Utilizing ARIMA to forecast Uber's market demand

Python notebook using data from Uber Pickups in New York City · 4,656 views

Version 13
13 commits

Notebook
Data
Log
Comments

Uber market demand prediction using seasonal ARIMA with grid seach

Market demand plays a crucial part in the marketing strategy of any company. Forecasting such demand becomes crucial when the market is filled with competition, and a small mismatch in supply and demand can lead to a customer switching to another service provider.

In this notebook we look at a classical algorithm(ARIMA) which can be used to predict the demand for user trips in the upcoming week for a particular location. Particularly, we will be utilizing Uber's 2014 user trips data of New York city, to accomplish the same.

The dataset can be found on kaggle.com(https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city (https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city))

In [1]:

```
#Importing necessary libraries
import pandas as pd
```

Notebook Data Log Comments

```
from statsmodels.tsa.statespace.sarimax import SAR
IMAX
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pac
f
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
from math import sqrt
```

```
/opt/conda/lib/python3.6/site-packages/statsmodel
s/compat/pandas.py:56: FutureWarning: The pandas.
core.datetools module is deprecated and will be r
emoved in a future version. Please use the panda
s.tseries module instead.
  from pandas.core import datetools
```

First we upload our data-set into a single data-frame:

In [2]:

```
#Preparing the uber 2014 main dataset
def prepare_2014_df():
```

```
    #Loading datasets
    uber_2014_apr=pd.read_csv('../input/uber-raw-d
ata-apr14.csv',header=0)
    uber_2014_may=pd.read_csv('../input/uber-raw-d
ata-may14.csv',header=0)
    uber_2014_jun=pd.read_csv('../input/uber-raw-d
ata-jun14.csv',header=0)
    uber_2014_jul=pd.read_csv('../input/uber-raw-d
ata-jul14.csv',header=0)
    uber_2014_aug=pd.read_csv('../input/uber-raw-d
ata-aug14.csv',header=0)
    uber_2014_sep=pd.read_csv('../input/uber-raw-d
ata-sep14.csv',header=0)


    #Merging
    df = uber_2014_apr.append([uber_2014_may,uber_
2014_jun,uber_2014_jul,uber_2014_aug,uber_2014_sep
], ignore_index=True)

    #returning merged dataframe
    return df

#Uber 2014 dataset
uber_2014_master = prepare_2014_df()
uber_2014_master.head()
```

Out[2]:

	Date/Time	Lat	Lon	Base
0	4/1/2014 0:11:00	40.7690	-73.9549	B02512
1	4/1/2014 0:17:00	40.7267	-74.0345	B02512
2	4/1/2014 0:21:00	40.7316	-73.9873	B02512
3	4/1/2014 0:28:00	40.7588	-73.9776	B02512
4	4/1/2014 0:33:00	40.7594	-73.9722	B02512

Feature Engineering

Next, we prepare the data-frame so that it is in a time-series format, which can then be utilized for modelling. Since we are only looking at a basic time-series forecast model, we will only be utilizing the Date/Time column for now.

I plan to predict the demand at a day level and hence we will be resampling the data at a day level. However deping on the need, we can sample the data at different levels(Hour,Month,Year etc.)

In [3]:

```
# Feature Engineering
def create_day_series(df):
```

```
    # Grouping by Date/Time to calculate number of trips
    day_df = pd.Series(df.groupby(['Date/Time']).size())
    # setting Date/Time as index
    day_df.index = pd.DatetimeIndex(day_df.index)
    # Resampling to daily trips
    day_df = day_df.resample('1D').apply(np.sum)

    return day_df

day_df_2014 = create_day_series(uber_2014_master)
day_df_2014.head()
```

Out[3]:

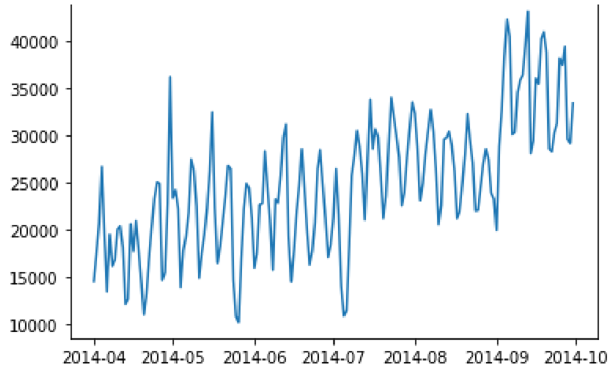
```
Date/Time
2014-04-01    14546
2014-04-02    17474
2014-04-03    20701
2014-04-04    26714
2014-04-05    19521
Freq: D, dtype: int64
```

Now that we have the time-series data-frame ready, we can look into some initial visualizations of the data to decide our parameters for the ARIMA model

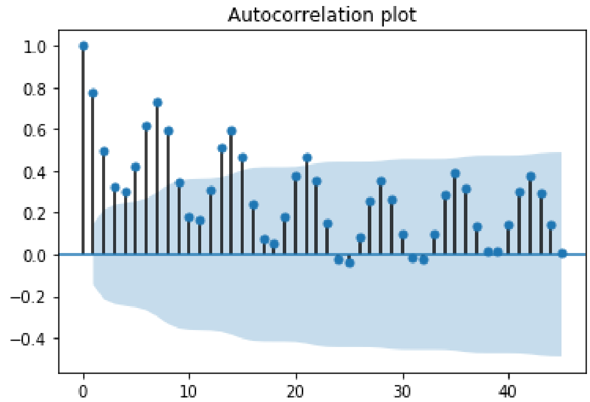
In [4]:

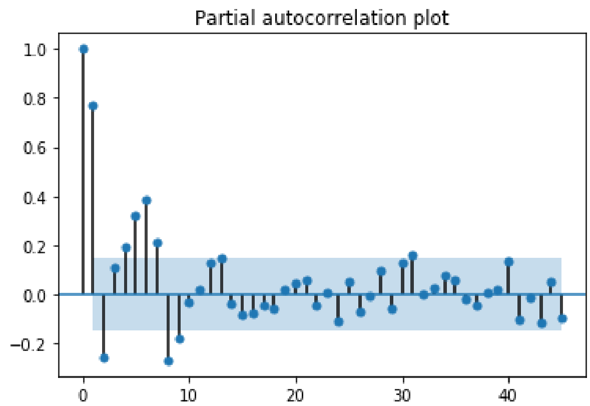
```
#Checking trend and autocorrelation
def initial_plots(time_series, num_lag):

    #Original timeseries plot
    plt.figure(1)
    plt.plot(time_series)
    plt.title('Original data across time')
    plt.figure(2)
    plot_acf(time_series, lags = num_lag)
    plt.title('Autocorrelation plot')
    plot_pacf(time_series, lags = num_lag)
    plt.title('Partial autocorrelation plot')

    plt.show()


#Augmented Dickey-Fuller test for stationarity
#checking p-value
print('p-value: {}'.format(adfuller(day_df_2014)[1]))

#plotting
initial_plots(day_df_2014, 45)
```

```
p-value: 0.8458980824898368
```



<matplotlib.figure.Figure at 0x7f1916b6fb70>





Looking at the ADF test we see that clearly the time-series is not stationary(p-value>0.05 i.e for a confidence level of 95%), hence differencing is required.

Before we even analyse the ACF and PACF plots we need to difference and test for stationarity

In [5]:

```
#storing differenced series
diff_series = day_df_2014.diff(periods=1)

#Augmented Dickey-Fuller test for stationarity
#checking p-value
print('p-value: {}'.format(adfuller(diff_series.dr
opna())[1]))
```

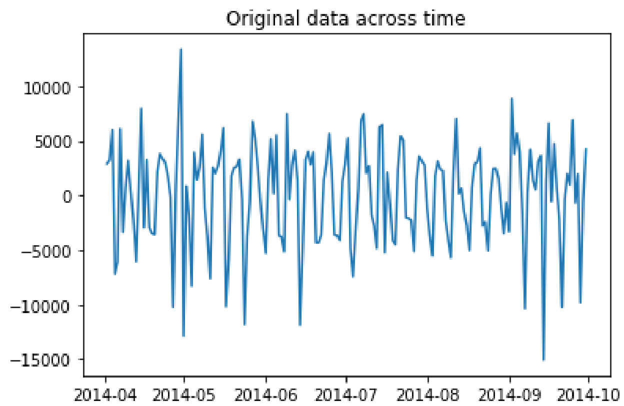
```
p-value: 1.5163641177435235e-08
```

Looks like the series is stationary now (as p-value < 0.05, we can assume stationarity with a confidence level of 95%, even higher actually). So a differencing of 1 should be perfect!

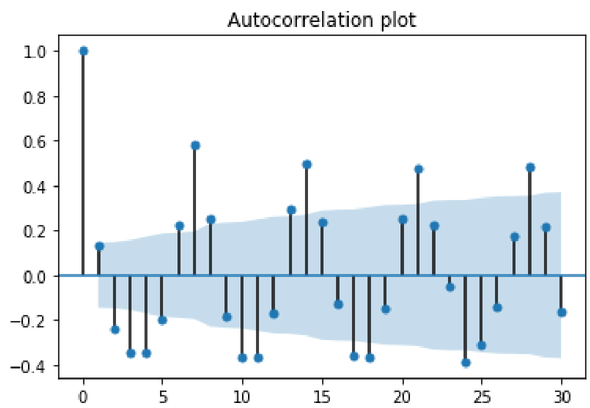
Now lets look at the ACF and PACF plots:

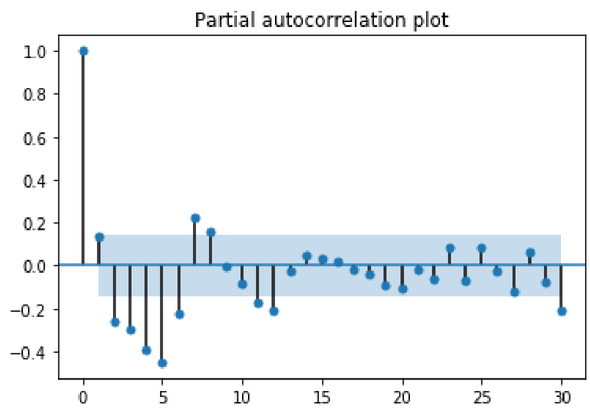
In [6]:

```
initial_plots(diff_series.dropna(), 30)
```




```
<matplotlib.figure.Figure at 0x7f19169816d8>
```





From the ACF and PACF plots we can see a clear spike at every 7 day interval. And since this appears clearly in the ACF plot this whows a seasonal MA component of 1.

Fitting SARIMAX models

Although at this point the components can be guessed at ARIMA(0,1,0)(0,0,1)[7], we will implement a grid search to find the best fitting values, using RMSE as the deciding factor:

In [7]:

```
#Defining RMSE
def rmse(x,y):
    return sqrt(mean_squared_error(x,y))

#fitting ARIMA model on dataset
def SARIMAX_call(time_series,p_list,d_list,q_list,
P_list,D_list,Q_list,s_list,test_period):

    #Splitting into training and testing
    training_ts = time_series[:-test_period]

    testing_ts = time_series[len(time_series)-test
_period:]

    error_table = pd.DataFrame(columns = ['p','d',
'q','P','D','Q','s','AIC','BIC','RMSE'],\

index = range(len(ns_ar)*len(ns_diff)*len(ns_ma)*l
en(s_ar)\

*len(s_diff)*len(s_ma)*len(s_list)))
    count = 0

    for p in p_list:
        for d in d_list:
            for q in q_list:
                for P in P_list:
                    for D in D_list:
                        for Q in Q_list:
                            for s in s_list:
                                #fitting the model
                                SARIMAX_model = SA
RIMAX(training_ts.astype(float),\

order=(p,d,q),\

seasonal_order=(P,D,Q,s),\

enforce_invertibility=False)
                                SARIMAX_model_fit
= SARIMAX_model.fit(disp=0)
                                AIC = np.round(SAR
IMAX_model_fit.aic,2)
```

```
                                BIC = np.round(SAR
IMAX_model_fit.bic,2)
                                predictions = SARI
MAX_model_fit.forecast(steps=test_period,typ='leve
ls')
                                RMSE = rmse(testin
g_ts.values,predictions.values)

                                #populating error
 table
                                error_table['p'][c
ount] = p
                                error_table['d'][c
ount] = d
                                error_table['q'][c
ount] = q
                                error_table['P'][c
ount] = P
                                error_table['D'][c
ount] = D
                                error_table['Q'][c
ount] = Q
                                error_table['s'][c
ount] = s
                                error_table['AIC']
[count] = AIC
                                error_table['BIC']
[count] = BIC
                                error_table['RMSE'
][count] = RMSE

                                count+=1 #incremen
ting count

    #returning the fitted model and values
    return error_table

ns_ar = [0,1,2]
ns_diff = [1]
ns_ma = [0,1,2]
s_ar = [0,1]
s_diff = [0,1]
s_ma = [1,2]
s_list = [7]

error_table = SARIMAX_call(day_df_2014,ns_ar,ns_di
ff,ns_ma,s_ar,s_diff,s_ma,s_list,30)
```

```
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
```

```
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/opt/conda/lib/python3.6/site-packages/statsmodel
s/base/model.py:496: ConvergenceWarning: Maximum
```

```
Likelihood optimization failed to converge. Check
mle_retvals
  "Check mle_retvals", ConvergenceWarning)
```

Now that we have obtained the RMSE values for different combinations, we can take a look at the lowest 5 RMSE values:

In [8]:

```
# printing top 5 lowest RMSE from error table
error_table.sort_values(by='RMSE').head(5)
```

Out[8]:

	p	d	q	P	D	Q	s	AIC	BIC	RMSE
5	0	1	0	1	0	2	7	2903.28	2915.4	5045.49
3	0	1	0	0	1	2	7	2752.78	2761.87	5105.3
7	0	1	0	1	1	2	7	2754.8	2766.92	5136.57
6	0	1	0	1	1	1	7	2754.5	2763.6	5278.11
2	0	1	0	0	1	1	7	2752.71	2758.77	5336.34

We see that ARIMA(0,1,0)(1,0,2)[7] gives the lowest RMSE. However, next best RMSE is of ARIMA(0,1,0)(0,1,2)[7] has lower AIC and BIC scores, which tells us that it fits the data much better than the lowest RMSE ARIMA fit.

Here we can chose the second ARIMA if we want a more robust solution, which can generalize well to other situations as well, however this ensures that we do not get the best RMSE for this testing data.

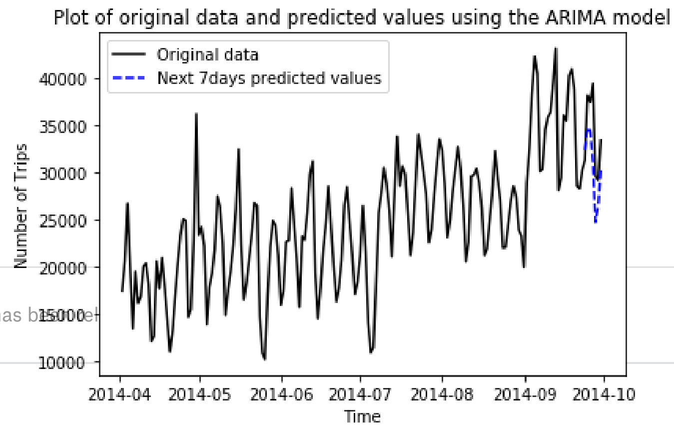
Forecasting

I will forecast using ARIMA(0,1,0)(0,1,2)[7] as I want the model to fit the data better along with giving me a lower RMSE:

In [9]:

```
#Predicting values using the fitted model
def predict(time_series,p,d,q,P,D,Q,s,n_days,conf
):

    #Splitting into training and testing
    training_ts = time_series[:-n_days]

    testing_ts = time_series[len(time_series)-n_da
ys:]

    #fitting the model
    SARIMAX_model = SARIMAX(training_ts.astype(flo
at),\
                            order=(p,d,q),\
                            seasonal_order=(P,D,Q,
```

```
s),\
                                            enforce_invertibility=
False)
    SARIMAX_model_fit = SARIMAX_model.fit(disp=0)

    #Predicting
    SARIMAX_prediction = pd.DataFrame(SARIMAX_mode
l_fit.forecast(steps=n_days,alpha=(1-conf)).values
,\
                                      columns=['Prediction'])
    SARIMAX_prediction.index = pd.date_range(train
ing_ts.index.max()+1,periods=n_days)

    #Plotting
    plt.figure(4)
    plt.title('Plot of original data and predicted
values using the ARIMA model')
    plt.xlabel('Time')
    plt.ylabel('Number of Trips')
    plt.plot(time_series[1:],'k-', label='Original
data')
    plt.plot(SARIMAX_prediction,'b--', label='Next
{}days predicted values'.format(n_days))
    plt.legend()
    plt.show()

    #Returning predicitons
    return SARIMAX_prediction

#Predicting the values and builing an 80% confidenc
e interval
prediction = predict(day_df_2014,0,1,0,0,1,2,7,7,
0.80)
```



This kernel has b

Thus using this simple classical Seasonal ARIMA model we can utilize the forecasted data to plan and alter our marketing strategy for the upcoming

Data

Data Sources

Uber Pickups i...

Uber Pickups in New York City

- 6 columns
- other-Carme...
- other-Dial7_...
- other-Diplo_...
- 7 columns
- other-FHV-s...
- other-Firstcl...
- other-Highcl...
- other-Lyft_B...
- other-Prestig...
- 9 more

Trip data for over 20 million Uber (and other for-hire vehicle) trips in NYC
Last Updated: 2 years ago (Version 2)

About this Dataset

Uber TLC FOIL Response

This directory contains data on over 4.5 million Uber pickups in New York City from April to September 2014, and 14.3 million more Uber pickups from January to June 2015. Trip-level data on 10 other for-hire vehicle (FHV) companies, as well as aggregated data for 329 FHV companies, is also included. All the files are as they were received on August 3, Sept. 15 and Sept. 22, 2015.

FiveThirtyEight obtained the data from the NYC Taxi & Limousine Commission (TLC) by submitting a Freedom of Information Law request on July 20, 2015. The TLC has sent us the data in batches as it continues to review trip data Uber and other HFV companies have submitted to it. The TLC's correspondence with FiveThirtyEight is included in the files `TLC_letter.pdf`, `TLC_letter2.pdf` and `TLC_letter3.pdf`. TLC records requests can be made here.

This data was used for four FiveThirtyEight stories: Uber Is Serving New York's Outer Boroughs More Than Taxis Are, Public Transit Should Be Uber's New Best Friend, Uber Is Taking Millions Of Manhattan Rides Away From Taxis, and Is Uber Making NYC Rush-Hour Traffic Worse?.

Run Info

Succeeded	True	Run Time	97.3 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python (Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Log

```
Time    Line #  Log Message
4.3s         1  [NbConvertApp] Converting notebook script.ipynb to
                html
4.4s         2  [NbConvertApp] Executing notebook with kernel:
                python3
96.7s        3  [NbConvertApp] Support files will be in
                __results___files/
                [NbConvertApp] Making directory __results___files
96.7s        4  [NbConvertApp] Making directory __results___files
                [NbConvertApp] Making directory __results___files
                [NbConvertApp] Making directory __results___files
                [NbConvertApp] Making directory __results___files
                [NbConvertApp] Making directory __results___files
                [NbConvertApp] Making directory __results___files
```

```
              [NbConvertApp] Writing 298076 bytes to
              __results__.html
96.7s      5
96.7s      7  Complete. Exited with code 0.
```

Comments (3)

Sort by
All Comments
Hotness

Click here to enter a comment...

Long Chen • Posted on Latest Version • 9 months ago • Options • Reply 0

did you try LSTM on this data? what is the performance? My LSTM with a keras implementation performs quite terrible on this dataset.

AnkitAgar... • Posted on Latest Version • 3 months ago • Options • Reply 0

For a similar problem to forecast monthly sales I am running your code to train the model and getting following error?
Line of code - "error*table = SARIMAX*call(day*df*2014,ns*ar,ns*diff,ns*ma,*sar,*sdiff,*sma,s_list,30)"
Error - "IndexError: index 0 is out of bounds for axis 0 with size 0"
Can you please help where am I making the mistake?

Swathi • Posted on Latest Version • 2 months ago • Options • Reply 0

"We can split the data-set based on location and obtain forecasts for different pick-up zones in NYC" - could you please elaborate on this point? How to prepare the data set with locations and what models to use for forecasting. How do we understand which area will have the highest demand in the near future?

Our Team Terms Privacy Contact/Support