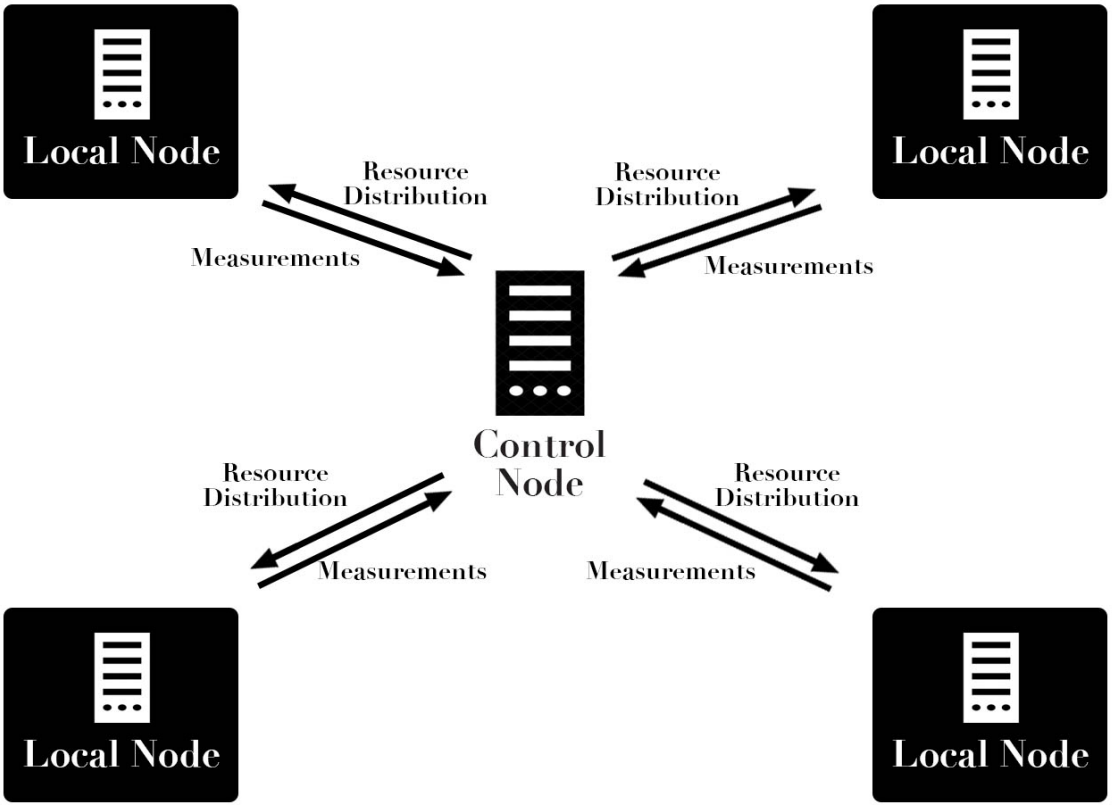
Imperial College London

Department of Electrical & Electronics Engineering

Final Year Project Report 2020



Project Title: Machine Learning to Track Cloud Computing

Student: Yee Hong, Low

CID: 01202613

Course: EE4

Project Supervisor: Prof. Kin K. Leung

Second Marker: Prof. Patrick A. Naylor

Contents

Acknowledgements

I would like to thank my project supervisor, Professor Kin K. Leung for his continuous guidance and support throughout this entire project. I would like to also thank Tiffany Tuor for her help in completing this project.

Abstract

An accurate yet efficient cloud resource forecasting system is crucial for the management of large scale computing systems. The paper (Tuor et al. 2019) proposed a novel solution for the online forecasting of cloud resource utilisation using a three-step process of adaptive transmission, evolutionary clustering and temporal forecasting. This project aims to outline potential improvements which can be made to the last two steps of said framework. Following analysis on the data set, we propose a correlation-based online forecasting framework which exploits the linear relationship between strongly correlated time-series to forecast future cloud resources utilisation. The newly proposed correlation-based framework comprises of four parts: time-series clustering, computation of a general model, temporal forecasting and forecast scaling. The framework is benchmarked against other existing frameworks including the brute force framework, the naive framework and a simplified version of the framework proposed in (Tuor et al. 2019). Initial tests showed that the correlation-based framework managed to capture the dynamics of the the time-series to be forecasted. However, the framework underperforms the naive method in a sense that that its accuracy and precision do not justify its computational complexity. It was concluded that a correlation-based framework is feasible but the proposed framework requires additional fine-tuning. Future work includes improving the proposed framework by implementing additional features. After which, the framework needs to be benchmarked against the full version of the framework proposed in (Tuor et al. 2019).

1 Introduction

Modern cloud computing systems often consist of thousands of individual machines. Resource management systems are hence needed to effectively manage such large distributed systems. They are crucial to prevent resource over-provisioning (allocating too much resources) and resource under-provisioning (allocating too little resources), in which the former causes wastage and high operational cost while the latter causes deterioration in user experience.

These resource management systems typically comprise of two parts: a **forecasting model** which forecasts future resource utilisation and a **resource allocation model** which uses the results from the forecasting model to optimally allocate resource to each individual machine. In particular, measurements of resource utilisation at each individual machine (local node) is transmitted to a central controller (control node). Using the information, the central node forecasts future resource utilisation at each local node. The control node then uses the forecasted values to allocate tasks to the local nodes according to available resource at each local node.

Coming up with an efficient resource management system for such large-scale distributed systems poses a few key problems:

1. A system based solely on feedback would be too slow to optimally manage the system. Hence, the current and future available resource at each individual machine needs to be inferred so that resources can be allocated in real time

2. The predictive model for forecasting usually has high computational complexity, making it computationally expensive to run a predictive model for each individual machine

3. To transmit all measurements from the local node to the central node would consume too much bandwidth

The paper *Online Collection and Forecasting of Resource Utilization in Large-Scale Distributed Systems* (Tuor et al. 2019) proposed a novel framework for the online forecasting of cloud resource utilisation. The framework comprises of three parts:

1. **Adaptive transmission** of measurements from the local nodes to the control node to reduce the bandwidth consumption of the system

2. **Evolutionary clustering** of local nodes to reduce the dimensionality of the entire optimisation problem hence reducing the computational complexity of the required forecasting model

3. **Temporal forecasting** to infer future resource utilisation in each machine

This project starts with analysis on the data set to identify key properties which can be exploited to design the forecasting model. Following the analysis, we then propose and implement a forecasting framework to predict future cloud resource utilisation. The framework is then validated and benchmarked against a naive framework, a brute force framework and other existing frameworks.

2 Related Work

2.1 Forecasting of Cloud Resource Utilisation

Most existing work which utilises forecasting models for cloud resource management focuses on forecasting aggregated workloads or demands using a single time series (Shen & Chen 2018),(Cai et al. 2018),(Grechanik et al. 2016),(Nikravesh et al. 2015). This technique, though useful for forecasting future demands, is unable to capture the dynamics of resource utilisation at each machine. In this project, we focus on forecasting the resource utilisation at each individual machine which is a much complex problem given that each individual machine gives a different time-series.

2.2 *Online Collection and Forecasting of Resource Utilization in Large-Scale Distributed Systems*

Given that this report heavily references the paper *Online Collection and Forecasting of Resource Utilization in Large-Scale Distributed Systems* (Tuor et al. 2019), this section will be dedicated to give an overview of the framework proposed in the paper. From this point in the report, the framework proposed in the mentioned paper will be called "the Tuor framework".

The Tuor framework consists of three parts:

1. **Adaptive Transmission**
   The framework starts by deploying an adaptive transmission algorithm to reduce the bandwidth consumption used in the transmission of measurements from local nodes to control nodes. This is done with a cost function in which the local node will only transmit the measurements if the error in not transmitting the measurement surpasses a user defined threshold. If the control node do not receive the transmission from a given local node at a given time-step, it will feed-forward the measurement from the previous time-step.

2. **Evolutionary Clustering**
   The framework then performs evolutionary clustering on all the machines to reduce the dimensionality of the overall optimisation problem. Evolutionary clustering (Chakrabarti et al. 2006) is a technique of clustering time-series over time. The intuition behind evolutionary clustering is that instead of clustering the time-series independently at each time-step, there exists a memory cost where time-series are biased to remain in the same cluster as they are in previous time-steps. The cost function for evolutionary clustering is given as:

   $$sq(C_t, M_t) - cp \cdot hc(C_{t-1}, C_t) \tag{1}$$

   where

   sq = the snapshot quality, denoting how faithful the clustering is to the actual data
   hc = the history cost which biases time-series to stay in the same cluster as previous time-steps
   C_t = a bipartite graph outlining clustering of points at time-step t
   M_t = distance matrix between each time-series to be clustered
   cp = the change parameter which is a user defined control parameter to weigh sq and hc terms

   The evolutionary clustering algorithm needs to find a C_t for each time t to optimise (1). The algorithm gives a different clustering at each time-step. Note that when $cp = 0$, (1) gives the cost function of a regular clustering algorithm (i.e. the time-series are clustered independently at each time-step).

3. **Temporal Forecasting**
   For each cluster, the framework defines a centroid, $c_{j,t}$, which acts as a generalisation model for all the time-series in each cluster:

   $$c_{j,t} = \frac{1}{|C_{j,t}|} \sum_{i \in C_{j,t}} z_{i,t}$$

   where

$$
\begin{aligned}
C_{j,t} &= \text{the set of machines in cluster j at time t} \\
|\cdot| &= \text{the cardinality (size) of the set} \\
z_{i,t} &= \text{the time-series given by each node i}
\end{aligned}
$$

Temporal forecasting is then performed on the centroid either using the RNN or the ARIMA forecasting model to get the forecasted value $\hat{c}_{i,t+1}$. After that, the forecasted value for the centroid is scaled to fit the other time-series in the cluster using a rolling offset term, s.

$$\hat{z}_{t+1} = \hat{c}_{j,t+1} + s_{i,t+1}$$

where

$$
\begin{aligned}
s_{i,t+1} &= \frac{1}{M+1} \sum_{m=0}^{M} \alpha_{t-m}(z_{t-m} - c_{j,t-m}) \\
\alpha_{t-m} &\in (0,1] \text{is a scaling coefficient} \\
M &= \text{rolling offset window}
\end{aligned}
$$

The intuition for the scaling is that we use mean of the past distance between the centroid to the individual time-series to offset the current forecast. The framework hence end up with a forecast for each machine at each time-step. In this project, we will propose and implement potential improvements to this framework.

2.3 Time-series Clustering

Time-series clustering is a widely discussed topic. (Aghabozorgi et al. 2015), (Kotsakos et al. 2014), (Roelofsen 2018) discuss the properties as well as the advantages/disadvantages of various clustering algorithm and distance measure used in time-series clustering.

The technique of using a general model to generalise a group of clustered time-series is seen used in the energy forecasting industry. (Shchetinin 2019) uses K-means to cluster consumers with different energy consumption together and uses generalisation models to perform forecasting on each of the clusters. While (CAI et al. 2019) uses particle swarm optimisation K-means algorithm to cluster energy consumption of residential building.

In this paper, we will consider the advantages and disadvantages of different time-series forecasting techniques discussed in the mentioned papers to formulate a strategy to cluster time-series of cloud resource utilisation.

3 Requirements Capture

The main aim of this project is to come up with a computationally inexpensive framework to forecast cloud resource utilisation. Analysis will first be performed on the data set to identify key properties which can be exploited in the forecasting framework. The framework proposed in this project will need to be benchmarked against other existing solutions for forecasting cloud resource utilisation. Ideally, the proposed framework will be computationally cheaper yet more accurate compared to other existing frameworks.

4 Data Set

The data set used in this project is the Google cluster usage trace (version 2) (Reiss et al. 2014). It contains job/task utilisation information of 12,476 machines. For each machine, measurements of resource utilisation is sampled at 5 minute interval for approximately 29 days. Essentially, each machine gives one time-series (of 8351 samples) for each type of resource (CPU utilisation and memory). The raw data is pre-processed to a normalized CPU and memory utilisation for each individual machine. All the measurements are normalised to have a value within the range of 0 to 1 inclusive.

In this report, we only consider the CPU utilisation of the machines. Due to constraints on computational power, the analysis on all 12,476 machines may be impractical for some computationally expensive analysis. Some analysis are hence done on a set of randomly chosen machines.

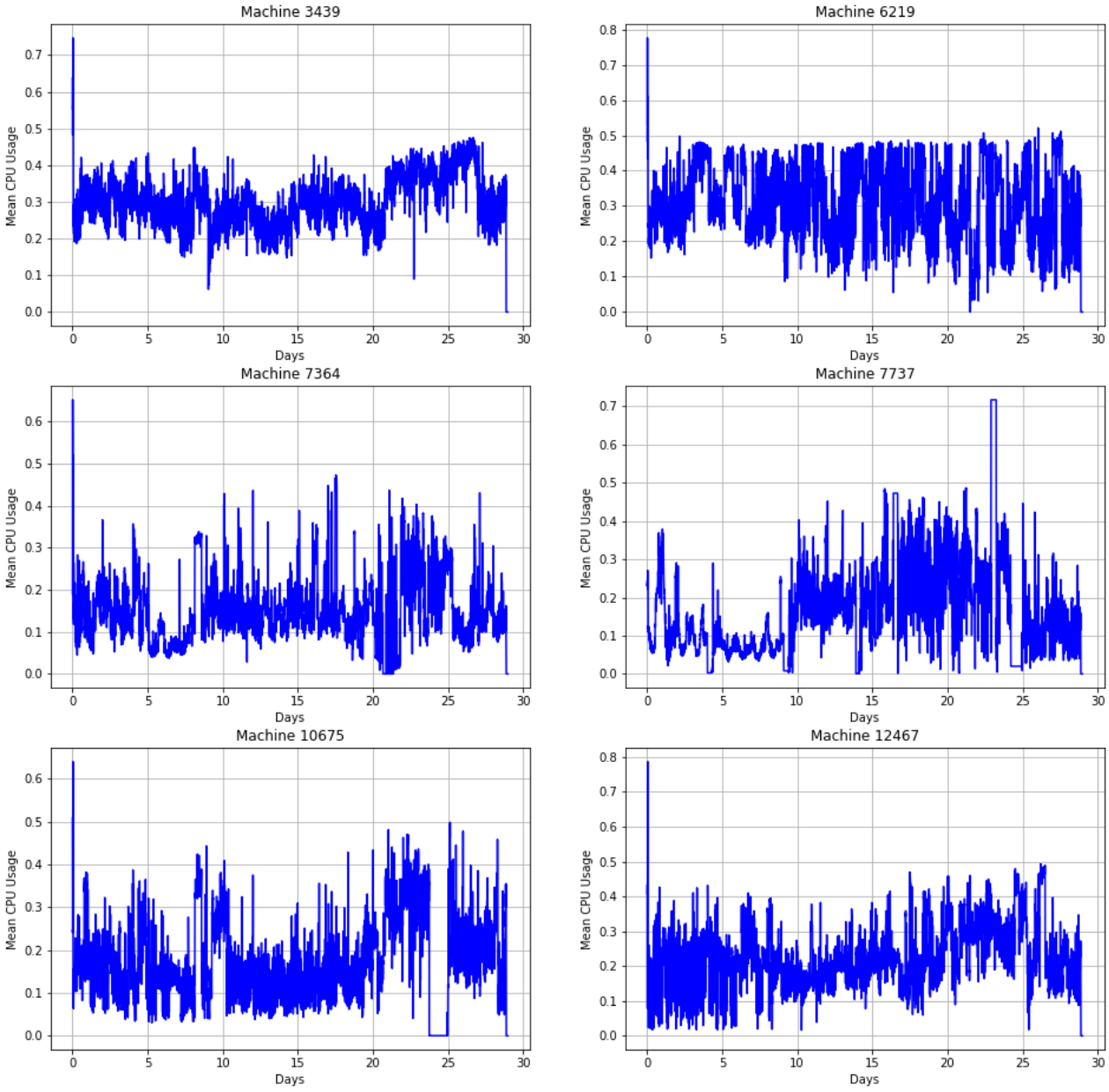


Figure 1: Mean CPU utilisation of six randomly chosen machines.

5 Definitions

In this section, we define some key words and notations which will be used throughout the entire report.

5.1 Timeseries vs Machine

Since in the dataset, the measurements of each machine is sampled at 5 minutes interval for 29 days, each machine will essentially give a time-series. From this point of the report on-wards, a time-series will be referred to as the time-series given by each machine. The notation $\mathbf{x}_i$ will be use to refer to the time-series given by machine indexed i while $\mathbf{x}$ will refer to a generic time-series given by a machine of any index.

5.2 Indexing of Samples

Since the code for this project is written entirely in Python, we will follow the indexing convention in Python. The notation is given as follows: $x[i]$ denotes sample i of the time-series. Note that the first sample of all time-series has an index of 0.

The notation for a segment of $\mathbf{x}$ is given as:

$$\mathbf{x}[i:j] := \text{sample } i \text{ to sample } j-1 \text{ (inclusive) of the time-series } \mathbf{x}$$

This makes it easy to specify a segment of a time-series of any given length, N using the notation:

$$\mathbf{x}[i:i+N]$$

Negative index denotes counting from the back i.e.

$$\mathbf{x}[-i]$$

denotes the last i^{th} sample.

Open ended segments denotes the rest of the time-series i.e.

$$\mathbf{x}[i:] \tag{2}$$

$$\mathbf{x}[:j] \tag{3}$$

where (2) denotes the segment from the i^{th} sample to the last sample (inclusive), while (3) refers to the first sample to the $(j-1)^{th}$ sample (inclusive).

5.3 Autocorrelation vs Cross-correlation

Autocorrelation refers to the correlation between a time-series and a lagged version of itself:

$$corr(\mathbf{x}_i[n:n+N], \mathbf{x}_i[n+lag:n+lag+N])$$

Cross-correlation refers to the correlation between different time-series:

$$corr(\mathbf{x}_i, \mathbf{x}_j)$$

5.4 Shift vs Lag

In this report, the term "shift" will be used when comparing segments of two different time-series. For example when comparing segments of two different time-series, $\mathbf{x}_i$ and $\mathbf{x}_j$, we denote the two segments as:

$$\mathbf{x}_i[n+shift:n+N+shift] \text{ vs } \mathbf{x}_j[n+shift:n+N+shift]$$

where N denotes the length of the segment.

The term "lag", will be reserved for when different segments of the same time-series are compared. For example, when comparing different segments of the time-series $\mathbf{x}_i$:

$$\mathbf{x}_i[n:n+N] \text{ vs } \mathbf{x}_i[n+lag:n+N+lag]$$

5.5 Notations

Here, we compile all the notations which will be used throughout the entire report.

- $\mathbf{y}_k$ will be used to refer to the general model for a cluster of index k while $\mathbf{y}$ refers to a generic general model.
- $\mathbf{C}_k$ will be used to refer to the cluster of index k. Note that $\mathbf{C}_k$ is a set consisting of the index of the machines in the cluster.
- The *hat* notation for example $\hat{x}[n]$ refers to the forecasted $x[n]$ value.
- $e_i[n]$ refers to the prediction error of machine i at sample i i.e. $e_i[n] = x_i[n]$ - $\hat{x}_i[n]$
- K refers to the total number of clusters.
- w refers to the rolling correlation window size.
- I refers to the total number of selected time-series.
- $\mathbf{M}$ refers to the correlation matrix. $M_{i,j}$ refers to element of index (i, j) in $\mathbf{M}$ and it gives $corr(\mathbf{x}_i, \mathbf{x}_j)$.
- $\mathbf{m}[n]$ refers to the rolling correlation matrix at time-step n. $m_{i,j}[n]$ refers to element of index (i, j) in $\mathbf{m}[n]$ and it gives $corr(\mathbf{x}_i[n+1-w:n+1], \mathbf{x}_j[n+1-w:n+1])$. $\mathbf{m}_k[n]$ refers to the rolling correlation sub-matrix corresponding the cluster k.

6 Metrics

The metric used for measuring distance between different time-series is crucial in defining their similarity. Below are some of the metric considered in this report:

6.1 Pearson Correlation

The Pearson correlation coefficient of two random variables, x and y, is define as:

$$corr(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4}$$

Pearson correlation looks at the linear association between two random variables of equal length and has a range between 1 and -1 inclusive. Two random variables, $a(t)$ and $b(t)$, can be modeled as (5) or (6) if the Pearson correlation coefficient is close to 1 or -1 respectively.

$$a(t) = s \cdot b(t) + d \tag{5}$$

$$a(t) = -s \cdot b(t) + d \tag{6}$$

where the scaling factor, s is a positive constant and the displacement, d is a constant.

Heuristically, Pearson correlation is a measure of the strength of linear relationship between two time-series. In other words, it measures the similarity in the overall shape of two time-series by tracking if the two time-series increase or decrease at the same time. In this report, Pearson correlation will be referred to simply as "correlation".

6.2 Minkowski Distance

The Minkowski distance is the L_p-norm of the difference between two vectors of equal length. It is defined as:

$$d(x,y) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{1/p} \tag{7}$$

and it is the generalisation of the commonly used Euclidean distance ($p = 2$):

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{8}$$

and Manhattan distance ($p = 1$):

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i| \tag{9}$$

Minkowski distance is commonly used as the loss function of time-series predictive models. Note that, unlike the Pearson correlation, the Minkowski distance is a point based distance which exploits similarities at individual time-step between two time-series. The similarity in shape is disregarded.

For this report, we will only be using the the Euclidean distance and the Manhattan distance. Note that both the measures are sign invariant. The difference between the two measures is that the Euclidean distance emphasizes large differences while Manhattan distance treats all distances proportionally.

6.3 Mean Squared Error vs Mean Absolute Error

The mean absolute error (MAE) mean squared error and mean squared error (MSE) are the equivalent of using the Manhattan distance and Euclidean distance to calculate error respectively:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{10}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{11}$$

where $\hat{y}$ is the inferred (or forecasted) value and y is the actual value.

Both the MAE and the MSE are unbiased indicators of performance. The difference between the two measures is that MSE puts more weight on large errors. This means that the MSE is more useful when large errors are particularly undesirable which is the case for cloud resource forecasting. Hence, MSE will be our metric of choice to measure and compare the accuracy of different forecasting method.

6.4 Mean Percentage Error

The definitions of **Mean Absolute Percentage Error (MAPE)** and **Mean Squared Percentage Error (MSPE)** are given below:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{\hat{y}_i}\right| \tag{12}$$

$$MSPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{y}_i - y_i}{\hat{y}_i}\right)^2 \tag{13}$$

The mean percentage error is an error measure in relation to the actual value. It provides an intuition regarding the magnitude of the error compared to the magnitude of the actual value. This makes it a compelling measure to be used to assess the performance of any time-series predictive models. However, percentage based errors are biased indication of accuracy because they discount the error whenever the actual value is large and emphasize the error whenever the actual value is small. This skews the measure of error and results in a biased indication of performance. To avoid this problem, we will instead use MAE or MSE.

6.5 Computational Complexity

To measure computational complexity, we use "Time for One Prediction" (TOP) which is defined as:

$$TOP = \frac{\text{Total time needed to compute all predictions}}{\text{Number of Predictions}} \tag{14}$$

The higher the TOP, the higher the computational complexity. It is ideal for the TOP of any algorithm to be low. Also note that in this report, forecast and prediction refer to the same thing.

7 Analysis

Analysis performed on the data set can be categorised into three different categories:

- **CPU-Memory Analysis** where we look at the relationship between CPU utilisation and memory utilisation
- **Spatial Analysis** which looks at the relationship between different machines
- **Temporal Analysis** which looks at the relationship at different points/segments in time in the same machine

7.1 CPU Utilisation vs Memory Utilisation

Relying solely on past values of the time-series to make forecasts normally result in inaccurate forecasts especially when the forecasts are made further into the future. It is often more reliable to depend on parameters other than the past values of the time-series. Since we also have access to the memory utilisation of each of the machine, it would be reasonable to study the relationship between the two resources. In this section, we will analyse the relationship between CPU utilisation and memory utilisation to determine if it is feasible to forecast CPU utilisation based on memory utilisation. We begin by looking at the correlation between the CPU utilisation and memory utilisation of all machines: $corr(\mathbf{x}_{i,CPU}, \mathbf{x}_{i,memory})$.

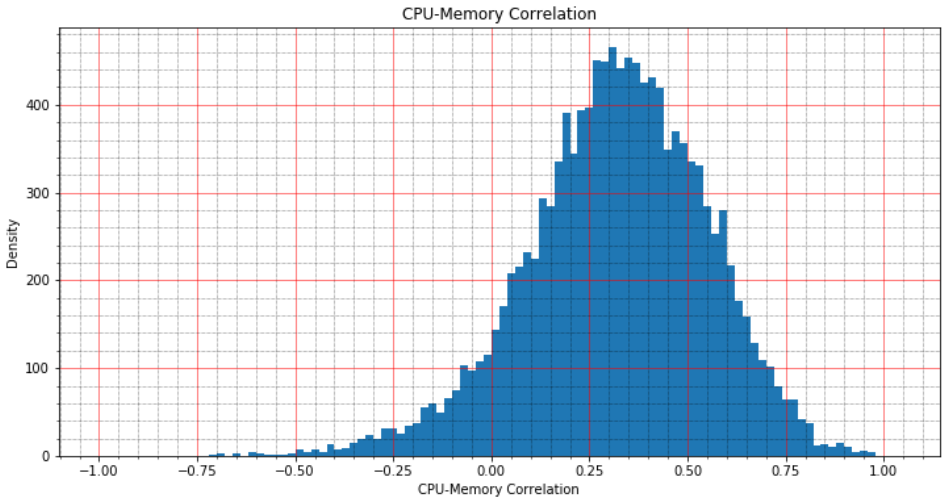


Figure 2: Distribution of correlation between CPU utilisation and memory utilisation for all machines.

It is observed that the CPU-memory correlation has a mean of 0.318 which is low. Only 465 in 12,476 (3.73%) machines have a CPU-memory correlation of higher than 0.7. Figure 3 shows the plot of CPU utilisation vs memory utilisation of 10 machines which has the highest CPU-memory correlation. It is observed that there is no clear relationship between the two variables.

Since there is no clear relationship between the two variables and that the two variables do not have strong correlation in general, we conclude that it is not feasible to forecast CPU utilisation with memory utilisation.

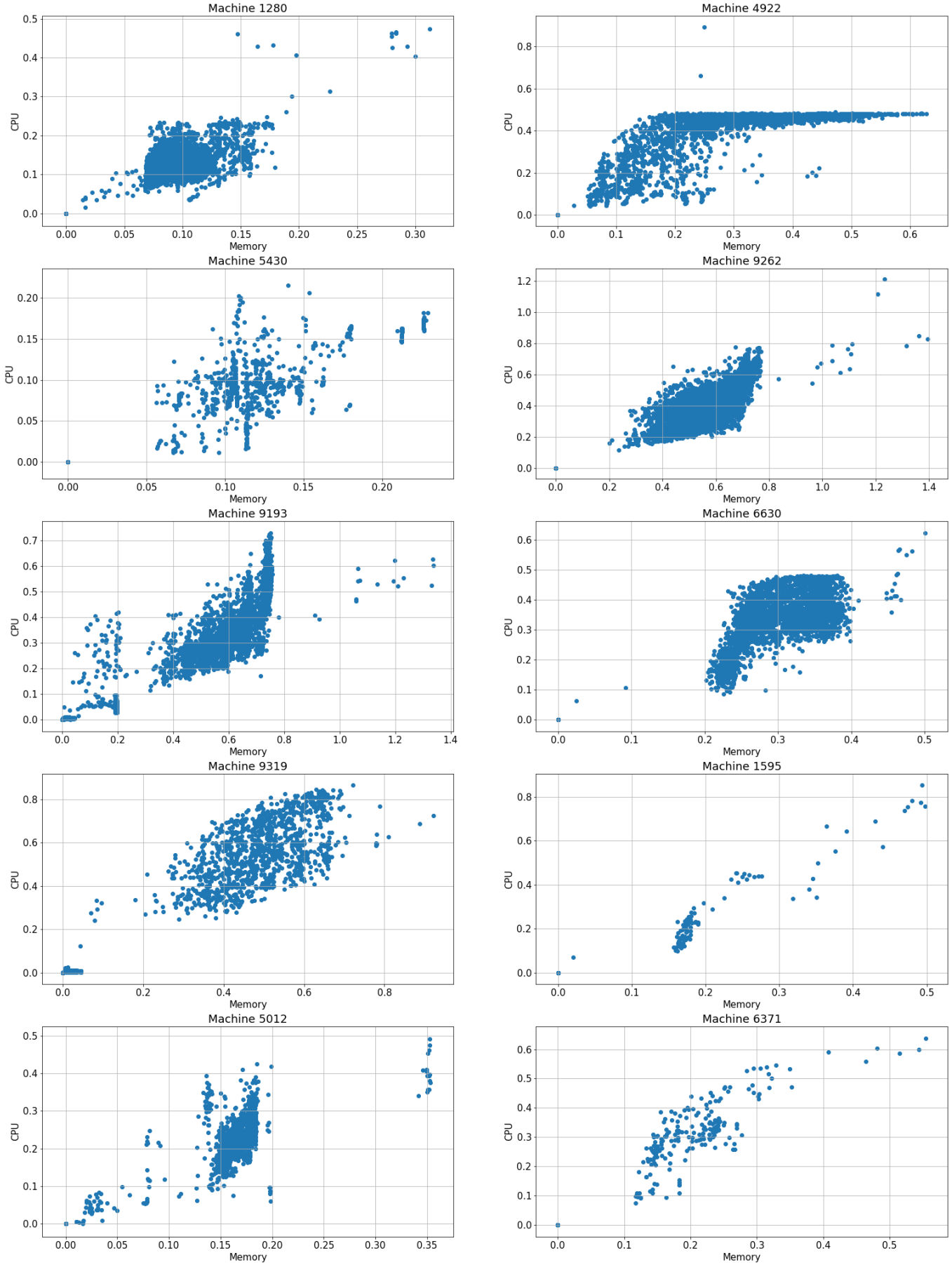


Figure 3: CPU utilisation vs memory utilisation of 10 machines with the highest CPU-memory correlation.

7.2 Temporal Analysis

The temporal analysis is mainly necessary to determine some design choices for the temporal forecasting model. Particularly, it is necessary to determine how much of the past data can be exploited to forecast future data i.e. the amount of memory required in the temporal forecasting model.

7.2.1 Seasonal Decomposition

For initial analysis, we attempt to decompose the time-series into three components: trend, seasonal and residue. The trend gives the overall change in the time-series over time. The seasonal component gives the periodic component of the time-series and it is the component in which is the most predictable out of the three components. The residue gives the random component of the time-series. The residue behave similarly to random noise. The decomposition performed is additive i.e.

$$\text{original time-series} = trend + seasonal + residue$$

The trend can be obtained by performing a moving average on the time-series. We choose the window of the moving average filter to be one day and one week. This is because it is hypothesized CPU utilisation exhibits daily or weekly periodicity, given that it should be related to internet usage. The individual components of the time-series can be seen in Figure 4 and Figure 5 where the moving average window of the size one day and one week are used respectively. The average of the mean and standard deviation of the components of all the 12,476 machines are computed and are presented in Table 1 and Table 2.

Component	Mean	Standard Deviation
Trend	0.188349	0.042838
Seasonal	0.000001	0.020159
Residue	0.000189	0.056555

Table 1: Mean and standard deviation of each component. Moving average window size = one day.

Component	Mean	Standard Deviation
Trend	0.188242	0.026896
Seasonal	0.000351	0.040223
Residue	0.001947	0.054064

Table 2: Mean and standard deviation of each component. Moving average window size = one week.

For us to be able to use the decomposition of the time-series to make long-term predictions, the seasonal component needs to be strong compared to the other components. However, it is observed that generally the seasonal component of the time-series is weak (low mean and standard deviation). The residua generally has a high standard deviation which means that the time-series is generally random in nature. These evidence leads us to conclude that decomposing the time-series into different components to forecast them separately is not a feasible strategy. It also suggests that it is not feasible for us to exploit the periodicity of the time-series to make forecasts.

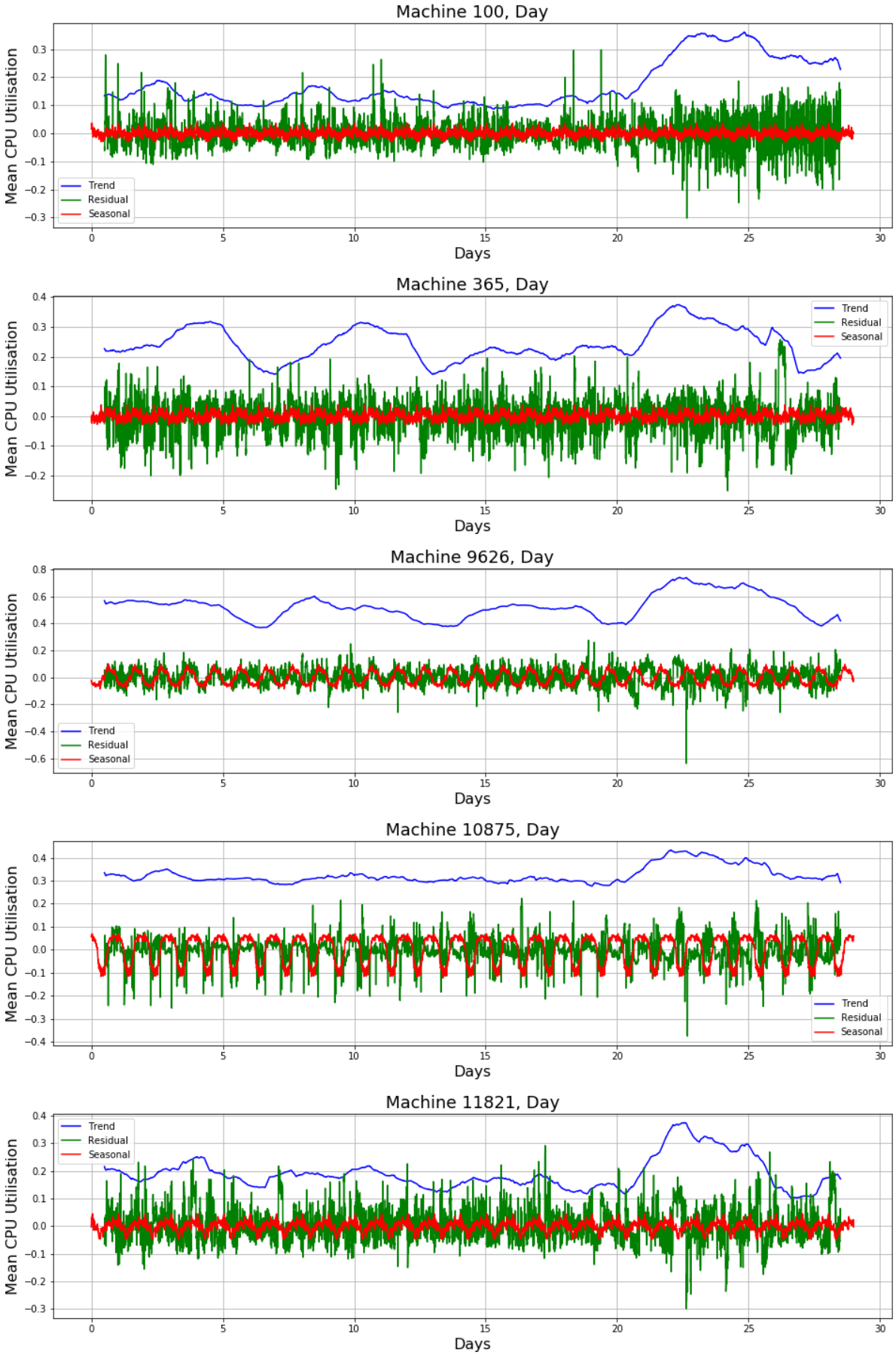


Figure 4: Decomposition of five machines into trend, seasonal and residue. Moving average window size = one day.

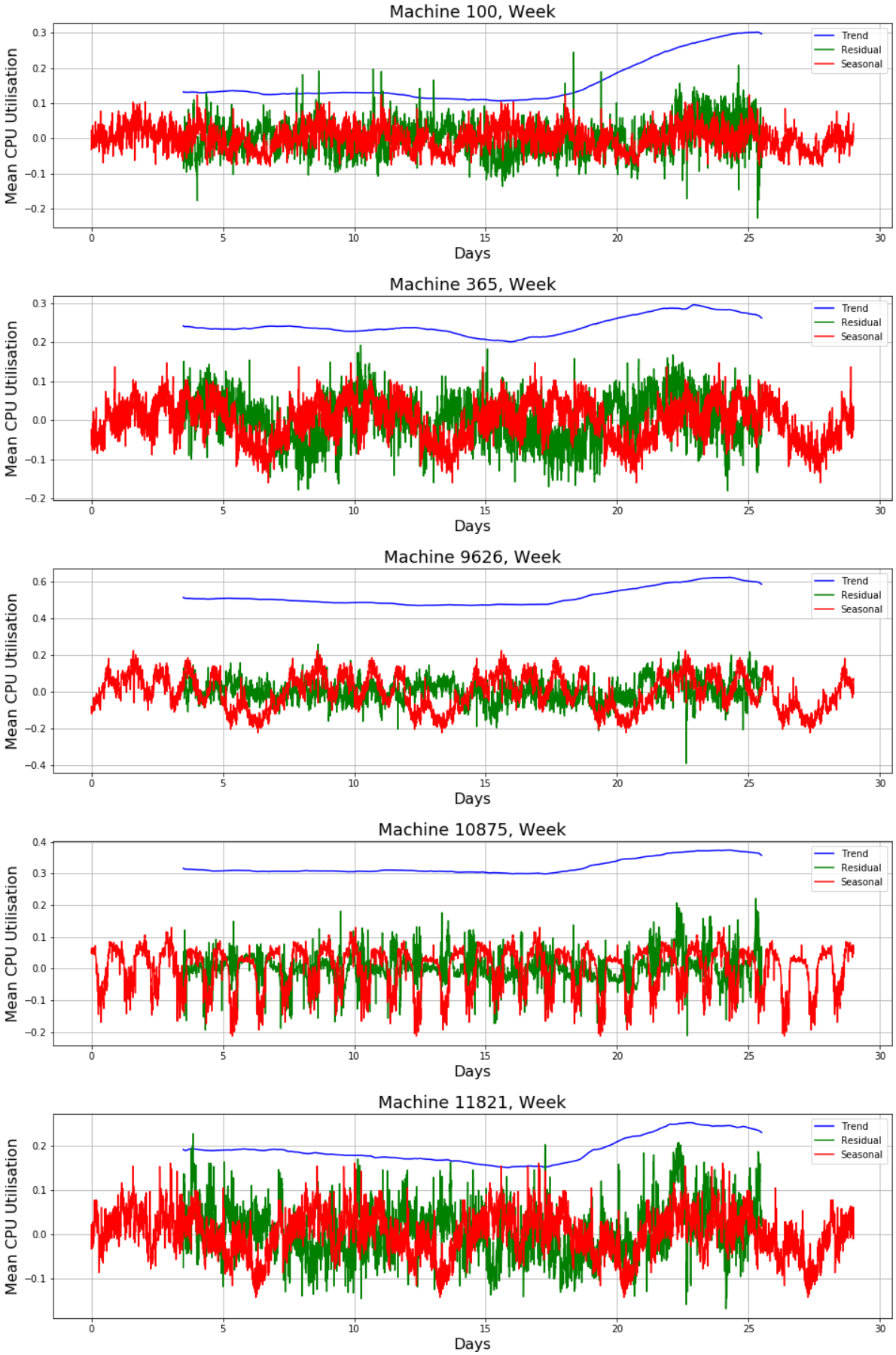


Figure 5: Decomposition of five machines into trend, seasonal and residue. Moving average window size = one week.

7.2.2 Stationarity

A time-series is stationary if there is no systematic change in its mean (no trend), if there is no systematic change in its variance, and if strictly periodic variations have been removed (Chatfield 2004). Most time-series analytic procedures require the data to be stationary. If the data is not stationary, transformations or differencing can be used to make the data stationary. This is extensively used when ARIMA models are used for forecasting.

To test the stationarity of time-series, we use the Augmented Dickey-Fuller test where the null hypothesis is that the time-series has a unit-root. The null hypothesis needs to be rejected to infer that the time-series is stationary.

The test is performed on 240 randomly chosen time-series. Only one out of the 240 time-series exhibit non-stationarity showing that about 0.4% of the time-series exhibits non-stationarity. ARIMA forecasting models is hence suitable to be used to forecast the time-series in the given data set.

7.2.3 Full Autocorrelation

For preliminary autocorrelation analysis, the entire length (29 days) of each of the machines is used as the window. The primary aim of this analysis is to look for periodicity in the data. It is hypothesized that most of the machines will exhibit daily or weekly periodicity as cloud resource utilisation should be related to internet usage which is periodic.

Analysis is done on 100 randomly chosen machines due to computational constraints. It is observed that out of the 100 machines analysed, only two machines showed strong daily periodic correlation. An example of this can be seen in machine 10875 in Figure 7. Machine 9626 is the only machine which showed both daily and weekly periodic correlation. The other 97 machines exhibits the same property: a steep decrease initially when the lag is small, followed by a gradual decrease. After which the autocorrelation varies around the range of 0.2 to -0.2. Machines 11821 and 100 are examples of a typical autocorrelation for the rest of the machines.

Note that the spike in correlation as the lag gets close to 29 days is due to the edging effect and should be ignored. It is also observed that the autocorrelation of each machine is always strong when the lag is small. This can be clearly seen in the zoomed in Figure 8. The distribution of autocorrelation of 100 machines for small amounts of lag shown in Figure 6 further corroborates this claim.

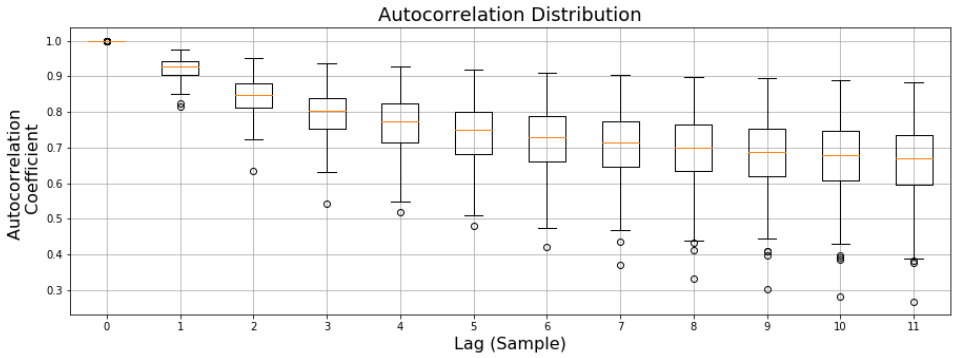


Figure 6: Distribution of the autocorrelation for 100 different machines.

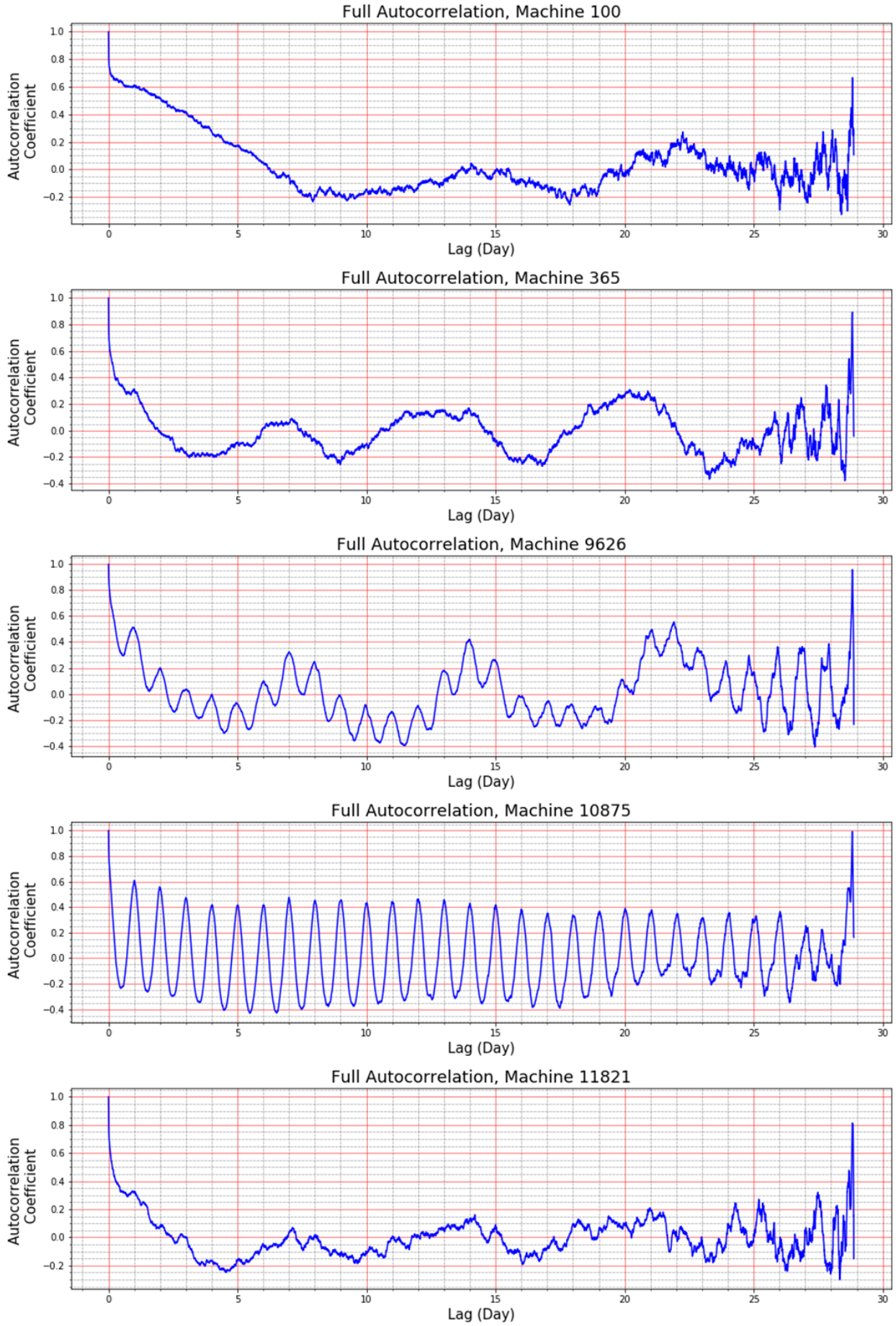


Figure 7: Full autocorrelation of five machines.

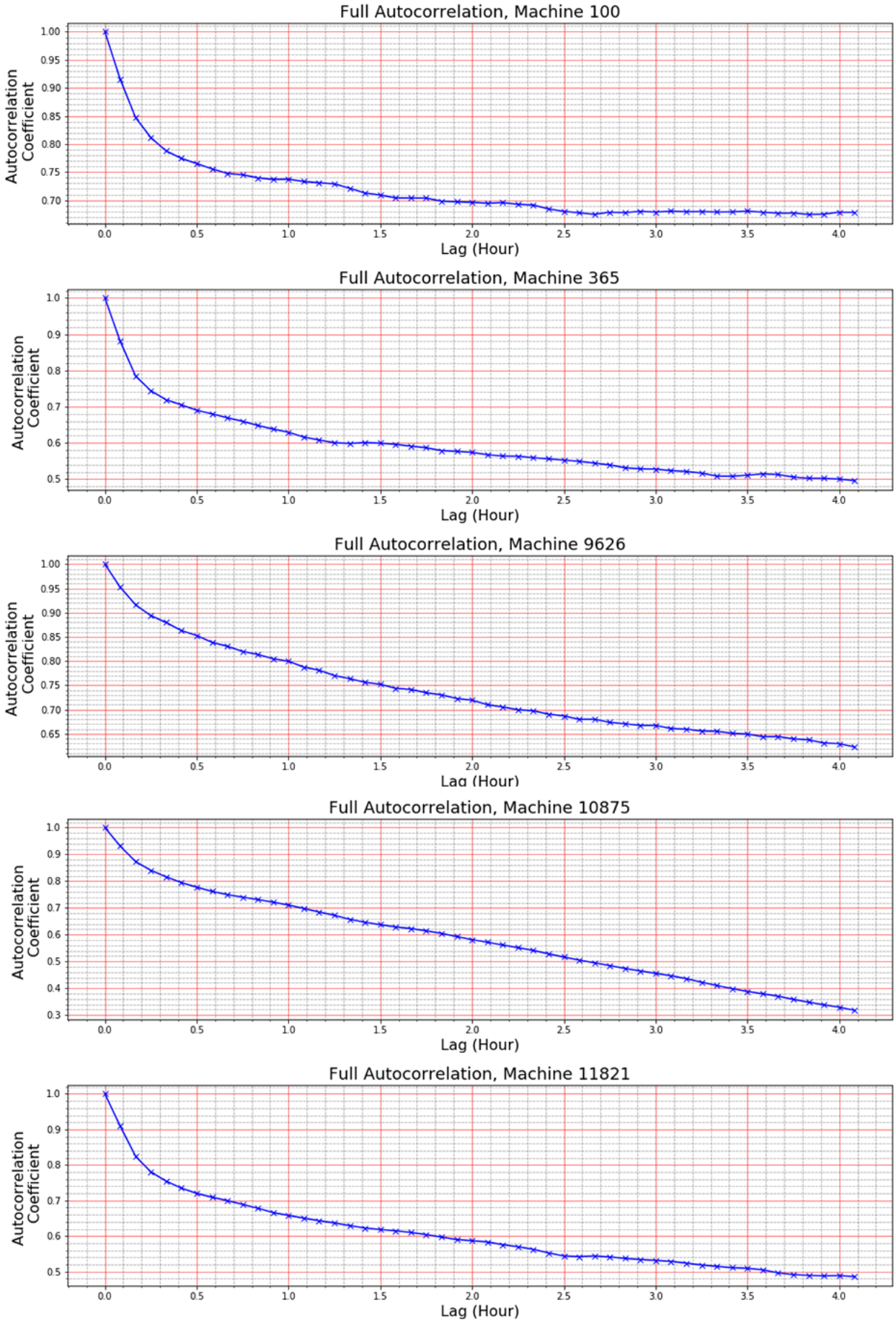


Figure 8: Full autocorrelation of five machines (zoomed).

7.2.4 Correlation between Adjacent Days

For this analysis, we look at the correlation of time-series between adjacent days. An example of this analysis is shown in Figure 9. The purpose of this analysis is to identify any further daily correlation present in each time-series which fails to show in the previous analysis. It is a way for us to check if it is feasible to break each time-series into groups of 24-hour segments and make 24-hour in advance forecast based on past segments. The mathematical formulation of this analysis is given below.

$$\text{correlation between adjacent days}[n] = corr(\mathbf{x}[n : n + 288], \mathbf{x}[n + 288 : n + 288 + 288])$$

Note that 288 samples equates to one day.

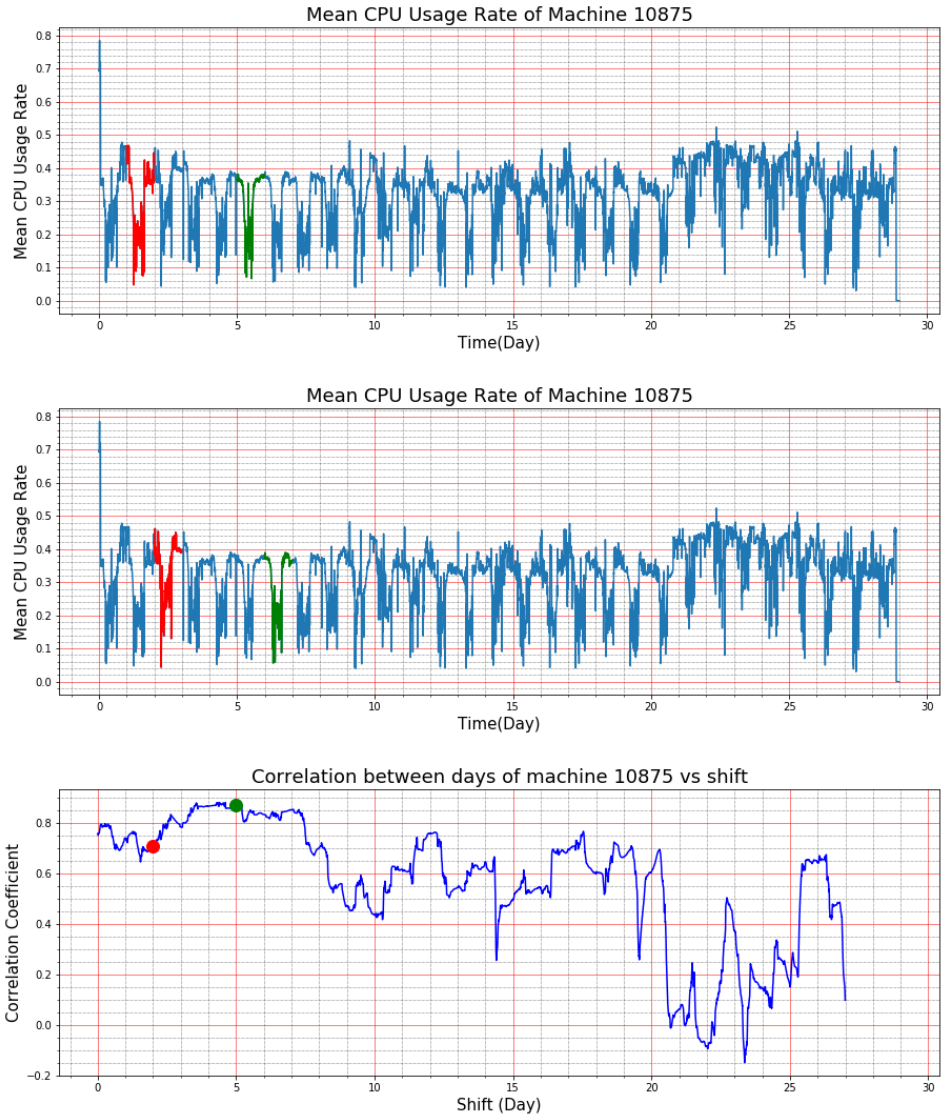


Figure 9: Correlation between adjacent days of machine 10875. The red dot and green dot in the bottom graph are the correlation coefficient for the time segments coloured in red and green respectively in the top two graphs.

This analysis is performed on 100 randomly chosen machines. Five of the 100 plots are shown in Figure 10. It is observed that the correlation between adjacent days for all machines fluctuates rapidly even in machine 10875, which showed daily periodicity in the full autocorrelation analysis. This suggests that it is not feasible to use past 24-hour segments to make 24-hour in advance forecasts.

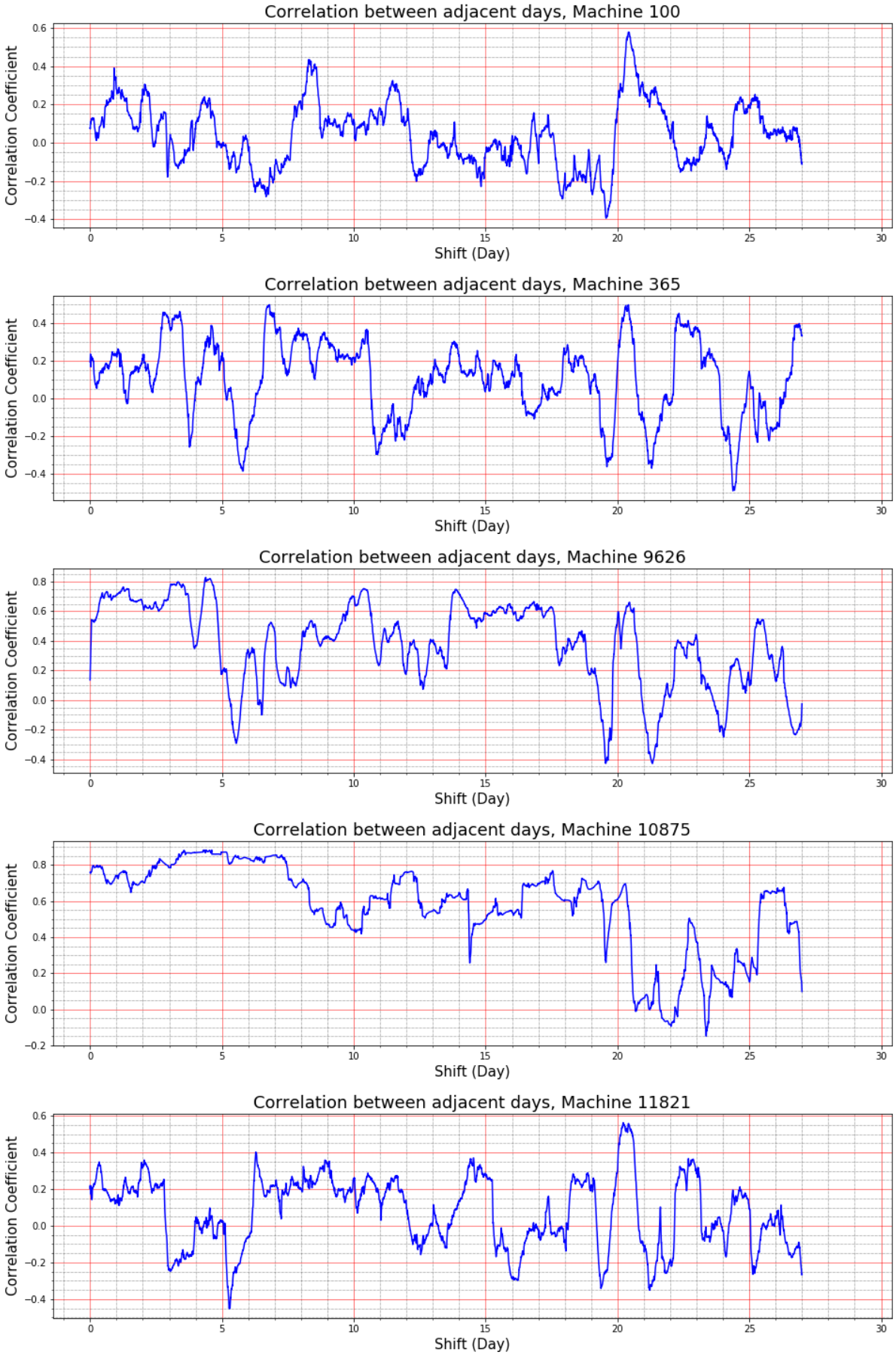


Figure 10: Correlation between adjacent days of five machines.

7.2.5 Correlation between Adjacent Time-step

For this analysis, we observe the correlation between segments of time-series of adjacent time-step using different window size. The mathematical formulation of this is given as:

$$\text{correlation between adjacent time-step}[n] = corr(\mathbf{x}[n : n + w], \mathbf{x}[n + 1 : n + w + 1])$$

where w is the window size. The purpose of this analysis is to observe how the correlation between adjacent time-step vary with time. It also provides us with insights on how the size of the window affect the correlation between adjacent time-step.

From Figure 11, it is seen that as the window size increases, the more the correlation between adjacent time-step fluctuates. This implies that if a small window is used, the forecasting model used needs to be able to quickly adapt to the rapidly fluctuating data i.e. the model needs to be able to update quickly.

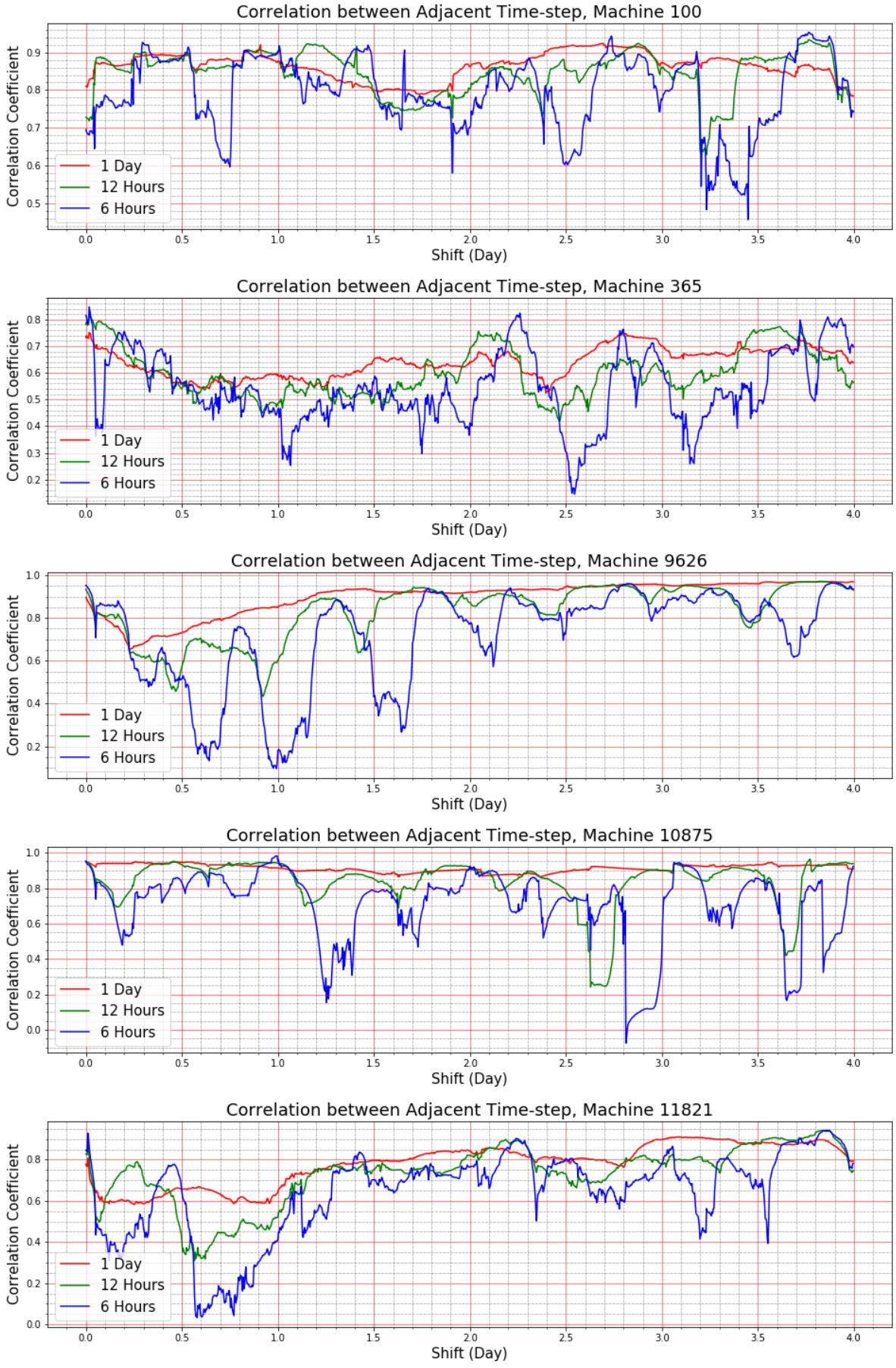


Figure 11: Correlation between adjacent time-step of five machines.

7.2.6 Spectral Analysis

In this section, we perform spectral analysis on the time-series. To do this, we perform Fast Fourier Transform (FFT) on the time-series. Five examples of this analysis is plotted in Figure 15. It is observed that for all of the time-series, their frequency components are saturated in the very low and very high frequency region.

We then take the mean of the FFT of all the 12,476 time-series. This is plotted in Figure 12. From this analysis, we observe that the frequency components of all the time-series are saturated in the high frequency and low frequency region. The low frequency component implies that the time-series is periodic about its entire length (or close to its entire length) while the high frequency component implies that the time-series is periodic about a few samples. This does not give us any useful information. Hence, it is not feasible for us to exploit the frequency domain to for forecasting.

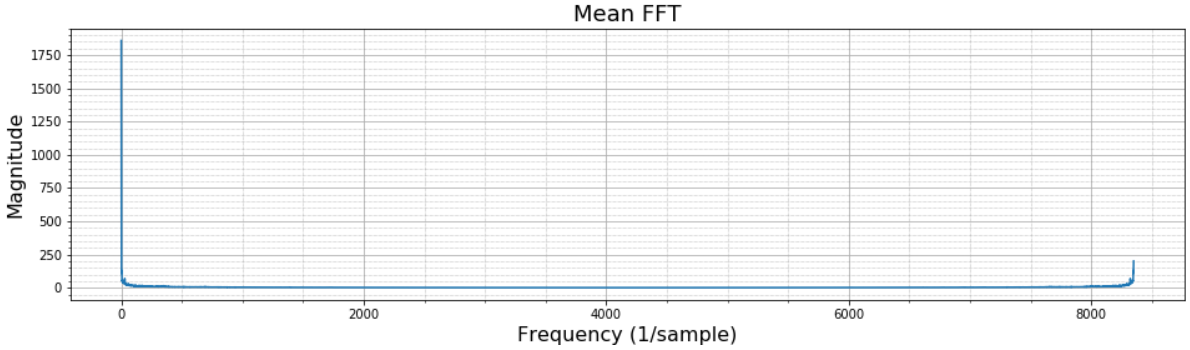


Figure 12: Mean of FFT of all machines.

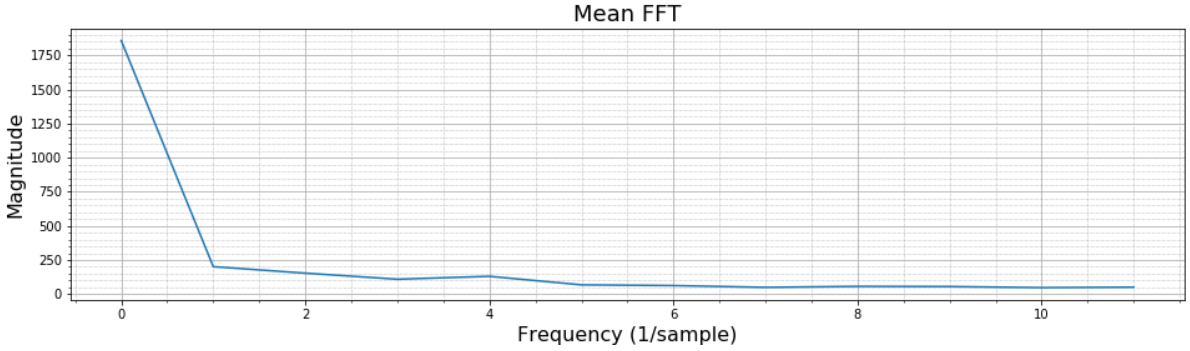


Figure 13: Mean of FFT of all machines, first 12 samples.

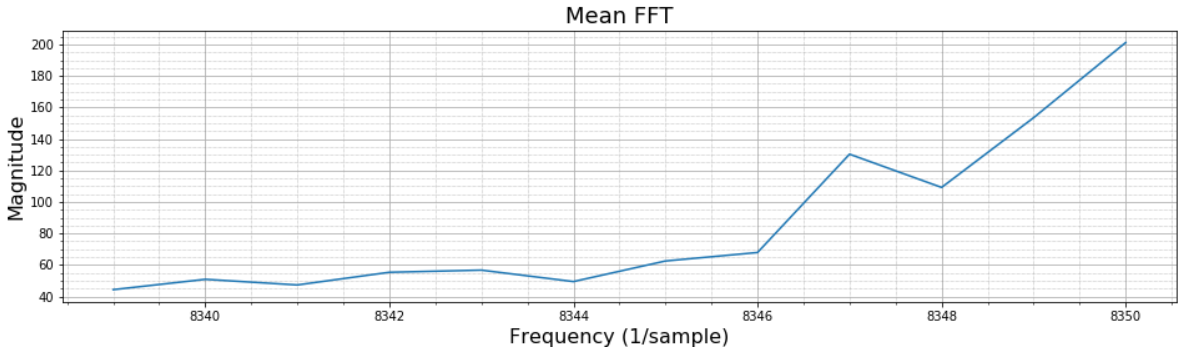


Figure 14: Mean of FFT of all machines, last 12 samples.

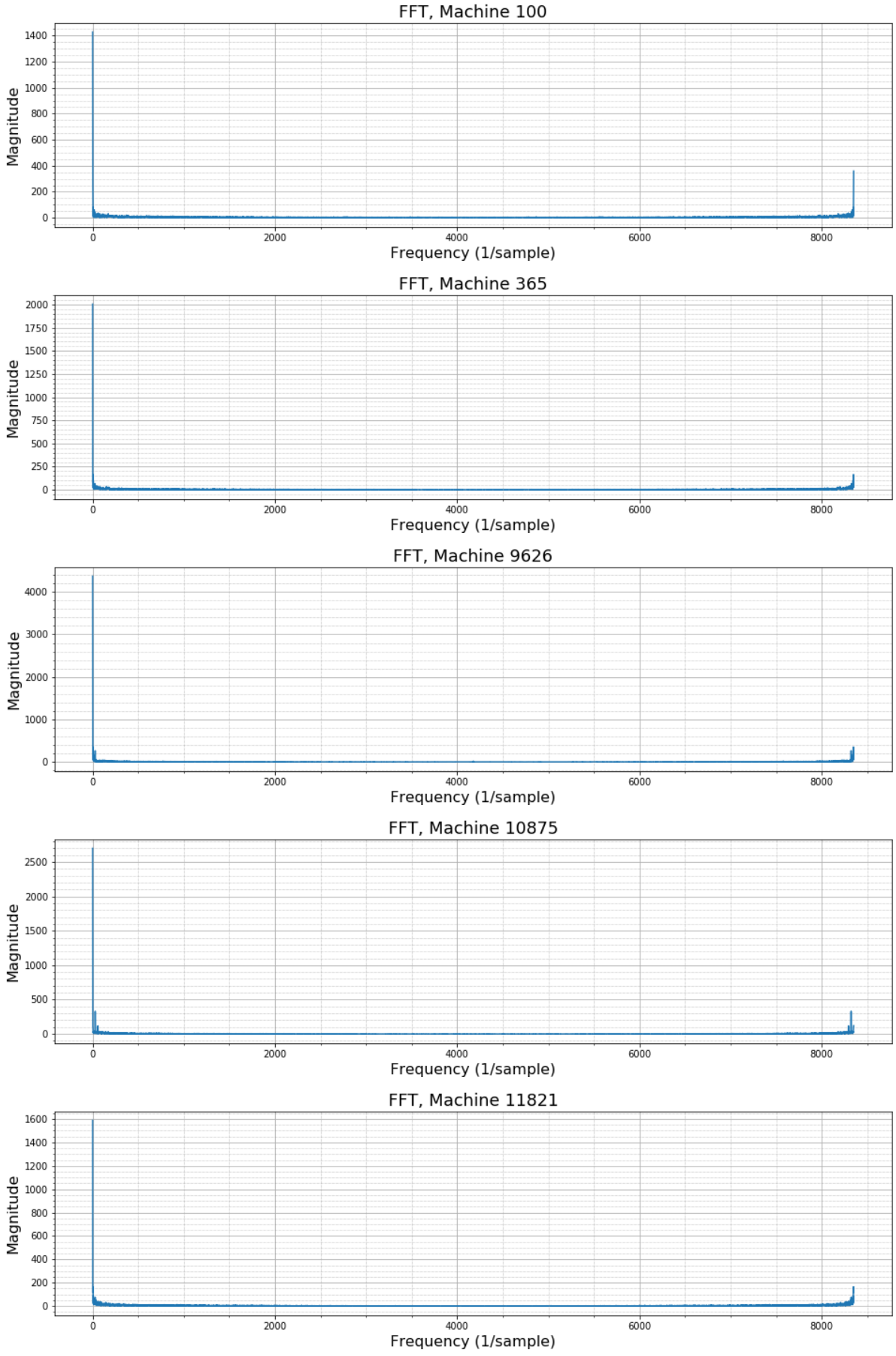


Figure 15: FFT five machines.

7.3 Spatial Analysis

As mentioned in Section 1, running one forecasting model for each individual machine is computationally expensive. The purpose of the spatial analysis is to identify some form of spatial characteristics between different machines which we can exploit to reduce the dimensionality of our optimisation problem.

7.3.1 Full Cross-correlation

For preliminary analysis, the cross-correlation between time-series of all the machines are calculated with the full length of the time-series used as the window. A correlation matrix with cross-correlation between all pairs of time-series is compiled giving a 12,476 by 12,476 matrix. Note that we take the absolute value of the cross-correlation as we only care about the strength of the correlation.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.000000	0.182548	0.293396	0.147556	0.371543	0.434098	0.198072	0.413441	0.296714	0.489531	0.164437	0.414534	0.268098	0.364665	0.282952
1	0.182548	1.000000	0.243375	0.189525	0.006541	0.171590	0.244268	0.087363	0.352426	0.361932	0.085538	0.266899	0.121294	0.149884	0.092882
2	0.293396	0.243375	1.000000	0.140648	0.112303	0.053745	0.186293	0.054868	0.186472	0.278411	0.077946	0.074898	0.160709	0.001928	0.012881
3	0.147556	0.189525	0.140648	1.000000	0.058056	0.253711	0.241811	0.251068	0.339572	0.300277	0.094345	0.207781	0.136579	0.216478	0.197561
4	0.371543	0.006541	0.112303	0.058056	1.000000	0.014277	0.045071	0.115329	0.105065	0.121746	0.125211	0.093713	0.066317	0.056509	0.057156
5	0.434098	0.171590	0.053745	0.253711	0.014277	1.000000	0.283545	0.407114	0.326234	0.345385	0.160917	0.398853	0.239169	0.431660	0.346313
6	0.198072	0.244268	0.186293	0.241811	0.045071	0.283545	1.000000	0.274247	0.381159	0.438145	0.000399	0.340569	0.170725	0.225147	0.325733
7	0.413441	0.087363	0.054868	0.251068	0.115329	0.407114	0.274247	1.000000	0.338810	0.444664	0.187897	0.280162	0.254886	0.375323	0.440676
8	0.296714	0.352426	0.186472	0.339572	0.105065	0.326234	0.381159	0.338810	1.000000	0.577517	0.292374	0.337993	0.217038	0.372573	0.439583
9	0.489531	0.361932	0.278411	0.300277	0.121746	0.345385	0.438145	0.444664	0.577517	1.000000	0.098194	0.452990	0.269100	0.403685	0.447851
10	0.164437	0.085538	0.077946	0.094345	0.125211	0.160917	0.000399	0.187897	0.292374	0.098194	1.000000	0.093260	0.032346	0.228335	0.341564
11	0.414534	0.266899	0.074898	0.207781	0.093713	0.398853	0.340569	0.280162	0.337993	0.452990	0.093260	1.000000	0.242623	0.367548	0.225990
12	0.268098	0.121294	0.160709	0.136579	0.066317	0.239169	0.170725	0.254886	0.217038	0.269100	0.032346	0.242623	1.000000	0.211140	0.174837
13	0.364665	0.149884	0.001928	0.216478	0.056509	0.431660	0.225147	0.375323	0.372573	0.403685	0.228335	0.367548	0.211140	1.000000	0.398317
14	0.282952	0.092882	0.012881	0.197561	0.057156	0.346313	0.325733	0.440676	0.439583	0.447851	0.341564	0.225990	0.174837	0.398317	1.000000

Figure 16: Sub-matrix of correlation matrix indicating the cross-correlation between different pairs of machines. The label on both axis indicate the index of each machine while each of the values indicate the cross-correlation between the machines. It is expected that the matrix is symmetrical and that the values at the diagonal is always one.

To check whether it is feasible to cluster the machines using cross-correlation, we identify the maximum absolute cross-correlation for each machine with any other machine. If the maximum cross-correlation for the machine is high, we can ensure that it will be clustered together with at least one other machine with high cross-correlation.

Figure 17 shows the distribution of the maximum absolute cross-correlation index for all machines. It is observed the distribution has a mean value of 0.648 which is sufficiently high. There are only 165 in 12,476 machines (about 1.32%) which has a maximum absolute cross-correlation of lower than 0.4 while 2,989 of 12,476 machines (about 24%) has a maximum absolute cross-correlation of higher than 0.7. This implies that clustering with cross-correlation is feasible.

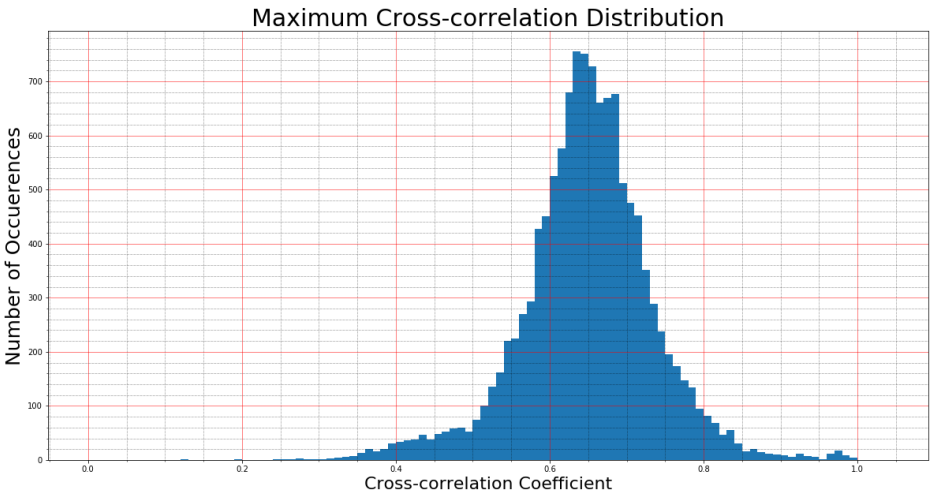


Figure 17: Distribution of maximum absolute cross-correlation.

7.3.2 Cross-correlation with One Day Window

In this section, we would like to observe the change in the cross-correlation between pairs of machines. The analysis is done by calculating the cross-correlation between machines using a one-day window:

$$\text{cross-correlation with one day window}[n] = corr(x_i[n : n + 288], x_j[n : n + 288])$$

The analysis is illustrated in Figure 18 where the cross-correlation between machine 10875 and machine 11821 is calculated using a one day window. The value at the red dot and green dot in the bottom diagram are the correlation between the segments highlighted in red and green in the top two diagrams respectively.

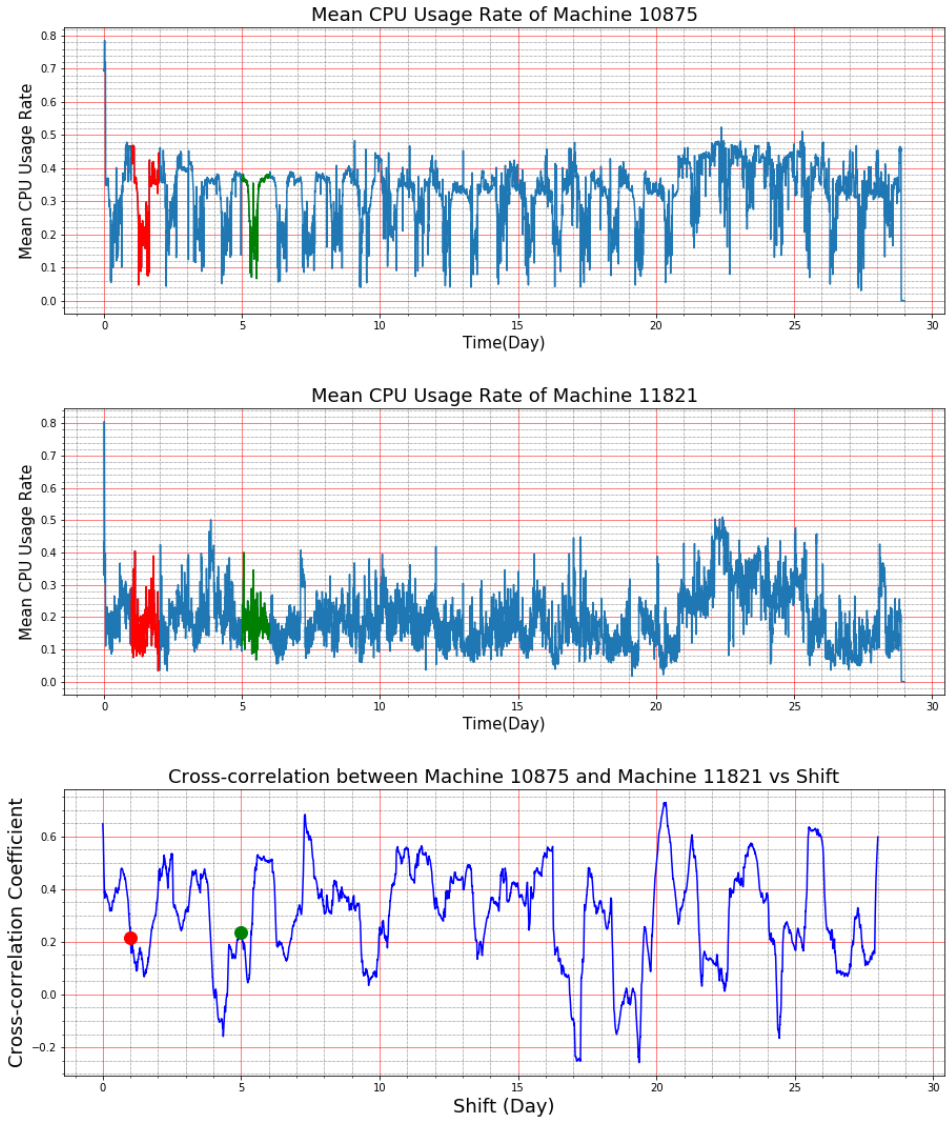


Figure 18: Cross-correlation with One Day Window between Machine 10875 and Machine 11821

More examples of this analysis can be seen in Figure 19. It is seen that the correlation between any two time-series fluctuates rapidly over time. Hence we deduce that the clustering of the time-series has to be dynamic as a static clustering will not be able to capture the change in the cross-correlation between time-series over time.

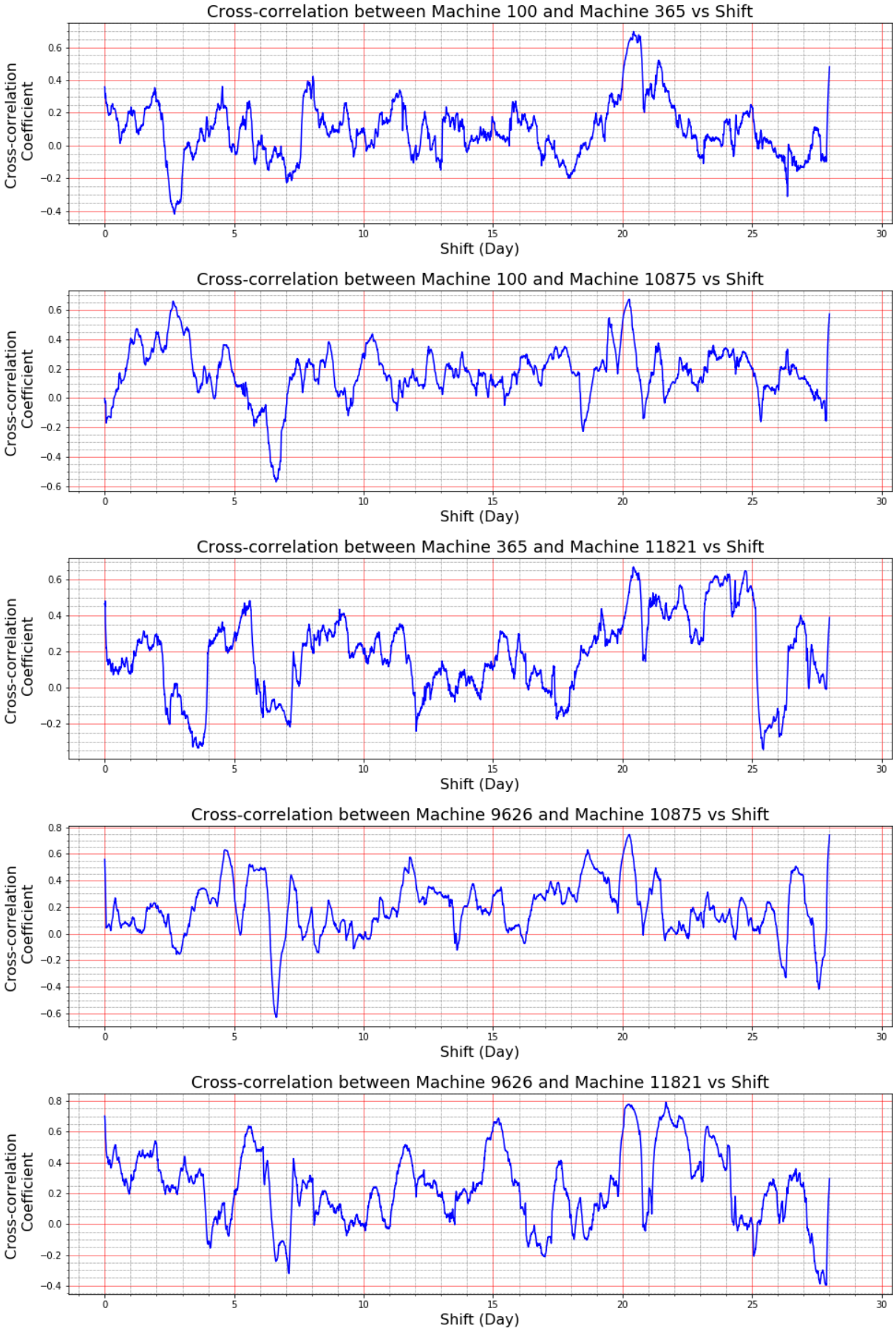


Figure 19: Cross-correlation with one day window of five pairs of machines.

7.3.3 Out of Phase Correlation

It is hypothesized that some of the time-series are out of phased version of some other time-series i.e.

$$\mathbf{x}_i[n] = \mathbf{x}_j[n + lag]$$

It might also be possible that for machines of different capacity, while the absolute value of CPU utilisation may differ but the trend might still be similar i.e.

$$\mathbf{x}_i[n] = s \cdot \mathbf{x}_j[n + lag] + d$$

where s and d are constants. This hypothesis again stems from the assumption that cloud CPU utilisation is related to internet usage. Logically, internet usage will peak during the day and drop during the night but given that the machines represent different geographical locations with different time-zones, these trends are out of phase from one another. To prove this hypothesis, we define the out of phase correlation where we observe the cross-correlation between lagged version of some time-series i.e.

$$\text{out of phase correlation}(lag) = corr(\mathbf{x}_i[lag:], \mathbf{x}_j[:-lag])$$

If the hypothesis is true, pairs of time-series will exhibit a strong correlation at a lag in which they are out of phase from each other. The correlation with lag of five machines are shown in Figure 20. This analysis is done on 400 pairs of randomly selected machines due to computational constraint. We then observe the lag in which the strongest correlation occurs for these 400 machines.

It was observed that for most of the pairs of machines, the strongest correlation occurs when the lag is zero, meaning they are in phase. For the pairs of machine which are out of phase, the lag in which the strongest correlation occurred exhibits no identifiable patterns. Hence, it is not feasible for us to group time-series together using an the out of phase correlation.

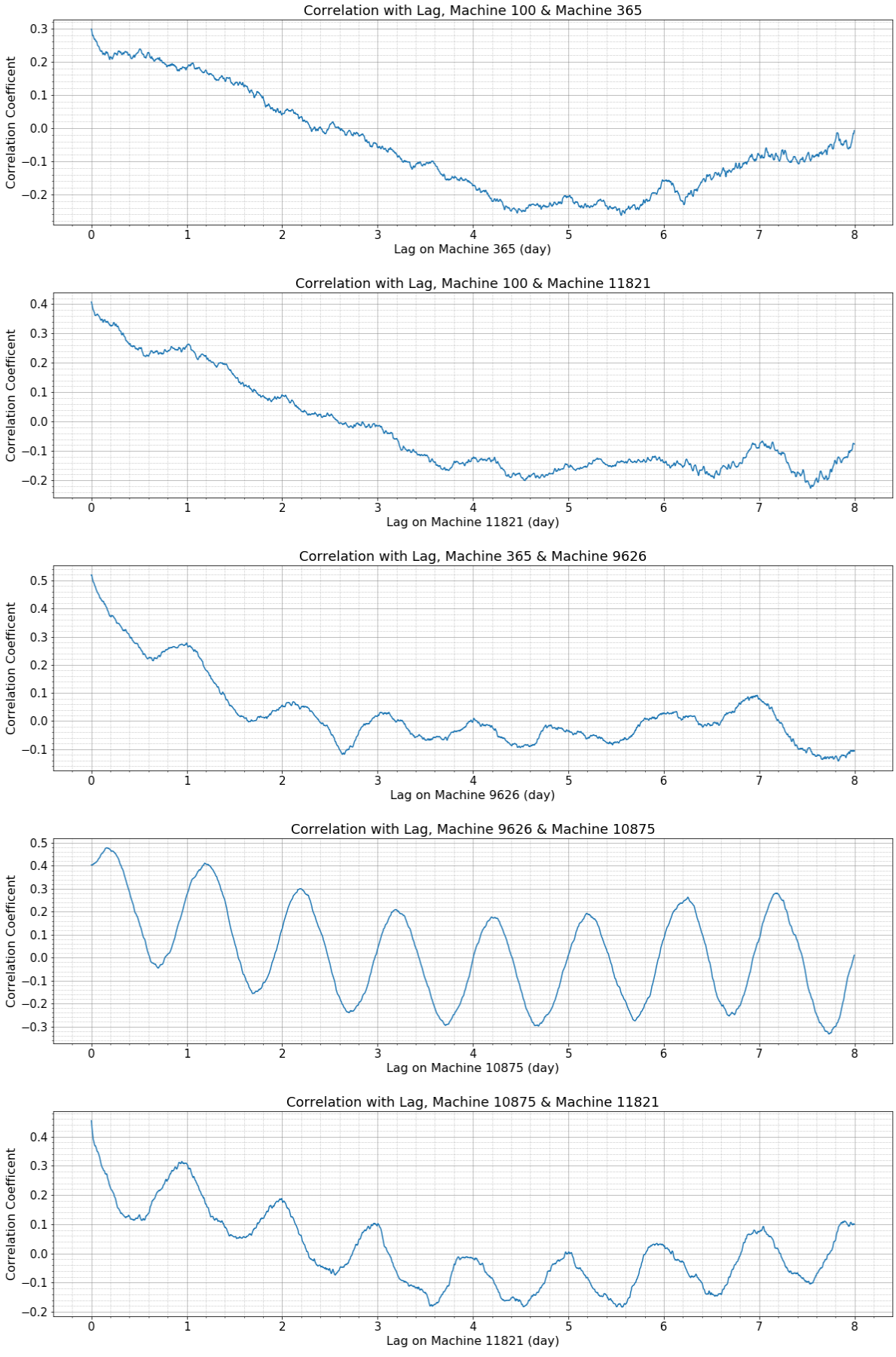


Figure 20: Correlation with lag of five pairs of time-series. It is observed that machine 9626 and machine 10875 are out of phase by about 5 hours. The other pairs of machine are in phase as their out of phase correlation peaked at lag = 0.

8 Proposed Solution

In this section, we will discuss the proposed framework to forecast cloud resource utilisation starting with the overview. The design of each of the section of the proposed framework is further elaborated in the section immediately after this.

8.1 Overview

Following the analysis, we propose a correlation-based online model which retains the general framework of the Tuor framework. Our proposed framework consists of four parts:

1. **Dynamic clustering of time-series** which clusters time-series which are highly correlated together in a dynamic manner.
2. **Computation of general model** where we compute a general time-series to generalise all the time-series in each cluster. Each cluster will have one general model. The notation, $\mathbf{y}_j$ will refer to the general model for cluster indexed j.
3. **Temporal forecasting** where we preform one time-step forecast on the general model. The forecasted sample of the general model will be given the notation $\hat{y}$.
4. **Forecast scaling** where we scale the forecast made on the general model to fit all of the time-series in respective clusters.

Note that for this project, we do not consider the adaptive transmission as in the Tuor framework. We will assume that all of the measurements from the local nodes are transmitted to the control node. This project focuses on improving the time-series clustering and the temporal forecasting portion of the Tuor framework. The idea behind our proposed framework is that we are trying to first cluster time-series of similar shape together instead of looking at their absolute values directly. It is hypothesized that the time-series with similar shape will have the same trajectory.

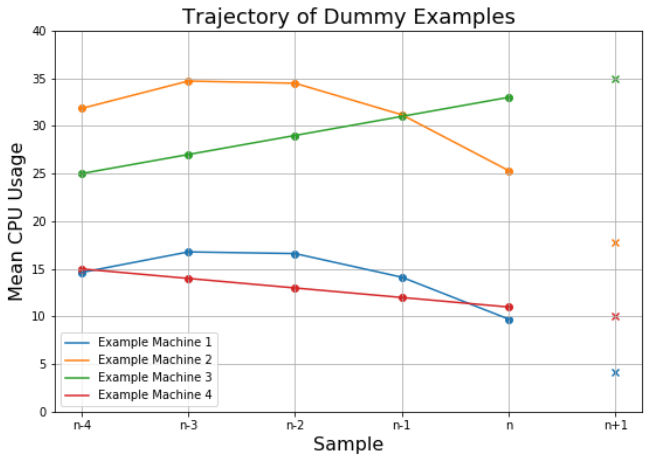


Figure 21: Trajectory of four example time-series.

In Figure 21, we show an artificial dummy example of this idea. Sample [n] denotes the current time. We have access to all the samples at and prior to sample [n]. Sample [n+1] is the sample in the future and is the sample in which we have to forecast. It is seen that time-series of machine (1) and (2) have similar shape and have a high positive cross-correlation. Time-series of machine (3) and (4) also exhibit high correlation even though the correlation is negative. We cluster these four time-series into two clusters based on the strength of their correlation.

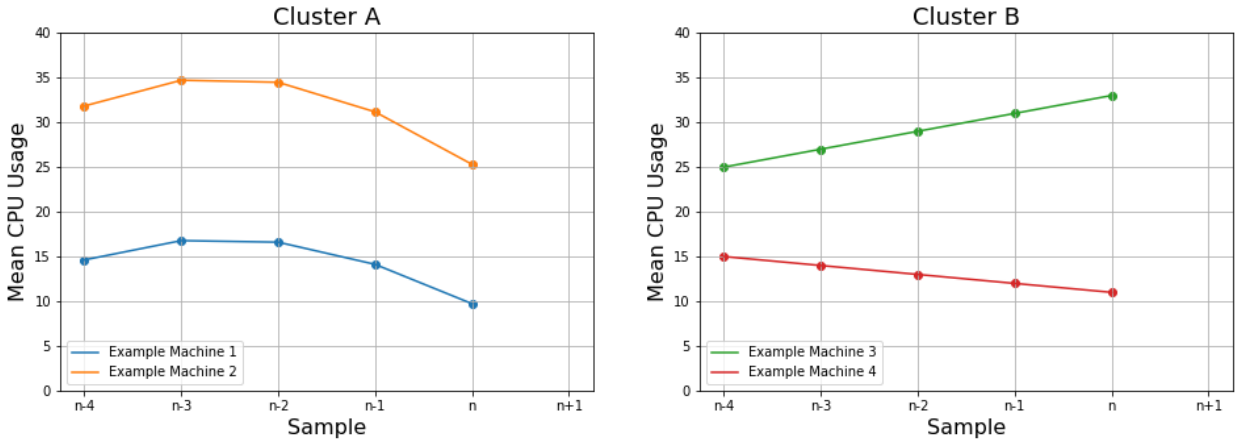


Figure 22: Clustering of the four example time-series.

We then come up with a general model, for each of the cluster. The general model has a high correlation with all the time-series in the cluster. After that, we perform forecasting on the general model. This can be seen in Figure 23.

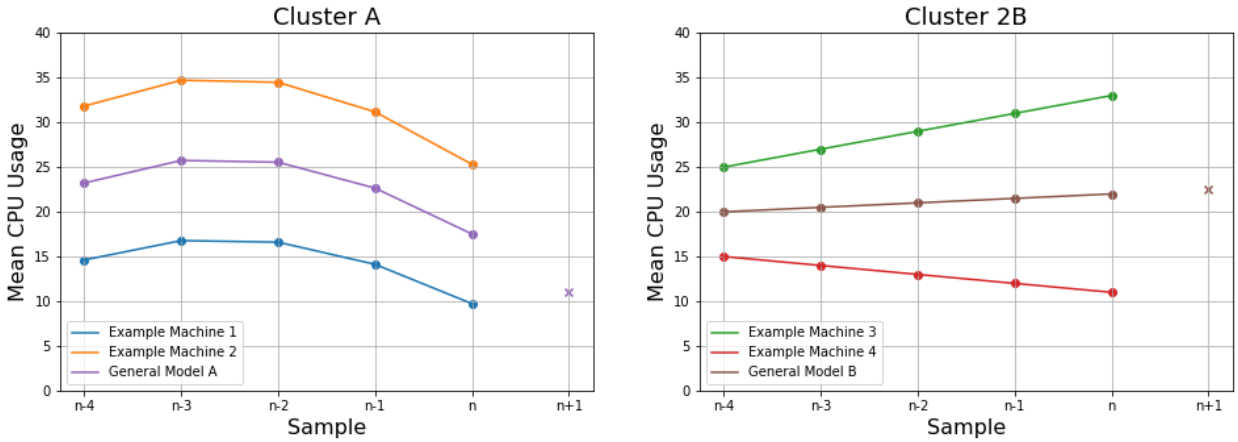


Figure 23: Forecasting on the general model for each cluster. The "x" denotes the forecasted value of the general time-series.

Since we know that the general model, $\mathbf{y}_k$ has a high correlation with all the time-series in the cluster, we can approximate all the time-series within each cluster as:

$$\mathbf{x}_i[n+1-w:n+1] = s[n] \cdot \mathbf{y}_k[n] + d[n]$$

where s is the scaling factor while d is the displacement, both of which are obtained from linear regression. We can then use the constants, s and d, to scale the forecast of the general model to fit all the time-series in each cluster.

$$\hat{x}_i[n+1] = s[n] \cdot \hat{y}_k[n+1] + d[n]$$

Doing this for all time-series in all clusters at each time-step and we end up with the desired forecasts.

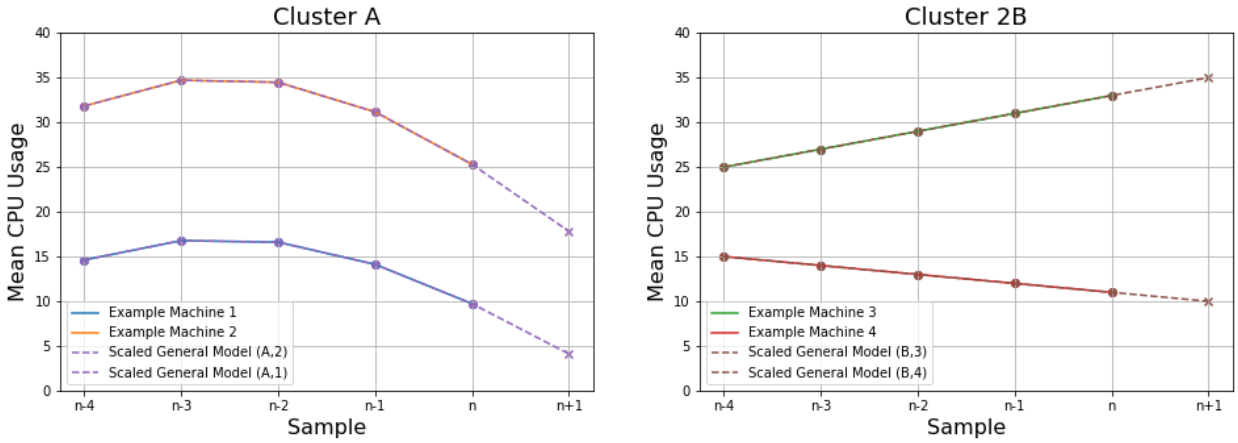


Figure 24: Scaling of the general model to fit all the time-series in each cluster.

8.2 Long-Term vs Short-Term Forecasting

The first thing we have to decide before designing a time-series forecasting framework is to consider the type of forecasting we would like to do: long-term forecasting or short-term forecasting. Long-term time-series forecasting normally relies heavily on exploiting the seasonality in the data set or other parameters which are strongly correlated with the observations in which we are trying to forecast.

From Section 7.2.2, it is seen that the seasonal component is generally the weakest component in all the time-series in our data set. Besides that, we do not have access to other parameters which has strong correlation with CPU utilisation. From Section 7.1, it is observed that memory utilisation, which is the only other parameter which we have access to, has a weak correlation with CPU utilisation.

Other than that, for the purpose of cloud resource utilisation, it is unnecessary for us to make long-term forecasts as the system should be fast enough to distribute resources in real time. It is hence reasonable for us to design a forecasting framework which makes only short-term forecasts.

8.3 Online vs Offline Forecasting

We will need to consider the type of model we would like to design: online or offline. The online model is updated at every time-step and is normally used to make short-term forecasting. The offline model is trained on a static pool of data and it is not retrained as frequently as the online model. In contrast to the online models, offline models are normally used for long-term forecasting. It is reasonable to assume that for models forecasting cloud resource utilisation, an offline model is retrained everyday and it is used to make forecasts of one day in advance. Just as choosing the distance measure, choosing the appropriate method requires thorough consideration as both methods has different advantages and disadvantages. Below are some differences between the two types of models: (Hunt 2017):

1. **Velocity**
   Offline learning trains data on static pools of data while online learning makes a small incremental update continuously as each data point arrives. This allows for faster model update

2. **Volume**
   Training an online model uses less memory as the it only looks at the most recent data point. The data point is forsaken once it is used to update the model.

3. **Variety**
   Since online learning allows faster model update, it allows the model to adapt faster to rapidly changing data. Furthermore, a "forgetting factor" can be introduced to online models which gradually discounts the importance of past data.

4. **Accuracy**
   Offline learning uses multiple epoch (passes through the entire training data set) to train the model while online learning only allows the training data to pass once. This allows offline learning to reach a better local minimum which might give more accurate results. Offline learning might also be more potent to noise depending on the training data set.

5. **Stability**
   Models trained by online learning methods are generally harder to maintain as online models can become unstable or corrupted due to contaminated data.

In summary, the online approach should be used when the forecasting model needs to quickly react to rapidly changing trend in the data. It should also be used when computational resources (both CPU and memory) is limited. The offline approach is a good alternative when the data is consistent or when the data available is diverse enough to represent most of the possible use cases.

Considering the advantages and disadvantages of both online and offline models, as well as considering the purpose of our model, an online model would be the more appropriate choice.

8.4 Time-series Clustering

From the analysis in Section 7.3.1, it is observed that the maximum correlation between different time-series is high. Hence, correlation is a reasonable distance measure to be used for clustering. However, from Section 7.3.2, it is observed that the correlation between any two time-series fluctuates rapidly over time. It is hence unreasonable to cluster the time-series in a static manner. Instead, the clustering needs to change over time as the correlation between time-series change.

The first step to this process is to calculate the rolling correlation between every pair of time-series to obtain a rolling correlation matrix at each time-step n, $\mathbf{M}[n]$.

$$\mathbf{M}[n] := \left\{ M_{i,j}[n] = corr(\mathbf{x}_i[n-w+1:n+1], \mathbf{x}_j[n-w+1:n+1]) \right\}$$

where w is the rolling correlation window i.e. the number of past points to be considered for clustering. Note that $\mathbf{M}[n]$ acts as a "rolling correlation matrix" analogous to the well-known rolling mean.

Next, we consider the two most popular clustering algorithm to be used: K-means clustering and agglomerative hierarchical clustering. K-means clustering has a much lower computational complexity than of agglomerative hierarchical clustering. K-means clustering has a complexity of $O(n)$ while agglomerative hierarchical clustering has a complexity of $O(n^2)$. It is noted that K-means clustering is the algorithm of choice used in the Tuor framework. However, to use K-means, we require the "coordinates" of the time-series instead of the distance matrix. A way around this problem would be to set a few reference points and map the time-series into a plane based on the correlation matrix to give us the "coordinates". This method is however highly counter-productive as well as computational expensive [1]. Hence, it is reasonable to instead use hierarchical clustering which takes in a distance matrix directly, despite it having a higher computational complexity.

Also note that in the Tuor framework, evolutionary clustering (clustering with a memory element in time) is used. For this project, we consider a simplified approach in which we do not incorporate evolutionary clustering. Instead, the time-series are clustered independently at every time-step.

For hierarchical clustering, we need to decide on two hyperparameters: the affinity and the linkage. The linkage criterion determines which distance measure to use between sets of observation. The algorithm will merge the pairs of cluster that optimises this criterion. Affinity refers to the metric used to compute the linkage. For our purposes, we set euclidean distance as the affinity and Ward's method as the linkage. The ward linkage criterion refers to a minimum variance approach where the hierarchical clustering algorithm will minimise the variance of the affinity chosen in each of the cluster. In other words, our clustering algorithm will cluster the time-series in such a way that the variance of the euclidean distance between every point in each cluster is kept to a minimum.

In summary, the agglomerative hierarchical clustering takes in the rolling correlation matrix and produces a clustering, $\mathbf{C}$ for each time-step.

8.5 Generalisation Model

After the clustering, we will need to create a general model, $\mathbf{y} \in IR^w$ to generalise each of the cluster. This is done by computing a general model where the distance from every time-series in the cluster to the general model is minimum. The distance measure used in the minimisation is conventionally the same distance measure used in the previous clustering step. This should not come as a surprise as in the clustering step, essentially clustering processes is a method of minimising a certain "difference" or maximising a certain "similarity" within each cluster. The "similarity" or "difference" is characterised by the defined distance measure. It is hence logical to minimise the same distance measure to compute a general model which best characterise each of the cluster. The minimisation problem is formulated below:

[1] The algorithm for mapping of coordinates from a distance matrix is discussed in this paper (Crippen 1978).

$$\underset{\mathbf{y}\in\mathbb{R}^w}{\arg\max} \sum_{i\in\mathbf{C}_k} \Big(similarity(\mathbf{y}, \mathbf{x}_i[n+1-w:n+1])\Big)^2 \tag{15}$$

which in our case gives:

$$\underset{\mathbf{y}\in\mathbb{R}^w}{\arg\max} \sum_{i\in\mathbf{C}_k} \Big(corr(\mathbf{y}, \mathbf{x}_i[n+1-w:n+1])\Big)^2 \tag{16}$$

Note that we take the square of the correlation because we want the solution to be sign invariant and also we would like to penalize low correlation and reward large correlation.

It is deduced that the solution to the minimisation problem (16) does not have a unique solution. The proof to this is trivial. Consider that the time-series $\mathbf{y}$ is a solution to the problem, due to linear properties of the correlation distance measure, $\alpha\mathbf{y}+\beta$ will also be a solution, where α and β are constants. We can force the solution of the minimisation problem to be unique by introducing a regularisation term:

$$\underset{\mathbf{y}\in\mathbb{R}^w}{\arg\max} \sum_{i\in\mathbf{C}_k} \Big(corr(\mathbf{y}, \mathbf{x}_i[n+1-w:n+1])\Big)^2 + \lambda||\mathbf{y}||_1 \tag{17}$$

Note that we choose L1 normalisation as it normally results into a sparse solution. However, it is obvious that (17) does not have a closed form solution and it is highly mathematically complex [2]. This problem will most likely need to be solved iterative which if implemented would again increase the computational complexity of our overall problem.

Instead we choose one of the time-series in the cluster which maximises the correlation between all time-series in the cluster to be the general model. We hence change the maximisation problem to:

$$\underset{j\in\mathbf{C}_k}{\arg\max} \sum_{i\in\mathbf{C}_k} \Big(corr(\mathbf{x}_j, \mathbf{x}_i)\Big)^2 \tag{18}$$

which is an integer optimisation problem in which can easily be solved easily using exhaustive search. (18) is well defined and has a unique solution (given that no pairs of time-series has a correlation of one which is proven in Section 7.3.1). It is also computationally cheap to compute since we already have the correlation matrix.

8.6 Temporal Forecasting

To make the forecast on the general model, we choose to use the ARIMA forecasting model. The ARIMA model is a time-series forecasting model which describes future observations as linear combinations of past values. It can be used to model any 'non-seasonal' time series that exhibits patterns and is not a random white noise.

ARIMA stands for "Auto-regressive Integrated Moving Average" and it is made up of three individual parts: the Auto-regressive (AR), Integrated (I) and Moving Average (MA). The ARIMA model can be characterised using three parameters. Conventionally, the notation p is used to describe the order of the AR term, q for the MA term and lastly d for the I term. Hence, an ARIMA model is denoted as ARIMA(p,d,q).

The AR term of the model describe the linear relationship between the current observation and the past q observations:

$$\hat{y}[n] = \mu + \beta_1 y[n-1] + \beta_2 y[n-2] + ... + \beta_p y[n-p]$$

where μ is a constant intercept term and β are constant coefficients. It is hence noted that a pure AR model attempts to describe $y[n]$ with its own lags.

The MA term of the model describes the linear relationship between the current observation and the past q lagged forecast errors:

$$\hat{y}[n] = \gamma + \phi_1 \epsilon_{n-1} + \phi_2 \epsilon_{t-2} + ... + \phi_p \epsilon_{n-q}$$

where γ is a constant intercept term and ϕ are constant coefficients. Epsilon gives the lagged forecast error defined as:

[2] The maximum correlation fitting problem is further discussed in (Livadiotis & McComas 2013)

$$y[n] = \hat{y}[n] + \epsilon_n$$
$$y[n-1] = \hat{y}[n-1] + \epsilon_{n-1}$$

The I term describes the differencing term where instead of describing y directly, the model instead attempt to describe the "differenced" term, $z[n]$:

$$z[n] = y[n] - y[n-d]$$

The purpose of differencing is to make the time-series stationary. Differencing can be think of as differentiating the time-series with the order of d. It is noted that in the paper (Tuor et al. 2019), on top of the ARIMA forecasting model, the RNN model is also used. In this report, we will stick with the ARIMA model as it is seen in Section 7.2.3 that for all the time-series, there is strong autocorrelation for small values of lag. This means that for small values of lag, the values of the time-series exhibit strong linear relationships which can be modeled using an ARIMA model. Besides that, the RNN model is much more computationally expensive as compared to the ARIMA model.

The ARIMA forecasting model is used to make one-step forecast on the general model.

8.7 Forecast Scaling

After forecasting on the general model is done, we need to scale the forecast of the general model, $\hat{y}[n+1]$ to fit all the other time-series in the cluster. Since we know that the general model maximises the correlation between itself and all the other time-series in the cluster, we can exploit the linear characteristics which is exhibited when the correlation between two time-series is high:

$$\hat{x}_i[n+1] = s_i[n+1]\hat{y}[n+1] + d_i[n+1]$$

where $\hat{y}[n+1]$ is the forecasted value of the general model, $s_i[n+1]$ is a scaling term and $d_i[n+1]$ is a displacement term. The s and d term can easily be obtained from simple linear regression by solving the least-square minimisation problem:

$$s_i[n+1], d_i[n+1] = \underset{s,d \in \mathbb{R}}{\arg\min} \sum_{j=0}^{w-1} \Big((s \cdot y[n-j] + d) - x_i[n-j]) \Big)^2$$

w is again the number of past points to be considered for clustering.

To prevent the error from drifting off as n increases, we add a rolling error term, $e_i[n+1]$ to offset the effect of errors, giving:

$$\hat{x}_i[n+1] = s_i[n+1]\hat{y}[n+1] + d_i[n+1] + e_i[n+1]$$

where we define $e_i[n+1]$ as:

$$e_i[n+1] = \frac{1}{r} \sum_{j=0}^{r-1} x_i[n-j] - \hat{x}_i[n-j]$$

where the rolling error window, r is a user-defined parameter. Doing this for all time-series for all clusters will give us the required forecasted values for each time-series.

9 Implementation

9.1 Sub-data set

Due to the constrain on computational power, it is not feasible for us to implement the framework on all the 12,476 time-series. Since the proposed framework is designed on the assumption that different time-series have strong correlation with one another, it is sufficient for us to test the framework on a sub-set of the time-series which has strong cross-correlation with one another. In this section, we will discuss the process of choosing the subset of time-series.

We would like to choose a number small enough in which our computational capacity is able to handle, but not too small to which it defeats the purpose of clustering the time-series. It is reminded that the entire purpose of clustering the time-series is to essentially reduce the dimensionality of the overall problem as it was not feasible for us to come up with a single forecasting model for each time-series. To do this, we look at the distribution maximum correlation of each of the time-series. Note that we consider the maximum correlation of each time-series instead of the correlation between all time-series because in clustering, as long as there exist a single time-series which has a high correlation with another time-series, the two time-series will be cluster together. By choosing the time-series based on their maximum correlation, we can make sure that all the selected time-series will be clustered with at least one other time-series with high correlation.

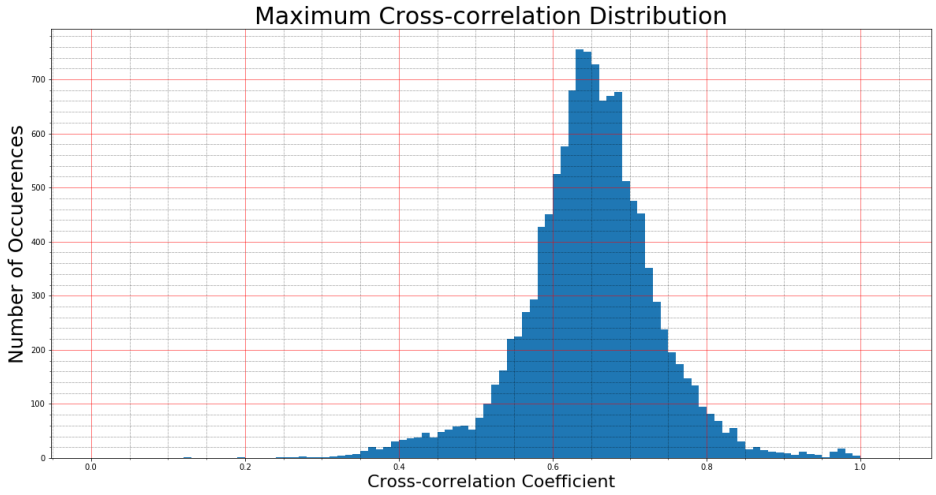


Figure 25: distribution of maximum absolute cross-correlation.

Besides that, from initial testing, it was empirically determined that around 1,000 time-series is a good number for computational complexity as it does not result in an overly long run time. To select the best time-series for implementation, the following steps were taken:

1. We first obtain the correlation matrix of all the 12,476 time-series.
2. We then obtain the maximum correlation for each of the machine by taking the maximum value across each row of the correlation matrix
3. We then sort the time-series according to their maximum correlation and we take the time-series with maximum correlation larger than 0.75
4. We then remove some of the corrupted time-series which contains too many zeros

Following the mentioned steps, we end up with 1154 time-series which is an appropriate amount for implementation and testing.

9.2 Time-series Clustering

Per mentioned in Section 8.4, we will be using agglomerative hierarchical clustering to cluster the time-series. There are two further hyperparameters in which we need to optimise: the number of clusters and the size of the window in which the time-series are clustered on. If the number of clusters is large, the time-series in each cluster will all be strongly correlated but the reduction in computational complexity from the clustering will

be small. Conversely, if the number of clusters is small, the opposite is achieved: we get a large reduction in computational complexity but the time-series in each of the cluster will not be guaranteed to be high. Hence, the number of clusters is a trade off between reduction in computational complexity and the strength of the correlation between time-series in each cluster. Note that we fix the number of clusters instead of fixing a minimum threshold of correlation for the time-series to be clustered. This is because we would like to guarantee a constant reduction in computational complexity at every time-step. Furthermore, if in the future where we would like to implement evolutionary clustering, it is necessary for us to have a fixed number of clusters [3].

The window in which the time-series are clustered using rolling correlation is analogous to the window used in the commonly known "rolling mean". We will refer this window to as the "rolling correlation window". Similarly to the rolling mean window, the size of the rolling correlation window gives a trade-off between noise rejection and its sensitivity to change in the data. The bigger the window, the better the framework will be at rejecting noise. However, the framework will also be less sensitive to change. Besides that, it is observed that the smaller the moving correlation window, the stronger the correlation between machines in general.

It is noted that both the mentioned hyperparamters have an effect of the correlation of the time-series. Hence they are not independent. To optimise these two hyperparameters, we need to first define some measure of performance for different combinations of the hyperparameter. For the reduction in computational complexity, we define the "mean number of Time-series Per Cluster", TPC:

$$TPC = \frac{\text{Total number of time-series}}{\text{Number of clusters}}$$

where for our case, the total number of time-series = 1154 since we are only considering the most correlated 1154 time-series per previously discussed. Ideally we want the TPC to be huge so that we get a large decrease in computational complexity. To benchmark the minimum correlation in each cluster, we define the "Minimum Correlation across All Clusters" (MCAC). To calculate the MCAC, we take the following steps:

1. Using the chosen combination of hyperparameters, we cluster the time-series for 50 time-step.
2. For each time-step,
   - we determine the minimum correlation between all machines in each cluster.
   - We then sort the cluster in ascending minimum correlation i.e. cluster index 0, $\mathbf{C}_0$ has the smallest minimum correlation where as $\mathbf{C}_{K-1}$ has the largest minimum correlation. K is the total number of clusters.
3. We consider the minimum correlation in $\mathbf{C}_0$. Since the clusters were sorted in the previous step, minimum correlation in $\mathbf{C}_0$ gives the minimum correlation in all clusters. We take the mean of the minimum correlation in $\mathbf{m}_0$ across all 50 time-steps. We define this value as the "Minimum Correlation across All Clusters" (MCAC).

For each combination, we consider its MCAC and TPC. Multiple different combinations of the hyperparameters are tested and it is empirically determined that 100 clusters with a rolling correlation window of 5 samples gives the optimal combination.

[3] Evolutionary clustering requires the number of clusters to be the same for every time-step as it is required to track the "movement" of each cluster via a mapping process. Having different number of clusters at each time-step would distort that mapping process. (Chakrabarti et al. 2006)

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94
0	0.624641	0.695330	0.753479	0.759231	0.768212	0.774902	0.777356	0.781403	0.792514	0.793978	...	0.945782	0.948633	0.951010	0.953743	0.954125
1	0.716709	0.748666	0.750241	0.769885	0.795677	0.800299	0.803523	0.805889	0.806269	0.807712	...	0.944146	0.950160	0.952725	0.954215	0.957212
2	0.724726	0.739771	0.751149	0.754549	0.766992	0.783895	0.784828	0.787121	0.790775	0.794373	...	0.945911	0.946399	0.951401	0.951572	0.951936
3	0.734899	0.742886	0.746910	0.750964	0.781582	0.784917	0.785672	0.788902	0.797889	0.798976	...	0.942141	0.951341	0.951632	0.951709	0.951970
4	0.706765	0.749853	0.757637	0.771153	0.772628	0.777880	0.779320	0.781904	0.796800	0.803706	...	0.944865	0.950928	0.951539	0.952452	0.954762
5	0.579794	0.649109	0.722225	0.734117	0.737912	0.746868	0.751544	0.767013	0.775924	0.776650	...	0.938205	0.940751	0.946425	0.953274	0.954046
6	0.717397	0.723722	0.750336	0.751732	0.754515	0.763444	0.775876	0.779909	0.783759	0.788841	...	0.944309	0.946256	0.947739	0.948829	0.952669
7	0.677775	0.710342	0.757594	0.760360	0.764308	0.769878	0.783788	0.791743	0.792041	0.796710	...	0.945420	0.946823	0.947738	0.948252	0.951775
8	0.684597	0.738889	0.746262	0.749432	0.766979	0.770545	0.772616	0.778907	0.790819	0.797450	...	0.941481	0.941846	0.953100	0.953247	0.954788
9	0.666382	0.737433	0.771294	0.781506	0.786159	0.793969	0.796341	0.798145	0.804547	0.806409	...	0.946530	0.948055	0.948694	0.950311	0.953528

Figure 26: The minimum correlation in each cluster for 10 time-steps. The rows are the time-step while the columns are the clusters. Here we can observed the change in the minimum correlation in each cluster. The clusters are arranged in such a way that the cluster with smaller index (columns on the left) have a smaller minimum correlation. The first column gives the change in minimum correlation in cluster index 0 which also gives the minimum correlation across all clusters (MCAC). We would like to choose a combination of hyperparameters where the MCAC is not too small.

9.3 Generating General Model for Each Cluster

For each of the cluster, we need to choose the time-series which has the highest correlation with all the other time-series in the cluster and make it the general model for the cluster. To do this, we need to calculate the sum of the squares of correlation (SOC) for each time-series:

$$SOC(i) = \sum_{j \in \mathbf{C}_k} \Big(corr(\mathbf{x}_i, \mathbf{x}_j) \Big)^2 \tag{19}$$

However since we already have the rolling correlation matrix, $\mathbf{m}_k$ for each of the cluster, the calculation simplifies to:

$$SOC(i) = \sum_{j \in \mathbf{C}_k} \Big(m_{k,(i,j)} \Big)^2 \tag{20}$$

After we have the SOC, we then set the time-series with the highest SOC to be the general model.

9.4 Forecasting

For the forecasting, we will need to determine the order of the ARIMA model to be used. To do this, we use a simple grid search. We perform forecasting on the same set of data using ARIMA models of different order. We then analyse the performance of each of the model to decide which order to be used. For each of the model of different order, we analyse its accuracy and computation complexity using its Mean Square Error (MSE) and Time for One Prediction respectively.

ARIMA Order	**MSE**	**Time for One Prediction** ($\times 10^{-2}$ **sec**)
(2,0,0)	0.2391	0.8940
(3,0,0)	0.1837	2.356
(4,0,0)	0.2361	3.543
(5,0,0)	0.2375	5.255
(3,1,0)	0.1890	2.112
(3,0,1)	0.2381	1.898
(3,0,3)	0.1760	4.769
(3,1,3)	0.1764	79.92

Table 3: Accuracy and computational complexity of ARIMA models of different order.

It is not surprising that generally the accuracy of the model increases with computational complexity. However, the ARIMA model suffers severely from the effects of diminishing marginal returns i.e. after a certain optimal point, any increase in computational complexity only gives marginal increase in accuracy. This is clearly seen in the increase in computational complexity from (3,0,0) to (3,0,3)

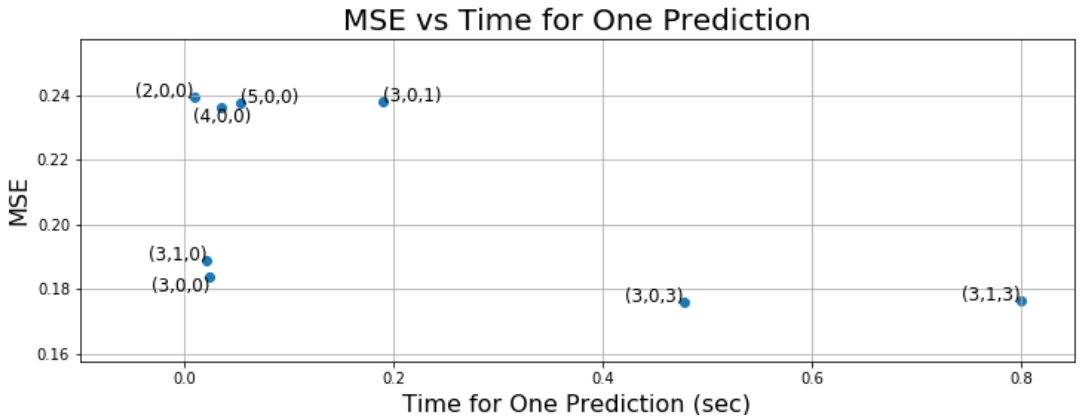


Figure 27: MSE vs Time for One Prediction.

It is also observed that differencing does not improve the performance of the model. This is to be expected as it was concluded in Section 7.2.2 that almost all of the time-series are stationary. Hence differencing is not needed.

It is also seen that any small increase in the MA order significantly increases the complexity of the ARIMA model. This can be seen from the increase in the TOP from (3,0,0) to (3,0,1) and to (3,0,3). From the analysis, we hence choose ARIMA(3,0,0) to be the order of choice as it is accurate yet relatively computationally cheap. It is then used to make forecast on all of the general model to give a one-step forecast of the general model, $\hat{y}[n+1]$

9.5 Forecast Scaling

We perform linear regression with $(\mathbf{x}_i[n+1-l:n], \mathbf{y})$ to obtain the regressors for all of the time-series in each cluster. We then use the regressors to scale the forecast of the general model to fit each of the time-series. Doing this for all the time-series for all the clusters and we end up with a forecast for every time-series.

9.6 Out of Bounds Catch

From initial testing, it was observed that some of the forecasts made were out of bounds. All observations should be in the range [0,1]. The out of bounds forecast error affects about 0.2% of all forecasts. It is most likely caused by outliers in the data.

To solve this problem, we check every forecast to make sure it is within the bounds of acceptable values. If not, we will instead use the ARIMA model to replace the faulty forecast.

9.7 Rolling Error

After some initial testing, the scaling of the forecasts using a rolling error does not improve the forecasts. Even after we introduce a multiplier, η to reduce the weight of the rolling error:

$$\hat{x}_i[n+1] = s_i[n+1]\hat{y}[n+1] + d_i[n+1] + \eta \cdot e_i[n+1]$$

and tested multiple values of r for the calculation of e_i:

$$e_i[n+1] = \frac{1}{r}\sum_{j=0}^{r-1}\Big(x_i[n-j] - \hat{x}_i[n-j]\Big)$$

The error term still does not improve the forecasts. It was deduced that the rolling error term introduces unwanted artifacts in the forecasts causing it to deviate from the actual value. It was then in our best interest to discard the error term in our scaling calculation.

10 Validation

In this section, we will discuss the frameworks used to measure the performance of our proposed framework and compare it to other framework. We will measure the performance of each of the frameworks in terms of three criteria: accuracy, precision and computational complexity. Per previously discussed, we will use MSE to measure and compare the accuracy of different forecasting frameworks.

To measure precision, we will need to look at the standard deviation of the squared error. We can also plot the distribution of the error using histograms to provide an illustrative view of the spread of the squared error.

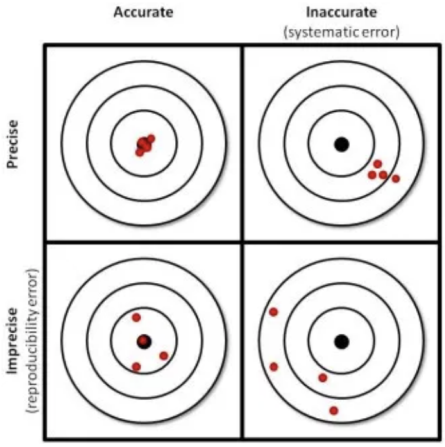


Figure 28: Accuracy vs precision. Accuracy refers to how close the forecasted values are to the actual values. Precision is a measure for the spread of the forecasting error. (Miller 2015)

Finally, to measure the computational complexity of each of the frameworks, we will measure the TOP. The other forecasting frameworks which the proposed framework is compared to are also discussed in this section.

10.1 Naive Framework

We define the naive framework as the one-step time delayed version of the time-series:

$$\hat{x}[t] = x[t-1] \tag{21}$$

The naive framework sets a lower bound performance benchmark. Any proposed framework which are less accurate than the naive framework needs to be reevaluated. The naive framework also sets a lower bound on the computation cost i.e. no other framework should theoretically have a lower computational cost than the naive framework.

10.2 Brute Force Framework

Per previously discussed, having one forecasting model for each time-series is computationally expensive and is often unnecessary. For validation purposes, we will create one ARIMA model for each time-series. This framework sets an upper bound benchmark for computational complexity. Any proposed framework which has a higher computational complexity than the brute force framework needs to be reevaluated. This framework also sets an upper limit for accuracy in which proposed frameworks should try to achieve.

10.3 Simplified Tuor Framework

This framework is a simplified version of the Tuor framework where we neglect the adaptive transmission and evolutionary clustering. We will assume that all of the measurements at the local node are fully transmitted to the control node and that the time-series are clustered together independent at each time-step. The clustering is done on the last w observations using the K-means clustering algorithm, where w is empirically chosen to be 3 samples. Euclidean distance is used as the distance measure. The number of clusters, K is kept at 100, which is the same as the number chosen in the proposed correlation-based framework.

The general model for this framework is computed using the aggregated mean of all of the time-series at each time step i.e.

$$\mathbf{y}_k[n] = \sum_{i \in \mathbf{C}_k} \mathbf{x}_i$$

The forecasts for the general model is made using an ARIMA model and finally the forecasts of each individual machine is scaled using a rolling error term i.e.

$$\hat{x}_i[n+1] = \hat{y}[n+1] + e_i[n+1]$$

where e_k is the mean past distance from the centroid to the each time-series:

$$\begin{aligned} e_k[n] &= \frac{1}{r}\sum_{i=1}^{r} e_k[n-i] \\ &= \frac{1}{r}\sum_{i=1}^{r} (x_k[n-i] - y_k[n-i]) \end{aligned}$$

r is the rolling error window in which is empirically chosen to be 1.

10.4 Validation Methodology

The validation tests are run on my personal computer with the following specs:

- Computer model: MacBook Pro (15-inch, 2017)
- Processor: 2.9 GHz Quad-Core Intel Core i7
- Memory: 16 GB 2133 MHz LPDDR3

The environment used was Jupyter Notebook 6.0.2.
All four frameworks were used to make forecasts for four segments of 100 time-steps:

- $\mathbf{x}[5 * 288 : 5 * 288 + 100]$
- $\mathbf{x}[10 * 288 : 5 * 288 + 100]$
- $\mathbf{x}[15 * 288 : 5 * 288 + 100]$
- $\mathbf{x}[20 * 288 : 5 * 288 + 100]$

11 Results

The performance of each of the framework is given below:

Framework	MSE	Error Standard Deviation	TOP (sec)
Naive	0.000949	0.003338	1.381495e-08
Brute Force	0.000876	0.003024	1.532633e-01
Correlation	0.001867	0.014987	1.625666e-02
Simplified Tuor	0.002165	0.007740	2.273263e-02

Table 4: Performance of each of the tested framework.

It came at no surprise that the brute force framework is the most computationally expensive while the naive framework is the least computationally expensive. It was also expected that the brute force framework has the least MSE. However, it is observed that the brute force framework only gave minor improvements in accuracy and precision as compared to the naive framework. This suggests that the ARIMA forecasting model is not powerful enough to accurately forecast the time-series of this particular data set. The simplified Tuor framework and the correlation has similar performance in its accuracy and computational complexity. However, it is observed that the standard deviation of the error of the correlation-based framework is much higher. Besides that, it is also observed that both the correlation-based method and the simplified Tuor framework failed to give a lower MSE than the naive framework. The plot of MSE vs TOP is given in Figure 29.

Forecasts of five different segments using different frameworks are plotted in Figure 31. Despite the difference in MSE and standard deviation of error in each of the framework, it is observed that all of the frameworks track the actual value well. The large error of the simplified Tuor framework at the beginning of each segment is to be expected as the framework relies on past distances to track the actual value. It is also observed that the correlation-based framework experiences random spikes in error.

The distribution of the error of each of the framework is plotted in Figure 30. It is observed that most of the error of each of the method is concentrated close to zero. However, it is observed that the correlation-based framework has the biggest standard deviation among all the frameworks followed by the simplified Tuor framework.

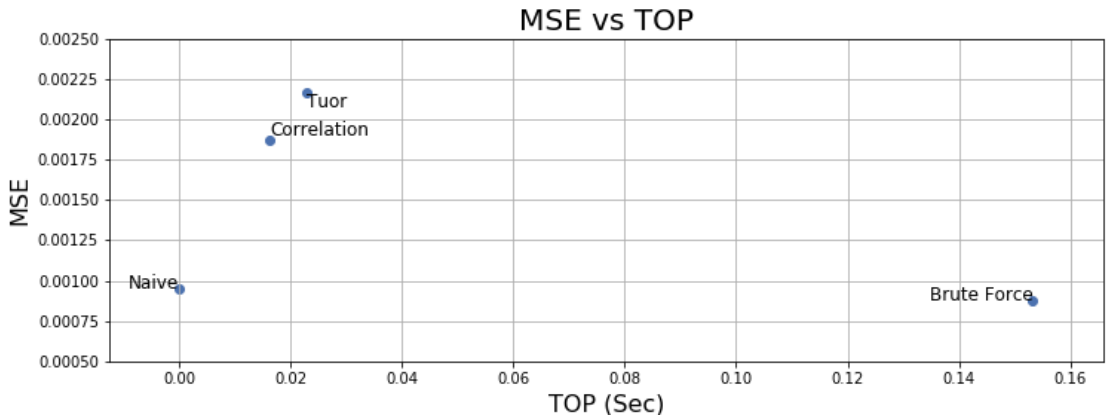


Figure 29: MSE vs TOP of different frameworks.

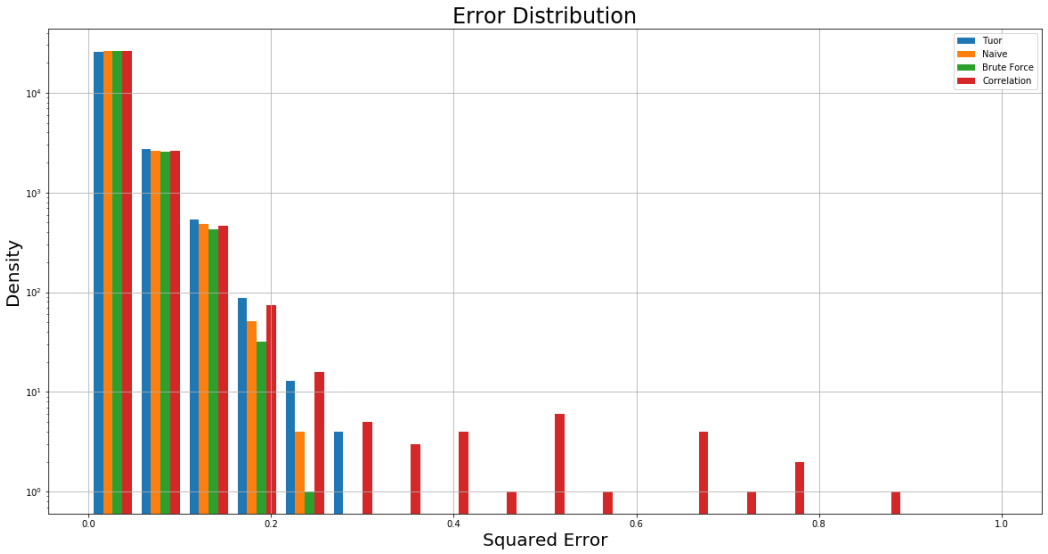


Figure 30: Error distribution of each framework. Note that the y-axis is in log scale.

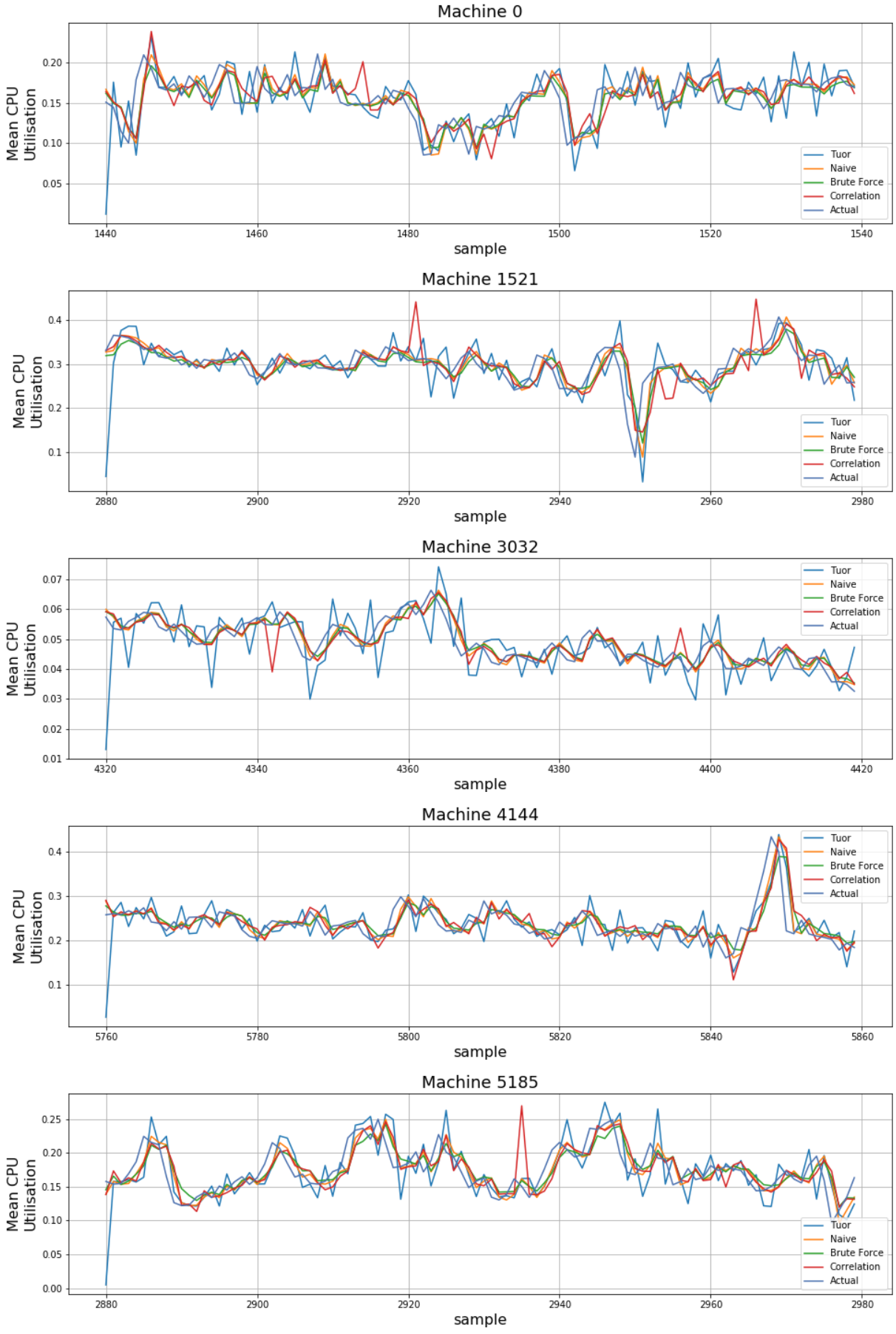


Figure 31: Forecast using different frameworks.

12 Evaluation

For any forecasting model, we usually have a trade-off between computational complexity and accuracy. A higher computational normally results in higher accuracy. Ideally we would want high accuracy with low computational complexity. Hence any computationally expensive models with low accuracy requires reevaluation or are neglected completely as they are not an efficient use of resources.

From our validation tests, it is observed that the brute force framework only gave minor improvements in accuracy and precision as compared to the naive framework. This suggests that the ARIMA forecasting model is not adequate to forecast the data set. The simplified Tuor framework failed to outperform the naive framework in terms of accuracy and precision despite having higher computational complexity. This might be caused by the difference between the simplified version and the full version of the framework.

It is also observed that the correlation-based framework managed to track the actual value well. However, it also failed to give better accuracy and precision than the naive framework even though it is higher in computational complexity. These observations suggests that the correlation-based framework is feasible but it requires additional fine-tuning before its accuracy and precision justifies its computational complexity. There are a few improvements in which we can implement to improve the accuracy and precision of the framework:

1. **Replace ARIMA forecasting models with RNN forecasting models**
   As previously mentioned, the ARIMA forecasting model makes forecasts by exploiting the linear relationship between lagged values of the time-series. It is however, unable to capture any non-linear relationships in the time-series. A recurrent neural network (RNN) despite having a much higher computational complexity, is able to capture any non-linear relationship of the time-series.

2. **Including extra parameters for forecasting**
   As mentioned in Section 7.1, it is often unreliable to make forecast on a time-series by merely relying on the lagged values of the time-series, especially when making long-term forecast. To improve the accuracy of our forecasts, we can considering feeding additional parameters to the forecasting model.

3. **Implementation of evolutionary clustering**
   For initial testing purposes, evolutionary clustering is not implemented in the proposed framework. The purpose of evolutionary clustering is to provide a noise rejection mechanism in the framework. For the simplified Tuor framework, since we consider the past few observations for clustering and that we use the Euclidean distance as the distance measure, we are essentially using a form of "rolling mean" to cluster the time-series. There is hence a weak form of rejection mechanism in the framework.

   In the correlation-based framework, we use a "rolling correlation" to cluster the time-series. As a reminder, Pearson correlation is a shape based distance measure which considers the entire time-series as a whole instead of considering each point individually. This makes the rolling correlation clustering very susceptible to noise. Any noise in the time-series will completely change the shape of the segment and causing the time-series to be clustered in the wrong cluster. This in turn results in inaccurate forecasts. This also explains the spikes in large error in the correlation-based framework.

4. **Improved computation of general model**
   For this project, we used one of the time-series in each cluster as the general time-series to generalise all of the time-series in the cluster. A more complete approach would be to compute an artificial time-series which maximises the correlation between itself and all of the time-series in the cluster. This approach potentially will give a better generalisation of all of the time-series in each cluster hence resulting in better forecasts.

Despite the correlation-based framework giving a lower accuracy and precision, this project succeeded in proving that the idea behind a correlation-based forecasting framework is founded. The proposed correlation-based framework did manage to lower the computational complexity by a large margin as compared to the brute force framework. From Figure 31, it is observed that the correlation-based framework managed to capture the dynamics of the time-series despite having spikes in large error. This project however, also proved that the framework requires additional fine-tuning using the some of the proposed solutions mentioned above.

It should also be noted that the correlation-based framework was tested on a set of time-series which are highly correlated to one another. Consequently, the framework should fall off in accuracy if the time-series are not as strongly correlated. The rate in which the correlation-based framework decreases with correlation should also be benchmarked once a more complete version of the framework is finalised.

13 Conclusion and Future Work

In this project, we proposed a correlation-based framework for cloud CPU utilisation. The newly proposed framework is based on the framework proposed in the paper Online Collection and Forecasting of Resource Utilization in Large-Scale Distributed Systems (Tuor et al. 2019) which consists of four parts: time-series clustering, computation of a general model, temporal forecasting and scaling.

The newly proposed framework is benchmarked against three other existing frameworks: the brute force framework, the naive framework and a simplified version of the framework proposed in (Tuor et al. 2019). Validation tests were done on a subset of the data set in which the time-series included are strongly correlated between one another. The tests showed that despite the the correlation based framework underperformed the naive framework, it managed to capture the overall dynamics of each time-series. We hence conclude that a correlation based framework is feasible solution for online cloud resource forecasting. However, the framework requires additional fine-tuning before the framework can be useful in a sense that its accuracy and precision justifies its computational complexity. Future work includes implementing some of the improvements mentioned in Section 12 and benchmarking the performance against the full version of the framework proposed in (Tuor et al. 2019). Since the correlation based method is designed on a basis that the correlation between different time-series is strong, the performance of the framework should also be benchmarked against the correlation between the time-series to be forecasted.

14 User Guide

The code for this project can be found in https://github.com/yhl116/Machine-Learning-to-Track-Cloud-Computing. The directory is arranged as follows:

- All the analysis can be found in the `analysis` folder
- All of the implementation including hyperparameters tuning are done in the files in `implementation`
- Validation is performed in the `implementation//validation` file
- All the frameworks are compiled in the file `implementation//all_framework.py`
- The data set is not included in the GitHub repository as it is too huge to be included

The guide to use each of the framework is given below:

1. **Correlation based framework**
   `corr_predictions(cpu_data, window = 5, number_of_cluster = 100, start_time = 288, end_time = None, rolling_error_window = 0)`
   - Description: High level function for making correlation based single-step forecasts. The function will make forecasts from `start_time` to `end_time - 1` inclusive
   - `cpu_data`: pandas DataFrame with CPU index as columns and time-step as rows. The function will make forecasts for all the machines included in this parameter
   - `window`: The rolling correlation window size
   - `number_of_cluster`: Number of clusters used for clustering
   - `rolling_error_window`: Number of past error to be used to scale the final forecasts
   - `returns`: pandas DataFrame with all the required forecasts
2. **Simplified Tuor framework**
   `mse_predictions(cpu_data, number_of_cluster = 100, start_time = 288, end_time = None, past_error_range = 1, rolling_cluster_window = 3)`
   - Description: High level function for making MSE based single-step forecasts. The function will make forecasts from `start_time` to `end_time - 1` inclusive
   - `cpu_data`: pandas DataFrame with CPU index as columns and time-step as rows. The function will make forecasts for all the machines included in this parameter
   - `rolling_cluster_window`: The size of the past segment in which the clustering algorithm should consider
   - `number_of_cluster`: Number of clusters used for clustering
   - `past_error_range`: Number of past error to be used to scale the forecasts
   - `returns`: pandas DataFrame with all the required forecasts
3. **Brute force framework**
   `one_timeseries_one_model(cpu_data, start_time, end_time):`
   - Description: High level function for using one ARIMA model for each machine to make forecasts. The function will make forecasts from `start_time` to `end_time - 1` inclusive
   - `cpu_data`: pandas DataFrame with CPU index as columns and time-step as rows. The function will make forecasts for all the machines included in this parameter
   - `returns`: pandas DataFrame with all the required forecasts
4. **Naive framework**
   `lagged_timeseries(cpu_data, start_time, end_time):`
   - Description: High level function for using the lagged value as the forecasted value. The function will make forecasts from `start_time` to `end_time - 1` inclusive
   - `cpu_data`: pandas DataFrame with CPU index as columns and time-step as rows. The function will make forecasts for all the machines included in this parameter
   - `returns`: pandas DataFrame with all the required forecasts

References

Aghabozorgi, S., Shirkhorshidi, A. S. & Wah, T. Y. (2015), 'Time-series clustering – a decade review'.

Cai, B., Zhang, R., Zhao, L. & Li, K. (2018), *Less Provisioning: A Fine-grained Resource Scaling Engine for Long-running Services with Tail Latency Guarantees.*

CAI, H., SHEN, S., LIN, Q., LI, X. & XIAO, H. (2019), 'Predicting the energy consumption of residential buildings for regional electricity supply-side and demand-side management'.

Chakrabarti, D., Kumar, R. & Tomkins, A. (2006), Evolutionary clustering, *in* 'Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', KDD '06, Association for Computing Machinery, New York, NY, USA, p. 554–560.
URL: *https://doi.org/10.1145/1150402.1150467*

Chatfield, C. (2004), 'The analysis of time series : an introduction'. ID: 44IMP$_A$LMA$_D$S5153864560001591.

Crippen, G. (1978), 'Rapid calculation of coordinates from distance matrices', *Journal of Computational Physics - J COMPUT PHYS* **26**.

Grechanik, M., Luo, Q., Poshyvanyk, D. & Porter, A. (2016), *Enhancing Rules For Cloud Resource Provisioning Via Learned Software Performance Models.*

Hunt, X. (2017), 'Online learning: Machine learning's secret for big data'.
URL: *https://blogs.sas.com/content/subconsciousmusings/2017/10/17/online-learning-machine-learnings-secret-big-data/*

Kotsakos, D., Trajcevski, G., Gunopulos, D. & Aggarwal, C. C. (2014), *Time-Series Data Clustering.*

Livadiotis, G. & McComas, D. J. (2013), 'Fitting method based on correlation maximization: Applications in space physics', *Journal of Geophysical Research: Space Physics* **118**(6), 2863–2875.
URL: *https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/jgra.50304*

Miller, B. (2015), 'Accuracy vs precision'.
URL: *http://glsi.agron.iastate.edu/2015/01/18/accuracy-vs-precision/*

Nikravesh, A. Y., Ajila, S. A. & Lung, C. (2015), Towards an autonomic auto-scaling prediction system for cloud resource provisioning, pp. 35–45.

Reiss, C., Wilkes, J. & Hellerstein, J. (2014), 'Google cluster-usage traces: format + schema', (2.1).

Roelofsen, P. (2018), Time series clustering, PhD thesis.

Shchetinin, E. Y. (2019), 'Cluster-based energy consumption forecasting in smart grids', *Journal of Physics: Conference Series* **1205**, 012051.
URL: *http://dx.doi.org/10.1088/1742-6596/1205/1/012051*

Shen, H. & Chen, L. (2018), 'Resource demand misalignment: An important factor to consider for reducing resource over-provisioning in cloud datacenters', *IEEE/ACM Transactions on Networking* **26**(3), 1207–1221.

Tuor, T., Wang, S., Leung, K. K. & Ko, B. J. (2019), 'Online collection and forecasting of resource utilization in large-scale distributed systems'.