COSC 6380 – Data Analytics
Fall 2020 Final Project

Name: Chan Dat Thai

1. Project description and dataset

This project aims to predict sales of certain products of certain stores over a period of time. The dataset used in this project is retrieved from this inactive Kaggle [1]. The dataset is relatively simple and clean, probably artificial. The dataset consists of 5 years of sales of 50 items in 50 stores. The goal of the competition is to predict 3 months of sales for all the items.

Nevertheless, the source code is written as most generalized as possible, so we can switch to another dataset if we want, as long as they have the same column names.

2. Exploratory Data Analysis

The data is relatively simple and straightforward, with only 4 columns: date, store ID, item ID, and number of items sold at that particular time.

Figure 1 shows the sale trend over time, we can clearly spot the yearly seasonality within it. The sales have an increasing trend overtime, but always have a spike at the middle of the year, and also a slight increase at around October. This trend is also depicted in the seasonal decompose components.

Then when we look at the autocorrelation and partial autocorrelation plots as in Figure 2, we could also see a second seasonality, which happens weekly at every 7-day mark.

Since the dataset has multiple seasonality, I decided to use TBATS, a method specifically designed to tackle this situation. The traditional SARIMA and SARIMAX models will also

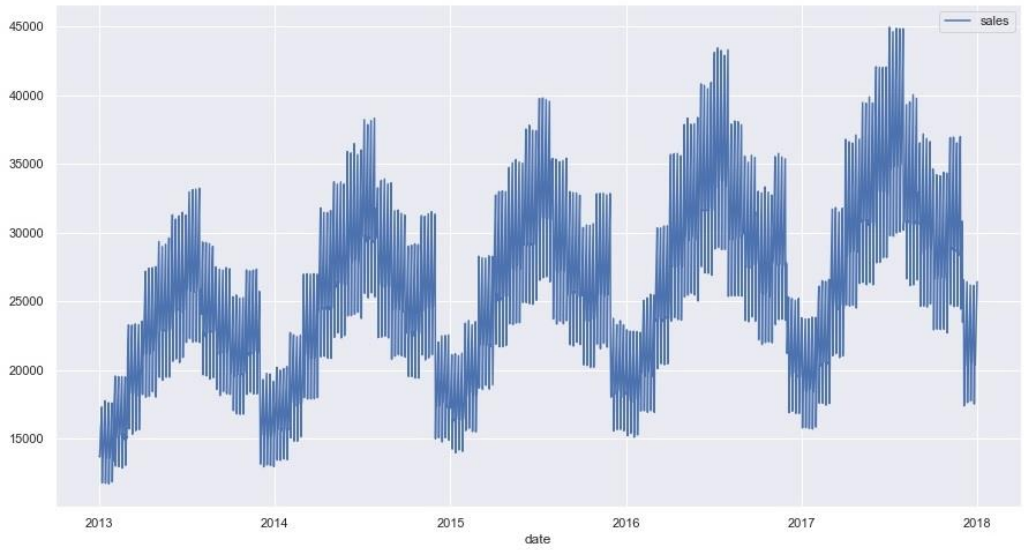

Figure 1 The sale trend over time

be implemented to compare the results. Apart from that, the competition actually implies that XGBoost might perform great on this dataset, so it will also be conducted.

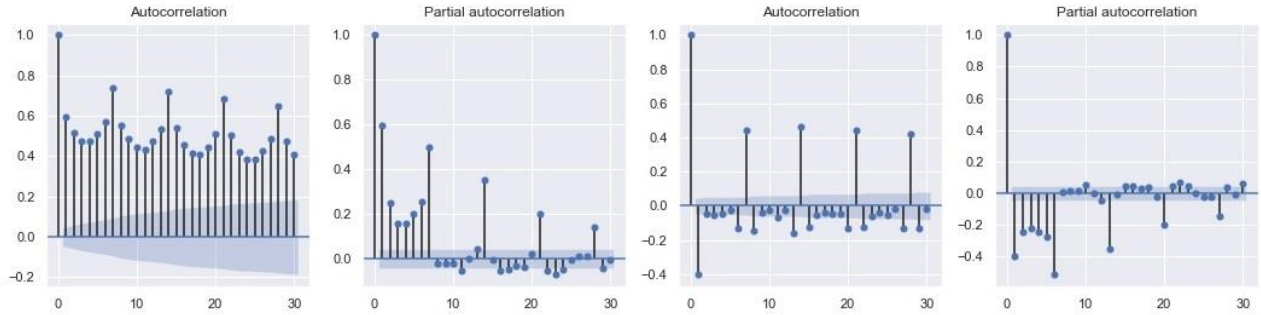

Figure 2 ACF and PACF

3. Methods

SARIMA

Seasonal Autoregressive Integrated Moving Average

SARIMA is an extension of the traditional ARIMA model, which supports the handling of the seasonal components of time series. Apart from 3 parameters of the ARIMA model, SARIMA has an additional of 4 parameters:

- P: Seasonal autoregressive order.
- D: Seasonal difference order.

- Q: Seasonal moving average order.
- m: The number of time steps for a single seasonal period.

Since we know that the dataset has a weekly seasonality, m should probably be 7 (m cannot be too large a period, so yearly is out of the question). A brute-force approach is also conducted to confirm this. The optimal parameters appear to be (0, 1, 1) (3, 0, 1, 7).

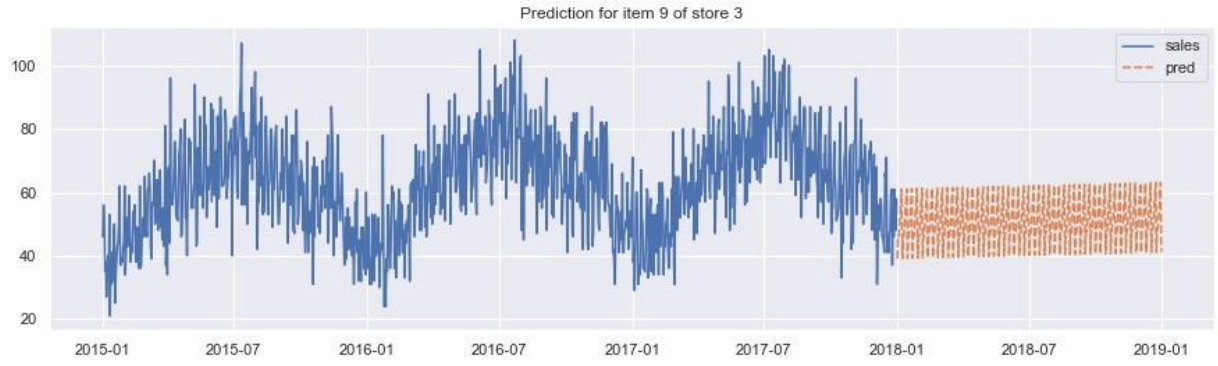

Figure 3 Prediction of SARIMA(0,1,1)(3,0,1,7)

Figure 3 shows the prediction of our SARIMA model. As expected, since SARIMA is not intended to handle multiple seasonality, it completely neglects the yearly seasonality.
To compensate for this, we need to explicitly tell it the extra component by incorporating it as an exogenous variable.

SARIMA with Fourier terms

The idea is originally from [2]. The model is implemented as the previous section, but with the addition of exogenous variables of Fourier terms. The model can be represented mathematically as

$$y_t = a + \sum_{k-1}^{K} \left[\alpha_k \sin \frac{2\pi kt}{m} + \beta_k \cos \frac{2\pi kt}{m} \right] + N_t$$

where N_t is an ARIMA/SARIMA process, and K can be chosen by minimizing AIC.
This approach complements two drawbacks of the traditional ARIMA/SARIMA models, which are inadequate for long seasonality and multiple seasonality. This allows us to work

with any length periodicity and multiple seasonality (we only need to add a separate Fourier term for each frequency).

Admittedly, I am far from familiar with Fourier terms, so I could not explain the intuition behind this, and even how it works. However, it apparently works, as depicted in Figure 4.

Figure 4 Prediction of SARIMAX with Fourier terms

With the Fourier term as the exogenous variable, the model is much better at capturing both seasonality. However, it apparently still could not fully capture the pattern. If we look closely at the training data, we could see a slight increase at around October period, but this does not appear in our prediction.

TBATS [3]

Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal components.

TBATS model is designed to handle complex seasonality, e.g. non-integer seasonality or long seasonality, using exponential smoothing.

Under the hood TBATS will consider various alternatives:

- with Box-Cox transformation and without it.
- with and without Trend
- with and without Trend Damping
- with and without ARMA(p,q) process used to model residuals
- non-seasonal model
- various amounts of harmonics used to model seasonal effects

The library is available on *pypi*, with a relatively simple and straightforward usage. We only need to supply it with our desired frequencies, and it will do all the heavy works for us. The tradeoff of such simplicity is obvious the lack of flexibility, since the API does not provide us many parameters, we have little control over how the API works. Another big drawback is the slow running time, since it needs to consider all above alternatives. It takes around 10.5 hours to run this on Riddler.

Figure 5 Prediction of TBATS model

Nevertheless, the result looks much promising in Figure 5, compared to previous SARIMA models. The model accurately captures both the peaks at July and October.

XGBoost

eXtreme Gradient Boosting is an implementation of gradient boosting machines. It is widely used in Kaggle competition, thanks to its execution speed and model performance. For XGBoost to work properly, we need to add more features to the original dataset. I did this by (1) extracting features from datetime, e.g. weekday, is weekend, day of year, week of year, month, quarter, is holiday,… and (2) the sales lag value, e.g. a month ago, a quarter ago, a year ago, and so on. The attributes also need to be converted to numerical values. This is accomplished using one-hot encoding.

Similar to previous models, a GridSearch is also performed to find out the best parameters for the regressor model. Unfortunately, the approach takes longer than I expected, and is still running at the time of writing. So, only the evaluation for the base model is available, but not the one with optimized parameters.

The library also supports GPU execution, but I failed to make it work on both OpenStack (incompatible GPU) and Google Colab (insufficient memory).

4. Performance evaluation

Figure 6 shows score results from Kaggle competition. The score is evaluated using the SMAPE metric. Out of all methods, TBATS is the closest one to the best model on leaderboard (12.58 vs 13.4 for TBATS). The competition was already ended 2 years ago, but the TBATS should rank around 261/459.

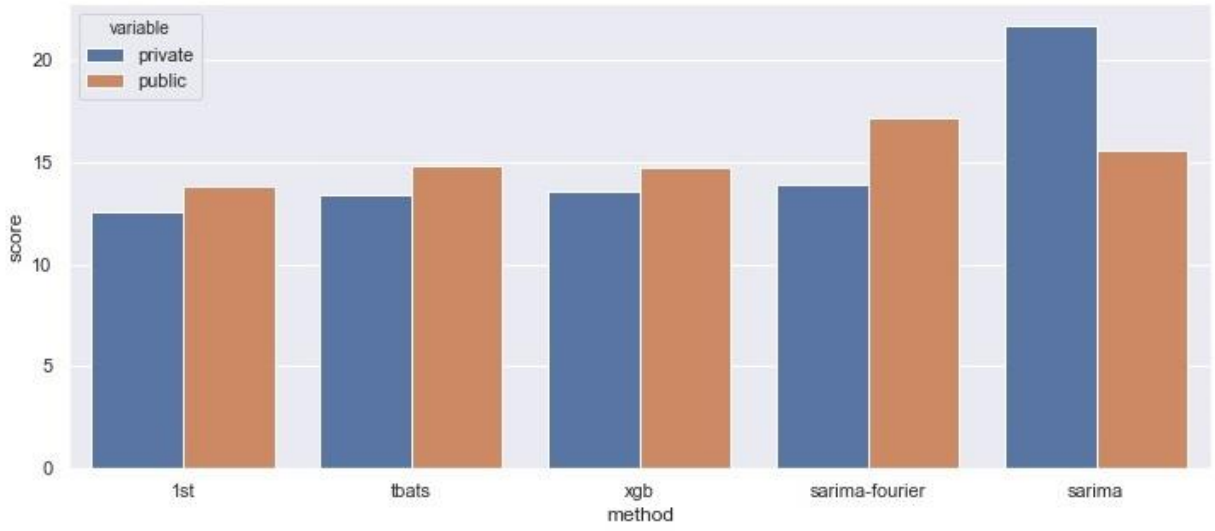

Figure 6 Comparison of all methods' final score (lower is better)

Table 1 provides exact score value, along with the time it takes to run each model.

Method	Score (private)	Time elapsed
SARIMA	21.672	2-3 hours*
SARIMA with Fourier terms	13.920	2-3 hours*
TBATS	13.396	10.5 hours
XGBoost	13.526	1-3 minutes
XGBoost (optimal)	N/A	1-2 days

**including GridSearch for optimized parameters*

Even though ***XGBoost*** takes the minimal work to set up and least time to train, its private score is quite close to TBATS. Unfortunately, the optimal version of it is still running at the time of writing, so I cannot include its result. Out of all models, however, ***XGBoost*** is the only one that has the potential to obtain better result, since there is little thing we can tune with the other models.

5. General discussion

A lot of ideas of this project follow this tutorial, but all the code is written by myself, except for the construction of Fourier terms. In hindsight, I should have picked a different dataset, since this dataset is probably overly ideal. I did try with another similar dataset, but the result is fairly poor, and I did not have enough time to retune it.

A big mistake I made is that I spent quite a bit of time to make the code run with multiprocessing on Riddler, but the performance was not improved as I expected. I'd suspect that the API has already utilized parallel programming under the hood, so doing it manually might even hurt the performance with the overhead cost of data passing between processes. However, it does have a peculiar behavior: it takes approximately the same amount of time for 4 processes and 8 processes, but I couldn't figure out why it is that way.

6. References

[1] "Store Item Demand Forecasting Challenge," Kaggle, [Online]. Available: https://www.kaggle.com/c/demand-forecasting-kernels-only. [Accessed 7 12 2020].

[2] R. J. Hyndsight, "Forecasting with long seasonal periods," 29 9 2010. [Online]. Available: https://robjhyndman.com/hyndsight/longseasonality/. [Accessed 07 12 2020].

[3] Alysha M De Livera, Rob J Hyndman, Ralph D Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *Journal of the American Statistical Association*, vol. 496, no. 1513-1527, p. 106, 2010.