Pattern Recognition 120 (2021) 108144

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Explainable boosted linear regression for time series forecasting

Igor Ilic[a], Berk Görgülü[b], Mucahit Cevik[a,*], Mustafa Gökçe Baydoğan[c]

[a] Ryerson University, Toronto, ON, Canada
[b] University of Toronto, Toronto, ON, Canada
[c] Bogazici University, Istanbul, Turkey

A R T I C L E I N F O

Article history:
Received 18 September 2020
Revised 1 June 2021
Accepted 27 June 2021
Available online 8 July 2021

Keywords:
Time series regression
Probabilistic forecasting
Decision trees
Linear regression
ARIMA

A B S T R A C T

Time series forecasting involves collecting and analyzing past observations to develop a model to extrapolate such observations into the future. Forecasting of future events is important in many fields to support decision making as it contributes to reducing the future uncertainty. We propose explainable boosted linear regression (EBLR) algorithm for time series forecasting, which is an iterative method that starts with a base model, and explains the model's errors through regression trees. At each iteration, the path leading to highest error is added as a new variable to the base model. In this regard, our approach can be considered as an improvement over general time series models since it enables incorporating nonlinear features by residual explanation. More importantly, use of the single rule that contributes to the error most enables access to interpretable results. The proposed approach extends to probabilistic forecasting through generating prediction intervals based on the empirical error distribution. We conduct a detailed numerical study with EBLR and compare against various other approaches. We observe that EBLR substantially improves the base model performance through extracted features, and provide a comparable performance to other well established approaches. The interpretability of the model predictions and high predictive accuracy of EBLR makes it a promising method for time series forecasting.

1. Introduction

Time series forecasting has important applications in various domains including energy [21], finance [31] and weather [9]. Accurate forecasts provide insights on the trends in the domain, and serve as inputs to decisions involving future events. For instance, in supply chain operations, sales and demand forecasts of the products are essential for inventory control, production planning and workforce scheduling. Accordingly, effective forecasting tools are directly linked to increased profits and reduced costs.

Quantitative forecasting methods are generally divided into two categories: general time series models and regression-based models. General time series models such as exponential smoothing and autoregressive integrated moving average (ARIMA) are typically derived from the statistical information in the historic data. On the other hand, regression models rely on constructing a relation between independent variables (e.g., features such as previous observations) and dependent variables (e.g., target outcomes). There is a wide range of regression approaches used for time series forecasting including linear regression, ensemble methods and neural networks.

While majority of the previous studies on time series prediction focus on point forecasts, many applications benefit from having probabilistic/interval forecasts that can provide information on future uncertainty. For instance, in retail businesses, probabilistic forecasts enable generating different strategies for a range of possible outcomes provided by the forecast intervals. A probabilistic forecast typically consists of upper and lower limits, and the corresponding interval can be taken as a confidence interval around the point forecasts. Standard methods such as exponential smoothing and ARIMA generate probabilistic forecasts through simulations or closed form expressions for the target predictive distribution [11]. Recent studies propose deep learning models for probabilistic forecasting that target predicting the parameters of the underlying probability distribution (i.e., mean and variance) for the next time step, and show performance improvements over standard approaches for large datasets consisting of a large number of time series [39,41].

In predictive modeling, often times the models are evaluated by measuring their prediction performance obtained using a test set based on metrics such as mean absolute error and mean squared error, disregarding the interpretability of the model predictions. However, interpretable models have certain benefits such

* Corresponding author at: Ryerson University 350 Victoria Street Toronto, ON, M5B 2K3, Canada.

E-mail addresses: iilic@ryerson.ca (I. Ilic), bgorgulu@mie.utoronto.ca (B. Görgülü), mcevik@ryerson.ca (M. Cevik), mustafa.baydogan@boun.edu.tr (M.G. Baydoğan).

https://doi.org/10.1016/j.patcog.2021.108144
0031-3203/© 2021 Elsevier Ltd. All rights reserved.

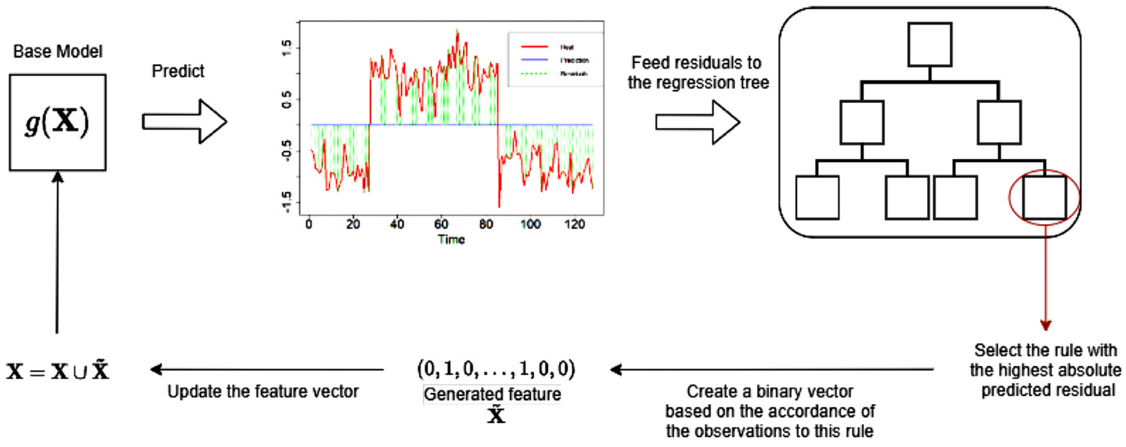

Base Model
$g(\mathbf{X})$
Predict
Time
Feed residuals to the regression tree
Select the rule with the highest absolute predicted residual
Create a binary vector based on the accordance of the observations to this rule
$(0,1,0,\ldots,1,0,0)$
Generated feature
$\tilde{\mathbf{X}}$
Update the feature vector
$\mathbf{X} = \mathbf{X} \cup \tilde{\mathbf{X}}$

Fig. 1. A visual illustration of EBLR.

as creating a trust toward the model by explicit characterization of the factors' contribution to the predictions and providing a better scientific understanding of the model. Value of model interpretability has been acknowledged in recent studies, and lead to new avenues for research [2,25].

Several recent studies on time series forecasting resort to complex deep learning architectures, which typically yield relatively accurate results when the available data is abundant. The drawbacks of such approaches include their computational burden and the black-box nature [39,41]. In this regard, linear models may provide a good trade-off between accuracy and simplicity. Specifically, linear models with simple mathematical forms are generally preferred for their interpretability and explainability of the model outcomes.

This study proposes a time series forecasting method suitable for deterministic and probabilistic forecasting that iteratively improves its predictions through feature generation based on residual exploration. Our model has two stages. In the first stage, a generic forecasting model (i.e., a base learner) is trained to obtain the base forecasts. Second stage explores the residuals (i.e., errors) of the existing model with a regression tree trained on all the available features. This tree identifies a set of rules that points to regions in the feature space with high error. The rule contributing to the error the most is used to generate a new feature and it is utilized by the forecasting model in the first stage. The iterations continue until a certain stopping criterion is met, e.g., a certain number of features are generated, the regression tree cannot make a split or forecasting model error cannot be improved further. The idea of learning interaction features based on decision trees is introduced by Friedman et al. [23]. The method proposed in this work extends this idea by generating interaction features based on the unexplored residuals. A visual representation of the proposed Explainable Boosted Linear Regression (EBLR) method is provided in Fig. 1.

The proposed method shares similar ideas with gradient boosting regression (GBR) trees. Specifically, GBR trees fit a decision tree on the residuals obtained from the fitted prediction model (or base learner), and aim to improve the prediction model by adding new decision trees to update the base prediction. At each iteration, predicted residuals are used to update the base prediction after they are multiplied with a learning rate. In other words, GBR trees use all the rules to update the prediction with a fixed learning rate. On the other hand, our approach uses the residuals to determine the highest source of errors and creates a single and interpretable feature to the fitted model to improve its predictive performance at each iteration. Compared to the use of a fixed learning rate, the base learner in the first stage is retrained to assign an appropriate weight to the introduced feature. Unlike generic boosting algorithms, our method does not sacrifice interpretability while improving the prediction performance. Moreover, our method generates fewer but more meaningful features, which leads to a more memory efficient method.

The basic time series models and other linear models rely on trends, and seasonality, and leave unexplained components as noise. Due to exponential number of potential interactions, kernel-based methods [36] are introduced to capture such interactions as well as the nonlinearity without explicitly introducing the features to the model. However, these kernels cause the model to lose its interpretability. Our proposed approach identifies interactions between features through a regression tree in the second stage, and explicitly adds this information to the model. Use of the single set of rules that contributes to the error most not only improves the predictive performance, but also allows for interpretable results. For instance, for retail sales forecasting, if there is an implicit interaction between holiday and promotion day variables that creates a higher effect than their individual effects, a new (nonlinear) variable is included in the fitted model. This variable implies that the promotion has a higher effect on the sales when applied on a holiday. In a similar manner, our approach is able to capture many polynomial interaction terms and incorporate those to the fitted model depending on the choice of the base learner (i.e., linear regression).

After generating point forecasts, the proposed model extends to probabilistic forecasting. It generates probabilistic forecasts based on the empirical error distribution which is representative of the underlying real distribution. Prediction intervals are generated by extracting quantiles through bootstrapping the residuals of the time series [20].

The rest of this paper is organized as follows. In Section 2, we provide a review of the relevant works. Section 3 introduces EBLR and its extensions along with necessary background information. Section 4 presents the numerical experiments, and Section 5 provides concluding remarks and future research directions.

2. Related works

Time series forecasting has been well studied in the literature. While the earlier studies focused on linear statistical forecasting methods such as exponential smoothing and ARIMA, highly nonlinear models including deep neural networks has been shown to perform better for various forecasting tasks [6,16]. Moreover, decision trees and their ensembles such as random forests and gradient boosted trees were frequently used for time series forecasting due to their predictive power and interpretable structure [24,43]. High performance of tree-based ensembles and strength of the simple models were highlighted in the recent M5 forecasting competition where Light Gradient Boosting Machines-based methods were consistently among the top performing models [35].

Recent studies particularly focused on deep learning and metalearning approaches, reporting substantial performance improvements for the time series forecasting task over the linear models as well as standard supervised learning approaches [33,39,41]. While these models show significant promise in generating highly accurate results especially when data is abundant, they are typically regarded as black-box models, which are not deemed as interpretable/explainable. We refer the reader to Parmezan et al. [38]'s study for a detailed overview of statistical and machine learning models for time series forecasting.

Several hybrid approaches have been considered to incorporate nonlinear relations between input and output variables. Zhang [47] assumed that each time series can be represented as a combination of linear and nonlinear components, and developed a hybrid ARIMA and artificial neural network (ANN) model for forecasting. Predictions in the model were obtained as a combination of ARIMA's forecast for the linear component and ANN's forecast for the nonlinear component. Khashei and Bijari [30] investigated the performances of hybrid forecasting models by comparing generalized hybrid ARIMA/ANN model, Zhang [47]'s hybrid ARIMA/ANN model and ANN(p,d,q) models. Their analysis with three different datasets showed that generalized hybrid ARIMA/ANN model, which aimed to find linear relations using ARIMA in the first stage and nonlinear relations using ANNs in the second stage, performed best among the three approaches. Aladag et al. [4] replaced the feed forward neural network in Zhang [47]'s model with a recurrent neural network (RNN), which lead to improvements in forecasting accuracy. Taskaya-Temizel and Casey [44] performed comparative analysis on ARIMA and ANN hybrids using nine different datasets, showing that components of the hybrid models outperformed their hybrid counterparts in five of the nine instances. They concluded that careful selection of the models to be combined is important for the performance of the hybrid models. Arunraj and Ahrens [8] proposed a hybrid model with seasonal ARIMA (SARIMA) and quantile regression where the latter was used for forecasting the quantiles rather than individual data points. Various other studies considered hybrid models constructed from variants of ARIMA models such as SARIMA and SARIMAX [17,18].

The success of hybrid approaches and the power of simple models in capturing time series characteristics were emphasized as the main outcomes of the M4 competition [34]. Smyl [42], the winner of the M4 competition, proposed a hybrid forecasting method that combines exponential smoothing with LSTM neural networks where exponential smoothing and LSTM respectively aim to capture simple patterns (e.g., seasonality) and nonlinear complex trends. This method, and in general the hybrid methods, carry similarities with EBLR. EBLR utilizes regression trees to represent important specific nonlinear features in the linear domain and combines them with the general pattern that is captured by the linear model. Therefore, it can also be treated as a hybrid method.

While the common approach for forecasting is the prediction of the expected value of a target value, understanding the uncertainty in a model's predictions can be important in different areas such as macroeconomics and financial risk management. Accordingly, many studies on forecasting focus on modeling uncertainties that lead to probabilistic forecasts. A common approach is to use quantile regression [32], while some other studies considered ensemble of learned models to generate probabilistic forecasts [3]. Recent studies employed deep neural networks to generate mean and variance parameters of the predictive distributions. Specifically, Salinas et al. [41] proposed DeepAR model, which is an autoregressive recurrent network-based global model that considers observations from different training time series to build a single probabilistic forecasting model. Rangapuram et al. [39] combined state space models with deep learning by parametrizing a linear state space model using a recurrent neural network (RNN). Wang et al. [45] integrated global deep neural network backbone with local probabilistic graphical models where global structure extracts complex nonlinear patterns, and local structure captures individual random effects. Wen et al. [46] suggested that assuming an error distribution for making probabilistic forecasting might provide inaccurate forecasts for some applications, and proposed combining RNN and convolutional neural network (CNN) with quantile regression for probabilistic forecasting. Oreshkin et al. [37] developed a deep neural architecture for univariate time series prediction based on deep fully-connected layers and residual links. Chen et al. [14] proposed a CNN-based probabilistic forecasting framework with the aim of estimating the probability density given multiple related time series. Combining different probabilistic forecasting methods, Alexandrov et al. [5] provided an extensive Python library of probabilistic time series models.

Few other studies in the literature focused on building a forecasting model through residual exploration. Aburto and Weber [1] considered a combined ARIMA and neural network model where the ARIMA model was used to model the original time series and the neural network was used to predict possible forecasting errors. The resulting forecast was taken as the summation of the predicted values by these two models. Gur Ali and Pinar [26] proposed a two-stage information sharing model for multi-period retail sales forecasting problem. In their model, the first stage estimated various variables such as calendar, seasonality and promotions through a regression analysis, and second stage extrapolated the residual time series. The resulting prediction was obtained by combining the forecasts from the first stage with the extrapolated residuals from the second stage.

3. EBLR

This section introduces our proposed framework, EBLR, for time series forecasting and nonlinear feature generation. EBLR framework consists of two stages that are applied recursively: model training and feature generation. The first stage is intuitive and utilizes well-known linear models such as linear regression, least absolute shrinkage and selection operator (LASSO) regression, or ARIMA. The second stage generates nonlinear features based on regression tree transformation. Below, we first provide necessary background on the regression trees and tree-based representation, then introduce EBLR and its extensions.

3.1. Regression trees and tree-based representation

Regression trees [13] are tree structures that recursively partitions the observation space based on some rules. These rules are greedily generated by evaluating all possible splits in the data and selecting the one that provides the highest mean squared-error (MSE) reduction in the children nodes.

Each terminal node in a regression tree can be represented by a set of rules. Due to the nature of the split formation in the trees, each terminal node refers to a hyperrectangle in the feature space – assuming that all features are numerical without loss of generality. From a probabilistic view, regression trees model mixture of Gaussian distributions [19]. Feature representations based on the tree structures are shown to provide successful results in different domains and they are sometimes referred to as hashing [48]. Similarly, each observation is represented by a binary vector based on its presence or absence in a terminal node. For instance, an observation residing in the 3*rd* node of a regression tree with 5 terminal nodes is represented as (0,0,1,0,0). This representation implicitly encodes the feature space based on the distribution of the target variable [13].

3.2. Methodology

Consider a time series dataset containing N time series of length T. Let $y_t^{(i)}$ represent the observation at time t of the ith time series and, assume that there is a $(1 \times F)$ feature vector associated with each observation $y_t^{(i)}$, represented by $X_t^{(i)}$. Moreover, $\mathbf{y}$ and $\mathbf{X}$ respectively represent the vector of all observations of size $(N \times 1)$ and the matrix that contains complete feature space of size $(N \times F)$.

In the first phase, an initial feature set of size $f \in \{1, \dots, F\}$ is selected. This feature set can contain a single feature of time information (t) or a collection of features, and is represented by an $(N \times f)$ matrix $\mathbf{X}'$. Then, a linear base learner $g(\boldsymbol{x}) = \alpha + \beta^T \boldsymbol{x}$ that maps $\mathbf{X}'$ to a $\hat{\mathbf{y}} \in \mathbb{R}^n$ vector of size $(N \times 1)$ is chosen. The base learner is trained on $(\mathbf{X}', \mathbf{y})$ pair to obtain the base model parameters (β) by minimizing a targeted loss function. Based on the prediction vector $\hat{\mathbf{y}}$ obtained from $g(\mathbf{X}')$, the residuals are calculated and represented as $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$. Note that due to the linear nature of the base learner, residuals potentially contain additional information that is related to the nonlinear patterns in the feature vectors and/or interactions.

In the second phase, the residuals obtained from the base learner are examined to model the unexplored components. Here, a regression tree that predicts $\mathbf{e}$ using $\mathbf{X}$ is trained. Each observation $e_t^i \in \mathbf{e}$ resides in a single terminal node of the regression tree. Regression trees are constructed to minimize MSE of the target values that end up in each of the terminal nodes. Therefore, when they are trained on the residuals, they essentially aim to group the errors from the same sources. That is, they discover the undiscovered, potentially nonlinear features that explains a proportion of the errors. Each terminal node in the regression tree represents an error source. Specifically, the terminal nodes are selected such that the absolute value of the target means (i.e., residuals) in the terminal nodes is the largest, i.e., terminal node with the largest error source.

The selected terminal node can be represented with a binary vector $\tilde{\mathbf{X}} \in \{0, 1\}^N$ such that $\tilde{\mathbf{X}}_i = 1$ if the observation i resides in the selected terminal node and 0 otherwise. Once this vector is generated, it is added to the current feature vector $\mathbf{X}'$, which is updated by setting $\mathbf{X}' = \mathbf{X}' \cup \tilde{\mathbf{X}}$. This process is repeated until a stopping condition is met.

3.3. Parameters

There are three sets of parameters involved in EBLR: (1) hyperparameters of the base learner, (2) hyperparameters of the regression trees and (3) stopping condition related parameters. Firstly, hyperparameters for the selected base learner can be specified based on the prior knowledge and/or through hyperparameter tuning. Since we focus on linear models such as simple linear regression and LASSO regression, only hyperparameter that should be specified is LASSO penalty which can be easily determined by cross-validation.

Secondly, EBLR requires specification of the decision tree hyperparameters, namely, tree-depth, complexity parameter or minimum observation in terminal nodes. Determining the depth of the tree is of great importance for our method because it directly determines the degree of nonlinearity (i.e., degree of complexity) for the generated features. There are two main approaches that could be used for determining the depth: pre-pruning (determining the tree depth before construction) and post-pruning (pruning the leaf nodes after construction). For our purpose, utilization of pre-pruning is challenging since the “right amount of complexity” required for each generated feature is not known apriori to tree construction, therefore, it might need cross-validation by re-constructing the regression tree for many times, which might introduce an additional complexity. Unlike pre-pruning, post-pruning based on complexity parameters provide a highly efficient pruning strategy that allows generating features of various complexities in each iteration without re-constructing the regression trees. Post-pruning requires an initial complexity parameter, η, to be specified. Our preliminary analysis show that setting η to a small value would be enough for our method to perform well.

Lastly, the parameters are specified for the stopping criteria. In this work, the number of features to be generated ($F^{\max}$) is used as the stopping criteria which essentially implies the number of iterations that EBLR runs. It is important to note that there is a trade-off between the base model selection and stopping criteria. If the selected base model is a primitive learner such as simple linear regression, then $F^{\max}$ significantly effects the degree of overfitting that might occur, and therefore it should be carefully selected. Whereas, if our base learner is a penalized method such as LASSO regression, $F^{\max}$ could be selected as a very large integer, and coefficient of penalization can be tuned to eliminate the features that cause overfitting. A pseudocode of EBLR is provided in Algorithm 1 .

Algorithm 1: Pseudocode of EBLR.

```
Input  : Input dataset $\mathcal{D}$, number of features $F^{\max}$, initial
         complexity parameter $\eta$
Output: Trained model $g$
1  Construct initial feature matrix $\mathbf{X}\prime$;
2  for $i = 1 \dots F^{\max}$ do
3      Train the base model $g$ on $(\mathbf{X}\prime, \mathbf{y})$;
4      Calculate the residuals $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$ ;
5      Train a regression tree on the residuals based on $\eta$ and
       using all features $\mathbf{X}$ ;
6      Apply post-pruning to the regression tree;
7      Extract a feature $\tilde{\mathbf{X}}$ from the terminal nodes with the
       largest absolute error ;
8      Update $\mathbf{X}\prime = \mathbf{X}\prime \cup \tilde{\mathbf{X}}$;
9  end
10 Update feature space to include all raw features $\mathbf{X}\prime = \mathbf{X}\prime \cup \mathbf{X}$ ;
11 Train the base model $g$ on $(\mathbf{X}\prime, \mathbf{y})$;
12 Return $g$;
```

3.4. Complexity analysis

We conduct a worst-case theoretical complexity analysis for EBLR. The complexity of the method is dictated by the feature generation phase due to simple and linear nature of the base regressors. We take B as the complexity of the base regressor and focus on the feature generation phase.

Consider a decision tree of depth d constructed on a time series database of size $N \times T$. Then, in the vertical format, it corresponds to NT number of observations. Since each observation requires d comparisons, the worst-case complexity of the feature generation is $O(NTd)$. Then, the overall complexity of each iteration becomes $O(NTd + B)$. This process is repeated for $F^{\max}$ times, which yields $O(F^{\max}(NTd + B))$ complexity. Note that the complexity of EBLR is similar to both gradient boosting regression and random forest, which is $O(F^{\max}NTd)$ assuming $F^{\max}$ many iterations.

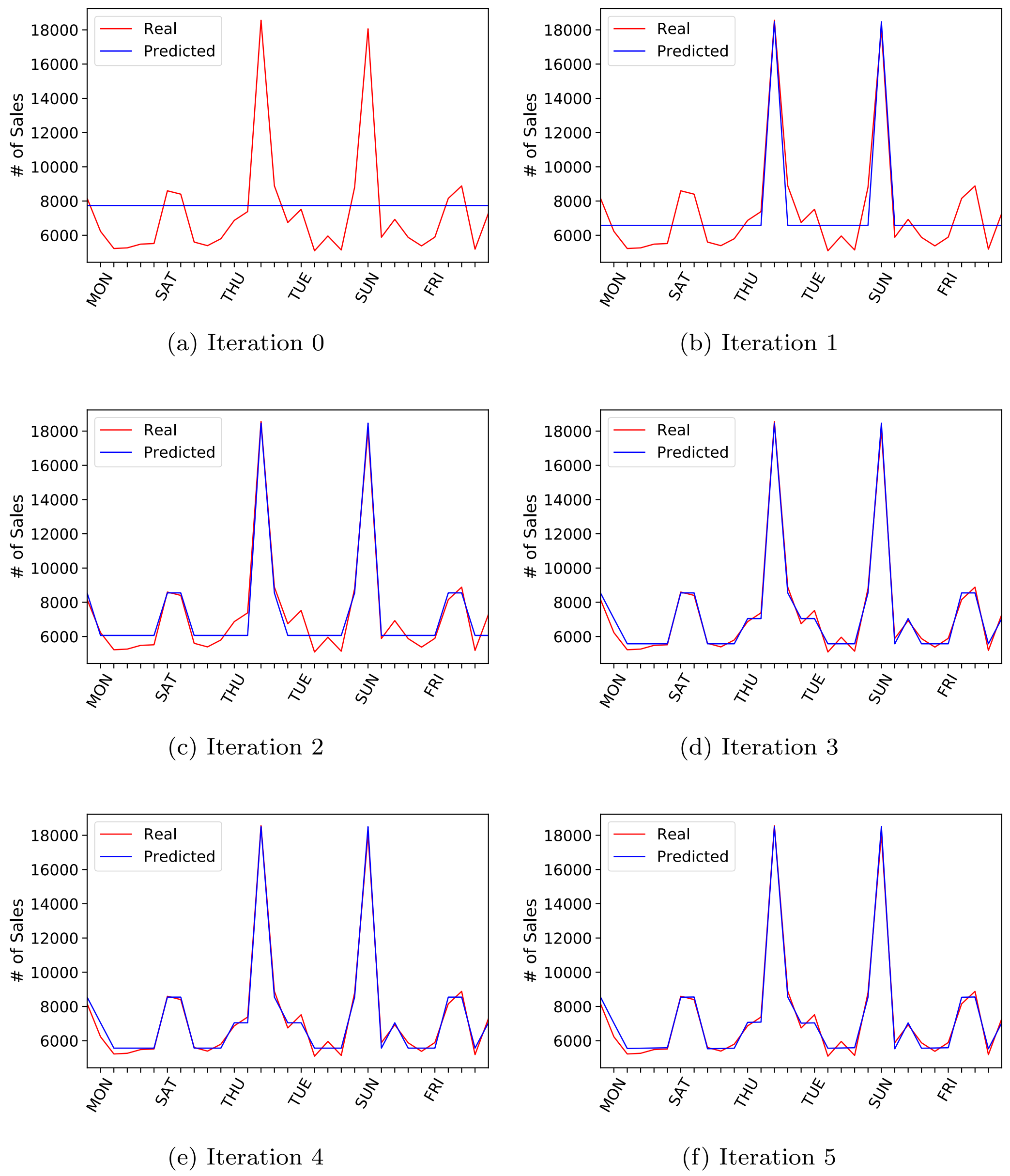


Fig. 2. A sample synthetic time series and predictions of EBLR illustrating the learning process in each iteration.

3.5. Illustration

We illustrate EBLR on a simple example to provide a better understanding. Consider a time series coming from the following model, initialized with $y_0 = y_1 = 0$:

$$y_t = -0.4y_{t-1} + 0.5y_{t-2} + 5500 * isWeekend * isPromotion$$
$$+ 1500 * isPromotion + 3000 * isWeekend + \mathcal{N}(5000, 150).$$

In this example, in addition to the auto-regressive terms, there are two factors affecting the sales: day of the week and promotion. Also, note that the interaction of these features are also important, and have certain heterogeneous effects. For example, there is a significant difference in the effect of promotion on weekdays and weekends.

Fig. 2 illustrates the predictions obtained throughout the EBLR iterations. In Fig. 2a, red line shows the original time series ($\mathbf{y}$) and blue line represents our initial guess ($\hat{\mathbf{y}}$), which is the mean of the time series (i.e., we start with an empty feature set). Then, the residuals are calculated (the difference between $\mathbf{y}$ and $\hat{\mathbf{y}}$) and a regression tree is fitted to the residuals using the data with complete features ($\mathbf{X}$) presented in Fig. 3. From the terminal nodes of the tree, the node with the largest mean absolute value is selected (i.e., terminal node (4)). Based on this node, a new feature ($\tilde{\mathbf{X}}$) is created such that it takes value 1 if the observation ends up in the selected terminal node and 0 otherwise. This newly generated feature corresponds to the rule of (Is Weekend, Yes) & (Is Promo, Yes). Then, this new feature is added to the feature matrix $\mathbf{X}'$ and the base regressor is retrained. New predictions ($\hat{\mathbf{y}}$) are illustrated with the blue line in Fig. 2b. Note that some of the patterns are captured, however, it is still not enough to discover the underlying model.

This feature generation and retraining phases are repeated for five iterations (each iteration is illustrated in Fig. 2), which provides a good fit to the original time series. Another observation made here is that the early features provide major fixes in the model which are followed by minor fixes that are generated as more and more features are added to the model. This also illustrates that the selection of $F^{\max}$ parameter is important, and if it is set to a large value, EBLR can overfit to the data.

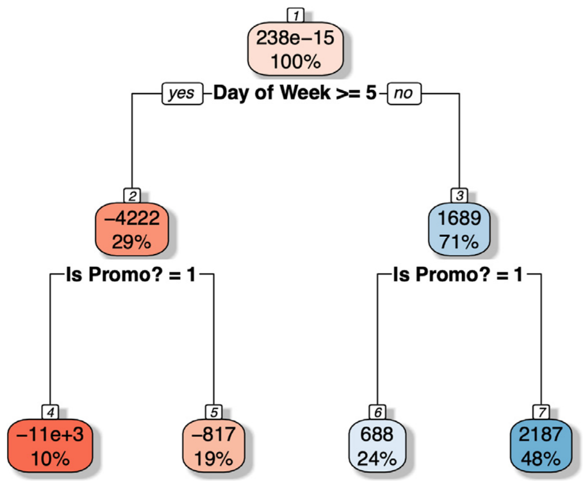


Fig. 3. Regression tree trained on the residuals in the first iteration. The top and bottom values in each node respectively illustrates the mean value of the observations and the percentage of the observation falls into the node.

3.6. Feature generation

Feature generation is an essential part of the EBLR. The features that are generated from the terminal nodes of the decision tree aim to capture the nonlinear relations and interaction effects. Note that these generated features do not have any specific connection to the associated learner, and can be used in other prediction models to improve their performance. From this aspect, our method can also be considered as a feature generation method.

In a dataset with F features, there is a large number of possible interaction terms (i.e., $2^F - (F+1)$). Note that each generated feature maps to a set of consecutive decision rules, and the variables included in the same set of rules represent a possible interaction in between. In addition to using these features as is, this kind of intuitive relation also allows us to consider these interaction terms in the prediction models, and to include them from the beginning for more complex learners.

Here, only the terminal nodes of the decision trees are explored, and the node with the largest mean absolute value is selected to generate the features. However, it is possible to apply alternative approaches such as exploring all nodes rather than terminal nodes and selecting many nodes rather than only a single one in each iteration. These approaches might be more prone to overfitting, however, they speed up the feature generation process.

3.7. Interpretability

Interpretability is a desired attribute for a prediction method since it brings transparency to the prediction, and it increases the overall reliability of the algorithm. Therefore, prediction algorithms that lack interpretability might not be preferred in some applications or can create hesitancy about making a decision based on the observed results. The biggest advantage of EBLR is its ability to generate nonlinear features that are also interpretable. The reason behind its interpretability is that, associated with each generated feature, there is a set of decision rules. These decision rules explicitly shows what this feature represents. Consider the first feature generated in the example provided in Section 3.5. This feature corresponds to a decision rule of {(Is Weekend, Yes) & (Is Promo, Yes)}, which explicitly implies that promotion has a different effect on the weekends than weekdays. When this feature is added to the model, the corresponding coefficient shows the magnitude and direction of the effect. For instance, in the example provided in Section 3.5, it has a positive effect, which implies that promotions that are made on weekends are more effective than the weekdays.

Note that even in this simple example with two features, there are 14 possible interaction terms (7 days $\times$ 2 promo) and EBLR can easily identify that there is no difference among weekdays or weekends, and it is enough to consider whether a day is a weekday or weekend, which reduces the total number of interactions to be considered to four. With an increasing number of features in the data, identification of these interpretable features gets exponentially more difficult, however, EBLR can easily handle any number of features thanks to the efficient construction and interpretable structure of the decision trees.

3.8. Extension to probabilistic forecasting

Section 3.2 describes the proposed algorithm for forecasting a single value (i.e., mean). The algorithm can also be extended to probabilistic forecasting. We carry out this task by constructing prediction intervals on the mean estimation based on the empirical error distributions. Specifically, assume that we construct a prediction interval on the $\hat{y}_i = g(X_i)$, and let $E(\cdot)$ be the distribution function of the errors. Then, the $\alpha\%$ prediction interval is constructed as $\hat{y} \pm E^{-1}(\alpha/100)$. This interval can be constructed for any given α based on the required confidence level.

4. Numerical experiments

This section provides a set of experiments to demonstrate the prediction performance of EBLR in both deterministic and stochastic settings. Before presenting the numerical results, we first provide the experimental setup, which includes the information regarding datasets, performance metrics, model settings and performance evaluation.

4.1. Experimental setup

EBLR is created using `scikit-learn` and `r-forecast` packages and is provided in [28]. The experiments are conducted on a 2.7 GHz dual-core i5 processor with 8GB of RAM, using the Python programming language.

4.1.1. Performance metrics

Three performance metrics are considered in our analysis [41]. Point forecasting performance is compared in terms of the normalized root mean squared error (NRMSE) and the normalized deviation (ND), which are provided as follows:

$$\mathrm{NRMSE}(y,\hat{y}) = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}}{\frac{1}{N}\sum_{i=1}^{N}|y_i|}, \qquad \mathrm{ND}(y,\hat{y}) = \frac{\sum_{i=1}^{N}|\hat{y}_i - y_i|}{\sum_{i=1}^{N}|y_i|} \tag{1}$$

For the probabilistic forecasting, the models are evaluated on the weighted scaled pinball loss at the 0.05, 0.25, 0.50, 0.75 and 0.95 quantiles. To combine these five metrics into one general metric, the mean of the quantile losses is also reported. For any quantile ρ, the weighted scaled pinball loss is calculated as follows:

$$\mathrm{WSPL}_\rho(y,\hat{y}^\rho) = \frac{\sum_{i=1}^{N}\max\{\rho(y_i - \hat{y}_i^\rho), (1-\rho)(\hat{y}_i^\rho - y_i)\}}{\sum_{i=1}^{N}|y_i|} \tag{2}$$

4.1.2. Datasets

Three datasets are used in the experiments: synthetic, Rossman [40] and Turkish electricity[1]. Summary information regarding these datasets can be found in Table 1.

[1] https://seffaflik.epias.com.tr/transparency/tuketim/gerceklesen-tuketim/gercek-zamanli-tuketim.xhtml

Table 1
Descriptive information and statistics on the datasets.

	Synthetic	Rossmann	Turkish electricity
# features	1	6	2
# time covariates	1	53	31
# time series	1	100	1
# avg data points / time series	2048	918.1	8760
Granularity	Daily	Daily	Hourly
Seasonality	None	None	Daily

Synthetic dataset. The first dataset is a synthetically generated and mimics a simple version of daily sales of a retail store (see Fig. 2 for data samples).

Rossman dataset. The Rossmann sales data is a feature-rich dataset, containing features related to the date, number of customers, holiday-specific information, promotional information, and store closures. The sales of an individual store is set to the target variable, and the customer covariate was removed because it is related to the target variable, and requires to be forecasted. The date feature is further decomposed into the day of the week, day of the month, month of the year, and year features for enriching the feature space.

Turkish electricity dataset. This dataset contains electricity consumption values in Megawatt-hours (mwh) for Turkey's most populous city, Istanbul. The target variable is created by interpolating the total electricity consumption in Turkey to Istanbul (based on monthly factors). This dataset is decomposed into hourly consumption rates with daily seasonality. Then, the date is decomposed into the day of the week, month and year variables. Moreover, the data contains daily high and low temperatures of Istanbul, which is an important determinant of electricity consumption.

4.1.3. Model settings

In the point forecasting experiments, EBLR is compared against two naive methods (a baseline method (Mean) and linear regression (LR)), statistical models (ARIMAX), ensemble methods (gradient boosting regressor (GBR) [22], random forest regressor (RF) [12]) and deep learning-based approaches (LSTM, GRU and four representative models included in GluonTS package, namely, DeepAR [41], DeepState [39], MQRNN [46] and MQCNN [46]). LASSO penalty is selected based on 5-fold cross-validation. For ARIMAX, a step-wise algorithm is used to tune the model (see [27] for details). In the case of the seasonal Turkish electricity dataset, as the dataset is inherently seasonal, the seasonal parameters are included in the ARIMAX model as well [29]. For GBR and RF, a single parameter setting is utilized without performing hyperparameter tuning, and this setting is used consistently across all datasets. The parameters for GBR and RF can be found in Table 2.

The parameter setting for each dataset for EBLR is also specified in Table 2. Only a small number of parameters were selected in the synthetic dataset since there were only a few underlying features. In the Rossmann dataset, 50 features were arbitrarily chosen to be learned. When this setting was reused for the Turkish Electricity dataset, it was found that the model was still learning, therefore the number of features was doubled. All the initial complexity parameters where chosen such that they were significantly small.

For LSTM and GRU, we use two hidden layers with 32 and 16 hidden units and a fully connected dense layer. During model training, first a cross-validation is performed, and a model is fitted once the validation accuracy does not improve for five consecutive epochs.

From the deep learning-based approaches considered in our analysis, we train only LSTM and GRU for all three datasets. This is mainly because deep learning models typically perform well for large datasets. In particular, models such as DeepAR, DeepState, MQRNN and MQCNN are mainly developed for global time series forecasting and designed to take advantage of data streams coming from multiple time series. Among the datasets considered in our analysis, only Rossmann contains multiple time series. Accordingly, we train DeepAR, DeepState, MQRNN and MQCNN models for the Rossmann dataset by adopting the global time series forecasting approach, that is, by using all 100 time series together during the training phase. In this regard, these models have competitive advantage over the other models considered here as they are trained with a larger dataset. Nevertheless, we consider DeepAR, DeepState, MQRNN and MQCNN models for benchmarking purposes in our analysis. In addition, we perform hyperparameter tuning for DeepAR and DeepState models to examine the impact of the tuned parameters on deep learning model performances. Specifically, we adopt the Tree Parzen Estimator approach in conducting the hyperparameter tuning experiments, and run 50 trials over 100 epochs for each algorithm considering a predetermined range of values for the parameters provided in Table 2 [10]. We did not perform a specific hyper parameter tuning for the lookback (i.e., context_length) parameter for DeepAR and DeepState as the implementations in GluonTS package internally selects this parameter based on the frequency of the data set [7]. That is, in the case of Rossmann dataset, lookback windows are chosen to best to fit the data with daily frequencies.

In the probabilistic forecasting experiments, EBLR was compared against both ensemble methods (i.e., RF and GBR) as well as deep learning methods. In order to generate probabilistic forecasting results, the loss function was adjusted in the gradient boosted regressor model to use quantile loss, and a model was trained for each quantile. In addition, in RF, the quantiles were collected based on the individual regressors. The other models use a more naive forecasting approach by training prediction intervals based on the training residuals. All the experiments were repeated with the same feature setups, and performance evaluation is done using the 5%, 25%, 50%, 75% and 95% quantile predictions. As expected, it is observed that all of the 50% quantiles perform similarly to the ND scores. LSTM and GRU models were not considered for probabilistic forecasting as their native implementations were not designed for this task. On the other hand, DeepAR, DeepState, MQRNN and MQCNN models were specifically designed for generating probabilistic forecasts, and their predictions were compared against other approaches.

4.1.4. Performance evaluation

For each dataset, we provide the number of tests considered when reporting the forecasting performance in Table 3. Both synthetic and Turkish electricity datasets consist of a single time series. Accordingly, we consider an augmented out of sample comparison approach [29] using 25 tests for testing the performance over synthetic and Turkish electricity datasets. More specifically, this evaluation approach utilizes an initial training horizon and a forecast horizon such that the forecast horizon immediately follows the training horizon. For each replication, the models are trained on the training data and tested on the test data. Then, test data is included in the training data and test data is updated to the subsequent part that has not been observed by the learning methods. We employ this comparison method for two main reasons: firstly, it provides a more realistic way of comparing different approaches where the same observations are not only used single time, but they accumulate over time. Secondly, it simplifies and speeds up the re-training process especially for deep learning-based approaches. Note that initial training horizon can be obtained after determining the forecast horizon and the number of tests. For instance, for the synthetic dataset, if the number of tests to be performed is 25 and the forecast horizon is 14 days, then the data coming from last $25 \times 14 = 350$ days can be used for testing

Table 2
The hyperparameter settings used in the experiments for all the employed models.

Model	Hyperparameters
EBLR	*# of features*: {synthetic: 5, Rossmann: 50, Turkish Electricity: 100}, *initial complexity parameter* (for all datasets): 0.001
RF	*# of trees*: 100, *splitting criterion*: MSE, *max depth*: ∞
GBR	*# of trees*: 100, *splitting criterion*: Friedman MSE, *max depth*: 3, *loss function*: Least Squares Regression, *learning rate*: 0.1
LSTM, GRU	*hidden units*: {32,16}, *lookback*: {Synthetic: 30 days, Rossmann: 60 days, Turkish Electricity: 336 h}, other parameters were set to default
DeepAR, DeepState MQCNN, MQRNN	default parameters in GluonTS [5]
DeepAR-tuned	*learning rate*: 0.0053, *batch size*: 32, *# of layers*: 4, *# of cells*: 20, *cell type*: gru, *gradient clip*: 10.933, *dropout rate*: 0.098
DeepState-tuned	*learning rate*: 0.0074, *batch size*: 32, *# of layers*: 1, *# of cells*: 60, *cell type*: gru, *gradient clip*: 6.271, *dropout rate*: 0.145

Table 3
Performance evaluation settings for each dataset.

	Synthetic	Rossmann	Turkish electricity
Forecast horizon	14 days	28 days	14 h
# of tests	25	100	25

and the data for initial $2,048 - 350 = 1,698$ days can be used for training.

Rossmann dataset contains 100 time series, which were randomly sampled from 1115 available time series (see Table 1). As such, we train separate models for each one of the 100 time series in Rossmann dataset by keeping the last 28 days as the test set and the remaining portion as the training set. Then, we report the average of the performance values (e.g., ND and NRMSE) over predicting a single forecast horizon (hence 100 tests for this case).

4.2. Improvement over base regressors

This section presents an experiment on EBLR's contribution to improving the prediction performances of linear regression, LASSO regression and ARIMA methods. Table 4 illustrates the average forecasting performances of these methods with and without EBLR on synthetic dataset and Rossmann dataset. From these results, it can be observed that utilizing EBLR provides significant decrease in the NRMSE and ND for both linear regression, LASSO and ARIMA methods, especially for the synthetic dataset. Note that, as these base regressors are linear, they fail to capture the interaction effect in the synthetic data whereas incorporating these base regressors in EBLR allows the generation of interaction features which reduces the error significantly. Since EBLR provides similar results with all base regressors, we utilize LASSO regression as the base regressor in the subsequent experiments with the aim of preventing potential overfitting.

4.3. Point forecasting

The point forecasting prediction performances over three datasets are provided in Table 5, and the predictions of four important regressors on single time series from each of these three datasets are illustrated in Fig. 4. The first set of experiments are run on a synthetic dataset and the results are presented in the second and third columns of Table 5. From these results, three specific observations can be made: (1) EBLR significantly improves the performance of the linear regression (decrease NRMSE from 0.327 to 0.047) by introducing 5 additional features that captures the important interactions, (2) EBLR outperforms all primitive regression algorithms including ARIMAX, linear regression and naive baseline, (3) EBLR provides comparable performances to the ensemble methods.

Secondly, we focus on the Rossmann dataset where the results are presented in the fourth and fifth columns of Table 5. In the Rossmann dataset, EBLR performs slightly worse than GBR, but once again outperforms ARIMAX, linear regression and naive baseline models. When EBLR is compared to RF, a more interesting relationship is revealed. EBLR has a lower NRMSE whereas RF has a lower ND. This means that EBLR's predictions tend to be more consistent. RF on average makes more accurate predictions, however, if it fails to predict certain region, it provides much worse predictions than EBLR's. LSTM and GRU models do not perform well for the Rossmann dataset. Note that, to be consistent with other methods, we trained individual LSTM/GRU models for each time series in the Rossmann dataset. As these models may substantially benefit from large datasets, their performances can be improved by adopting global forecasting approach and training a single model by using all the time series together. In addition, a detailed hyperparameter tuning may improve the performances of these models [15]. Overall, we observe that both with and without hyperparameter tuning, DeepAR performs the best for the Rossmann dataset by slightly outperforming GBR in terms of NRMSE and ND. On the other hand, MQCNN and MQRNN performs worst among all the prediction models. It is important to note that DeepAR, DeepState, MQCNN and MQRNN are trained by using a global forecasting approach, hence they have an unfair advantage over other models.

Finally, the last two columns of Table 5 demonstrates the results on Turkish electricity dataset, where, consistent with the previous datasets, EBLR outperforms ARIMAX, linear regression, and naive baseline models. In addition, EBLR also outperforms GBR, but not RF. We note that LSTM and GRU models, which were trained with the hyperparameters used for a similar electricity dataset in Ilic et al. [29]'s study, provide comparable results to other models. The improved performance of LSTM and GRU models for Turkish electricity dataset compared to other two datasets can also be attributed to inherent seasonality in this dataset, which makes it a relatively easier forecasting task.

We observe that EBLR provides substantial improvement to the linear regression by incorporating additional features. Moreover, even though EBLR is completely interpretable, it provides comparable performance to the state-of-the-art ensemble and deep learning-based approaches.

4.4. Probabilistic forecasting

This section presents probabilistic forecasting results on the three datasets. The first set of results focus on the synthetic dataset for which the summary statistics are presented in Table 6a. Overall, EBLR shows similar performance to GBR and outperforms all other competitors. EBLR performs well in terms of capturing the tails as well as the median predictions. This suggests that EBLR extends to probabilistic forecasting successfully.

Secondly, the performances of the selected methods are evaluated on the Rossmann dataset (Table 6b). Here, EBLR performs worse than GBR, however, it outperforms all other generic methods. One important observation is the relatively good performance of EBLR over GBR in extreme quantiles (e.g., 0.95 quantile). This indicates that EBLR accounts for unlikely outcomes better than GBR.

Table 4
Point forecasting results for linear regression, LASSO regression and ARIMA methods with their EBLR versions.

		Linear Regression		LASSO		ARIMA	
		w/o EBLR	w EBLR	w/o EBLR	w EBLR	w/o EBLR	w EBLR
Synthetic	NRMSE	0.3265	0.0471	0.3264	0.0472	0.3383	0.0471
	ND	0.2474	0.0383	0.2473	0.0384	0.2626	0.0383
Rossmann	NRMSE	0.1731	0.1528	0.1750	0.1543	0.1754	0.1613
	ND	0.1252	0.1085	0.1270	0.1088	0.1284	0.1164

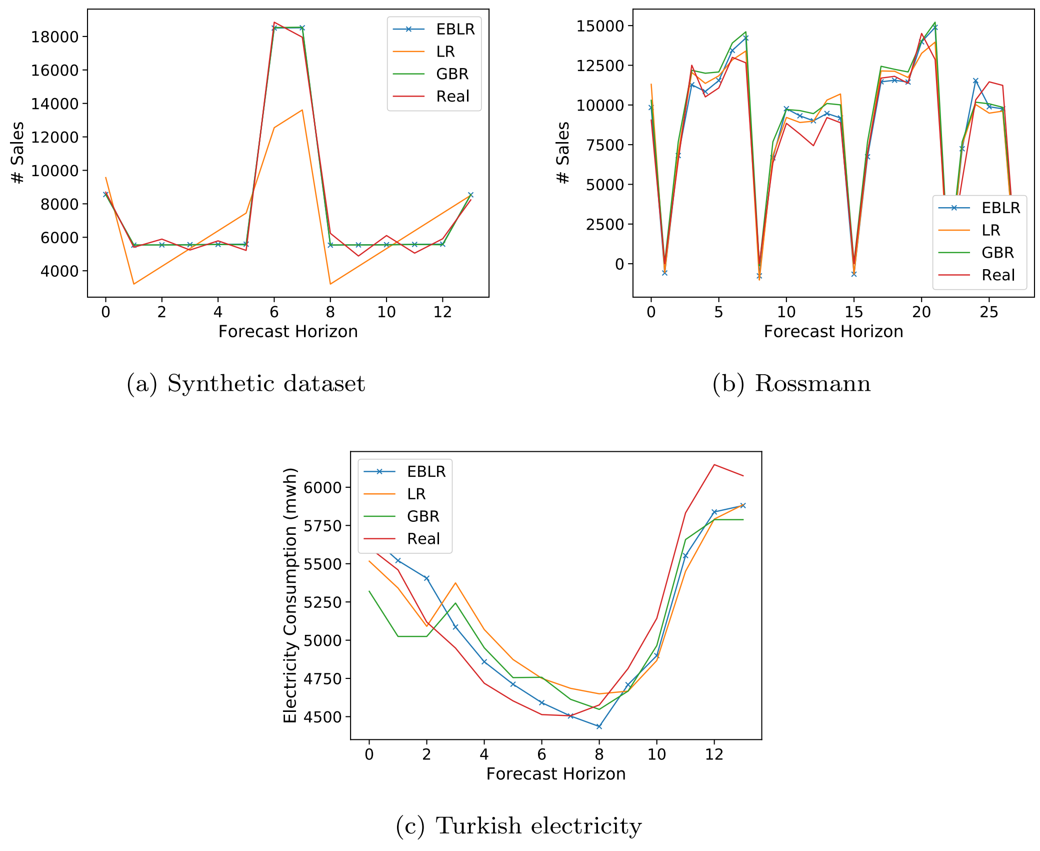


Fig. 4. Three different point forecasting tests of EBLR, linear regression (LR), gradient boosting regressor (GBR). The order (from left to right, top to bottom) include a 14 period forecast with respective predictions, a 28 period sales forecast, and a 14 h electricity consumption forecast.

Among the deep learning-based probabilistic forecasting methods, we note that DeepAR performs the best. In comparison to others, DeepAR with tuning (DeepAR-tuned) outperforms any other model however, without tuning (DeepAR) performs similarly with GBR. Specifically, the average WSPL across the considered quantiles for DeepAR-tuned and DeepAR are respectively 0.0290 and 0.0320. This shows that without costly parameter tuning, DeepAR performs slightly worse than GBR in terms of average WSPL, and marginally better than EBLR. As well, all this overhead comes at the cost of interpretability since DeepAR is regarded as a black-box model.

Lastly, Table 6c illustrates the numerical results for different methods over the Turkish electricity dataset. These results show that, in contrast to the Rossman dataset, EBLR outperforms GBR at all quantiles. This could be attributed to GBR overfitting the dataset and starting to include noise from training data into the model. EBLR is able to outperform the baseline model as well as the ARIMAX model. A unique observation is that, since the ARIMAX model is given extra seasonal regressors in this case, it is able to properly formulate the auto-regressive nature of electricity consumption. On the other hand, EBLR is able to incorporate this seasonality without having these explicit auto-regressive features.

We also provide an illustration of the predictions for EBLR, GBR, ARIMAX and DeepAR methods for a sample time series from the Rossmann dataset in Fig. 5. We note that all four models are able to generate conformal predictions, and DeepAR provide the lowest level of uncertainty in the predictions for this example.

4.5. Model interpretability

This section focuses on the explainability by illustrating the interpretable nonlinear features generated by EBLR for these datasets. It is illustrated in Section 3.5 that EBLR is capable of discovering the interaction effects. For example, in the process of learning the underlying model of the synthetic dataset, the following features are learned:

1. Is the day a weekend with a promotion?
2. Is the day a weekend without a promotion?
3. Is the day a weekday with a promotion
4. Is the day a weekday without a promotion?

This result aligns well with the synthesized features. First, EBLR learns how weekends deal with promotions, and then it determines how weekdays deal with promotions. This can clearly be

Table 5
Point forecasting results over all three datasets for EBLR, RF, GBR, ARIMAX, linear regression, naive baseline and deep learning-based methods (bold faced entries show the results for the best performing model).

Model	Synthetic		Rossmann		Turkish Electricity	
	NRMSE	ND	NRMSE	ND	NRMSE	ND
EBLR	**0.047**	**0.038**	0.153	0.109	0.043	0.035
Baseline	0.478	0.301	0.588	0.416	0.117	0.102
Linear Regression	0.327	0.247	0.175	0.127	0.053	0.042
ARIMAX	0.338	0.263	0.175	0.128	0.050	0.030
RF	**0.047**	**0.038**	0.167	0.102	**0.030**	**0.023**
GBR	**0.047**	**0.038**	0.151	0.099	0.058	0.047
LSTM	0.405	0.197	0.492	0.299	0.049	0.037
GRU	0.405	0.196	0.351	0.206	0.057	0.043
DeepAR	-	-	0.149	0.099	-	-
DeepAR-tuned	-	-	**0.139**	**0.091**	-	-
DeepState	-	-	0.234	0.172	-	-
DeepState-tuned	-	-	0.184	0.128	-	-
MQCNN	-	-	0.366	0.232	-	-
MQRNN	-	-	0.618	0.433	-	-

seen in Fig. 2, where the strong weekend features are learned first, and then the weekday features are learned. As well, after these four features are learned, the algorithm is terminated since the regression tree cannot find a split.

Similarly, in the Rossmann dataset, a key learned feature is “if the day is a Monday”, “without any promotions”, and “no school holiday”. This feature contributes negatively to the prediction which is intuitive to understand, e.g., people typically do not shop on Mondays unless there is a special event. Other contributing features are tied to the relations between the month of the year, promotional activity, and school holidays. This is easily extracted through EBLR, which provides valuable insights about the data.

The progression of EBLR’s learning process can be seen through plotting the NRMSE against the number of features. A sample learning curve has been plotted for a particular Rossman store in Fig. 6a.

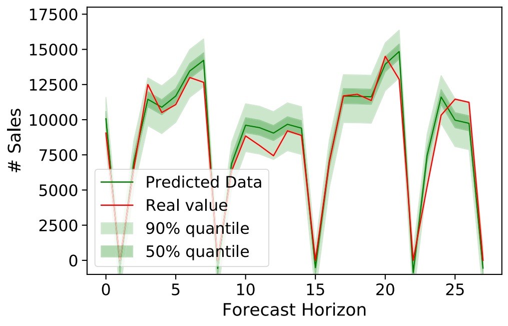
Feature importance scores are generated based on these rules, combined with the learning curve. For each rule that is created, there is a change in the residual error, Δe. Earlier learned features decrease the residual error substantially, whereas fine-tuning features are not deemed as important. This error is allocated to all boolean decisions in the feature. For example, when EBLR first learns the feature (Is Weekend, Yes), Δe_1 is assigned to each feature for *isWeekend* and *isPromo*. This is done for each feature generated, and then all the scores are added together and normalized. For example, in the synthetic dataset, as discussed in Section 3.5, every generated feature consists of a combination of *isPromotion* and *isWeekend*. Since these features exist in all the rules, as well as the only features that EBLR utilizes, they both receive an equal feature importance score of 0.5.

In Fig. 6a, there are initially a few features that drastically improve the model. Then, the model learns more complex nonlinear features to continue learning. At each iteration, there is a change in how the underlying baseline linear model performs. By utilizing the proposed feature importance scoring technique, the most contributing features in the generated rules are extracted. A plot of the feature importance scores of the top 10 most important features in the Rossmann dataset is provided in Fig. 6b.

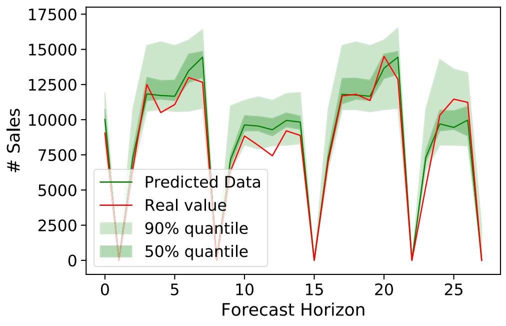
For the Turkish electricity dataset, there is a wider spread of feature importance as observed in Fig. 7b. The two most important features are the daily high and low temperatures, followed by information about the hour of the day and if the day was a Sunday. Together, these feature importance scores imply that electricity usage is highly dependent on temperature and the hour of the day. EBLR cares about knowing if the day is a Sunday or not, which means Sundays behave differently than the remaining days of the

Table 6
Probabilistic forecasting results for the five specified quantiles and their combined average (bold faced entries show the results for the best performing model).

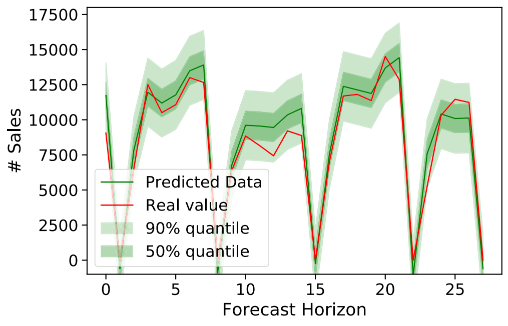
(a) Synthetic dataset						
Model	WSPL(ρ)					
	0.05	0.25	0.50	0.75	0.95	average
EBLR	**0.0051**	0.0157	**0.0192**	0.0153	**0.0048**	**0.0120**
MEAN	0.0397	0.0848	0.1504	0.1829	0.0976	0.1111
ARIMAX	0.0337	0.0986	0.1246	0.1175	0.0473	0.0843
RF	0.0168	0.0186	**0.0192**	0.0187	0.0169	0.0180
GBR	0.0053	**0.0156**	**0.0192**	**0.0152**	**0.0048**	**0.0120**
(b) Rossmann						
Model	WSPL(ρ)					
	0.05	0.25	0.50	0.75	0.95	average
EBLR	0.0185	0.0466	0.0539	0.0409	0.0140	0.0348
MEAN	0.0626	0.2032	0.2078	0.1575	0.0556	0.1373
ARIMAX	0.0195	0.0538	0.0627	0.0506	0.0182	0.0410
RF	0.0226	0.0421	0.0527	0.0429	0.0144	0.0349
GBR	0.0161	0.0401	0.0474	0.0376	0.0162	0.0315
DeepAR	0.0143	0.0398	0.0497	0.0411	0.0149	0.0320
DeepAR-tuned	**0.0142**	**0.0370**	**0.0453**	**0.0362**	**0.0126**	**0.0290**
DeepState	0.0223	0.0635	0.0859	0.0819	0.0445	0.0596
DeepState-tuned	0.0210	0.0500	0.0642	0.0596	0.0348	0.0459
MQCNN	0.0644	0.1054	0.1159	0.0996	0.0522	0.0875
MQRNN	0.0500	0.1661	0.2167	0.1958	0.0923	0.1442
(c) Turkish electricity						
Model	WSPL(ρ)					
	0.05	0.25	0.50	0.75	0.95	average
EBLR	0.0049	0.0149	0.0177	0.0130	**0.0035**	0.0108
MEAN	0.0123	0.0395	0.0509	0.0353	0.0120	0.0300
ARIMAX	0.0075	0.0158	0.0148	0.0131	0.0055	0.0113
RF	**0.0036**	**0.0089**	**0.0105**	**0.0089**	0.0039	**0.0072**
GBR	0.0089	0.0211	0.0226	0.0179	0.0079	0.0157



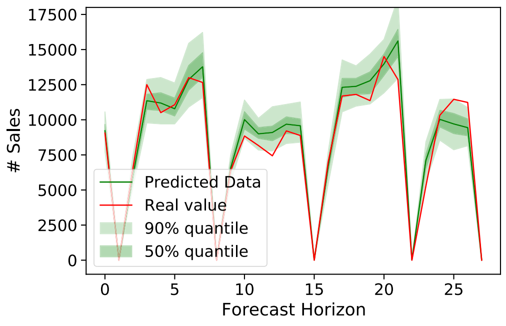
(a) EBLR



(b) GBR

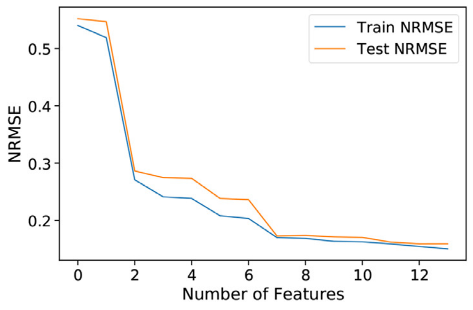


(c) ARIMAX

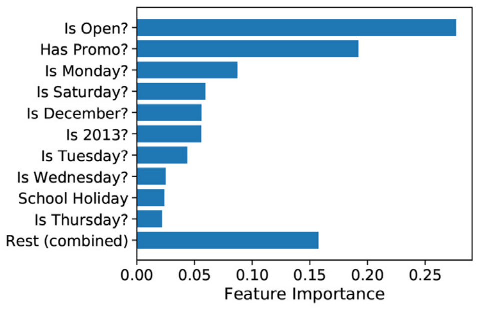


(d) DeepAR-tuned

Fig. 5. Four different probabilistic forecasting interval results on a 28 day forecast horizon for a selected Rossmann store. Each forecast horizon contains the median forecast (dark green line), the 50% forecast interval (medium green fill), and 90% forecast interval (light green fill). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

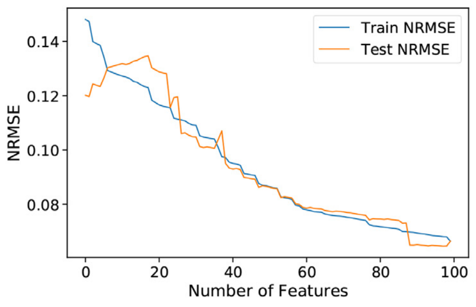


(a) Sample Learning Curve

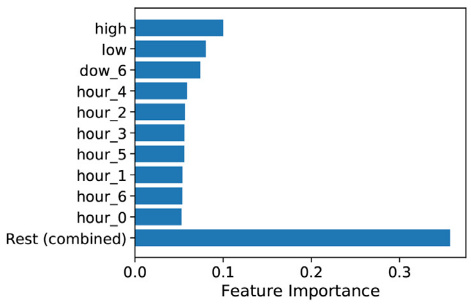


(b) Top 10 Feature Importance

Fig. 6. (Left) EBLR's improvement in NRMSE for a sample Rossmann sales store, as more features are generated and added to the base linear learner. (Right) The top 10 important features in all of the Rossmann stores, organized from the most important to the least.



(a) Sample Learning Curve



(b) Top 10 Feature Importance

Fig. 7. (Left) The NRMSE learning curve compared to the number of features in a the Turkish electricity dataset. (Right) The top 10 most importance features in the Turkish electricity dataset, organized from the most important to the least.

week. As well, there is an even distribution of feature usage in the Turkish electricity dataset, compared to a few key features in the Rossmann dataset.

5. Conclusion

Through the probabilistic and point forecasting experimentation, it is evident that EBLR is a strong boosting algorithm. While there was no clear separation between the EBLR, GBR, and RF, it is clear that EBLR is much simpler than the other two ensemble methods. EBLR consists of simple binary features, compared to the complexity of storing numerous decision trees. On top of this, the information lost by only storing these relevant binary features is minimal. Our numerical analysis with various (black-box) deep learning models reveal that EBLR is able to perform similarly to the best models from this group.

EBLR generates simple binary features through a two-step process. First, a simple baseline model is fitted to a training dataset. The residuals are extracted and passed into a feature generating regression tree. This regression tree extracts the largest source of error in the form of an interpretable feature which is passed back into the baseline learner. This process is repeated until a stopping condition is met. By learning in this manner, EBLR has inherent interpretability baked into the method itself.

EBLR is extended to probabilistic forecasting by utilizing the empirical distribution of the residuals. Quantiles are determined from this distribution, and used in constructing prediction intervals. While this was able to yield strong results in the three provided experiments, future work should focus on different ways to generate these prediction intervals. By generating prediction intervals in this manner, there are constant prediction intervals across all points. In reality, some specific points are more difficult to predict than others. Some potential research paths include using a quantile-based linear learner or extracting more information from the feature generating decision trees.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] L. Aburto, R. Weber, Improved supply chain management based on hybrid demand forecasts, Appl. Soft Comput. 7 (1) (2007) 136–144.
[2] A. Adadi, M. Berrada, Peeking inside the black-box: a survey on explainable artificial intelligence (XAI), IEEE Access 6 (2018) 52138–52160.
[3] A. Ahmed Mohammed, Z. Aung, Ensemble learning approach for probabilistic forecasting of solar power generation, Energies 9 (12) (2016) 1017.
[4] C.H. Aladag, E. Egrioglu, C. Kadilar, Forecasting nonlinear time series with a hybrid methodology, Appl. Math. Lett. 22 (9) (2009) 1467–1470.
[5] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D.C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, et al., GluonTS: probabilistic and neural time series modeling in python, J. Mach. Learn. Res. 21 (116) (2020) 1–6.
[6] I. Alon, M. Qi, R.J. Sadowski, Forecasting aggregate retail sales: a comparison of artificial neural networks and traditional methods, J. Retailing Consum. Serv. 8 (3) (2001) 147–156.
[7] Amazon SageMaker, URL https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html 2020
[8] N.S. Arunraj, D. Ahrens, A hybrid seasonal autoregressive integrated moving average and quantile regression for daily food sales forecasting, Int. J. Prod. Econ. 170 (2015) 321–335.
[9] S.S. Baboo, I.K. Shereef, An efficient weather forecasting system using artificial neural network, Int. J. Environ. Sci.Dev. 1 (4) (2010) 321.
[10] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, Advances in Neural Information Processing Systems, vol. 24, 2011.
[11] G.E. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time Series Analysis: forecasting and Control, John Wiley & Sons, 2015.
[12] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[13] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and Regression Trees, CRC press, 1984.
[14] Y. Chen, Y. Kang, Y. Chen, Z. Wang, Probabilistic forecasting with temporal convolutional neural network, Neurocomputing (2020).
[15] H. Cho, Y. Kim, E. Lee, D. Choi, Y. Lee, W. Rhee, Basic enhancement strategies when using Bayesian optimization for hyperparameter tuning of deep neural networks, IEEE Access 8 (2020).
[16] C.-W. Chu, G.P. Zhang, A comparative study of linear and nonlinear models for aggregate retail sales forecasting, Int. J. Prod. Econ. 86 (3) (2003) 217–231.
[17] M. Cools, E. Moons, G. Wets, Investigating the variability in daily traffic counts through use of ARIMAX and SARIMAX models: assessing the effect of holidays on two site locations, Transp. Res. Rec. 2136 (1) (2009) 57–66.
[18] L. Cornelsen, C. Normand, Impact of the smoking ban on the volume of bar sales in Ireland: evidence from time series analysis, Health Econ. 21 (5) (2012) 551–561.
[19] A. Criminisi, J. Shotton, E. Konukoglu, Decision forests: a unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning, Found. Trends Comput. Graphics Vis. 7 (2–3) (2012) 81–227, doi:10.1561/0600000035.
[20] A.C. Davison, D.V. Hinkley, Bootstrap Methods and Their Application, No. 1, Cambridge University Press, 1997.
[21] C. Deb, F. Zhang, J. Yang, S.E. Lee, K.W. Shah, A review on time series forecasting techniques for building energy consumption, Renew. Sustain. Energy Rev. 74 (2017) 902–924.
[22] J.H. Friedman, Greedy function approximation: a gradient boosting machine, Ann. Stat. (2001) 1189–1232.
[23] J.H. Friedman, B.E. Popescu, et al., Predictive learning via rule ensembles, Ann. Appl. Stat. 2 (3) (2008) 916–954.
[24] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, F. Martínez-Álvarez, Multi-step forecasting for big data time series based on ensemble learning, Knowl. Based Syst. 163 (2019) 830–841.
[25] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Comput. Surv. (CSUR) 51 (5) (2018) 1–42.
[26] O. Gur Ali, E. Pinar, Multi-period-ahead forecasting with residual extrapolation and information sharing-utilizing a multitude of retail series, Int. J. Forecast. 32 (2) (2016) 502–517.
[27] R.J. Hyndman, Y. Khandakar, et al., Automatic Time Series for Forecasting: The Forecast package For R, No. 6/07, Monash University, Department of Econometrics and Business Statistics, 2007.
[28] I. Ilic, B. Gorgulu, M. Cevik, 2020a, (https://github.com/irogi/eblra).
[29] I. Ilic, B. Gorgulu, M. Cevik, Augmented out-of-sample comparison method for time series forecasting techniques, in: Canadian Conference on Artificial Intelligence, Springer, 2020, pp. 302–308.
[30] M. Khashei, M. Bijari, Which methodology is better for combining linear and nonlinear models for time series forecasting? J. Ind. Syst. Eng. (2012).
[31] B. Krollner, B.J. Vanstone, G.R. Finnie, Financial time series forecasting with machine learning techniques: a survey., ESANN, 2010.
[32] B. Liu, J. Nowotarski, T. Hong, R. Weron, Probabilistic load forecasting via quantile regression averaging on sister forecasts, IEEE Trans. Smart Grid 8 (2) (2015) 730–737.
[33] S. Ma, R. Fildes, Retail sales forecasting with meta-learning, Eur. J. Oper. Res. (2020).
[34] S. Makridakis, E. Spiliotis, V. Assimakopoulos, The M4 competition: 100,000 time series and 61 forecasting methods, Int. J. Forecast. 36 (1) (2020) 54–74.
[35] S. Makridakis, E. Spiliotis, V. Assimakopoulos, The M5 accuracy competition: results, findings and conclusions, Int. J. Forecast. (2020).
[36] K.-R. Müller, A.J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, V. Vapnik, Predicting time series with support vector machines, in: ICANN, Springer, 1997, pp. 999–1004.
[37] B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-BEATS: neural basis expansion analysis for interpretable time series forecasting, 2019 arXiv preprint arXiv:1905.10437
[38] A.R.S. Parmezan, V.M. Souza, G.E. Batista, Evaluation of statistical and machine learning models for time series prediction: identifying the state-of-the-art and the best conditions for the use of each model, Inf. Sci. 484 (2019) 302–337.
[39] S.S. Rangapuram, M.W. Seeger, J. Gasthaus, L. Stella, Y. Wang, T. Januschowski, Deep state space models for time series forecasting, in: Advances in Neural Information Processing Systems, 2018, pp. 7785–7794.
[40] Rossmann Store Sales, 2020, URL https://www.kaggle.com/c/rossmann-store-sales.
[41] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, DeepAR: probabilistic forecasting with autoregressive recurrent networks, Int. J. Forecast. 36 (3) (2020) 1181–1191.
[42] S. Smyl, A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting, Int. J. Forecast. 36 (1) (2020) 75–85.
[43] S.B. Taieb, R.J. Hyndman, A gradient boosting approach to the Kaggle load forecasting competition, Int. J. Forecast. 30 (2) (2014) 382–394.
[44] T. Taskaya-Temizel, M.C. Casey, A comparative study of autoregressive neural network hybrids, Neural Netw. 18 (5–6) (2005) 781–789.
[45] Y. Wang, A. Smola, D. Maddix, J. Gasthaus, D. Foster, T. Januschowski, Deep factors for forecasting, in: ICML, PMLR, 2019, pp. 6607–6617.
[46] R. Wen, K. Torkkola, B. Narayanaswamy, D. Madeka, A multi-horizon quantile recurrent forecaster, 2021arXiv preprint arXiv:1711.11053
[47] G.P. Zhang, Time series forecasting using a hybrid ARIMA and neural network model, Neurocomputing 50 (2003) 159–175.
[48] M. Zhou, X. Zeng, A. Chen, Deep forest hashing for image retrieval, Pattern Recognit. 95 (2019) 114–127.

Igor Ilic received his M.Sc. degree in Data Science and Analytics from Ryerson University, and BMath degree in Mathematical Physics from University of Waterloo. He worked as a research assistant at Data Science Lab at Ryerson University conducting research on focusing on probabilistic time series forecasting and explainable AI.

Berk Gorgulu is a PhD candidate in Department of Mechanical & Industrial Engineering at University of Toronto, Toronto, ON. He received his B.S. and M.S. degrees from Department of Industrial Engineering at Bogazici University, Istanbul, Turkey in 2017 and 2018 respectively. He is interested in stochastic processes, queueing theory, statistical learning and time-series data mining.

Mucahit Cevik is an assistant professor in the Faculty of Engineering, Ryerson University. He obtained his Ph.D. in Industrial and Systems Engineering from University of Wisconsin - Madison, and received his B.Sc. and M.Sc. in Industrial Engineering from Bogazici University, Turkey. His current research focus is on machine learning, reinforcement learning and their applications in healthcare. He is a member of INFORMS.

Mustafa Gokce Baydogan is an Assistant Professor in Department of Industrial Engineering at Bogazici University, Istanbul, Turkey. Before joining Bogazici University, he worked as a postdoctoral research assistant in the Security and Defense Systems Initiative at Arizona State University (ASU) between 2012–2013. He received his Ph.D. degree in Industrial Engineering from ASU in 2012. His B.S. and M.S. degrees are in Industrial Engineering both from Department of Industrial Engineering at Middle East Technical University, Ankara, Turkey in 2006 and 2008 respectively. His current research interests focus on statistical learning, with applications in temporal data mining (time series and images) and data mining for massive, multi- variate data sets. Details about him and his work can be reached through: http://www.mustafabaydogan.com