Time Series Prediction on E-commerce Sales Data

Given OList E-commerce dataset use Deep feed-forward neural network, LSTM Neural Network, XGBoost Regression, Random Forest Regression to predict revenue of e-commerce website OList.

Introduction:

Time series prediction has always received special treatment when compared with other prediction problems. Its temporal nature makes it different from the others. We are trying to predict a time series of sales from OList, a Brazilian e-commerce website. The data only covers one and a half years, making it a bad representative of yearly seasonality. The small amount of data is also likely to overfit the models trained on it. Furthermore, It is a growing time series hence it is also essential for models to learn the growth patterns in it.

We are evaluating four techniques for this prediction: XGBoost regression, Random Forest regression, LSTM neural network, and Deep Neural Network. Except for LSTMs, none of them are traditionally used for time series prediction. The XGBoost and Random forest typically need small amounts of data to train which makes them good candidates. Although LSTMs are used in time series problems we would like to see how challenging it could be for them to work in such a small amount of data. A simple Deep Learning model is tried to see where it stands when compared with other sophisticated techniques.

We tried the techniques for different variations of the same problem. Various window sizes(no. of past days) are provided as input. Each window size is used to predict a different sales value. That is accumulated sales of 1, 7, 14, and 30 days in the future. We discuss the results and try to give explanations for them and also evaluate for ease of use (for LSTM and Deep Learning) and implementation(For XGBoost and Random Forest).

Relevant Work:

Transformer is the newest addition in the favorite choices for series prediction and it took the spot for being the most favorite choice. The benefit of a transformer over recurrent nets is that it processes the whole input at once. This lets the transformer naturally give equal importance over the inputs at all time steps than just the recent ones. Other models like plain deep neural networks can take the whole input at once like the transformers but that may make them over parameterized and very difficult to train. Transformers need a large input to perform well and hence have a huge number of parameters. This makes transformer training difficult and time-consuming. Hence for the smaller data size transformers are reconsidered in case of small datasets.

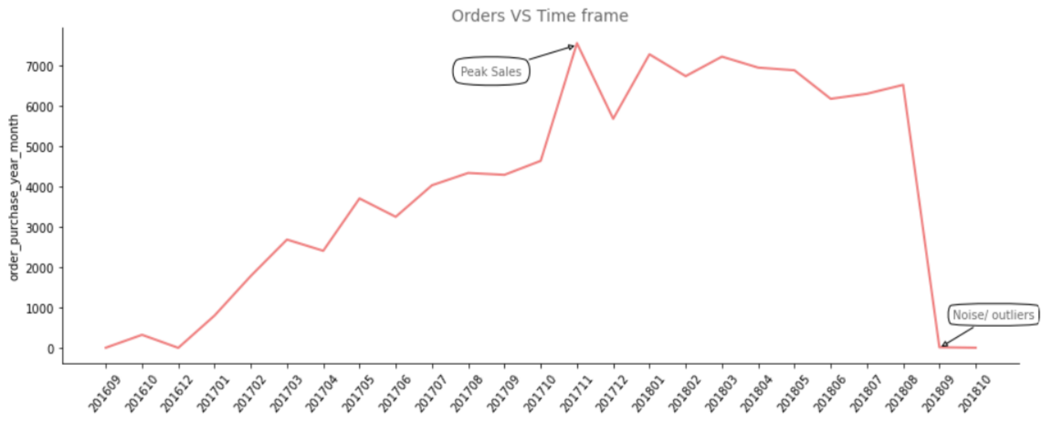
Long Short-Term Memory (LSTM) has been long known to solve the time series prediction problem in this research space. The paper: ‘Applying LSTM to Time Series

Predictable through Time-Window Approaches'[4], explores LSTM's application on the mentioned problem. Here the approach was taking every time step by looking at some past observations. One major observation is LSTM does not show much improvement even with some tasks that can be solved with timed window approaches. Consequently, the argument is found that if lagged observations are close to the time being forecasted, LSTM may not be the best choice for forecasting. The problem that we are solving deals with less amount of data and is also susceptible to seasonality, meaning that the observations are close to the time at which they are being forecasted. We further explore by applying LSTM as one of the implementation methods for our problem.

Decision Tree algorithms, along with SVM, are commonly used to development of the regression analysis models, in addition to solving data classification problems. The paper "Development and Research of the Forecasting Models Based on the Time Series Using the Random Forest Algorithm " discusses the random forest algorithm to the problem of predicting the remaining useful life of the equipment of a complex technical system by developing the appropriate regression model using datasets based on multivariate time series.

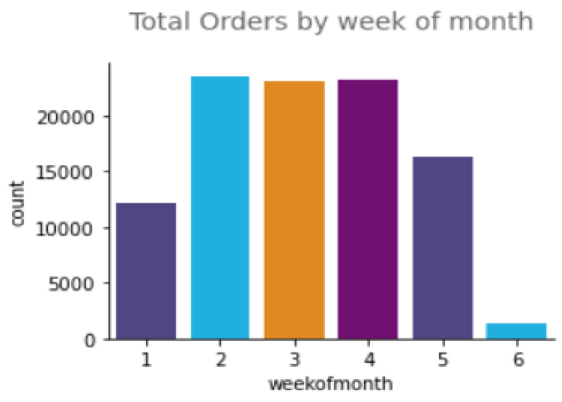
Preprocessing and Data visualization:

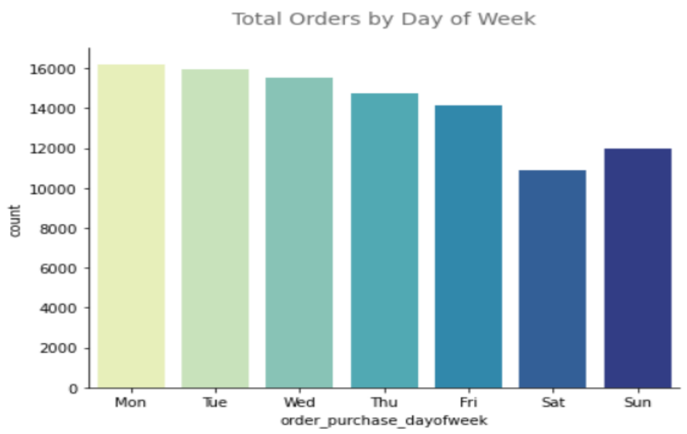
- We preprocessed the original data to make it usable for the problem statement.
    - The original data length is from Sept'16 to Oct'18. After visualization, we inferred that it was incomplete for the first four and last two months. We observed noise/outliers and missing values in the last segment ie. after Sept' 18 and before Jan '17, Hence, we considered the data lying between Jan' 17 - Aug' 18.



    -
    - The original data did not directly give the sales for a day. The data was divided into 2 parts:
        - olist_orders_dataset
        - olist_order_payments_dataset
    - We joined both tables to get the order purchase timestamp and payment value in the same table.

- We then aggregated the payment value by order purchase timestamp in such a way that we got payments received for the respective dates.
- After data visualization we observed,
  - Mondays are when most orders are placed
  - Weekends are when the least orders are placed.





Feature Engineering:

- We have selected features for our models.
  - We primarily found the date time-related features to be useful towards our goal. The rest of the features do not affect the sales at a specific time.
  - The feature that we selected are as follows:
    - Sales for past (1 or 7 or 30) day/s.
    - To be predicted date's day of the week.
    - To be predicted date's week of the month.
    - Whether the predicted date is a weekend or not.
  - Since the timeline of data is smaller than 2 years we couldn't learn from features that depend on the part of a year

Architecture Experiments for LSTM:

- Here we experiment with the model architecture to solve the given problem, and choose the one giving comparable/best performance.
- Note: The hyperparameter tuning was done for all the architectures below except simple architecture

1. Simple 1-layer architecture:
   - We added a single layer of LSTM using keras framework, with a single dense layer for the output.
   - This architecture was chosen as a baseline since it shows low complexity with no tuned hyperparameters.

2. Stacked LSTM architecture with dropout:
    - We created a *2_layers x LSTM* + *Dropout* + *Dense* which adds another LSTM layer to the existing architecture, an additional layer of dropout and one Dense layer.
    - Stocking the layers as above was done with the intention to make the network deeper and to identify more complex patterns in the prices in the dataset if any.
    - In addition to this we added a dropout layer as well with the aim of adding a little more randomness to avoid the model from overfitting.
    - The dropout rate along with other hyperparameters were tuned and explained with more detail below.

3. Stacked LSTM architecture:
    - We created a similar structure of *2_layers x LSTM* + *Dense* as described above but without the dropout layer.
    - The addition of the dropout layer wasn't proving as a benefit as we observed in the experiments conducted.
    - Among the mentioned architectures, this performed well than others consistently with more accuracy and less error

Hyperparameter Tuning for LSTM:

- We came across various hyperparameters in our model that we could experiment with all those are listed below with the configurations we tried for the same:

- Number of neurons(units):
    - This was the input for LSTM as well as the dense layer.
    - Since there is no way of knowing the specific number of neurons for each layer given the complexity of model and problem, we tried values or 32, 64, 128, and 256 for all the layers.
    - Keeping the network more dense in the beginning we found that 128 , 64, 32 gave comparably better results for 2 lstm and dense layer respectively.

- Dropout rate:
    - The dropout rate ranges from 0.1 to 1 where it refers to the percentage of neurons we want to hide.
    - We found that keeping the dropout rate as 0 i.e not using dropout was more effective.
    - This brings in light the fact that the network has less data to work with which makes it less susceptible to overfitting. Thus it;s prevention would not be necessary with the dropout layer.

- Activation:

    - Similar to neurons we cannot determine first hand which activation can work better for which set of problem, although in some cases we see Relu being faster and empirically more correct.
    - We conducted experiments with Tanh, Sigmoid, Softmax and Relu.
    - Here Relu performed better than the rest for the layers which were used to create the model

- Optimizer:
    - This is one of the more important hyperparameters which helps in reducing the loss and increasing the accuracy.
    - We looked for an adaptive optimizer that would subsequently take care of the learning rate, which is another hyperparameter. Thus we found the 'adam' optimizer that suits well for our use case.
    - We still compared adam with various other optimizers: RMSprop , Adadelta, Adagrad, Adamax, Nadam and Ftrl but expectedly Adam performed better than others.

- Loss function:
    - We know that we are solving a regression problem here and thus experimented with the loss functions: Mean squared error , huber loss, mean squared logarithmic error , mean absolute error.
    - Comparable results were gained for both 'mean squared error' and 'mean absolute error, but the latter performed more consistently when for more number of trials on the dataset.

- Go_backwards:
    - Aiming to experiment with the input sequence we made this parameter 'true' in the first layer of LSTM (since we only care about inputs), for the model to learn more.
    - This proved good for the window size of 30 days because we model was subsequently trained to get a view of a much more complex pattern by the reversing of the inputs.

- Recurrent_regularizer:
    - Here we aim to regularize the recurrent connections in the network by using l1 and l2 loss.
    - We used the values of 0.01 for both the losses to minimize the error of absolute differences and deviations.
    - This also showed improvement in the 30 day window used in the experimentation.

Experimental Evaluation

- These algorithms are evaluated by the following evaluation criteria.
    - Performance of model on different input sizes. That is data from the past 1 day, 7 days, 14 days and 30 days as seen in below figures.
    - Performance of model for predicting sales of different tenures. That is cumulative sales of the next 1 day, 7 days, 14 days and 30 days.
    - Mean error percentage from original value and standard deviation of it.
    - Comparison of these performances with Moving Average.
- We found that parameter tuning for XGBoost and Random forest did not improve the performance further hence the default parameters are used. We tuned the parameters for LSTM and DNN but still, their performance did not surpass the algorithms using trees.
- The XGBoost and Random continued to perform better for the given problem. One of the reasons could be because of the dataset size. LSTM and Deep Learning models are susceptible to overfitting when trained with smaller datasets.

Performance and Results:

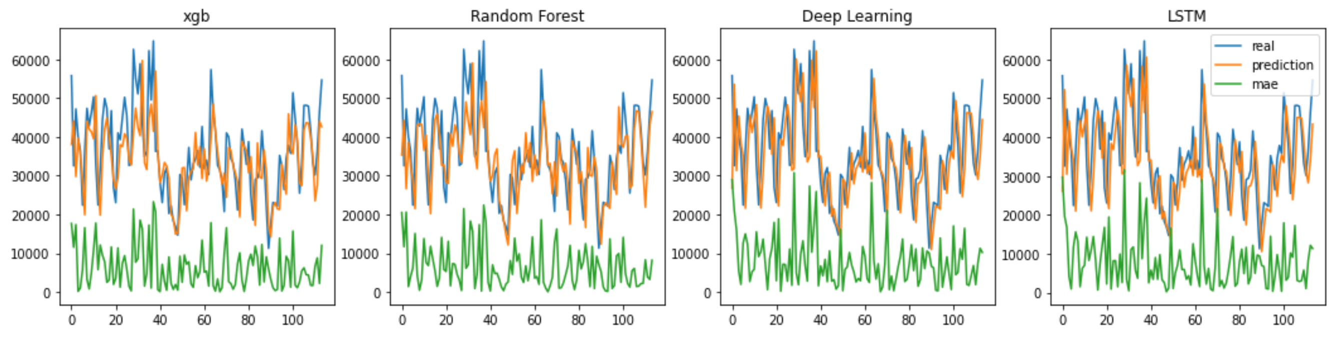


Figure 1: **Plots for: Prediction for 1 day with window size 1**

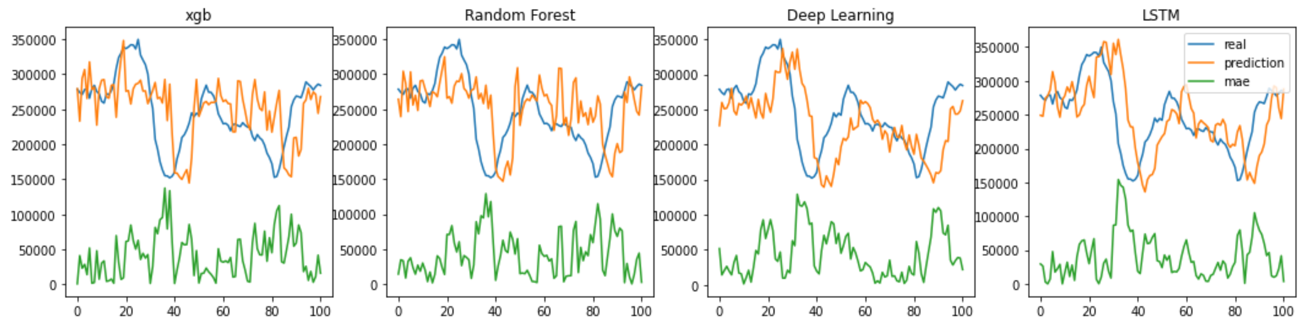


Figure 2: **Plots for: Prediction for 7 day with window size 7**

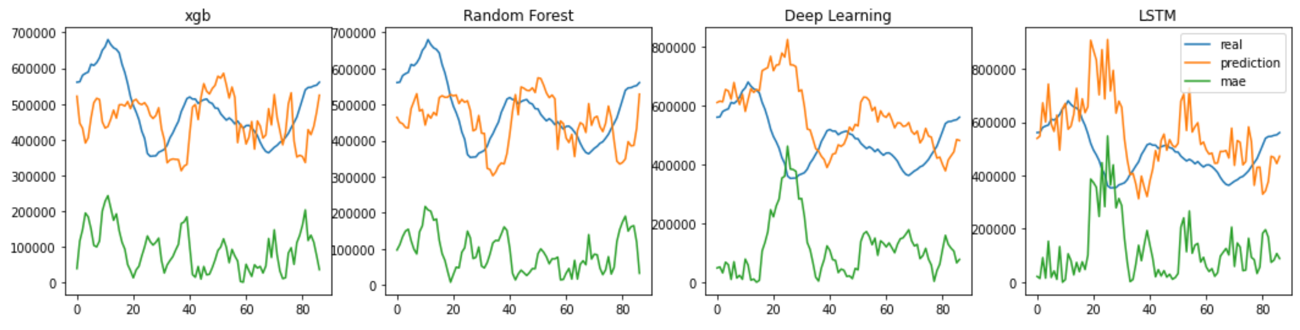


Figure 3: **Plots for: Prediction for 14 day with window size 14**

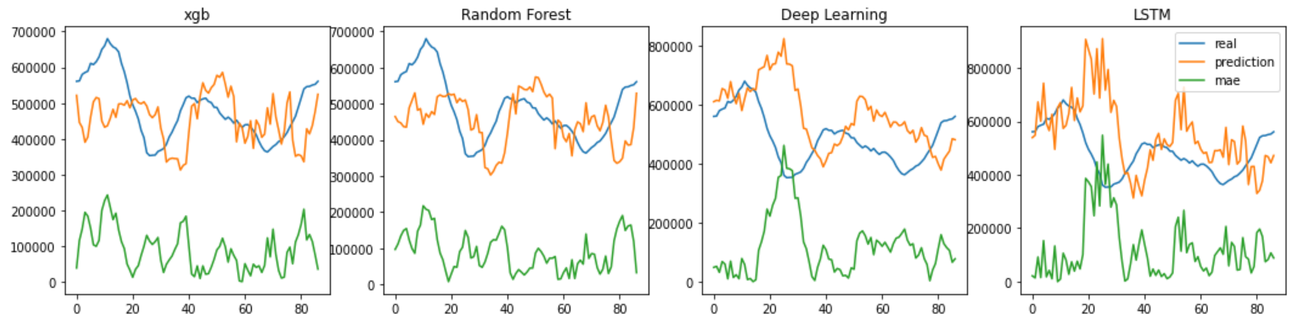


Figure 4: **Plots for: Prediction for 30 day with window size 30**

Mean and standard deviation of Mean absolute error for different techniques.

(Input days, Forecast window)	XGBoost		Random Forest		DNN		LSTM		Moving Avg.	
	Mea n	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
1,1	0.17	0.14	0.18	0.14	1.00	0.00	0.22	0.17	0.23	0.18
7,7	0.19	0.18	0.19	0.19	0.18	0.14	0.19	0.20	0.85	0.03
14,14	0.18	0.10	0.18	0.09	0.22	0.19	0.24	0.22	0.92	0.02
30,30	0.07	0.06	0.06	0.06	0.27	0.19	0.33	0.20	0.96	0.00

- As we can see almost all the algorithms are able to perform decently. (The DNN for 1,1 is an exception here, we observed that DNN is zeroing out its output occasionally. This might be due to the vanishing gradient.)
- 14,14 and 30,30 are smoother curves that are better predicted by XGB and random forest, the DNN and LSTM models have spikes in their prediction. We guess that this is due to the sparsity of training data. XGB and Random Forest are better with small data than the other two.

Conclusion:

- We found that parameter tuning for XGBoost and Random forest did not improve the performance further hence the default parameters are used. We tuned the parameters for LSTM and DNN but still, their performance did not surpass the algorithms using trees.
- XGBoost and Random Forest can be used for time-series data. Their specialty is robustness to the size of the dataset.
- The results we obtained for these techniques are not promising but that might be due to noise in the underlying data set.
- DNN and LSTM are difficult to train given the small and noisy dataset.
- The data that we used is noisy and it is difficult to filter out for the models when provided as is.

Future Work:

The data that we used for evaluating the techniques seems to be too small for the current techniques to be useful, for the same techniques to be fruitful we can try different time series preprocessing techniques to remove the noise.

For the smaller data sets, we recommend using XGBoost or Random Forest but these models won't scale with the inputs, and hence for each new batch of data, we should consider retraining these models.

For Neural networks we can try strong regularization techniques. These models are strong learners but can also learn the noise in the data.

References:

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794. ACM, 2016.

[2] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. 2020. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. arXiv:2001.08317 [cs.LG]

[3] Liliya Demidova and Mariya Ivkina. Development and Research of the Forecasting Models Based on the Time Series Using the Random Forest Algorithm. Lipetsk, Russia IEEE 2020 ISBN:978-1-7281-8113-4

[4] F. Gers, D. Eck, and J. Schmidhuber. Applying LSTM to time series predictable through time-window approaches. Technical report IDSIA-IDSIA-22-00, IDSI/USI-SUPSI, Instituto Dalle Molle di studi sull' intelligenza artificiale, Manno, Switzerland, 2000. http://www.idsia.ch/~felix/Publications.html.