Forecasting with FB Prophet

By: Raj Laxmi Prakash

Time series analysis helps to solve business problems which are time-based and identify this time-based pattern in the future. Techniques of time series forecasting could answer business questions like how much inventory to maintain, what will be sales for next month, what will be the demand of products, what will be the website traffic in the e-store and likewise. The basic objective of time series analysis usually is to determine a model that describes the pattern of the time series data which can help in forecasting the data.

A time series is modeled through a stochastic process Y(t), i.e a sequence of random variables. In a forecasting setting we find ourselves at time t and we are interested in estimating Y(t+h), using only information available at time t.

One of the traditional methods to forecast the time series data is through ARIMA model. I have posted an article on ARIMA model. One can find the details and python implementation of the ARIMA method in my previous post on LinkedIn.

In this article, I will be talking about FB Prophet, an open sourcing forecasting tool developed by Facebook Inc. This method takes two input from data set. The date stamp, which is abbreviated as ‘ds’ and the dependent variable ‘y’, which need to be forecasted. This article provides you details about FB Prophet, and a comparison between FB Prophet and ARIMA model output.

For the analysis purpose, I have taken Walmart sales dataset from Kaggle. For the ease of model, I have selected store-1 and department- 1.

Link of the data set: https://www.kaggle.com/iamprateek/wallmart-sales-forecast-datasets

How FB Prophet works?

At its core, the Prophet procedure is an additive regression model with four main components:

- A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data.
- A yearly seasonal component modeled using Fourier series.
- A weekly seasonal component using dummy variables.
- A user-provided list of important holidays.

The Prophet uses a decomposable time series model with three main model components: Trend, Seasonality, and Holidays. It combines and form equation as mentioned below:

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

- g(t): piecewise linear or logistic growth curve for modeling non-periodic changes in time series
- s(t): periodic changes (e.g. weekly/yearly seasonality)
- h(t): effects of holidays (user provided) with irregular schedules
- εt: error term accounts for any unusual changes not accommodated by the model

Using time as a regressor, Prophet is trying to fit several linear and non-linear functions of time as components. Prophet frames the forecasting problem as a curve-fitting exercise rather than looking explicitly at the time-based dependence of each observation within a time-series.

Validation of data is like any other time-series forecasting. Due to the temporal dependencies in time series data, we ensure that training set contains observations that occurred prior to the ones in validation sets.

To evaluate the performance of the model, I have used sMAPE (Symmetric mean absolute percentage error) criteria. sMAPE is the error evaluated between forecasted and actual values.

Let’s code for FB Prophet:

Importing the required libraries:

```
import warnings
warnings.filterwarnings("ignore")

# loading packages
# basic + dates
import numpy as np
import pandas as pd
from pandas import datetime

# data visualization
import matplotlib.pyplot as plt
import seaborn as sns # advanced vizs
%matplotlib inline

# statistics
from statsmodels.distributions.empirical_distribution import ECDF

# time series analysis
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# prophet by Facebook
from fbprophet import Prophet
```

Reading the Data set

```
train = pd.read_csv(r"/Users/y0p00uk/Documents/filter.csv")
```

```
train.head()
```

	Date	Weekly_Sales
0	2/5/10	24924.50
1	2/12/10	46039.49
2	2/19/10	41595.55
3	2/26/10	19403.54
4	3/5/10	21827.90

Now as we can see the date column is in month/date/year, the column name should be “ds” and Weekly sales should be “y”. So next step will be renaming the columns. This is one of the must step in FB Prophet forecasting.

```
data['ds']=data['Date']
data['y'] = data['Weekly_Sales']
```

```
df=data[['ds','y']]
```

```
df.head()
```

	ds	y
0	2010-02-05	24924.50
1	2010-02-12	46039.49
2	2010-02-19	41595.55
3	2010-02-26	19403.54
4	2010-03-05	21827.90

Splitting the data into train and test, so that we can train the model on training part and test the accuracy of the model on test data.

```
train = df[:100]
test = df[100:]
```

```
train.tail(10)
```

	ds	y
90	2011-10-28	31579.90
91	2011-11-04	39886.06
92	2011-11-11	18689.54
93	2011-11-18	19050.66
94	2011-11-25	20911.25
95	2011-12-02	25293.49
96	2011-12-09	33305.92
97	2011-12-16	45773.03
98	2011-12-23	46788.75
99	2011-12-30	23350.88

Now calling Prophet function and performing prediction,

```
m=Prophet()
m.fit(df)
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

```
<fbprophet.forecaster.Prophet at 0x1a34adc510>
```

```
future=m.make_future_dataframe(periods=365)
future.tail()
```

	ds
503	2013-10-22
504	2013-10-23
505	2013-10-24
506	2013-10-25
507	2013-10-26

```
forecast=m.predict(future)
forecast[['ds','yhat','yhat_lower','yhat_upper']].tail()
```

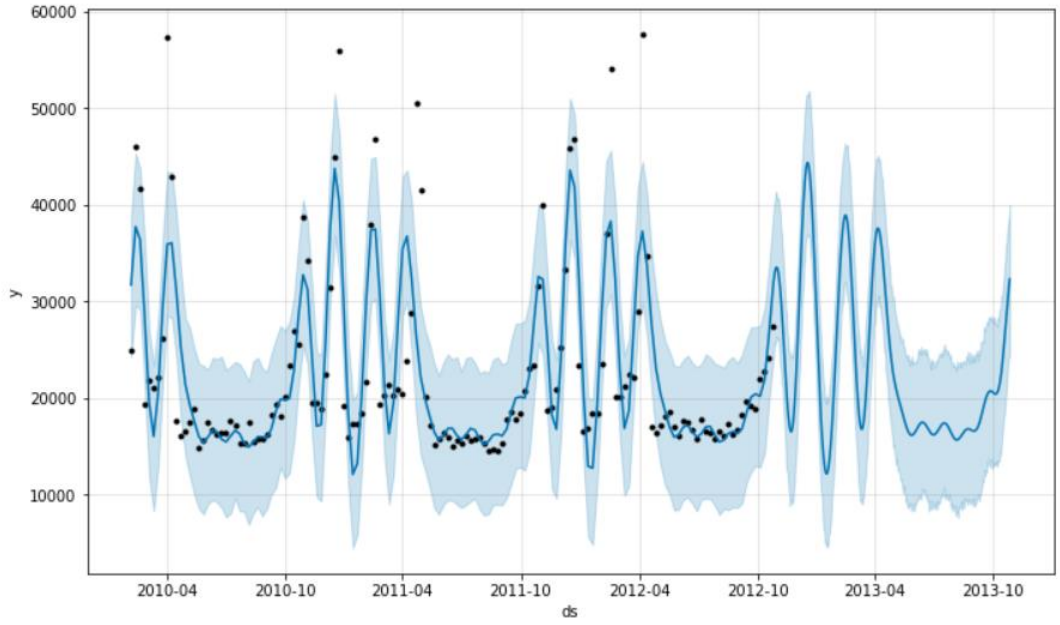
	ds	yhat	yhat_lower	yhat_upper
503	2013-10-22	29399.908907	22151.742483	37065.465225
504	2013-10-23	30209.196533	23300.482521	37694.901143
505	2013-10-24	30972.836447	23921.873316	38218.241751
506	2013-10-25	31674.363306	24173.953066	38858.649416
507	2013-10-26	32297.805279	24384.318266	39945.174887

The yhat is the predicted value. Yhat_lower and yhat_upper is the prediction uncertainty interval.

Hence, for the error calculation we have to consider yhat.

#Plotting the forecast

```
fgil=m.plot(forecast)
```



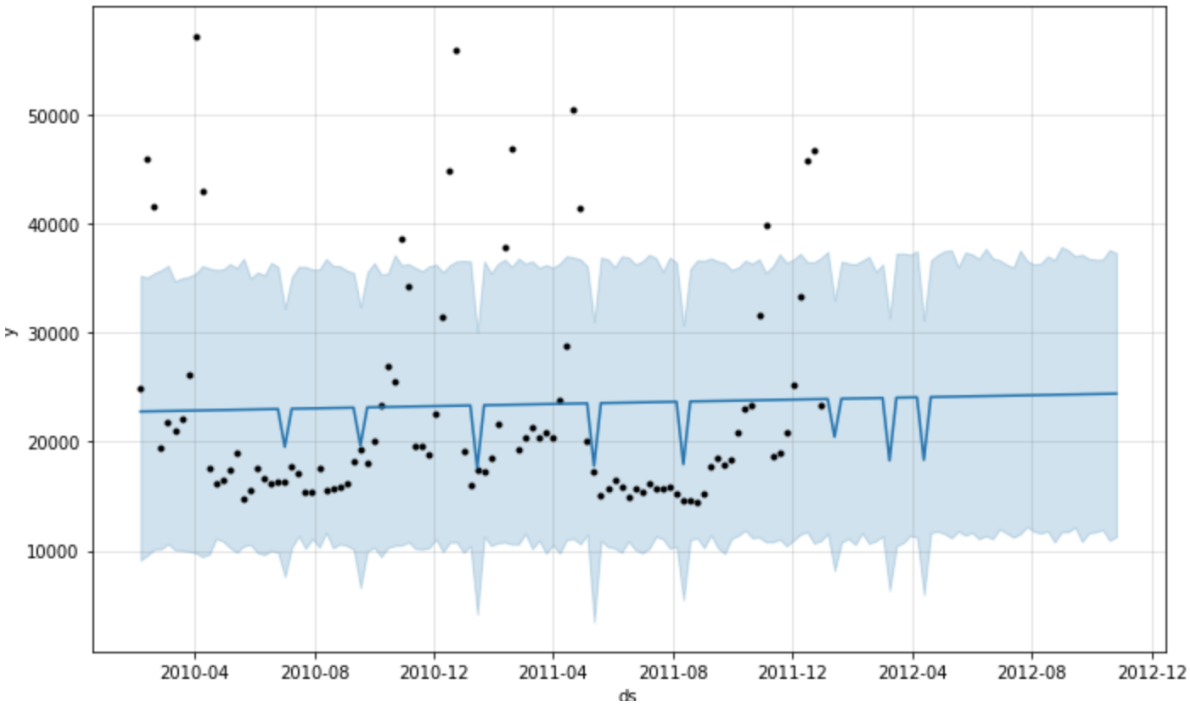
In the above graph, the blue line is the forecasted values and black dots are the actual values. We can see there are the black dot i.e the actual values are till October-2012 and as the model has forecasted for next year for which actual values are not available so, black dots are not present. From the above graph, the model seems to be captured the actual values quite well, without including holidays in the model.

Now, let's include holidays in the model and see the difference.

Next step is to introduce the holidays, we have a number of holidays in a year. That we have to provide to the model, so that it can identify the trend of holidays every year.

```
playoffs = pd.DataFrame({
  'holiday': 'playoff',
  'ds': pd.to_datetime(['2010-01-13', '2010-01-16',
                        '2010-01-24', '2010-02-07', '2010-03-08',
                        '2010-04-12', '2010-05-12', '2010-06-19',
                        '2010-07-02', '2010-08-11', '2010-09-17',
                        '2010-10-24', '2010-11-07','2011-01-13', '2011-01-16',
                        '2011-01-24', '2011-02-07', '2011-03-08',
                        '2011-04-12', '2011-05-12', '2011-06-19',
                        '2011-07-02', '2011-08-11', '2011-09-17',
                        '2011-10-24', '2011-11-07','2012-01-13', '2012-01-16',
                        '2012-01-24', '2012-02-07', '2012-03-08',
                        '2012-04-12', '2012-05-12', '2012-06-19',
                        '2012-07-02', '2012-08-11', '2012-09-17',
                        '2012-10-24', '2012-11-07',]),
  'lower_window': 0,
  'upper_window': 1,
})
superbowls = pd.DataFrame({
  'holiday': 'superbowl',
  'ds': pd.to_datetime(['2010-02-07', '2014-02-02', '2016-02-07']),
  'lower_window': 0,
  'upper_window': 1,
})
holidays = pd.concat((playoffs, superbowls))
```

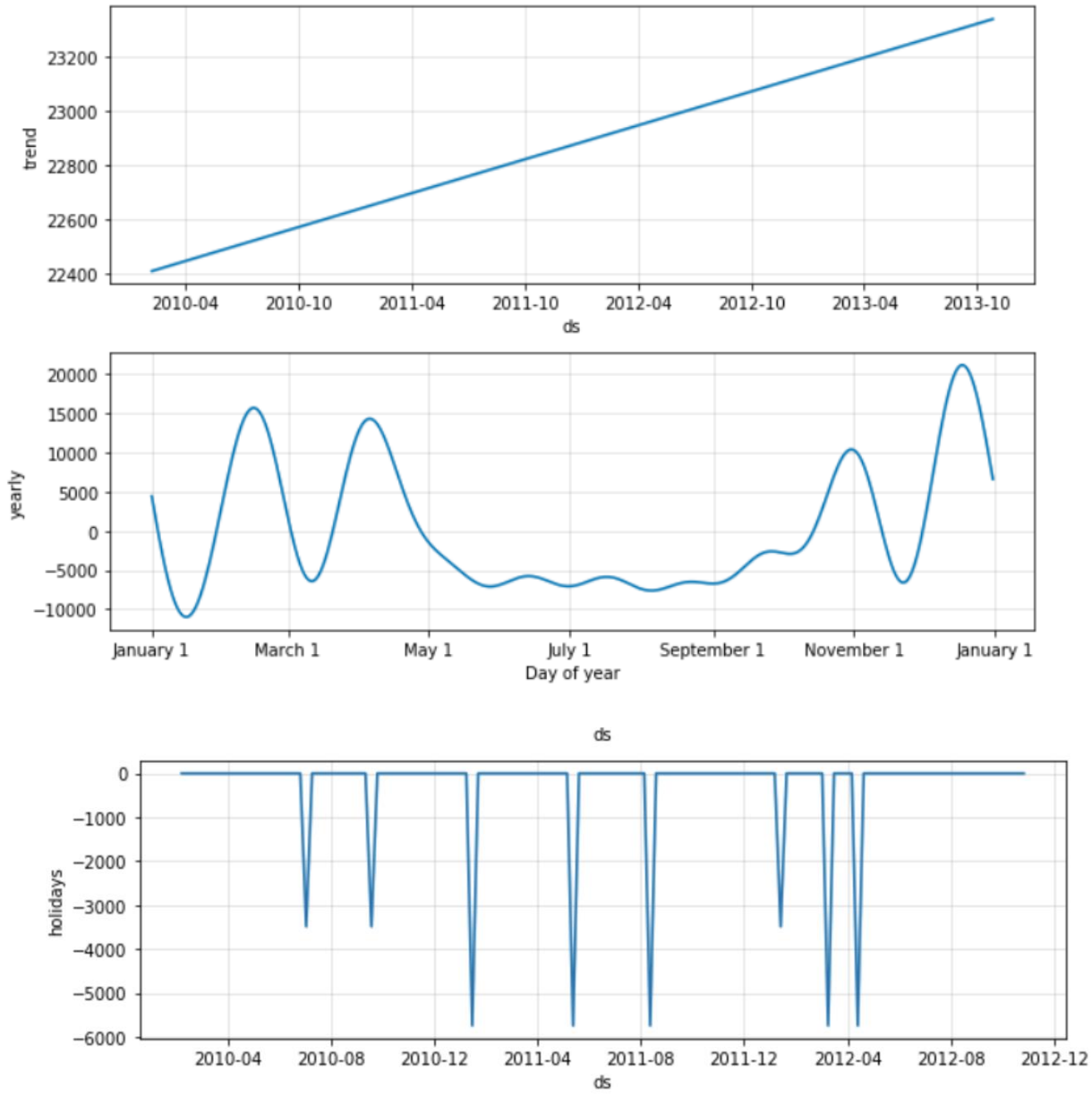
The above code lists out the dates of holidays (These are not the real holidays date of any country, I have considered some dates to make a list of holidays)



From the above graphs, we can see the model training is capturing the lows and highs in the data and based on that it is predicting for future.

Check the trend and seasonality component

```
fig2=m.plot_components(forecast)
```



The model is also capturing the holidays year by year.

Calculating sMAPE error

```
df['SMAPE_ERR']=(abs(forecast['yhat']-df['y'])/((abs(forecast['yhat']) + abs(df['y']))/2))*100
df['forecast'] = forecast['yhat']
df
```

```
df['SMAPE_ERR'].mean()
```

```
13.938222217801277
```

As, we can see that the sMAPE error mean is 13.93 %.

Now, let's go through the ARIMA code:

Importing the libraries and data for model building. Ultimate goal is to calculate the sMAPE of forecasted value.

```
import pandas as pd
from pandas import datetime
import matplotlib.pyplot as plt
import statsmodels
from statsmodels.tsa.ar_model import AR
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from math import sqrt
from time import time
import warnings
warnings.filterwarnings("ignore")
```

```
/Users/y0p00uk/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: FutureWarning: The pandas.datetim
e class is deprecated and will be removed from pandas in a future version. Import from datetime module instead.
```

```
def parser(x):
    return datetime.strptime(x, '%Y-%m-%d')

df = pd.read_csv(r"/Users/y0p00uk/Documents/filter.csv", parse_dates = [0], date_parser=parser)
df
```

	Date	Weekly_Sales
0	2010-02-05	24924.50
1	2010-02-12	46039.49
2	2010-02-19	41595.55
3	2010-02-26	19403.54
4	2010-03-05	21827.90

Next, we have to split the data into train and test sets and trained the ARIMA model on train set. Once the model is trained, we can test the accuracy of the model using test set and calculating sMAPE value. Below code is splitting the data and training the model on train data.

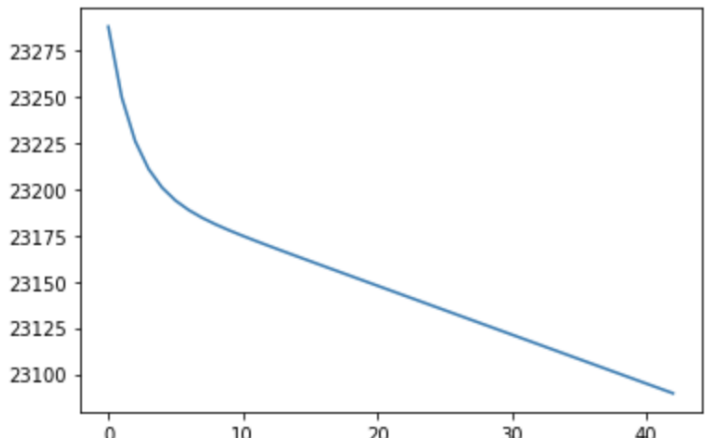
```
x = df.Weekly_Sales
x.size
```

```
143
```

```
train = x[:100]
test = x[100:]
```

```
model_arima = ARIMA(train, order = (1,1,1))
model_arima_fit = model_arima.fit()
print(model_arima_fit.aic)
prediction1 = model_arima_fit.forecast(steps = 43)[0]
print(prediction1)
plt.plot(prediction1)
```

```
2078.9242923101474
[23288.0365058  23249.7710843  23226.04828132 23210.93021137
 23200.90348244 23193.88925296 23188.65749157 23184.48040007
 23180.92734701 23177.74353172 23174.77819101 23171.9421197
 23169.18253589 23166.46820903 23163.78066028 23161.10895591
 23158.44662649 23155.78984416 23153.13634399 23150.48478583
 23147.83437675 23145.18464757 23142.53532068 23139.88623182
 23137.2372838  23134.58841911 23131.93960373 23129.29081753
 23126.64204859 23123.99328986 23121.34453718 23118.69578807
 23116.04704108 23113.39829534 23110.74955034 23108.10080578
 23105.45206148 23102.80331733 23100.15457327 23097.50582927
 23094.8570853  23092.20834135 23089.55959741]
[<matplotlib.lines.Line2D at 0x1a1a05cdd0>]
```



Calculating the sMAPE:

```
error=(abs(prediction1-test)/((abs(prediction1) + abs(test))/2))*100
error
```

```
error.mean()
```

```
27.134686937096692
```

If we will do the forecasting using ARIMA with (1,1,1) grid, we will get sMAPE as 27.13% as shown above. So, we can say

that FBProphet is performing much better than ARIMA in this case. Although, there other criteria's through which we can see the performance of the model and can make our judgements. But, for this paper, I am keeping our discussion limited to sMAPE criteria. As, we know that we can improve the model by hyper-tuning the parameters. Feel free play with parameters for better result.

Any comment for improvement in the document is highly appreciated.