To: Professor Allee
From: Tyler Angell
Subject: Time-Series Forecasting using LSTM Networks and Prophet
Date: April 30, 2018

Time-series forecasting is applicable to all events tracked over time, whether it is predicting the stock market, determining when someone is likely to have a health condition, or census data tracking population growth. This time-series study will forecast monthly sales of a Russian software firm, 1C Company, using two time-series forecasting algorithms in Python: Long Short-Term Memory (LSTM) and Facebook's Prophet.

The importance of this study is not only to predict future sales for the Russian software firm but to demonstrate the forecasting abilities of the algorithms. This study also presents a unique approach to forecasting using LSTMs. Additionally, this research has provided significant learning for the researcher and which will hopefully be imparted to the interested reader.

Previous work with LSTM and Prophet forecasting models has been done by Jason Brownlee, Ph.D., and William Koehrsen respectively. Regarding LSTM networks, Jason Brownlee has developed decent tutorials on LSTM networks, and this study has built upon his tutorial [1]. Regarding Prophet, William Koehrsen has also published a tutorial for which this study has similarly built upon [2].

The first step for both of these algorithms is to prepare the data. The data was obtained from Kaggle's "Predict Future Sales" challenge. Although data preparation is arguably the most time-consuming task, for the brevity of this study, more focus will be put to the actual algorithms and how to control them. There were a number of categories given with the sales data, but only the sales and date information were used for this study. The sales data was given as sales per shop with a daily frequency which was quite noisy, so the data was resampled to a monthly frequency summing total sales for all the shops.

LSTMs are a variant of Recurrent Neural Networks (RNNs) such that they overcome the long-term learning deficiency of RNNs. They essentially have "memory" since the neurons retain features of previous blocks in addition to their own selected feature. For the LSTM model, the data must be further prepared to create a supervised learning problem. The trend is removed using the standardized method of taking the difference of current value minus the previous value: $x_t - x_{t-1}$. The input data (difference values) is then duplicated and shifted such that the difference values of the next month (t+1) is the output (target to be predicted). The data is then divided into training and test and then run as a typical supervised learning problem. The LSTM knobs for building the model are the batch size (1), number of epochs (3000), number of neurons (4), the loss function (mean squared error) and the optimizing algorithm (stochastic gradient descent). Batch size is usually a control, but must be a factor of the size of the training and test data. The number of epochs determines the number of times the weights are updated. The number of neurons determines the number of memory blocks. The optimizing algorithm is what

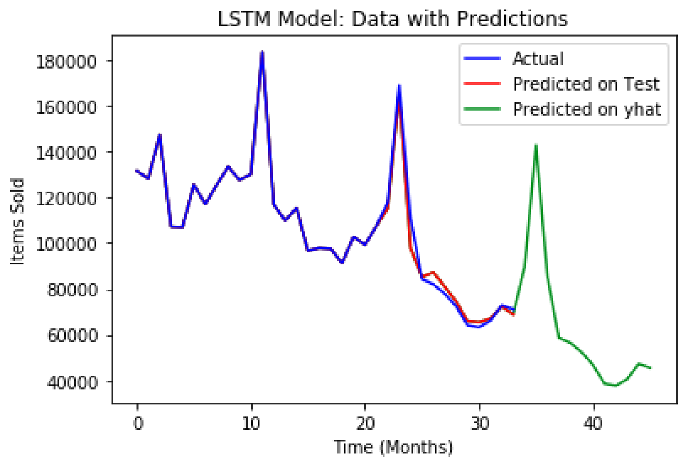

Figure 1: LSTM Result

operates on the loss function to find an optimal solution. The result for the LSTM model is shown in Figure 1.

The blue line in the figure shows the actual sales values (months 0 to 33). The red line shows the predicted values using walk-forward validation on the test data (months 22 to 33). The green line shows the predicted value using the previously predicted value. For walk-forward validation, the model takes the actual value, x_t, and only predicts one step ahead, $\hat{y}_{t+1}$. The predicted value ($\hat{y}_{t+1}$) and the actual value (x_{t+1}) can be compared to determine the error in the predicted value. The calculated root-mean-square-error of predicted to actual values was about 4,628 samples, and this value makes sense since the predicted values for the test data (red line) were fairly close to the actual values; this was the primary determinant of the model's forecasting ability. The novelty of this research is in green line which predicts one step at a time, but instead of using the actual value, x_t, to predict the next value ($\hat{y}_{t+1}$), it uses the predicted value, $\hat{y}$. The model appears to do a decent job capturing the characteristics of the data in the forecasted green line: seasonality, the sharp peak, decreasing trend.

Prophet resembles a generalized additive model, a class of regression models for which the linear predictor depends linearly on smoothed functions of the regressors (predictor variables) [3]. The algorithm automatically detects trends in data by selecting changepoints which are datapoints where there are sharp changes in the slope of the data. The key tuning parameter for Prophet is the changepoint_prior_scale which modulates automatic changepoint selection flexibility: large values allow more and small values allow fewer changepoints [4]. For this data, it was found that a small changepoint_prior_scale value (0.02) was sufficient to capture the characteristics of the data without overfitting. Prophet can also incorporate seasonality which are variations in data occurring at regular intervals; this can be observed by simply plotting the data which would show a yearly seasonality for this dataset. The result for Prophet is shown in Figure 2.

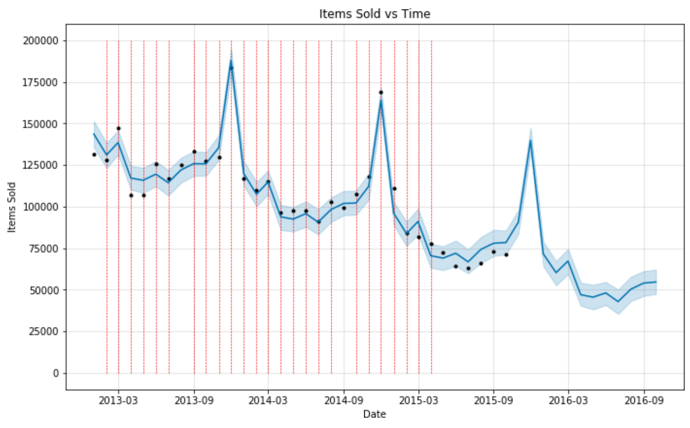

Figure 2: Prophet Result

The red lines in the figure show the locations where Prophet has selected changepoints within the first 80% of the data by default. The black dots are the actual data points. The solid blue line is the model that Prophet has fit to the data, and the shaded blue is the uncertainty in the model. The model appears to do a decent job capturing the characteristics of the data: seasonality, sharp peaks, decreasing trend. It is worth noticing that the margin of uncertainty is tight but not hugging the model which is a sign of overfitting. Another point worth mentioning is typically the margin of uncertainty fans out the further in time the model predicts. This does not appear to be the case which is an indicator of the model's confidence in the prediction.

Comparing LSTMs to Prophet, Prophet is the more elegant approach. Prophet is also more user-friendly by hiding the complexities below the surface. LSTMs are more complex but offer more customization. For consistency of results, Prophet generates the same plot every run, whereas the LSTM prediction has a decent amount of variance. In regards to runtime, Prophet runs quite a bit faster for this study, but the LSTM model also runs 3000 epochs. Some limitations to these models are that events must have happened in the past in order to be trained into the network. Thus, these models would not be able to account for events not trained into the network.

References:

[1] Brownlee, J. (2017). Time Series Forecasting with the Long Short-Term Memory Network in Python. [Blog] *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/ [Accessed 29 Apr. 2018].

[2] William, K. (2018). Time Series Analysis in Python: An Introduction. [Blog] *Towards Data Science*. Available at: https://towardsdatascience.com/time-series-analysis-in-python-an-introduction-70d5a5b1d52a [Accessed 29 Apr. 2018].

[3] Taylor, S. and Letham, B. (2018). Forecasting at Scale. *The American Statistician*, [online] 72(1), pp.37-45. Available at: https://amstat.tandfonline.com/doi/citedby/10.1080/00031305.2017.1380080?scroll=top&needAccess=true#.WuU0EIjwaUk [Accessed 29 Apr. 2018].

[4] GitHub. (2018). *facebook/prophet*. [online] Available at: https://github.com/facebook/prophet/blob/master/python/fbprophet/forecaster.py [Accessed 29 Apr. 2018].