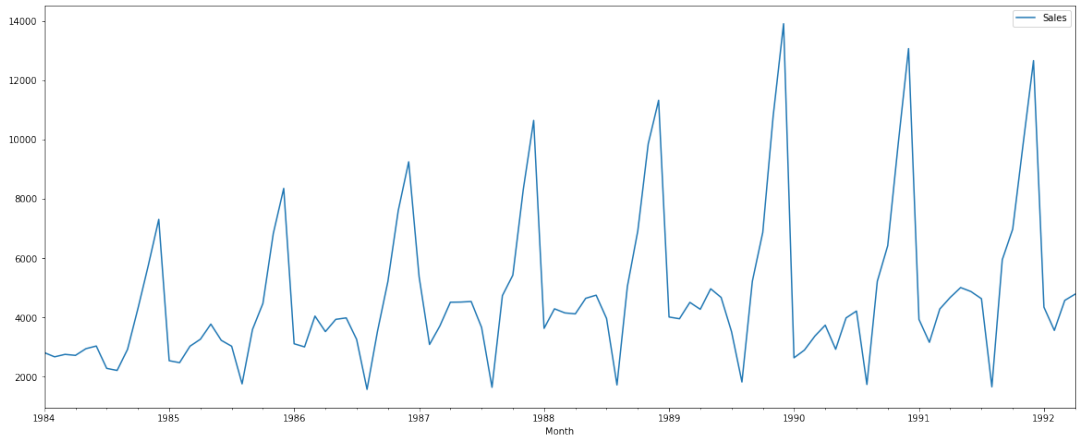
CASE STUDY

Introduction:

In this case study, it is given a timeseries dataset consisting of monthly sales. It is composed of 8 years and 9 months sales information. Then it is expected to accurately estimate 9-month sales by using at least two different machine learning models.



Models:

In this case, 3 different time series models are used. Notebook is created and executed on Google-Colabs.
Acording to Dickey-Fuller test, our data is not stationary. So I need to use different models and I need to compare their results. In cases where the data is not stationary, it is a very useful method to take the difference of the data and use it in the model.

1) XGBRegressor: In this model, I have to convert time series data to supervised scaled one. By the help of this method, data is evaluate to windowing. After this step, problem converge to classical regression problem.

   In hyper parameter tunning, I used grid search from Sklearn. I use parameters which is listed below.

```
parameters = { 'gamma' : [0, 0.1, 0.3, 1], 'learning_rate' : [0.001, 0.01, 0.1],
               'max_depth' : [2, 4, 6, 7, 12],
               'n_estimators' : [10, 45, 90, 100, 150, 250],
               'nthread' : [-1], 'reg_alpha' : [1], 'reg_lambda' : [1], 'seed' : [10] }
```

Acording to our data, best parameters are

```
xgb_grid.best_estimator_
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=2, min_child_weight=1, missing=None, n_estimators=90,
             n_jobs=1, nthread=-1, objective='reg:linear', random_state=0,
             reg_alpha=1, reg_lambda=1, scale_pos_weight=1, seed=10,
             silent=None, subsample=1, verbosity=1)
```

After model creation with these parameters, results are not so accurate. In my opinion, our data is not big enough. If we had 100 thousand rows of data, our model would produce more accurate results. Moreover since a brute force method is applied, the process time will also be high. After converting to supervised model, we have just 83 lines of row. Because of this reason, XGBRegressor is the worst one with 29 percent MAPE error. The model is created in 150 seconds.

```
print("Test RMSE:", np.mean(rmse))
print("Test MAPE:", np.mean(mape))
```

```
Test RMSE: 1352.360301915142
Test MAPE: 29.424130526377652
```

2) Arima: Arima has a version which determine most accurate parameters. It is called auto-arima. Especially “p”, “q” and “d” values are very import in auto regression. I determine the ranges to this parameters.

```
arima_stepwise_model = auto_arima(train, start_p=0, start_q=0,
                          max_p=13, max_q=13, m=12,
                          start_P=0, seasonal=True,
                          d=1, D=1, trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True,
                          n_jobs=4)#n_jobs for parallel process
```

Between these three models, arima is the most accurate one with 8 percent MAPE error. The other advantage of this method, it takes less time. Our model needs 30 seconds.

```
print("Test RMSE:", np.mean(arima_rmse))
print("Test MAPE:", np.mean(arima_mape))
```

```
Test RMSE: 431.36506422601934
Test MAPE: 8.867407129635856
```

3) Long Short-Term Memory: Last method is the deep learning method. In this model, I use differences because data is not stationary. Firstly I scale data acording to one step

differences. Models trains on this difference data. After prediction, I need to inverse scale process.

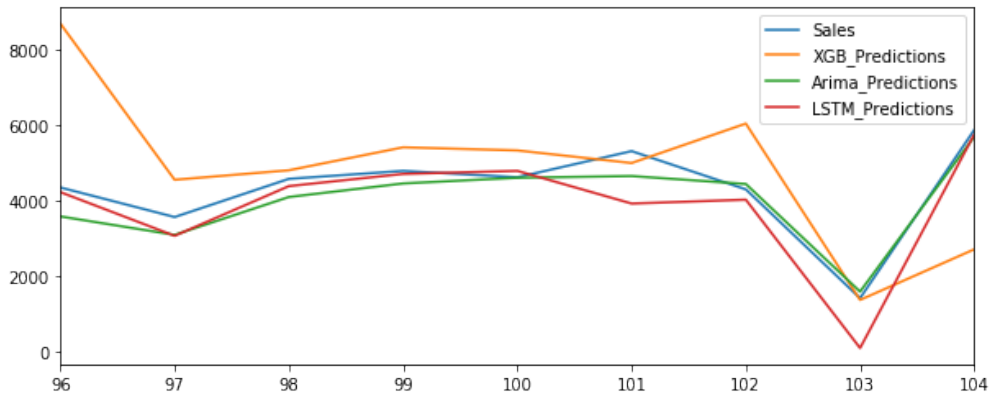
```
In [0]: # define parameters
verbose, epochs, batch_size = 1, 50, 10
#n_timesteps, n_features, n_outputs = train_X_lstm.shape[1], train_X_lstm.shape[2], test_X_lstm.sh
ape[1]
n_timesteps, n_features, n_outputs = train_X_lstm.shape[1], train_X_lstm.shape[2], train_Y_lstm.sh
ape[1]

In [371]: n_timesteps, n_features, n_outputs

Out[371]: (1, 12, 1)

In [0]:

In [0]: # define model
lstm_model = Sequential()
lstm_model.add(Conv1D(filters=32, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=(n_timesteps, n_features)))

In [0]: lstm_model.add(LSTM(200, activation='relu', return_sequences=True))
# lstm_model.add(RepeatVector(n_outputs))
lstm_model.add(LSTM(200, activation='relu', return_sequences=True))
lstm_model.add(TimeDistributed(Dense(1, activation='relu')))  # 100
#lstm_model.add(TimeDistributed(Dense(1)))
#lstm_model.compile(loss='mse', optimizer=Adam(lr=0.001))
lstm_model.compile(loss='mse', optimizer=Adam(lr=0.001),
                   metrics=['acc'])
```

After 50 epoches, I reach 17 percent MAPE error accuracy.

```
print("LSTM RMSE:", np.mean(lstm_rmse))
print("LSTM MAPE:", np.mean(lstm_mape))
```

```
LSTM RMSE: 463.2440815932228
LSTM MAPE: 17.16972564789466
```

Finally, this below plot appeares :



Reported by

Eser İnan Arslan