FORECASTING FUTURE ERRORS USING AN ARIMA MODEL FOR TIME-SERIES ERROR LOGS

Anish Rao

University of Michigan: College of Engineering

Advisors: Dr. Eric Johnsen, Suyash Tandon

April 12, 2019

Contents

1 Abstract

Errors and error handling have always been a core concept in computer science, but the step from a single-computer system to a multi-nodal, and sometimes even multi-cluster supercomputer has made this much harder and more important. Errors in a supercomputer can have affects on other machines within the network, making debugging an error much harder.While home and business machines have, on average one to two Graphical Processing Units (GPUs), and one Central Processing Units (CPU), allowing for simpler error handling and definitive sources for errors, most supercomputers have hundreds to thousands of both GPUs and CPUs. The question for a supercomputer is not only where an error manifested, but also what errors it caused. A Non-Seasonal Autoregressive Integrated Moving Average (ARIMA) machine-learning prediction model for predictive analysis of time-series data is one way that we can begin to see the relations between errors, and predict new occurences at high likelyhoods.

The ARIMA model is a good one to predict these errors with, as it is dedicated to moving over chunks of time-series data that may have moving averages, which we are granted due to the error logs (a log of time-series data on errors) held by supercomputers. ARIMA models are still widely used in predictive statistics for user retention rate studies, and purchase forecasts. In this study, I test my ARIMA model against given data, and find that my model's succesful prediction rate is that of approximately 74.7%. From here, due to my model's relatively high success rate, I can state that there most certainly are true realtions in the data between errors, and that the creation of a neural network such as an LSTM could lead to better results.

2 Introduction

Error reduction and prediction is necessary for the efficiency of a supercomputer. Without it, a supercomputer can propagate errors and cause critical failures in the program, or fail to output. I begin this project under the assumption that there is in fact some relation between when an error happens, and what error will happen in the future. Prediction and understanding of said relation would help mitigate error propagation, and allow for supercomputer proccessess to go unhindered.

The error logs that various supercomputers are wont to produce are the object of interest, as they all have one thing in commmon—their time-series nature. Despite having different nodal structures (e.g. multi-cluster versus singular cluster) they all report an error with its time of conception, and report the time of its fixing, as well as the type of error. It is of interest to analyze these time-series error logs for relations between error conception times, in order to predict future error conception times.

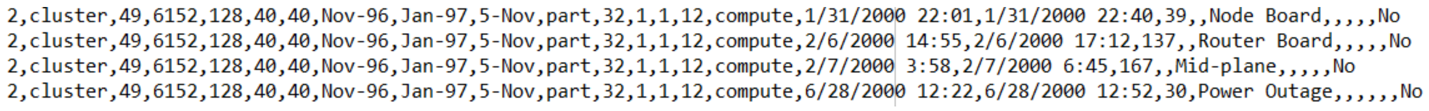
This study is focused on the creation of a machine learning model (using ARIMA) in order to predict future error occurence times, and types. This study will exclusively work with the non-seasonal ARIMA model for prediction, and will compare the model's prediction to actual data gathered to see a correct rate. In this, I create an ARIMA model with custom boundaries, as well as reshaped the data to fit my purposes, and fit the model.

3 Background Information

Within any supercomputer, there are multitudes of nodes which are the computers contained within a cluster, with there being numerous clusters. Our problem is that in many cases errors propagate throughout a cluster, which makes errors appear to have no cause. To figure out where errors propagate from, and from this, predict a new errors location, we use machine learning techniques.

Machine learning is, essentially, automated data analysis that learns from data, and identifies patterns within data that can apply to future trends. There are various machine learning techniques such as: neural networks, decision trees, and Bayes classifier algorithms, all which help predict future occurences. We use machine learning due to the size of error logs, as well as in order to achieve the goal of being able to predict future terms accurately. To this affect, we use an ARIMA model to predict our data.

An ARIMA model is a model that is fitted to time-series data to predict future terms. For this project, ARIMA was chosen due to its flexibility in allowing both Autoregressive (AR) and Moving Average (MA) components. We choose to use ARIMA here rather than an LSTM or another RNN due to ARIMA's capability to make actual probabilistic statements about temporal correlations. LSTMs on the other hand, are black boxes which cannot truthfully say if there exists any such relation. The usefulness of modeling AR components can be thought of, in laymans terms, as modeling change since prior occurence. Modeling MA components captures smoothed trends within our data, while the I in ARIMA, Integration, gives us differencing levels, for making data stationary. Our data is time-series, connected to events, as shown in Figure 1, below.



```
2,cluster,49,6152,128,40,40,Nov-96,Jan-97,5-Nov,part,32,1,1,12,compute,1/31/2000 22:01,1/31/2000 22:40,39,,Node Board,,,,,No
2,cluster,49,6152,128,40,40,Nov-96,Jan-97,5-Nov,part,32,1,1,12,compute,2/6/2000 14:55,2/6/2000 17:12,137,,Router Board,,,,,No
2,cluster,49,6152,128,40,40,Nov-96,Jan-97,5-Nov,part,32,1,1,12,compute,2/7/2000 3:58,2/7/2000 6:45,167,,Mid-plane,,,,,No
2,cluster,49,6152,128,40,40,Nov-96,Jan-97,5-Nov,part,32,1,1,12,compute,6/28/2000 12:22,6/28/2000 12:52,30,Power Outage,,,,,,No
```

Figure 1: Error Log example

3.1 Time-Series Data

Time series data presents itself as figure 1 shows, but we want it to be more friendly to our model. We start discretizing data into small intervals of time in order to have some semblance of weight for each "chunk" of time. Time series data that we wish to predict looks like Figure 2. We can use this in order to regress our errors on their

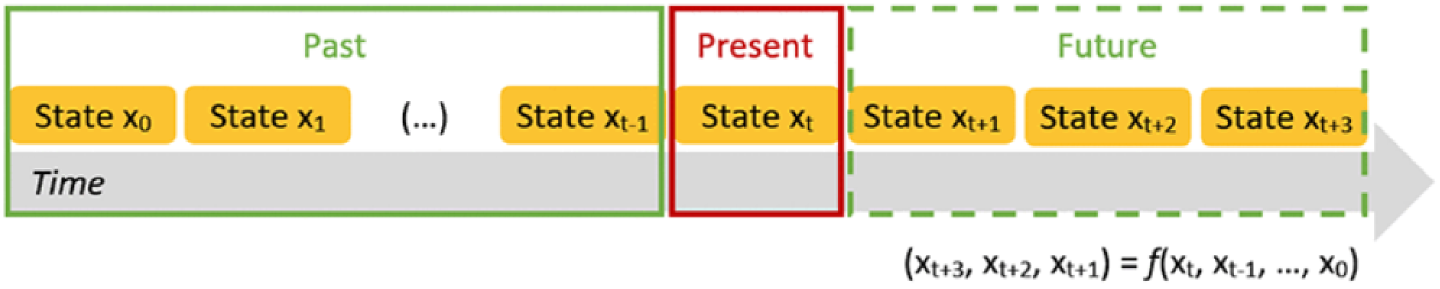


Figure 2: Time-Series diagram

previous values, which is how this is "machine learning," as well as making liberal use of when an error happened, not just which error happened.

4 ARIMA Overview

The time-series nature of our data means that we need to see the correlations between different errors, which lends a hand to finding out error causality. While exponential models and linear regression may seem to be easier, many times, these models are simply special cases of ARIMA models (simple exponential smoothing is nothing but ARIMA(0,1,1)). To better understand this model, we can see that ARIMA is based on a few key rules:

- The object of interest is regressed on its prior values
- The regression error is a linear combination of error terms
- The data values are replaced by the difference between the value, and its prior value.

Our Non-Seasonal variation has three non-negative variables, p, d, and q:

- p: Number of time-lags of the autoregressive model (Using p=2)
- d: Number of times the data subtracted prior values (Using d=1)
- q: Order of moving-average model (Using q=0)

The ARIMA(p,d,q) model is given by:

$$X_t - \alpha_1 - \ldots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \tag{1}$$

This can be generalized as:

$$(1 - \sum_{i=1}^{p'} \alpha_i L^i) X_t = (1 + \sum_{i=1}^{q} \theta_i L^i) \varepsilon_t \tag{2}$$

Assuming $(1 - \sum_{i=1}^{p'} \alpha_i L^i)$ has a unit root of multiplicity d:

$$(1 - \sum_{i=1}^{p'} \alpha_i L^i) = (1 - \sum_{i=1}^{p'-d} \phi_i L^i)(1 - L)^d \tag{3}$$

Then an ARIMA(p,d,q) model with d roots can be given by:

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d X_t = (1 + \sum_{i=1}^{q} \theta_i L^i)\varepsilon_t \tag{4}$$

X_t: Time series data of real numbers

t: Times

α_i: Parameters of autoregressive component (AR(p))

L: Lag operator

d: Integration order

ϕ_i: Parameters of the autoregressive component (AR(p))

ε_t: Error terms

θ_i: Parameters of moving average component (MA(q))

Since our d=1, this means we only use one difference per value, essentially:

$$Y_t = Y_t - Y_{t-1} \tag{5}$$

In terms of y, and mean μ our forecasting equation for the predictions is:

$$\hat{y}_t = \mu + (\sum_{i=1}^{p} \phi_i y_{t-i}) - (\sum_{j=1}^{q} \theta_j e_{t-j}) \tag{6}$$

5 Methodology

Methodology:

Our data was very poorly formatted, so to begin with, I reformatted the data to fit our specifications. First, I changed the mm/dd/yyyy format into one of just minutes. Then, I zeroed the data around the initial value, such that the first occurence within each node was at time 0, and sorted the data as follows:

```
 for vector in data:
       Date_Hour_Min=vector[16].split()
       currDate=Date_Hour_Min[0].split('/')
       currDay=float(currDate[0])*30+float(currDate[1])+float(currDate[2])*365
       hour_min=Date_Hour_Min[1].split(':')
       Time=(((float(hour_min[0])+float(hour_min[1])/60)/24)+currDay)
       vector[16]=float(Time)/(24*60)  #time in minutes

   data.sort(key=lambda x:x[16])
   minele=data[0][16]
   maxele=data[len(data)-1][16]-minele

   for vector in data:
       vector[16]=vector[16]-minele

   data.sort(key=lambda x:x[0]) #sort the nodes, while keeping time order
```

Then, I produced CSV files for every node in the clusters, with two columns, one for error type (Software, Hardware, Human, Network, Facility and Undetermined errors), and one for timestamp.:

```
for integer in range(len(data)):
       if data[integer][0] != current_number:
           curr_name="Node "+str(current_number)+" in file "+file
           f=open('%s.txt' % curr_name, 'w+')
           f.write("Type,Time")
           f.write("\n")

           for row in range(old_index, new_index-1):
               cstr=""
               for column in range (19,25):
                   if data[row][column] != "":
                       if column==19:
                           cstr="Facility Error,"
                       if column==20:
                           cstr="Hardware Error,"
                       if column==21:
                           cstr="Human Error,"
                       if column==22:
                           cstr="Network Error,"
                       if column==23:
                           cstr="Undetermined Error,"
                       if column==24:
                           cstr="Software Error,"
               if cstr=="":
                   continue
```

```
                cstr+=str(data[row][16])
                f.write(cstr)
                f.write("\n")
            c_name=curr_name+".txt"
            f=open(c_name,'r')
            curr_name=curr_name+".csv"
            csvfile=open(curr_name,'w+')
            for line in f:
                csvfile.write(line)
            old_index=new_index+1
            new_index=new_index+1
            current_number=data[integer][0]
        else :
            new_index=new_index+1
```

From there, I found the average amount of each error that happened within a certain "chunk" size, which I chose to be a three second chunk, as it worked best. I took these average amounts for each error, relabeled them errors 1-6 and wrote a CSV file for each error, with its average error rate within the chunk size, and respective timestamp for the last error within the chunk (the following shows a file being created for Hardware Errors):

```
    for t in time:
        if t>(start+chunk):
            cArr=taker(data_type,tiredOld,tiredCurr)
            tiredOld=tiredCurr
            one.append((t,cArr.count(1)))
            two.append((t,cArr.count(2)))
            three.append((t,cArr.count(3)))
            four.append((t,cArr.count(4)))
            five.append((t,cArr.count(5)))
            six.append((t,cArr.count(6)))
            start=chunk
            #now we have array of all errors in current chunk sized interval
        tiredCurr=tiredCurr+1

    with open('test.csv', 'w') as f:
        writer = csv.writer(f , lineterminator='\n')
        for Tuple in three:
            writer.writerow(Tuple)
```

Once I had the averages and timestamps, I was able to make my predictions using the ARIMA model in python through the statsmodels library:

```
X = series.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
predictedBuffer=[]
for t in range(len(test)):
    model = ARIMA(history, order=(2,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
```

```
     yhat = output[0]
     predictions.append(yhat)
     obs = test[t]
     history.append(obs)
     predictedBuffer.append((yhat,obs))
     print('predicted=%f, expected=%f' % (yhat, obs))
error = mean_squared_error(test, predictions)
```

Ultimately, the prediction outputted this:

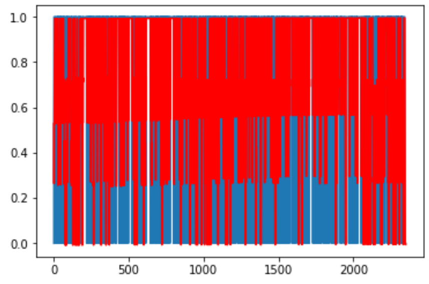


Figure 3: ARIMA prediction with prediction in red, and actual values in blue

The ARIMA model ended up giving me a 70.6% accurate prediction, with a Mean Squared Error of .235.

References

[1] Jeremy Curuksu: Forecasting financial time series with dynamic deep learning on AWS
`https://aws.amazon.com/blogs/machine-learning/forecasting-time-series-with-dynamic-deep-learning-on-aws/`

[2] New Knowledge
`https://www.newknowledge.com/articles/nk-labs-how-new-knowledge-uses-classic-time-series-models-to-predict-the-future/`

[3] Statsmodels Library
`https://pypi.org/project/statsmodels/`

[4] Robert Nau: Notes on nonseasonal ARIMA models,
Nau, Robert. Notes on Nonseasonal ARIMA Models.
`Http://People.duke.edu/~Rnau/Notes_on_nonseasonal_ARIMA_models--Robert_Nau.Pdf.`