Demand Forecasting: Time Series Modelling

Introduction to Time Series Forecasting

The time-series data arise when tracking business processes or metrics over a period of time. The significant difference between time series analysis compared to other modeling methods is that it involves carefully collecting the past data and studying the past observations rigorously to get the inherent structure of the series. In layman terms, time series modeling can be deemed as the art of predicting the future by learning from the past[1].

In our project, we primarily focus on three forecasting techniques - Triple Exponential Smoothing(TES), ARIMA, and Prophet. We used these methods to forecast product sales for future weeks for a grocery retailer. In this article, I will introduce how we built demand forecasting models using Prophet and TES.

Time Series Data

For selected 16 stores under a grocery chain, we have weekly sales in units for hundreds of products across 20 categories for a span of 122 weeks (approx. 2 years). We trained our models 112 weeks to generate forecasts for 10 weeks into the future.

1. Triple Exponential Smoothing

1.1 About TES

Also known as the Holt-Winters seasonal method, Triple Exponential Smoothing consists of a forecast equation and three smoothing equations - level $\boldsymbol{l_t}$, trend $\boldsymbol{b_t}$, and seasonal component $\boldsymbol{s_t}$, with respective smoothing parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$* and $\boldsymbol{\gamma}$.

There exist two variations on how to account for seasonality in this method - additive and multiplicative. The additive method is preferred when seasonal variations are consistent through the series, for e.g. peaked sales of a particular product during the holiday season every year. The multiplicative method is used when the seasonal variations are proportional to a series level which is quite uncommon in a retail setting. We use the additive method in our analysis, the component form is as follows:

$$
\begin{aligned}
\hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h-m(k+1)} \\
\ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},
\end{aligned}
$$

[1] T. Lursinsap, P. Sanguanbhoki, “Application of critical support vector machine to time series prediction”, Circuits and Systems, 2003.

where $\boldsymbol{l_t}$ is the level (or the smoothed value) of the series at time ***t***.
m denotes frequency for seasonality, i.e., the number of seasons in a year. For e.g, m=4 for quarterly , and m=12 for monthly
Unlike ARIMA, stationarity of the series is not a condition in exponential smoothing.

1.2 Model Hyper-parameters[2]

Exponential smoothing forecasting methods are similar to ARIMA in that a prediction is a weighted sum of past observations. However, ES models explicitly use an exponentially ***decreasing*** weight for past observations contrary to ARIMA.

trend: Set to *None* as default. It can be set as 'add' or 'mul' depending on domain expertise and observation. In our case, trend component was set as None as we observed no improvement with other values

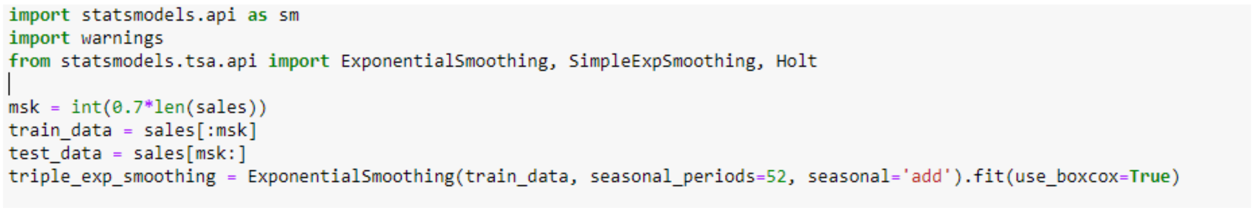
seasonal: Set to 'add' as we observed our seasonal component is additive in nature (also retail data have additive seasonality)

damped: It decides whether the trend component should be damped or not. It affects the trend of the forecast. It can be used in cases where we do not expect the sales to grow year over year with the same increasing (or decreasing) trend in which case we damp the forecast that was created using the historical trend. We left it untouched i.e. *None* as we didn't see any improvement in forecasting with varied values.

seasonal_periods: The number of periods in a complete seasonal cycle. As we are dealing with weekly data we set it as 52 i.e. 52 weeks in a yearly cycle.

1.3 Boxcox Transform[3]

We applied box cox transform to obtain the suitable configurable transform (square root or log) for our series while fitting using exponential smoothing

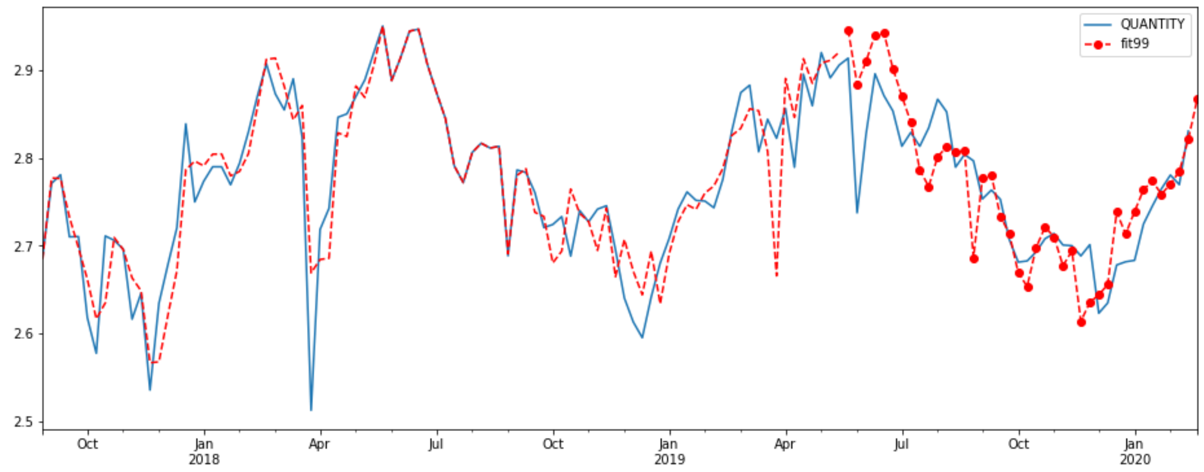


```python
import statsmodels.api as sm
import warnings
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt

msk = int(0.7*len(sales))
train_data = sales[:msk]
test_data = sales[msk:]
triple_exp_smoothing = ExponentialSmoothing(train_data, seasonal_periods=52, seasonal='add').fit(use_boxcox=True)
```

[2] https://www.statsmodels.org/stable/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html?highlight=statsmodels%20tsa%20holtwinters#id1
[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html

1.4 Implementation of TES

```
fit99 = ExponentialSmoothing(train_data, seasonal_periods=52, seasonal='add').fit()

#Forecast & fitted for fit 99
rev99.plot(label='Actual', figsize=(16,6)) #actual sales
fit99.fittedvalues.plot(style='--', color='red',figsize=(16,6)) #training
fit99.forecast(40).plot(style='--', marker='o', color='red', label='fit99',legend=True, figsize=(16,6)) #forecasts
plt.show()
```



1.5 Challenges

1. ***Too simplistic for our case***: The forecast is almost an exact replication of the historical trend. If you look at the graph above closely, the forecasts (red dotted lines) for each of the months have the same peaks and falls as the training data (red smooth lines) for the same months.
2. No additional variable can be added

2. Prophet

Prophet is an open-source forecasting tool developed by Facebook's Data Science team, available in both Python and R. Prophet[4] is designed to create a great number of reliable forecasts by combining automatic forecasting with "analyst-in-the-loop". It begins by modeling a time series using hyperparameters specified by analysts, producing forecasts, and then evaluating them. In other words, it can efficiently automate the forecast task at scale without much manual effort.

2.1 Why is Prophet good for retail unit sales?

The prophet is based on multiple decomposable models: trend, seasonality, holiday, and events.

[4] https://facebook.github.io/prophet/

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

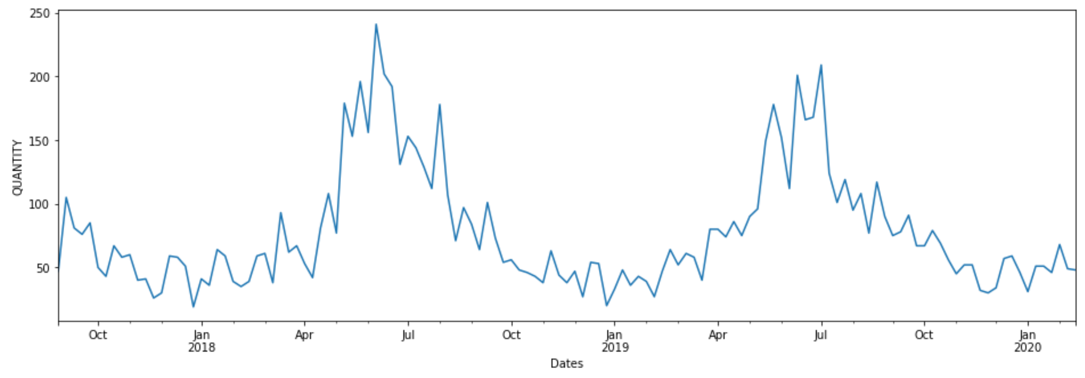
$g(t)$: Non-periodic changes

$s(t)$: Periodic changes (e.g. weekly/yearly seasonality)

$h(t)$: Effects of holidays

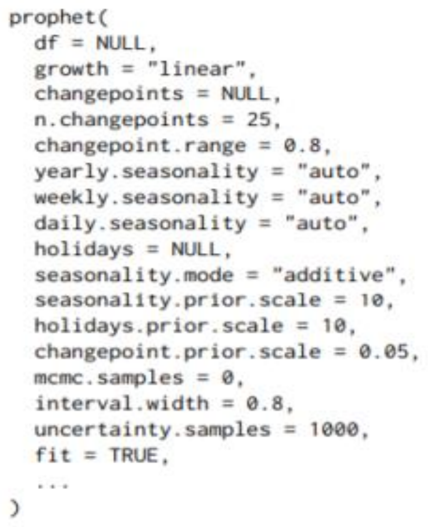
ε_t: Any unusual changes not accommodated by the model

Therefore, Prophet works efficiently with time series having strong seasonal effects, rich historical data, important holidays known in advance, and a limited number of missing observations, which are exactly the characteristics of retail sales data. Below is a time series plot that shows trend and seasonality in retail sales data.



2.2 Hyperparameters optimization for Prophet model

To optimize the Prophet model, first of all, we need to take a look at the hyperparameters and understand their meanings, instead of simply creating many loops and changing the values. Our goal is to adjust hyperparameters to obtain good enough predictions but at the same time avoid overfitting. Prophet has many tuning parameters that can be tweaked by users based on their domain knowledge. The following are the hyperparameters found in Prophet:



```
prophet(
  df = NULL,
  growth = "linear",
  changepoints = NULL,
  n.changepoints = 25,
  changepoint.range = 0.8,
  yearly.seasonality = "auto",
  weekly.seasonality = "auto",
  daily.seasonality = "auto",
  holidays = NULL,
  seasonality.mode = "additive",
  seasonality.prior.scale = 10,
  holidays.prior.scale = 10,
  changepoint.prior.scale = 0.05,
  mcmc.samples = 0,
  interval.width = 0.8,
  uncertainty.samples = 1000,
  fit = TRUE,
  ...
)
```

Growth

Growth can be ’linear’ or ’logistic’ specifying the nature of the trend. It can be found by plotting the trend and visually analyzing it. Based on the data trend and the fact that our model uses sales data, our model has ‘linear’ growth.

Changepoints

Changepoints are the points in the data where there are sudden and abrupt changes in the trend. By default, Prophet will automatically detect changepoints and will allow the trend to adapt appropriately. However, we wish to have finer control over this process, as Prophet may miss a rate change or be overfitting rate changes in history. Then there are four hyperparameters we can use:

changepoint hyperparameter is used when we manually specify the locations of potential changepoints, telling Prophet these specific dates are known to be likely to have changed. We would not recommend doing this as in our case, we observed that Prophet determined them automatically and provided better results.

n_changepoints allows us to set the number of potential changepoints. By default, the n_changepoints is 25. From our experience, the default Prophet model is too smooth and not flexible enough to fit the weekly sales data. However, even if we maximize n_changepoints, i.e., all data points are changepoints, the results are still under fitted, so we set it as default and try other hyperparameters.

changepoint_range, by default, changepoints are only inferred for the first 80% of the time series in order to have plenty of runway for projecting the trend forward and to avoid overfitting fluctuations at the end of the time series. From our experience, this hyperparameter doesn’t have much impact on the model fitting, so we also set it as default value.

changepoint_prior_scale adjusts the strength of the sparse prior, which indicates how flexible the changepoints are allowed to be. By default, this hyperparameter is set to 0.05. As our model is underfitting, we increase it to 1.0 to make the trend more flexible. As a result, we prevented overfitting.

Seasonality

The easiest control of seasonality is to set hyperparameters of yearly_seasonality, weekly_seasonality, and daily_seasonality. Our model provided the best results when we set our ***year_seasonality*** as True and rest as False. In addition, there are other hyperparameters like seasonality_mode and seasonality_prior_scale to help us get more power and control over seasonality, but we don’t dive deep into them.

Additional Regressor

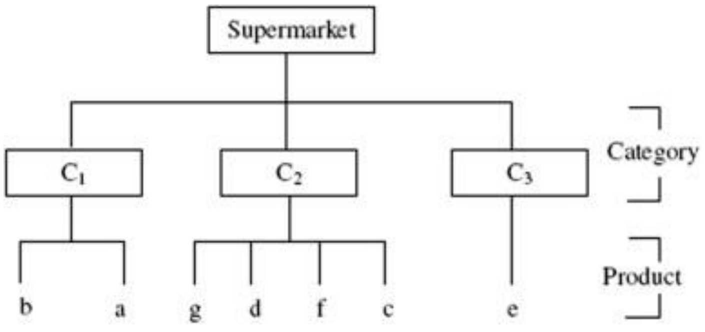
Prophet has the flexibility to add an additional regressor to the linear part of our model using the add_regressor method. We included **Price^(-Elasticity)** in order to take the influence of price and elasticity into account of our sales forecasting. The regressor value must be known for both the history and for future dates, which means if we want to forecast future sales, future prices would have to be calculated beforehand.

The code snippet below is how to train the model and get predictions

```
# train model
prophet_model = Prophet(yearly_seasonality=True, interval_width=0.95, n_changepoints = 25, changepoint_prior_scale = 1)
prophet_model.add_regressor('PRICE_TO_MINUS_ELASTICITY')
prophet_model.fit(sum_sales_train)

future = prophet_model.make_future_dataframe(periods = int(TEST_PERIOD), freq = "7d")
future["PRICE_TO_MINUS_ELASTICITY"] = sum_sales["PRICE_TO_MINUS_ELASTICITY"].reset_index(drop = True)
prediction = prophet_model.predict(future)
prediction_oos = prediction[(prediction["ds"]>=TEST_START)&(prediction["ds"]<=TEST_END)]
```

2.3 Prophet Model Implementation and Evaluation

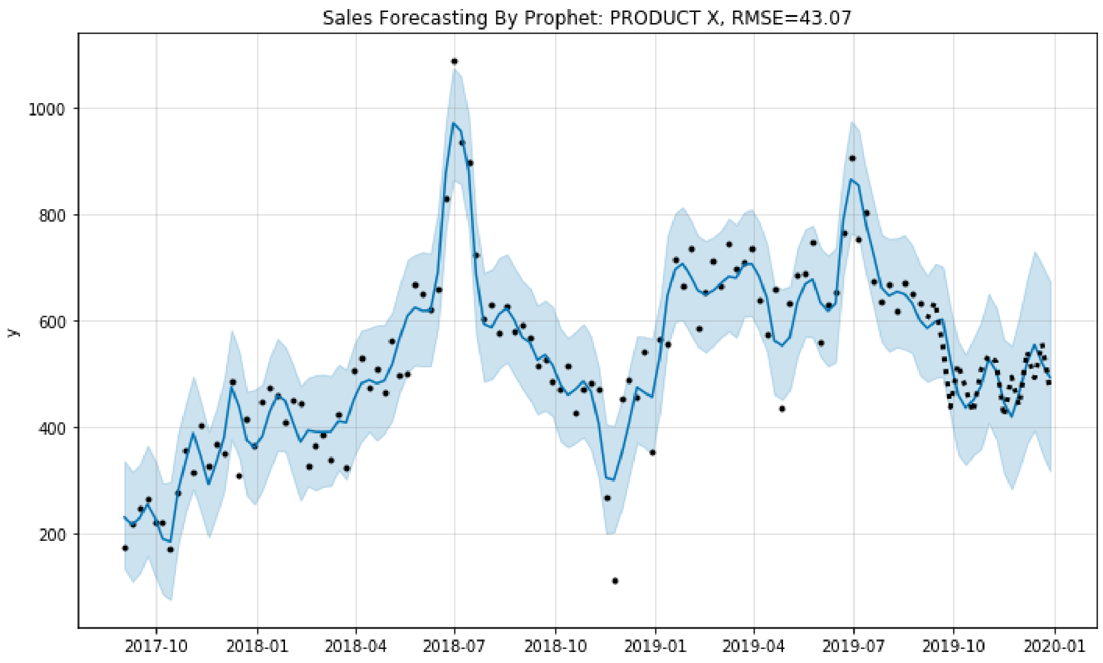
Since retail data usually follows a category-product hierarchy, the best practice is to implement the forecasting model on data of the lowest level of the hierarchy with no aggregation, i.e., we applied the Prophet model on sales of a specific product in a specific category in a given store. Later we can always aggregate the sales predictions on the product level to obtain the predictions on the category/store level.



RMSE: Like all other forecasting models, we first applied the Prophet model on the training data set using hyperparameters mentioned above, and then fit the model on the test data set. We evaluate model performance by computing Root Mean Square Error (RMSE) of the forecasting results of the test data set. Notably, it is also convenient to look at the result using a generic plot function.

Plot: Below is the plot of sales prediction for a product where we observed good forecasting performance. The test period is from September 07, 2019 to December 28, 2019, and the training period is prior to the test. The back dots represent actual sales. The dark blue curve is the forecasting result of our Prophet model and the black dots represent actual sales.

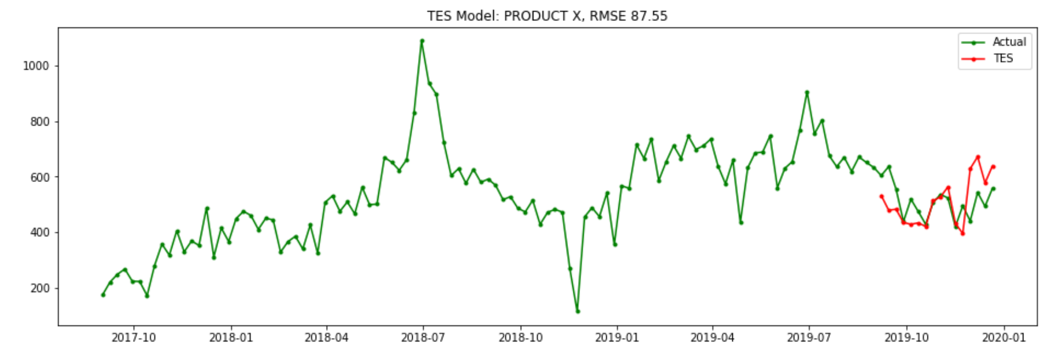
```
#plot
prophet_model.plot(prediction)
```

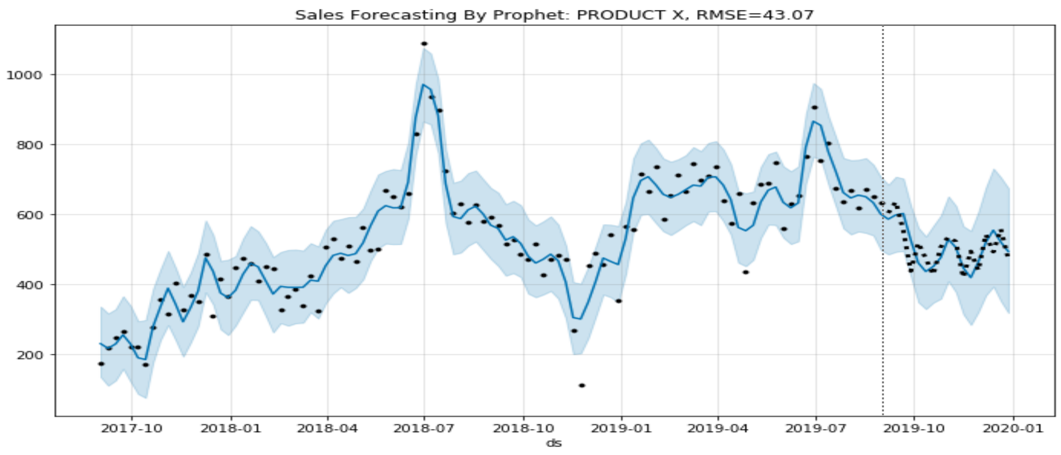


The light blue band is the confidence interval of the Prophet model. In some cases, the confidence interval can be used to identify outliers by looking at data points that fall out of the interval

3. Comparison

Prophet outperformed other methods in three major aspects - ***accuracy, flexibility and scalability***. Given the scope of the paper, we chose one product which is representative of the overall performance of these techniques. To better comprehend our analyses results, let's consider a product X and its forecast for one of the 16 stores. As it can be seen in the plots below. Prophet gave the closest approximation to actual sales although both TES and Prophet have low RMSE.


TES Model: PRODUCT X, RMSE 87.55
Actual
TES
1000
800
600
400
200
2017-10
2018-01
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01


Sales Forecasting By Prophet: PRODUCT X, RMSE=43.07
1000
800
600
400
200
2017-10
2018-01
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01
ds