PARV BHARGAVA [19BAI10116]

- I have worked on Validation of the Forecasts and got to the following observation
  We could check if a better representation of our true predictive power can be obtained using dynamic forecasts. In this case, we only use information from the time series up to a certain point, and after that, forecasts are generated using values from previous forecasted time points.
  In the code chunk below, we specify to start computing the dynamic forecasts and confidence intervals from May 2017 onwards.

```
In [25]: pred_dynamic = results.get_prediction(start=pd.to_datetime('2017-05-19'), dynamic=True, full_results=True)
         pred_dynamic_ci = pred_dynamic.conf_int()
```

Figure: Make the Forecasts

Once again, we plot the real and forecasted values of the average daily temperature to assess how well we did:

```
In [26]: ax = one_step_df.T_mu_actual['2015':].plot(label='observed', figsize=(20, 15))
         pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

         ax.fill_between(pred_dynamic_ci.index,
                         pred_dynamic_ci.iloc[:, 0],
                         pred_dynamic_ci.iloc[:, 1], color='k', alpha=.25)


         ax.set_xlabel('Date')
         ax.set_ylabel('Temperature (in Celsius)')
         plt.ylim([-20,30])
         plt.legend()
         plt.show()
```

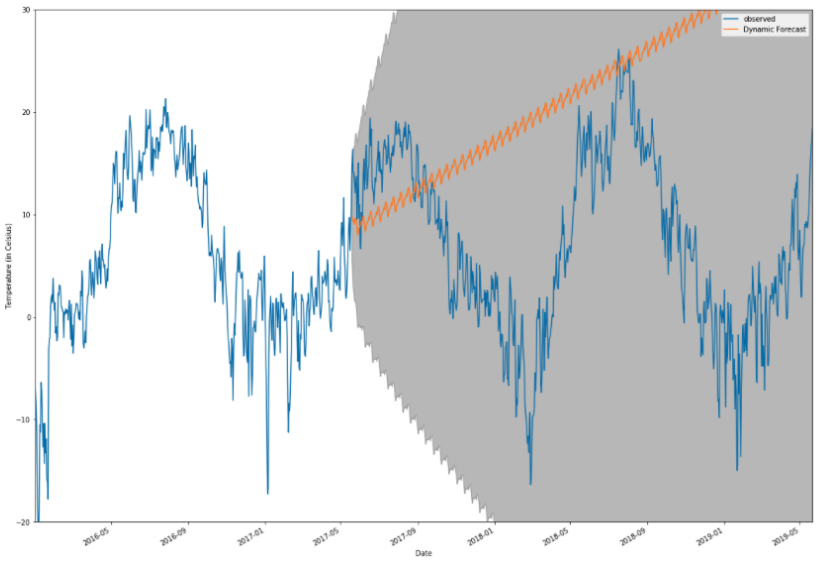

Figure: Observed vs. Forecast

Once again, we quantify the predictive performance of our forecasts by computing the RMSE:

```
In [21]: # Extract the predicted and true values of our time series
         y_forecasted = pred_dynamic.predicted_mean
         y_truth = one_step_df.T_mu_actual['2017-05-19':]

         # Compute the mean square error
         mse = sqrt(MSE(y_truth, y_forecasted).mean())
         print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))

         The Root Mean Squared Error of our forecasts is 20.04
```

Figure: The RMSE value

The predicted values obtained from the dynamic forecasts yield an RMSE of 20.04. This is significantly higher than the one-step ahead, which is to be expected given that we are relying on less historical data from the time series.

- I also have contributed to core code for JAX implantation of new model. JAX is an automatic differentiation (AD) toolbox developed by a group of people at Google Brain and the open source community. It aims to bring differentiable programming in NumPy-style onto TPUs. On the highest level JAX combines the previous projects XLA & Autograd to accelerate linear algebra-based projects.
- In our Model we decreased the training time from 30 mins to 3 mins. That's a great improvement in training the model.
- The implementation of our Time Series and related materials can be found here at my GitHub Repository
- I have previously worked with JAX which you can find here, and I found it to be the future of machine learning as an API over Numpy as JAX provide DeviceArrays which are immutable over the traditional NDArrays provided by Numpy which make the calculation here much faster as compared to Numpy and hence can be used to accelerate our model performance significantly as our project here require continuous crunching of numbers for the results.
- I have also worked hard on preparing the overall presentation for our project and on the collection of content and writeups for the presentation
- I have also helped on creating this report and worked on content and structuring of this report.
- Overall it was a group effort and I am happy that each and every one of us contributed to the project as a team.