ARTICLE IN PRESS

International Journal of Forecasting xxx (xxxx) xxx

Contents lists available at ScienceDirect

International Journal of Forecasting

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

Blending gradient boosted trees and neural networks for point and probabilistic forecasting of hierarchical time series

Ioannis Nasios [a], Konstantinos Vogklis [a,b,*]

[a] Nodalpoint Systems, Athens, Greece
[b] Department of Tourism, Ionian University, Corfu, Greece

ARTICLE INFO

Keywords:
M5 Competition
Point forecast
Probabilistic forecast
Regression models
Gradient boosted trees
Neural networks
Machine learning

ABSTRACT

In this study, we addressed the problem of point and probabilistic forecasting by describing a blending methodology for machine learning models from the gradient boosted trees and neural networks families. These principles were successfully applied in the recent M5 Competition in both the Accuracy and Uncertainty tracks. The key points of our methodology are: (a) transforming the task into regression on sales for a single day; (b) information-rich feature engineering; (c) creating a diverse set of state-of-the-art machine learning models; and (d) carefully constructing validation sets for model tuning. We show that the diversity of the machine learning models and careful selection of validation examples are most important for the effectiveness of our approach. Forecasting data have an inherent hierarchical structure (12 levels) but none of our proposed solutions exploited the hierarchical scheme. Using the proposed methodology, we ranked within the gold medal range in the Accuracy track and within the prizes in the Uncertainty track. Inference code with pre-trained models are available on GitHub.[1]

1. Introduction

Machine learning (ML) methods are well established in the academic literature as alternatives to statistical methods for time series forecasting (Crone, Hibon, & Nikolopoulos, 2011; Kim, Hong, & Kang, 2015; Makridakis, Spiliotis, & Assimakopoulos, 2018, 2020, 2022). The results obtained in the recent M-competitions also demonstrated a trend from the classical exponential smoothing and autoregressive integrated moving average (ARIMA) methods (Hyndman & Athanasopoulos, 2018) toward more data-driven generic ML models, such as neural networks (Bishop et al., 1995) and gradient boosted trees (Friedman, 2001, 2002).

One of the most important results in the recent M5 Competition (Makridakis et al., 2022) was the superior performance of ML methods, particularly gradient boosted trees and neural networks, compared with statistical time series methods (e.g., exponential smoothing and ARIMA). This result supports the finding established in the M4 competition (Makridakis et al., 2020) that ML has huge potential for forecasting, especially when combined with statistical methods.

The M5 forecasting competition was designed to empirically evaluate the accuracy of new and existing forecasting algorithms in a real-world hierarchical unit sales series scenario. The data set provided contained 42,840 hierarchical sales data from Walmart and it covered stores in three US states (California (CA), Texas (TX), and Wisconsin (WI)), including sales data per item level, department and product categories, and store details ranging from February 2011 to April 2016. The products had a (maximum) selling history of 1941 days. The competition was divided into two tracks, where one required point forecasts (Accuracy track) and the other required estimation of the uncertainty distribution (Uncertainty track). The time horizon for both tracks was 28 days ahead. In the Accuracy track, the task involved predicting the sales

* Corresponding author.
E-mail addresses: voglis@nodalpoint.com, voglis@ionio.gr (K. Vogklis).

[1] https://github.com/IoannisNasios/M5_Uncertainty_3rd_place.

https://doi.org/10.1016/j.ijforecast.2022.01.001
0169-2070/

Please cite this article as: I. Nasios and K. Vogklis, Blending gradient boosted trees and neural networks for point and probabilistic forecasting of hierarchical time series. International Journal of Forecasting (2022), https://doi.org/10.1016/j.ijforecast.2022.01.001.

Table 1
M5 series aggregation levels.

Level	Level description	Aggr. level	#of series
1	Unit sales of all products aggregated for all stores/states	Total	1
2	Unit sales of all products, aggregated for each state	State	3
3	Unit sales of all products aggregated for each store	Store	10
4	Unit sales of all products aggregated for each category	Category	3
5	Unit sales of all products aggregated for each department	Department	7
6	Unit sales of all products aggregated for each state and category	State/Category	9
7	Unit sales of all products aggregated for each state and department	State/Department	21
8	Unit sales of all products aggregated for each store and category	Store/Category	30
9	Unit sales of all products aggregated for each store and department	Store/Department	70
10	Unit sales of product x aggregated for all stores/states	Product	3,049
11	Unit sales of product x aggregated for each state	Product/State	9,147
12	Unit sales of product x aggregated for each store	Product/Store	30,490
Total			42,840

for each of the 42,840 hierarchical time series after day 1941. In the Uncertainty track, the task involved providing probabilistic forecasts for the corresponding median and four prediction intervals (50%, 67%, 95%, and 99%).

Table 1 presents all of the hierarchical groupings of the data, where level 12 containing 30,490 unique combinations of product per store is the most disaggregated level. According to the competition rules, only these 30,490 series needed to be submitted and the forecasts of all higher levels were automatically calculated by aggregating (summing) those for this lowest level. Thus, regardless of how the hierarchical information was used to produce 28-day ahead predictions, only those for the level 12 series were actually submitted.

2. ML-based forecasting

ML methods are designed to learn patterns from data and they make no assumptions about their characteristics. Time series forecasting can be readily formulated as a supervised learning task. The goal is to approximate a function $f(\cdot;\theta) : \mathbf{R}^d \rightarrow \mathbf{R}$ controlled by a set of parameters θ, which corresponds to the relation between a vector of input variables (features) and a target quantity. The ML setup is complete after we define: (a) a (training) data set $\mathcal{D} = (x^{(k)}, y^{(k)}), \quad k = 1...n$ comprising a set of tuples containing features $x^{(k)} \in \mathbf{R}^d$ that describe a target $y^{(k)} \in \mathbf{R}$ (number of daily sales in the context of the M5 Competition), and (b) a suitable loss function that needs to be minimized $\mathcal{L}(Y, f(x;\theta))$. The parameters of $f(\cdot;\theta)$ are then tuned to minimize the loss function $\mathcal{L}$ using the tuples in the training data set $\mathcal{D}$.

2.1. Feature engineering

One of the key components of any ML method involves selecting representative and information-rich input features $x^{(\cdot)} \in \mathbf{R}^d$. The feature engineering task is a largely ad hoc procedure, which is considered a science and art in equal parts, and it mainly depends on the experience of the practitioner in similar tasks. In the context of the M5 Competition, we considered the following feature groups.

1. Categorical id-based features: These special types of features can only take discrete values. Their numerical encodings can take many forms that depend on the methodology employed.

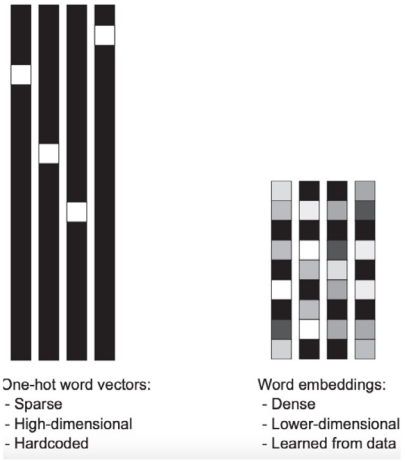

Fig. 1. Trainable embeddings vs one-hot encoding.
Source: Image taken from (Chollet et al., 2018).

(a) Categorical variables encoded via the mean target value: The target in a predictor variable is perfectly suited for categorical variables in this encoding process. Each category is replaced with the corresponding average of the target value in the presence of this category (Potdar, Pardawala, & Pai, 2017).
(b) Trainable embedding encoding: This common paradigm has increased in popularity since its application to natural language processing (Akbik et al., 2019; Al-Rfou, Perozzi, & Skiena, 2013), where the aim is to project the distinct states of a categorical feature to a real-valued, low-dimensional latent space. This space is usually implemented as a neural network layer (called embedding layer) and employed to compactly encode all the discrete states of a categorical feature. This method contrasts with one-hot encodings, which are binary, sparse, and very high-dimensional, trainable embeddings comprising low-dimensional floating-point vectors (see Fig. 1).

2. Price related: Sell prices are provided on a weekly level for each combination of store and product. These prices are constant on a weekly basis but they may change over time. Using this information, we can calculate statistical features such as the maximum, minimum, mean, standard deviation, and the number of unique historical prices for each combination of store and product (level 12). Intuitively, using the number of unique prices as a feature can reflect the volatility in a specific store/product combination. Frequent price changes indicate marketing effort and they should be flagged appropriately.
3. Calendar related
    - Special events and holidays (e.g. Super Bowl, Valentine's Day, and Orthodox Easter) are organized into four classes, i.e., sporting, cultural, national, and religious.
    - Supplement nutrition assistance program (SNAP) activities that serve as promotions. A binary variable (0 or 1) indicate whether the stores in CA, TX, or WI allow SNAP purchases on the date examined.
4. Lag related features:
    - Lag only: These features are based on the historical sales for each store/product combination (level 12) during $28 + k$ days before a given date t where k ranges from 1 to 14, thereby spanning two weeks. The selection of this k look-back parameter is based on the performance of the model with the validation splits (see Section 2.2)
    - Rolling only: Rolling mean and standard deviation of historical sales for each store/product (level 12) ending 28 days before a given date t.
    - Lag and rolling: Rolling mean and standard deviation until a lag date in the past.

In total, we devised around 80 input features and each was used to predict the unit sales of a specific product/store (level 12) for one specific date.

2.2. Cross validation

The inherent ordering in time series forecasting (i.e., the time component) forces ML practitioners to define special cross-validation schemes and avoid k-fold validation (Bengio & Grandvalet, 2004) random splits back and forth in time. We selected the following different training/validation splits.

- Validation split 1:
    - Training days $[d_1, d_2, \ldots, d_{1940}]$
    - Validation days $[d_{1914}, d_{1915}, \ldots, d_{1941}]$ (last 28 days)
- Validation split 2:
    - Training days $[d_1, d_2, \ldots, d_{1885}]$
    - Validation days $[d_{1886}, d_{1887}, \ldots, d_{1913}]$ (28 days before last 28 days)
- Validation split 3:
    - Training days $[d_1, d_2, \ldots, d_{1577}]$
    - Validation days $[d_{1578}, d_{1579}, \ldots, d_{1605}]$ (exactly one year before)

Modeling and blending was conducted with the aim of improving the mean of the competition metric using these splits. For each split, we performed the following two-step modeling procedure.

> **Tuning**: Using all three validation sets and the competition metric to select the best architecture and parameters for the models.
> **Full train**: Using the fine-tuned parameters from the previous step to perform a full training run until day d_{1941}.

The high added value obtained from using multiple validation sets eliminated any need for external adjustments of the final prediction (see Finding 5 in Makridakis et al. (2022)).

3. Point forecasting method

Point forecasting is based on using ML models to predict the number of daily sales for a specific date and a specific product/store combination. In order to forecast the complete horizon of 28 days, we applied a recursive multi-step scheme (Ben Taieb, Hyndman, et al., 2012), which involved using the prediction by a model at a specific time step (day) as an input in order to predict that for the subsequent time step. This process was repeated until the desired number of time step forecasts were obtained. We found that this method was superior to a direct multi-step scheme where the models were predicted only once 28 days in the future. We implemented this scheme but it was rejected due to its inferior performance.

The performance metric selected for this forecasting task was a variant of the mean absolute scaled error (Hyndman & Koehler, 2006) called the root mean squared scaled error (RMSSE). This metric for a 28-day horizon is defined in Eq. (1):

$$RMSSE = \sqrt{\frac{\frac{1}{h}\sum_{t=n+1}^{n+h}\left(y_t - \hat{y}_t\right)^2}{\frac{1}{n-1}\sum_{t=2}^{n}\left(y_t - y_{t-1}\right)^2}}, \tag{1}$$

where y is the actual future value of the time series examined (for a specific aggregation level l) at point t, $\hat{y}$ is the predicted value, n is the number of historical observations, and h is the 28-day forecasting horizon. The selection of this metric is justified by the intermittency of the forecasting data comprising sporadic unit sales with many zeros.

The overall accuracy of each forecasting method at each aggregation level was computed by averaging the RMSSE scores across all series in the data set using appropriate price-related weights. This metric called the

Table 2
Parameters for LightGBM models.

parameter	lgb_cos	lgb_nas
boosting_type	gbdt	gbdt
objective	tweedie	tweedie
tweedie_variance_power	1.1	1.1
subsample	0.5	0.6
subsample_freq	1	1
learning_rate	0.03	0.02
num_leaves	2047	2047
min_data_in_leaf	4095	4095
feature_fraction	0.5	0.6
max_bin	100	100
n_estimators	1300	see Table 3
boost_from_average	false	false

weighted RMSSE (WRMSSE) by the organizers is defined in Eq. (2).

$$WRMSSE = \sum_{i=0}^{42,840} w_i \cdot RMSSE_i, \tag{2}$$

where w_i is a weight assigned based on the i_{th} series. This weight was computed based on the last 28 observations of the training sample in the data set, i.e., the cumulative actual dollar sales for each series in that particular period (sum of units sold multiplied by their respective price) (Makridakis et al., 2022). The weights w_i were computed once and they remained constant throughout the analysis.

3.1. Models

In the following, we briefly describe the specific types of gradient boosting machine (GBM) and neural network models applied.

3.1.1. LightGBM models

LightGBM (Ke et al., 2017) is an open source gradient boosting decision tree method (Friedman, 2002; Ye, Chow, Chen, & Zheng, 2009) implemented by Microsoft, which uses a histogram-based algorithm to speed up the training process and reduce the memory required. It has been shown that LightGBM models are highly efficient in terms of their speed and quality in many practical regression problems. For the Accuracy track, we trained two different LightGBM models:

1. `lgb_cos`: Single LightGBM regression model for all available data (see Table 2);
2. `lgb_nas`: A different LightGBM regression model for each store (10 models in total) (see Table 2).

The target output from each LightGBM model was the sales count for a specific product. The target quantity (sales count) was intermittent with many zeros ($\approx$ 68% of the daily sales were zeros), so we implemented a special loss function that followed the Tweedie Gaussian distribution (Tweedie et al., 1957). Tweedie regression (Zhou, Qian, & Yang, 2020) is designed to deal with right-skewed data where most of the target values are concentrated around zero. Fig. 2 shows a histogram of the sales for all available training data, where it is obviously right skewed and highly concentrated around zero.

Table 3
Number of tree estimators per store.

store_id	n_estimators
CA_1	700
CA_2	1100
CA_3	1600
CA_4	1500
TX_1	1000
TX_2	1000
TX_3	1000
WI_1	1600
WI_2	1500
WI_3	1100

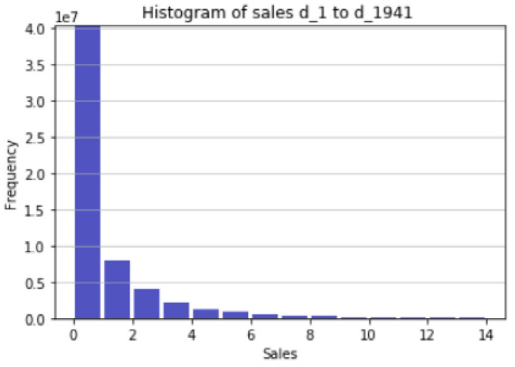

Fig. 2. Histogram of daily sales for product/store (level 12) aggregation level.

The formula of the Tweedie loss function given a predefined parameter p is presented in Eq. (3). In our implementation, we used the default value of $p = 1.5$, which achieves a good balance between the two terms:

$$\mathcal{L}(y, \hat{y}) = -\sum_k y^{(k)} \cdot \frac{\left(\hat{y}^{(k)}\right)^{1-p}}{1-p} + \frac{\left(\hat{y}^{(k)}\right)^{2-p}}{2-p}, \quad \hat{y}^{(k)} = f(x^{(k)}; \theta), \tag{3}$$

where $f(\cdot; \theta) : \mathbf{R}^d \rightarrow \mathbf{R}$ is a regression model controlled by a set of parameters θ, which maps a set of multidimensional features $x^{(k)} \in \mathbf{R}^d$ to a target value $y^{(k)}$ (daily sales count), and $\hat{y}^{(k)}$ is the output of the regression model for input $x^{(k)}$.

3.1.2. Neural network models

We implemented the following two classes of neural network models.

- `keras_nas` - Keras MLP models.
  Keras (Chollet et al., 2018) is a model-level library that provides high-level building blocks for developing neural network models. Keras has been adopted by Google as the standard interface for its flagship ML framework Tensorflow (Abadi et al., 2016). Complex neural network architectures can be defined using the highly intuitive descriptions of the Keras building modules, and then used for experiments based on training and inference.

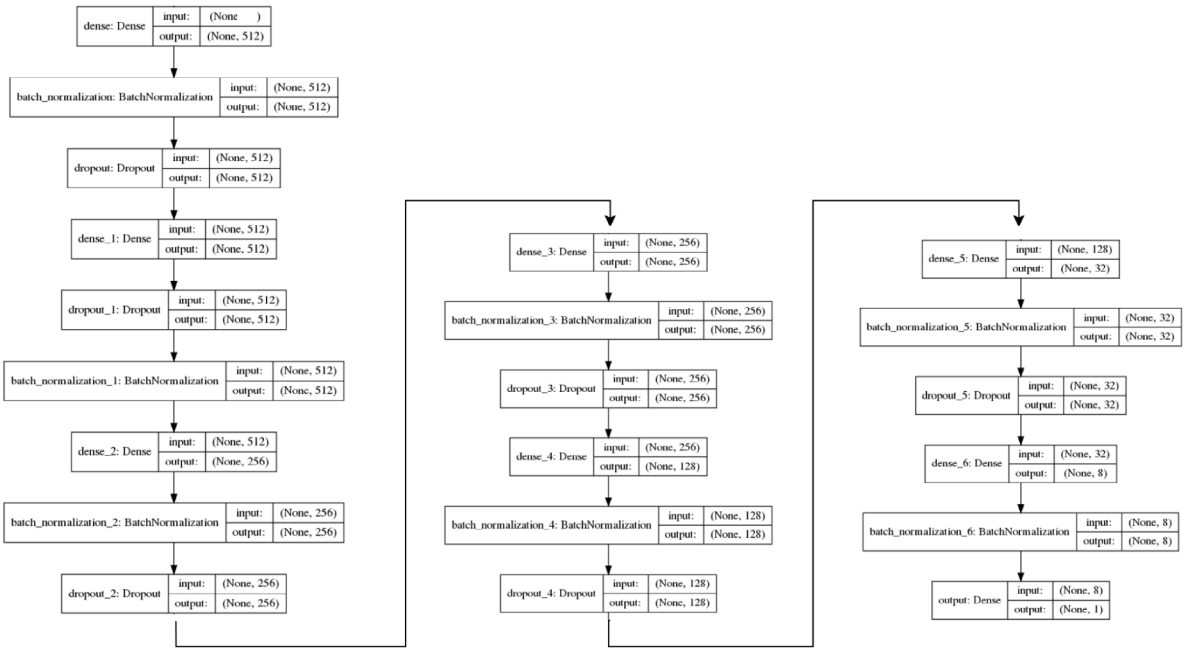

Fig. 3. Shared model architecture for the `keras_nas` case. The part connected immediately after the input layers is shown.

We implemented three slightly different architectures (`keras1`, `keras2`, and `keras3`), which are presented in detail in Appendix B. Each architecture shared a common backbone with a sequence of seven dense batch normalization and dropout layers, as shown in Fig. 3, but they differed slightly in the input layer. The input features used for these models are shown in Table A.6 in Appendix A. All of the models used embeddings to encode categorical input information (i.e., event names 1 and 2, event types 1 and 2, department id, store id, category id, and state id). Among the available Keras loss functions, we selected the mean squared error, which obtained the best results with all of the validation splits described in Section 2.2. Finally, we trained these models using the Adam optimizer with a constant learning rate of 0.0001 for 35 epochs and retained the weights for the last 5 epochs, thereby yielding 15 models and the arithmetic mean of their predictions was obtained as the output.

- `fastai_cos` - FastAI MLP models. FastAI (Howard & Gugger, 2020) is a Pytorch (Paszke et al., 2019) based deep learning library that provides high-level components for many ML tasks. To address the regression task, we incorporated the *tabular* module, which implements state-of-the-art deep learning models for tabular data. Importantly, the FastAI tabular module also uses embedding layers for categorical data. Similar to Keras, using the embedding layer allowed good interactions among the categorical variables and exploited the inherent automatic feature extraction mechanism of deep learning.
  Our modeling approach involved implementing a special Tweedie loss function for use during training. The FastAI model is quite different from the Keras/Tensorflow `keras_nas` model because it implements a different architecture and uses a different loss function and features. At the time when the competition was conducted, a reliable implementation of the Tweedie loss function was readily available in the Pytorch library.

3.2. Ensembling

The final part of our modeling approach involved carefully blending the predictions obtained by the diverse set of models. We divided our models into the following four groups:

- `lgb_cos`: One LightGBM model using all available data;
- `lgb_nas`: 10 LightGBM models (one model per store) using all available data for every store;
- `keras_nas`: 15 Keras models (three different architectures $\times$ five different weights for each) using only the last 17 $\times$ 28 days of data and simple averaging;
- `fastai_cos`: One FastAI model using all available data

All groups were fine tuned so they performed best with the average of the three validation splits described in Section 2.2. After fine tuning, all groups were retrained using the best parameters and the information available until d_{1941}, before they were applied to produce forecasts for days d_{1942} until d_{1969}.

The individual predictions were blended using geometric averaging according to Eq. (4).

$$\text{blend} = \begin{cases} \left(\text{lgb\_nas}^{3.5} \cdot \text{lgb\_cos}^{1.0} \cdot \text{keras\_nas}^{1.0} \cdot \text{fastai\_cos}^{0.5}\right)^{\frac{1}{6}} & \text{for days 1–27} \\ \left(\text{lgb\_nas}^{3.0} \cdot \text{lgb\_cos}^{0.5} \cdot \text{fastai\_cos}^{1.5}\right)^{\frac{1}{5}} & \text{for day 28} \end{cases} \tag{4}$$

Table 4
Competition scores for each component.

	lgb_nas		lgb_cos		keras_nas		fastai_cos		Ensemble	
Val. split 1	0.474		0.470		0.715		0.687		0.531	
Val. split 2	0.641		0.671		0.577		0.631		0.519	
Val. split 3	0.652		0.661		0.746		0.681		0.598	
Mean/std	0.589	0.08	0.602	0.09	0.679	0.05	0.667	0.02	0.549	0.03

$$SPL = \frac{1}{h}\frac{\sum_{t=n+1}^{n+h}(Y_t - Q_t(u))\,u\mathbf{1}\{Q_t(u) \le Y_t\} + (Q_t(u) - Y_t)\,(1-u)\mathbf{1}\{Q_t(u) > Y_t\}}{\frac{1}{n-1}\sum_{t=2}^{2}|Y_t - Y_{t-1}|}, \tag{5}$$

Box I.

We tested many aggregation strategies to obtain the final blend, such as the arithmetic mean and median, but we selected the geometric mean because it obtained the best combined performance with all of the validation splits.

After reviewing several of the predictions made by the keras_nas group of models, we observed an unexpected large peak on the last day in the private set (day d_{1969}). After confirming that this behavior was dominant in most cases, we decided to exclude that group's prediction for the last day, which explains why the keras_nas predictions were not used in the lower branch of Eq. (4). Table 4 presents the individual WRMSSE scores for each model category and each validation split as well as the final blending score.

3.3. Post-processing

We tested several post-processing smoothing techniques to improve the forecasting accuracy with all of the validation splits. Finally, we employed a simple exponential smoothing ($\alpha = 0.96$) per product and store id (level 12–30,490), which obtained substantial improvements with both validation splits and the final evaluation (private leader board). We considered that smoothing enhanced the dependence of one specific product sales count on the near past and future sales, which was already captured by the models.

We found that this post-processing method performed will in both competition tracks, but we only used it in the Uncertainty track because it was identified two hours before the competition closed. If this post-processing method had also been applied to the Accuracy track submissions, we would have placed three positions higher in the Accuracy track leader board.

4. Probabilistic forecasting method

The performance metric selected for this competition track was the scaled pinball loss (SPL) function, which was calculated for the 28-day horizon for each series and quantile using Eq. (5) in Box I:

where Y_t is the actual future value of the time series examined at point t, $Q_t(u)$ is the forecast generated for quantile u, h is the forecasting horizon, n is the number of historical observations, and $\mathbf{1}$ is the indicator function (1 if Y was within the postulated interval but 0 otherwise). The values u were set to $u_1 = 0.005$, $u_2 = 0.025$, $u_3 = 0.165$, $u_4 = 0.25$, $u_5 = 0.5$, $u_6 = 0.75$, $u_7 = 0.835$, $u_8 = 0.975$, and $u_9 = 0.995$, which correspond to the required median and 50%, 67%, 95%, and 99% prediction intervals.

After estimating the SPL for all time series and all of the quantiles required for the competition, the Uncertainty track competition entries were ranked using the weighted SPL (WSPL) according to Eq. (6):

$$WSPL = \sum_{i=1}^{42{,}840} w_i * \frac{1}{9}\sum_{j=1}^{9} SPL(u_j) \tag{6}$$

where the weights w_i are the same as those described in Section 3.

4.1. Quantile estimation via optimization

Using our best point forecast as the median, we optimized the WSPL objective function with validation split 1 (last 28 days) and calculated the factors in Table 5. Due to time restrictions in the competition, we could not extend our analysis to cover all three validation splits. These factors were used to multiply the median solution (quantile $u = 0.50$) and produce the remaining upper and lower quantiles. We assumed symmetric distributions for levels 1–9 and skewed distributions for levels 10–12. Furthermore, due to the right-skewness of our sales data (zero-bounded on the left) at every level, the proposed factor for the last quantile (99.5%) distributions was multiplied by either 1.02 or 1.03. These factors were determined in order to minimize WSPL with validation split 1.

4.2. Quantile correction using statistical information

Level 12 (the only non-aggregated level) was the most difficult for obtaining accurate estimations and the previously calculated factors could be further improved. Statistical information for the previous days was very important for this level. For every product/store id, eight statistical sales quantiles (four intervals excluding median) were calculated over the last 13 × 28 days (1 year)

Table 5
Quantile factors for all levels.

Level	Aggr	#	0.005	0.025	0.165	0.25	0.5	0.75	0.835	0.975	0.995
1	Total	1	0.890	0.922	0.963	0.973	1.000	1.027	1.037	1.078	1.143
2	State	3	0.869	0.907	0.956	0.969	1.000	1.031	1.043	1.093	1.166
3	Store	10	0.848	0.893	0.950	0.964	1.000	1.036	1.049	1.107	1.186
4	Category	3	0.869	0.907	0.951	0.969	1.000	1.031	1.043	1.093	1.166
5	Dept.	7	0.827	0.878	0.943	0.960	1.000	1.040	1.057	1.123	1.209
6	State/Cat.	9	0.827	0.878	0.943	0.960	1.000	1.040	1.057	1.123	1.209
7	State/Dept.	21	0.787	0.850	0.930	0.951	1.000	1.048	1.070	1.150	1.251
8	Store/Cat.	30	0.767	0.835	0.924	0.947	1.000	1.053	1.076	1.166	1.272
9	Store/Dept.	70	0.707	0.793	0.905	0.934	1.000	1.066	1.095	1.208	1.335
10	Product	3.049	0.249	0.416	0.707	0.795	1.000	1.218	1.323	1.720	2.041
11	Product/State	9.147	0.111	0.254	0.590	0.708	1.000	1.336	1.504	2.158	2.662
12	Product/Store	30.490	0.005	0.055	0.295	0.446	1.000	1.884	2.328	3.548	4.066

$$\begin{aligned}\text{level } 12 = {} & 0.2 \cdot \text{quantile estimation} \\ & + 0.7 \cdot \left(\frac{\text{statistical correction}_{13\times 28 \text{ days}} + 1.75 \cdot \text{statistical correction}_{28 \text{ days}}}{2.75} \right) \\ & + 0.1 \cdot \left(\frac{\text{weekly statistical correction}_{13\times 28 \text{ days}} + \text{weekly statistical correction}_{3\times 28 \text{ days}}}{2} \right)\end{aligned}$$

Box II.

$$\begin{aligned}\text{level } 11 = {} & 0.91 \cdot \text{quantile estimation} \\ & + 0.09 \cdot \left(\frac{\text{statistical correction}_{13\times 28 \text{ days}} + 1.75 \cdot \text{statistical correction}_{28 \text{ days}}}{2.75} \right)\end{aligned}$$

Box III.

and over the last 28 days. These quantiles were first averaged and then used to correct the quantile estimation described in Section 4.1 for the corresponding level. For the same reason, we calculated *w*eekly sales quantiles over the last 13 × 28 days and 3 × 28 days. The final formula used for estimating level 12 quantiles, which yielded the minimum WSPL score for validation split 1, is as given in Box II.

Level 11 was also corrected in a similar manner and the final formula is as given in Box III.

No corrections were applied for levels other than levels 12 and 11, so the respective quantile factors for levels 1–10 were as shown in Table 5.

The changes in our submission attempts (SPL score/ranking) for the Uncertainty track of the competition are shown in Fig. 4. This scatter plot highlights:

(a) The importance of ensembling diverse models because our rank increased by almost 90 places only by ensembling;
(b) The contribution of statistical correction for levels 11 and 12, which led to a winning place.

The overall ensembling, optimizing, and correcting procedure for probabilistic forecasting is depicted in Fig. 5.

5. Discussion

We developed a ML solution for point and probabilistic forecasting of hierarchical time series representing daily unit sales of retail products. This methodology involves two state-of-the-art ML approaches comprising gradient boosting trees and neural networks, which we tuned and combined using carefully selected training and validation sets. The proposed methodology was applied successfully in the recent M5 Competition to reach a prize position place in the Uncertainty track.

5.1. Point forecasting score breakdown

In order to obtain deeper insights into the point forecasting task, we analyzed the WRMSSE calculation using Eq. (2) at each hierarchical level. The resulting per-level WRMSSE results for validation split 1 are shown in Fig. 6. Clearly, the performance varied significantly among the different aggregation levels, where Levels 10, 11, and 12 were the hardest to predict. Interestingly, although Level 1 was simply the aggregation of all the Level 12 predictions, the Level 1 loss was less than half the magnitude of the Level 12 loss. Thus, it appeared that aggregation overcame the poor predictive capacity at Level 12. This was

Fig. 4. Leader board ranking vs. score.

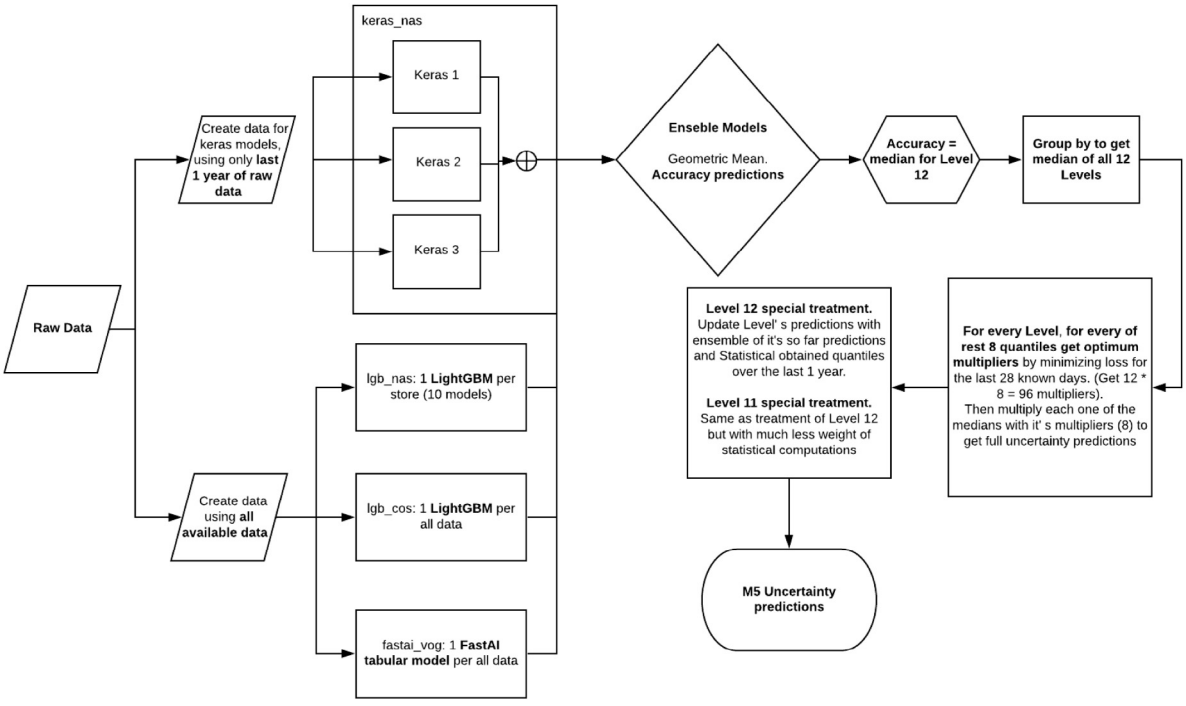

Fig. 5. Flowchart illustrating the probabilistic forecasting pipeline.

also demonstrated more clearly by the fact that although a mixture of traditional forecasting techniques (ARIMA and exponential smoothing) achieved lower errors at levels 10, 11, and 12 than that shown in Fig. 6, the overall score was actually poor.

The mean WRMSSE score obtained in the validation using the final blend with Eq. (4) was 0.549 and the score for our final submission was 0.552, and thus they only differed by 0.003 units. This close tracking of the unseen test data error by the validation error is generally the aim of ML practitioners in real-world problems, and it provides an extra indicator of the reliability of our methodology.

5.2. Visualization of probabilistic forecasting factors

Fig. 7 presents a graphical plot of the factors sorted by increasing level. Asymmetrical widening of the calculated factors occurred as the number of series at the

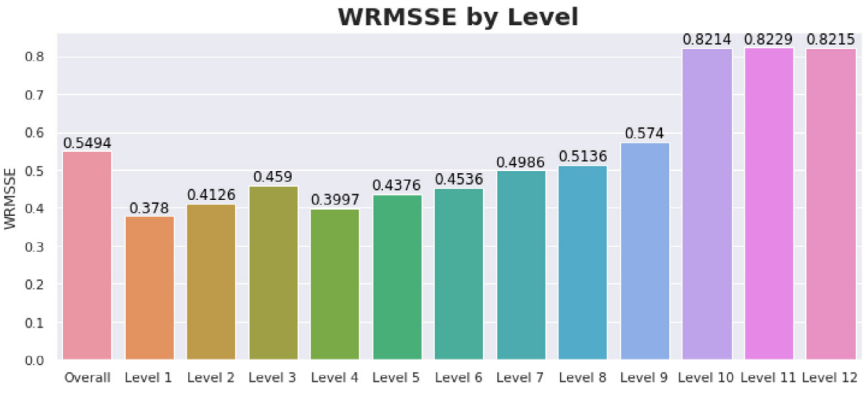

Fig. 6. Loss per level for predictions with validation split 1.

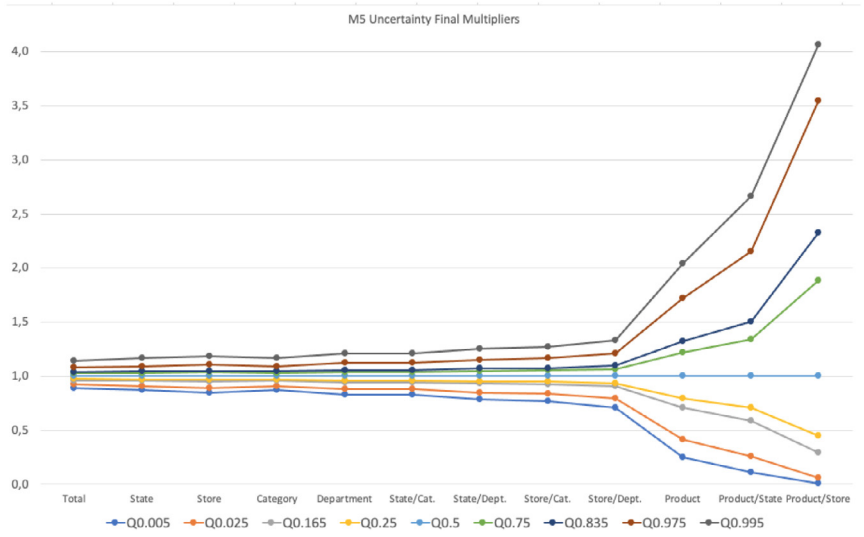

Fig. 7. Quantile factors for increasing levels of aggregation.

Table A.6
M5 competition features.

Feature category	Feature name	Type	Meaning	LightGBM	Keras	FastAI
non-trainable features	id	cat	30490 unique ids that correspond to combinations of item_id/store_id			
	d	integer	increasing number of days 1–1941			
id-based categorical features	item_id	cat	3049 distinct item ids	*		*
	dept_id	cat	7 distinct departments	*	*	*
	cat_id	cat	3 product categories	*	*	*
	store_id	cat	10 different stores	*	*	*
	state_id	cat	3 different states		*	

(*continued on next page*)

corresponding level increased. This delta-like shape was correlated with the increasing WRMSSE error at levels 10–12, as shown in Fig. 6.

5.3. Takeaways

It should be noted that our point and probabilistic forecasting results were highly correlated because we

Table A.6 (*continued*).

Feature category	Feature name	Type	Meaning	LightGBM	Keras	FastAI
price related features	release	integer	week when the first price was set for the product	*		*
	sell_price	float	current price of the product	*	*	*
	sell_price_rel_diff	float	relative difference between two consecutive price changes		*	
	price_max	float	maximum price this product ever reached	*		*
	price_min	float	minimum price this product ever reached	*		*
	price_std	float	standard deviation of the historical prices of the product	*		*
	price_mean	float	average value of the historical prices of the product	*		*
	price_norm	float	normalized price (price/price_max) for the day [0 1]	*		*
	price_nunique	integer	number of unique prices for this product	*		*
	item_nunique	integer	number of unique products that reached the same price in a specific store.	*		*
	price_momentum	float	ratio of the previous price relative to the current price of the product	*		*
	price_momentum_m	float	mean price of the product in the last month	*		*
	price_momentum_y	float	mean price of the product in the last year	*		*
calendar related features	event_name_1	cat	If the date includes an event, the name of this event	*		*
	event_type_1	cat	If the date includes an event, the type of this event	*	*	*
	event_name_2	cat	If the date includes a second event, the name of this event	*	*	*
	event_type_2	cat	If the date includes a second event, the type of this event.	*	*	*
	snap_CA	binary	SNAP activities that serve as promotions in California on a specific date.	*	*	*
	snap_TX	binary	SNAP activities that serve as promotions in Texas on a specific date.	*		*
	snap_WI	binary	SNAP activities that serve as promotions in Wisconsin on a specific date.	*		*
	log_d	float	$log(1+d)$ where d is the increasing number of days		*	
	tm_d	integer	day of the month for a specific date	*		*
	tm_w	integer	week of the year for a specific date	*		*
	tm_m	integer	month of the year for a specific date	*		*
	tm_y	integer	year of the specific date (5 years in total)	*		*
	tm_wm	integer	number of the week within a month	*		*
	tm_dw	integer	number of the day within a week	*	*	*
	tm_w_end	binary	is weekend or not	*		
target encoding features	enc_cat_id_mean	float	average value of sales for each category id	*		
	enc_cat_id_std	float	std value of sales for each category id	*		
	enc_dept_id_mean	float	average value of sales for each department id	*		
	enc_dept_id_std	float	std value of sales for each department id	*		
	enc_item_id_mean	float	average value of sales for each item id	*		
	enc_item_id_std	float	std value of sales for each department id	*		
	enc_item_id_state_id_mean	float	average value of sales for each combination of item id and state id	*		
	enc_item_id_state_id_std	float	std value of sales for each combination of item id and state id	*		
	enc_state_id_dept_id_mean	float	average value of sales for each combination of item id and department id	*		*
	enc_state_id_dept_id_std	float	std value of sales for each combination of item id and department id	*		*
	enc_state_id_cat_id_mean	float	average value of sales for each combination of item id and category id	*		
	enc_state_id_cat_id_std	float	std value of sales for each combination of item id and category id	*		

(*continued on next page*)

Table A.6 (*continued*).

Feature category	Feature name	Type	Meaning	LightGBM	Keras	FastAI
lag features	sales_lag_28	integer	the sales value for each id and each date 28, 29, 30, …42 days ago (span of 2 weeks)	*	*	*
	sales_lag_29	integer				
	sales_lag_30	integer		*		*
	sales_lag_31	integer		*		*
	sales_lag_32	integer		*		*
	sales_lag_33	integer		*		*
	sales_lag_34	integer		*		*
	sales_lag_35	integer		*		*
	sales_lag_36	integer		*		*
	sales_lag_37	integer		*		*
	sales_lag_38	integer		*		*
	sales_lag_39	integer		*		*
	sales_lag_40	integer		*		*
	sales_lag_41	integer		*		*
	sales_lag_42	integer		*		*
rolling statistics 28 days ago	rolling_mean_7	float	rolling means and averages for the sales of a specific product that ended 28 days before current date	*	*	*
	rolling_median_7	float			*	
	rolling_std_7	float		*		*
	rolling_mean_14	float		*		*
	rolling_std_14	float		*		*
	rolling_mean_30	float		*	*	*
	rolling_median_30	float			*	
	rolling_std_30	float		*		*
	rolling_mean_60	float		*		*
	rolling_std_60	float		*		*
	rolling_mean_180	float		*		*
	rolling_std_180	float		*		*
rolling means within the 28 day window	rolling_mean_tmp_1_7	float	rolling means and averages for the sales of a specific product within 28 days before current date. We could train our models with these features but we needed to be extra careful when predicting 28 days ahead because these features had to be calculated recursively.	*		
	rolling_mean_tmp_1_14	float		*		*
	rolling_mean_tmp_1_30	float		*		*
	rolling_mean_tmp_1_60	float		*		*
	rolling_mean_tmp_7_7	float		*		*
	rolling_mean_tmp_7_14	float		*		*
	rolling_mean_tmp_7_30	float		*		*
	rolling_mean_tmp_7_60	float		*		*
	rolling_mean_tmp_14_7	float		*		*
	rolling_mean_tmp_14_14	float		*		*
	rolling_mean_tmp_14_30	float		*		*
	rolling_mean_tmp_14_60]	float		*		*

used the point forecasts as the starting points for our probabilistic analysis. Clearly, starting with better point forecasts will also yield better probabilistic forecasts.

The model diversity and selection of the training/validation split were crucial for the overall performance, and appropriate choices eliminated any need for external adjustments of the final predictions in both tracks, which was also highlighted in the M5 Competition Summary (Makridakis et al., 2022). After the competition ended, we found that using the magic multiplier 0.97 would have placed us first in the Accuracy track.

Finally, although the M5 Competition data set was hierarchical, we did not use this information explicitly in a reconciliation procedure, as described by Hyndman and Athanasopoulos (2018), Hyndman, Athanasopoulos, et al. (2014). Hierarchy was only considered implicitly based on the characteristics of the WRMSSE objective function.

5.4. Extensions

The competition data were carefully curated and cleaned to provide a rich variety of potential features. Many ML regression methods could also be tested. However, we consider that the most crucial decision was not the selection of the features or ML models, but the selection of representative validation sets. The three validation splits described in Section 2.2 were sufficient to stabilize our point forecasting results but we consider that using more validation splits might have been beneficial, and it could even have eliminated the need for *magic* external multipliers (0.97) to reach the top place.

An obvious extension to the probabilistic forecasting methodology was using a weighted averaged of three validation splits instead of validation split 1 (last 28 days).

Acknowledgment

Our involvement in the M5 competition was both tasked and sponsored by our company Nodalpoint Systems, Athens, Greece.

Appendix A. Complete feature list

See Table A.6.

Appendix B. Keras models

```
def keras1(num_dense_features, lr=0.002):
    dense_input = Input(shape=(num_dense_features, ), name='dense1')
    wday_input = Input(shape=(1,), name='wday')
    event_name_1_input = Input(shape=(1,), name='event_name_1')
    event_name_2_input = Input(shape=(1,), name='event_name_2')
    event_type_1_input = Input(shape=(1,), name='event_type_1')
    event_type_2_input = Input(shape=(1,), name='event_type_2')
    dept_id_input = Input(shape=(1,), name='dept_id')
    store_id_input = Input(shape=(1,), name='store_id')
    cat_id_input = Input(shape=(1,), name='cat_id')
    state_id_input = Input(shape=(1,), name='state_id')
    ---------------------------------------------------------------
    wday_emb = Flatten()(Embedding(7, 2)(wday_input))
    event_name_1_emb = Flatten()(Embedding(31, 2)(event_name_1_input))
    event_name_2_emb = Flatten()(Embedding(5, 2)(event_name_2_input))
    event_type_1_emb = Flatten()(Embedding(5, 2)(event_type_1_input))
    event_type_2_emb = Flatten()(Embedding(5, 2)(event_type_2_input))
    dept_id_emb = Flatten()(Embedding(7, 2)(dept_id_input))
    store_id_emb = Flatten()(Embedding(10, 3)(store_id_input))
    cat_id_emb = Flatten()(Embedding(3, 2)(cat_id_input))
    state_id_emb = Flatten()(Embedding(3, 2)(state_id_input))
    ---------------------------------------------------------------
    x = concatenate([dense_input, wday_emb, event_name_1_emb,
                     event_type_1_emb,  event_name_2_emb,
                     event_type_2_emb, dept_id_emb,
                     store_id_emb, cat_id_emb, state_id_emb])

    x = Dense(256*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(256*2, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = BatchNormalization()(x)
    x = Dense(128*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(128*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(64*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(16*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.1)(x)
    x = Dense(4*2, activation="relu")(x)
    x = BatchNormalization()(x)
    ---------------------------------------------------------------
    outputs = Dense(1, activation="linear", name='output')(x)
    return Model(inputs = ..., outputs)
```

```
def keras2(num_dense_features, lr=0.002):
    dense_input = Input(shape=(num_dense_features, ), name='dense1')
    wday_input = Input(shape=(1,), name='wday')


    event_type_1_input = Input(shape=(1,), name='event_type_1')
    event_type_2_input = Input(shape=(1,), name='event_type_2')
    dept_id_input = Input(shape=(1,), name='dept_id')
    store_id_input = Input(shape=(1,), name='store_id')
    cat_id_input = Input(shape=(1,), name='cat_id')
    state_id_input = Input(shape=(1,), name='state_id')
    ---------------------------------------------------------------
    wday_emb = Flatten()(Embedding(7, 2)(wday_input))


    event_type_1_emb = Flatten()(Embedding(5, 2)(event_type_1_input))
    event_type_2_emb = Flatten()(Embedding(5, 2)(event_type_2_input))
    dept_id_emb = Flatten()(Embedding(7, 2)(dept_id_input))
    store_id_emb = Flatten()(Embedding(10, 3)(store_id_input))
    cat_id_emb = Flatten()(Embedding(3, 2)(cat_id_input))
    state_id_emb = Flatten()(Embedding(3, 2)(state_id_input))
    ---------------------------------------------------------------
    x = concatenate([dense_input,wday_emb,
                     event_type_1_emb,  event_type_2_emb,
                     dept_id_emb,  store_id_emb,
                     cat_id_emb,  state_id_emb])

    x = Dense(256*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(256*2, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = BatchNormalization()(x)
    x = Dense(128*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(128*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(64*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(16*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.1)(x)
    x = Dense(4*2, activation="relu")(x)
    x = BatchNormalization()(x)
    ---------------------------------------------------------------
    outputs = Dense(1, activation="linear", name='output')(x)
    return Model(inputs = ..., outputs)
```

```
def keras3(num_dense_features, lr=0.002):
    dense_input = Input(shape=(num_dense_features, ), name='dense1')
    wday_input = Input(shape=(1,), name='wday')
    event_name_1_input = Input(shape=(1,), name='event_name_1')
    event_name_2_input = Input(shape=(1,), name='event_name_2')
    event_type_1_input = Input(shape=(1,), name='event_type_1')
    event_type_2_input = Input(shape=(1,), name='event_type_2')
    dept_id_input = Input(shape=(1,), name='dept_id')
    store_id_input = Input(shape=(1,), name='store_id')
    cat_id_input = Input(shape=(1,), name='cat_id')
    state_id_input = Input(shape=(1,), name='state_id')
    ---------------------------------------------------------------
    wday_emb = Flatten()(Embedding(7, 2)(wday_input))
    event_name_1_emb = Flatten()(Embedding(31, 1)(event_name_1_input))
    event_name_2_emb = Flatten()(Embedding(5, 1)(event_name_2_input))
    event_type_1_emb = Flatten()(Embedding(5, 1)(event_type_1_input))
    event_type_2_emb = Flatten()(Embedding(5, 1)(event_type_2_input))
    dept_id_emb = Flatten()(Embedding(7, 2)(dept_id_input))
    store_id_emb = Flatten()(Embedding(10, 3)(store_id_input))
    cat_id_emb = Flatten()(Embedding(3, 2)(cat_id_input))
    state_id_emb = Flatten()(Embedding(3, 2)(state_id_input))
    ---------------------------------------------------------------
    x = concatenate([dense_input,wday_emb,  event_name_1_emb,
                     event_type_1_emb,  event_name_2_emb,
                     event_type_2_emb,dept_id_emb,  store_id_emb,
                     cat_id_emb,  state_id_emb])

    x = Dense(256*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(256*2, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = BatchNormalization()(x)
    x = Dense(128*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    x = Dense(128*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(64*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(16*2, activation="relu")(x)
    x = BatchNormalization()(x)
    x = Dropout(0.1)(x)
    x = Dense(4*2, activation="relu")(x)
    x = BatchNormalization()(x)
    ---------------------------------------------------------------
    outputs = Dense(1, activation="linear", name='output')(x)
    return Model(inputs = ..., outputs)
```

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation* (pp. 265–283).

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., & Vollgraf, R. (2019). FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics (Demonstrations)* (pp. 54–59).

Al-Rfou, R., Perozzi, B., & Skiena, S. (2013). Polyglot: Distributed word representations for multilingual NLP. arXiv preprint arXiv:1307.1662.

Ben Taieb, S., Hyndman, R. J., et al. (2012). *Recursive and direct multi-step forecasting: The best of both worlds, vol. 19*. Citeseer.

Bengio, Y., & Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal Of Machine Learning Research*, *5*(Sep), 1089–1105.

Bishop, C. M., et al. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Chollet, F., et al. (2018). *Deep learning with Python, vol. 361*. New York: Manning.

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.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals Of Statistics*, *29*(5), 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*(4), 367–378.

Howard, J., & Gugger, S. (2020). Fastai: A layered API for deep learning. *Information*, *11*(2), 108.

Hyndman, R., & Athanasopoulos, G. (2018). *Forecasting: principles and practice, 2nd edition*. Melbourne, Australia: OTexts, OTexts.com/fpp2 Accessed on 10/10/2021.

Hyndman, R. J., Athanasopoulos, G., et al. (2014). Optimally reconciling forecasts in a hierarchy. *Foresight: The International Journal Of Applied Forecasting*, (35), 42–48.

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688.

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).

Kim, T., Hong, J., & Kang, P. (2015). Box office forecasting using machine learning algorithms based on SNS data. *International Journal of Forecasting*, *31*(2), 364–390.

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). The M5 accuracy competition: results, findings and conclusions. *International Journal of Forecasting*, http://dx.doi.org/10.1016/j.ijforecast.2021.11.013, in this issue.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703.

Potdar, K., Pardawala, T. S., & Pai, C. D. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal Of Computer Applications*, *175*(4), 7–9.

Tweedie, M. C., et al. (1957). Statistical properties of inverse Gaussian distributions. I. *Annals Of Mathematical Statistics*, *28*(2), 362–377.

Ye, J., Chow, J.-H., Chen, J., & Zheng, Z. (2009). Stochastic gradient boosted distributed decision trees. In *Proceedings of the 18th ACM conference on information and knowledge management* (pp. 2061–2064).

Zhou, H., Qian, W., & Yang, Y. (2020). Tweedie gradient boosting for extremely unbalanced zero-inflated data. *Communications In Statistics-Simulation And Computation*, http://dx.doi.org/10.1080/03610918.2020.1772302.