URBINT Coding Challenge

Forecasting NYC Subway Daily Usage
from
2016 MTA Turnstile data

Julien Baur
May 22, 2019

<u>Instructions</u>

1/ Using January to November as a training set, we wish to step through every day in December, and using all data prior to that date, forecast the total number of entries on that day. Ensure that you do whatever data cleaning and filtering is necessary. What is the most appropriate metric for evaluating success?

2/ Similar to 1, forecast turnstile entries for every station individually for every day in December. Is this problem easier or harder than forecasting for the system as a whole?

3/ Augment your solution to 2. With whatever external data (one or two sources) you think is most appropriate. Explain why you think this data would be useful in predicting ridership and the improvement in predictive accuracy

ANSWERS

1/ The last section of this report, and specifically the discussion around figure 6 describes the day-by-day forecast for the month of December. A good metric to judge whether a model is good is the Student test statistic, also known as the Chi2 (per degree of freedom). Each model parameter, in addition to all observations are being considered in this test statistic.

2/ The script I wrote for running the time series feed-forward forecasting is applicable to any one of the Ns stations' daily entries. I wrote the 'parallel.py' script to launch several instances on different processors to speed up this step as fitting the model can be quite lengthy. Although a handful of individual time series have similar trends and seasonalities, a lot exhibit trends that are far from conformal in relation to the station-average, especially those that witness relatively low usage overall. As a result, if one were to train on the station-averaged time series (from question 1), I anticipate the forecasts to be poor on a station-to-station basis. If one were to train on each individual time series independently (which is what my script intends to do), it would be both computationally expensive, and the question of non-stationarity would arise for the same aforementioned outliers. However, the trade-off of having a model per station would be very beneficial. Cross-validation between the average of predictions for a day and the prediction for a day on average would have to be looked at as well.

3/ Intuitively, the weather should have an impact on ridership, and one could incorporate a multi-variate time series forecasting, for instance with NYC daily weather data from 2016 (daily precipitation levels over time), but this approach does not sound insightful or original. Instead, I would be interested in investigating urban mobility subjects. For instance, quantify the amount of ride-sharing vehicles (like Lyft, Uber, Waze Carpool etc) in a geographic zone in a given time increment. These platforms' data is not only time-stamped but also geo-localised, which make it ideal for predicting an individual station's subway ride share. I would start with only one of these data sets, assuming it is a good (biased) proxy for the total ride-sharing. Defining a radius around a station and quantifying the flow of ride-sharers through this zone would not be adequate since most would be simply passing through the zone. Instead, I would use pickup and destination addresses as the markers. The idea would be to count the 'in' and 'out' points around a station. This data would be very resolute in time, and may impact the ridership share on time scales below the 4-hour nominal interval of the MTA turnstile data. The best we could do is bin it down to the 4 hour time scale.

Procedure

In anticipation of question 2, I decided to work with Ns time series, where Ns is the number of individual stations. Each time series is the daily entries registered by all turnstiles of each unit at that particular station. With a time bin size of one calendar day, the data set I want to work with is a Ns x Nd data frame, with Nd the number of days in the entire data set (a bit short of 365). To obtain the total daily usage on the network, it will thus be sufficient to sum over all the Ns time series to answer question 1.

Each time series (TS) will then be treated as a vector where the consecutive daily entries are features, and the total entries on the following day will be treated as a label. I will use an algorithm to forecast the day following a given vector of consecutive daily entries. Once fitting and optimizing is complete on a training/testing sample, I will use the full set (pre-december) to forecast on a validation set (december), one day at a time.

Choice of Algorithms

Predicting the future based on past trends hints at pattern recognition. In the category of recurrent neural networks, long short-term memory neural networks (LSTM) are fit for this type of walk-forward time series prediction. Specifically, I am going for sequence-to-sequence encoder-decoder architecture, with each one being an LSTM. There are undoubtedly many other alternatives, but at this point, LSTM encoder – decoder presents several advantages:

- the time series can be un-stationary,
- it is generalization to multi-variate time series, such as for instance if we were to predict the total daily usage from each station’s time series individually.

As such, I undertook the challenge with in mind training / testing, optimizing, and forecasting / validation with this type of RNN. Special attention had to be put into having the correct vector shape for feeding into and out of the network. However, I encountered a major set-back on this particular step. It is computationally expensive to train and optimize such a network given a set of hyper-parameters, and in many cases I would run out of RAM while working on this step. Exploring a parameter space for batch size, epochs and units to use was made difficult, even with a very small and rudimentary grid of parameters. I have managed to explore some of the parameter space, and was confident I was training more accurate networks. However, interactively evaluating a model to lower the stochastic-ity was impractical on my laptop. To not completely have days of work go to waste, I decided to go with a model that is relatively well fitted for the limited set of hyper-parameters I explored, and decided to forecast with this one, knowing there are many ways it could be improved.

To compensate, I developed an alternative forecaster and decided I would compare the performance of both on the december validation set. I chose an auto-regressive integrated moving average (ARIMA) model. There again, I would explore the performance when varying the orders (p, d, q) around a set of values that I determined from visually inspecting the time series. Given the seasonality (mostly on a weekly time scale) of the data, SARIMAX would have probably been a more suited choice. ARIMA was still useful, under the condition that the time series is applied some transformations to make it stationary (a key condition). A drawback from this alternative method is that it does not support multi-variate time series forecasting.

General Workflow

In the code I submitted, there are several scripts that are intended to perform similar operations depending on whether it is forecasting with the neural network or the ARIMA. In essence, it follows 2 steps:

1/ a pre-processing step, where I extract the features I want from the raw data set, filter, and clean it to obtain my desired object (the aforementioned Ns x Nd array).

2/ a forecasting step, once a model has been fitted in the training/learning stage.

Ideally, a third and final step would have been the post-processing of several forecasts on several time series, to answer the 3 challenge questions.

Data Cleaning

From the data set nomenclature provided by the MTA, and after a short analysis, I decided to equivocate a unique subway "station" as its Unit identication number, with the understanding that a unit has several booths, sub-units, and turnstiles.

From the raw data set, I drop the non-relevant labels such as lines serviced, station name (which isnt unique), and exist counts. Since we are re-binning the data into daily occurrences, I figured the rows of non-regular transmission intervals aren't relevant either. As long as we differentiate the cumulative entries number at a similar time on consecutive days, an under- or over-abundance of entry data isn't a concern.

There was, however, missing entries for consecutive days and weeks, mainly during the June, July and December periods, as showcased on figure 1 below.

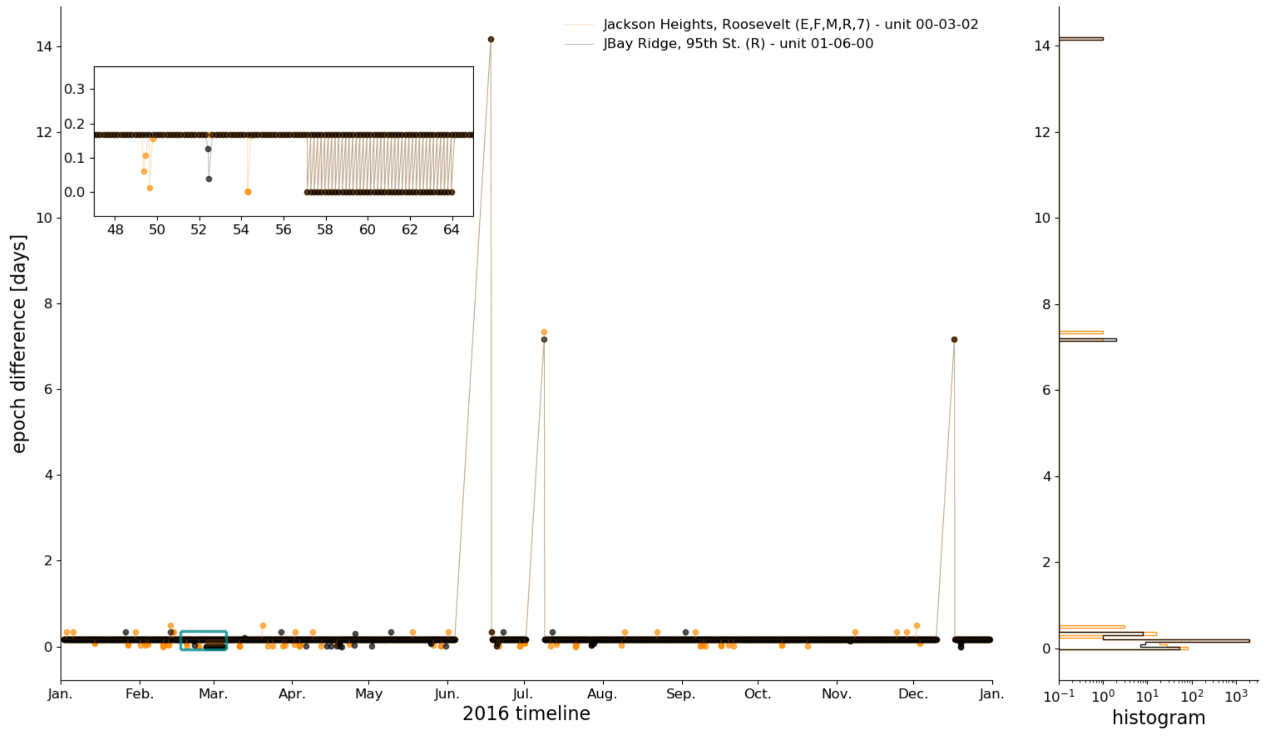

Figure 1 – the jumps in days between two consecutive time indices for 2 particular stations (black and orange). While most of the data is transmitted every 4 hours (0.1667 days), there are 3 main gaps of missing data to worry about.

I filled the gaps by performing some seasonality analysis, illustrated on Figure 2 below. I re-bin the data per week day (Monday, Tuesday …) and per week number (week 1 – 52) and take their average after normalizing by each station. On the one-week folding basis, the weekend days witness less subway usage, and is otherwise relatively constant during the week day (the day labels on the violin plot at off by one day). Intuitively, this could be related to working commutes. One the seasonal base, it appears that the summer months experience lower usage, probably due to holidays. By taking the seasonal average and smoothing it in Fourier space, I make a smoothed seasonal trend line.

For the missing data in the large gaps, I make the assumption that the usage on that day is a combination of its week-day trend (slower on weekends) and a seasonal trend (slower in summer and winter). Since these trends were retrieved from the station-averages, the reconstructed usage must therefore be re-converted (re-normalized) back to the station-specific level.

The result of the cleaning procedure is illustrated on figure 3. On figure 4, I illustrate the sum of all columns to obtain the total daily usage, and highlight the instances were data was reconstructed from the described method. Although it is passable at first sight, significantly more effort needs to be put into adequately parametrising and optimizing this reconstruction process, as it directly influences the results of the forecasting.

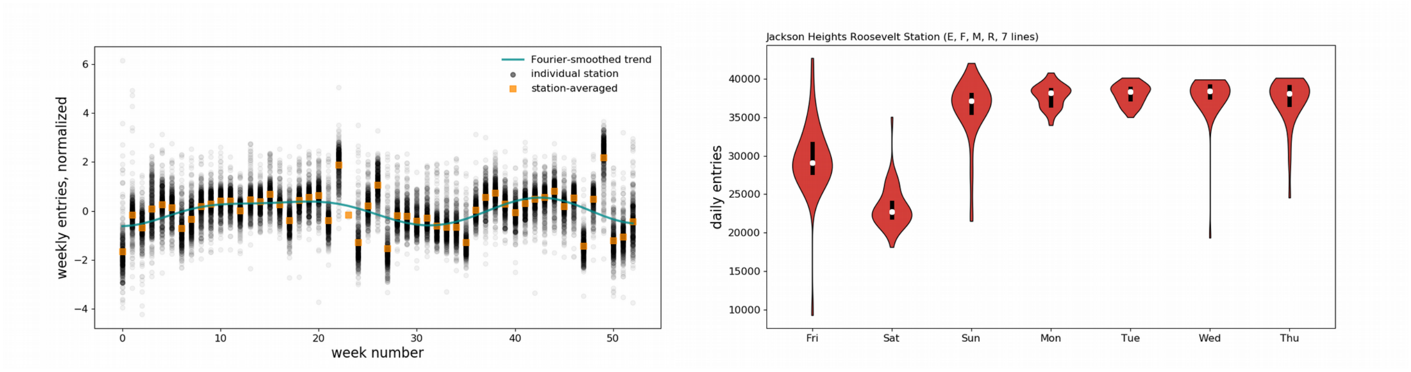

Figure 2 – Left: seasonal trend. Right: an instance of the weekly trend for a specific station.

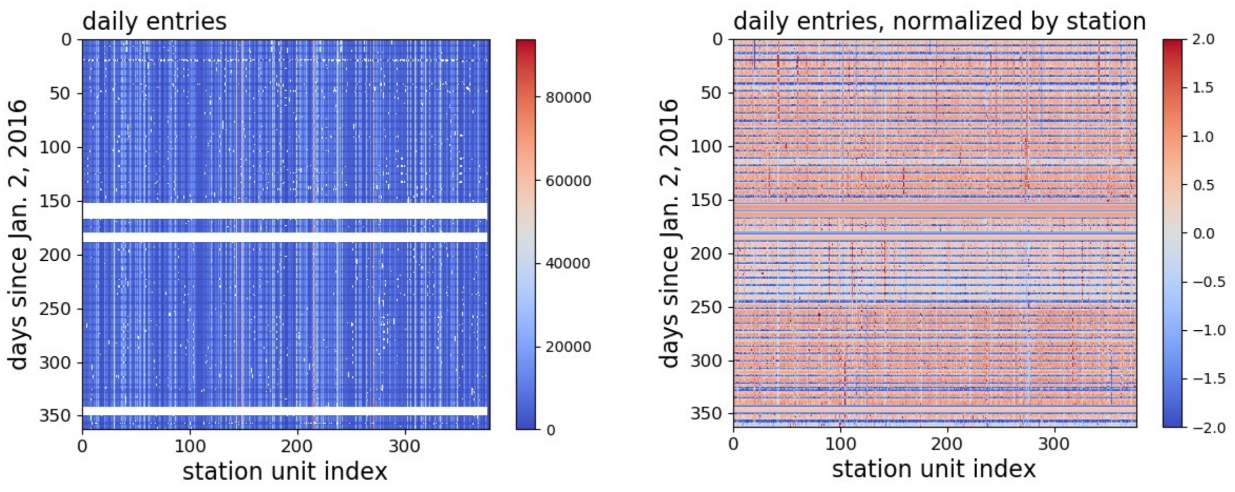

Figure 3 – Time series of each station before (left) and after (right) running the ‘clean.py’ code in the pipeline.

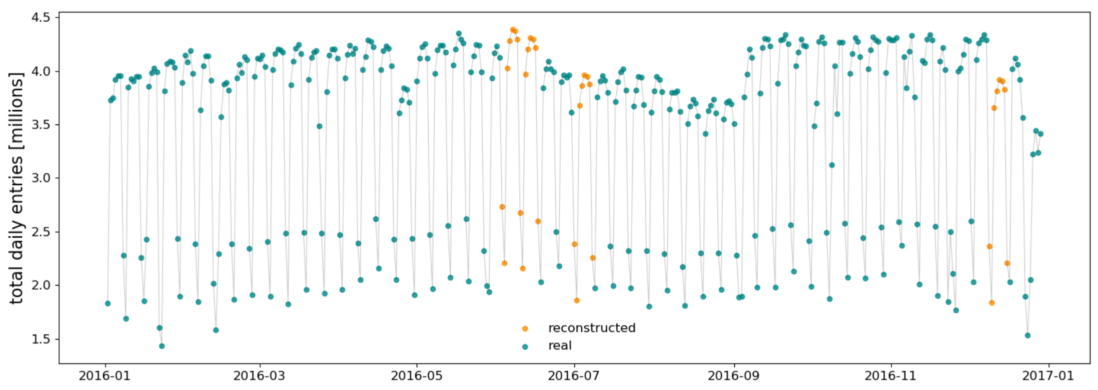

Figure 4 – Real (teal) versus reconstructed (orange) time series of the daily usage totaled over all stations.

Given the time constraint, this was not pursued at this time but is one of the most urgent points to improve in the pipeline.

Forecasting

The time series as such are non-stationary, as they fail the augmented Dickey-Fuller test (whose null hypothesis that the series is non-stationary could not be rejected at the 1% level). However, time-lagging the series by a factor of 7 days shows little auto-correlation beyond 2-3 days, at they reject the ADF null hypothesis, hence we can treat them as stationary time series. So long as we don’t forget to re-transpose the predicted outputs back by the reverse-differentiation, we can apply ARIMA to forecast the next day’s total usage.

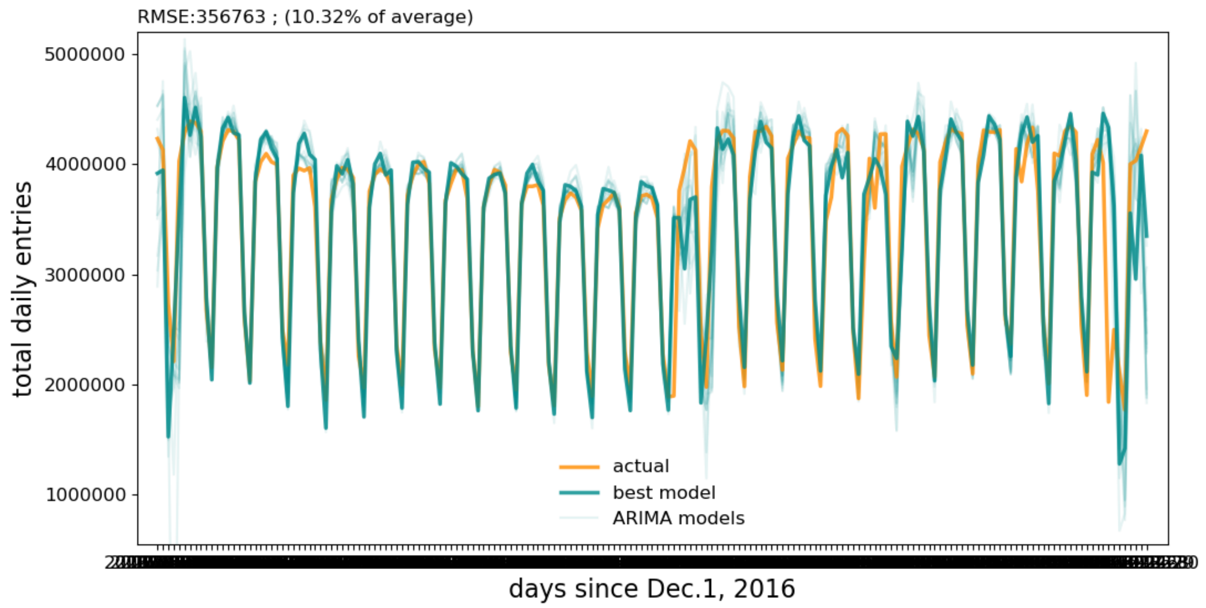

Figure 5 – the day-by day prediction on the testing sample by several ARIMA models with differing orders p, d and q, versus the actual observations. ARIMA(2,0,5) is highlighted in thick teal.

From the correlograms of the time series, a good starting guess for the auto-regressive and moving-average orders is around 2 and 4-5 respectively. I train/test and validate several ARIMA models with orders in the neighboring range of these best guess values. I use the root mean square error metric to compute the score on each forecast, and determine that the best fitted model is ARIMA(2,0,5), not too far off from our initial guess. The training set consists of consecutive days prior to June 1st, 2016, and the testing set spans from June to the 30th of November. Other train/test splits should be considered, especially if we are only to capture seasonal trends from a year’s worth of data.

Once the model determination is set, I train it on the full data set (train + testing) and score its accuracy on the validation set (december), again using root mean square error as the metric. I repeat the procedure with the LSTM encoder-decoder I chose and compare their accuracy as well as the statistics on the residuals (see figure 6). Since data needs to be normalized when training a neural network, the root mean square errors are in terms of the standard deviation from average daily usage. Their summed RMSE are about 0.42 and 0.58 standard deviations. A good metric to evaluate whether the trends and seasonalities were adequately modeled is to take a look at the distribution of the residuals (prediction substracted from outcome), or, alternatively, their power spectrum (Fourier Transform). A good model would have normally distributed residuals, or essentially white noise errors that are purely statistical in nature and do not encapsulate any signal.

There are 2 main issues I see in the residuals’ distribution function (right panel of figure 6). First off, they are not centered around zero, suggesting the errors are systematic in nature. In other words, some unaccounted-for time-dependence or auto-correlation has not been captured by either model. Second of all, there are clusters of probability distribution away from the central spread, which suggest something is clearly wrong with the modeling, as the errors are not even Gaussian in nature.

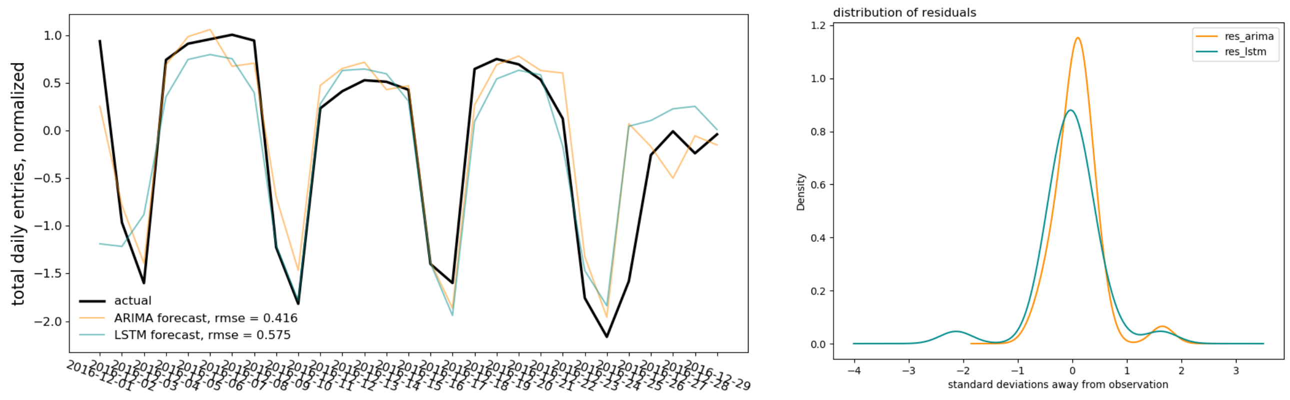

Figure 6 – Left: forecast on the validation (december, black) set for the ARIMA(2,0,5) model and the chosen LSTM encoder-decoder in orange and teal respectively. Right: distribution function of the residuals.

I allocated a little too much of my time and effort into developing the pipeline that would be flexible and modular. As a consequence, given the short time constraint, that is as far as I was able to go. I am not, however, in any shortage of ideas for further development, and with a lengthy piece of code already written, once they are implemented, results could follow promptly.

Technical / coding difficulties aside, here is the outline of how I would improve the pipeline in order to adequately and accurately answer the challenge questions:

1/ Model the seasonality and trends more thoroughly to piece together the missing data, using more standard methods rather than doing it by hand on my own;

2/ Iteratively apply transforms to the time series before the training part, subtracting rolling averages, power transforms, etc, until the residual distribution from the forecasting are normally distributed.

3/ Repeat the testing and forecasting steps for a given model to get a stochastic-averaged performance.

4/ Once this is done, I would batch the training and testing sets differently. I would also try to feature engineer on some of the time series. For instance, group together stations on the same line or servicing the same geographical location, forecast 2 – 3 days in advance rather that only one.