Comparing Time-Series Models to Predict Missing Person Data from San-Francisco Police Database

Jessica Brown, He Yang

Abstract:

1. Introduction

The San Francisco government of California publishes their public data onto their government website. Among these are police incident reports. This paper aims to predict when specifically Missing Person incidents will occur in a matter of number of occurrences per month.

Predicting crimes could be useful to a police department in terms of allocating resources which in term saves time and money for the police department itself the the tax paying citizens of a city.

The methods employed utilize the programming language R and Python with various libraries openly available on the internet. R will be used to forecast time-series data with multiple statistical models. Of these models are Simple Exponential Smoothing, Autoregressive Integrated Moving Average (ARIMA), Facebook Prophet, and K-Nearest Neighbors (KNN). We will also use Long Term Short Memory (LTSM) but this was done with Python instead of R.

2. Data

As mentioned previously, all data is pulled from the San-Francisco government's official website (data.sfgov.org). We will be analyzing the dataset called *Police Department Incident Reports: 2018 to Present*. This dataset consists of over 760,000 data points and 35 columns. Missing Person data is only 16,457 data points. Because much of 2023's data is incomplete, we decided to narrow our focus to 2018 through 2022 making the number of data points 14,814.

This makes 14,814 data points to be analyzed from 2018 to 2022. The data was split into two separate parts for the purpose of forecasting. The data from 2018-2021 will be used to predict the number of incidents per month in the first half of 2023. The terminology for 2018-2021 is the training set whereas 2023 is the testing set. The more accurately the training set can predict the testing set, the more accurate the model.

Data pre-processing for this dataset was quite simple. We only wanted to analyze "initial reports" as supplemental reports are extensions of the same incident. Counting supplemental as well would result in double-counting. When only taking the initial

reports, the data points fall from 14,814 to 9,494 data points. That is 9,494 data points throughout the entirety of 2018 till the end of 2023. After this, since we are only counting the number of incidents, the only thing to do is count the number of incident reports per month and by year. From this point, any null values in other columns are irrelevant as long as each incident has a datetime. Most models were able to run R's built-in time-series structure indicated as ts(). However, Facebook Prophet requires its own data structure so the data had to be restructured into two columns denoted ds (datetime) and y (values). LTSM (Long Term Short Memory) on the other hand was run in Python using

3. Methodology

3.1 Exponential Smoothing Algorithm

The R library used for exponential smoothing was HoltWinters [7]. Exponential smoothing forecasts time-series data by giving more weight the recent observations while giving lessening weight to older observations.

The formula that the R HoltWinters library uses is [7]:

$$\hat{Y}[t+h] = a[t] + hb[t] + s[t-p+1+(h-1) \bmod p],$$

Here, a is alpha, b is beta, and s is gamma. Y is the forecast, t is the value at the observed time, p is the time series' period length, and h is the selected number of periods to forecast. Alpha is a weight that prioritizes the previous value between 0 and 1. For example alpha of .5 would be equal to averaging the previous and current period value to get the forecast.

We used two exponential smoothing models. They are denoted as SMS for Simple Exponential Smoothing and Holt's for Holt's Method which is also known as Double Exponential Smoothing in *Figures 6-9*. The difference between the two is SMS uses only alpha while Holt's Method uses alpha and beta [6]. The addition of beta allows trends to be modeled. Whereas alpha denotes weights as described in the previous paragraph.

HoltWinters automatically calculates alpha and beta through embedded formulas denoted as [7]:

$$a[t] = \alpha(Y[t] - s[t-p]) + (1-\alpha)(a[t-1] + b[t-1])$$

$$b[t] = \beta(a[t] - a[t-1]) + (1-\beta)b[t-1]$$

3.3 Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) was calculated using the Arima function in the R package forecast [8]. ARIMA utilizes the dependent relationship between observations and lagged observations to create a moving average.

The formula used by forecast's ARIMA [8]:

$$X_t = a_1 X_{t-1} + \cdots + a_p X_{t-p} + e_t + b_1 e_{t-1} + \cdots + b_q e_{t-q}$$

Importantly, Arima uses values p, d, and q where p is the AR (auto-regressive) order, d is the degree of differencing, and q is the MA (moving average) order. The function auto.arima in forecast will automatically provide you with the correct p, d, and q.

3.4 Facebook Prophet

Our prophet observation was granted with the use of the R package Prophet. Prophet uses trends, seasons, and holidays to make trendlines then choose the best predictors for forecasting [9]. As prophet is modeled after daily data, unlike with the other methods used, we had to feed prophet the daily crime reports instead of feeding it per month data. For comparison with other methods, we took the results for each day and summed them up into their respective months to get an overall month value.

3.5 K Nearest Neighbors

This model was achieved with the help of the R package tsfknn [10]. K Nearest Neighbors (KNN) is a model where data is grouped into clusters based on similarity to each other. It forecasts a value by comparing its properties with already existing values.

3.6 Long Term Short Memory

LTSM is a type of RNN (recurrent neural network) that is meant to differ from other RNNs by capturing long-learning range dependencies. To achieve this, we used Python and the package tensorflow. Tensorflow is ubiquitously a complicated package. Due to the difficulty of getting it to function in R using RStudio, we had to default to using Python in Spyder specifically with Anaconda Navigator. Otherwise, we could not get it to function.

Important concepts in LSTM and neural networks in general are epochs and learning rates. To get the best LSTM rates, you need to test for the best epoch and

learning rate. The number of epochs is the number of passes through the training dataset where too few epochs leads to underfitting and too many epochs leads to overfitting the training data. Each of which leads to bad results. Learning rates determine the adjustment of weights in each cycle where again a learning rate too low or high will lead to poor results.

Unlike other models, there were no built-in functions in the packages to determine the best parameters. Here those parameters were the learning rate and epochs, so all had to be done manually. Therefore, this will be detailed in *Section 4*. Similarly to Facebook Prophet, we used the by day reports instead of by month for practicality of the model.

4. Results & Comparison

In this section, we will detail how all of these methods compare with each other. First of all, we will go more into detail about LSTM epochs and learning rates.

Epochs 100, 200, 300, 500, and 1,000 were tested. Learning rates 1, 0.1, 0.01, 0.001, 0.0001, and 0.00001 were tested. This means in total, we performed 30 different epoch and learning rate combinations.

Below are the results of error parameters of choice. This includes MAE (mean absolute error), RSFE (running sum forecast error), TS (tracking signal), and MSE (mean squared error).

MAE is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

It measures the absolute average difference between average and predicted values in predictive models. The smaller the MAE, the better. An MAE of 0 would mean the prediction is the same as the actual value.

RSFE is given by:

$$\text{RSFE}_t = \sum_{i=1}^{t}(F_i - A_i)$$

The RSFE is the cumulative sum of the prediction minus the actual value. Again, the closer to 0, the better.

Tracking signal is given by:

$$\text{Tracking Signal} = \frac{\text{RSFE}}{\text{MAE}}$$

Tracking signal indicates systematic bias or error in forecasting based on the ratio of RSFE to MAE. The closer TS is to 0, the better the prediction.

MSE is given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

It gives the squared difference between actual and predicted values. The closer to 0, the more accurate the prediction.

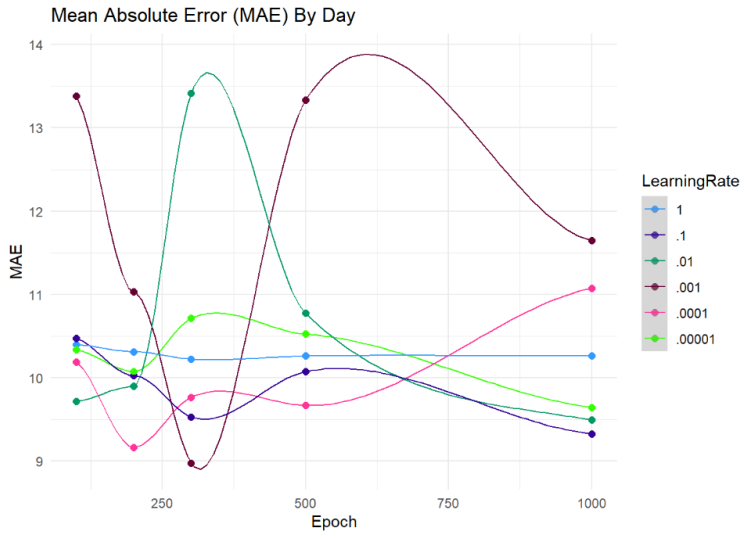


Figure 1

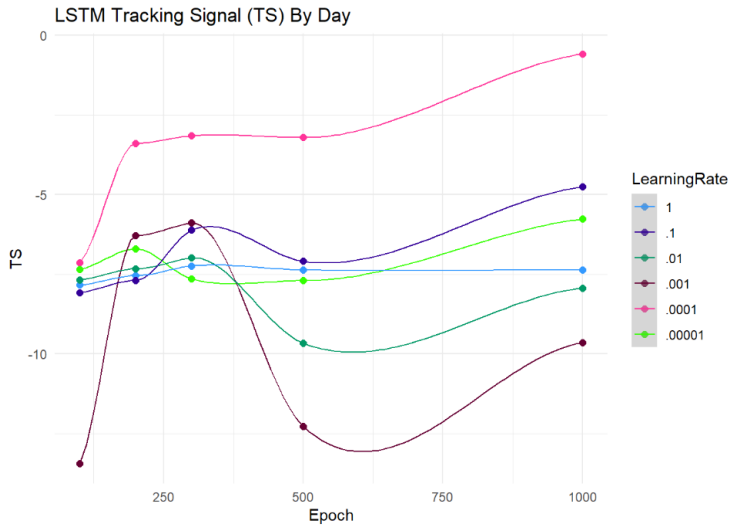


Figure 2

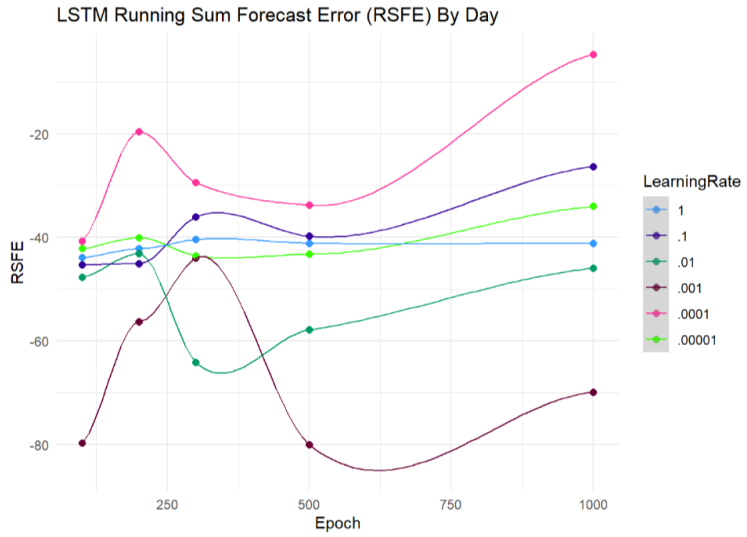


Figure 3

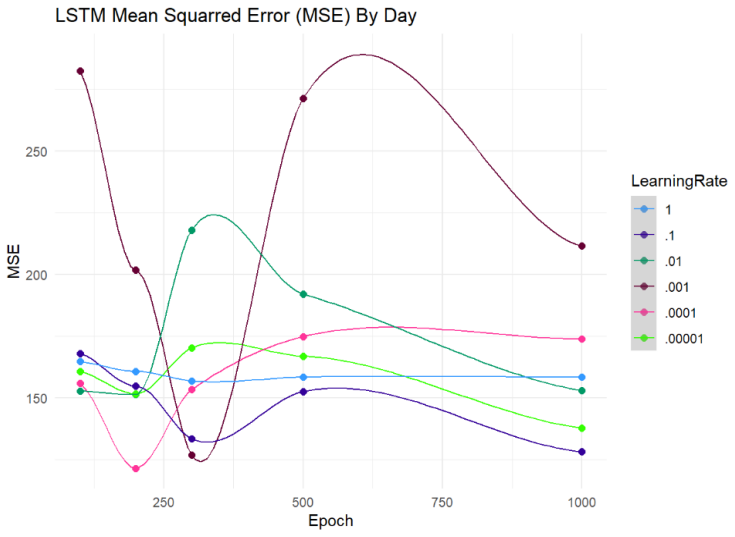


Figure 4

The learning rates 0.001 and 0.01 for some reason spike at certain epochs, typically 300 and 500 epochs. The learning rates 0.001 and 0.01 overall have very inconsistent values across epochs. This can be seen especially in MAE and MSE. The worst performance is given to the epoch of 500 and learning rate of 0.001. It performed the worst on every single metric.

The learning rate 0.0001 and epoch 1000 have the best results in TS and RSFE. The values are significantly closer to zero than all other combinations. Its performance in MSE and MAE pales in comparison to other combinations, yet it still has acceptable values.

The learning rates of 0.1 and .00001 perform similarly at all epoch values. Their epochs of 1000 perform the best or second and third best on every single metric. The learning rate of 0.0001 and 1000 epochs was only picked over it because it performed significantly better in TS and RSFE than 0.1 and 0.00001 which we weighted as more important metrics than MSE and MAE. This is because a bad tracking signal indicates systematic bias or error and the RSFE indicates the overall difference in prediction and actual values. A very small RSFE means that the overall prediction and actual values are almost the same.

4.1 Baseline Comparison

here insert a figure with all actual predictions ina graph near each other

Figure 5

Figure 6

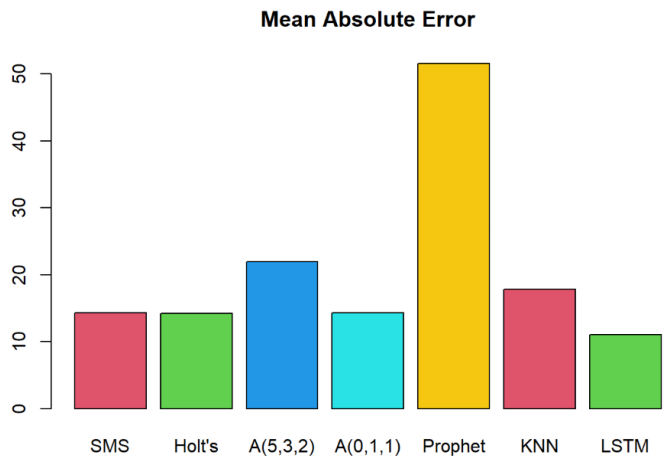
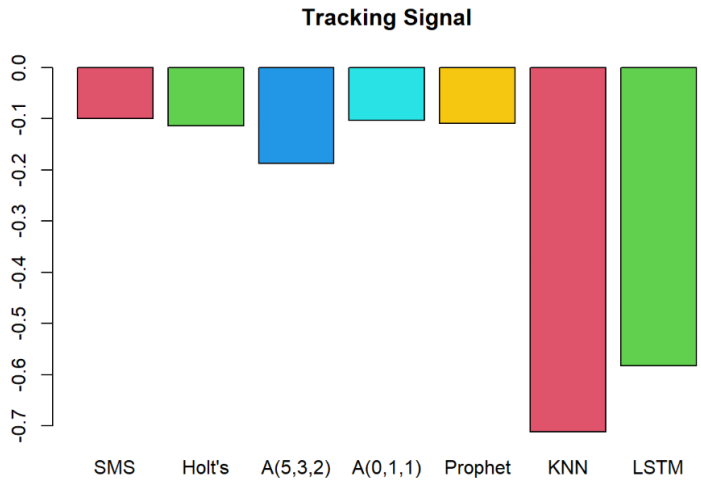


Figure 7



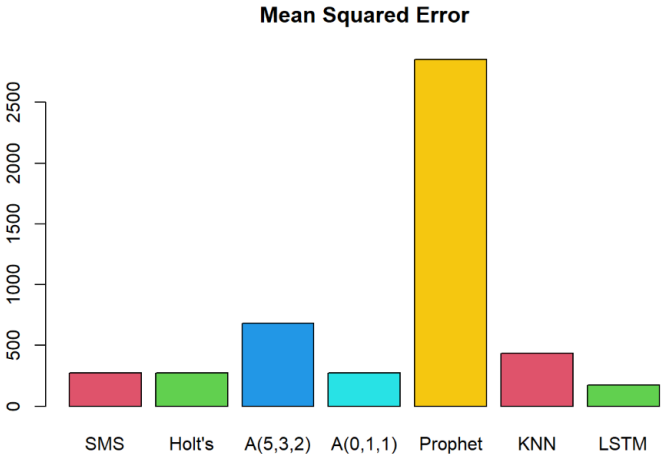
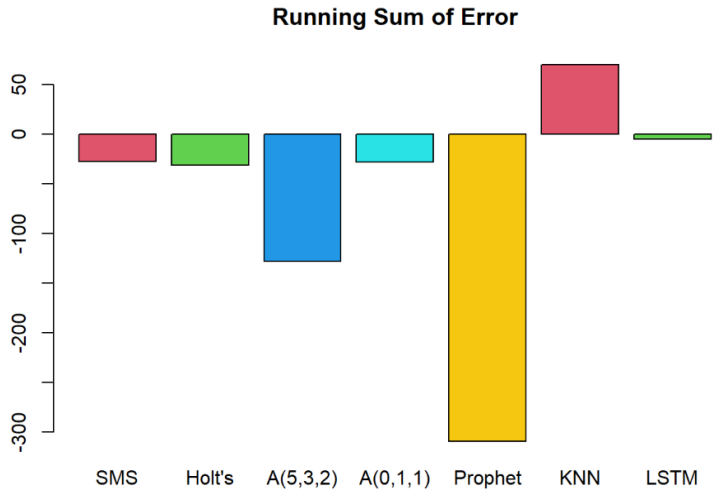


Figure 8

Figure 9



***need to remove A(5, 3, 2) and just call A(0,1,1) ARIMA**

4.2 Discussion & Analysis

Now we will discuss the performance of these six models which are Simple Exponential Smoothing, Holt's Method, ARIMA, Facebook Prophet, K Nearest Neighbors, and Long Term Short Memory. We will use the same parameters used to compare LSTM parameters being MAE, TS, MSE, and RSFE.

For MAE, Prophet performs the worst with a value of 51.578. LSTM gives the best performance of 11.073. Prophet's performance is significantly worse by comparison as all other models have a value at 22 or lower.

Tracking Signal has KNN giving the worst performance with -0.711. LSTM is not too far behind KNN with a value of -0.583. The best performance is Exponential Smoothing with -0.1. KNN and LSTM perform significantly worse than all other models as they all are between -0.1 and -0.2.

MSE has Facebook prophet again performing the poorest with a value of 2853.398. The best would be LSTM with the value of 173.897. Outside of Prophet, no other metrics hit the 1000 marks, so once again it is showing much worse results.

RSFE has LSTM giving another best performance with -4.664 with Prophet giving the worst of -309.465. All other metrics give a value within 70 points of 0 showing the prophet's poor performance. Interestingly, KNN is the only metric that overestimates whereas all other metrics underestimate the total of actual values.

Overall, LSTM gives the best performance, Prophet the worst, and KNN would be the second worst. All other metrics are comparable to LSTM performance with LSTM performing better in all error measures apart from a bad tracking signal outcome. If someone were to be wary of the tracking signal from LSTM, picking ARIMA, Exponential Smoothing, and Holt's Method all give comparable results.

5. Conclusion & Future Works

This study estimated the amount of missing person cases using the time series models Simple Exponential Smoothing, Holt's Method, ARIMA, Facebook Prophet, K Nearest Neighbors, and Long Term Short Memory. The performance was compared with MAE, MSE, TS, and RSFE.

The results indicate that LSTM is best suited for this dataset while Prophet is the worst. LSTM performed best in MSE, MAE, and RSFE while Prophet performed worst in MAE, MSE, and RSFE. Therefore we conclude that LSTM is the best for short-term forecasting for missing persons.

The limitations of this study would be that a small dataset was used of 9,494 data points from 2018 to 2022, so bigger datasets may need to be tested to prove the validity

of these conclusions. Also, as this is analyzing missing persons data in San Francisco, California USA, these results may not be generalizable for other regions.

Future works would point to looking at additional parameters other than just datetime to predict how many cases will happen and when.

We would recommend for other to look further into LSTM and how to get even better results with different parameters or even other types of recurrent neural networks.

RESOURCES

https://data.sfgov.org/Public-Safety/Police-Department-Incident-Reports-2018-to-Present/wg3w-h783 [1]

https://datasf.gitbook.io/datasf-dataset-explainers/sfpd-incident-report-2018-to-present [2]

https://cran.r-project.org/web/packages/tsfknn/vignettes/tsfknn.html [3]

http://rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-r/ [4]

http://www.iapress.org/index.php/soic/article/view/1767/1025 [5]

https://statisticsbyjim.com/time-series/exponential-smoothing-time-series-forecasting/ [6]

https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/HoltWinters [7]

https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/arima [8]

https://youtu.be/2XFro0nIHQM?si=Wbqnvh383Xx_Y2CC [9]

https://cran.r-project.org/web/packages/tsfknn/vignettes/tsfknn.html [10]

https://www.youtube.com/watch?v=c0k-YLQGKjY&t=301s [11]