Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019

Project Title:	**Machine Learning for Cloud and Distributed Computing**
Student:	**Koral B. Hassan**
CID:	**01096803**
Course:	**4T**
Project Supervisor:	**Professor Kin K Leung**
Second Marker:	**Dr Wei Dai**

Abstract

We first look at the current state of the distributed and cloud computing market. We identify the problems the industry is facing today. We go on to justify better resource occupancy predictions via machine learning as a good solution to these problems. We use a large dataset provided by Google for our technical investigations. We conduct exploratory analysis on the dataset to determine the dynamics of the system. We then identify well-suited prediction models, implement them, and compare their performance to some baseline models.

Acknowledgements

Firstly, I thank my supervisor Dr Kin K. Leung for all his help, and for allowing me to partake in this project. He convinced me on the spot through sheer passion that this is the project I would like to do. Simply discussing ideas with him and getting exposed to his rationale has been more than enough reward.

I'm also grateful to Tiffany Tuor who helped me through some technical problems and gave me advice. Having her camaraderie has helped me through uncertainty.

I would like to thank my friends, not just for now but for years past. They have kept me sane.

Another essential mention is my girlfriend, Anam Balbolia. Sharing my life with her, good and bad, brings me more joy than any other.

My final thanks go to my family, the timeless constant of my life. I cannot say more than that they have been the bow to my arrow.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this paper, we discuss the state of the cloud and distributed computing industry. We illustrate that accurate resource utilisation predictions are vital, and that machine learning is a promising solution. We then explore the nature of the particular prediction problem at hand, using a dataset provided by Google. We design and implement a few machine learning models that are well-suited to the dataset. After evaluating their performances through comparisons, we draw some conclusions on the best approach.

1.1 Definitions

1.1.1 Distributed Computing

Computing network technologies have improved in leaps and bounds in the last 20 years. These improvements have allowed the exploitation of distributed computing systems for uses such as parallel processing, redundancy and low latency among others.

Distributed computing can be defined as 'the use of a distributed system to solve a single large problem by breaking it down into several tasks where each task is computed in the individual computers of the distributed system'. [14] The computers in the system communicate through a network. They try to tackle a common goal by using their local *resources*.

1.1.2 Cloud Computing

Cloud computing can be defined as 'the delivery of computing services – servers, storage, databases, networking, software, analytics, intelligence and more – over the Internet ("the cloud")'. [15] Figure 1.1 has an overview of the current cloud computing markeyplace.

In cloud computing, users typically only pay for the resources they actually use. Not having to pay for building and running specific infrastructure personally leads to great flexibility and scalability, both economically and logistically. This enables very quick innovation, especially in web services.

Cloud infrastructure is often distributed in nature due to price-to-performance benefits over monolithic centralised systems.

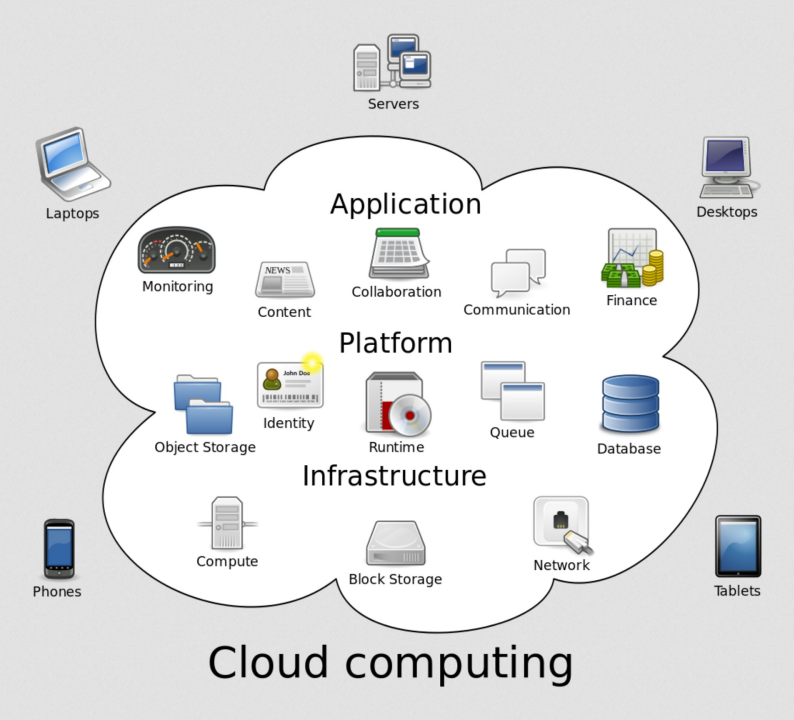

Figure 1.1: Diagram showing overview of cloud computing. [2]

1.1.3 Machine Learning

Machine learning is a branch of Artificial Intelligence (AI) where computer systems use algorithms and statistical models to progressively improve their performance on a specific task. All machine learning models make predictions or decisions without being explicitly programmed to do the task. The algorithms and statistical models need to be fed representative data in order to improve. This dataset is called 'training data'. An example of this self-improvement can be seen in Figure 1.2.

With the recent abundance in data collection that resulted from the so called 'digital revolution', machine learning has had an explosion in both popularity and successful applicability. It finds common use in modeling the complicated relationships among various components and performance measurements of computer systems, among other things.

1.2 Project Specification

This project is part of the US-UK ITA project. [16] Its purpose can be defined as follows:

> Delevelop machine-learning techniques to monitor and predict resource utilisation and availability on a very large number of computers in cloud computing and distributed environments (*e.g.* tens of thousands of servers).

This will increase energy efficiency and reduce opportunity costs, by increasing utilisation and reducing idle time.

The data we have to transmit and process is what is known as a time series, meaning it is a series of values of a quantity obtained at successive times, often with equal intervals between them. Furthermore, it is a forecasting problem, meaning we are tasked

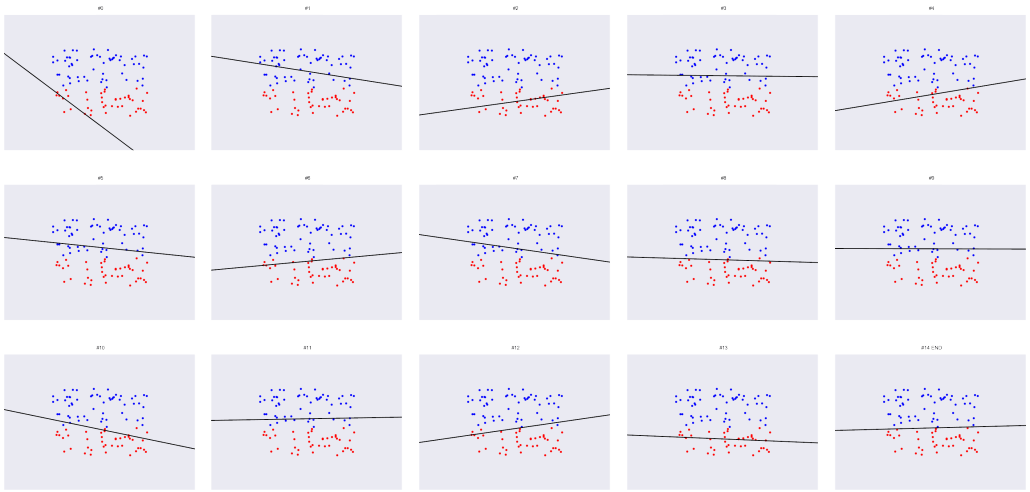

Figure 1.2: A machine learning algorithm called Perceptron learning to differentiate blue data points from red data points. [3]

with predicting future values. The most recent and successful breakthroughs in time series forecasting have come from the domain of machine learning. There are a number of reasons why it is a particularly good fit for use in cloud computing and distributed environments.

1.3 Stages

1.3.1 Research

The project begins with a survey of existing techniques for machine learning available and various underlying models reported in the literature. We contrast this against the problems faced in industry and the challenges of the machine learning approach. These are in turn presented in Chapter 2. Using this information, we outline the expected and desirable outcomes of the project in Chapter **??**.

1.3.2 Empirical Investigations

Using real datasets (*e.g.* Google servers), we identify patterns that may help us in forecasting, then design models around these patterns. All of of this is showcased in Chapter 3. In Chapter 4, the models are implemented to track the resource usage of many computer servers as a way to predict resource occupancy and workload on the computers in the near future. The methods for validating the effectiveness of the models are outlined in Chapter 5.

1.3.3 Findings

Several success metrics are presented in Chapter 6 and used for comparison in Chapter 7. In Chapter 8, we draw some conclusions about utilising machine learning models for the purpose of resource utilisation forecasts. Finally, the user guide for reproducing results can be found in Chapter 9.

Chapter 2

Background

2.1 Problem

2.1.1 Energy Efficiency

The EU's priorities are a good demonstration of the importance of energy efficiency.It is a focal point of their Europe 2020 Strategy. [17, 18] In the long term, it is one of the most cost effective ways to achieve energy goals and reduce environmental impact. [19, 18] Information and Communications Technology (ICT) is recognized as an important instrument for achieving these goals. [20, 18] However, ICT is also a big energy consumer because of equipment manufacture, use, and disposal [21, 18], which became a key issue of the Digital Agenda for Europe in 2010. [22, 18]

Cloud computing makes up a large portion of the total ICT energy consumption in providing elastic and on-demand ICT infrastructures, [23, 18] as demonstrated by the data from 2008 [24] in Figure 2.1 (see data centers). The energy-saving implications of utilising current cloud computing systems more efficiently are enormous.

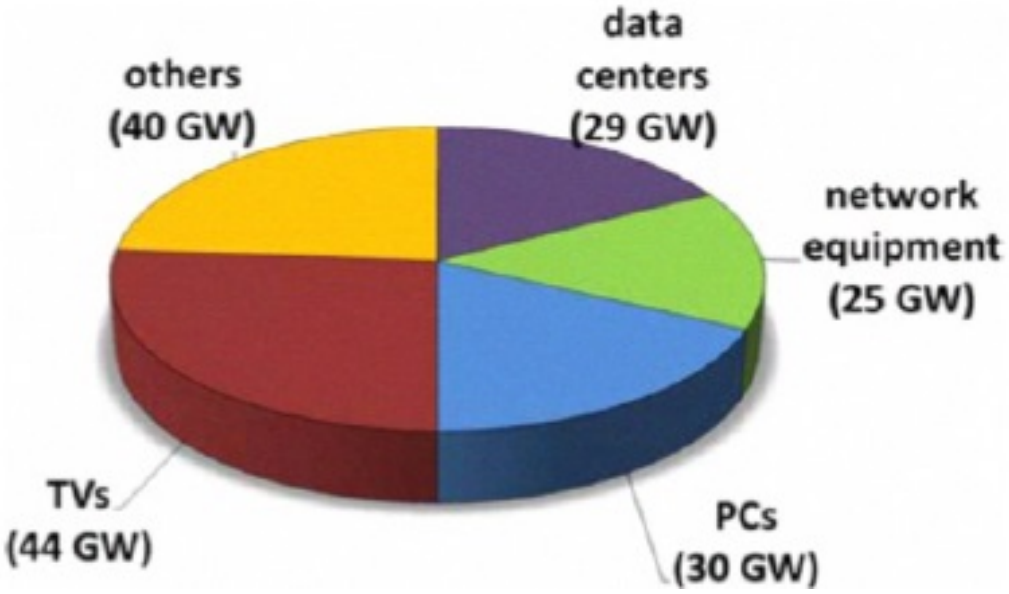

Figure 2.1: Annual electricity consumption of ICT equipment by category. [4]

2.1.2 Opportunity Cost

For the most part, cloud computing is a private enterprise with commercial aspirations. The main motivator of building cloud infrastructure is to generate profit. Profit is generated by charging the customer for utilising resources. Idle resources do not generate profit. Errors in resource occupancy predictions result in underutilisation. It follows that inaccurate occupancy predictions cause companies to miss opportunities for profit.

It is common for average server utilisation levels to be lower than 20%, and even hyper-scale Cloud Computing companies often have average utilisations around only 40%. [25] While inaccurate predictions are not the only cause of these low percentages, they show the wasted potential quite well.

2.1.3 Manual Labour

Amazon Web Services easily owns more than $1\,000\,000$ servers. [26, 27] It employs around $23\,000$ people in total. [28] The company has a server-to-employee ratio of roughly $43 : 1$, which is visualised in Figure 2.2. This is a realistic number compared to other tech giants, [29] and it is only going to increase with time.

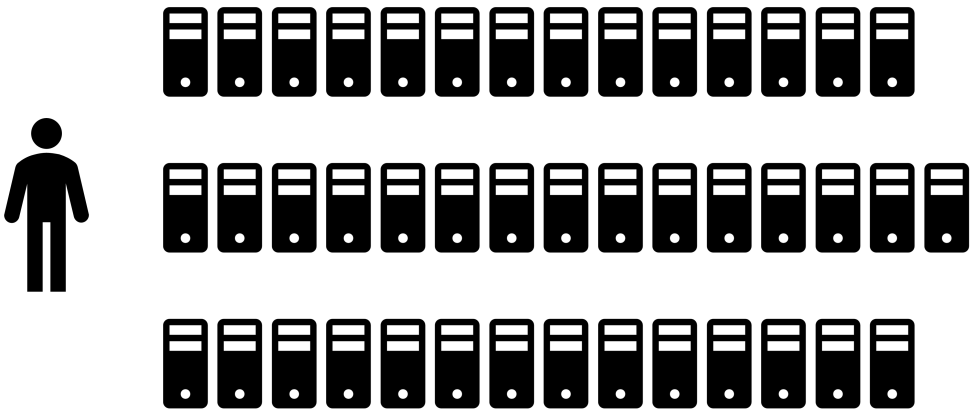

Figure 2.2: A $43 : 1$ server to employee ratio. [4]

While utilising servers more efficiently certainly is profitable, servers cost a lot less than highly-skilled workers in the long run. On top of this, it is very difficult for humans to predict occupancies reliably and consistently. To ensure scalability, the occupancy prediction process should be highly automated; and if possible should require very little maintenance and oversight.

2.1.4 Time Constraints

Even if humans could match the reliability and consistency of computer algorithms, they would still not be able to match their decision-making speed. In certain applications, this might not be a major concern; but the highly dynamic and online nature of cloud infrastructure requires incredibly quick response times. This is beyond the limits of normal human cognitive ability.

2.1.5 Hardware Constraints

Often, distributed computing systems use a master/slave architecture such as the one seen in Figure 2.3. In such a case, there will be a central node (the *master*) which is in charge of scheduling new jobs to servers. For a given job, the master must decide which node (*slave*) will have the appropriate amount of resources available. This requires the master to possess a forecast of future availability, which in turn requires information about the current and past availabilities of the slaves. A problem arises in trying to obtain and process this information. The slaves have access to the information regarding their current availability, but they do not store information regarding their past availability.

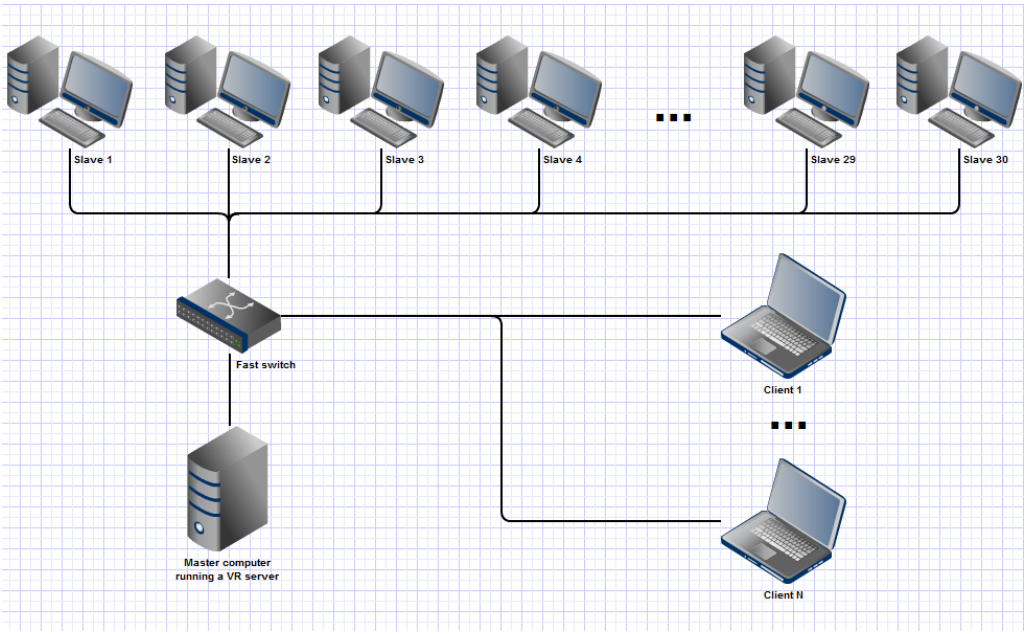

Figure 2.3: Exemplar distributed computing system with a master/slave architecture. [5]

One option is for each node to forecast its own future availability, and transmit this information alone to the master. Each slave would have access only to the information at its own node, which may or may not undermine its forecasts.

Another option would be for each slave to always transmit their current availability to the master in real time, even though this arrangement is usually quite bandwidth-consuming.

In either case, it would be too complex to process all of the data. The system would be wasting too much computational power just on forecasting. Overall, it is clear that the large volume of data must somehow be condensed in order to ease the processing of it.

2.2 Machine Learning

2.2.1 History

In 1950, Alan Turing described a machine that could learn and become artificially intelligent. [30] Through the 1950's various new models were conceived and computers could already play games such as checkers. [31] The 1960's saw the introduction of Bayesian

methods. The field stagnated in the 1970's, following a disillusionment about the limitations of machine learning. This period came to be known as the AI Winter. However the field picked up again in the 1980's with topics such as backpropogation, reinforcement learning and Artificial Neural Networks (ANN) being studied further.

Machine learning methods started to really demonstrate competency in the 1990's, with IBM's Deep Blue beating the world chess champion [32] being seen as a huge milestone. The focus of machine learning shifted from a knowledge-driven approach to a data-driven approach. Unsupervised machine learning methods became widespread in the 2000's. During the 2010's, deep learning became feasible for commercial applications due to the abundance of data. This led to and explosion in the popularity of machine learning methods because for the first time they became highly competent and profitable in a variety of real-world applications. This can also be observed in Figure 2.4. Computers made huge strides in beating humans at games, labelling images, recognising faces and more.

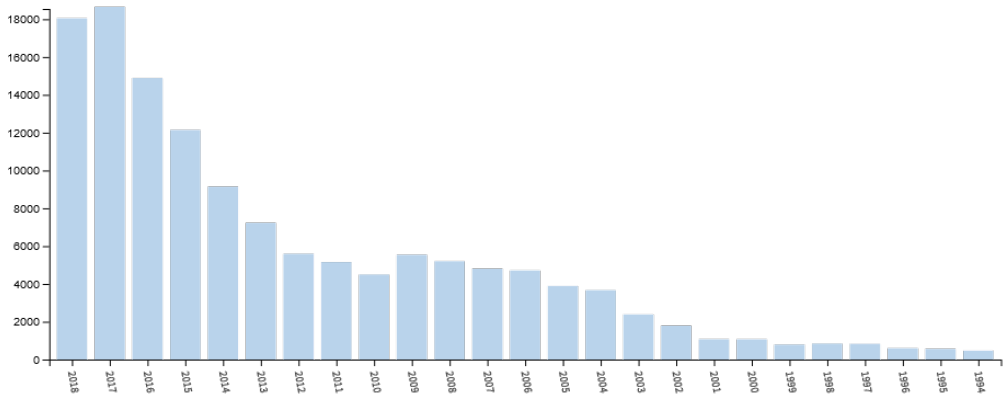

Figure 2.4: Number of times 'machine learning' has appeared in publications from 1994 to 2018. [6]

2.2.2 Strengths

Once a machine learning model is developed and trained, it is be fully capable of making correct and informed predictions within a predictable error margin. The developer does not even need to justify or explicitly program, why the model arrives at a particular prediction. This eliminates a lot of the human errors, since no human needs to understand or be skilled at forecasting for the algorithm itself to be skilled at forecasting. This is an important factor since it implies our forecasting won't be relying on any manual labour.

Another advantage is that while a model gains experience through consuming data, it automatically self-improves. So even if a machine learning model is initially inferior to the status quo (as it invariably will be), in time, with very little supervision or maintenance, it will improve. This also means that it can adapt to new patterns in data without any intervention.

A good example of these principles is AlphaGo, and its sibling AlphaZero. They are computer programs that play Go and chess respectively. What's more, they are better

at these games than any other player in history, implying they have long since beaten their developers. What perhaps drives the point home is that AlphaZero was not given any prior knowledge of the game other than its rules. It learned tactics and strategies by itself, not from any human. [7] Figure 2.5 illustrates AlphaZero beating the previous world champion computer programs in several different games.

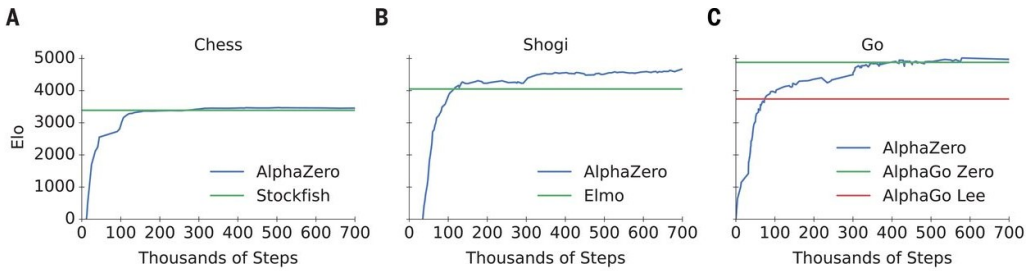

Figure 2.5: AlphaZero training for 700,000 steps. [7]

We have previously mentioned that the nodes of our distributed system will often be generating too much data, and that it will have to somehow be condensed. This too, is a well suited problem for machine learning. Feature extraction is a common practice in machine learning. The data is transformed in such a way as to extract the most relevant information and minimise losses of insight. Indeed, an entire seperate machine learning model can be trained on how to do this best.

2.2.3 Weaknesses

Machine learning models initially need a lot of data and possibly time to ramp up their competency. While this is usually worth it in the long run, companies must have long time horizon to see the benefits of this strategy. Fortunately, in our use case it is very easy to generate massive amounts of data; and the technology giants that own cloud infrastructures have a long enough time horizon for spending time training their models.

Machine learning models do not require domain expertise from developers to become experts themselves. However, their decision-making process can often seem very opaque and unintuitive humans. This makes it very difficult to justify, rationalise, or otherwise explain why an action was taken when it was an AI that made the decision. From a moral perspective, it is very difficult to hold AI accountable when mistakes are made. Nor is it easy to ensure the same mistake won't happen again. In the case of resource utilisation, the losses are relatively limited to logistic ineffincies which are unlikely to have any devastating effect so this should be managable; but it should still be kept in mind during ethical planning.

2.2.4 Batch Learning versus Incremental Learning

Batch learning, also called offline learning, is the historical and practical default when it comes to training algorithms. The training dataset is collected in its entirety before any training begins. Then, the model is trained on all of the training dataset during one continuous session where no predictions are yet made by the algorithm. At the end of the process the algorithm is considered fully trained and remains static moving forward. The

learning stage is done and the algorithm is considered to have reached its peak potential. It will now start making predictions.

The counter-part of offline learning is online learning. Incremental learning refers to 'online learning strategies which work with limited memory resources'. [33] For our purposes we will use the terms online learning and batch learning inter-changeably, and assume limited memory resources. Incremental learning involves continuously using the input data to extend the model's existing knowledge. Even as the model makes predictions and finds out whether they are correct or not, it will tune itself.

Incremental learning is generally considered to converge to minima slower. This makes sense considering the fact that an online algorithm only has access to its current state and one datapoint to decide on its next improvement step; whereas an offline algorithm also has access to all the other data it has encountered. In other words, it is much easier to decide on an optimal algorithm when the model's memory has access to all datapoints simultaneously.

On the other hand, incremental learning allows the model to adapt to patterns changing over time. Offline models may get outdated as the facts they have learned about the data change. To improve them would require starting from scratch. The challenge with online learning is to ensure that existing knowledge is not forgotten while incorporating new knowledge. This negative effect is known as catastrophic interference, and precautions must be taken for minimising the damages of it. [34] These precautions include built-in parameters, assumptions and persistent representations of the old training data.

Incremental learning is most useful when the data is not available as a batch. For example;

- when the training data only becomes available gradually over time,
- when the dataset is generated as a function of time,
- when the prediction has to be made in a finite amount of time because its value is very time-sensitive,
- or when the dataset does not fit in system memory.

If the aim is to construct a machine learning algorithm that adapts in real time, training speed becomes a much bigger concern; to the point where simpler models may be preferred. [35] Time series forecasting is often done using online learning models due to the dynamic nature of the patterns.

Many of the traditional machine algorithms inherently support online learning, or can easily be adapted to facilitate this.

Figure 2.6 shows a model that incrementally learns spam keywords by using the titles of new emails identified as spam. The Sliding Window stores words that have been encountered within the last 30 days. If a keyword is already in the Sliding Window its frequency is incremented by 1. [8] This way, the model efficiently keeps track of the frequency of words used in spam emails. This information can then be used identify other spam emails.

2.2.5 Supervision

The primary categories of learning algorithms relate to supervision. While there are other types of learning (*i.e.* reinforcement learning), they are not relevant to this project. The categories of supervision are;

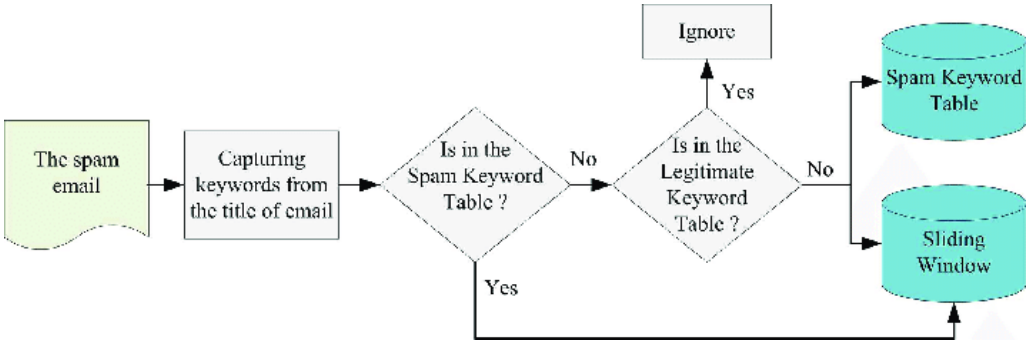

Figure 2.6: The incremental learning of spam keywords of the Sliding Window. [8]

- supervised learning,
- semi-supervised learning,
- and unsupervised learning.

Which category an algorithm falls into depends entirely on the kind of data it needs to be trained on. More specifically, it depends on whether the dataset is *labelled* or not. If an input (*e.g.* a single datapoint) has a label, its correct output is known, and vice versa. If all of the data needs to be labelled, the learning is said to require supervision. If only some of the data is labelled, the learning is semi-supervised. Learning with an entirely unlabelled dataset is called unsupervised learning. The differences are illustrated in Figure 2.7.

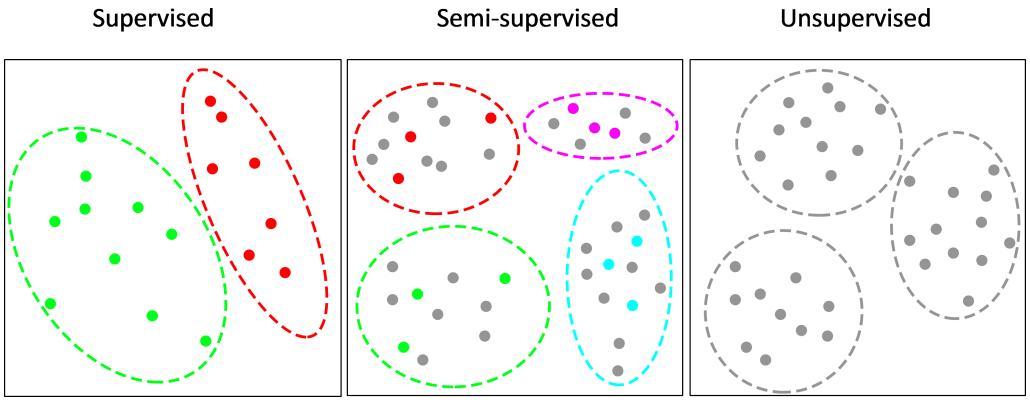

Figure 2.7: Labelled and semi-labelled classification versus unlabelled clustering.

This project will not be focusing on semi-supervised learning as it is not relevant to its use case.

Supervised Learning

All supervised learning is done on labelled datasets. For our purposes, supervised learning can be split into 2 categories. The categories depend on the nature of the labels, they are;

- regression,
- and classification.

Regressive models aim to find the mathematical relationship between inputs and outputs. Regression problems have labels that are numeric values along a continous (and often unbounded) range. An example of a regression problem would be forecasting values of shares on the stock market.

On the other hand, classification models aim to place each input in a predefined category. Classification problems have discrete labels. An example of a classification problem would be identifying the faces of a pre-defined set of people in images.

Figure 2.8 demonstrates the difference between classification and regression through 2 further examples. The labels in the first graph are indicated by the colours of the datapoints (red or blue). The labels in the second graph are indicated on the y-axis (continuous numerical value). The first model designates areas of the graph as either red or blue, which is indicated by the boundary drawn on the graph. The second model produces a linear equation that best describes the relationship between the inputs and outputs.

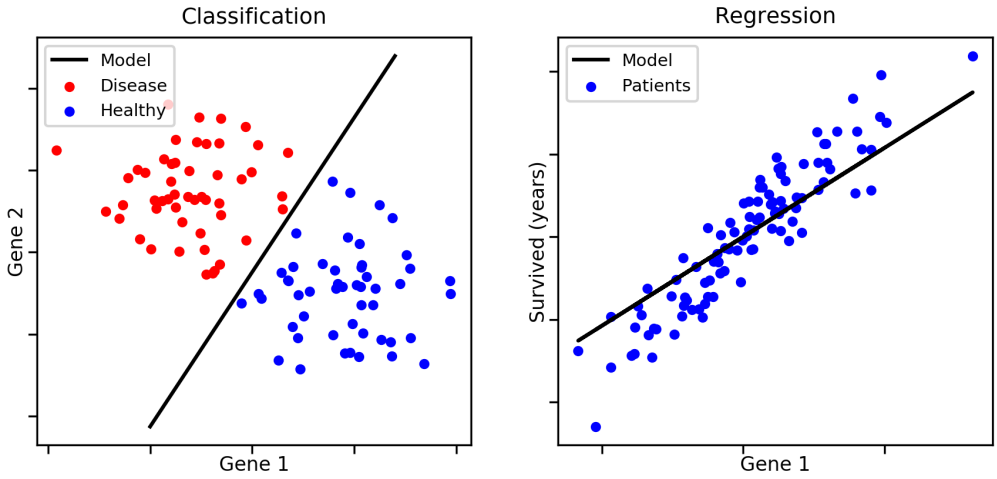

Figure 2.8: Predicting whether a patient has a disease or not versus predicting how many years a patient will survive. [9]

Unsupervised Learning

Unsupervised machine learning models are trained with unlabelled datasets. Instead comparing predictions to labels for feedback, they aim to identify commonalities in the dataset via statistical analysis. The two categories of unsupervised learning that we will study are;

- dimensionality reduction,
- and clustering.

Dimensionality reduction is the task of decreasing the number of variables being considered, [36] by transforming the datapoints into a lower dimensional format. This new set of engineered variables are called *features* and the mathematical range that they can span is called the *feature space*. The training datapoints get *projected* onto the feature space. When done successfully, dimensionality reduction can conserve most of the relevant information while removing most of the variables.

Dimensionality reduction can be useful for;

- reducing computational complexity,
- reducing required storage space,
- preventing overfitting.

The aim of clustering is to group a dataset into categories (also called *clusters*). Datapoints within one cluster should be similar to each other and dissimilar to datapoints from other clusters for a given similarity metric. The goal of clustering is similar to that of classification, except the categories are not predefined and it is not known which training datapoints belong to the which cluster.

Interestingly, clustering can also be used to reduce computational complexity since cluster centres are usually very representative of their category's characteristics. Multiple datapoints within one category can be collapsed into a single datapoint that is effectively their average.

Chapter 3

Analysis and Design

3.1 Dataset

The dataset that will be used is a trace that was produced by the Google cluster management software and systems, called 'clusterdata-2011-2'. [37, 1] The trace represents 29 day's worth of Borg cell information on a cluster of about $12\,500$ machines. The trace starts at 7pm May 1, 2011 US Eastern time. [1]

Google's Borg system is 'a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines'. [38]

'A Google cluster is a set of machines, packed into racks, and connected by a high-bandwidth cluster network. A cell is a set of machines, typically all in a single cluster, that share a common cluster-management management system that allocates work to machines. Work arrives at a cell in the form of jobs. A job is comprised of one or more tasks, each of which is accompanied by a set of resource requirements used for scheduling (packing) the tasks onto machines. Each task represents a Linux program, possibly consisting of multiple processes, to be run on a single machine.' [39] 'Resource requirements and usage data for tasks are derived from information provided by the cell's management system and the individual machines in the cell. A single usage trace typically describes several days of the workload on one of these compute cells.' [39]

The following resource usage statistics are reported for each task from each measurement period (typically lasting 5 minutes): [39]

- start time of the measurement period
- end time of the measurement period
- job ID
- task index
- machine ID
- mean CPU usage rate
- canonical memory usage
- assigned memory usage

- unmapped page cache memory usage
- total page cache memory usage
- maximum memory usage
- mean disk I/O time
- mean local disk space used
- maximum CPU usage
- maximum disk IO time
- cycles per instruction (CPI)
- memory accesses per instruction (MAI)
- sample portion
- aggregation type
- sampled CPU usage: mean CPU usage during a random 1s sample in the measurement period

We will be focusing on mean CPU usage rate and canonical memory usage in particular, henceforth simply referred to as *CPU usage* and *MEM usage*.

The known anomalies in the trace are listed in Appendix A.

3.2 Exploratory Analysis

There are exactly $12\,476$ machines in total. For both CPU usage and MEM usage, and for each machine, there exists 8351 sequential datapoints. Since each datapoint is representative of 5 minutes in reality;

$$8351 \times 5 = 41755$$
$$\frac{41755}{60} = 695.19\overline{6}$$
$$41755 - 60 \times 695 = 55\ minutes$$
$$\frac{695}{24} = 28.958\overline{3}$$
$$695 - 24 \times 28 = 23\ hours$$
$$28\ days$$

we have access to a timespan of 28 days, 23 hours, and 55 minutes. Figure 3.1 gives an idea of what the time series look like.

Before starting to train models, some statistical analysis of our dataset will be required to narrow down the focus to the most promising approaches. Namely, we must first observe how strong the *spatial and temporal correlations* are in the data. Strong spatial cross-correlation would imply we can use the values from other servers for predicting the

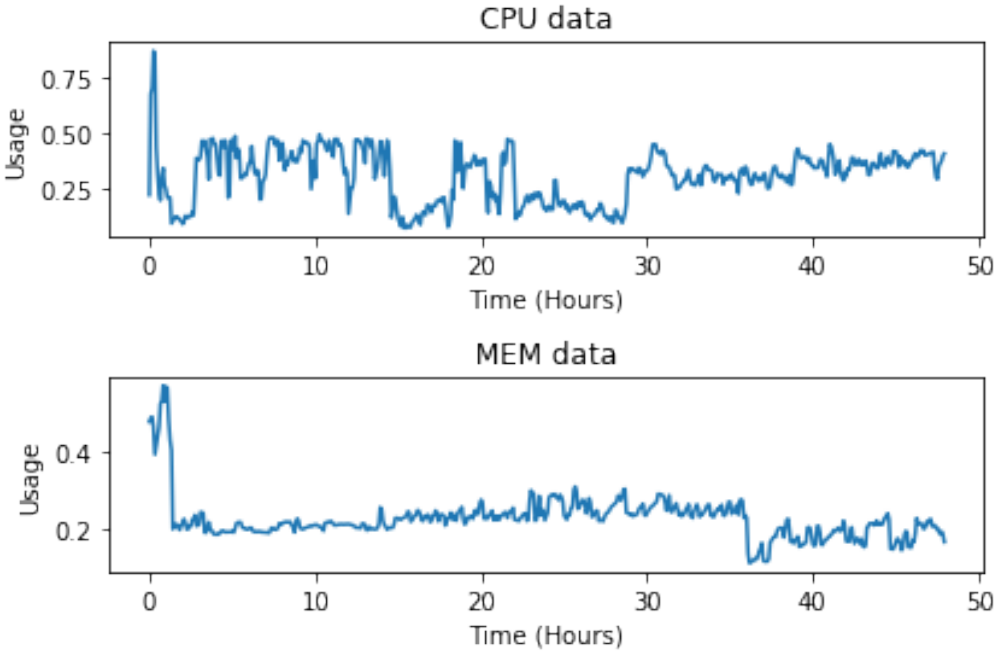

Figure 3.1: Usage data of one machine.

next value of a given server. Strong temporal auto-correlation would imply we can use the previous values from a given server to predict the next value of the same server.

We sample 10% of the dataset for our analyses. This provides us with computational efficiency while maintaining a big enough portion for statistical significance. We also want certain portions of the dataset to be previously unseen when it's time to test our models. The reasoning for this is discussed further in Chapter 5.

We compute the correlation as generally defined in signal processing texts, such that for 1-dimensional arrays x_i and x_j;

$$c_i, j[k] = (x_i * x_j)(k - N + 1) = \sum_{l=0}^{||x_i||-1} x_i[l] x_j^*[l - k + N - 1]$$

$$\forall x_i, x_j \in X,\ \forall k \in \{0, 1, ..., ||x_i|| + ||x_j|| - 2\}$$

where $N = max(||x_i||, ||x_j||)$. In this equation, k represents the time-shift of our convolution function. c is the full convolution, or in other words, a list of correlation values that represents each time-shift. In our case;

$$||x_i|| = ||x_j|| = 8351,\ \forall i, j \in \{0, 1, \ldots, 1247\}$$

$$N = 16700$$

3.2.1 Temporal Correlations

For any given time-shift, the non-overlapping parts of the signals will be zero-padded. The boundary effects of the zero padding can be seen in Figure 3.2. The bigger the overlapping area, the bigger the correlation value. In our case, we do not care about what

percentage of the signals overlap, we only care about the similarity of their patterns at that time-shift. We can get rid of the boundary effects like so:

$$C = \{c_{0,0}, c_{0,1}, \ldots, c_{1,0}, c_{1,1}, \ldots, c_{1247,1247}\}$$

$$C^{no\ effect}[k] = \frac{N \times C[k]}{N - |N - k - 1|}\ for\ k \in \{0, 1, ..., 16700\}$$

It should be noted however, that the values for time-shifts with very little overlap will be highly unreliable. In the most extreme cases we will be extrapolating off of just a single datapoint of overlap. This is to be taken into consideration when analysing the boundaries.

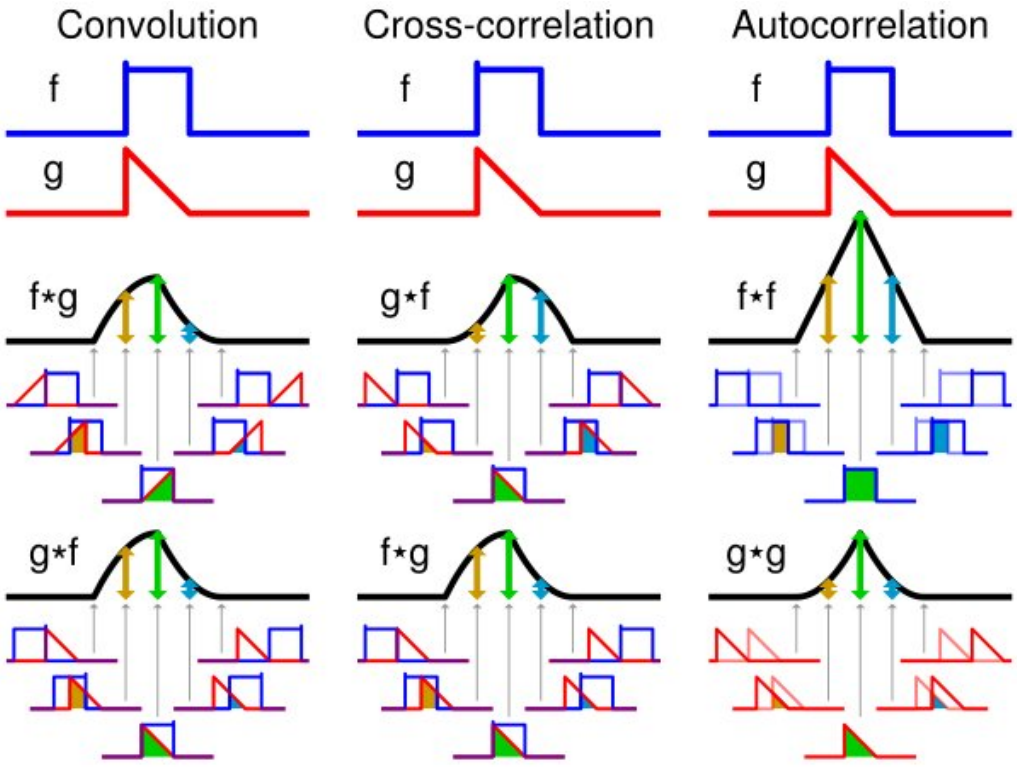

Figure 3.2: Visual interpretations of convolution and correlation.

We also want our similarity metric to be easily comparable across multiple different signals. Currently, the the bigger the magnitude of the two signals, the bigger the correlation value. We can normalise this to a range between 0 and 1 like so:

$$c_{i,j}^{norm} = \frac{c^{no_effect_{i,j}}}{\sigma_{x_i}\sigma_{x_j}N}) \ \forall i, j \in \{0, 1, \ldots, 1247\}$$

In Figure 3.3, we can see this processing taking place. We can also observe that for auto-correlation, the graphs are symmetrical around 0 time-shift. This can also be derived from the equation for convolution since;

$$(x_i * x_j)[k] = \sum_{m=-\infty}^{\infty} x[m]g[n-m] = \sum_{m=-\infty}^{\infty} x[n-m]g[m]$$

$$and\ x_i = x_j \implies (x * x)[k] \equiv (x * x)[-k]$$

We discard negative time-shifts on auto-correlation calculations from here onwards.

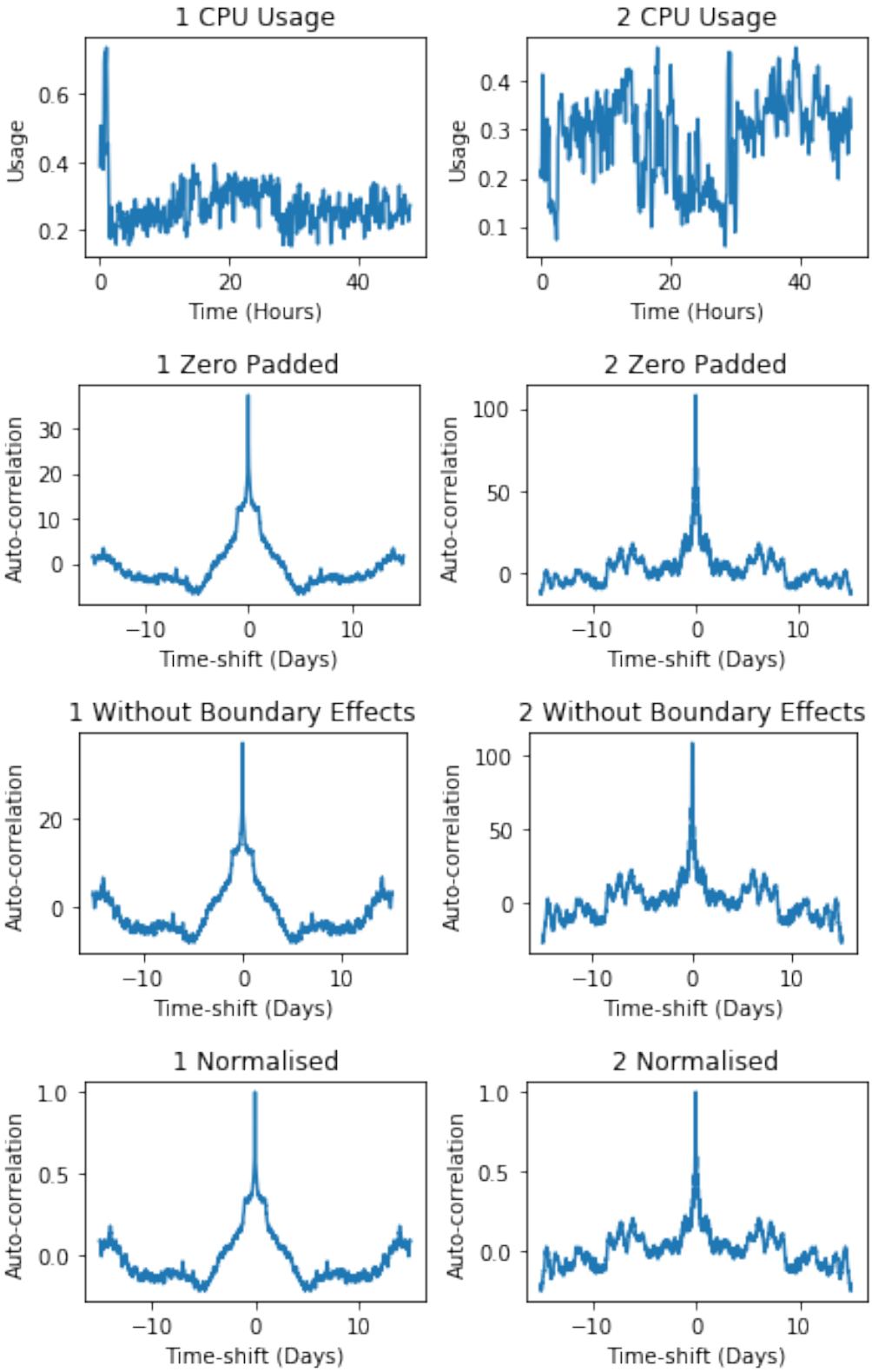

Figure 3.3: Calculating the normalised temporal correlation of 2 machines.

We take the average temporal autocorrelation of $\lfloor \frac{12476}{10} \rfloor = 1247$ machines for each time-shift.

$$c^{avg} = \sum_{i=1}^{1247} c_{i,i}^{norm}$$

We can see from Figure 3.5 that the most recent memory usage measurements are very valuable and relevant. This result is what makes even the simplest algorithms such as a persistence model so powerful. We can also note that there is not really any hourly periodicity. Though interestingly, there does seem to be a slight pattern every 2 hours. This may be related to the amount of time a machine takes to do one task, but that is uncertain.

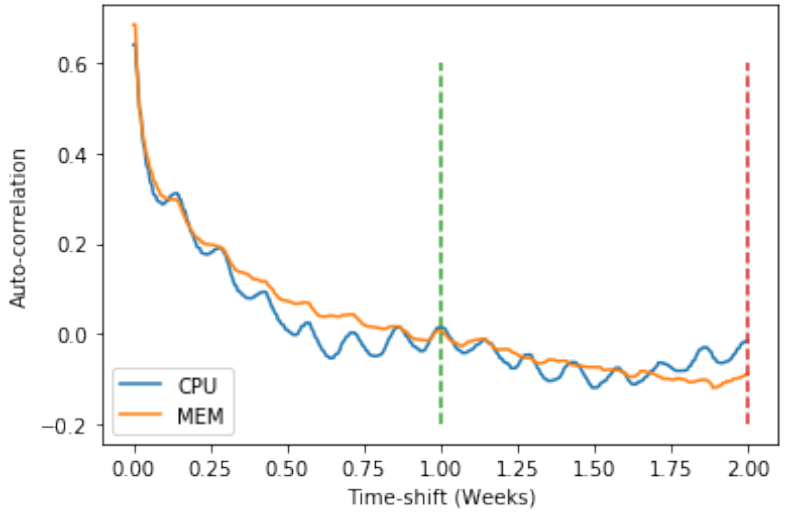

Figure 3.4: Temporal Auto-Correlation with weekly time-shifts.

Let us now focus our attention to long term patterns, which can be better inspected via Figure 3.4. It is clear that the most relevant information by far is in the recent usage values (*i.e.* near 0 time-shift). Furthermore, though there is some weekly cyclicity, it is quite insignificant. With these insights, we can safely justify discarding all information older than a week as stale.

It can clearly be seen from the graph that there are regular spikes occuring more frequently than in weeks. These are likely daily, and appear to be more significant. We will now investigate these using Figure 3.6. The value of the information drops exponentially with time. We now see that all information older than 3 days has insignificant auto-correlation, and can essentially be regarded as noise. We will discard this information as well.

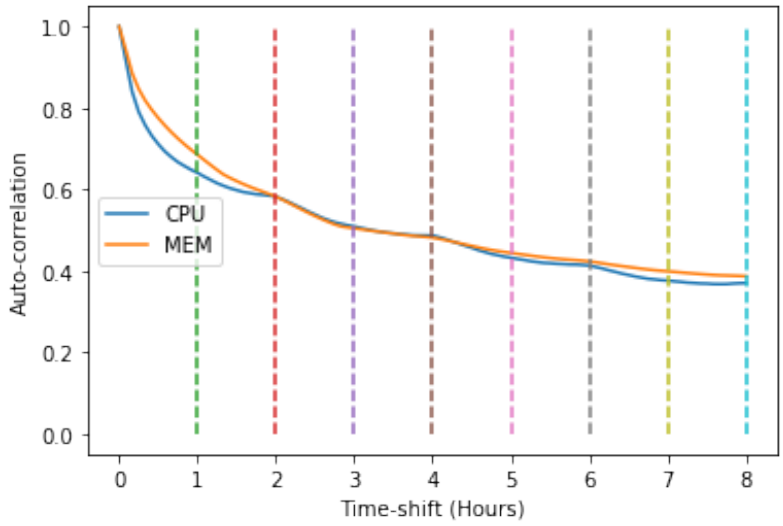

Figure 3.5: Temporal Auto-Correlation with hourly time-shifts.

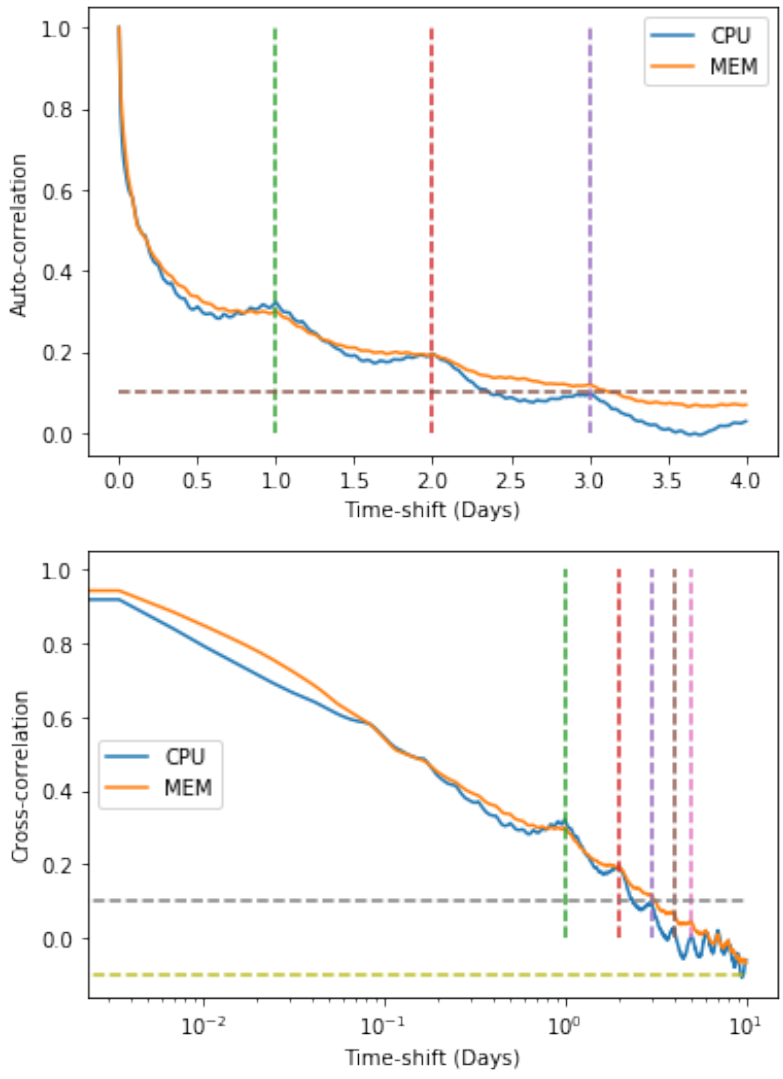

Figure 3.6: Temporal Auto-Correlation with hourly time-shifts.

Overall, we can say that there are certain time-shifts that consistently deliver high auto-correlation for all machines. We can definitely utlise them for our forecasting.

3.2.2 Spatial Correlations

For testing spatial correlations, we keep our time-shift constant at 0 and use cross-correlation values between different pairs of machines as our metric of similarity. It should be noted that zero-padding is no longer necessary. With 0 time-shift, the signals from the two machines (each with the same amount of datapoints) will overlap completely. This removes the need for handling boundary effects, since there are none. We do, however, still need to normalise our dataset so that all values fall between 0 and 1.

$$c_{norm} = \frac{c}{min(||x||, ||y||)}$$

We do not have any information on physical location of any of the machines so cannot theorise about which ones are likely to have a strong correlation. We must test every possible pair, or alternatively, pick pairs of machines at random.

This also relates to why we will not calculate cross-correlation with time-shifts. If we knew there was a certain time zone difference between machines, it might be promising to calculate cross-correlation with that amount of time-shift. In another example, physically adjacent machines might hand off tasks to one another in a periodic sequence. All of this is unknown to us, so the only option again would be to brute-force every possible time-shift for every pair of machines. This is too computationally demanding to be practical, so is not attempted.

Let us first have a look at whether there are any strong long-term cross-correlations between machines. We calculate the correlation between every possible pair within our subset of 1247 machines, using all 8351 datapoints. The results can be seen in Figure 3.7.

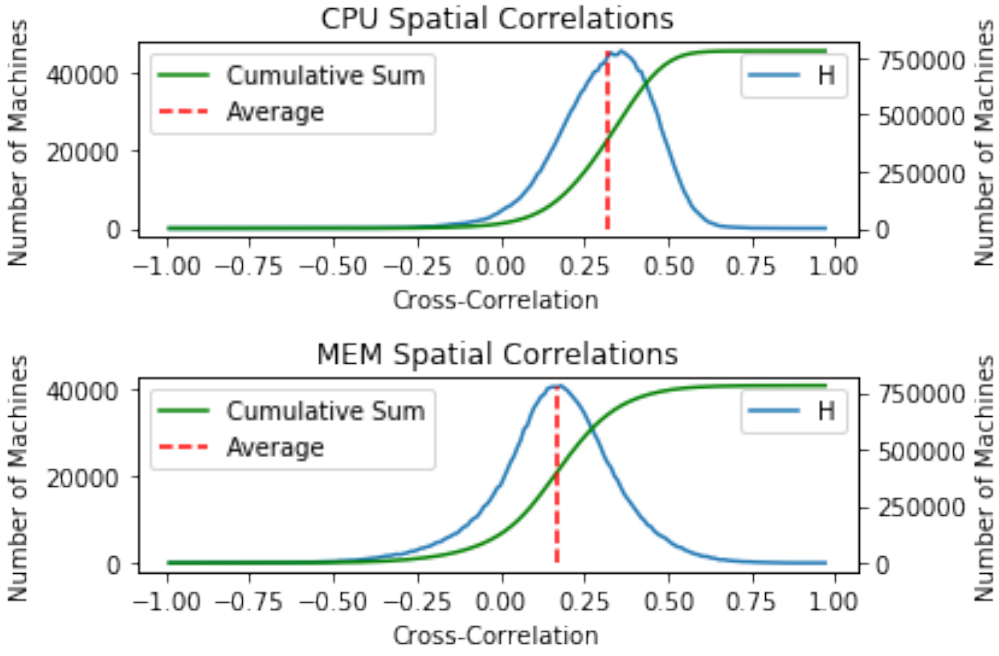

Figure 3.7: Histogram of cross-correlation values.

Generally speaking, the results are not very promising. There are a few pairs with moderate amounts of correlation, but there is no easy way to distinguish which ones they will be. Though long-term correlations seem to be weak (over the full span of 29 days), there may be shorter-term correlations that are stronger. We now test for these. Table 3.1 shows the results.

Timespan (Days)	CPU	MEM
29	0.32	0.17
14	0.12	0.07
7	0.14	0.07
1	0.07	0.06
0.5	0.10	0.08

Table 3.1: Average cross-correlation at different timespans.

Shorter timespans perform even worse. Furthermore, even if we wanted to proceed by utilising these weak correlations, they are so weak that there is no guarantee they won't wildly fluctuate going forward. We test for this next.

We take the pairs that have a higher than average correlation in our first time window. We then calculate the correlations of these pairs in the next time window. An example of the drifts of the values has been plotted in Figure . The green solid line is a histogram of the high-correlation pairs in the first time window. The yellow solid line is the correlation of those same pairs in the second time window. The red solid line is the difference between the two values. The dashed lines represent the average value of their respective histograms.

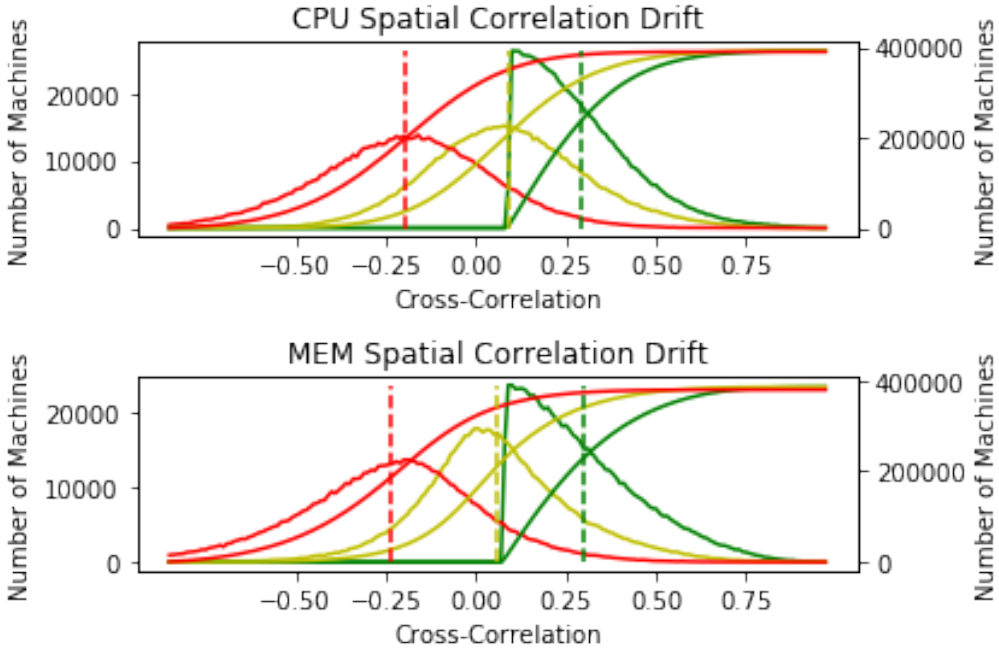

Figure 3.8: Correlation drift between two consecutive time windows lasting a day each.

The results for the above and a few other combinations of windows lengths are given in Table 3.2 and Table 3.3. Possible combinations that are not included in the table also

led to similar results.

Window 1 (Days)	Window 2 (Days)	Average Correlation 1	Average Correlation 2	Drift
7	7	0.27	0.20	-0.07
1	1	0.23	0.13	-0.10
0.5	0.5	0.29	0.09	-0.20
14	0.5	0.23	0.09	-0.14

Table 3.2: Correlation drift of CPU usage between two consecutive time windows.

Window 1 (Days)	Window 2 (Days)	Average Correlation 1	Average Correlation 2	Drift
7	7	0.22	0.08	-0.14
1	1	0.23	0.06	-0.17
0.5	0.5	0.30	0.06	-0.24
14	0.5	0.21	0.06	-0.15

Table 3.3: Correlation drift of MEM usage between two consecutive time windows.

It can be seen that even the relatively high-correlation pairs of machines drift into insignificance quickly, with most pairs failing to remain above 0.2 . Based on these results, we henceforth disregard all spatial correlations and treat all machines as parallel and independent signals.

3.3 Model Design

3.3.1 Feature Reduction

After our analyses, we no longer need to train a model for this task. We have already discovered that for a given machine, we can discard specific and significant portions of the data;

- data from all other machines,
- data older than 3 days.

3.3.2 Clustering

To ease the computational load, we would like to reduce the number of time series somehow. The most logical and established way to do this is by forming clusters of machines, with one time series representing the entire cluster. This has the added advantage of filtering out high-frequency noise.

In the interest of keeping data of different natures seperate, CPU and MEM data will never be clustered together.

Averaging with Deltas

One method would be to simply take the average of all signals, while also recording the offset of each from the average.

$$\mu = \sum_{i=0}^{12476} \frac{x_i}{12476}$$

$$\delta_i = x_i - \mu,\ \forall i \in \{0, \ldots, 12476\}$$

After making a prediction on the average of the signals, the individual predictions are obtained using the offsets δ.

$$\hat{y}_i[T+1] = \hat{\mu}[T+1] + \delta_i[T],\ \forall T \in N$$

K-means with Deltas

Averaging all machines results in just one cluster that contains everything. It is dubious that this is a flexible enough solution to be representative of the data. To improve on this, we need methods with more clusters.

The K-Means algorithm has K cluster centres. The ideal value of K is a trade-off between efficiency and accuracy. Given a set of observations (*i.e.* a set of one-dimensional time series) $X = \{x_0, x_1, \ldots, x_n\}$, the algorithm tries to partition the n observations into the $k < n$ sets $S = \{S_0, S_1, \ldots, S_k\}$ that minimise within-cluster sum of squares.

$$\mu_i = \sum_{x \in S_i} \frac{x}{|S_i|},\ \forall i \in \{0, 1, \ldots, k\}$$

$$\arg\min_S \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 = \arg\min_S \sum_{i=1}^{k} |S_i| Var\ S_i$$

The end result of the algorithm is illustrated in Figure 3.9.

The process for getting from cluster centres (called *centroids*) back to individual predictions is similar to the one in the averaging method.

$$\delta_j = x_j - \mu_i,\ \forall j\ where\ x_j \in S_i$$

$$\hat{y}_j[T+1] = \hat{\mu}_i[T+1] + \delta_j[T],\ \forall j\ where\ x_j \in S_i,\ \forall T \in N$$

Weighted K-means with Deltas

Currently, when we are calculating the euclidian distance between a time series and a cluster centre, we are weigthing all datapoints in the time series equally. Our temporal correlation analysis from before suggests this is not representative of actual signal similarities.

We should instead focus on clustering together the signals whose most recent values are similar to each other. The similarity of values in the distant past is much less relevant. In our weigthed K-means method, we multiply each value of the time series with the average temporal auto-correlation value from our analyses. The most recent values stay almost constant, while at the other extreme, values older than 3 days are multiplied with correlation values < 0.1 which represent their actual significance in our system.

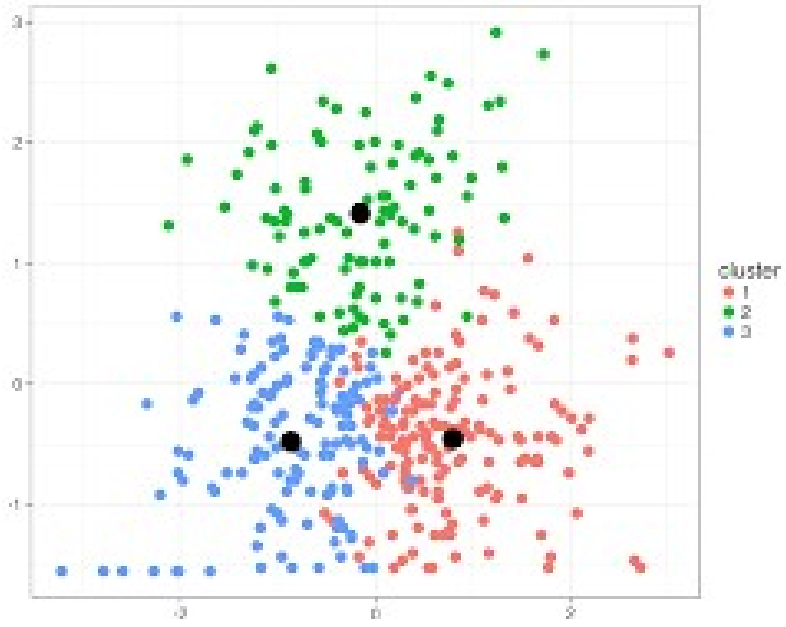

Figure 3.9: A possible set of clusters. [10]

$$x_i^{weighted} = c^{avg} \times x_i,\ \forall i \in \{0, \ldots, 12476\}$$

3.3.3 Baseline Predictors

We implement 3 baseline forecasting models for the purpose of establishing a basic target to surpass.

Persistence

This model simply predicts that the upcoming value in the time series will be identical to the previous one.

$$\hat{y}[T+1] = y[T],\ \forall T \in N$$

Expanding Window

This model takes the average of all previous values in the time series as a prediction.

$$\hat{y}[T+1] = \sum_{t=0}^{T} t,\ \forall T \in N$$

Rolling Window

This model takes the average of the last n values in the time series as a prediction.

$$\hat{y}[T+1] = \sum_{t=T-1-n}^{T} t,\ \forall T \in N,\ \forall n \in \{0, \ldots, 8351\}$$

3.3.4 Advanced Predictors

Seasonal Autoregressive Integrated Moving Average

SARIMA is built up of multiple components that are integrated. Let us first explain the components.

The Autoregression (*AR*) method models the next step in the sequence as 'a linear function of the observations at prior time steps'. [40] The model has one hyperparameter p which specifies the order of the linear function used. This hyperparameter must be opmtimised in order to avoid overfitting or underfitting, as previously mentioned (see Chapter 5).

$$X[T] = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X[t-i]$$

where $\{\varphi_1, \ldots, \varphi_p\}$ are the parameters of the model, c is a constant, and ε_t is white noise.

The Moving Average (*MA*) method models the next step in the sequence as 'a linear function of the residual errors from a mean process at prior time steps'. [40]. Similarly, it has one hyperparameter q to optimise, which specifies its order.

$$X[T] = \mu + \varepsilon_t + \sum_{i=1}^{q} \vartheta_i \epsilon_{t-i}$$

where μ is the mean of the series, $\{\vartheta_1, \ldots, \vartheta_q\}$ are the parameters of the model, and $\{\varepsilon_t, \varepsilon_{t-1}, \ldots, \varepsilon_{t-q}\}$ are white noise error terms.

The Autoregressive Moving Average (*ARMA*) model simply combines the previous two. Therefore, it has two hyperparameters; p and q.

$$X[T] = \mu + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X[t-i] + \sum_{i=1}^{q} \vartheta_i \epsilon_{t-i}$$

The Autoregressive Integrated Moving Average (*ARIMA*) model adds a diffrencing pre-processing step called Integration (*I*), which has the hyperparameter d for its difference order. The Seasonal ARIMA (SARIMA), in turn, adds 4 more hyperparameters; P, D, Q for the seasonal counterparts of the already-existing trend hyperparameters, and finally m, the seasonal period length.

Since we have already observed daily seasonality in our dataset, we don't have to spend time searching for m. For the other hyperparameters, we will perform a grid search during implementation in order to optimise them.

Long Short-Term Memory

An *LSTM* [41] is a type of Recurrent Neural Network (*RNN*). The key characteristic of RNN's is that they have feedback loops. The output of the network gets fed back into the same network. This is equivalent to stacking up multiple neural networks with the same architecture and same weight parameters. The motivation behind adding feedback to the system is to give the network a memory of the past sequence of events, since they might be relevant to the current state of the system and therefore the decision making process. RNN's, and especially LSTM's, have been providing great results and breakthroughs in recent years.

The primary challenge when working with RNN's is training them to capture long-term patterns. Backpropogation is quite ineffective for this because of the vanishing gradient problem. [42] LSTM's were specifically invented to solve this problem.

The basic unit of an LSTM is called a *cell*. Each cell contains 3 regulators;

- a forget gate,
- an input gate,
- and an output gate.

These parts are illustrated in Figure 3.10. It can be seen that the internal state c flows through each cell relatively untouched if need be.

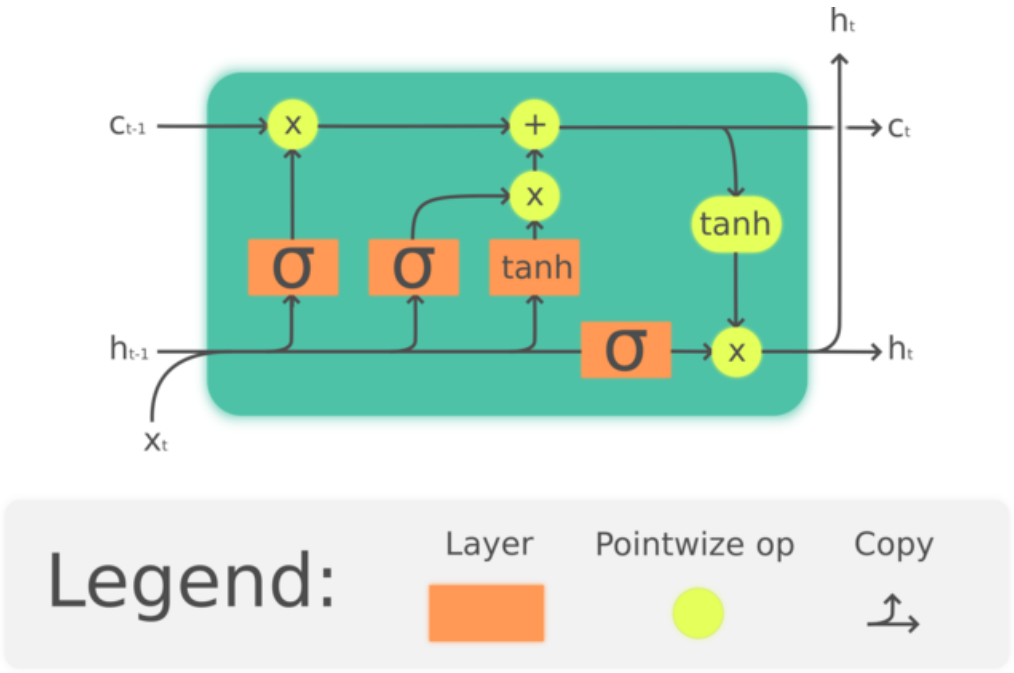

Figure 3.10: Components of a classic LSTM cell. [11]

We use a windowed LSTM; for each time series, we feed in the values from a constant time window (*i.e.* the last 3 days) in order to predict the next value. This reduces computational complexity, and filters out noise since we've established that information goes stale after 3 days.

Our model consists of two stacked layers of LSTM, with ReLu actication functions. This is followed by a Dense layer which outputs a scalar value.

We frame the problem as a multivariate one, where there is multiple parallel series of inputs and their corresponding outputs. One instance of our model is trained on the CPU data of all machines, while another is trained on the MEM data of all machines.

Chapter 4

Implementation

The dataset was organised and saved into the files *CPU.npy* and *MEM.npy*. Each row represents a machine. The columns sequentially represent points in time.

All code was written in Python3. A virtual environment was used to document dependencies. The Pipfile detailing the dependencies can be found in Appendix B.

4.1 Clustering

4.1.1 Averaging with Deltas

We implement the algorithm in a simple function (see Figure 4.1) that takes the dataset as an input and returns the average of each column.

```
import numpy as np
def average_all_machines(data):
    clusters = dict()

    for data_type in data:
        clusters[data_type] = np.array(
            [np.average(data[data_type], axis=0)])
    return clusters
```

Figure 4.1: Function for Averaging with Deltas.

4.1.2 K-means with Deltas

The algorithm was wrapped in a function (see Figure 4.2) which runs it 10 times and selects the cluster centres that give the best result. It takes the dataset and the number of clusters k as inputs. Initially, we place the cluster centres at random. Then, we deploy Lloyd's Algorithm [43] for 300 iterations with an added break clause;

- Assignment: assign each machine to the cluster of the nearest centroid (in euclidean space).
- Break: if no machines have changed clusters, stop iterating.

- Update: calculate the new centroids after the new assignments.

```
from sklearn.cluster import KMeans
def kmeans(data, n_clusters, return_labels=False):
    clusters, labels = dict(), dict()

    for data_type in data:
        kmeans = KMeans(
                n_clusters=n_clusters,
                random_state=0).fit(data[data_type])

        clusters[data_type] = kmeans.cluster_centers_
        labels[data_type] = kmeans.labels_
    if return_labels:
        return clusters, labels
    else:
        return clusters
```

Figure 4.2: Function for K-means with Deltas.

Finally, we return the centroids and the assignment of each machine.

4.1.3 Weighted K-means with Deltas

We utlise the function (see Figure 4.3) that we used in K-means with Deltas, except before we input the dataset we do an element-wise multiplication with the average temporal autocorrelaitons. When the function returns our centroids, we element-wise divide them with the average temporal autocorrelations.

4.2 Baseline Predictors

Our baseline models require no training. They have neither hyperparameters nor parameters that require tuning.

4.2.1 Persistence

We implement a function that takes the testing data as input and returns the most recent value in the testing data.

4.2.2 Expanding Window

We implement a function that takes the testing data as input and returns the average of the testing data.

```
def weighted_kmeans(data, n_clusters, return_labels=False):
    clusters, labels = dict(), dict()

    for data_type in data:
        if ZERO_SHIFT_TIMESTAMP > data[data_type].shape[1]:
            weights = avg_correlation[data_type][
                ZERO_SHIFT_TIMESTAMP
                - data[data_type].shape[1] :
                ZERO_SHIFT_TIMESTAMP
            ]
        else:
            weights = (
                [0.01 for _ in
                range(data[data_type].shape[1])])
            weights[-ZERO_SHIFT_TIMESTAMP:] = (
                avg_correlation[data_type][
                :ZERO_SHIFT_TIMESTAMP])
        kmeans = KMeans(
            n_clusters=n_clusters, random_state=0).fit(
            data[data_type] * weights)

        clusters[data_type] = (
            kmeans.cluster_centers_ / weights)
        labels[data_type] = kmeans.labels_
    if return_labels:
        return clusters, labels
    else:
        return clusters
```

Figure 4.3: Function for Weighted K-means with Deltas.

4.2.3 Rolling Window

We implement a function that takes the testing data and window length n as input, and returns the average of the n most recent values in the testing data. We take $n = \frac{3\times24\times60}{5}$.

4.3 Advanced Predictors

We will need to spend time training these models in order to optimise their performance. For this we will use our training set (see Chapter 5).

4.3.1 SARIMA

We use the implementation of the model provided by the statsmodel library. We perform a grid search over the following ranges of values:

- $p \in [0, 2]$,

- $d \in [0, 1]$,
- $q \in [0, 2]$,
- $P \in [0, 2]$,
- $D \in [0, 1]$,
- $Q \in [0, 2]$

We execute configurations in parallel on our multi-core CPU to save time. We first use the averaging cluster model on the CPU and MEM datasets. We fit the model to 60% of the resulting time series. We then test the trained model on another 20% and obtain Root Mean Squared Error (*RMSE*) as our metric for accurate predictions. The top 3 best performing hyperparameters are displayed in Table 4.1.

p	d	q	P	D	Q	RMSE
0	1	1	1	0	2	0.00367
2	0	0	2	0	0	0.00369
1	1	2	0	0	0	0.00375

Table 4.1: Best hyperparameters from SARIMA grid search.

4.3.2 LSTM

The architecture of our LSTM can be seen in Figure 4.4. Each layer is followed by a ReLu activaiton function. 864 datapoints is equivalent to $\frac{864 \times 5}{24 \times 60} = 3$ days' worth of data. In order to train the network, the sequential data was split into samples.

$$X_T = \{X[T+0], X[T+1], \ldots, X[T+863]\},$$
$$y_T = X[T+864],$$
$$\forall T < ((0.6 \times 8351) - 863) \in N$$

The network was trained for 10 epochs.

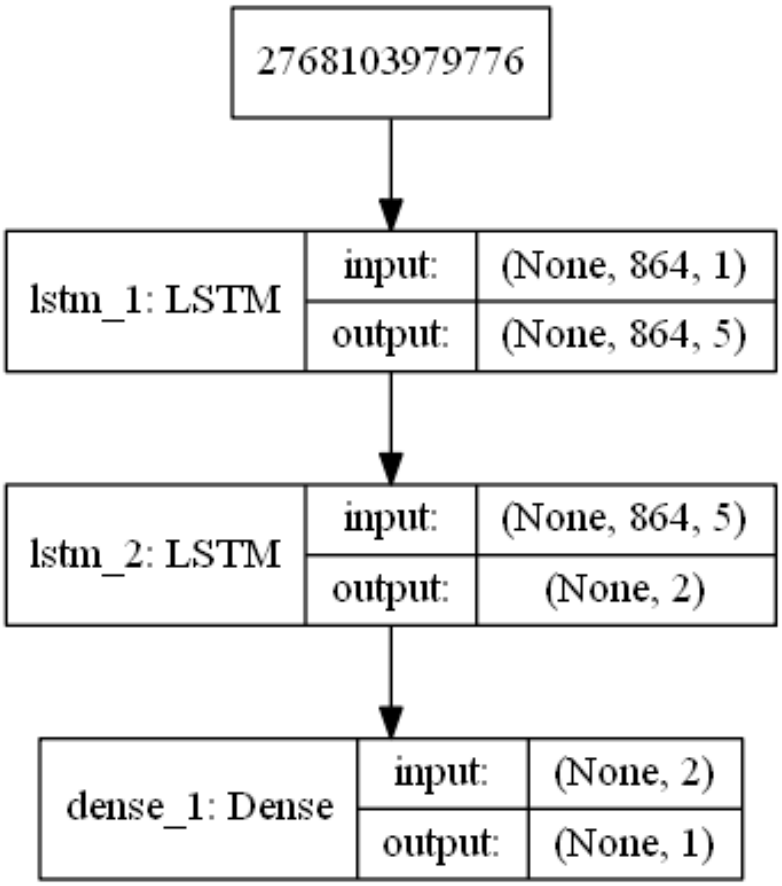

Figure 4.4: Architecture of LSTM network used.

Chapter 5

Testing

As mentioned earlier, all models are using a real dataset. We split the dataset into 3 subsets;

- training,
- testing,
- validation.

The training set will be used to educate the model. The testing set will be used to compare different models or the same model with different parameters. The validation set will be used for evaluating the success of our chosen model.

This data split is a common practice in machine learning. The purpose of it is to have a realistic evaluation of success that takes noise into account. Noise is a quantisation of the fact that our data is not a perfect representation of reality.

Let X represent a dataset and Y the true output values of that dataset in a real-world situation. The function we want our machine learning algorithm to approximate is f where

$$Y = f(X)$$

However our training data

$$X' = X + \epsilon \approx X$$

where ϵ represents some error or noise in our training data. Now our training dataset represents a function f' where

$$f' \approx f$$
$$Y = f'(X')$$

This represents a problem; if our model learns to fit the noise ϵ it will not perform well with other datasets since that particular pattern of random noise was exclusive to our training set and does not represent any underlying truth about the actual relationship between X and Y. This is called *overfitting*, and it prevents the model from *generalising* well. On the other hand, we could be discarding genuine information by treating it as noise as well. This is called *underfitting*, and it also leads to poor generalisation. This is demonstrated by the polynomials in Figure 5.1. A polynomial of degree 1 is not sufficient

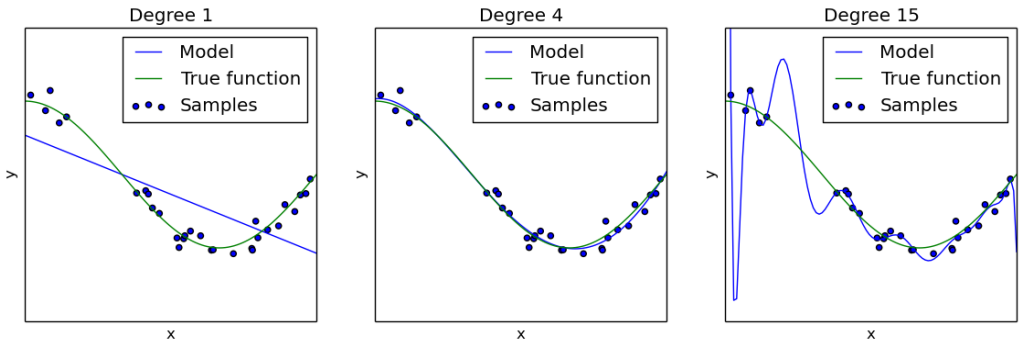

Figure 5.1: 3 linear regression models with polynomial features of different degrees. [12]

to fit the training samples. A polynomial of degree 4 gives an approximation of the true function that is almost perfect. A polynomial of degree 15 overfits the data. [44]

The balance between overfitting and underfitting must be handled carefully, but how can we measure generalisation? Having a second dataset solves this issue.

We do not actually care about how well our model approximates f', we only care about how well it approximates f. To measure this, we reserve a part of our dataset and which we don't use it for training. This subset is called the *testing set*. We compare the performance of our models with different parameters via the testing set, and use these results to pick the best model.

Figure 5.2 illustrates the effects of underfitting and overfitting on training and testing errors. Note how increasing the model complexity indefinitely is undesirable even though it reduces training error, because after a certain point it causes the testing error (also known as out-of-sample error) to increase.

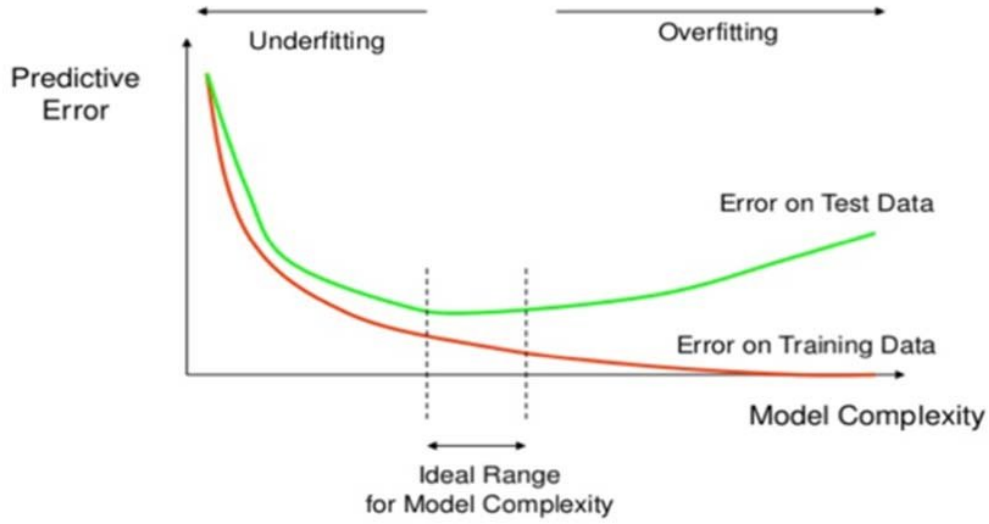

Figure 5.2: Model complexity versus predictive error. [13]

We now have another problem. We have repeatedly allowed the testing set, and therefore the noise contained within the testing set, to influence our decisions about how to tune our models. The models have now learned to fit the noise in the testing set as well.

We now need a third subset, completely unused for any decision making. This set is called the $validationset$ and it can never be used to make any decisions about the models. It is only used to evaluate the performance of a finalised model that will not be further tuned.

This method of splitting the dataset in order to subject the model to previously unseen data is called *cross validation* or *out-of-sample-testing*. It is common to perform multiple rounds of cross validation, changing which of the subsets are used for testing and validation each time. However, in our case this will not be possible since the dataset is a sequential time series and should not be shuffled.

Chapter 6

Results

The more accurately the models can forecast future resource occupancy, the more accurately efficiently they can schedule tasks to nodes in the distributed system. RMSE is a metric that is an industry standard for accuracy, and it also what we are using. We use the out-of-sample errors of the models for our measurements.

The success of the machine learning algorithms will first of all depend on their performances compared to a simplistic baseline method. Secondly, they will be compared to each other in terms of performance.

6.1 Baseline Predictors

6.1.1 Persistence

The predictions of the model are illustrated in Figure 6.1. Its average RMSE was 0.154 .

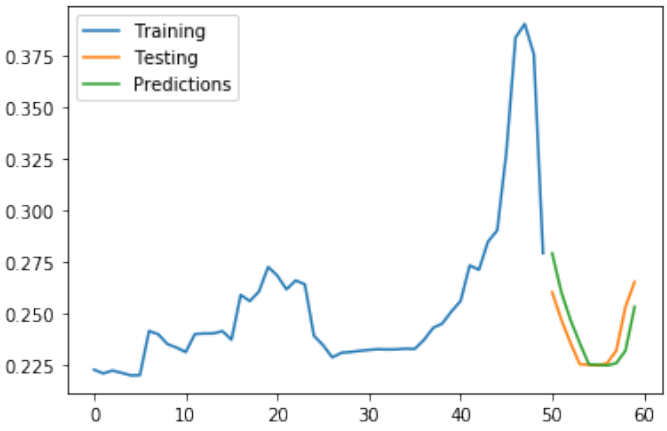

Figure 6.1: Predictions of Persistence Algorithm.

6.1.2 Expanding Window

The predictions of the model are illustrated in Figure 6.2. Its average RMSE was 0.0795 .

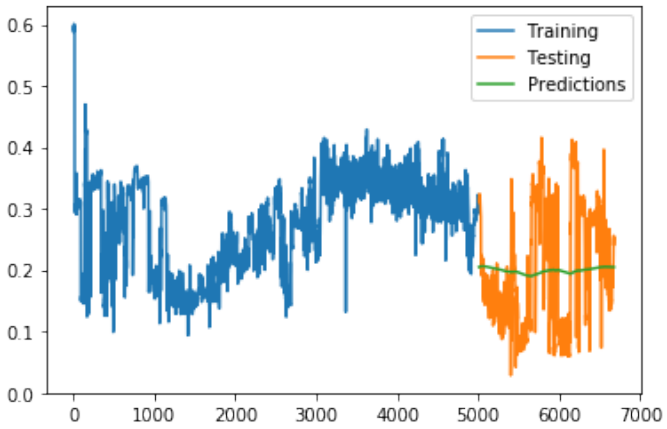

Figure 6.2: Predictions of Expanding Window Algorithm.

6.1.3 Rolling Window

The predictions of the model are illustrated in Figure 6.3. Its average RMSE was 0.0810 .

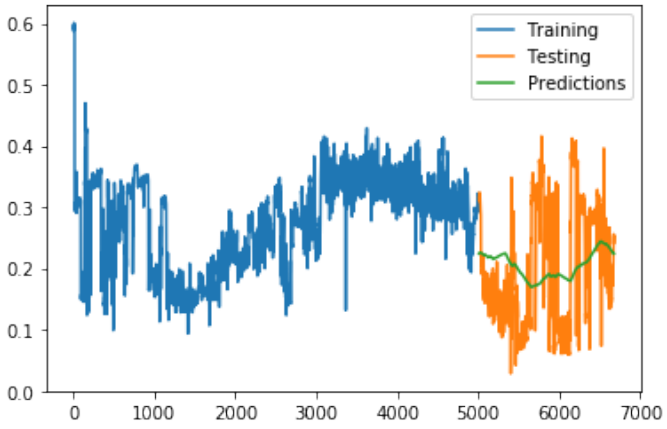

Figure 6.3: Predictions of Rolling Window Algorithm.

6.2 Advanced Predictors

6.2.1 SARIMA

The RMSE of our optimised model on the validation set is 0.00310 . It fits the model parameters at every new timestep and updates its history as it goes.

6.2.2 LSTM

Our LSTM had an RMSE of 0.129 .

Chapter 7

Evaluation

Table 7.1 compares the RMSE's of the different predictors. SARIMA performed the best by far. LSTM's performance was comparable to our baselines but much worse than SARIMA.

Model	RMSE
Persistence	0.154
Expanding Window	0.0795
Rolling Window	0.0810
SARIMA	0.00310
LSTM	0.129

Table 7.1: Validation RMSE of different models.

Chapter 8

Conclusions and Further Work

8.1 Conclusions

SARIMA has presented itself as a very good model choice for resource utilisation forecasts. Our LSTM did not have a very good performance. ARIMA is probably a better and simpler choice for most scenarios, unless the designer is willing to spend significant computing resources and time in order to get the LSTM to a similar performance level.

8.2 Further Work

The project could be continued by trying different LSTM architectures. It would also be very insightful to see a more detailed comparison of the different clustering methods in terms of their impacts on the accuracy of different models.

Chapter 9

User Guide

All the data used for this project has been voluntarily published by Google in an attempt to 'make visible many of the scheduling complexities that affect Google's workload, including the variety of job types, complex scheduling constraints on some jobs, mixed hardware types, and user mis-estimation of resource consumption'. [39]

The usage trace is located in a public Google Cloud Platform bucket at `https://console.cloud.google.com/storage/browser/clusterdata-2011-2`.

All code we have used is publicly avaiable as well, at `https://github.com/larok00/ML-for-Cloud-and-Distributed-Computing`.

Bibliography

[1] John Wilkes and Charles Reiss. Clusterdata2011 2 traces.

[2] Sam Johnston. File:cloud computing.svg — wikimedia commons, the free media repository, 2018. [Online; accessed 27-January-2019].

[3] obviouscarelessblackfootedferret. perceptron pla 14 steps, May 2017.

[4] Peter Corcoran. Cloud computing and consumer electronics: A perfect match or a hidden storm?[soapbox]. *Consumer Electronics Magazine, IEEE*, 1:14–19, Apr 2012.

[5] Milan Tair. Distributed computing solution for hardware virtualization. *ResearchGate*.

[6] Showing 132,080 records for all fields: (machine learning). *Web of Knowledge*, 2019.

[7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[8] Jyh-Jian Sheu, Ko-Tsung Chu, Nien-Feng Li, and Cheng-Chi Lee. An efficient incremental learning mechanism for tracking concept drift in spam filtering. *PLOS ONE*, 12:e0171518, 02 2017.

[9] Alexandre Drouin. Microbiome summer school 2017.

[10] Czar Yobero. K-means clustering tutorial, 2018.

[11] Wikimedia Commons. File:the lstm cell.png — wikimedia commons, the free media repository, 2019. [Online; accessed 19-June-2019].

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Hayder Al-Behadili, Ku Ku-Mahamud, and Rafid Sagban. Rule pruning techniques in the ant-miner classification algorithm and its variants: A review. 04 2018.

[14] Cloud computing vs. distributed computing. *DeZyre*, Apr 2015.

[15] What is cloud computing? *Microsoft Azure*.

[16] Dais-ita. *Imperial College*.

[17] EUROPEAN COMMISSION. Communication from the commission to the european parliament, the council, the european economic and social committee and the committee of the regions, 2010.

[18] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, et al. Cloud computing, 2015.

[19] EUROPEAN COMMISSION. Communication from the commission to the european parliament, the council, the european economic and social committee and the committee of the regions, 2011.

[20] EUROPEAN COMMISSION. Communication from the commission to the european parliament, the council, the european economic and social committee and the committee of the regions, 2008.

[21] DG-Information Society and Media. Ict for energy efficiency, 2008.

[22] EUROPEAN COMMISSION. Communication from the commission to the european parliament, the council, the european economic and social committee and the committee of the regions, 2010.

[23] Jonathan G. Koomey. Growth in data center electricity use 2005 to 2010, 2011.

[24] M. Pickavet, W. Vereecken, S. Demeyer, P. Audenaert, B. Vermeulen, C. Develder, D. Colle, B. Dhoedt, and P. Demeester. Worldwide energy needs for ict: The rise of power-aware networking. In *2008 2nd International Symposium on Advanced Networks and Telecommunication Systems*, pages 1–3, Dec 2008.

[25] Josh Whitney and Pierre Delforge. scaling up energy efficiency across the data center industry: evaluating key drivers and barriers. Aug 2014.

[26] Jack Clark. 5 numbers that illustrate the mind-bending size of amazon's cloud. Nov 2014.

[27] Larry Dignan. Aws cloud computing ops, data centers, 1.3 million servers creating efficiency flywheel. *Between the Lines*, Jun 2016.

[28] Aws's competitors, revenue, number of employees, funding and acquisitions. *Owler*.

[29] Angus Kidman. Why your server/employee ratio matters, Sep 2013.

[30] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950.

[31] John McCarthy and Edward A Feigenbaum. In memoriam: Arthur samuel: Pioneer in machine learning. *AI Magazine*, 11(3):10, 1990.

[32] Bernard Marr. A short history of machine learning – every manager should read, Feb 2016.

[33] Viktor Losing, Barbara Hammer, and Heiko Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.

[34] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109 – 165. Academic Press, 1989.

[35] Max Pager. What is online machine learning? *Medium*, 2018.

[36] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[37] Borg cluster traces from google. *GitHub repository*.

[38] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[39] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.

[40] Jason Brownlee. 11 classical time series forecasting methods in python (cheat sheet), 2018.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[42] J. F. Kolen and S. C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*. IEEE, 2001.

[43] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[44] Underfitting vs. overfitting. *scikit-learn*.

Appendix A

Known anomalies in clusterdata-2011-2 [1]

- disk-time-fraction data is only included in about the first 14 days because of a change in our monitoring system.
- some jobs are deliberately omitted because they ran primarily on machines not included in this trace. the portion that ran on included machines amounts to approximately 0.003% of the machines' task-seconds of usage.
- there is only one known example of a job that retains its job id after being stopped, reconfigured, and restarted (job number 6253771429).
- approximately 70 jobs (*e.g.* job number 6377830001) have job event records but no task event records. it believed that this is legitimate in a majority of cases: typically because the job is started but its tasks are disabled for its entire duration.
- approximately 0.013% of task events and 0.0008% of job events in this trace have a non-empty missing info field.
- it is estimated that less than 0.05% of job and task scheduling event records are missing and less than 1% of resource usage measurements are missing.
- some cycles per instruction (cpi) and memory accesses per instruction (mai) measurements are clearly inaccurate (for example, they are above or below the range possible on the underlying micro-architectures). it is believed that these measurements are caused by bugs in the data-capture system used, such as the cycle counter and instruction counter not being read at the same time. to obtain useful data from these measurements, it is suggested to filter out measurements representing a very small amount of cpu time and measurements with unreasonable cpi and mai values.

Appendix B

Pipfile for Python virtual environment

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]
pylint = "*"
rope = "*"
autopep8 = "*"

[packages]
numpy = "*"
matplotlib = "*"
jupyter = "*"
scipy = "*"
pandas = "*"
sklearn = "*"
statsmodels = "*"
keras = "*"
tensorflow = "*"

[requires]
python_version = "3.7"
```