367–2376, 2018. doi:10.1145/3219819.3220058.

85. C. Voyant, G. Notton, S. Kalogirou, M.L. Nivet, C. Paoli, F. Motte, and A. Fouilloy. Machine learning methods for solar radiation forecasting: A review. In *Renewable Energy*, 105, pp. 569–582, 2017. ISSN 18790682. doi:10.1016/j.renene.2016.12.095. URL `http://dx.doi.org/10.1016/j.renene.2016.12.095`.

86. P.R. Winters and F.J. Diaz Borbón. Forecasting Sales by Exponentially Weighted Moving Averages. In *Management Science*, 6(3), pp. 324–342, 1960. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2627346`.

87. P. Yoo, M. Kim, and T. Jan. In Machine Learning Techniques and Use of Event Information for Stock Market Prediction: A Survey and Evaluation, pp. 835–841, 2006. doi:10.1109/cimca.2005.1631572.

88. G. Zhang, B. Eddy Patuwo, and M. Y. Hu. Forecasting with artificial neural networks: The state of the art. In *International Journal of Forecasting*, 14(1), pp. 35–62, 1998. ISSN 01692070. doi:10.1016/S0169-2070(97)00044-7.

89. H. Zou and T. Hastie. Regularization and variable selection via the elastic net. In *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 67(5), p. 768, 2005. ISSN 13697412. doi:10.1111/j.1467-9868.2005.00527.x.