# TIME-SERIES FORECASTING ON SPARSE DATA

Justin T. Pacella
October 2022

## TABLE OF CONTENTS



## EXECUTIVE SUMMARY

One simple yet seemingly unsolvable problem that plagues employment of any form is the ability to predict the future. Whether it be staffing, food prep, or hospital capacity, the ability to preemptively know these things is invaluable to employers and employees alike. Although humans are good at making such predictions, computer programs known as Time-Series Forecasters (TSFs) notice complex trends that we do not. Today, TSFs are mainly used by massive companies that have billions of data points at their disposal, the reason being that TSFs require copious amounts of data to be effective. Also, there is a trade-off[1] between a TSF's accuracy and the breadth of its *prediction range*[2]. Discovering methods of circumventing these

[1] Akhter (2019)
[2] How far into the future a TSF can predict, also referred to as the *forecasting horizon*

issues is among the primary goals of modern data analytics. This report will explore one of these strategies.

I designed one such method to predict the *NEDOC score*[3] of the Presbyterian University Hospital (Presby) Emergency Department. The method combines several common forecasting and image processing techniques to improve the model's accuracy and prediction range. To make a prediction, the trained model looks at NEDOC data from the past two weeks and uses it to predict a score curve for the next day. The model is inexpensive to train and works well on sparse data, making it well suited for a wide variety of small-scale forecasting applications.

## PROBLEM OVERVIEW

The problem started as a simple question: How can machine learning principles be applied to forecast the NEDOC Score for the Presby ED? I started by trying a few existing TSFs, including F. Chollet's *Xception*, K. He's *ResNet50,* and H. Sanchez's CNN-RNN Hybrid Network, but found that none were successful[4]. These three are complex models known as *Deep Neural Networks*, which are designed to emulate a human brain. For my application, these could only predict a short while (e.g., a few hours) ahead before the prediction accuracy becomes too low. This is because these TSFs are not designed for sparse data sets. Circumventing this was the key challenge of the project.

## GENERAL PROCESS

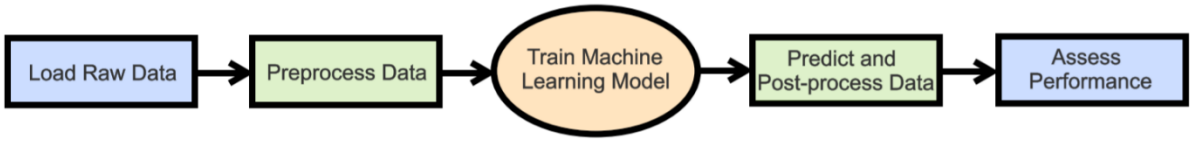


Figure 1

When solving a machine learning problem, the programmer must experiment with many different techniques before they can reach a solution. These experiments often have many moving parts and can become overwhelming, so it is useful to simplify the problem. The

[3] The *NEDOC Score* is a measure of how busy a hospital's emergency department is, based on patient count, admission times, ICU patients, etc.
[4] Chollet (2016), He (2015), and Sanchez (2022), respectively

flowchart above (Figure 1) breaks a machine-learning training program into five fundamental steps:

1) The first step is to load the data set. This entails loading the completely unaltered data set, then cleaning it by filling in, or *imputing*, missing values, as well as parsing it for relevant information. The data set contains many samples, each consisting of an input and a response. The aim of the process is to train a model that predicts an accurate response for a given input. This step is universal and does not vary much between programs.
2) The next step is to preprocess the data. Although this step is often simple, there are many techniques that can improve performance and training time prior to training. Some of these techniques are *Autoregression*, *Data Folding*, and *Principle Component Analysis* (PCA), all of which usefully reshape the data. The universal part of this step is to partition the data into two sets. The first set, called the training set, is used in the third step to train the model. Effectively, the model "sees" only the training set. The second set, known as the testing set, is withheld from the model. The programmer uses this set to assess the model's performance in the fourth and final steps.
3) In this step, the training set is repeatedly passed into the model. The model starts out making random predictions but improves with each *epoch*[5]. As the number of epochs grows, the model's performance improves. Once the training has run for the programmer-specified number of epochs, the training routine terminates. There are many options to select before training that alter the speed and effectiveness of the training process. The most important of which is to select a model, such as a *Binary Decision Tree* or *Neural Network*. For complicated models and large data sets, the training program can take days or even weeks to complete. This step yields a fully trained model ready to predict.
4) The second to last step uses the trained model to make predictions from the testing set. The testing set's inputs are passed into the model, which yields an output for each input. To properly visualize the responses, the programmer must undo the preprocessing steps from step two, or *post-process* the model outputs. The resulting responses should be identical in structure to the target (desired) responses after completing step one.

[5] An epoch refers to one complete pass of the training set

5) The last step is to compare the original testing responses to the responses from the model. The main goal is for these responses to be as close as possible to each other, but there are other factors to consider. For example, if one prediction is close to but shaped differently from its target, and another prediction is farther but more accurately shaped, there is a dilemma. In terms of closeness, the former performs better, whereas a human viewer might think the latter is a better prediction. It is the responsibility of the programmer to devise a proper means of comparing the responses, or a *performance metric*. This metric will look at both responses and return a performance value, which is scaled such that a lower value means better performance. These performance values should be consistent with how the programmer (and their colleagues) rank the prediction's accuracies.

# SOLUTION

The solution I devised occurs in the preprocessing step. It involves reshaping the NEDOC score data using the three techniques mentioned in step two of the General Process. Counterintuitively, these reshaping techniques reduce the number of samples and make the data significantly sparser. However, the reshaping, particularly the *Data Folding*, innately extracts a daily pattern from the NEDOC scores. These patterns, referred to as a daily score curves (DSCs), make up the new folded data, which is used as the model response. This means the model automatically predicts one full day ahead of time, far surpassing the forecasting horizons from the previous attempts. For the model, I opted for a small *Recurrent Neural Network* (RNN), which is much less complex than *Xception* and *ResNet50*. RNNs are deep neural networks that specialize in serial data, making them useful for time-series forecasting. The benefit of using a smaller model with a smaller data set is that the training process finishes faster. The figure (Figure 2) is a detailed illustration of the preprocessing routine.

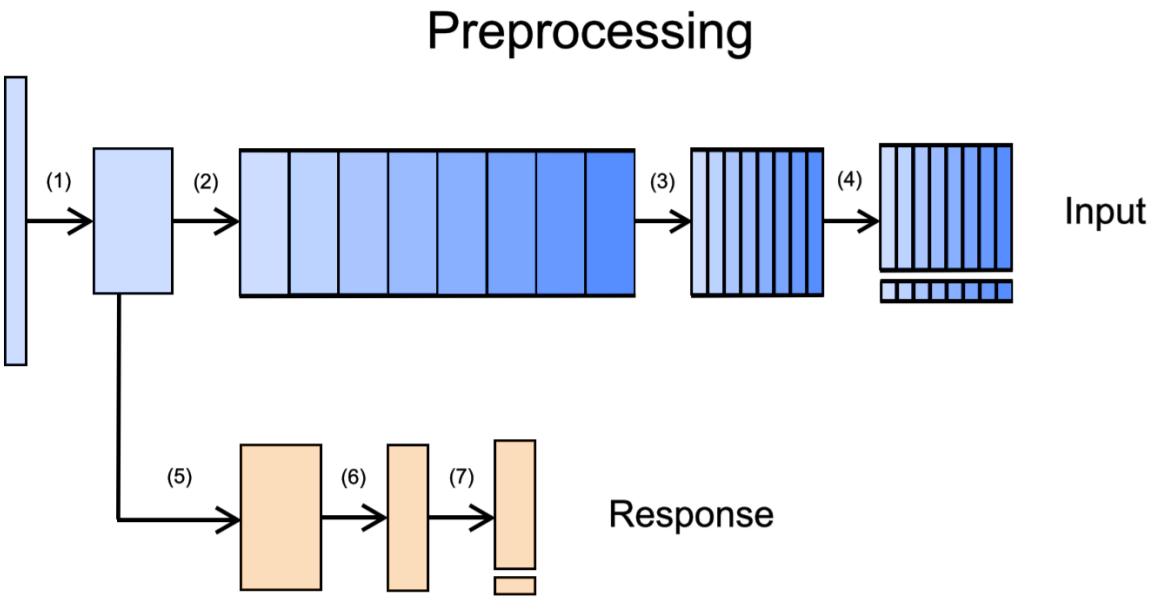


Figure 2

(1) Data Folding

The data folding step casts the original NEDOC score data from a list of about three years' worth of scores sampled on half-hour intervals (about 60,000 data points) to a new shape. The result is a list of every day inside the timeframe, with a list of half-hourly-sampled scores for each day (about 1300 days, 48 samples per day).

(2) Autoregression

This step applies a few[6] lags to the folded data. Each lag appends an identical copy of the folded data that is pushed back by one more day than the previous lag (starting at one). This is what allows the program to look back at the past few days to make a prediction.

(5) Response

As mentioned above, the model uses the folded score data as its response. This step copies the folded data and relabels the duplicate as a response.

[6] The figure shows eight applied lags. In actuality, the model uses fourteen lags.

(3) & (6) Principle Component Analysis (PCA)

Steps (2) and (5) inadvertently introduce a problem: if the *dimensionality*[7] of the data is too high, the concept of distance becomes meaningless. This is known as the *Curse of Dimensionality*, and it hinders the performance of the model. This problem is most often encountered in image processing algorithms and has many solutions, one of them being PCA. PCA uses statistical concepts to compromise some of the data's variance to reduce its dimensionality.

(4) & (7) Training/Testing Split

The last step(s) of the of the preprocessing routine is to partition the data into a training set and testing set.

# RESULTS & CONCLUSION

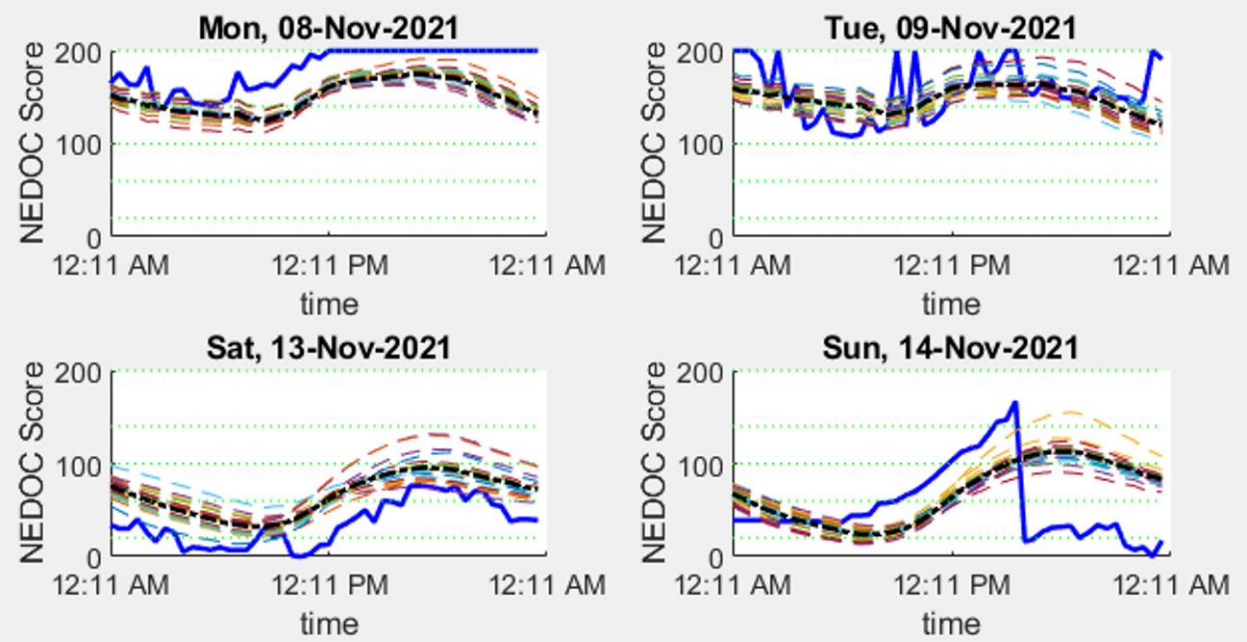


Figure 3

Each plot in Figure 3 shows how the NEDOC score changes throughout the date above the plot. The thick blue line is the observed DSC, and the black dot-dashed line represents the

[7] In Figure 2, *dimensionality* can be thought of as the width of each rectangle

average predicted DSC of several identical[8] models. The rest of the dashed curves represent the individual models.

All these plots show that the TSF, operating on sparse data, can reliably predict one day into the future. Despite being the least accurate plot of the group, the bottom right plot prediction matches the general trend of observed DSC. As time progresses and the hospital database grows, this algorithm will see vastly improved performances and much wider prediction ranges.

At Presby, this program could increase the efficiency of operations in the emergency department. Beyond that, scheduling managers at most workplaces would benefit immensely from such a predictor. To go even further, natural sciences, political sciences, and any other fields of study involving small-scale time-series data sets could benefit from such forecasters.

[8] The models' predictions are not identical due to the random initializations from the training process. The models' structures and options are all identical.

# BIBLIOGRAPHY

Akhter, M.N., Mekhilef, S., Mokhlis, H. and Mohamed Shah, N., "Review on forecasting of photovoltaic power generation based on machine learning and metaheuristic techniques", 2019 IET Renewable Power Generation. 13: 1009-1023. https://doi.org/10.1049/iet-rpg.2018.5649

Chollet, François, "Xception: Deep Learning with Depthwise Separable Convolutions", 2016 arXiv, doi: 10.48550/ARXIV.1610.02357. https://arxiv.org/abs/1610.02357

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian, "Deep Residual Learning for Image Recognition", 2015 arXiv, eprint: 1512.03385. https://arxiv.org/abs/1512.03385

Sanchez, H., "Time Series Forecasting Using Hybrid CNN – RNN", 2022, MATLAB Central File Exchange. https://www.mathworks.com/matlabcentral/fileexchange/91360-time-series-forecasting-using-hybrid-cnn-rnn