



Discrete optimization

Simplifying tree-based methods for retail sales forecasting with explanatory variables

Arnoud P. Wellens^{a,*}, Robert N. Boute^{a,b,c}, Maximiliano Udenio^a

^a Research Center for Operations Management, KU Leuven, Naamsestraat 69, Box 3555, 3000 Leuven, Belgium

^b Technology and Operations Management Area, Vlerick Business School, Vlamingenstraat 83, 3000 Leuven, Belgium

^c Flanders Make@KU Leuven, Gaston Geenslaan 8, 3001 Leuven, Belgium

ARTICLE INFO

Keywords:

Forecasting
Global forecasting methods
Tree-based methods
Inventory simulation

ABSTRACT

Despite being consistently outperformed by machine learning (ML) in forecasting competitions, simple statistical forecasting techniques remain standard in retail. This is partly because, for all their advantages, these top-performing ML methods are often too complex to implement. We have experimented with various tree-based ML methods and find that a 'simple' implementation of these can (substantially) outperform traditional forecasting methods while being computationally efficient. Our approach is validated with a dataset of 4,523 products of a leading Belgian retailer containing various explanatory variables (e.g., promotions and national events). Using Shapley values and slightly adjusted tree-based methods, we show that superior performance depends on the availability of explanatory variables and additional feature engineering. For robustness, we show that our findings also hold when using the M5 competition dataset. Extensive numerical experimentation finally shows how the forecast superiority of our proposed framework translates to higher service levels, lower inventory costs, and improvements in the bullwhip of orders and inventory. Our framework, with its excellent performance and scalability to practical forecasting settings, we contribute to the growing body of research aimed at facilitating the higher adoption rate of ML among 'traditional' retailers.

1. Introduction

Global retail is currently estimated to be a 25 trillion dollar industry.¹ Their core activity is getting the right product at the right place and time. In short, to efficiently organize the flow of products from supplier to customer is a large supply chain optimization problem. As most retailers face high fixed costs and low-profit margins, small revenue increases can drastically improve their bottom line. This motivates using innovative technologies to improve sales forecasts (Fisher & Raman, 2018). Sales forecasts are also used to support various business decisions such as workforce scheduling, inventory replenishment, and safety stock calculations. Many of these operational decisions happen daily or weekly at the product-store level, where some products are sold in large quantities and others only sporadically. This requires forecasting methods that can accommodate a variety of time series with different levels of variability and intermittency (periods with zero sales) (Petropoulos, Nikolopoulos, Spithourakis, & Assimakopoulos, 2013; Spiliotis, Makridakis, Semenoglou, & Assimakopoulos, 2020). These forecasting methods can also incorporate explanatory variables such as promotions and calendar events to improve their accuracy. In

addition to forecast accuracy, scalability is important. Fildes, Ma, and Kolassa (2019) note that scalability is one of the major forecasting challenges mentioned by retailers — even a small-sized retailer with 2,000 products and 15 stores requires 420,000 unique forecasts per day if the daily sales of each product-store combination for two weeks is of interest. In this paper, we show how a 'simple' decision-tree machine-learning framework performs well for retail forecasting. The motivation for tree-based methods comes from their recent success in retail forecasting competitions (Bojer & Meldgaard, 2021) and their ease-of-use compared to neural networks (Januschowski et al., 2021).

Studies indicate that, today, most retailers still use simple statistical methods like exponential smoothing (Fildes & Petropoulos, 2015; Weller & Crone, 2012). These simple methods are the default in commercial software packages (Fildes, Ma, & Kolassa, 2019), are easy to compute and understand, and have historically been able to achieve similar levels of forecast accuracy as more complex methods (Makridakis, Spiliotis, & Assimakopoulos, 2020). At the same time, several more recent studies and forecasting competitions plead in favor of more complex ML methods, which have recently started to outperform simple statistical methods (Bojer & Meldgaard, 2021; Huber &

* Corresponding author.

E-mail addresses: arnoud.wellens@kuleuven.be (A.P. Wellens), robert.boute@kuleuven.be (R.N. Boute), maxi.udenio@kuleuven.be (M. Udenio).

¹ Retrieved December 21, 2021, from <https://www-statista-com/statistics/443522/global-retail-sales/>.

Stuckenschmidt, 2020; Makridakis, Spiliotis, & Assimakopoulos, 2022; Mukherjee et al., 2018; Salinas, Flunkert, Gasthaus, & Januschowski, 2020; Spiliotis et al., 2020). Yet, many of the suggested ML methods are insufficiently validated regarding benchmarks, availability of explanatory variables, accuracy metrics, and test data. Their model complexity and computational requirements may also prevent ‘traditional’ retailers from putting these ML forecasting techniques into production as most retailers require millions of forecasts to cover each product in each store (Bojer & Meldgaard, 2021; Fildes, Ma, & Kolassa, 2019; Hyndman, 2020; Seaman, 2018; Wellens, Udenio, & Boute, 2021). Besides the computational cost, Petropoulos, Grushka-Cockayne, Siemsen, and Spiliotis (2021) emphasize that complex forecasting methods hinder retailers from making forecasts for all their product offerings or adhering to best practices, such as proper hyperparameter tuning. The forecasts generated by these more sophisticated methods also pose challenges in interpretation and explanation for decision-makers. This lack of transparency may lead to an aversion towards algorithms, even if these methods outperform simpler-to-understand methods (Dietvorst, Simmons, & Massey, 2015).

To understand the trade-off between model complexity and practical applicability, we investigate various tree-based ML methods with different levels of complexity. We compare them to popular statistical methods regarding forecast accuracy, bias, and inventory performance on a rich dataset of a leading Belgian retailer. The dataset includes 4,523 products with different levels of variability and intermittency. It contains various explanatory variables such as daily prices, promotions (including competitor’s promotions), weather forecasts, product hierarchy, national events, and holidays. In addition, we explore the importance of these different data sources using Shapley values, a popular explainable artificial intelligence framework, and various tree-based methods trained on subsets of the data. To ensure our insights are structural and independent of the specific dataset, we also apply our framework to the dataset of the M5 competition.

On our private dataset, We find that a ‘simple’ tree-based ML implementation improves the forecast accuracy of traditional methods² by 11.48% on average, while being computationally efficient. With ‘simple,’ we mean that publicly available open-source software can be used off the shelf without modifying core algorithms or conducting advanced data transformations. These results are robust for multiple products and time series categories over time. More complex versions of tree-based ML only marginally improve forecast accuracy. Using Shapley values and various tree-based methods, we find that the superior performance of our tree-based framework is explained by the availability of explanatory variables and their capability to learn non-linear time series. Despite the ‘simplicity’ of this framework, we note that it is not a plug-and-play method and still requires data science knowledge. Our analysis shows the value – and need – of additional (manual) feature engineering. Finally, we show, using an inventory simulation, how the improved forecast performance translates into higher service levels, lower inventory levels, and improvements in the bullwhip of orders and inventory.

The contribution of our paper is threefold. First, we show that a simple, off-the-shelf tree-based method performs exceptionally well. For most retailers, the gains of more sophisticated tree-based methods may not be worth the increased model complexity and computational requirements. Second, we describe the conditions under which our method outperforms popular statistical methods, offering managers the tools to identify promising implementation opportunities quickly. Third, we investigate how these improved sales forecasts can benefit the daily replenishment of a retailer. In short, our findings provide evidence that a simple tree-based method is competitive for retail forecasting to improve their forecast accuracy and replenishment when sufficient data is available.

2. Literature review

There is ample evidence that retailers use managerial judgment and simple statistical methods to forecast demand (Fildes & Petropoulos, 2015; McCarthy, Davis, Golcic, & Mentzer, 2006). A decade ago, Weller and Crone (2012) surveyed 200 demand planning experts and found that exponential smoothing, moving average, and the naive method account for 82.1% of the statistical forecasting methods used in practice. Only 13.5% of the respondents made use of advanced time series models such as econometric models (6.9%), autoregressive integrated moving average (ARIMA; 3.5%), and neural networks (NNs; 1.5%). Once estimated, forecasts are often adjusted with experts’ knowledge to incorporate the effect of explanatory variables such as promotions and holidays. The effectiveness of such manual adjustments is questionable. Multiple studies have shown that the forecast accuracy is often higher when these manual adjustments are limited (Fildes, Goodwin, Lawrence, & Nikolopoulos, 2009; Fildes, Goodwin, & Önköl, 2019).

The dominance of simple statistical methods in practice should not surprise. Historically, these methods achieved similar levels of forecast accuracy as more complex methods (Makridakis et al., 2020). This observation was confirmed by multiple forecasting competitions (Makridakis et al., 1982, 1993; Makridakis & Hibon, 2000). For instance, the winning method of the M3 forecasting competition combined linear regression and simple exponential smoothing with some minor tweaks (Assimakopoulos & Nikolopoulos, 2000). Although simple by design, it outperformed the more sophisticated methods in the M3 as well as every method in the NN3 competition, which was organized nearly a decade later to promote the use of ML for forecasting (Crone, Hibon, & Nikolopoulos, 2011; Hyndman, 2020). In fact, until 2015, most retail forecasting competitions on Kaggle,³ the world’s largest data science community, were won by relatively simple statistical methods (Bojer & Meldgaard, 2021). As a result, the superiority of ML in the field of forecasting has been questioned until very recently (Fildes, Ma, & Kolassa, 2019; Makridakis, Spiliotis, & Assimakopoulos, 2018; Makridakis et al., 2020).

More recent studies and forecasting competitions, however, plead in favor of ML methods (Huber & Stuckenschmidt, 2020; Makridakis et al., 2022; Mukherjee et al., 2018; Salinas et al., 2020; Spiliotis et al., 2020). Since 2015, all major large-scale retail forecasting competitions on Kaggle have been dominated by ML (Bojer & Meldgaard, 2021). It started with the Rossmann Store Sales competition, which was won by a tree-based ML method, more specifically, an ensemble of 12 Extreme Gradient Boosting (XGBoosting) methods. Two years later, similar results were found at the Corporación Favorita Grocery Sales Forecasting competition, where only ML methods were among the top-performing methods. This time, the winner used an ensemble of NNs and Light Gradient Boosting Machine (LightGBM) methods, which is an improved implementation of XGBoost (Ke et al., 2017). Finally, in 2021, the M5 Accuracy competition was dominated by tree-based ML methods; all but one of the top 50 performing methods used LightGBM (the third place was taken by a method solely based on NNs) (Makridakis et al., 2022). The M5 was set up to estimate the sales forecasts at different hierarchical levels for 3,049 products in ten Walmart stores. The time series were hierarchically organized, starting at the product-store level and aggregated per department, category, store, and other combinations. The winning method estimated every sales forecast with an ensemble of recursive and non-recursive LightGBM methods. (A recursive method uses its prior forecasts as inputs to make predictions for later timestamps.) Interestingly, while the top-performing methods outperformed the benchmarks by more than 20% on average, the outperformance deteriorated completely at the product-store level. This

² With traditional forecasting methods we mean commonly used forecasting methods in practice by retailers such as exponential smoothing and moving average.

³ <https://www.kaggle.com/>

raises the question of whether these ML methods can outperform the benchmarks at the more granular levels.

The recent victories of ML in sales forecasting are largely driven by innovations in ML (Benidis et al., 2020; Bojer & Meldgaard, 2021). Examples are embedded layers in NNs, which appeared in 2016 (Guo & Berkahhah, 2016), and the wider adoption of the Long-short term memory (LSTM) cell. Decision-tree methods have also become more advanced since the launch of XGBoost in 2014⁴ and LightGBM in 2016.⁵ Another major innovation is the increased use of *global* forecasting methods. Global methods estimate model parameters by using multiple time series simultaneously. This is in sharp contrast to most traditional forecasting methods. These methods are *local* as these build one method per time series and do not share any parameters (Januschowski et al., 2020). Global methods can easily exploit cross-series information by learning across different time series. This is especially valuable when time series are related (Bandara, Bergmeir, & Smyl, 2020; Makridakis et al., 2022; Smyl, 2020). As such, global methods have access to more training data and can be more complex (e.g., regarding the total amount of inputs and learning non-linear time series patterns) than their local counterpart, yet without necessarily overfitting the training data (Montero-Manso & Hyndman, 2021).

Global tree-based methods and NNs have demonstrated state-of-the-art performance in recent retail forecasting competitions. However, decision trees can achieve similar results as NNs using simpler model architectures (i.e., without modifying the core algorithms or software) and without extensive data preprocessing (Januschowski et al., 2021). This renders them more amenable to practical application. The winner of M5, an undergrad student with little knowledge and experience in forecasting, demonstrates the ease of use of tree-based methods (Makridakis et al., 2022). Shwartz-Ziv and Armon (2021) compared tree-based methods with varying deep NN methods on different tabular datasets. They showed that tree-based methods consistently outperform NNs but require less tuning. We acknowledge that not all NNs have sophisticated model architectures. NNs with more straightforward architectures exist but still need extensive pre- and postprocessing of the input data, and they do not necessarily outperform the simpler statistical forecasting methods (Hewamalage, Bergmeir, & Bandara, 2021; Spiliotis et al., 2020). The extensive data processing entails data transformations, such as deseasonalizing the time series, stabilizing the variance, and normalizing the mean and trend (Hewamalage et al., 2021). Although simple to understand theoretically, it substantially complicates the programming code (we refer to Benidis et al. (2020) for an extensive review on large-scale forecasting with NNs). Note that some of these data transformations can also benefit tree-based methods (especially in the case of training global forecasting methods with heterogeneous time series); however, it is not a strict requirement for trees (Makridakis et al., 2022; Montero-Manso & Hyndman, 2021).

Despite the incredible performance of these top-performing ML methods, they are rarely used in practice (Fildes, Ma, & Kolassa, 2019). Multiple authors argue that large-scale adoption is partly hampered due to its insufficient validation and difficult implementation. ML forecasting papers and competitions have been criticized for including too few informative explanatory variables, overfitting the test data, or not using proper benchmarks and accuracy metrics (Bojer & Meldgaard, 2021; Hyndman, 2020). This lack of proper validation is also seen as a major issue by retailers (Fildes, Ma, & Kolassa, 2019). Besides, we believe that the practical applicability of the proposed methods is often neglected as the goal of forecasting competitions (and many papers) is to achieve the highest possible level of forecast accuracy regardless of their implementation complexity. For example, the M4 winning method is said to be too complicated to implement in terms of costs and effort as it combines statistical features with multiple blocks

of LSTMs (Gilliland, 2020; Makridakis et al., 2022). The M5 winning method uses an ensemble of pure LightGBM methods. While the code of each LightGBM method is straightforward to model, the ensemble consists of LightGBM methods with different pooling strategies, recursive and non-recursive inputs, and advanced feature engineering and feature selection procedures. This complicates the codebase and the computational requirements to train each global LightGBM method in practical retail settings (Wellens et al., 2021).

We contribute to the literature by showing that a straightforward tree-based method with explanatory variables and basic feature engineering suffices to outperform popular benchmarks. We illustrate how the model complexity and computational requirements of top-performing tree-based methods, such as the M5 winning method, can be reduced with a minimal negative impact on forecast accuracy. Our approach achieves this by simplifying common complexities observed in the M5 winning methods. This includes transforming recursive inputs into non-recursive ones; focusing on a single forecasting method instead of ensembles; training a single forecasting method over all time series (compared to, first, creating multiple pools of similar time series, and, second, training a forecasting method per pool); and, lastly, by only focusing on ‘basic’ feature engineering — omitting feature selection procedures and the adding of complex inputs. In the next section, we describe our decision-tree framework for retail forecasting and validate it in Sections 4 and 5.

3. Our decision-tree framework for retail forecasting

We describe our decision-tree framework (DTF) for retail sales forecasting at the product-store level. Given the higher complexity of NNs and our goal to simplify tree-based ML methods for retail applications, we focus on the ‘simpler-to-use’ decision-tree methods. More specifically, we use gradient boosting decision trees (GBDTs) currently dominating Kaggle. Implementing a tree-based ML method requires three phases: (1) an experimental phase to decide on the GBDT method’s inputs and hyperparameters, (2) a training phase to train these methods on the most recent data, and (3) a prediction phase that uses the trained methods to produce sales forecasts (Januschowski et al., 2020). Most top-performing ML methods require sophisticated optimization steps during these three phases. Examples are data transformations (e.g., normalizing the average unit sales), feature selection (e.g., recursive feature elimination (May, Dandy, & Maier, 2011)), optimizing forecast ensembles with meta-learning (Ma & Fildes, 2021), modifying the core ML algorithms to improve performance (Januschowski et al., 2021), and more. These steps require an advanced level of ML knowledge and often extensive computation. The simplification of our framework is motivated by various papers that have shown that a suboptimal selection of forecasting methods has a negligible effect on the out-of-sample forecast accuracy (Nikolopoulos & Petropoulos, 2018; Petropoulos et al., 2021). In what follows, we describe each phase of our ‘simplified’ framework that can still obtain accurate forecasts. Fig. 1 visualizes its key elements.

(1) Experimental phase: The experimental phase aims to find the best-performing inputs and hyperparameters for our tree-based method. (a) First, you must prepare the input data for your methods. (b) Second, you need a process to select the best-performing method. (c) Third, the hyperparameters of the selected methods need to be optimized.

(a) Input data: As mentioned before, GBDT methods do not require sophisticated data transformations such as NNs. GBDT methods can handle time series with different levels of variability or mean without any additional processing (Makridakis et al., 2022). Given that these data transformation steps often require extensive computation time and expert knowledge, our simplified framework omits sophisticated data transformations. This also makes it more scalable. Moreover, most GBDT implementations can handle input data with missing values, categorical variables, and even variables with text (strings). This reduces

⁴ <https://github.com/dmlc/xgboost/releases>

⁵ <https://github.com/microsoft/LightGBM/releases>

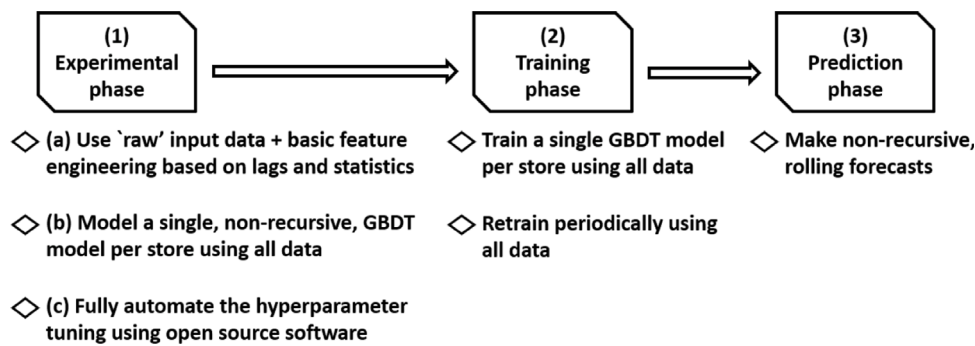


Fig. 1. Summary of our decision-tree framework. Our framework simplifies the experimental, training, and prediction phases of a tree-based method to make it scalable while still obtaining accurate forecasts.

the need for even simpler preprocessing operations. We only require some basic cleaning, such as looking for errors in the data.

We extract all available data relevant to the forecasting task from the data warehouse, such as the historical sales data, the sales timestamps, and the available explanatory variables such as price and promo data. We call this available data the 'raw' data. Creating additional inputs based on the 'raw' data, known as feature engineering, typically improves the learning of ML methods. Therefore, we manually fit basic operations on the available data to create new inputs. Examples of these 'basic' operations are lagged variables (e.g., lags of promotions and holidays) and basic statistical operations on the time series data and explanatory variables (e.g., the mean of the sales and price, the maximum price per product, etc.). Once the inputs have been created, using only a subset of these inputs is a best practice. Known as feature selection, this removes redundant inputs from the method to reduce the dimensionality of the forecasting problem, which improves learning (May et al., 2011). However, we suggest using all the available 'raw' inputs as-is and the inputs with some basic manual engineering. The total number of inputs is limited as we only select the 'raw' inputs and the inputs with some basic manual engineering. Due to this limited amount, we avoid the need for any sophisticated feature selection procedure because decision trees naturally ignore (a reasonably small amount of) noisy inputs. Remark that most top-performing ML methods improve the feature engineering step by using automatic feature engineering packages such as *tsfresh*,⁶ *featuretools*,⁷ and *tsflex*,⁸ or by extensively searching (manually) for more sophisticated inputs such as computing price elasticities. While these techniques typically improve forecast accuracy, they also increase the model complexity and computational requirements considerably. In addition, these techniques may produce hundreds and even thousands of potentially relevant inputs. This large amount of inputs, in turn, needs to be reduced through sophisticated feature selection procedures that, again, increase complexity and computational requirements.

(b) Model selection: Our DTF uses an 'off-the-shelf,' non-recursive, single, global GBDT method on all available data per store. By 'off-the-shelf,' we mean that no changes to the original software code are made to improve the forecast accuracy further. Our recommendation is to employ non-recursive forecasting methods. While we acknowledge that recursive GBDT methods can be more accurate (Makridakis et al., 2022), especially for long-term forecasting, they come with increased complexity in implementation. These methods are not prebuilt in most forecasting packages, and their computation time is higher due to the need to recompute recursive inputs after each forecast. Therefore, we highlight that this study primarily focuses on short-term forecasting. We omit ensembles that include multiple forecasting

methods. Although known for improving forecast accuracy, ensembles increase computation time and typically require additional strategies to weight the forecasts of the ensemble (Ma & Fildes, 2021). Our non-recursive GBDT method can be implemented easily using 'off-the-shelf' software without a deep understanding of the algorithm. Lastly, our model architecture consists of a single global method per store. Multiple authors (Bandara et al., 2020; Montero-Manso & Hyndman, 2021) show that the forecast accuracy of global forecasting methods can be improved by grouping the set of time series into smaller groups. This results in training multiple global methods on subsets of the time series. As grouping time series is a non-trivial optimization problem, we suggest to simply pool all the time series per store.

(c) Hyperparameter tuning: We implement ready-to-use open-source packages that work with most ML methods to find the best-performing set of hyperparameter values. These packages are typically computationally efficient and fully automate the hyperparameter tuning. Consequently, these require little expertise on the subject of hyperparameters.

We suggest to only optimize the hyperparameters that largely impact the learning capabilities of GBDT methods such as the number of trees, the maximum number of leaves, the minimum amount of data per leaf, the learning rate and L1 and L2 regularization. Note that many of these hyperparameters overlap in their quest to reduce the tree complexity and the chance of overfitting. For example, L1 and L2 regularization are computed on the leaf scores to reduce the depth of the tree; this is also controlled by the minimum amount of data per leaf. Therefore, one can opt to optimize fewer hyperparameters to reduce computational requirements. To further reduce computational requirements, we suggest to implement packages based on more advanced optimization techniques such as Bayesian optimization instead of random search or grid search. Finally, we suggest to limit the total amount of iterations of retraining the methods with different sets of hyperparameters as finding the optimal hyperparameters only slightly impacts the forecast accuracy (Nikolopoulos & Petropoulos, 2018; Petropoulos et al., 2021).

(2) Training phase: The training phase trains the methods identified during the experimental phase on the most recent data. The computational requirements for training the determined GBDT methods are low as no ensembles are used, and only one global method per store must be trained. Note that ML methods must be retrained when new data or products come in. However, this can happen only sporadically as Huber and Stuckenschmidt (2020) show during a five-month-test period that retraining ML methods only improves forecast accuracy by 1% to 3%.

(3) Prediction phase: The prediction phase uses the trained methods obtained in the training phase to produce sales forecasts. Once the non-recursive GBDT methods are trained, they estimate the sales forecasts for the required forecast horizon. While training ML methods can be time-consuming, the inference is typically fast (Januschowski et al., 2020). Especially in this case, as we do not use any ensembles, meta-learners, or recursive methods. Note that we do not include any postprocessing either. Postprocessing is often necessary when the input data has been normalized or deseasonalized.

⁶ <https://tsfresh.readthedocs.io/en/latest/>

⁷ <https://www.featuretools.com/>

⁸ <https://github.com/predict-idlab/tsflex>

4. Experimental setup

In this section, we describe the experimental setup used to validate our DTF. We explore the dataset and describe how we apply our DTF. Next, we explore various tree-based methods with varying levels of complexity. This allows us to understand better the impact of simplifying these tree-based methods. We then give an overview of our performance measures and benchmarks. Section 5 discusses the results.

4.1. Dataset

We validate our DTF on a private dataset of a large Belgian retailer. The dataset includes daily sales of 4,523 unique products from 09/01/2017 to 10/22/2020 (1,148 days) in one large supermarket in Belgium and is enriched by explanatory variables from various data sources. The retailer sells food products, including perishable products (e.g., strawberries and tomatoes) and products with longer shelf life (e.g., ice cream and frozen pizzas). Products can be bought in-store or online. In the latter case, the basket with products is collected by the retailer for pick-up in the store. The entire dataset includes data from eight different data sources. Table 1 summarizes the data and their source.

The first two sources contain point-of-sales data (unit sales and timestamp). The first data source includes the daily sales and the daily sales lags, indicated as `sales_lag_1`, `sales_lag_2`, `sales_lag_3`, and so on. The second data source includes information about the timestamp of each daily sale. This includes `day_of_the_month`, `week_of_the_year`, `month`, and `year`. The point-of-sales data differs in terms of variability and intermittency. Some products are sold sporadically (e.g., expensive wines), while others experience volatile sales patterns due to promotions or seasonality. We use the formulation of Syntetos, Boylan, and Croston (2005) to categorize the time series of the sales data in four different groups according to the squared coefficient of variation of the units sold (CV^2) and their average inter-demand interval (ADI), which is the average amount of days between registered sales (see Fig. 2). Smooth time series have low levels of variance ($CV^2 < 0.5$) and intermittency ($ADI < 4/3$ days). These represent the majority of our dataset (46.01%). The second largest category of our time series are erratic (18.22%). These are characterized by higher levels of variability but with a similar level of intermittency. The third group includes the intermittent time series (18.09%), which have little variance, but a high level of intermittency. Finally, 17.68% of the products can be categorized as lumpy. These time series have high levels of variability and intermittency.

The six additional data sources contain data on explanatory variables. The third data source describes the hierarchical structure of each product. Each product is hierarchically structured in five different levels of aggregation (`productgroup_1_1`, `productgroup_1_2`, `productgroup_1_3`, `productgroup_1_4`, and `productgroup_1_5`) by the retailer using managerial judgment and is based on product characteristics (e.g., is the product drinkable or eatable, is it a vegetable, etc.). For example, a six-pack of beer bottles can be hierarchically structured as a food/drink product (level one), a beverage (level two), an alcoholic beverage (level three), beer (level four), and beer sold in glass (level five). The first level consists of three categories, the second level of eight categories, the third level has 55 different categories, the fourth level has 246 categories, and the most disaggregated level consists of 1,135 categories. Although this hierarchical structure was originally set up for various business operations, learning similar time series patterns across related products proved useful. The fourth data source describes two sets of holiday information. The first set includes national public holidays like Christmas and Easter. These periods are typically linked to, e.g., a closure of the store or family reunions and can be useful to identify temporary changes in the time series patterns. The input `public_holiday` includes this information and the date of each public holiday. The

second set includes data about school holidays that may cause shifts in demand. The input `school_holiday` includes the name and date of each school holiday.⁹ The dataset also includes two inputs called `public_holiday_dummy` and `school_holiday_dummy` that indicate whether it is a holiday (or not) without specifying the name of the holiday. The fifth data source includes weather forecasts (`weather_type_forecast` and `temperature_forecast`). These forecasts include the daily temperature and the type of weather (such as sunny or cloudy) in Belgium. We distinguish between 14 different types of weather. This data is based on forecasts with a two-week forecast horizon. As a result, even when forecasting the next day, we use weather forecasts that are made two weeks ago. The sixth data source describes the price information. We denote `daily_price` as the daily price of a product if it is bought physically in-store, and `daily_price_pickup` when ordered online. For discounted products bought in bulk, we have `daily_price_large_quantity` and `daily_price_large_quantity_pickup`. These prices are always lower or equal to `daily_price(_pickup)` and only apply if a certain volume is bought. If no volume discounts apply, `daily_price(_pickup)` is set equal to `daily_price_large_quantity(_pickup)`. The seventh data source includes promotional information. It describes whether there is a promotion (`promo_dummy`), the type of a promotion (18 different types are identified) (`promo_type`), the discount percentage (`promo_depth`), and whether the price discount is due to a nearby competitor with lower prices (`price_reaction`, `price_reaction_pickup`). Finally, we have access to a dataset of national events. This includes events such as Tournée minérale, a national campaign to reduce alcohol consumption during February, and other events such as festivals. The national events are split over four different inputs as sometimes multiple events happen at the same time (`events_1`, `events_2`, `events_3`, `events_4`).

4.2. The application of our decision-tree framework

We make out-of-sample daily sales forecasts at the product-store level with a forecast horizon of seven days. We use LightGBM as our core algorithm due to its success in the M5 Accuracy competition. To apply this algorithm without overfitting the data, we divide the dataset into a training set, a validation set, and a test set. During the *experimental* phase, we train the methods only on the training set, which includes the first two years (09/01/2017 to 07/15/2019) of the dataset. The third year (07/16/2019 to 07/15/2020) is the validation set required for the hyperparameter tuning. We train the methods for the *training and prediction* phase using the training and the validation set together (09/01/2017 to 07/15/2020). Finally, we use the final three months of the data (07/16/2020 to 10/22/2020) to evaluate our (trained) methods against the benchmarks (this is the test set).¹⁰ We review the experimental, training, and prediction phases in the next subsections.

4.2.1. Experimental phase

Removing erroneous data and feature engineering: LightGBM can handle missing values, categorical variables, different levels of variability and mean, and even inputs with text (strings). Therefore, the data cleaning phase is limited to corrections such as negative sales or prices.

We extract all available data (see Table 1) directly from the data warehouse. To improve the LightGBM method's learning capabilities,

⁹ Note that the effect of school holidays or public holidays may also be implicitly captured by using seasonality. This is, however, not the case for all holidays as some may not have a fixed date (e.g., Easter) (Huber & Stuckenschmidt, 2020).

¹⁰ Note that we do not make adjust the dataset to structural breaks such as the COVID-19 pandemic, which could not be foreseen.

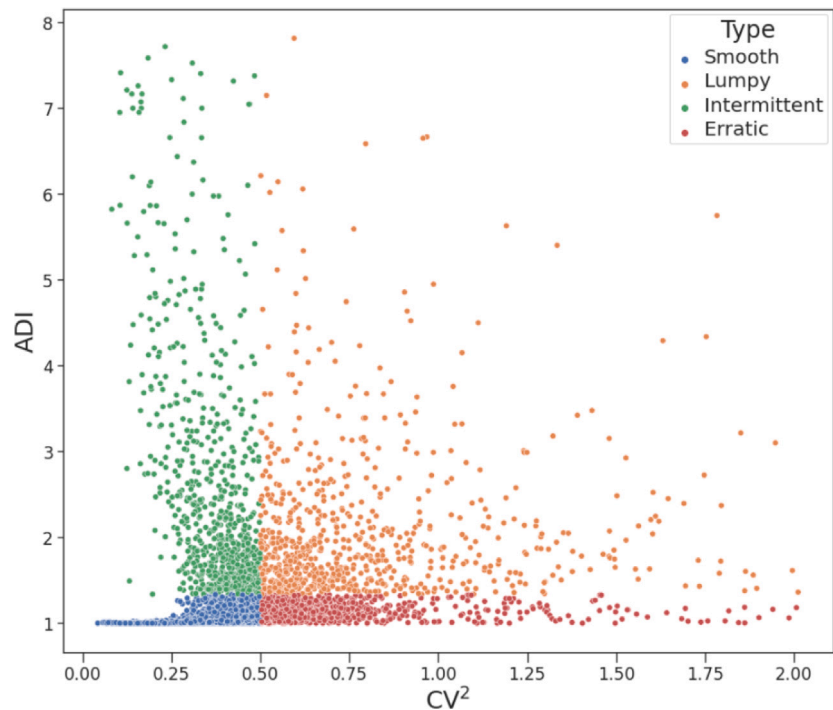


Fig. 2. Classification of the sales data of 4,523 products based on their intermittency (ADI) in days and variability (CV^2). The dataset includes 818 intermittent, 800 lumpy, 824 erratic, and 2,081 smooth time series.

Table 1
Overview of the ‘raw’ data in the dataset.

Data source	‘Raw’ data
Sales	sales_lag_1, sales_lag_2, sales_lag_3, sales_lag_4, sales_lag_5, sales_lag_6, sales_lag_7, sales_lag_8, sales_lag_9, sales_lag_10, sales_lag_11, sales_lag_12, sales_lag_13, sales_lag_14
Date	day_of_the_month, week_of_the_year, month, year
Hierarchy	productgroup_l_1, productgroup_l_2, productgroup_l_3, productgroup_l_4, productgroup_l_5
Holidays	public_holiday, school_holiday, public_holiday_dummy, school_holiday_dummy
Weather	weather_type_forecast, temperature_forecast
Price	daily_price, daily_price_large_quantity, daily_price_pickup, daily_price_large_quantity_pickup
Promo	promo_dummy, promo_depth, promo_type, price_reaction, price_reaction_pickup
Events	events_1, events_2, events_3, events_4

we add inputs by performing basic operations on the available datas, such as lags and basic statistical operations. The engineered inputs are all based on the ‘raw’ inputs and rely on simple calculations. They do not require advanced Python or sophisticated domain knowledge. We describe them in [Appendix A](#). The ‘raw’ inputs, combined with the engineered inputs, provide a set of 104 inputs per time series.

Input data and model selection: We use all available inputs of [Tables 1](#) and [A.6](#) to train our LightGBM method. We do not use any feature selection procedure, as decision trees can handle noisy inputs. As discussed, not considering any sophisticated feature engineering or feature selection step enhances the ‘simplicity’ of our proposed method. We implement an ‘off-the-shelf,’ non-recursive, single, global LightGBM method for our model architecture. Remark that this is easily done

using the LightGBM package.¹¹ We use all available training data in one store to train this single method.

Hyperparameter tuning: We apply a publicly available, ready-to-use hyperparameter optimization framework called Optuna.¹² This framework sequentially samples sets of hyperparameters that potentially improve the objective function (validation error) by using the history of previously evaluated hyperparameter sets ([Akiba, Sano, Yanase, Ohta, & Koyama, 2019](#)). We use Optuna’s default sampler, tree-structured Parzen estimator ([Bergstra, Bardenet, Bengio, & Kégl, 2011](#)). We limit the total amount of iterations to 21 as the improvement

¹¹ <https://lightgbm.readthedocs.io/en/latest/>
¹² <https://optuna.org/>

per iteration drastically slowed down close to the 20th iteration. We use a three-fold time series cross-validation strategy on the validation set, which is implemented using scikit-learn.¹³ In this particular cross-validation approach, random data shuffling is not employed. Instead, it ensures that the test indices are consistently greater than the indices used for training the method to prevent data leakage. The best-performing hyperparameters are then used in the training phase, and to evaluate the forecasting method on the test set in the prediction phase. The final hyperparameters are shown in the Supplementary material, Section 1.

4.2.2. Training and prediction phase

After the experimental phase, we train the determined global LightGBM method with the best-performing hyperparameters on the training dataset and the validation set. This gives us 1,049 days of training data per time series. Then, for the prediction phase, we generate rolling forecasts for each day in the three-month-long test set. For each day, we make forecasts for the next seven days. As we retrain the global LightGBM method monthly with the most recent data, we repeat the experimental phase every month and add the test data of the previous month to the training and validation set. We train and optimize each method three times with three different seed values to cope with the stochasticity of training GBDT methods.

4.2.3. Shapley values

To understand how our DTF interacts with its inputs, we make use of Shapley values. ML, in general, is typically hard (if not impossible) to interpret and is therefore referred to as a black box. A recent popularized method to explain such a black box is SHAP (SHapley Additive exPlanations), developed by Lundberg and Lee (2017). The idea of SHAP is based on the work of Shapley (1953), which describes a method used in cooperative game theory to distribute the total surplus fairly to each player. These contributions are called Shapley values. In the case of forecasting, the total surplus can be seen as the prediction made by the black box ML algorithm, and the players are defined as different inputs. Thus, given a black box and a set of data points, SHAP estimates how each input impacts the prediction of a black box model. The Shapley values are defined as the average marginal contribution of an input across all possible combinations of the inputs and are estimated for each prediction and each input. As such, the Shapley values can be used to understand how a certain input influences the final prediction. When aggregating the average absolute value of each Shapley value over all forecasts per input, each input's global impact/importance can be estimated. As such, the Shapley values help us understand what drives the good performance of the DTF. We apply the TreeExplainer from the package SHAP¹⁴ on the DTF method that is trained on the training and the validation set.

4.3. Decision-tree framework variations

We analyzed various trees with varying levels of complexity to investigate how more complex (e.g., an ensemble method) and simpler (e.g., the DTF with fewer inputs) versions influence prediction performance. We start by explaining the more sophisticated versions first.

- **Recursive DTF (RDTF):** We implement a recursive version of the DTF, which we call the RDTF. RDTF is similar to the DTF but includes four additional recursive inputs. These inputs are rolling sales averages of the last 7, 14, 30, and 60 days. These are strictly different from the rolling mean inputs of the DTF, as we have lagged these by the forecast horizon.

- **Ensemble:** We implement a simple ensemble that takes the average of the DTF and the RDTF forecasts.
- **DTF-l-1 and DTF-l-2:** We implement our DTF with different pooling strategies. Instead of training one method per store, we train one per subgroup of products per store. More specifically, we focus on `productgroup_1_1` containing three unique categories, and `productgroup_1_2` with eight different categories. We call these methods DTF-l-1 and DTF-l-2, respectively. This means that DTF-l-1 consists of three LightGBM methods and DTF-l-2 of eight LightGBM methods, each trained on a separate dataset. Note that we optimize the hyperparameters only once for DTF-l-1 and once for DTF-l-2. We pick one random group of time series to optimize the hyperparameters and reuse these hyperparameters for the other groups to reduce the computational requirements.
- **DTF-sl-70:** We implement our DTF with additional sales lags, from `sales_lag_1` up to `sales_lag_70`.
- **DTF-fs-sl-70:** We also implement the DTF-sl-70 using a proper feature selection procedure based on Rinderknecht and Klopfenstein (2021), which we denote as DTF-fs-sl-70. This approach uses Shapley values to determine the most important inputs to make the forecasts. To apply this feature selection procedure, we first optimize the hyperparameters of the LightGBM method using all inputs. In the second step, we compute the average Shapley value of each input on a random subset of the training data with a maximum of 5,000 data samples to restrict the computational requirements. This returns the average marginal contribution of each input. In line with Rinderknecht and Klopfenstein (2021), we only keep the inputs that explain 95% of the forecasts and drop the rest. Finally, we retrain the method using this subset of inputs.
- **DTF-m5:** The most sophisticated tree-based method we test is the M5 winning method. We replicate this method by creating an ensemble of recursive and non-recursive methods, methods with different pooling strategies, and a more advanced feature engineering and feature selection procedure. More specifically, we train our DTF, DTF-l-1, and DTF-l-2 with additional inputs and the automatic feature selection procedure based on Shapley values. We also make a recursive version of each. The additional inputs include 70 sales lags and two more sophisticated engineered inputs. These two engineered inputs respectively measure for each product the number of products sold at a given price and the number of days that the product was offered at that price. We call this ensemble the DTF-m5. It consists of 24 different LightGBM methods.¹⁵

Next to the more sophisticated versions of our DTF, we implement five LightGBM methods that are simpler by using fewer inputs.

- **pos-nfe-DTF:** We implement the pos-nfe-DTF, which relies only on point-of-sales data (pos) and uses no feature engineering (nfe). The pos-nfe-DTF only includes sales lags and time-related inputs that are based on the 'raw' data related to the date (see first two rows (sales and date) of Table 1).
- **pos-fe-DTF:** We implement the pos-fe-DTF which relies on point-of-sales data but makes use of feature engineering (fe). As a result, it uses all inputs related to the first two rows of Tables 1 and A.6.
- **nfe-DTF:** The nfe-DTF uses all data sources (like our implementation of the DTF). However, it does not use any feature engineering. Hence, it uses all the inputs of Table 1, but none of Table A.6.

¹³ https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

¹⁴ https://shap.lrjball.readthedocs.io/en/docs_update/generated/shap.TreeExplainer.html

¹⁵ The DTF-m5 consists of two LightGBM methods that are trained on the entire dataset, six methods per `productgroup_1_1`, and 16 per `productgroup_1_2`. Half of these are recursive, and the other half non-recursive.

Table 2
Overview of tree-based methods with varying levels of complexity.

Method	Overview
DTF	Decision-tree framework
RDTF	Recursive decision-tree framework
Ensemble	Ensemble of the DTF and the RDTF
DTF-l-1	DTF trained per productgroup_1_1
DTF-l-2	DTF trained per productgroup_1_2
DTF-sl-70	DTF with additional sales lags
DTF-fs-sl-70	DTF with feature selection and with additional sales lags
DTF-m5	Large ensemble of multiple versions of our DTF
pos-nfe-DTF	DTF with point-of-sales data and no feature engineering
pos-fe-DTF	DTF with point-of-sales data
nfe-DTF	DTF with no feature engineering
pp-DTF	pos-fe-DTF with price and promo data
npp-DTF	DTF without price and promo data

- **pp-DTF:** The pp-DTF is based on the pos-fe-DTF, but it also includes the inputs related to the price and the promotions (pp) of the products. Thus, the pp-DTF uses all available data besides the data sources related to the hierarchy, holidays, weather forecasts, and events. This enables the analysis of the importance of subsets of the available data.
- **npp-DTF:** The npp-DTF uses all data sources except the data related to price and promo (npp).

See Table 2 for an overview of all the tree-based variations.

4.4. Benchmarks

We benchmark the forecast accuracy against nine forecasting methods commonly used in practice. The first seven methods are based on the benchmarks of the M5 Accuracy competitions (Makridakis et al., 2022). Their code is publicly available on Github.¹⁶ We also benchmark against two Prophet-based benchmarks due to their recent rise in popularity. In our study, we do not benchmark against NNs. The main reason is that most plain-vanilla NN methods are not competitive with the other benchmarks. In the M5 Accuracy competition, for example, the NN benchmarks were one of the worst. More sophisticated NNs architectures with sufficient data transformations can perform extremely well. However, given our focus on simple-to-use tree-based methods, we believe that adding plain-vanilla NN methods will not add much value.

1. **Naive:** The naive method is the simplest of all forecasting techniques. This method sets all forecasts equal to the value of the last observed period. Although inherently naive, according to a study of Morlidge (2014), 52% of the forecasts produced by eight different consumer and industrial companies do not even outperform this approach.
2. **Seasonal Naive (sNaive):** The sNaive method adjusts Naive by taking seasonality into account. It uses the last observed value of the same period. When estimating daily sales forecasts with a frequency set to seven, the method uses the last observed value of the same weekday to forecast daily sales.
3. **Moving Averages (MA):** Moving averages (MA) is another commonly used method in practice (Syntetos & Boylan, 2005) that computes its forecasts by taking the average of the last k observations. This k is optimized by using an in-sample MSE.
4. **Exponential Smoothing (ES):** One of the most popular forecasting methods in practice and the best-performing benchmark at the M5 is exponential smoothing (ES). As different variations exist, we use the smooth package in R to select the best-performing ES method per time series automatically.¹⁷

5. **Croston's method (CRO):** Croston's method (Croston, 1972) is a popular technique to forecast intermittent sales patterns. It separates the non-zero demand periods, z_t , from those without sales, p_t . Using simple exponential smoothing (this is the simplest version of ES as it does not include any trend or seasonality), it forecasts \hat{z}_t and \hat{p}_t separately. Afterwards, it divides \hat{z}_t by \hat{p}_t to compute the final forecast.
6. **Aggregate-Disaggregate Intermittent Demand Approach (ADIDA):** Nikolopoulos, Syntetos, Boylan, Petropoulos, and Assimakopoulos (2011) propose another approach to forecasting intermittent time series by using temporal aggregation. Aggregating a high-frequency time series to a lower frequency (e.g., aggregating daily sales to weekly sales) reduces intermittency and variance. We then apply simple exponential smoothing on the aggregated time series to produce the forecasts.
7. **Exponential Smoothing with explanatory variables (ESX):** ESX includes explanatory variables to improve the forecast accuracy of ES further. Note that local statistical methods are typically incapable of dealing with high-dimensional input space. Therefore, we use a greedy forward feature selection procedure. First, based on domain knowledge, we manually rank all available inputs from most to least valuable. Second, we add the most valuable input from the list to each local method. Third, we check for highly correlated inputs before training the ESX per time series. If the correlation is higher than 0.8, we remove this input for this single time series. Fourth, if the additional input improves the average forecast accuracy over all time series on the test set, we add the input to the method and go to the next input in the list. Once the forecast accuracy declines, we remove the last input and stop training. In our dataset, the final ESX method includes promo_dummy, promo_depth, price_reaction, and daily_price_large_quantity.
8. **Prophet:** Some retailers, e.g., Target in the US, have been experimenting with generalized additive models as these are simple to use, explainable, and achieve good levels of forecast accuracy on a wide variety of time series (Yelland, Baz, & Serafini, 2019). Prophet, developed by Facebook, is a popular method to implement generalized additive models (Taylor & Letham, 2018). It is a decomposable forecasting method that consists of three major components: trend, seasonality, and explanatory variables. We implement this method using the default parameters.
9. **Prophet with explanatory variables (Proph.X):** We further improve the forecast accuracy of Prophet by including explanatory variables (Proph.X). Therefore, we use a similar feature selection procedure as in the case of ESX. Given our dataset, the final method includes promo_dummy, promo_depth, promo_type, price_reaction, daily_price_large_quantity, daily_price, public_holiday_dummy, school_holiday_dummy, public_holiday, school_holiday, events_1, events_2, events_3, events_4.

Remark that, in contrast to most tree-based methods, the benchmarks require that the data contains no missing values. We, therefore, interpolate the sales data on public holidays when the store is closed. Moreover, the benchmarks that use explanatory variables require additional data preparation as these methods cannot handle categorical variables or variables with text (strings). These additional data preparation steps are time-consuming and may be imperfect, as interpolating sales data is just a proxy of the real demand. We find that these data preparation steps only have a minor negative effect on forecast accuracy.¹⁸

¹⁸ We checked the impact of imputing missing values and making dummies of categorical variables on the forecast of our DTF. We found that the forecast accuracy of our proposed DTF decreases by less than 0.5%, on average, when using this additionally processed data instead of using the data with less data preparation.

¹⁶ <https://github.com/Mcompetitions/M5-methods/blob/master/validation/Point%20Forecasts%20-%20Benchmarks.R>

¹⁷ <https://cran.r-project.org/web/packages/smooth/index.html>

4.5. Performance measures

The forecast accuracy is evaluated using the root mean squared scaled error (RMSSE). The RMSSE scales the forecasting error by the one-step ahead forecasting error of the naive method. As a result, the errors can be compared between time series and can be used on time series with intermittency (Hyndman & Koehler, 2006). For these reasons, the RMSSE was also the key metric during the M5 Accuracy competition.¹⁹ Denote y_t the actual sales at period t , \hat{y}_t the forecast at period t , h the forecast horizon, and n the length of the training data, then the RMSSE is defined by:

$$\text{RMSSE} = \sqrt{\frac{\frac{1}{h} \sum_{t=n+1}^{n+h} (y_t - \hat{y}_t)^2}{\frac{1}{n-1} \sum_{t=2}^n (y_t - y_{t-1})^2}}. \quad (1)$$

Next to the forecast accuracy, we also keep track of the bias. Bias and inventory performance are shown to be interconnected (Kourentzes, Svetunkov, & Trapero, 2021). Similarly to the RMSSE, we scale the bias of each forecasting method by the absolute forecasting error of the one-step ahead naive forecasting method. We denote this the scaled mean error (SME):

$$\text{SME} = \frac{\frac{1}{h} \sum_{t=n+1}^{n+h} (y_t - \hat{y}_t)}{\frac{1}{n-1} \sum_{t=2}^n |y_t - y_{t-1}|}. \quad (2)$$

5. Results

This section describes the validation of our decision-tree framework (DTF) and the analysis of why our framework outperforms the benchmarks. In Section 5.1 we report the forecast accuracy over time, per time series category, and per product. Section 5.2 compares the performance to the various tree-based methods. Section 5.3 describes how Shapley values are used to give insight into how and when the DTF outperforms the benchmarks. In Section 5.4, we validate our insights on the M5 dataset. Finally, Section 5.5 briefly discusses the computational requirements of the different methods.

5.1. Results

Fig. 3 summarizes the forecast accuracy and the bias of our DTF, described in Section 4.2, and the benchmarks over time, averaged over all 4,523 products. We report the RMSSE and the SME over the test period of three months with a forecast horizon of seven days. The RMSSE reveals that the DTF substantially outperforms the best-performing statistical benchmark, ESX. The DTF outperforms ESX by 11.48%, on average. Moreover, the forecasting methods that make use of explanatory variables substantially outperform the methods that only use point-of-sales data. This shows the value of (investing in the data collection of) explanatory variables. It is noteworthy that Proph.X performs slightly worse than ESX. This is remarkable given that Proph.X can incorporate more explanatory variables and data sources than ESX. Finally, we find that our DTF has the smallest bias (SME) in absolute terms.

When we compare the average forecast accuracy and bias per time series category (smooth, erratic, intermittent, and lumpy) on the test set (we refer to Appendix B for the detailed results), we find that the DTF systematically outperforms the benchmarks for each time series category in terms of RMSSE. This shows the robustness of our proposed method regarding different time series categories. Interestingly, the forecasting methods designed specifically to deal with intermittency (i.e., CRO and ADIDA) do not outperform DTF or ESX.

Lastly, we compare the average forecast accuracy per product for the DTF and the ESX. More specifically, for each product, we look

whether the DTF or the ESX is more accurate over the three-month test set. For the RMSSE, the DTF outperforms ESX in 91% of the 4,523 products. We conclude that our DTF provides better daily sales forecasts for the large majority of the products.

5.2. Sensitivity analysis of our decision-tree framework

Table 3 compares our implementation of the DTF against its variants with varying levels of complexity. First, we discuss its performance against the more sophisticated versions, namely RDTF, Ensemble, DTF-l-1, DTF-l-2, DTF-sl-70, DTF-fs-sl-70, and DTF-m5. Our most sophisticated method, the DTF-m5, has the lowest RMSSE. However, the differences in RMSSE among these methods are relatively minor. The more sophisticated methods exhibit a maximum improvement of only 2.0% compared to the DTF in terms of RMSSE while using 24 times more LightGBM methods. Henceforth, we find little evidence that the more sophisticated methods substantially improve the forecast accuracy of the DTF. For most retailers, the higher model complexity may not be justified. A closer analysis of these results reveals that the DTF-sl-70 and the DTF-fs-sl-70 have nearly identical forecast errors. The feature selection procedure seems to have only a negligible impact on the average RMSSE, although successfully eliminating 40% to 73% of the inputs. When comparing against the DTF-l-1 and the DTF-l-2, we find that less pooling yields better results. Specifically, the DTF-l-1 performs slightly worse than our DTF, and the DTF-l-2 performs worse than the DTF-l-1, on average. This indicates that access to a larger dataset benefits the DTF. The comparison against the RDTF reveals that the recursive method is only slightly better. We note that the forecast horizon is limited to only seven days, which may be disadvantageous for the recursive methods. As some of the inputs of the non-recursive methods (such as the daily sales) need to be lagged by the length of the forecast horizon, longer forecast horizons result in less relevant inputs. As a consequence, for longer forecast horizons the difference between the DTF, the RDTF, and thus the ensemble may increase. We acknowledge that the DTF is mainly designed to produce short-term retail sales point forecasts.

Next, we discuss the performance of further simplifying the DTF method. For the pos-nfe-DTF, our most basic global LightGBM method, we find that the RMSSE drops substantially (i.e., by almost 25%) compared to our DTF implementation. This method does not use any engineered inputs or explanatory variables. It only uses the ‘raw’ point-of-sales data (sales lags and the date). It performs even worse than the ESX and the ES, which utilizes the same data. With engineered inputs, we find that the pos-fe-DTF slightly outperforms the ES according to the RMSSE. This suggests that, without explanatory variables, our global LightGBM method performs similarly to traditional forecasting methods without explanatory variables. By incorporating all data sources but without feature engineering, the forecast accuracy improves and becomes competitive compared to the ESX. This is shown by the nfe-DTF. These results indicate that the DTF outperforms benchmarks only when we can access and invest in explanatory variables and engineered inputs. Without these, simple statistical methods provide equal performance.

To identify the most valuable explanatory variables and engineered inputs for retailers, we compare subsets of our inputs. The pp-DTF (our pos-fe-DTF with price and promo data) shows that the inclusion of engineered price and promo-related variables mainly drives the outperformance of the DTF in our experiment. Despite not utilizing all data sources, its forecast accuracy remains similar to our DTF. This suggests that the other data sources, namely the hierarchy, holidays, weather forecasts, and events, have a limited impact on the average forecast accuracy. To check this hypothesis, we compare the results of the npp-DTF, which excludes the inputs related to price and promo. While it has access to all other inputs, it performs much worse than the DTF and barely improves the forecast accuracy of the pos-fe-DTF. These results indicate that retailers should mainly invest in price and promo-related inputs as the other explanatory variables are not equally important in terms of RMSSE.

¹⁹ In fact, the M5 Accuracy competition used a weighted version of the RMSSE by using the dollar value of each time series (Makridakis et al., 2022).

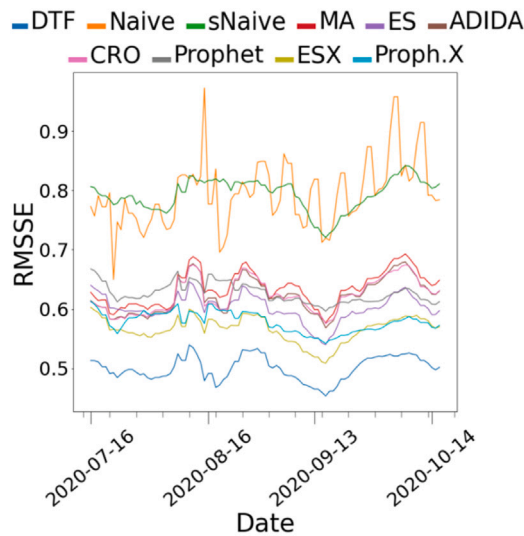


Fig. 3. Forecast accuracy of our proposed decision-tree framework (DTF) and the benchmarks.

Table 3

Average forecast accuracy and bias of our decision-tree framework (DTF) and variations. The RMSSE and the SME are based on the rolling forecasts of 4,523 products over a three-month test set with a one-week forecast horizon.

Method	RMSSE	Improvement over ESX	SME
DTF	0.501 ± 0.001	11.48%	-0.0007 ± 0.002
RDTF	0.499 ± 0.001	11.84%	0.0080 ± 0.019
Ensemble	0.497 ± 0.001	12.19%	0.0036 ± 0.008
DTF-l-1	0.503 ± 0.002	11.13%	0.0341 ± 0.007
DTF-l-2	0.514 ± 0.005	9.19%	-0.0006 ± 0.011
DTF-sl-70	0.502 ± 0.003	11.31%	-0.0025 ± 0.004
DTF-fs-sl-70	0.502 ± 0.002	11.31%	0.0157 ± 0.020
DTF-m5	0.491 ± 0.001	13.25%	0.0037 ± 0.006
pos-nfe-DTF	0.626 ± 0.001	-10.60%	-0.0426 ± 0.010
pos-fe-DTF	0.591 ± 0.002	-4.42%	0.0172 ± 0.037
nfe-DTF	0.549 ± 0.002	3.00%	-0.0182 ± 0.010
pp-DTF	0.508 ± 0.002	10.25%	0.0088 ± 0.014
npp-DTF	0.579 ± 0.001	-2.30%	-0.0060 ± 0.004

5.3. What is driving the good performance of the decision-trees?

We analyze why our implementation of the DTF outperforms the benchmarks using Shapley values. We provide an overview of the average absolute Shapley value per input in the Supplementary material, Section 2. To understand the value of each data source, we aggregate the average absolute Shapley value per data source and divide it by the total sum. We do a similar calculation for the ‘raw’ inputs and the engineered inputs. The results are summarized by Fig. 4. The left pie chart indicates that the point-of-sales data explains 62% (i.e., 53% + 9%) of the variance of the forecasts, while the explanatory variables explain 38%. This indicates the value of using explanatory variables in our DTF to predict sales. Specifically, price and promo-related variables explain 30% (22% + 8% respectively) of the variance. Data related to the product hierarchy explains 4%, and weather forecasts, holidays and events account together for only 4%.²⁰ The right pie chart compares the sum of the Shapley values of the ‘raw’ inputs against the engineered inputs. It shows that engineered inputs explain 79% of the variance

²⁰ Note that these percentage values only indicate the importance of the data source on average. For example, if weather data is only important for 1% of the products for a short period, it will result in low Shapley values on average. This does not mean that the data sources cannot be important for certain products or periods.

Method	RMSSE	Improvement over ESX	SME
DTF	0.501 ± 0.001	11.48%	-0.0007 ± 0.002
Naive	0.795	-40.46%	-0.0419
sNaive	0.793	-40.11%	-0.0024
MA	0.635	-12.19%	-0.0035
ES	0.602	-6.36%	-0.0301
CRO	0.623	-10.07%	-0.0253
ADIDA	0.623	-10.07%	-0.0116
ESX	0.566	Benchmark	-0.0501
Prophet	0.627	-10.78%	-0.1455
Proph.X	0.578	-2.12%	-0.1528

of the forecasts and the ‘raw’ inputs only 21%. This indicates the importance of the engineered inputs for our DTF. Note that these Shapley values do not indicate causality, nor can they be used to compute, for example, price elasticities. These values only explain how the black box model results in a different output when fed by different inputs.

Our results show that explanatory variables and engineered inputs are the largest contributors to the performance of our DTF. Our analysis also confirms the value of price and promo-related inputs, in line with the previous section. Thus, to outperform the benchmarks, the DTF requires both.

Our numerical analysis also reveals that the value of explanatory variables and engineered inputs is more limited for our benchmarks. While the RMSSE of DTF improves by 20% when explanatory variables are provided, Fig. 3 shows that the RMSSE improves by only 5.98% when ES uses explanatory variables (i.e., ESX). Compared to Prophet, Prophet.X improves by 7.81% when it includes explanatory variables. This indicates that the DTF benefits more from additional inputs than the benchmarks and can outperform these traditional forecasting methods. We believe that the reason for this is twofold. First, our DTF can deal with a high-dimensional input space to explain the unit sales. Second, the DTF can learn non-linear time series patterns.

High-dimensional input space: Our DTF uses 104 different inputs. Section 2 of the Supplementary material shows that most inputs have a non-zero Shapley value. This means that the method uses these inputs to make predictions. This is also confirmed by training our DTF with subsets of the inputs, which negatively affects the forecast accuracy (see Table 3). In the case of ESX and Prophet.X, respectively, only 4 and 14 different inputs improve the forecast accuracy. Including additional inputs has a neutral or even a negative impact on their forecast accuracy (see Section 4.4). The main cause of this large deviation (104 versus 4 or 14 inputs) is that DTF is a global method. Global methods use multiple time series to train a single method resulting in a much larger training dataset. Therefore, they can handle more inputs than the commonly used local forecasting methods without overfitting the data (Montero-Manso & Hyndman, 2021).

Non-linear time series patterns: Our DTF can identify non-linear time series patterns. Fig. 5 shows the SHAP dependency plots of four arbitrarily chosen inputs, namely sales_lag_3, day_of_the_month, daily_price/price_last_month_large_quantity, and promo_depth. Each dot represents the sales of one product for a certain day in the test set and explains the impact of a certain input value on the prediction. Combining these dots explains the learned relationship between each input and the prediction. The horizontal axis

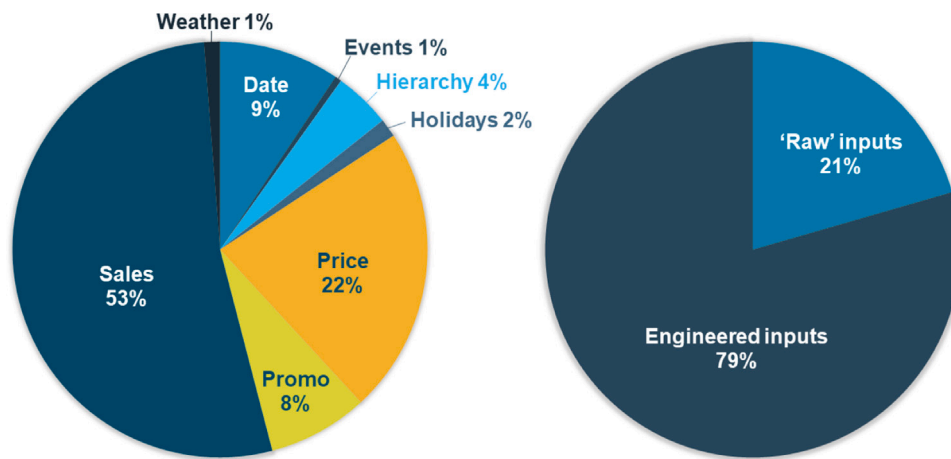


Fig. 4. Contribution of the different data sources (left pie chart) and feature engineering (right pie chart) according to the Shapley values of our decision-tree framework. The left figure shows that explanatory variables explain 38% of the variance of the forecasts and point-of-sales data 62%. The right figure indicates that 79% of the variance is explained by engineered inputs and only 21% by 'raw' inputs.

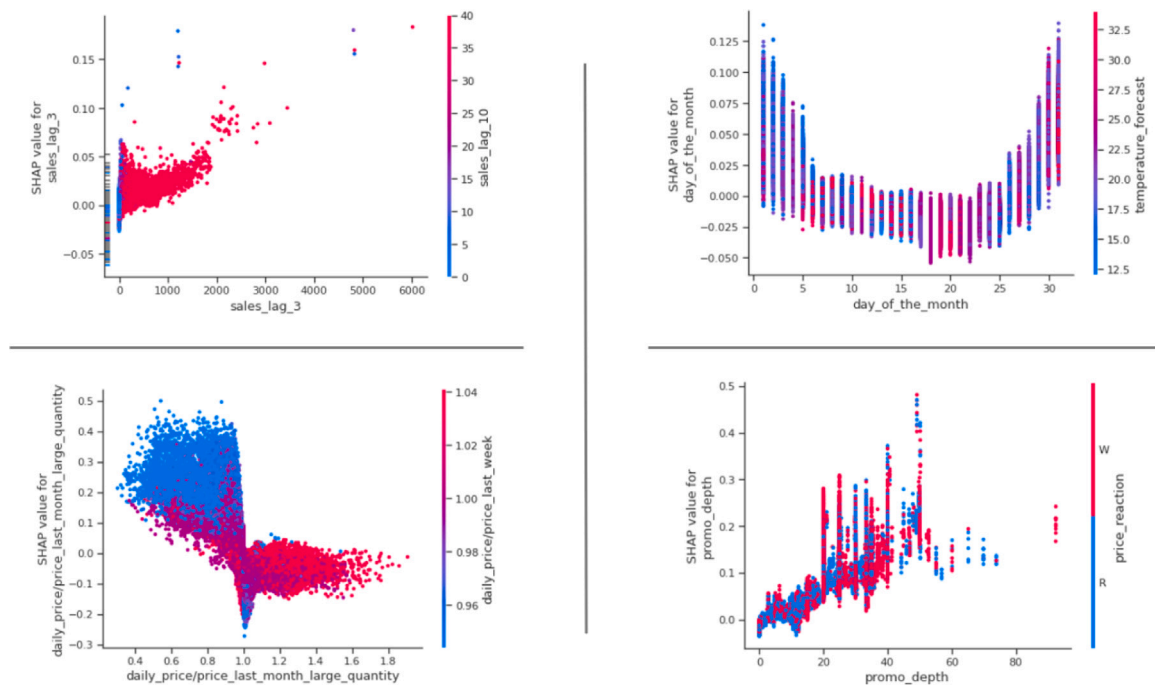


Fig. 5. SHAP dependency plots of four arbitrarily chosen inputs, namely sales_lag_3, day_of_the_month, daily_price/price_last_month_large_quantity, and promo_depth.

shows the input value. The vertical axis denotes the Shapley value, which visualizes the positive or negative impact on the prediction. The vertical (colorful) bar on the right-hand side of each dependency plot shows the input with the strongest interaction according to Friedman's H-statistic (Friedman & Popescu, 2008). Fig. 5 shows how the DTF can pick up non-linear time series patterns. Given the outperformance of our DTF and the fact that most common forecasting methods cannot learn non-linear patterns, we identify this as another important driver of its success.

The combination of learning non-linear time series patterns and coping with a wide variety of inputs explains how the DTF can outperform the benchmarks. However, our DTF requires access to explanatory variables and engineered inputs to exploit these capabilities.

5.4. Validation of the results on the M5 dataset

We apply the DTF on the M5 dataset to validate our findings and compare its forecasts to the M5 winning method and M5's most accurate benchmark, ES. The M5 dataset consists of sales data across ten stores with accompanying explanatory variables. The M5 competition aims to forecast sales for the next four weeks at 12 hierarchical levels, with the product-store level at the most granular level. As the DTF generates forecasts at the product-store level, we can easily aggregate these forecasts to higher levels of aggregation. We assess forecast accuracy using the weighted RMSSE (WRMSSE), with the errors weighted based on the monetary value associated with each product, see Makridakis et al. (2022). As before, we compute our method three times, each with

Table 4

Average forecast accuracy (WRMSSE) of the M5 winning method, our proposed decision-tree framework (DTF), and M5's winning benchmark (ES) on the M5 dataset. The WRMSSE is computed per aggregation level, with level (1) the most aggregated level and level (12) the most granular product-store level.

Aggregation level	ES	M5 winning method	DTF	Improvement DTF over ES
Total (1)	0.426	0.199	0.195 ± 0.004	54.23%
Per state (2)	0.514	0.310	0.297 ± 0.003	42.22%
Per store (3)	0.580	0.400	0.409 ± 0.005	29.48%
Per category (4)	0.478	0.277	0.253 ± 0.004	47.07%
Per department (5)	0.557	0.365	0.365 ± 0.007	34.47%
Per state-category (6)	0.577	0.390	0.373 ± 0.004	35.36%
Per state-department (7)	0.654	0.474	0.473 ± 0.005	27.68%
Per store-category (8)	0.643	0.480	0.482 ± 0.006	25.04%
Per store-department (9)	0.728	0.573	0.580 ± 0.006	20.33%
Per product (10)	1.012	0.966	0.988 ± 0.001	2.37%
Per product-state (11)	0.969	0.929	0.943 ± 0.001	2.68%
Per product-store (12)	0.915	0.884	0.894 ± 0.001	2.30%
Average	0.671	0.520	0.521 ± 0.003	22.35%

a different seed value. For the M5 winning method and ES, in contrast, we obtain these forecasts directly from Makridakis et al. (2022).

Table 4 summarizes the results. On average, our DTF approach demonstrates comparable accuracy to the more sophisticated (and computationally more expensive) M5 winning method. Recall that our DTF relies on only ten LightGBM methods, each employing the same hyperparameters and inputs. The M5 winning method, in contrast, utilizes 220 LightGBM methods with variations in hyperparameters and inputs (Wellens et al., 2021). Once more, we find that more sophisticated tree-based methods only marginally improve forecast accuracy compared to our DTF.

When comparing the results to ES, we find that the DTF, on average, substantially outperforms ES, especially at the aggregate levels. The difference between the DTF and ES is minor at the more disaggregated levels. For example, at the product-store level, i.e., level (12) in the M5 dataset, the DTF only outperforms ES by 2.30%, which is much less compared to the 11.48% outperformance on our private dataset, or the 54.23% outperformance at the most aggregated level, level (1). This can be attributed to the lack of price and promo-related inputs at level (12), which we identified as crucial for DTF's outperformance. The M5 competition provided explanatory variables, but feature information on price and promo-related data were limited at this level (12) — the available product prices are weekly averages that remain almost constant for most of the products throughout the training data. Additionally, the dataset only includes one type of promotion known as SNAP,²¹ which is not available to all customers. Although we recognize that certain retailers may engage in more promotions and price reductions than others, the dataset available appears to be limited in capturing the full extent of Walmart's activities. Walmart runs TV commercials, puts products on display, and even offers temporary discounts, among other strategies that may not be fully represented in the current dataset.²² As a result, the available price and promo-related data have limited predictive power. To conclude, we find that our DTF performs as well as more sophisticated tree-based methods on the M5 dataset. Traditional forecasting methods remain competitive at the product-store level when only a limited set of relevant explanatory variables is available.

5.5. Computational requirements

Our experiments were conducted on a local server, which features an Intel(R) Platinum 8160 CPU running at 2.10 GHz with 24 cores and 3 terabytes of RAM; 30% of the server's capacity was available for our DTF implementation, including hyperparameter tuning, training,

and predictions. Our DTF retrains monthly, while the hyperparameter tuning is a one-time investment. The DTF requires roughly 600 min for the hyperparameter tuning, 17 min to train one non-recursive LightGBM method, and only 0.05 min to make forecasts for the next seven days. Note that the computational demands of hyperparameter tuning are closely linked to the number of iterations. In our experiments, we set the iteration count to 21. In short, once the hyperparameter tuning has been conducted, the DTF requires roughly 20 min to make rolling forecasts for one month in one store. This is competitive with our top-performing benchmarks: ES requires around 25 min of computation time, Prophet 55 min, and Proph.X 95 min. The benchmarks were run on the same server in parallel, utilizing all available CPU cores, and following the practice of daily recalibration. We exert caution with these numbers, as computational requirements depend on various aspects such as code optimizations and the frequency of retraining and hyperparameter tuning. In addition, the DTF approach relies heavily on data extraction and feature engineering from the data warehouse, which can be time-consuming and dependent on the infrastructure. We conclude that the computational requirements of our DTF approach are not necessarily higher than traditional local forecasting methods. However, preparing the data for use can be cumbersome and time-consuming.

When we compare the computational requirements of our DTF approach with its advanced variations, we find that for the recursive versions of our DTF, the prediction phase takes significantly longer – up to 100 times more – while the hyperparameter tuning and method training only take around 5% longer. Methods involving a feature selection procedure require an additional 1–17 min per method. The computational impact of the tree-based methods with different pooling strategies is less explicit. The hyperparameter tuning procedure per method of the DTF-l-1 and DTF-l-2 is slightly shorter compared to DTF. However, if we had chosen to tune the hyperparameters for each method per group of time series, DTF-l-1 and DTF-l-2 would have taken significantly longer. The training time is very similar across the different methods. This is no surprise as, on the one hand, DTF-l-1 and DTF-l-2 train multiple LightGBM methods compared to the single DTF method. On the other hand, each of the LightGBM methods of DTF-l-1 and DTF-l-2 can have fewer model parameters (and are thus faster to train) as they use fewer time series. However, the prediction time for DTF-l-1 and DTF-l-2 is roughly twice as compared to DTF. In short, we do not find evidence that pooling increases or decreases the total computation time. Finally, as the DTF-m5 is a large ensemble of 24 different LightGBM methods, it requires roughly six times longer to tune the hyperparameters than the DTF, 24 times longer to train the methods, and it takes over 500 times longer to make the forecasts (up to roughly 25 min to make seven-day forecasts).

²¹ <https://www.kaggle.com/competitions/m5-forecasting-accuracy/data>

²² <https://edition.cnn.com/2022/02/18/business/walmart-rollbacks-promotions-inflation/index.html>

6. Impact on inventory control

In this section, we perform extensive numerical experimentation to investigate whether the observed forecast superiority translates into higher service levels, lower inventory costs, and improvements regarding the variability of orders and inventory (i.e., the bullwhip effect). To evaluate the impact of the forecasts of our DTF on the inventory control, we consider a discrete-time, periodic-review, single-echelon, general automatic pipeline, variable inventory, order-based, production control system (APVIOBPCS) with backlogging (Udenio, Vamatidou, Fransoo, & Dellaert, 2017). APVIOBPCS is an implementation of a generalized order-up-to policy that has been extensively used in the literature to investigate different aspects of the bullwhip effect. Relevant to our work, Dejonckheere, Disney, Lambrecht, and Towill (2003) use such a system to quantify the influence of several forecasting methods on the order and inventory variability, Li, Disney, and Gaalman (2014) show that dampened trend forecasting can help reduce the bullwhip effect in order-up-to systems, and Udenio, Vamatidou, and Fransoo (2022) compare the performance of simple and seasonal exponential smoothing forecasts under (seasonal) deterministic, stochastic, and empirical demands — the latter using data from the M5 forecasting competition. We refer the reader to Wang and Disney (2016) for, respectively, comprehensive reviews of the bullwhip effect in general and the application of the APVIOBPCS family of policies in particular.

6.1. Inventory model

Our numerical implementation of the general APVIOBPCS model is based upon Udenio et al. (2022). The structural parameters of the system are the inventory coverage ($C \in \mathbb{N}$) and the delivery lead time ($L \in \mathbb{N}$). In contrast with prior implementations of APVIOBPCS, which typically use simple exponential smoothing or other related forecasting methodologies, we incorporate forecasts calculated with our DTF – as well as 9 benchmark methodologies – into the inventory system. (See Section 4.4 for a detailed description of all benchmarks.)

The system maintains a target inventory (\hat{i}) equal to the expected demand over C periods and a target pipeline (\hat{p}) equal to the expected lead time demand. The lead time is assumed deterministic and defined as the time elapsed between placing and receiving a replenishment order under non-stockout conditions. Replenishment orders (o) depend on the gap between actual and desired values of the inventory (i) and pipeline (p). The behavioral smoothing parameters of the system are the inventory ($\gamma_I \in \mathbb{R}$) and pipeline ($\gamma_P \in \mathbb{R}$) adjustment factors. The behavioral parameters specify the fraction of the gap between target and actual values taken into account to calculate orders: γ_I is the fraction of the inventory gap to be closed, and γ_P is the fraction of the pipeline gap to be closed. For instance, a system with $\gamma_I = 1$ and $\gamma_P = 0$ completely closes the inventory gap with every order while it ignores the pipeline entirely. Prior research shows that setting the behavioral parameters such that $\gamma_I = \gamma_P$ is beneficial for the performance of the inventory system — policies with equal adjustments minimize the variability of orders and inventory (Udenio et al., 2017).

For a given forecasting method, let $f_{t,t+m}$ be the demand forecast calculated at period t for period $t+m$. Formally, the sequence of events and the equations in the model are as follows: at the beginning of period (t), a replenishment order (o_t) based on the previous period's demand forecast ($f_{t-1,t+L}$) is placed with the supplier. Following this, the orders placed L periods prior are received. Next, the demand for the period (d_t) is observed and served. Excess demand is back-ordered. Then, the demand forecast ($f_{t,t+L+1}$) is updated. The forecast represents the expected demand and is used to compute the target levels of both inventory and pipeline,

$$\hat{i}_t = \sum_{k=1}^C f_{t,t+k},$$

$$\hat{p}_t = \sum_{k=1}^L f_{t,t+k}.$$

The orders that will be placed in the following period (o_{t+1}) are generated according to an anchor and adjustment-type procedure,

$$o_{t+1} = \gamma_I (\hat{i}_t - i_t) + \gamma_P (\hat{p}_t - p_t) + f_{t,t+L+1}.$$

The balance equations for inventory (i) and pipeline (p) are: $i_t = i_{t-1} + o_{t-L} - d_t$, and $p_t = p_{t-1} + o_t - o_{t-L}$. Orders and inventories can be negative (i.e., backlogs and returns are allowed) to maintain the linearity of the model.

6.2. Performance metrics

We quantify the inventory performance of the system through several metrics common in the literature. Specifically, we compute the mean inventory, the achieved service level, and the bullwhip of orders (BW_O), and inventory (BW_I). Due to the existence of backlogs in the system, we calculate the mean (positive) inventory as $1/n^* \sum_n i_t^+$, where n is the number of periods in a given time series and n^* the number of periods without a backlog. We compute the achieved service level as the fraction of periods with positive inventory for a given time series, i.e., n^*/n . The BW_O is computed as σ_O^2/σ_D^2 with σ_O^2 the order variance and σ_D^2 the demand variance. The BW_I is similarly computed as σ_I^2/σ_D^2 with σ_I^2 defined as the variance of the inventory time series.

6.3. Numerical experimentation

Our experimental setup is as follows. For each of the 4,523 products, we feed the demand and sales forecast data into the inventory model to compute the evolution of the order and inventory time series. We compare 10 forecasting methods; in addition to our DTF and the benchmarks, we also include the sales forecasts of two slightly adjusted versions of our DTF. First, we retrain the DTF without using any engineered inputs. This method is denoted as the nfe-DTF, and uses all the inputs of Table 1, but none of Table A.6. Second, we retrain the DTF without any explanatory variables but with engineered inputs, which we denote as the pos-fe-DTF. This method uses all inputs related to the first two rows of Tables 1 and A.6. The hyperparameters of these methods are separately optimized with Optuna using the validation set. We only use the first seed value for our three DTF methods.

For consistency, we use the same test set (3 months) as in Section 5. We set the smoothing parameters $\gamma_I = \gamma_P = 0.5$, noting that the relative performance of the forecasting methods is consistent under different values. (See Hoberg and Thonemann (2015) for a detailed discussion on the effect of the equally-set smoothing parameters on the cost, variability, and responsiveness of the inventory system.) We set the lead time as $L = 2$ because it well approximates the ordering procedure at the retailer, where individual stores place orders daily to the central distribution center. We assume the latter has ample availability of all products.

Finally, to enable a fair comparison across products and forecasting methods, we set C individually for each combination of product and forecast method such that the achieved service level is at least 95%. The search space of C is limited between $C = 1$ and $C = 7$, which is typical for the product types in our dataset. As a result of this constraint on C , some products fail to achieve the target service level. Thus, we report this metric as an additional performance characteristic. In addition, as C is discrete, some products may face a service level that is higher than 95%. Given that we optimize for a service level of 95%, we prefer an inventory policy that achieves at least 95% with the lowest possible costs and variability in orders and inventory levels.

Table 5

Impact of different forecasting methods on the inventory performance. This table shows the different forecasting methods, the RMSSE, the SME, the mean inventory, the mean achieved service level, the number of products that have a service level below 95%, and the mean bullwhip of orders (BW_O) and inventory (BW_I).

Method	RMSSE	SME	Mean inventory	Mean service level	Products with SL <95%	Mean BW_O	Mean BW_I
DTF	0.500	−0.0029	34.39	96.54%	393	1.93	5.43
nfe-DTF	0.547	−0.0085	38.43	96.22%	460	1.80	6.52
pos-fe-DTF	0.589	0.0544	40.86	95.42%	829	1.77	5.15
Naive	0.795	−0.0419	48.31	90.10%	2492	16.41	17.75
sNaive	0.793	−0.0024	49.42	93.14%	1695	4.34	11.88
MA	0.635	−0.0035	41.22	94.42%	1254	1.91	8.32
ES	0.602	−0.0301	40.37	95.69%	766	2.55	6.85
CRO	0.623	−0.0253	39.22	96.04%	590	1.84	7.65
ADIDA	0.623	−0.0116	38.87	95.75%	743	1.66	5.98
ESX	0.566	−0.0501	39.29	95.84%	684	2.77	8.57
Prophet	0.627	−0.1455	43.93	95.18%	766	2.11	7.93
Proph.X	0.578	−0.1528	43.52	95.47%	683	2.77	9.92

6.4. Results

Table 5 summarizes the inventory results. First, we find that the forecasting methods that are more accurate, in terms of RMSSE, indeed achieve higher service levels with lower average inventory levels. More specifically, the DTF achieves a slightly higher service level than ESX, with 12.47% less inventory on average. We also note that, under these settings, ESX fails to meet the 95% target service level for 684 products, while this is only true for 393 products in the case of our DTF. This implies that our DTF is capable of either higher product availability at the same coverage level, or lower inventory requirements at the same service level.

Second, we discuss the results of the BW_I and the BW_O . The best performing forecasting method regarding the BW_I and the BW_O are the ADIDA method and the variants of the DTF.²³ While adding explanatory variables improves the forecast accuracy (and lowers the mean inventory and increases the service level), it does worsen the BW_I and the BW_O . This holds true for our DTF, ES, and Prophet. One potential explanation for this observation is that methods with explanatory variables are quicker to adjust forecasts down/up without having to wait for changes in the actual sales time series (e.g., by anticipating the end of a promotion). See Wellens, Boute, and Udenio (2023) for a detailed discussion.

The above results indicate that our DTF is competitive across all inventory metrics. Moreover, our results show that adding explanatory variables tends to improve the performance of the system as measured by the mean inventory and achieved service level but worsens the performance as measured by the bullwhip metrics; suggesting a trade-off between optimizing for availability vs. variability. We conclude that retailers need to select the forecasting method that best fits the objective of their operations.

7. Conclusion

In this paper, we validate various tree-based methods with different levels of complexity to support retailers on whether and how to invest in tree-based machine learning (ML) forecasting. We show how a straightforward implementation of a tree-based method outperforms traditional forecasting methods by 11.48% on average, while being computationally efficient. More sophisticated versions of the tree-based method only marginally improve the forecast accuracy, with improvements of up to 2.0%. It shows that the ‘raw’ performance of tree-based methods does not come from minor tweaks such as smart feature engineering or smart ways of pooling data. By analyzing various implementations of our decision-tree framework (DTF) and validating our results on the M5 dataset, we find that the superior performance of

DTF depends on the availability of explanatory variables and feature engineering. We believe this is an important result because it allows managers to quickly assess whether the implementation of a tree-based method has immediate potential and gives guidance regarding the allocation of resources for (better) data gathering. Extensive numerical experimentation finally shows how the forecast superiority of our proposed framework translates to higher service levels (+0.73%), lower inventory costs (−12.47%), and improvements in the bullwhip of orders (−30.32%) and inventory (−36.64%) compared to the most accurate benchmark. With our framework, its excellent performance, and its scalability to practical forecasting settings, we hope to increase the adoption rate of machine learning for ‘traditional’ retailers further.

We believe there is value in extending the DTF to, e.g., probability forecasting, long-term forecasting, hierarchical forecasting, or doing a similar analysis for ML methods based on neural networks. For instance, an interesting way to make probability forecasts with our DTF is by exploring a pinball loss function, enabling the prediction of different quantiles by only changing a few lines of code. Regarding the forecast horizon, it is important to note that our study primarily addresses short-term forecasting. It would be worthwhile to investigate the applicability and performance of the DTF in long-term forecasting scenarios such as the yearly planning of weekly truck capacity. In this case, more attention should be given to the recursive version of the DTF and its lagged values. Another potential extension of the DTF relates to temporal and hierarchical aggregation, which has proven to be effective in various other forecasting methods (Athanasopoulos, Hyndman, Kourentzes, & Petropoulos, 2017; Nikolopoulos et al., 2011). Finally, this paper focused on tree-based methods. It may be interesting to explore a similar analysis for neural networks to identify their key ingredients to outperform traditional forecasting methods. However, each of these would be full research projects on its own.

We are aware that our study also comes with some limitations. The ESX and Proph.X benchmarks are implemented using a feature selection procedure, while the DTF does not. Since global ML methods differ significantly from local statistical methods, adopting a similar feature selection procedure for both the tree-based methods and the benchmarks was not feasible. Another remark is that the feature selection procedure of ESX and Proph.X could be further improved by selecting different inputs for each time series individually. This would increase the model complexity of the benchmarks. Another limitation comes from the size of our dataset, as it is limited to products of only one store. Our analysis demonstrates that DTF-I-1 and DTF-I-2, which are trained on subsets of the data, perform worse than our DTF. Therefore, it would be interesting to investigate how incorporating a larger pool of time series from different stores would impact the forecast accuracy and computational cost of the DTF. Prior research using M5 data showed that pooling time series of multiple stores, instead of pooling per store, reduces computational costs while attaining similar levels of forecast accuracy (Wellens et al., 2021). This raises the question whether pooling at the highest hierarchical level is a good rule of thumb. Lastly,

²³ Note that MA and CRO slightly outperform our DTF regarding BW_O , but have a much higher BW_I .

leveraging weather forecasts spanning a one-week forecast horizon, rather than the current two-week span, could amplify their effect on our sales forecasts. However, such data was unavailable to the retailer at that time.

Declaration of competing interest

Arnoud Wellens is funded by Flanders Innovation Entrepreneurship (VLAIO), grant number HBC.2020.2215. Maximiliano Udenio and Robert N. Boute declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Engineered inputs

Table A.6 gives an overview of the engineered inputs. Below we briefly explain these.

Sales: Regarding the first data source, namely sales, we compute the mean and the standard deviation (stdv) of the unit sales of the last 7, 14, 30, 60, and 180 days. The determination of these windows is based on the intuition that the daily sales of retailers typically exhibit weekly and monthly seasonality. These are denoted as `rolling_mean_7`, `rolling_stdv_7`, `rolling_mean_14`, `rolling_stdv_14`, `rolling_mean_30`, `rolling_stdv_30`, `rolling_mean_60`, `rolling_stdv_60`, `rolling_mean_180`, and `rolling_stdv_180`. Before computing these inputs, we lag the unit sales according to our forecast horizon. This is necessary when forecasting non-recursively for longer forecast horizons than the step-one forecast. Otherwise, these inputs would use information which is not yet available, which is called leakage. For example, the `rolling_mean_7` uses the sales data of the last seven days. If we compute this for the step 2 forecast (forecasting the sales in two days), we require the unit sales of the next day (step 1), which is not known when making the forecast. As we do not forecast recursively, we therefore need to lag the unit sales by at least one period. When forecasting for longer horizons (as in our case), we need to lag the unit sales even more. We also compute the average daily sales per product (`enc_product_ID_mean`) and the stdv per product (`enc_product_ID_stdv`). These inputs are recomputed per period, using all the available unit sales data up to that period in time.

Date: For the second data source, date, we create inputs that indicate the day of the week (`day_of_the_week`), whether we are the end of the week (`end_of_the_week`), the week of the month (`week_of_the_month`), and the quarter (`quarter`).

Hierarchy: In terms of the product hierarchy, we compute the average unit sales and stdv per hierarchical group. This gives the following inputs: `enc_productgroup_1_1_mean`, `enc_productgroup_1_1_stdv`, `enc_productgroup_1_2_mean`, `enc_productgroup_1_2_stdv`, `enc_productgroup_1_3_mean`, `enc_productgroup_1_3_stdv`, `enc_productgroup_1_4_mean`, `enc_productgroup_1_4_stdv`, `enc_productgroup_1_5_mean`, and `enc_productgroup_1_5_stdv`. These inputs are recomputed per period, using all the available unit sales data up to that period in time.

Holidays: Regarding the holidays, we create lags and future values of the national public holidays for up to four days in advance and after the holiday. We denote this inputs as `lags_future_indication_public_holiday`.

Weather forecasts: In terms of weather forecasts, we compute the average temperature of last month (`average_temperature_last_month`).

Price: For the different daily prices, we compute the maximum value (`max_price`, `max_price_large_quantity`, and `max_price_large_quantity_pickup`), the minimum value (`min_price`, `min_price_large_quantity`, and `min_price_large_quantity_pickup`), the mean (`price_mean`, `price_mean_large_quantity`, and `price_mean_large_quantity_pickup`) and the stdv (`price_stdv`, `price_stdv_large_quantity`, and

`price_stdv_large_quantity_pickup`) of the daily price. These inputs are recomputed per period, using all the available unit sales data up to that period in time. In addition, we compute the national price of a product (by taking the average price over multiple stores), and the average national price of last month. These are denoted as `national_price`, `average_national_price_last_month`, and `average_national_price_last_month_pickup`. We compute the price differences between the daily price with and without a volume discount (`price_large_quantity_abs_diff` and `price_large_quantity_abs_diff_pickup`). These are denoted in absolute value. By taking a simple division, we compare the daily price with volume discount to the maximum price (`daily_price/max_price`, `daily_price/max_price_large_quantity`, and `daily_price/max_price_large_quantity_pickup`), the average price (`daily_price/price_mean`, `daily_price/price_mean_large_quantity`, `daily_price/price_mean_large_quantity_pickup`), and to the average price of the previous month (`daily_price/price_last_month`, `daily_price/price_last_month_large_quantity`, and `daily_price/price_last_month_large_quantity_pickup`) and week (`daily_price/price_last_week`, `daily_price/price_last_week_large_quantity`, and `daily_price/price_last_week_large_quantity_pickup`). Note that we use the different prices, namely `daily_price`, `daily_price_large_quantity`, `daily_price_pickup`, and `daily_price_large_quantity_pickup`, purely as an indication for the DTF. For instance, we do not disclose the quantity of products sold (either in absolute or relative terms) with their corresponding prices.

Promo: Regarding our promotional data, we create new inputs indicating the starting day of a promo (`first_day_promo_dummy`), the first week of a promo (`first_week_promo_dummy`), the second week of a promo (`second_week_promo_dummy`) or none of the above (`rest_promo_dummy`). Finally, we create lags and future values of the promos for up to seven days in advance and after the promo (`lags_future_indication_promo_dummy`).

Note that the feature engineering does not require advanced Python or sophisticated domain knowledge. The ‘raw’ inputs combined with the engineered inputs, gives us in total a set of 104 inputs in per time series.

Appendix B. Forecast metrics per time series category

Table B.7 compares the average forecast accuracy and bias per time series category (smooth, erratic, intermittent, and lumpy) on the three-month test set in greater detail. Regarding the RMSSE, the DTF outperforms ESX for smooth, lumpy, intermittent and erratic time series by 9.57%, 15.64%, 12.64%, and 11.52% respectively. Interestingly enough, the forecasting methods that are designed to deal with intermittency (i.e., CRO and ADIDA) do not outperform DTF or ESX. In addition, according to the RMSSE, the most difficult time series categories to predict are the time series with little intermittency. This means that for all methods, the RMSSE is the highest for the smooth and the erratic time series.

The results of the SME indicate that sNaive and our DTF have the lowest bias in absolute terms.

Appendix C. Supplementary data

Supplementary material related to this article can be found online at <https://doi.org/10.1016/j.ejor.2023.10.039>.

Table A.6
Overview of engineered inputs that are used by our proposed method.

Data source	Engineered inputs
Sales	rolling_mean_7, rolling_stdv_7, rolling_mean_14, rolling_stdv_14, rolling_mean_30, rolling_stdv_30, rolling_mean_60, rolling_stdv_60, rolling_mean_180, rolling_stdv_180, enc_product_ID_mean, enc_product_ID_stdv
Date	day_of_the_week, end_of_the_week, week_of_the_month, quarter
Hierarchy	enc_productgroup_l_1_mean, enc_productgroup_l_1_stdv, enc_productgroup_l_2_mean, enc_productgroup_l_2_stdv, enc_productgroup_l_3_mean, enc_productgroup_l_3_stdv, enc_productgroup_l_4_mean, enc_productgroup_l_4_stdv, enc_productgroup_l_5_mean, enc_productgroup_l_5_stdv
Holidays	lags_future_indication_public_holiday
Weather	average_temperature_last_month
Price	max_price, min_price, price_stdv, price_mean, max_price_large_quantity, min_price_large_quantity, price_stdv_large_quantity, price_mean_large_quantity, max_price_large_quantity_pickup, min_price_large_quantity_pickup, price_stdv_large_quantity_pickup, price_mean_large_quantity_pickup, national_price, average_national_price_last_month, average_national_price_last_month_pickup, price_large_quantity_abs_diff, price_large_quantity_abs_diff_pickup, daily_price/max_price, daily_price/price_mean, daily_price/price_last_month, daily_price/price_last_week, daily_price/max_price_large_quantity, daily_price/price_mean_large_quantity, daily_price/price_last_month_large_quantity, daily_price/price_last_week_large_quantity, daily_price/max_price_large_quantity_pickup, daily_price/price_mean_large_quantity_pickup, daily_price/price_last_month_large_quantity_pickup, daily_price/price_last_week_large_quantity_pickup
Promo	first_day_promo_dummy, first_week_promo_dummy, second_week_promo_dummy, rest_promo_dummy, lags_future_indication_promo_dummy

Table B.7
Forecast accuracy per time series category of our proposed decision-tree framework (DTF) and the benchmarks, averaged over all 4,523 products. The RMSSE and the SME are based on the daily rolling forecasts of 4,523 products over a three-month test set with a one-week forecast horizon.

Accuracy metric	Method	Smooth	Lumpy	Intermittent	Erratic
RMSSE	DTF	0.520 ± 0.001	0.453 ± 0.001	0.470 ± 0.002	0.530 ± 0.001
	Naive	0.834	0.735	0.724	0.826
	sNaive	0.789	0.784	0.762	0.843
	MA	0.656	0.598	0.578	0.673
	ES	0.604	0.583	0.571	0.647
	CRO	0.642	0.587	0.576	0.658
	ADIDA	0.644	0.584	0.570	0.660
	ESX	0.575	0.537	0.538	0.599
	Prophet	0.638	0.597	0.587	0.667
	Proph.X	0.602	0.522	0.533	0.610
SME	DTF	0.0020 ± 0.003	−0.0024 ± 0.007	−0.0179 ± 0.007	0.0050 ± 0.005
	Naive	−0.0552	−0.0294	−0.0224	−0.0401
	sNaive	−0.0006	−0.0055	−0.0053	−0.0012
	MA	−0.0009	−0.0083	−0.0056	−0.0030
	ES	−0.0238	−0.0356	−0.0561	−0.0148
	CRO	−0.0076	−0.0445	−0.0623	−0.0148
	ADIDA	−0.0046	−0.0183	−0.0283	−0.0063
	ESX	−0.0391	−0.0659	−0.0672	−0.0454
	Prophet	−0.1917	−0.0799	−0.1102	−0.1275
	Proph.X	−0.1858	−0.1064	−0.1104	−0.1565

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623–2631).

Assimakopoulos, V., & Nikolopoulos, K. (2000). The theta model: A decomposition approach to forecasting. *International Journal of Forecasting*, 16(4), 521–530.

Athanasopoulos, G., Hyndman, R. J., Kourentzes, N., & Petropoulos, F. (2017). Forecasting with temporal hierarchies. *European Journal of Operational Research*, 262(1), 60–74.

Bandara, K., Bergmeir, C., & Smyl, S. (2020). Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Systems with Applications*, 140, Article 112896.

Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, B., Maddix, D., Turkmen, C., et al. (2020). Neural forecasting: Introduction and literature overview. *arXiv preprint arXiv:2004.10240*.

- Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24.
- Bojer, C. S., & Meldgaard, J. P. (2021). Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, 37(2), 587–603.
- Crone, S. F., Hibon, M., & Nikolopoulos, K. (2011). Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, 27(3), 635–660.
- Croston, J. D. (1972). Forecasting and stock control for intermittent demands. *Journal of the Operational Research Society*, 23(3), 289–303.
- Dejonckheere, J., Disney, S. M., Lambrecht, M. R., & Towill, D. R. (2003). Measuring and avoiding the bullwhip effect: A control theoretic approach. *European Journal of Operational Research*, 147(3), 567–590.
- Dietvorst, B. J., Simmons, J. P., & Massey, C. (2015). Algorithm aversion: people erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology: General*, 144(1), 114.
- Fildes, R., Goodwin, P., Lawrence, M., & Nikolopoulos, K. (2009). Effective forecasting and judgmental adjustments: An empirical evaluation and strategies for improvement in supply-chain planning. *International Journal of Forecasting*, 25(1), 3–23.
- Fildes, R., Goodwin, P., & Önköl, D. (2019). Use and misuse of information in supply chain forecasting of promotion effects. *International Journal of Forecasting*, 35(1), 144–156.
- Fildes, R., Ma, S., & Kolassa, S. (2019). Retail forecasting: Research and practice. *International Journal of Forecasting*.
- Fildes, R., & Petropoulos, F. (2015). Improving forecast quality in practice. *Foresight: The International Journal of Applied Forecasting*, 36, 5–12.
- Fisher, M., & Raman, A. (2018). Using data and big data in retailing. *Production and Operations Management*, 27(9), 1665–1669.
- Friedman, J. H., & Popescu, B. E. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 916–954.
- Gilliland, M. (2020). The value added by machine learning approaches in forecasting. *International Journal of Forecasting*, 36(1), 161–166.
- Guo, C., & Berkhahn, F. (2016). Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737.
- Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1), 388–427.
- Hoberg, K., & Thonemann, U. W. (2015). Analyzing variability, cost, and responsiveness of base-stock inventory policies with linear control theory. *IEEE Transactions*, 47(8), 865–879.
- Huber, J., & Stuckenschmidt, H. (2020). Daily retail demand forecasting using machine learning with emphasis on calendric special days. *International Journal of Forecasting*, 36(4), 1420–1438.
- Hyndman, R. J. (2020). A brief history of forecasting competitions. *International Journal of Forecasting*, 36(1), 7–14.
- Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4), 679–688.
- Januschowski, T., Gasthaus, J., Wang, Y., Salinas, D., Flunkert, V., Bohlke-Schneider, M., et al. (2020). Criteria for classifying forecasting methods. *International Journal of Forecasting*, 36(1), 167–177.
- Januschowski, T., Wang, Y., Torkkola, K., Erkkilä, T., Hasson, H., & Gasthaus, J. (2021). Forecasting with trees. *International Journal of Forecasting*.
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., et al. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems* (pp. 3146–3154).
- Kourentzes, N., Svetunkov, I., & Trapero, J. R. (2021). Connecting forecasting and inventory performance: a complex task. Available at SSRN 3878176.
- Li, Q., Disney, S. M., & Gaalman, G. (2014). Avoiding the bullwhip effect using damped trend forecasting and the order-up-to replenishment policy. *International Journal of Production Economics*, 149, 3–16.
- Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 4768–4777).
- Ma, S., & Fildes, R. (2021). Retail sales forecasting with meta-learning. *European Journal of Operational Research*, 288(1), 111–128.
- Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., et al. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2), 111–153.
- Makridakis, S., Chatfield, C., Hibon, M., Lawrence, M., Mills, T., Ord, K., et al. (1993). The M2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, 9(1), 5–22.
- Makridakis, S., & Hibon, M. (2000). The M3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 16(4), 451–476.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLoS One*, 13(3), Article e0194889.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1), 54–74.
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2022). M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*.
- May, R., Dandy, G., & Maier, H. (2011). Review of input variable selection methods for artificial neural networks. *Artificial Neural Networks-Methodological Advances and Biomedical Applications*, 10, 16004.
- McCarthy, T. M., Davis, D. F., Golcic, S. L., & Mentzer, J. T. (2006). The evolution of sales forecasting management: A 20-year longitudinal study of forecasting practices. *Journal of Forecasting*, 25(5), 303–324.
- Montero-Manso, P., & Hyndman, R. J. (2021). Principles and algorithms for forecasting groups of time series: Locality and globality. *International Journal of Forecasting*, 37(4), 1632–1653.
- Morlidge, S. (2014). Forecast quality in the supply chain. *Foresight: The International Journal of Applied Forecasting*, (33).
- Mukherjee, S., Shankar, D., Ghosh, A., Tathawadekar, N., Kompalli, P., Sarawagi, S., et al. (2018). ARMDN: Associative and recurrent mixture density networks for eRetail demand forecasting. arXiv preprint arXiv:1803.03800.
- Nikolopoulos, K., & Petropoulos, F. (2018). Forecasting for big data: Does suboptimality matter? *Computers & Operations Research*, 98, 322–329.
- Nikolopoulos, K., Syntetos, A. A., Boylan, J. E., Petropoulos, F., & Assimakopoulos, V. (2011). An aggregate-disaggregate intermittent demand approach (ADIDA) to forecasting: An empirical proposition and analysis. *Journal of the Operational Research Society*, 62(3), 544–554.
- Petropoulos, F., Grushka-Cockayne, Y., Siemsen, E., & Spiliotis, E. (2021). Wielding occam's razor: Fast and frugal retail forecasting. arXiv preprint arXiv:2102.13209.
- Petropoulos, F., Nikolopoulos, K., Spithourakis, G. P., & Assimakopoulos, V. (2013). Empirical heuristics for improving intermittent demand forecasting. *Industrial Management & Data Systems*.
- Rinderknecht, M. D., & Klopfenstein, Y. (2021). Predicting critical state after COVID-19 diagnosis: Model development using a large US electronic health record dataset. *npj Digital Medicine*, 4(1), 113.
- Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3), 1181–1191.
- Seaman, B. (2018). Considerations of a retail forecasting practitioner. *International Journal of Forecasting*, 34(4), 822–829.
- Shapley, L. S. (1953). A value for n-person games. *Contributions to Theory Games (AM-28)*, vol. 2. Princeton, NJ, USA: Princeton University Press.
- Shwartz-Ziv, R., & Armon, A. (2021). Tabular data: Deep learning is not all you need. arXiv preprint arXiv:2106.03253.
- Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1), 75–85.
- Spiliotis, E., Makridakis, S., Semoglou, A.-A., & Assimakopoulos, V. (2020). Comparison of statistical and machine learning methods for daily SKU demand forecasting. *Operational Research - An International Journal*, 1–25.
- Syntetos, A. A., & Boylan, J. E. (2005). The accuracy of intermittent demand estimates. *International Journal of Forecasting*, 21(2), 303–314.
- Syntetos, A. A., Boylan, J. E., & Croston, J. (2005). On the categorization of demand patterns. *Journal of the Operational Research Society*, 56(5), 495–503.
- Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37–45.
- Udenio, M., Vatamidou, E., & Fransoo, J. C. (2022). Exponential smoothing forecasts: Taming the bullwhip effect when demand is seasonal. *International Journal of Production Research*, 1–18.
- Udenio, M., Vatamidou, E., Fransoo, J. C., & Dellaert, N. (2017). Behavioral causes of the bullwhip effect: An analysis using linear control theory. *IIE Transactions*, 49(10), 980–1000.
- Wang, X., & Disney, S. M. (2016). The bullwhip effect: Progress, trends and directions. *European Journal of Operational Research*, 250(3), 691–701.
- Wellens, A. P., Boute, R. N., & Udenio, M. (2023). Increased bullwhip in retail: A side effect of improving forecast accuracy with more data? Available at SSRN 4320911.
- Wellens, A. P., Udenio, M., & Boute, R. N. (2021). Transfer learning for hierarchical forecasting: Reducing computational efforts of M5 winning methods. *International Journal of Forecasting*.
- Weller, M., & Crone, S. F. (2012). *Supply chain forecasting: Best practices & benchmarking study*. Lancaster Centre for Forecasting.
- Yelland, P., Baz, Z. E., & Serafini, D. (2019). Forecasting at scale: The architecture of a modern retail forecasting system. *Foresight: The International Journal of Applied Forecasting*, 4(55).