Forecasting Methods for Cloud Hosted Resources, a comparison

by

Manrich van Greunen

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electric and Electronic Engineering in the Faculty of Engineering at Stellenbosch University

Department of Electric and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Dr. H.A. Engelbrecht

December 2015

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature: .....
M. van Greunen

Date: 24/11/2015 .....

Abstract

Forecasting Methods for Cloud Hosted Resources, a comparison

M. van Greunen

Department of Electric and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (E&E)

December 2015

Cloud computing has revolutionised the modern day IT industry and continues to foster the development of new products and services. Amid the dynamically changing workloads presented to cloud computing lies the challenge of ensuring sufficient resources are available when needed. Recently, proactive provisioning and auto-scaling schemes have emerged as solutions to this. Forecasting methods are inherent to these provisioning schemes and to the author's knowledge, no formal investigation has been performed in comparing different forecasting methods. The purpose of this research was to investigate various forecasting methods presented in recent research, adapt evaluation metrics from literature and compare these methods on prediction performance using two real-life cloud resource datasets.

It was found that less complex methods, such as moving average and auto-regression outperformed other more complex methods that were investigated, on the majority of used evaluation metrics. We also found that our 30th order auto-regression model achieved statistically significantly better results compared to the other forecasting methods. Furthermore, there was no single evaluation metric that gave concise comparative results between forecasting methods, but overload likelihood ratio as metric showed great promise to this end. It was argued that focus should be put on developing evaluation metrics that specifically relate to the cloud environment and further investigation should be performed on a closed-loop system or real-life cloud platform.

Cloud computing has become ubiquitous with the Internet as we know it today. We believe that effective provisioning of cloud computing resources should be at the core of modern cloud management systems and the primary objective of cloud platform providers.

Uittreksel

M. van Greunen
Departement Elektriese en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (E&E)
Desember 2015

Die wolk-verwerking revolusie in hedendaagse IT industrieë ontwikkel voortdurend nuwe produkte en dienste. Te midde van die dinamiese gedrag van wolk verwerking, as gevolg van veranderende werkslandings op wolke, is dit 'n uitdaging om te verseker dat genoeg verwerker-hulpbronne beskikbaar is voordat dit benodig word. Ontwikkelinge in pro-aktiewe voorsiening en outomatiese skallerings skemas was onlangs gemaak ter oplossing vir hierdie uitdaging. Inherent aan hierdie skemas is die gebruik van vooruitskattingsmetodes en sover die outeur se kennis strek, is daar tans geen resultate van formele ondersoeke in die vergelyking van verskeie vooruitskattingsmetodes, beskikbaar nie. Die doel van hierdie navorsing was om ondersoek in te stel aangaande verskeie vooruitskattingsmetodes en die aanpas van evalueringsmaatstawwe soos genoem in literatuur. Met behulp van werklike wolk hulpbron datastelle was hierdie metodes met mekaar vergelyk.

Daar is gevind dat eenvoudige metodes, soos gly-gemiddeld en outo-regressie, uitgeblink het wanneer dit gemeet was met die meerderheid van die maatstawwe. Ons 30ste orde outo-regressie model verkry die hoogste akkuraatheid. Verder, is daar gevind dat geen een evaluasie maatstaf 'n duidelike verskil tussen metodes uitwys nie, maar dat die oorbelas waarskynlikheidsverhouding vir hierdie doel belowend lyk. Daar is aangevoer dat fokus geplaas moet word op die ontwikkeling van evalueringsmaatstawwe wat spesifiek verwant is aan die wolk omgewing en verdere ondersoek op 'n geslote-lus stelsel of werklike wolk platform, gedoen moet word.

Wolk-verwerking is alomteenwoordig met die Internet soos ons dit vandag ken. Effektiewe voorsiening van wolk hulpbronne en die gebruik van vooruitskattingsmetodes is die kern van moderne wolk bestuurstelsels. Wolk platform verskaffers behoort dit as hul primêre doel tot sukses te beskou.

Acknowledgements

I would like to express my sincere gratitude to the following people:

- my supervisor, Dr Herman Engelbrecht, for his continued guidance and support throughout my research;
- my family and friends their encouragement and moral support;
- my best friend, Daniël Schoonwikel, for standing (and sitting) next to me, and taking on the great challenge which is MEng;
- to Holy Father, for keeping me and blessing me with opportunities, knowledge and abilities.

Dedications

To my wife, Carla, for your unconditional love, support and understanding.
Thank you.

Contents

Declaration i

Abstract ii

Uittreksel iii

Acknowledgements iv

Dedications v

Contents vi

List of Figures xi

List of Tables xiv

Nomenclature xv

Acronyms . . . xv
Symbols . . . xvi
Forecasting Methods . . . xvi
Neural Networks . . . xvii
Evaluation Metrics . . . xvii

1 Introduction 1

1.1 Motivation . . . 1
1.2 Background . . . 2
1.2.1 Cloud computing . . . 2
1.2.2 Cloud service levels . . . 2
1.2.3 Actors in the cloud environment . . . 3
1.2.4 Cloud workloads . . . 3
1.3 Related Work . . . 4
1.3.1 Auto-scaling techniques for elastic applications in cloud environments . . . 4
1.3.2 Resource management in clouds: survey and research challenges . . . 5

1.4 Research Objectives 6
1.5 Contributions 6
1.6 Overview 7
1.6.1 Literature and theory of forecasting methods 7
1.6.2 Implementation of forecasting methods 8
1.6.3 Results: Comparison of forecasting methods 8

2 Literature Study **9**
2.1 Cloud Resource Provision 9
2.1.1 Resource prediction using Exponential Smoothing 9
2.1.2 Resource prediction using Auto-regression 10
2.1.3 Resource prediction using Markov chains 11
2.1.4 Resource prediction using Neural Networks 14
2.2 Summary 16

3 Methods for Forecasting **18**
3.1 Defining Time-series and Forecasting 18
3.2 Moving Average (MA) 19
3.2.1 MA model parameter estimation 20
3.2.2 Forecasting using MA 21
3.3 Exponential Smoothing 22
3.3.1 Exponential Smoothing model parameter estimation . . . 22
3.3.2 Forecasting using Brown's Exponential Smoothing 23
3.4 Holt's Linear Exponential Smoothing 24
3.4.1 Holt's model parameter estimation 25
3.4.2 Forecasting using Holt's Exponential Smoothing 26
3.5 Holt-Winters' Additive Exponential Smoothing 26
3.5.1 Holt-Winters' model parameter estimation 27
3.5.2 Forecasting with Holt-Winters' method 28
3.6 Auto Regression (AR) 29
3.6.1 Simple Linear Regression 29
3.6.2 Linear regression parameter estimation 29
3.6.3 Forecasting with linear regression 31
3.6.4 Autocorrelation Function 31
3.6.5 Auto-regression definition 32
3.6.6 Auto-regression parameters estimation 34
3.6.7 Forecasting with an AR model 36
3.7 Markov Chains 37
3.7.1 First-order Markov chain model 37
3.7.2 First-order Markov chain parameter estimation 39
3.7.3 Forecasting using a first-order Markov model 39
3.7.4 Second-order Markov chain model 40
3.8 Neural Networks 41
3.8.1 Neural Network structure 42

3.8.2 Sigmoid Neuron . . . 43
3.8.3 Learning Neural Networks . . . 44
3.8.4 Forecasting using Neural Networks . . . 47
3.8.5 Recurrent neural networks . . . 47
3.8.6 Elman recurrent neural networks . . . 47
3.9 Summary . . . 49

4 Implementation of Forecasting Methods 51
4.1 Holt-Winters Implementation . . . 51
4.2 Auto-regression Implementation . . . 52
4.3 Markov Chain Implementation . . . 53
4.4 PRESS Implementation . . . 54
4.5 Agile Implementation . . . 54
4.6 Neural Network Implementation . . . 55
4.7 Resource Forecasting Pipeline . . . 55
4.8 Summary . . . 57

5 Experimental Investigation 59
5.1 Experimental Setup . . . 60
5.1.1 Evaluation parameters . . . 61
5.1.2 Datasets . . . 62
5.1.3 Statistical significance test . . . 63
5.2 Investigate PRESS And Agile's Results . . . 64
5.2.1 Motivation . . . 64
5.2.2 Setup . . . 64
5.2.3 Results . . . 65
5.2.4 Interpretation . . . 65
5.3 Evaluation Using Root Mean Squared Error . . . 68
5.3.1 Motivation . . . 68
5.3.2 Setup . . . 68
5.3.3 Results . . . 69
5.3.4 Interpretation . . . 69
5.4 Evaluation Using Correct Estimation Rate . . . 70
5.4.1 Motivation . . . 70
5.4.2 Setup . . . 71
5.4.3 Results . . . 71
5.4.4 Interpretation . . . 71
5.5 Evaluation Using Estimation Score . . . 72
5.5.1 Motivation . . . 72
5.5.2 Setup . . . 73
5.5.3 Results . . . 73
5.6 Evaluation Using Overload Likelihood Ratio . . . 75
5.6.1 Motivation . . . 75
5.6.2 Setup . . . 75

5.6.3 Results . . . 76
5.6.4 Interpretation . . . 76
5.7 Evaluation Using Overloaded State Likelihood Ratio . . . 78
5.7.1 Motivation . . . 78
5.7.2 Setup . . . 79
5.7.3 Results . . . 79
5.7.4 Interpretation . . . 79
5.8 Ensemble Model Evaluation . . . 81
5.8.1 Motivation . . . 81
5.8.2 Setup . . . 81
5.8.3 Results . . . 82
5.8.4 Interpretation . . . 82
5.9 Investigate Shorter Forecasting Window . . . 85
5.9.1 Motivation . . . 85
5.9.2 Setup . . . 85
5.9.3 Results . . . 85
5.9.4 Interpretation . . . 86
5.10 Summary . . . 87

6 Conclusions **89**
6.1 Summary of Work . . . 89
6.1.1 Cloud workloads and forecasting methods . . . 89
6.1.2 Evaluation metrics . . . 89
6.1.3 Experimental investigations . . . 90
6.2 Concluding Perspective . . . 90
6.3 Recommendations . . . 91
6.4 Future Work . . . 92

Appendices **93**

A Derivations **94**
A.1 Yule-Walker Equations . . . 94
A.1.1 For lag of 1 . . . 94
A.1.2 For lag of 2 . . . 95
A.1.3 For lag of k . . . 96

B Datasets **97**
B.1 2011 Google Cluster Dataset . . . 97
B.2 Wikipedia Pageview Dataset . . . 98

C Additional Results **99**
C.1 Statistical Significance Test Results . . . 99
C.2 Ensemble models: Statistical Significance Test Results . . . 100
C.3 Investigate Shorter Forecasting Window . . . 102

Bibliography **109**

List of Figures

1.1 Cloud computing service levels. 3

2.1 Illustration of PRESS. 13
(a) Extract dominant frequency. 13
(b) Calculate average-pattern and forecast the next window. . . 13
2.2 Example of a Wavelet-transform and Agile's method. 15

3.1 Moving Average applied to example data. 20
3.2 Forecasting with Moving Average. 21
3.3 Brown's Exponential Smoothing applied to example data. 23
3.4 Forecasting with Brown's method. 24
3.5 Holt's Exponential Smoothing method applied to example data. . . 25
3.6 Forecasting with Holt's linear method. 27
3.7 Forecasting with Holt-Winters' method. 29
3.8 Simple linear regression. 30
3.9 An Autocorrelation Function (ACF) plot of example data. 33
3.10 Auto-regression model as an IIR filter. 34
3.11 Comparing auto-regression models of increasing order. 35
3.12 Forecasting with an Auto-regression model. 36
3.13 An example of digitising data into Markov states. 37
3.14 Forecasting with a first-order Markov chain. 40
3.15 Neural Networks: The Perceptron. 42
3.16 The unit step function. 42
3.17 A simple Feed-Forward Neural Network. 43
3.18 The Sigmoid Function. 44
3.19 Learning a Neuron Network. 45
3.20 Forecasting using a Feed-forward Neural Network. 48
3.21 An example of a Recurrent Neural Network. 49
3.22 Elman Recurrent Neural Network. 50

4.1 Power density spectrum of our data. 52
4.2 Selection of Auto-regression model order. 53
(a) Z-plane of AR(8). 53
(b) PSD of AR(8). 53

(c) Z-plane of AR(16). 53
(d) PSD of AR(16). 53
(e) Z-plane of AR(30). 53
(f) PSD of AR(30). 53
4.3 Issue: Inspect RNN transient behaviour when forecasting. 56
4.4 Resource Forecasting Pipeline. 58

5.1 PRESS results. 66
(a) CPU usage data. 66
(b) Memory usage data. 66
(c) CPU usage data. 66
(d) Memory usage data. 66
5.2 Agile's results. 67
(a) CPU usage data. 67
(b) Memory usage data. 67
(c) CPU usage data. 67
(d) Memory usage data. 67
5.3 Root Mean Squared Error evaluation results. 70
(a) CPU usage data. 70
(b) Memory usage data. 70
(c) Pageview data. 70
(d) Network data. 70
5.4 Correct Estimation Rates for CPU, Memory, Pageview and Network data 72
(a) CPU usage data. 72
(b) Memory usage data. 72
(c) Pageview data. 72
(d) Network data. 72
5.5 Estimation score results 74
(a) CPU usage data. 74
(b) Memory usage data. 74
(c) Pageview data. 74
(d) Network data. 74
5.6 Overload Likelihood Ratio results. 77
(a) CPU usage data. 77
(b) Memory usage data. 77
(c) Pageview data. 77
(d) Network data. 77
5.7 Definition of an overloaded state. 78
5.8 Overloaded State Likelihood Ratio results 80
(a) CPU usage data. 80
(b) Memory usage data. 80
(c) Pageview data. 80
(d) Network data. 80

5.9 Ensemble models: Root Mean Squared Error results. 82
(a) CPU usage data. 82
(b) Memory usage data. 82
5.10 Ensemble models: Correct Estimation Rate results. 83
(a) CPU usage data. 83
(b) Memory usage data. 83
5.11 Ensemble models: Estimation Score results. 83
(a) CPU usage data. 83
(b) Memory usage data. 83
5.12 Ensemble models: Overload Likelihood Ratio results. 84
(a) CPU usage data. 84
(b) Memory usage data. 84
5.13 Ensemble models: Overloaded State Likelihood Ratio results. . . . 84
(a) CPU usage data. 84
(b) Memory usage data. 84

C.1 Comparison forecasting window lengths on RMSE 102
(a) CPU usage data. 102
(b) Memory usage data. 102
C.2 Comparison forecasting window lengths on Correct Est. Rate . . . 103
(a) CPU usage data. 103
(b) Memory usage data. 103
C.3 Comparison forecasting window lengths on Estimation score 106
(a) CPU usage data. 106
(b) Memory usage data. 106
C.4 Comparison forecasting window lengths on Overload Likelihood Ratio 107
(a) CPU usage data. 107
(b) Memory usage data. 107
C.5 Comparison forecasting window lengths on Overloaded State Likelihood Ratio 108
(a) CPU usage data. 108
(b) Memory usage data. 108

List of Tables

5.1 Evaluation Parameters. 61
5.2 Comparison forecasting window lengths results. 86

C.1 Statistical significance test results for the evaluations performed on the 2011 Google Cluster and Wikipedia datasets. 99
C.2 Ensemble models: Statistical Significance Test Results. 100
C.3 Investigate forecasting window lengthe results. 103

Nomenclature

Acronyms

ACF	The Autocorrelation Function used to determine stationarity of a time series and estimate the Auto-regression coefficients.
ANN	Artificial Neural Network.
AR	Auto-regression.
BFGS	The Broyden-Fletcher-Goldfarb-Shanno optimisation algorithm, together with the MSE are used to estimate the parameters of exponential smoothing models.
CER	Correct Estimation Rate.
ES	Estimation Score, a linear combination of the OER and UER.
FFNN	Feed-Forward Neural Network.
FPR	The False Positive Rate.
HW	Holt-Winter exponential smoothing.
IIR	Infinite Impulse Response.
LP	Linear Predictor.
LPA	Linear Prediction Analysis, a feature extraction technique popular in signal and speech processing.
LR+	Positive Likelihood Ratio.
LSE	Least Squares Estimation.
MA	Moving Average.
MLP	Multi-Layer Perceptron.
MSE	Mean Squared Error.
NN	Shorthand for Artificial Neural Network.
OER	Over-Estimation Rate.
OLR	Overload Likelihood Ratio, the positive likelihood ratio associated with correctly predicting overloaded samples.
OSLR	Overloaded State Likelihood Ratio, the positive likelihood ratio associated with correctly predicting overloaded states.
PGM	Probabilistic Graphical Model.

RFP	Resource Forecasting Pipeline.
RNN	Recurrent Neural Network.
SSE	Summary of Squared Error.
TPR	The True Positive Rate.
UER	Under-Estimation Rate.
WA	Weighted Average, used as method of combining forecasting methods.
WMA	Weighted Moving Average.

Symbols

Forecasting Methods

a_i	The model parameters of a Linear Predictor (LP).
$\hat{y}_{t+1}$	The predicted value for a time series at time $t+1$.
$\hat{s}(t)$	Approximated signal of a Linear Predictor.
m	The number of past values used in Moving Average model.
α	Level smoothing factor for exponential smoothing.
β	Trend smoothing factor for Holt's linear smoothing method.
γ	Seasonal smoothing factor for Holt-Winters' additive smoothing method.
s_t	The estimate of the level for a time series at time t.
b_t	The estimate of the trend for a time series at time t.
I_t	The estimate of the seasonal component of a time at time t.
L	The number of observations per season when modelling a time-series using Holt-Winters' method.
A_I	The the average of at time-series for the jth season in that time-series.
ϕ_0	Intercept parameter for a linear regression model.
ϕ_i	Model parameters for a linear regression or auto-regressive model with $i = 1, 2, ...$
E	The Mean Squared Error function.
$\mathbf{E}$	The Expected Value operator.
ϵ	Modelling error.
δ	A constant in the Auto-regression model.
ρ_m	The value of the Autocorrelation function at delay m.
$\mathbf{S}$	The set of distinct states used when fitting a Markov chain model.

x_i	A discrete Markov chain state, used to indicate the current state.
x_j	A discrete Markov chain state, used to indicate the next or new state.
p_{ij}	The transition probability for transitioning from a current state x_i to a new state x_j.
$\mathbf{P}$	Transition matrix (of size $k \times k$) for a Markov chain model, describing the probabilities of transitioning for any one state to any other state.
π_t	The probability distribution, at time t, across all states of a Markov chain model.

Neural Networks

$\mathbf{x}$	Input vector to a neuron with components x_i.
$\mathbf{w}$	The weight-vector which is multiplied with the input vector $\mathbf{x}$ and passed to the activation function.
b	The bias value or threshold at which a perceptron neuron activates.
$\mathbf{T}$	Training set of inputs and desired or **target** output pairs.
d_j	Target or desired output for input vector $\mathbf{x}_j$
Δw	Small changes to the weights in the neural network.
$\Delta \hat{y}$	Small changes to the network's output.

Evaluation Metrics

S_p	Scaling parameter used in statistical significance test.
Q	The duration, in samples, defining a overloaded state.
T_p	The true positive count.
F_p	The false positive count.
T_n	The true negative count.
F_n	The false negative count.

Chapter 1

Introduction

1.1 Motivation

The emergence of cloud computing and the adoption of elastic cloud services have enabled developers to host applications and services on cloud hosted resources. These resources can also be dynamically provisioned and scaled on demand. Effective provisioning of cloud resources, i.e. ensuring that sufficient resources are available when needed, has proven to be a challenging task for cloud users [61]. This is because applications hosted in the cloud typically face large amounts of traffic and unpredictable workloads due to end user behaviours [62]. Under-provisioning of resources hurts performance and may violate Service Level Agreements (SLAs) with end users, whereby over-provisioning of resources may incur unnecessary costs [43].

As a solution to this challenge, recent research has presented promising provisioning and auto-scaling schemes. Proactive provisioning aims to map performance requirements to the underlying cloud resources, employ forecasting methods to accurately estimate the resource requirement (or quantitative load) ahead of time and scale resources accordingly.

Forecasting methods adapted from the fields of statistics and machine learning has been applied to cloud resource provisioning. Much effort has been spent in improving the modelling and forecasting accuracy of these methods.

According to Lorido-Botrán et al. [42], there is a lack of formal investigation and comparison of these forecasting methods and their performance. Furthermore, Kupferman et al. [38] state that "representative metrics will have to emerge in order to realistically evaluate different scaling approaches."

The purpose of this thesis is to perform a formal investigation in comparing forecasting methods used in provisioning of cloud hosted resources. Performing evaluations using representative performance metrics and real-world datasets.

1.2 Background

1.2.1 Cloud computing

The idea of publicly available computing resources was first envisioned by John McCarthy in early 1960. The term 'cloud' was first used in 2006 by Google's CEO Eric Schmidt to describe the business model of providing computing resources and services over the Internet [69]. The National Institute of Standards and Technology (NIST) [44] defines cloud computing as: "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Cloud computing has become synonymous with the Internet as we know it today. A principle analyst at Rackspace, Roy Illsley, stated in his 2014 report [33] that 42% of small to medium enterprises globally use cloud computing. He predicts this number to reach 75% by the year 2016. Cloud computing is continuing to foster the development of new and emerging technologies. Developers of cloud based services and applications no longer need to make a large upfront investment in hardware or operation costs and are able to scale their cloud infrastructure according to the popularity of their product or service [5].

1.2.2 Cloud service levels

In general, the cloud computing environment can be categorised into three distinct service levels, namely:

- Infrastructure as a Service (IaaS): providing of virtual resources, referring to the simulation of computer hardware on physical computers within a datacenter. These include Virtual Machines (VMs), large scale storage, firewalls, load balancers, Virtual Local Area Networks (VLANs) and management software [3]. Examples of IaaS clouds include Amazon's Elastic Compute Cloud (EC2) and Google Compute Engine (GCE).

- Platform as a Service (PaaS): providing a platform and services to support application development and design. These include operating system support and software development environments [69]. Examples of PaaS clouds include Google Cloud Platform and Elastic Beanstalk.

- Software as a Service (SaaS): providing software applications to users over the Internet, typically only accessed online [27]. Examples of SaaS include Google's Gmail [25], Dropbox [16] and Online games.

Figure 1.1 illustrates these service levels and lists examples of the types of applications or resources provided at each level. For the purpose of this work

we will focus on IaaS type cloud resources, because of the availability of VM resource metrics at this level.

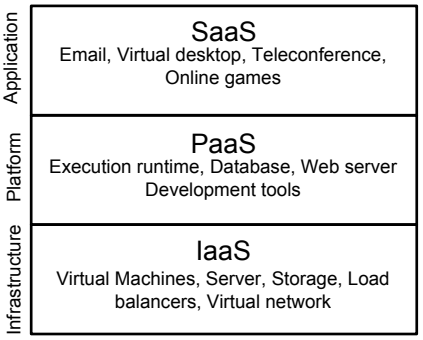


Figure 1.1: The service levels provided by cloud computing and examples of the types of applications and resources provided at each level.

1.2.3 Actors in the cloud environment

Following the terminology used by Jennings and Stadler [34], three separate parties are involved in the cloud environment and these are outlined as the following:

- The **cloud provider** manages a set of physical datacenter hardware and system software resources to provide cloud resources to cloud users, available on-demand and pay-for-use. The cloud provider is responsible for allocating cloud resources meeting Service Level Agreements (SLAs) with cloud users. Cloud providers such as Google and Amazon provide elastic cloud solutions, i.e. the ability to dynamically acquire and release cloud hosted resources, namely Google Compute Engine (GCE) [26] and Amazon EC2 [2].
- The **cloud user** uses cloud infrastructures to host applications or services and offers it to end users. Cloud user are typically concerned with minimising their running costs whilst maximising income from End users.
- **End users** use applications or services hosted on cloud resources and generates the workload processed by cloud resources.

1.2.4 Cloud workloads

According to Mao and Humphrey [43], the workloads presented to clouds can contain long-term variations such as time-of-day effects as well as short-term fluctuations. They characterise cloud workloads into four types: stable, trending, seasonal/cyclic and bursty.

- A **stable** workload is characterised by resources having a constant load for a long period. Example scenarios that present stable workloads include: cloud monitoring and logging services as well as research clusters running a series of batch jobs.
- A **trending** workload is observed when the load on a cloud is increasing over time, typically causing overload. Example scenarios of trending workloads include: a particular website or web service becoming more popular over time and the increasing number of users generating more load.
- **Seasonal/cyclic** workloads are characterised by having periodic elements. Example scenarios that present cyclic workloads include: online retailers where higher workloads are observed by day as opposed to lower workloads by night.
- A **bursty** workload is characterised by a sudden increase in load. An example scenario that presents a bursty workload includes: the increase in the number of views to a news site reporting on a breaking story.

1.3 Related Work

The work presented in this thesis is an investigation in comparing forecasting methods for cloud hosted resources. In this section we discuss recent comparisons of auto-scaling and provisioning methods for cloud hosted resources. This will give the necessary context for our work.

1.3.1 Auto-scaling techniques for elastic applications in cloud environments

In their recently published technical report, Lorido-Botrán, Miguel-Alonso and Lozano [42] investigate auto-scaling in the cloud environment by identifying the role-players in the cloud environment and the types of auto-scaling techniques being used in elastic clouds.

They contribute by listing different workloads (both synthetic and real world traces), possible application benchmarks and auto-scaling techniques that can be used for research.

Similar to the work presented in this thesis, Lorido-Botrán et al. classify auto-scaling techniques into six categories namely, static, threshold-based, reinforcement learning, queuing theory, control theory and time-series analysis.

For each class of auto-scaling method, Lorido-Botrán et al. review recent work that investigated the use of that method in the cloud domain and list the metrics, workloads and experimental platform used.

They conclude that for efficient scaling (and provisioning) of cloud resources, a predictive auto-scaling technique needs to be developed that could model and forecast on time-series data. They also highlight that the accuracy of auto-scaling techniques investigated greatly depend on the modelling parameters. Finally, the authors state that **there is a lack of a formal testing and comparison framework for auto-scaling in the cloud.** It is this last statement that serves as the basis for the work performed and presented in this thesis.

1.3.2 Resource management in clouds: survey and research challenges

Work similar to Lorido-Botrán et al. and inspiration for the work in this thesis is: "Resource Management in Clouds: Survey and Research Challenges" work by Jennings and Stadler [34] published in 2014.

The authors survey recent literature on cloud resource management, scaling and provisioning and compile a list of state-of-the-art methods for each of these topics. They identify five research challenges in cloud computing resources and systems that need to addressed, namely: providing predictable performance for cloud hosted applications, achieving global manageability for cloud systems, engineering scalable resource management systems, gaining an understanding of cloud pricing and economic behaviours and lastly developing solutions for the mobile cloud paradigm.

The work by Jennings and Stadler is a broader study into cloud resource management than the work done in this thesis. The sections in their paper on *Resource Demand Profiling*, *Resource Utilisation Estimation* and *Application Scaling and Provisioning* have the closest similarities to our work and will be discussed here.

Under *Resource Demand Profiling*, Jennings and Stadler investigate proactive, model-driven and model-free forecasting methods and identify similar types of methods, as presented in this thesis (see Chapter 2). These methods include time-series analysis approaches like auto-regression and PRESS as well as energy aware applications.

In the section entitled *Resource Utilisation Estimation*, Jennings and Stadler highlight that the majority of cloud resource management research depend on historical measurements which may be noisy and inaccurate. They thus suggest that better profiling of cloud application workloads would be beneficial for more accurate provisioning.

From their discussion on *Application Scaling and Provisioning* we see similarities to the cloud workloads described above. Jennings and Stadler present scaling and provisioning techniques that include approaches that use fuzzy logic, decentralised algorithms and probabilistic models.

In conclusion, Jennings and Stadler identify the following research challenges that need to be addressed: (1) More efficiency in resource placement (on physical datacenter hardware), (2) performance prediction of mutli-tiered applications, with specific focus on use cases and system constraints associated with these types of applications. (3) The use of Control theory as resource allocation technique, because these techniques have seen success in other applications outside of the cloud domain. (4) **Emphasis should be put on increasing the accuracy of forecasting methods. This critically affects the performance of proactive provisioning.** Load/demand should be classified and characterised in different scopes and different timescales whereby this information can be fed into the models being used.

In a broad sense, the work presented in this thesis investigates and aims to address challenge (4) by evaluating and comparing different types of forecasting methods and investigating the metrics used to evaluate forecasting accuracy.

1.4 Research Objectives

The objectives of this work is to:

1. Survey the field of cloud resource provisioning and scaling to identify prominent forecasting methods used to model and estimate the load presented to resources in the cloud.

2. Identify key performance measures from this survey that are used to evaluate and compare provisioning methods.

3. Compare the prominent forecasting methods identified through experimental evaluation, i.e. using datasets, evaluation parameters and performance metrics.

4. Investigate the increase in forecasting accuracy when combining methods that each address a characteristic of cloud workloads into an ensemble model.

5. Investigate the effects on performance when using a shorter forecasting window length.

1.5 Contributions

- Present a formal experimental investigation framework for evaluating and comparing forecasting methods towards more effective cloud resource provisioning.

- Conclude that there is no single forecasting method that is significantly better than the rest in terms of accurately forecasting load presented to cloud hosted resources.
- Show that there is no one performance metric that gives a concise result when evaluating and comparing forecasting methods on cloud usage data.
- The work presented in the thesis has been accepted for publication and will be presented at the 11th International Conference on Network and Service Management 2015 (CNSM '15).

1.6 Overview

This chapter gave an overview of the research done in this thesis. Section 1.1 states the motivation and basis for this work.Section 1.2 covers the necessary background of cloud computing and how provisioning of cloud resources, especially when using elastic cloud services, is a challenging task. A synopsis is given in Section 1.3 of work that relates to this research, and is followed by Section 1.4 which describes the objectives of this research. A summary of the contributions is given in Section 1.5.

The rest of the chapters in this thesis can be summarised into the following subsections:

1.6.1 Literature and theory of forecasting methods

In Chapter 2, a study is performed to identify prominent forecasting methods used in recent literature on provisioning and auto-scaling of cloud hosted resources. Eight forecasting methods are identified: (1) Moving Average, (2) Exponential Smoothing, (3) Auto-regression, (4) Markov Chains, (5) PRESS, (6) Agile, (7) Feed-Forward Neural Networks and (8) Elman-Recurrent Neural Networks. The chapter concludes that the squared error is a generic metric used when evaluating forecasting methods, and more importantly, that there exists no formal investigation or agreement on modelling of forecasting methods or what evaluation setup or datasets to use when comparing forecasting methods.

Background and theory required for developing and implementing the forecasting methods are described in Chapter 3. The chapter builds basic intuition by first describing simpler methods such as Moving Average (MA) and smoothing functions such as Exponential Smoothing, whilst highlighting similarities to Linear Prediction Analysis (LPA) throughout. The complexity of methods described increase from the Holt-Winters Exponential Smoothing method, to Auto-regression (AR) that employs the Autocorrelation function (ACF), to Markov Chains that model transitions across distinct values and finally to Neural Networks that learn functional relationships between past values and

future loads. Two types of Neural Networks (NNs) are important for the work in this thesis. They are Feed-Forward Neural Networks (FFNN) and Elman-Recurrent Neural Networks (RNNs).

1.6.2 Implementation of forecasting methods

Chapter 4 covers the implementation details of the forecasting methods investigated in this thesis, highlights the issues encountered and discusses how these were resolved. The chapter starts by mentioning that Python 2.7 is used for developing the forecasting methods and continues to describe method specific implementations and issue resolutions. Next, the chapter lists the differences between and assumptions used with development of PRESS [24] and Agile [47]. The chapter concludes by describing the development of a Resource Forecasting Pipeline (RFP). A formal investigation framework that facilitates data pre-processing, forecasting method modelling and evaluation metrics calculation. This pipeline allows for repeatable experiments to be performed.

1.6.3 Results: Comparison of forecasting methods

The final chapters of this thesis report on the experimental investigation performed on comparing forecasting methods as well as additional evaluations done. Chapter 5 starts of by discussing the experimental setup and evaluation parameters used throughout the investigation. The datasets used and statistical significance tests employed are also covered.

Firstly, the chapter compares PRESS and Agile's results reported by their respective authors to three AR models, each of increasing order. The 7 hour Google cluster dataset [28] is used to investigate PRESS's results and the 29 day dataset used for Agile's comparative evaluation.

The chapter continues to report on the evaluations executed using the five evaluation metrics, namely RMSE, Correct Estimation rate, Estimation score, Overload Likelihood Ratio and Overloaded State Likelihood Ratio. Additional evaluations were performed to investigate the use of combinations of methods in ensemble models. The investigation yields unexpected and inconclusive results. Finally, investigation into using a shorter forecasting window is performed and it confirms the comparative metric evaluations.

The thesis concludes in Chapter 6 by summarising the work done, emphasising the important results and findings, noting the limitations and recommending the future directions for the work.

Chapter 2

Literature Study

2.1 Cloud Resource Provision

In this chapter we identify prominent approaches to proactive provisioning in the cloud environment published in recent years. Each section in this chapter gives a summary of the work presented in a particular literature paper and comments on the limitation of the work, assumptions that where made by the authors or highlights details which were left unclear.

Proactive provisioning is defined as resource provisioning that forecasts server load ahead of time and reserves resources accordingly. As mentioned in Section 1.3.1, Lorido-Botrán et al. [42] classify provisioning and auto-scaling techniques into five categories: static, threshold-base, reinforcement learning, queuing theory, control theory and time-series analysis.

In this thesis we choose to focus on two classes of forecasting methods predominately used in the provisioning of cloud resources, namely machine learning and time-series analysis. These fields looks most promising above others.

2.1.1 Resource prediction using Exponential Smoothing

We identify Exponential Smoothing from the work done by Huang, Li and Yu in their paper entitled; "Resource Prediction Based on Double Exponential Smoothing in Cloud Computing" [29] from 2012. Huang et al. propose a time-series analysis prediction model based on Exponential Smoothing. They specifically investigated a Double Exponential Smoothing model, referred in this thesis as Holt's method. Double Exponential Smoothing employs two smoothing equations (one to estimate the level and another to estimate the trend of a time-series) and uses a linear combination of these to predict a value into the future.

Huang et al. aimed to improve accuracy of resource estimation in proactive provisioning by considering current and recorded data. They describe the mathematical development of Exponential Smoothing up to the formulation

of the prediction equation for forecasting m values into the future. In this thesis, these are described in Section 3.4. Huang et al. use two resource data types namely, CPU and Memory and propose the Summary of Squared Error (SSE) as evaluation metric. They evaluate their Double Exponential Smoothing method using a cloud simulator, *CloudSim* and compare their method to a simple mean- and Weighted Moving Average (WMA) method. They show that their method can better follow resource utilisation and predict future values with more accuracy compared to the mean and WMA.

Comments: From the work by Huang et al. the following was unclear:

- Modelling and evaluation parameters — these include the look-back window length, (i.e. the number of historical samples used to estimate the smoothing coefficients).
- The simulation time used and the forecasting window length — the number of values predicted into the future.
- Setup and implementation of the two comparison methods — simple Mean and WMA are not discussed. The absence of this makes it difficult for future verification of their results.

2.1.2 Resource prediction using Auto-regression

We identify Auto-regression (AR) as a forecasting method from work done by Chandra, Gong and Shenoy [13] and Kupferman et al. [38].

Chandra et al. propose a time-series analysis method that dynamically provisions cloud resources in shared datacenters by using online measurements. By developing a time-domain queuing model they aim to capture and model transient behaviours from cloud applications. They propose an AR model of order 1 — denoted as AR(1). They use this model as prediction algorithm for forecasting short-term application workload requirements.

Using their queuing model, Chandra et al. estimate the specific workload from an application's service requests and relate this to resource utilisation of that application. An online monitoring module captures these resource measurements and stores the most recent historical observations, which is used to fit the AR(1) model.

Chandra et al. evaluate their resource prediction method under simulated conditions (using a Poisson distribution as workload generator) and perform a trace driven investigation using the 1998 World Cup Soccer server logs [4]. They compare their method to static resource allocation and show that their AR(1) model better provisions resources and lowers over-utilisation.

In 2009, Kupferman et al. also proposed the use of a first-order AR model to predict system load in their paper entitled; "Scaling into the Cloud" [38]. They design a repeatable evaluation environment in the form of a cloud simulator

platform that maps incoming requests to CPU utilisation and simulates network traffic for various workload patterns. Kupferman et al. follow the same AR formulation as Chandra et al. and this is also covered in Section 3.6 of this work. They compare their AR(1) method to a static provisioning scheme (where the minimum number of Virtual Machines (VMs) are determined to achieve 100% of the peak utilisation) to a simple linear regression approach and RightScale [54]. RightScale is a provisioning platform that uses a voting scheme among VMs to determine if more resources need to be provisioned or not.

In terms of evaluation metrics, Kupferman et al. propose the use of a scoring algorithm, which considers the number of service requests dropped compared to the total number of requests received. This calculates the running cost (in USD) for each VM in operation. Using these metrics, Kupferman et al. show that static provisioning is most wasteful in terms of resources and that RightScale performs similar to linear regression. They also show that their AR(1) model achieves the best score and cost result.

Comments: Both Chandra et al. and Kupferman et al. choose to employ an AR(1) model to perform short-term forecasts and have proven it to be an efficient model. The paper by Chandra et al. is unclear on the specifics of their AR(1) parameters as well as the ranges of the performance metrics. One might question the relevance of using the 1998 World Cup Soccer server logs [4] for a data trace as representation of modern day cloud workloads. The evaluation metrics used by Kupferman et al. (score and running cost) are both metrics that are relevant in terms of the cloud domain, but requires a cloud platform to be measurable. In this thesis we choose to use time-series based accuracy metrics that relate to both time-series and cloud resources. This is discussed as part of the Experimental Investigation in Chapter 5.

2.1.3 Resource prediction using Markov chains

The use of Markov chains to predict time-series data has been investigated in other fields than cloud provisioning, but in recent years it has also been applied to provisioning of cloud resources. We identified three literature papers that propose Markov chains as forecasting method, namely work done by Lili et al. [39], PRESS by Gong, Gu and Wilkes [24] and Agile by Nguyen et al. [47]. In this thesis we reference both PRESS and Agile extensively and thus we discuss each in Section 2.1.3.1 and Section 2.1.3.2 respectively.

Lili et al. present their work in a paper entitled; "A Markov Chain Based Resource Prediction in Computational Grid" and propose the use of a first-order Markov chain for modelling and predicting cloud resources. They define five Markov states namely, (1) CPU over-utilisation, (2) CPU normal utilisation, (3) Network overload, (4) Network normal load and (5) Resource failure,Ṫhey aim to model the transitions between these states. Their model's Transition matrix P is estimated from historical observations using the frequency of state

transitions (similar to the description given in Section 3.7). They describe an accuracy metric based on the probability of the model predicting the correct state at each time interval and calculate the average over the entire evaluation time. Using a grid simulator platform, GridSim [9], they evaluate and compare their method to a simple mean and median based approach. They are able to show that their Markov chain model achieves higher prediction accuracies across various data-traces.

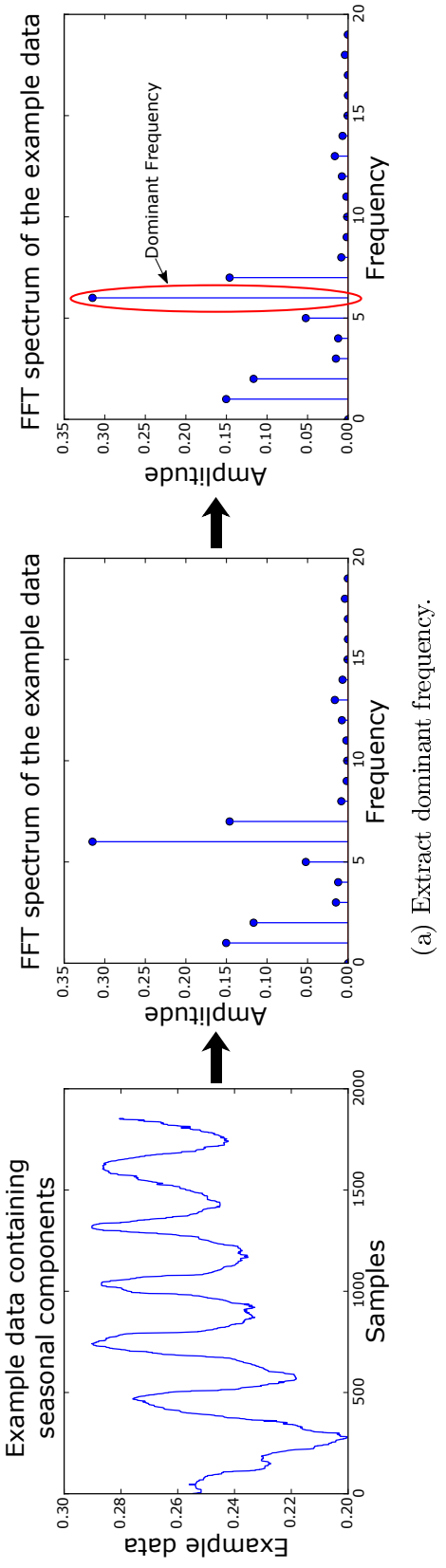
Comments: It is important to note that Lili et al. focussed their work on Grid computing, which is a subtype of cloud computing and is typically used for research jobs and batch processing. These traces may present different types of workloads compared to those presented to commercial clouds, the focus of this thesis. The use of five Markov states that relate to overload and under-load is an interesting design decision. This enables their Markov chain model to learn cloud specific features and be less impacted by time-series values. This approach still requires the cloud user to define her application specific over- and under-load thresholds, which again could be a difficult task to perform.

2.1.3.1 PRESS

Gong, Gu and Wilkes present "PRESS: PRedictive Elastic reSource Scaling for cloud systems" [24], a two fold provisioning scheme that uses both a time-series analysis and a machine learning approach to accurately predict short-term load changes. Using the Fast Fourier Transform (FFT), PRESS calculates the dominant frequency present in historical resource demand data and calculates a window containing a signature-pattern. Figure 2.1 on page 13 illustrates how PRESS uses the signature-patterns as reference, calculates an average-pattern and, using Dynamic Time Warping, finds the offset in order to forecast these values for the next window.

In cases where the past observations do not contain a significant repeating pattern, they employ a discrete first-order Markov chain. For this, Gong et al. define M-distinct Markov states by dividing the data into equal-sized discrete bins. Using the frequency count of state transitions, they construct a transition matrix P. We follow this approach for our Markov chain as described in Section 3.7.

They evaluate PRESS in simulation using the 1998 World Cup Soccer server logs [4] and Google's 7-hour workload cluster dataset [28] as real-world data. Gong et al. propose using under- and over-estimation rates as evaluation metrics and shows that PRESS outperforms comparative methods such as mean-max, auto-correlation and auto-regression.



(a) Extract dominant frequency.

(b) Calculate average-pattern and forecast the next window.

Figure 2.1: Illustration of how PRESS forecasts using a signature-pattern scheme. The dominant frequency and period is determined using the FFT and used to segment the original time-series into signature-patterns. An average-pattern is calculated using the information from each window, and using Dynamic Time Warping the offset of the current pattern to the average-pattern is determined. Values for the next forecasting window is predicted as this (shifted) average-pattern.

Comments: The 1998 World Cup Soccer server logs, may not be a representative dataset workloads presented to modern day cloud resources. The specific implementation of PRESS is still unclear as well as the three comparative methods used, making it difficult to fully investigate their proposed method. We identify under- and over-estimation rates as performance metrics that relates to both time-series and cloud provisioning accuracy.

2.1.3.2 Agile

Nguyen et al. [47] extend PRESS and propose *Agile* — a time-series analysis method which uses Wavelet-transforms to perform medium-term resource demand predictions. Wavelet-transforms decompose a time-series into a set of detail-signals at different scales, with each detail-signal representing the original time-series at a coarser granularity. Figure 2.2 on page 15 (taken from Nguyen's paper) illustrates an original time-series decomposed into four scaled detail-signals.

After subtracting the detail-signals from the original signal we obtain an approximation signal. As illustrated, forecasting is performed on each of these detail- and approximation-signals independently and using the inverse-wavelet transform, a prediction is synthesised on the original signal. Nguyen et al. employ a Markov chain model similar to PRESS for modelling and forecasting on each of the detail- and approximation-signals.

Nguyen et al. propose using overload prediction rates and overloaded state accuracy as evaluation metrics (also used in this thesis and described in detail in the Experimental Investigation in Chapter 5). They evaluate Agile on the 29 day Google cluster dataset [67] and compare it to PRESS and auto-regression, showing that Agile consistently outperforms these two methods when evaluated on CPU and Memory resource demand.

Comments: In order to perform a Wavelet-Transform, one chooses the type of wavelet and the number of scales to use. Nguyen et al. chose the number of scales according to the forecasting window, but omitted information about the specific set of wavelet functions they used in Agile. The order of the Auto-regression model used by Nguyen et al. as comparison model is unknown. The performance metrics Nguyen et al. proposed are measures that relate to time-series and cloud domain resources and thus are closer to realistic metrics for evaluating cloud provisioning methods. We choose to use both these metrics in this thesis.

2.1.4 Resource prediction using Neural Networks

We identify Neural Networks, a machine learning approach to resource prediction, through the work done by Caglar and Gokhale [11] and Nae, Iosup and Prodan [45].

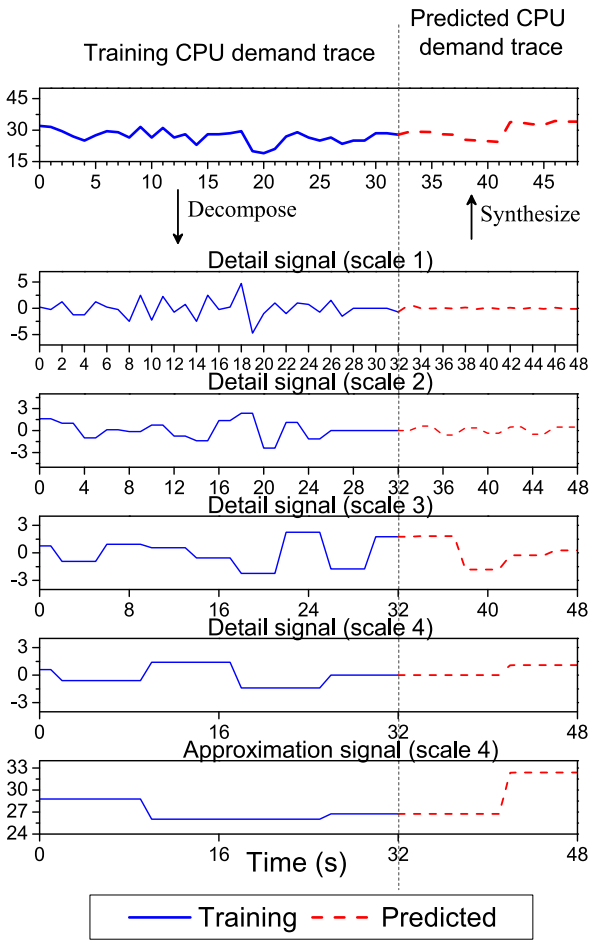


Figure 2.2: An example of how a time-series signal is transformed into detail signals of different scales using Wavelet-Transforms. This illustrates how Agile [47] forecasts CPU demand.

Caglar and Gokhale present *iOverbook*, an intelligent resource management tool that uses an Artificial Neural Network to predict overbooking rates in datacenters. They identify 'features' associated with resource allocations which include CPU requests and usage, Memory requests and usage, VM count, Memory capacity and CPU and Memory overload rates. These features are input into a Feed-Forward Neural Network (FFNN). Using the Levenberg-Marquardt backwards-propagation algorithm, the FFNN is trained and learns the functional requirements of each of the features and how they relate to resource allocation/provisioning. They use the 29 day Google cluster dataset [67] and the Mean Squared Error (MSE) as evaluation metric to evaluate their method's ability to forecast the mean hourly CPU and Memory usage. Caglar and Gokhale show that their method can accurately predict the next interval with a statistical R-value of 0.67.

Nae et al. propose a prediction algorithm that uses a Recurrent Neural Network (RNN) to predict load on a cloud hosted Massively Multiplayer Online Game (MMOG). Each VM is hosting a partition of the in-game state and the load on that VM is modelled as the number of entities in that region. In terms of workload types, Nae et al. define four player-behaviour-patterns, each presenting a different workload to the cloud resources. They employ an Elman network (which is a type of RNN) to estimate the load ahead of time and dynamically provision and scale the cloud resources used by their MMOG.

They develop a cloud based MMOG simulator, use real-world data traces of player-behaviours and the Mean Absolute Error (MAE) as performance metric to evaluate their estimator. Nae et al. show that their NN-based method accurately predicts various loads including flash-crowd behaviours.

Comments: Typical resource provisioning uses the time-series data as training data but the approach taken by Caglar and Gokhale is different. They define 'features' that are associated with resources rather than using the historical values as time-series data. Nae et al. understand that overload is a state-based condition and thus opts to use a RNN that is capable of 'remembering' state. In our work we implement both FFNNs and RNNs and investigate the capabilities of both these neural network approaches.

2.2 Summary

In this chapter we identified the prominent forecasting methods used in recent literature and discussed the work in which these where presented. The methods identified were Exponential Smoothing, Auto-regression, Markov Chains, PRESS, Agile and two types of Neural Networks (Feed-Forward Neural Networks and Elman-Recurrent Neural Networks).

We conclude with the following remarks:

- The squared error (or variants of it) is a popular evaluation metric used when measuring the performance of forecasting methods.
- There is no agreement on training- or prediction window-lengths when modelling or forecasting resource demand.
- The forecasting methods are primarily compared with naive models and not against other prominent approaches (with the exception of Agile being compared against PRESS). This supports the motivation of this thesis, to perform a formal investigation into comparing prominent forecasting methods in the same evaluation environment and dataset(s).
- The literature study showed that popular datasets used in evaluations include the 1998 World Cup Soccer server logs [4], the 7 Hour Google cluster dataset [28] of 2010 and the 2011 Google cluster dataset [67]

covering 29 days. For the purpose of the work performed in this thesis, we opt to use the most recent datasets: The 29 day Google cluster dataset [67] of 2011 and the Wikipedia Pageview dataset [65] of 2014.

The next chapter covers the formulation of theory and background knowledge required to model each of these forecasting methods identified. We also present the forecasting equations for predicting multiple values in to the future.

Chapter 3

Methods for Forecasting

The previous chapter identified prominent forecasting methods used in provisioning and auto-scaling schemes applied to cloud hosted resources. This chapter covers the background theory needed to better understand each forecasting method and highlights the strengths and weaknesses of these methods. First, we give the definition of a time-series, describe modelling of a time-series and discuss the procedure of forecasting. The chapter continues to discuss simpler forecasting methods like Moving Average and various Exponential Smoothing methods as well as cover methods of higher complexity such as Auto-regression, Markov Chains and finally Neural Networks.

The purpose of this chapter is to give an indication of the complexity of each forecasting method, describe how the method parameters are estimated from training data and formulate the equations used to forecast multiple values into the future. The formulations and theory presented in this chapter, was collected from the following resources: Croarkin and Tobias [15], Hyndman and Athanasopoulos [30], Robert Nau [46] and Kalekar [36].

3.1 Defining Time-series and Forecasting

A **Time-series** is defined as a set of sequential data points or measurements collected at regular time intervals. Time-series data is typically observed when monitoring industrial processes, business- or economic metrics [15]. For the purpose of this thesis, time series data is obtained when monitoring cloud resource utilisation. Quantitative forecasting involves the analysis and modelling of time-series data, using mathematical methods. A **model**, as defined in [22, p.15], is a mathematical description of a process that generates a given time-series, whereby **forecasting** is defined as a procedure where historical data is fed into a model as input and the output produced by the model is the prediction or estimate.

Provisioning of cloud resources can been viewed as a time-series forecasting problem with past observations (or measurements) of usages being modelled

and future estimation of load being forecasted.

3.2 Moving Average (MA)

Measurement data and specifically time-series data generated by processes, have inherent random variations that can be contributed to sensor noise or to the stochastic nature of the process being monitored. The effects causing these random variations can be reduced by using *Smoothing Functions*. Two prominent smoothing methods used in time-series and signal processing research are averaging methods (e.g. Moving Average) and Exponential Smoothing (discussed in Section 3.3). These methods are both simple and flexible, making them effective at revealing the underlying trends, seasonal and cyclic components in time-series.

The simplest way to smooth noisy data is to take the average of all past data values, using the equation of the weighted average:

$$\bar{y} = \frac{\sum_{t=0}^{N-1} y_t}{N} \tag{3.2.1}$$

where the weight N is the total number of past values.

The objective of smoothing is to reduce short-term fluctuations and highlight long-term trends or cycles. It is more beneficial to calculate the average of consecutive sets of observations within a smoothing window n (with n being smaller than the total number of values N). This method is termed Moving Average (MA), because the sample window is moved after each calculation of the average.

MA's expression is given by:

$$s_t = \frac{\sum_{i=1}^{n} y_{t-i}}{n} \tag{3.2.2}$$

where s_t is the smoothed value at time t and $\mathbf{s}$ the new time-series containing the smoothed values. The strength of smoothing, also referred to as the *weight of smoothing*, is controlled by the size of n. Larger n will highlight more of the long-term trends and seasonality, whereby smaller values of n will conserve short-term fluctuations.

Figure 3.1 illustrates two MA models applied to example data. We notice that no smoothing is applied to the first n observations, as these values are used to smooth the observation at sample $t = n$. To align the smoothed values with the variations of the data, one could calculate the *Centred Moving Average* using an equal number of values on either side of the current value y_t. This approach assumes we have full knowledge of values in the past and into the future, which for forecasting on time-series is not the case. For the purpose of this thesis we will use MA as formulated in equation 3.2.2.

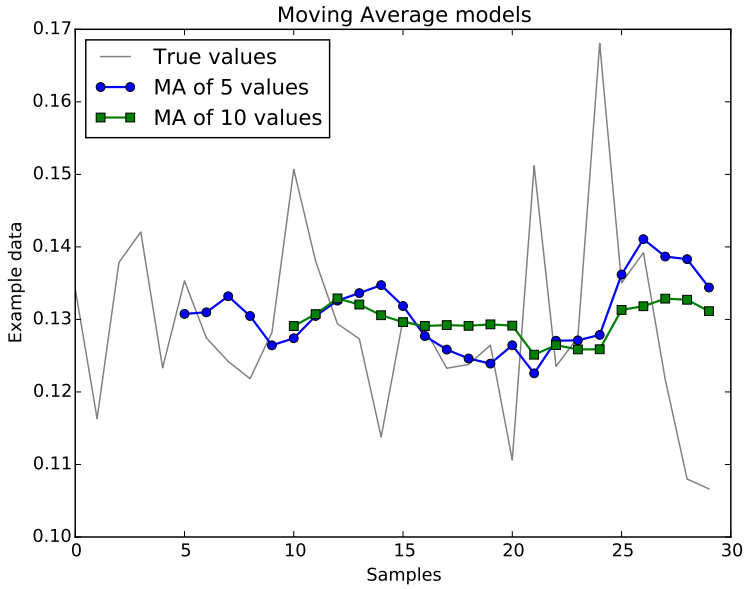


Figure 3.1: Moving Average applied to a noisy time-series with smoothing window sizes n, set to 5 and 10 data points respectively. We notice that no smoothing is applied to the first n observations, because these values are used to smooth the value at $t = n$.

3.2.1 MA model parameter estimation

Different to the other forecasting methods discussed in this chapter, Moving Average does not require modelling or estimation of model parameters. But when reviewing equation 3.2.2, we see similarities to *Linear Prediction Analysis* (LPA), a feature extraction technique from the field of signal processing (and especially speech processing).

The basic assumption of LPA is that future values can be represented by a linear combination of past values. A signal, $s(t)$ can be approximated to $\hat{s}(t)$ using the following:

$$\hat{s}(t) = \sum_{i=1}^{m} a_i s(t-i) \tag{3.2.3}$$

where a_i are the model parameters and m the order of the Linear Predictor (LP).

When defining an MA model, we observe that the model parameters a_i are similar to the weight of smoothing and thus we set $a_i = \frac{1}{m} \ \forall \ i$. The majority of techniques discussed in this chapter follow the LPA modelling form.

3.2.2 Forecasting using MA

To forecast a new value at $t+1$, denoted by $\hat{y}_{t+1}$, we employ the smoothing from expression 3.2.2 and apply it to the last m observations:

$$\hat{y}_{t+1} = \frac{y_t + y_{t-1} + ... + y_{t-(m-1)}}{m} \tag{3.2.4}$$

This type of forecasting is referred to as *one-ahead forecasting*, because the expression only predicts one value into the future at time $t+1$.

To predict values further into the future, say $t+2, t+3, ..., t+k$, we use the predicted value at the previous step ($\hat{y}_{t+k-1}$) as a 'true observation' and re-apply equation 3.2.4.

$$\hat{y}_{t+2} = \frac{\hat{y}_{t+1} + y_t + ... + y_{t+1-(m-1)}}{m} \tag{3.2.5}$$

Applying equations 3.2.4 and 3.2.5 to example data, Figure 3.2 illustrates an MA model (with $m = 5$) predicting values for $t+1$ up to $t+10$.

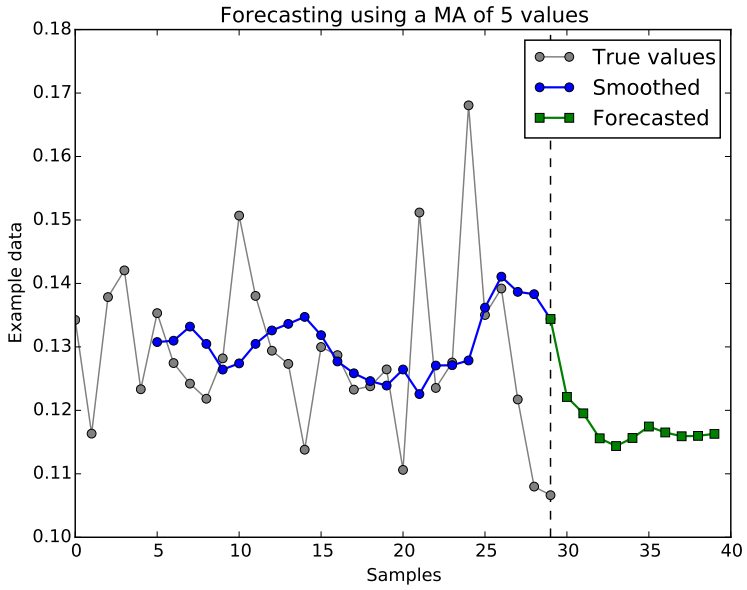


Figure 3.2: Forecasting 10 samples into the future using a Moving Average model with $m = 5$. The predicted values are on the right side of the vertical line.

In terms of cloud provisioning, Lorido-Botrán et al. [42] state that MA has poor long-term prediction results on noisy data and suggest that MA may be better suited for stable workloads.

3.3 Exponential Smoothing

In Section 3.2, we discussed Moving Average and showed (in equation 3.2.2) that past observations are weighted equally with a factor of $\frac{1}{m}$, where m is the number of observations used. Exponential Smoothing in comparison, assigns exponentially decreasing weights to each of the past data points and puts more emphasis on the most recent observations.

In 1956 Robert G. Brown [8] proposed the first Exponential Smoothing approach called 'Brown's Simple Exponential Smoothing'. Brown's method aims to improve short-term forecasting compared to MA by better estimating the level of a time-series.

Brown's method is calculated using the following equation:

$$\begin{aligned} s_t &= \alpha y_{t-1} + (1-\alpha)s_{t-1}, \qquad 0 \leq \alpha \leq 1 \\ s_1 &= y_1, \qquad\qquad\qquad\qquad\quad t \geq 3 \end{aligned} \tag{3.3.1}$$

where s_t is the smoothed value at t, α the level smoothing factor (a value between $[0, 1]$) and y_1 the observation at $t = 1$.

When substituting s_{t-1} into equation 3.3.1, the exponential decreasing weights applied to older values become more visible:

$$\begin{aligned} s_t &= \alpha y_{t-1} + (1-\alpha)[\alpha y_{t-2} + (1-\alpha)s_{t-2}] \\ &= \alpha y_{t-1} + \alpha(1-\alpha)y_{t-2} + (1-\alpha)^{\mathbf{2}} s_{t-2} \end{aligned} \tag{3.3.2}$$

Larger values for α have less of a smoothing effect. The most recent observations are weighted more in comparison with α values closer to zero, which have greater smoothing effect. This is illustrated in Figure 3.3 where smoothing is applied using different values for α.

3.3.1 Exponential Smoothing model parameter estimation

When fitting a Simple Exponential Smoothing model, we minimise the Mean Squared Error (MSE) between the true values and the smoothed values given by:

$$\begin{aligned} MSE &= \frac{1}{n}\sum_{i=1}^{n}(s_i - y_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[\alpha y_{i-1} + (1-\alpha)s_{i-1} - y_i\right]^2 \end{aligned} \tag{3.3.3}$$

where s_i is the smoothed value at $t = i$, also referred to as the estimate, y_i the true value of the series at $t = i$ and n the total number of samples used.

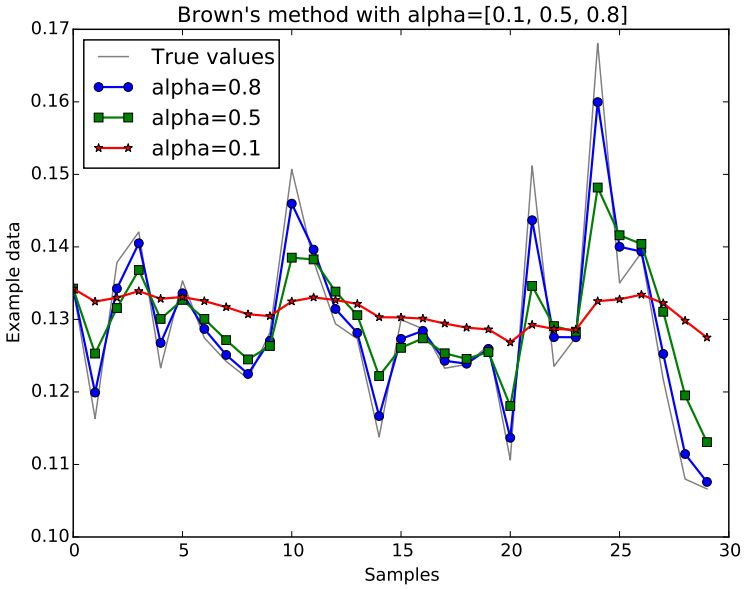


Figure 3.3: Brown's Simple Exponential Smoothing with different α's, illustrating that α values closer to zero have a greater smoothing effect.

In comparison to LPA, no closed-form solution exists for determining the optimum α parameter that will minimise the error. Thus we employ a numerical optimisation algorithm to find the optimum α. In this work we use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, an iterative non-linear optimiser provided by our scientific computing package SciPy [60]. The BFGS algorithm is one of many possible optimisation algorithms that could be used. Others include the Levenberg-Marquardt algorithm. The formulation of the BFGS algorithm is presented in [10].

3.3.2 Forecasting using Brown's Exponential Smoothing

We use equation 3.3.3, the optimum α and the last observation y_t to perform a one-ahead forecast using Brown's Exponential Smoothing:

$$s_{t+1} = \alpha y_t + (1-\alpha)s_t, \quad t > 0 \tag{3.3.4}$$

To forecast values for $t+2, ..., t+k$, we assume that the previous predicted value was a 'true value' and re-apply the forecast equation. Figure 3.4 illustrates the forecasts for $t+1$ up to $t+20$ on example data. We observe that for long-term predictions the values become a straight line. We remember that each forecast is an estimated mean of a future value and thus with no new information the forecasts reach a constant. This also indicates that Brown's Simple Exponential Smoothing is not suited for long-term forecasting on data that may contain a trend or seasonal components.

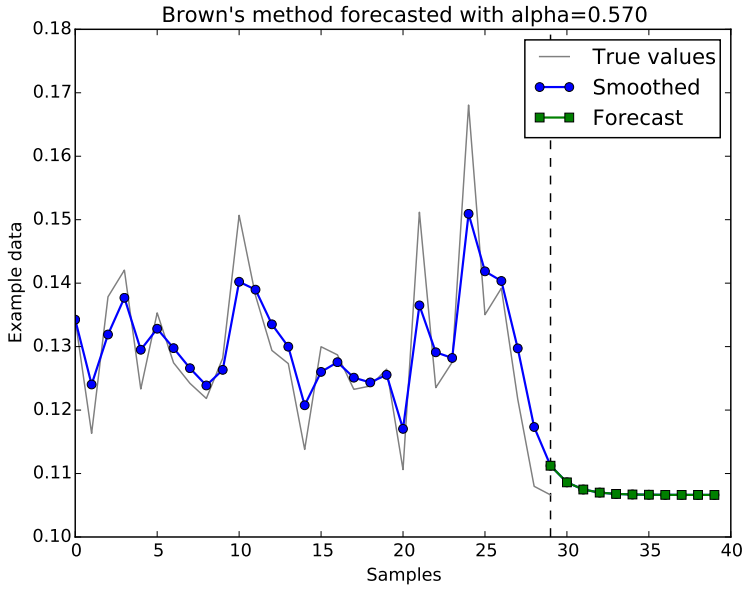


Figure 3.4: Forecast of 10 values using Brown's Simple Exponential Smoothing method and an optimum α=0.570.

In terms of cloud provisioning, Brown's method marginally improves on MA's forecasting on stable workloads. It is able to better estimate the level of a series using both the current and past observations [42]. More complex workloads, like cyclic or bursty workloads, require more complex Exponential Smoothing. We investigate two of these methods (Holt's method and Holt-Winters' method) in the next sections.

3.4 Holt's Linear Exponential Smoothing

In 1957, Charles C. Holt extended Brown's Simple Exponential Smoothing method to forecast on data that contains a trend and proposed Holt's method, (also referred to as Double Exponential Smoothing). This method uses the following two smoothing equations and initial values:

$$\begin{aligned} s_t &= \alpha y_t + (1-\alpha)(s_{t-1} + b_{t-1}), \qquad 0 \leq \alpha \leq 1 \\ b_t &= \beta(s_t - s_{t-1}) + (1-\beta)b_{t-1}, \qquad 0 \leq \beta \leq 1 \\ s_1 &= y_1, \\ b_1 &= y_2 - y_1 \end{aligned} \tag{3.4.1}$$

where α is referred to as the level smoothing factor and β the trend smoothing factor. The initial value for b_1 listed above, can be initialised using a variety of schemes according to [15]. In their online book entitled "Forecasting: Principles and Practice" [30, sec. 7.2], Hyndman and Athanasopoulos describe that the

value s_t denotes an estimate of the level of the series at time t and b_t denotes an estimate of the trend of the series.

Holt's method performs similar to Brown's method, with small values of α having a greater smoothing effect [31]. Figure 3.5 illustrates Holt's smoothing method applied to data, using different α values.

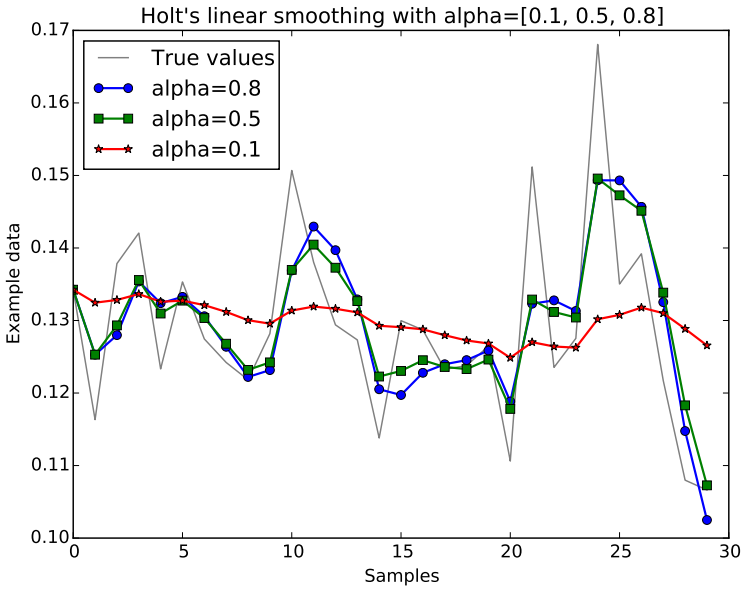


Figure 3.5: Holt's Linear Exponential Smoothing applied to example data using different α's and a constant β. Similar to Brown's method, values of α closer to zero have a greater smoothing effect.

3.4.1 Holt's model parameter estimation

Considering equation 3.4.1, we see that for Holt's method the smoothed value s_t is a combination of the scaled true value y_t and the sum of the previous level and trend estimates ($s_t - 1$ and $b_t - 1$). From the initial value of the trend estimate b_1 we observe that the trend estimate performs a similar function to a first-order differencing of the data.

Similar to Brown's method in Section 3.3, we estimate the parameters α and β by minimising the MSE between the true values and the smoothed values using the BFGS non-linear optimisation algorithm.

The MSE in terms of α and β is given as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}[\alpha y_i + (1-\alpha)(s_{i-1} + b_{i-1}) - y_i]^2 \tag{3.4.2}$$

According to Hyndman [31], Holt's method will have a better in-sample MSE compared to Brown's Exponential Smoothing, because it is optimised over one additional parameter. In terms of cloud provisioning, there is no single preferred optimisation algorithm. We choose to use the BFGS algorithm because of it being available in SciPy.

3.4.2 Forecasting using Holt's Exponential Smoothing

The equations for forecasting one-value ahead and k-values ahead, are obtained by adding the estimates s_t and b_t from equation 3.4.1:

$$\begin{aligned} y_{t+1} &= s_t + b_t \\ y_{t+k} &= s_t + kb_t \end{aligned} \tag{3.4.3}$$

where each value further into the future, is a combination of the level estimate s_t and a 'gradient' k of the trend estimate. These equations have a similar form to that of a straight line ($y = c + mx$) and may suggest Holt's reasoning behind formulating the forecasting method in this way.

Figure 3.6 illustrates forecasting using Holt's Linear Exponential Smoothing on the example data we have been using thus far. We notice that long-term forecasts are no longer a flat line but rather follow the last trend estimation. Holt's Linear Exponential Smoothing is an improvement over the Simple Exponential Smoothing but still struggles to forecast data that contains seasonal components.

3.5 Holt-Winters' Additive Exponential Smoothing

The Holt-Winters' method, also called 'Triple Exponential Smoothing' was suggested in 1960 by Peter R. Winters [68], a student of Holt. This method extends Holt's method and is capable of predicting on time series data that contain trends and/or seasonal components.

The Holt-Winters seasonal method contains three smoothing equations: one to smooth the level of the series, one to smooth the trend and one to smooth the seasonal components. Let the length of a single season be L samples. For example in quarterly data L is set to $L = 4$ and in monthly data $L = 12$.

The smoothing equations are not derived in this thesis but where formulated and presented by Winters in [68]:

1. Level smoothing:

$$s_t = \alpha \frac{y_t}{I_{t-L}} + (1 - \alpha)(s_{t-1} + b_{t-1}) \tag{3.5.1}$$

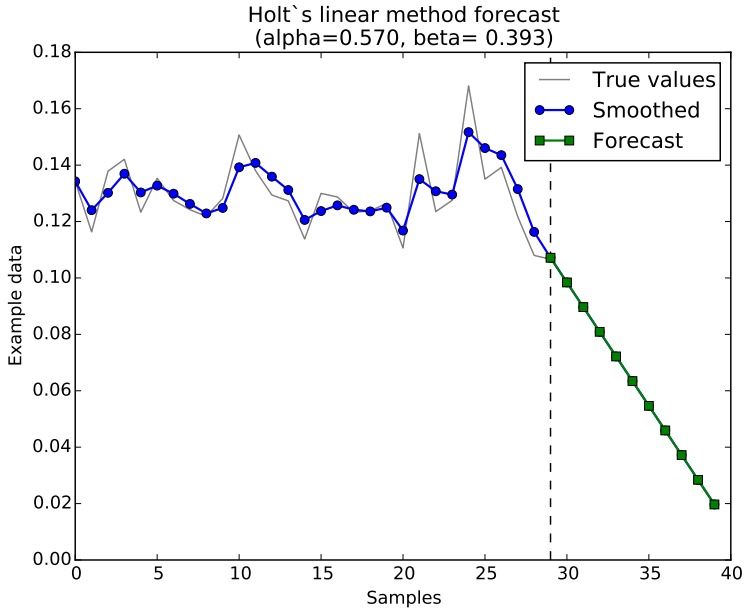


Figure 3.6: Holt's linear method, with optimum α and β used to forecast 10 values. Values forecasted further into the future follow the last trend estimate of the data, which in this case, is the slanted line downward.

where $0 \leq \alpha \leq 1$ is the level smoothing factor. By dividing y_t by the seasonal component I_{t-L}, deseasonalises the data and only the trend factor and the initial value b_1 remain in the updating process of s_t.

2. Trend smoothing:

$$b_t = \beta(s_t - s_{t-1}) + (1 - \beta)b_{t-1} \tag{3.5.2}$$

where $0 \leq \beta \leq 1$ is the trend smoothing factor. The trend estimate is the smoothed diffidence between two successive level estimates of the deseasonalised data.

3. Seasonal smoothing:

$$I_t = \gamma \frac{y_t}{s_t} + (1 - \gamma)I_{t-L} \tag{3.5.3}$$

where $0 \leq \gamma \leq 1$ is the seasonal smoothing factor. The seasonal estimate is a combination of the most recent observation of the data y_t divided by the deseasonalised level estimate s_t and the previous seasonal estimate L samples backward.

3.5.1 Holt-Winters' model parameter estimation

We will now describe the estimation of the smoothing factors and the setting of the initial values used to model a Holt-Winters' model. Again we use a

non-linear optimisation algorithm (e.g. BFGS) to minimise the MSE of the smoothed and the true values, obtaining α, β and γ.

In order to initialise the trend estimate b_t and seasonal estimate component I_t we need at least one complete season's data (i.e. L samples) .According to Croarkin and Tobias [15] "it is advisable to use two complete seasons (ie. $2L$ samples)."

1. The initial value for the level estimate s_1 is set to:

$$s_1 = y_1 \tag{3.5.4}$$

2. The initial value for the trend estimate b_1 is set to:

$$b_1 = \sum_{i=1}^{L} \left[\frac{y_{L+i} - y_i}{L} \right] \tag{3.5.5}$$

3. The initial value for the seasonal components I_i is estimated as:

$$I_i = \frac{1}{N} \sum_{j=1}^{N} \frac{y_{L(j-1)+1}}{A_j} \forall i = 1, 2, ..., L \tag{3.5.6}$$

where A_j is the average of y for the season corresponding to j index and i is the position within the season. The above equation produces N seasonal estimates where N is the number of complete seasons in the data.

3.5.2 Forecasting with Holt-Winters' method

After fitting the Holt-Winters' Additive Exponential Smoothing model, we obtain values for α, γ and β, initialised the estimate equations and forecast one-value and k-values ahead in time using the following formulas:

$$\begin{aligned} y_{t+1} &= (s_t + b_t) I_{t-L+1} \\ y_{t+k} &= (s_t + k b_t) I_{t-L+k} \end{aligned} \tag{3.5.7}$$

where k is the number of values ahead in time to be estimated.

Figure 3.7 shows the Holt-Winters' Additive method applied and forecasted on example data. From the figure we see that Holt-Winters' method is capable of forecasting the seasonal behaviour in the data.

In terms of cloud provisioning, the Holt-Winters' method appears to be the most promising smoothing method as it is able to model and predict three of the four workload types namely: stable, trending and cyclic/seasonal workloads. For the purpose of this thesis, we will only be implementing and evaluating the Holt-Winters' method (denoted as HW).

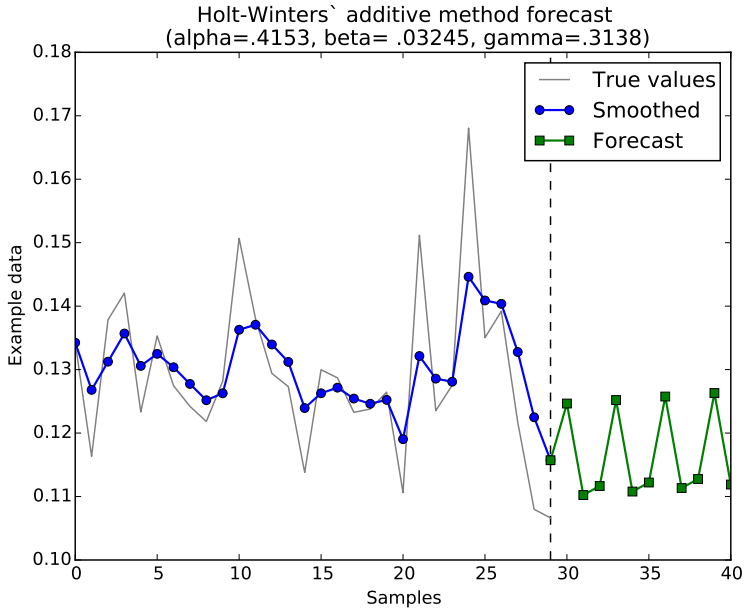


Figure 3.7: Holt-Winters' Additive method, with optimum α, β, γ used to forecast 10 values ahead. Values forecasted follow the last seasonal and trend estimate.

3.6 Auto Regression (AR)

In this section we introduce Auto Regression (AR), but first we discuss simple linear regression, the autocorrelation function and how its coefficients are used to compute the AR model parameters.

3.6.1 Simple Linear Regression

Simple Linear Regression is a statistical method used to determine a polynomial function that closely represents a set of data points and aims to fit a first-order polynomial of the form:

$$\hat{y}_t = \phi_0 + \phi_1 t + \epsilon \tag{3.6.1}$$

where $\hat{y}_t$ is the approximated value, ϕ_0 and ϕ_1 are the value of the intercept and the slope respectively, t the time dependant variable and ϵ the error or deviation [30, ch. 4.1].

Figure 3.8 illustrates a first-order simple linear regression model, fit to example data.

3.6.2 Linear regression parameter estimation

It is important to note, that there exists other linear regression models that considers both historical values and values into the future when estimating

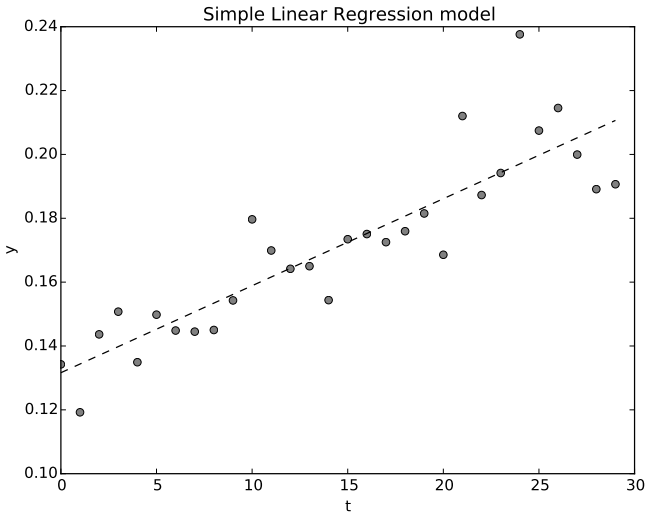


Figure 3.8: A simple linear regression model fit to data that contains a trend. The dashed line is the estimate value at every t. The error at time t, ϵ_t is the difference between the actual value y_t and the value estimated from the model y_t

model parameters. In this discussion we focus on the simple linear regression approach.

Reviewing equation 3.6.1, we observe that the simple linear regression model is a special case of a Linear Predictor (LP). We follow a parameter estimation approach similar to LPA, referenced in Section 3.2.1 by minimising the MSE.

The error of prediction, e_t is defined as the difference between the approximated value $\hat{y}_t$ and the true value y_t:

$$e_t = y_t - \hat{y}_t \tag{3.6.2}$$

Let E be defined as the error function:

$$E = \sum_{i=1}^{N} e_i^2 \quad = \sum_{i=1}^{N} [y_i - (\phi_0 + \phi_1 t_i)]^2 \tag{3.6.3}$$

where N is the number of observations considered and ϵ set to $\epsilon = 0$.

The error function is quadratic in the regression parameters. Therefore is has one unique global minimum. This can be determined by setting the partial derivative with respect to the parameters equal to zero.

$$\frac{\partial E}{\partial \phi_i} = 0, \quad i = 0, 1 \tag{3.6.4}$$

Performing the partial derivative we get:

$$\begin{aligned}\frac{\partial E}{\partial \phi_0} &= -2\sum_{i=1}^{N}\left[y_i - (\phi_0 + \phi_1 t_i)\right] = 0 \\ \frac{\partial E}{\partial \phi_1} &= -2\sum_{i=1}^{N} t_i\left[y_i - (\phi_0 + \phi_1 t_i)\right] = 0\end{aligned} \tag{3.6.5}$$

Two simultaneous equations:

$$\begin{aligned}N\phi_0 + \phi_1\sum_{i=1}^{N} t_i &= \sum_{i=1}^{N} y_i \\ \phi_0\sum_{i=1}^{N} t_i + \phi_1\sum_{i=1}^{N}(t_i)^2 &= \sum_{i=1}^{N} y_i t_i\end{aligned} \tag{3.6.6}$$

Represented in matrix form:

$$\begin{bmatrix} N & \sum_{i=1}^{N} t_i \\ \sum_{i=1}^{N} t_i & \sum_{i=1}^{N}(t_i)^2 \end{bmatrix}\begin{bmatrix}\phi_0 \\ \phi_1\end{bmatrix} = \begin{bmatrix}\sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} y_i t_i\end{bmatrix} \tag{3.6.7}$$

Let $\bar{y}$ and $\bar{t}$ be:

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \quad \text{and} \quad \bar{t} \ = \frac{1}{N}\sum_{i=1}^{N} t_i \tag{3.6.8}$$

When solving it we get estimates for ϕ_0 and ϕ_1:

$$\begin{aligned}\hat{\phi}_1 &= \frac{N\sum_{i=1}^{N} y_i t_i - \sum_{i=1}^{N} y_i \sum_{i=1}^{N} t_i}{n\sum_{i=1}^{N}(t_i)^2 - (\sum_{i=1}^{N} t_i)^2} \\ \hat{\phi}_0 &= \bar{y} - \hat{\phi}_1 \bar{t}\end{aligned} \tag{3.6.9}$$

3.6.3 Forecasting with linear regression

In order to perform a one-ahead forecast, the linear regression parameters are used in the following equation:

$$\hat{y}_{t+1} = \hat{\phi}_0 + \hat{\phi}_1(t+1) \tag{3.6.10}$$

The resulting value of $\hat{y}_{t+1}$ is called the 'fitted-value' or regressed value at time $t+1$.

3.6.4 Autocorrelation Function

When performing analysis on time-series in the time domain, one requires a way to describe the properties of stationarity of the stochastic process. The Autocorrelation Function (ACF) is an important tool in the investigation into

the stationarity of a stochastic time-series [23]. More importantly for the work presented in this thesis, the ACF is used to estimate the AR coefficients as discussed in Section 3.6.5.

Let Y be a process which generates a time-series y and let us define y_t, as time t since the start of the process. At any time t, the series has a mean μ_t and variance σ_t^2 and assuming the series is stationary, both the mean and variance is time independent, μ and σ^2.

The ACF describes the *correlation* of the series at different times separated by k samples. For example, the correlation between the series y_t at time t and a shifted version y_{t-k}. The equation for the ACF for this case is defined as:

$$\rho(k) = R(y_t, y_{t-k}) = \frac{\mathbf{E}\left[(y_t - \mu)(y_{t-k} - \mu)\right]}{\sigma^2} \tag{3.6.11}$$

where $\mathbf{E}[\cdot]$ is the expected value operator, μ and σ^2 the mean and variance respectively.

The ACF has the following properties:

1. $\rho(0) = 1$
2. $\rho(k) = \rho(-k), \quad \forall k = 0, 1, 2, ...$
3. $|\rho(k)| < 1, \quad \forall k = 0, 1, 2, ...$

Figure 3.9 illustrates the ACF plot of example data.

3.6.5 Auto-regression definition

As discussed above, simple linear regression uses a linear combination of parameters (ϕ_0 and ϕ_1) to forecast a value at $t+1$. In this section we will discuss AR which also uses a linear combination of past values. The weights used to combine these past values are estimated using the correlation of the series onto itself, i.e using the ACF.

The general equation for an AR model of order p, AR(p) is defined by:

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon \tag{3.6.12}$$

where ϵ is the modelling error and δ a constant calculated as:

$$\delta = (1 - \sum_{i=1}^{p} \phi_i)\bar{y} \tag{3.6.13}$$

where $\bar{y}$ is the mean of the time-series up to time t.

The order of an AR model p indicates the number of past values to include into the model as well as dictates the number of parameters of the model [30].

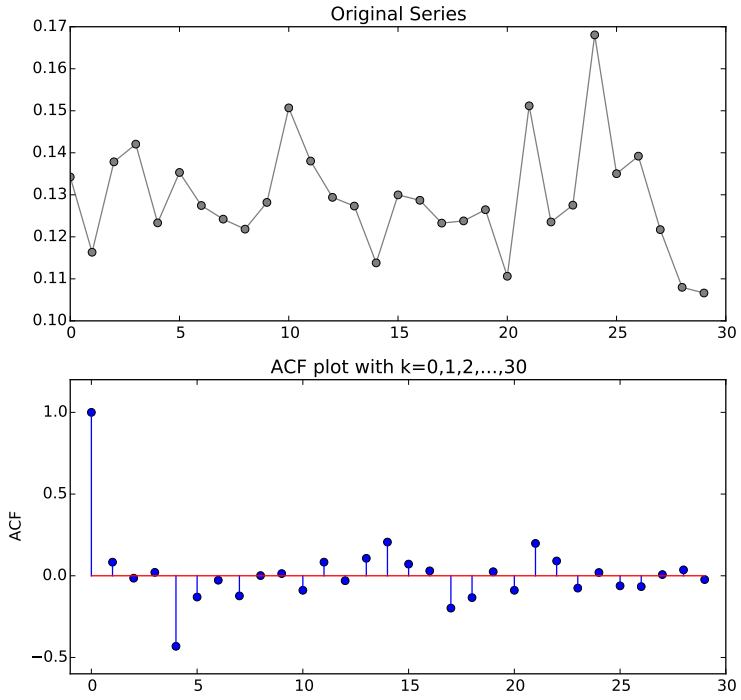


Figure 3.9: An Autocorrelation Function (ACF) plot of example data with $k = 0, 1, 2, ..., 30$. The first value, $\rho(0) = 1$ agrees with the first property of the ACF. From the decreasing values of $\rho(k)$ we can deduce that the time-series does not have a strong autocorrelation.

When fitting higher order AR models, care should be taken because the prediction performance is very sensitive to the model parameters [18]. Possible effects include 'ringing' and instability when fitting.

Various approaches exist for selecting the order for an AR model. These approaches include using Box-Jenkins's method, using digital filter design principles, or using information criteria (such as Akaike's information criterion, Parzen's criterion of autoregressive transfer function or Rissanen's minimum description length [7]).

Box-Jenkins's method uses the ACF discussed above and selects the order based on the number of lags before the ACF decays to zero [15]. Selecting the order using information criteria entails fitting multiple models to the data, each with an increasing order. After each fitting, the specific information criterion value is calculated, thereafter the optimum order is selected as the one that gives the lowest variance in the prediction error [12].

Order selection using digital filter design requires us to describe AR as a linear time-invariant filter with a transfer function that is presented as the

following:

$$H(z) = \frac{1}{1 - \sum_{i=1}^{p} \phi_i z^{-1}} \tag{3.6.14}$$

This transfer function describes an Infinite Impulse Response (IIR) filter. This is shown in Figure 3.10, with the input $e[n]$ zero-mean white noise [7]. A

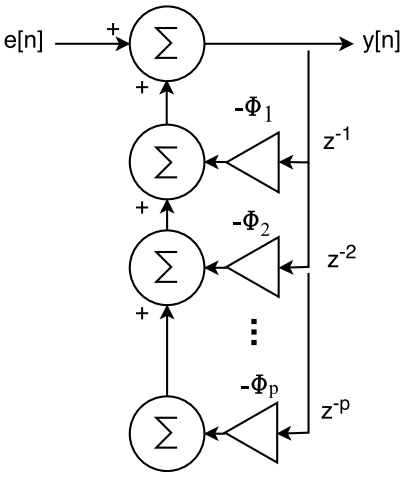


Figure 3.10: Auto-regression model as an Infinite Impulse Response (IIR) filter, where $e[n]$ is zero-mean white noise.

suitable AR order is identified by matching the power spectrum of the data to AR filters of increasing order. Work done by Boardman et. al [7] compared AR models with increasing order. The z-plane and power spectrum plots of these models are shown in Figure 3.11.

To perform order selection of an AR model it is required to estimate the parameters ϕ_i from the data.

3.6.6 Auto-regression parameters estimation

Similar to LPA, the coefficients of the AR model ϕ_i (with $i = 1, 2, ..., p$) are calculated by minimising the MSE and using Yule-Walker equations (first presented by Yule and Walker [63] in 1931).

The derivation of the Yule-Walker equations are presented in Appendix A, and defined as:

$$\rho_m = \sum_{i=1}^{p} \phi_i \rho_{m-i}, \quad m = 0, 1, 2, ..., p \tag{3.6.15}$$

where ρ_m are the values of the ACF at delay m and p the order.

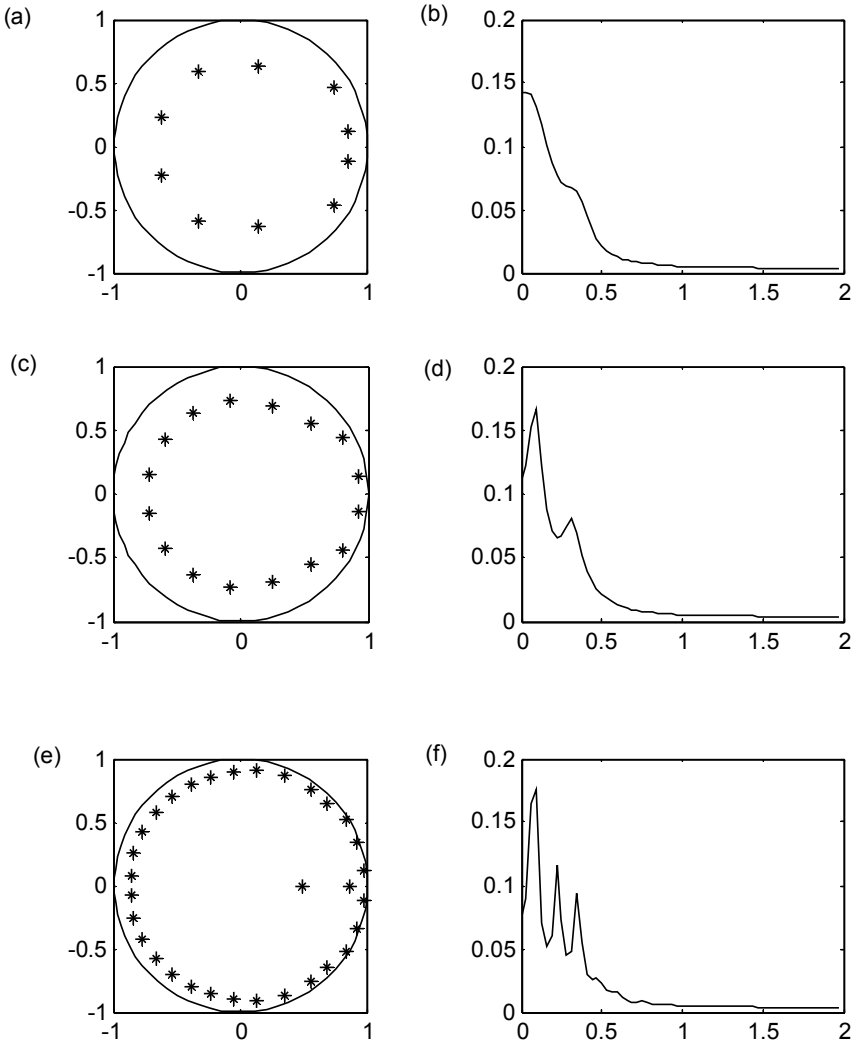


Figure 3.11: Comparison of AR models of increasing order, showing the poles on the z-plane (left) together with the corresponding power spectrum (right). An AR(10) is shown in (a) and (b); an AR(16) in (c) and (d), and in (e) and (f) an AR(32) model is shown. Figure taken from [7].

Expanding equation 3.6.15 explicitly for possible values of m, results in p equations that can be arranged in the following matrix formulation:

$$\mathbf{r}_t = \mathbf{R}_t \cdot \Phi_t$$

$$\begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \vdots \\ \rho_p \end{bmatrix} = \begin{bmatrix} \rho_0 & \rho_{-1} & \rho_{-2} & \cdots & \rho_{1-p} \\ \rho_1 & \rho_0 & \rho_{-1} & \cdots & \rho_{2-p} \\ \rho_2 & \rho_1 & \rho_0 & \cdots & \rho_{3-p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{p-1} & \rho_{p-2} & \rho_{p-3} & \cdots & \rho_0 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_p \end{bmatrix} \tag{3.6.16}$$

with $\rho_0 = 1$ and $\rho_m = \rho_{-m}$ taken from the properties of the ACF given in 3.6.4.

3.6.7 Forecasting with an AR model

After fitting a p-order AR model and calculating the model parameters ϕ_i with $i = 1, 2, ..., p$, we apply equation 3.6.12 to forecast one step-ahead ($\hat{y}_{t+1}$). For two-steps ahead prediction, we re-apply equation 3.6.12 for y_{t+2}, using $\hat{y}_{t+1}$ as a 'true-value'. This approach limits the number of predictions to p-ahead predictions, because the accuracy degrades as we use less of the true values and more estimates contain an estimate error.

Figure 3.12 illustrates an AR(10) model fit to example data and used to forecast 10 values ahead. We observe the ability of the AR model to regress short term trends as well as note the cumulative error of re-using forecasted values as 'true-values'.

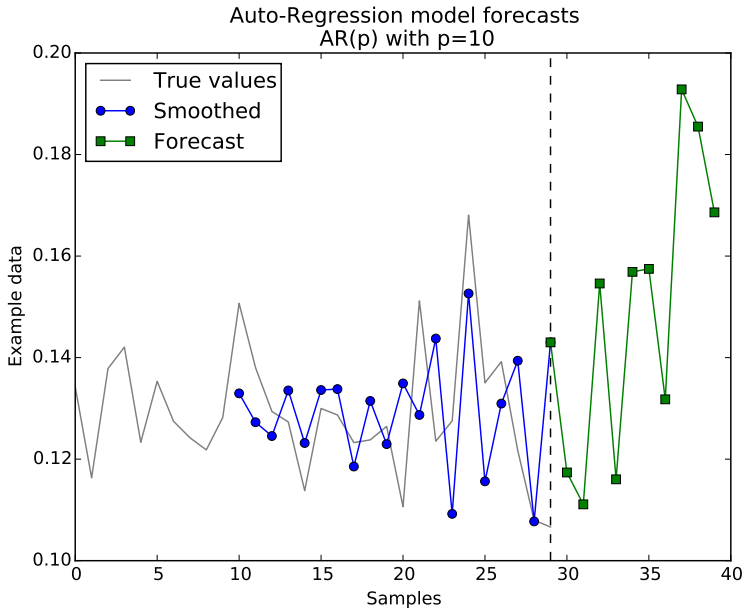


Figure 3.12: Fitting of a 10th order Auto-regression model, order AR($p = 10$) and forecasting 10 values ahead of time. We observe that the AR model regresses the last trend and here we also note the effect of re-using forecasted values as 'true-values', i.e. the forecasts steadily increasing.

In terms of cloud provisioning, AR is able to model cyclic patterns and regress short-to-medium-term trends [38]. More complex AR models can be obtained by combining AR and MA, creating what is referred to as an ARMA: Auto-Regressive Moving Average (or ARIMA when differencing is applied). ARIMA models are able to model stochastic process time-series data using a combination of lower order MA and AR models. These models have several limitations, one being the pre-assumption that there exists a linear form of the time-series model. For long-term forecasts, this is not the case [1]. We will

not investigate ARIMA further, since it is not suited for long-term predictions which are required for the work in this thesis.

3.7 Markov Chains

In this section, we introduce Markov Chains, a machine learning approach to fitting and predicting stochastic sequential data. Dublin et. al [17] defines a Markov chain model as one that has a set of *states* and a set of *transition probabilities* associated with those states. The transition probabilities describe the probability of 'moving' from any given state to any possible next state.

3.7.1 First-order Markov chain model

Following PRESS [24], we use a discrete Markov chain defined by having k discrete states (denoted as $\mathbf{S} = \{x_1, x_2, x_3, ..., x_k\}$) obtained as k levels after digitising data into equally spaced bins. Figure 3.13 illustrates example data digitised into 10 discrete Markov states.

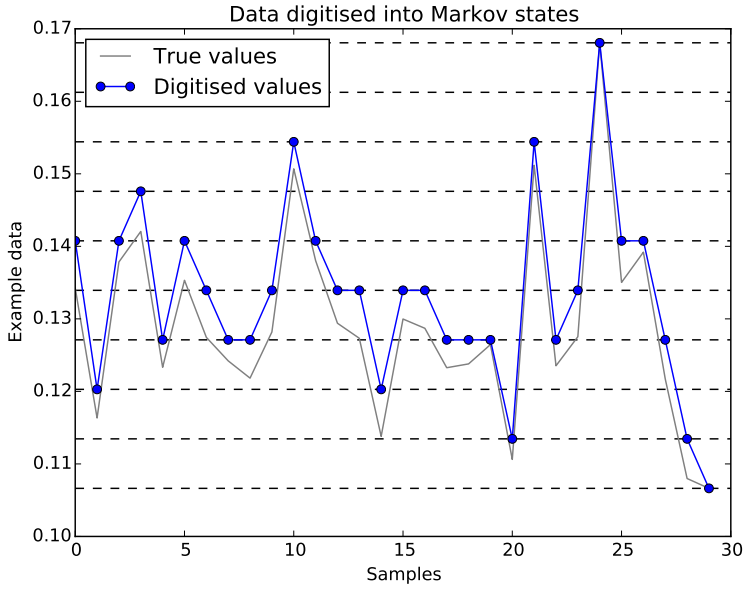


Figure 3.13: Example data digitised into 10 equally spaced Markov states (represented by the dashed lines).

Formally a Markov chain is defined as follows: Let X be a sequence of

random variables $X_1, ..., X_t \in \mathbf{S}$, from the chain rule of probability we get:

$$\begin{aligned} P(X_{t+1}) &= P(X_t, X_{t-1}, ..., X_1) \\ &= P(X_t|X_{t-1}, ..., X_1)\times \\ &\quad P(X_{t-1}|X_{t-2}, ..., X_1) \times ...P(X_1) \end{aligned} \tag{3.7.1}$$

A property of a first-order Markov chain is that each state X_i depends only on the previous state X_{i-1}. Thus equation 3.7.1 can be written as:

$$\begin{aligned} P(X_{t+1}) &= P(X_t|X_{t-1})P(X_{t-1}|X_{t-2})...P(X_2|X_1)P(X_1) \\ &= P(X_1)\prod_{i=2}^{t} P(X_i|X_{i-1}) \end{aligned} \tag{3.7.2}$$

We now define the Markov transition probability denoted as p_{ij}:

$$\begin{aligned} p_{ij} &= P(x_j|x_i) \\ p_{x_1}\prod_{i=2}^{t} p_{ij} &= P(x_1)\prod_{i=2}^{t} P(x_j|x_i), \quad j = i-1 \end{aligned} \tag{3.7.3}$$

where p_{x_1} is the transition probability from the first state.

The one-step ahead transition probabilities form a probability matrix:

$$\mathbf{P} = (p_{ij})_{k\times k} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{kk} \end{bmatrix} \tag{3.7.4}$$

and $\mathbf{P}$ has the property that every row sums to 1.

For a first-order Markov model the probability distribution π over all states for t can be calculated using the discrete form of the Chapman-Kolmogoroff equations [49]:

$$\begin{aligned} \pi_t &= \pi_{t-1}\mathbf{P} = \pi_{t-2}\mathbf{P}^2 \\ &\vdots \\ \pi_t &= \pi_0\mathbf{P}^t \end{aligned} \tag{3.7.5}$$

where π_0 is the initial probability distribution over all the states.

3.7.2 First-order Markov chain parameter estimation

The purpose of this section is to estimate the parameters of the transition matrix $\mathbf{P}$ of a first-order discrete Markov chain. A common approach to estimating the values of the initial probability distribution and transition probabilities, is to count the number of transitions from any state i to any other j ($\forall i, j \ \ \mathbf{S}$) and divide by the total number of transitions [17].

To estimate the initial probability distribution π_0 for each state we count the number of occurrences of that state in the time-sequence and divide by the total number of states. This produces a maximum likelihood estimation of the initial distribution over all states.

Let us look at the series produced after digitising the example data we have been using thus far. Written out in its corresponding states the series is as follows:

$$\mathbf{y} = 5, 2, 5, 6, 3, 5, 4, 3, 3, 4, 7, 5, 4, 4, 2, 4, 4, 3, 3, 3, 1, 7, 3, 4, 9, 5, 5, 3, 1, 0$$

From this we estimate the transition probabilities $p(ij)$ for all states in $\mathbf{S}$ with the initial probability distribution π_0, producing a $\mathbf{P}_{10\times10}$ transition matrix. For example: the probability of transitioning from state 4 to state 3 is calculated as:

$$\begin{aligned} p_{43} &= \frac{P(x_j = 3 | x_i = 4)}{P(x_i = 4)} \\ &= \frac{\frac{2}{29}}{\frac{7}{29}} \\ &= \frac{2}{7} \end{aligned} \tag{3.7.6}$$

The initial probability of state 4, for example, is calculated as:

$$\pi_{0,4} = P(x_i = 4) = \frac{7}{29} \tag{3.7.7}$$

After the fitting of the Markov chain model is done, a new value as forecast into the future can be calculated.

3.7.3 Forecasting using a first-order Markov model

A first-order Markov chain produces a probability distribution for the possible next state when forecasting rather than a single value. A popular approach in forecasting the next state is to select the state with the highest transition probability and output the value associated with this state as a one-ahead forecast [24].

Given that the current state x_c is known at time t, one can construct a probability distribution π_t with 1 at position c, thus $[0, 0, ..., 1, ..., 0, 0]$. Applying equation 3.7.5 using the newly constructed π_t and the transition matrix $\mathbf{P}$ we get the following forecasting formulas:

$$\begin{aligned} \hat{x}_{t+1} &= \max(\pi_t \mathbf{P}) \\ \hat{x}_{t+m} &= \max(\pi_t \mathbf{P}^m) \end{aligned} \tag{3.7.8}$$

where $\hat{x}_{t+1}$ is the state predicted at time $t+1$ and $\hat{x}_{t+m}$ the forecast for m values into the future.

Figure 3.14 illustrates a first-order Markov model with 10 states fitted on example data and used to predict 10 values into the future. As mentioned above, a first-order Markov chain model's next state only depends on the previous state. This affects the accuracy of the long-term predictions as seen in the figure, causing the model to 'state-lock' in one state.

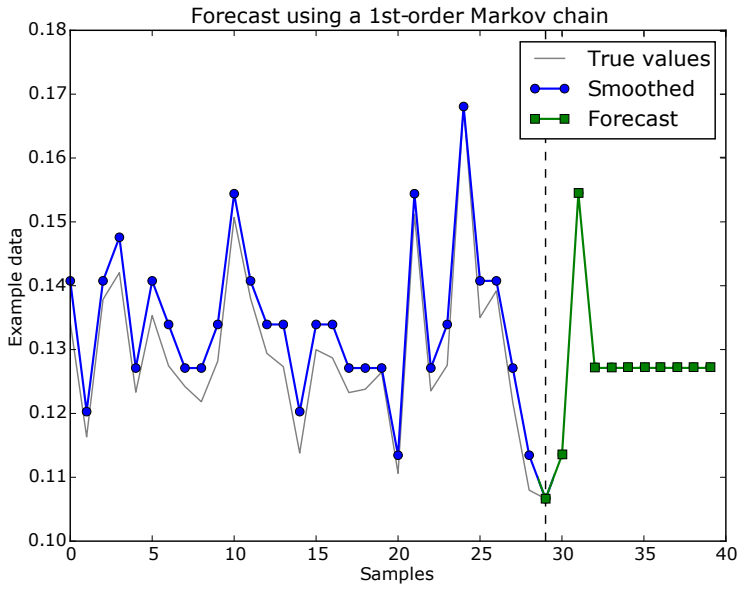


Figure 3.14: Forecasting 10 values using a first-order Markov model with 10-states. It is important to note that after the initial forecasts, the predictions tend to move into a straight line, indicating the limitation of a first-order Markov chain to not be able to perform long-term forecasts.

3.7.4 Second-order Markov chain model

Higher order Markov chains (with a second-order being the simplest) model the transitions of consecutive states rather than single transitions. For the

purpose of this thesis we will only investigate the use of second-order Markov chains for forecasting on cloud resources. We chose to do so because the size and computational cost of fitting $\mathbf{P}$ increases exponentially with the number of transition probabilities to estimate (e.g. from 10^2 to 10^3 for 10 states using the example above).

Estimating the transitions probability matrix $\mathbf{P}$ for a second-order Markov chain from a time series, requires the estimation of a sequence of transitions matrices $\mathbf{P}_i$. Each probability depends on the state of two lags back [40]. Each transition sequence can be written as transitions across three state. For example, the transition probability p_{431} describes the probability of moving from state 4 to 3 and then from state 3 to 1.

In this thesis, we consecutively apply the method for estimating a first-order Markov chain transition matrix as discussed in Section 3.7.2. The forecasting procedure is similar to the first-order model.

3.8 Neural Networks

The machine learning community have developed a statistical learning approach that is inspired by the theory of how cognitive functionality operates in the brain. Artificial Neural Networks (NNs for short) perform computations using a vast number of networked computational components, termed *neurons*. These neurons are stimulated by an input. Through applying the necessary learning, NN can compute complex functions or recognise intricate patterns within data.

NNs are considered as classic machine learning models and can be considered a special case of a *Bayes Network*, which is part of the larger research field of Probabilistic Graphical Models (PGMs).

One of the most basic neurons was first proposed by Frank Rosenblatt in 1957 [55] and is called a *perceptron*, after its biological counterpart. Figure 3.15 illustrates a simple perceptron that takes binary values as input x_i and produces a single binary output y according to the following [48]:

$$y = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{threshold} \end{cases} \tag{3.8.1}$$

where w_i are the *weights* that regulate the importance of each input and are typically real numbered values.

The *threshold* value, also referred to as the *bias* (typically labelled b), specifies where the neuron's output activates. Using vector algebra notation, $\sum_i w_i x_i$ can be written as a dot product of two vectors, $\mathbf{x} \cdot \mathbf{w}$, with equation 3.8.1 written as:

$$y = \begin{cases} 0 & \text{if } \mathbf{x} \cdot \mathbf{w} + b \leq 0 \\ 1 & \text{if } \mathbf{x} \cdot \mathbf{w} + b > 0 \end{cases} \tag{3.8.2}$$

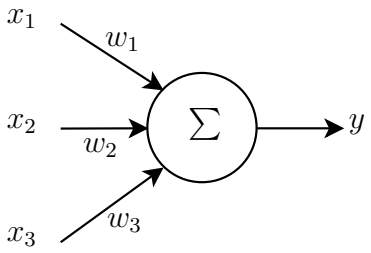


Figure 3.15: An example of a *perceptron* neuron with three inputs, weighted connections and a binary value output.

Formally, for perceptron: $\mathbf{x} \cdot \mathbf{w} + b > 0$ is called the *activation condition* and these conditions constrain a unit step function, as illustrated in Figure 3.16.

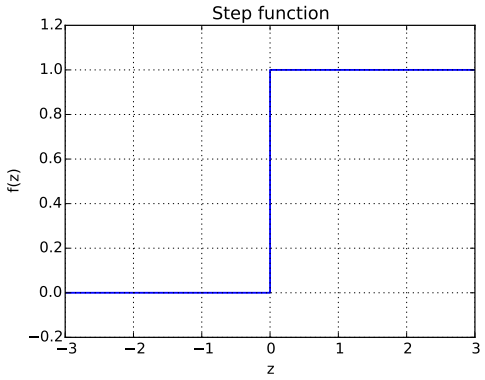


Figure 3.16: The unit step function used as activation function for a perceptron neuron.

This step function is what is referred to as the *activation function*, denoted as $\sigma(z)$, where:

$$\begin{aligned} z &= \mathbf{x} \cdot \mathbf{w} + b \\ y &= \sigma(\mathbf{x} \cdot \mathbf{w} + b) \end{aligned} \tag{3.8.3}$$

The step function is one of various activation functions used in NNs. In Section 3.8.2 we discuss the sigmoid function — a popular activation function used in neural networks.

3.8.1 Neural Network structure

NNs consist of *layers* of neurons with each layer typically fully connected to the next through weighted connections. Figure 3.17 illustrates an example network. The leftmost layer is called the input layer, the middle layers are termed *hidden* layers (because the state of these neurons are 'hidden') and the rightmost layer is called the output layer. A neural network may have multiple

hidden layers or none at all.

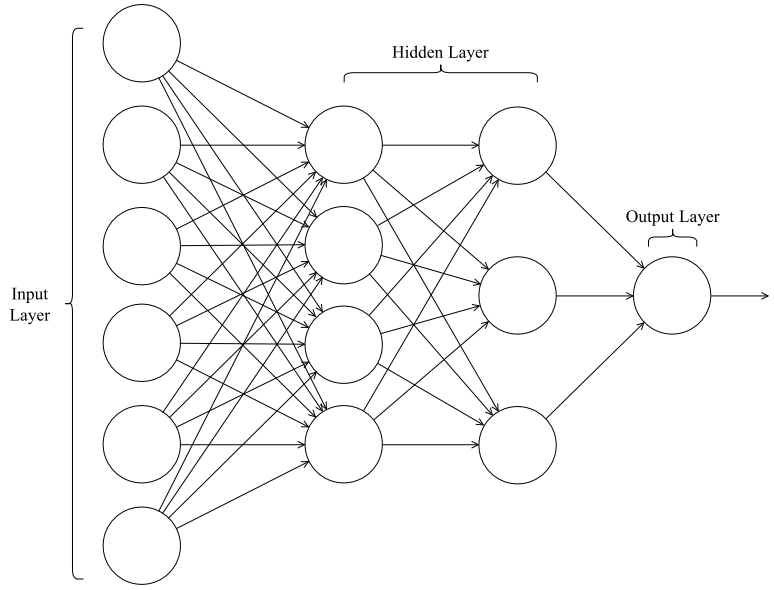


Figure 3.17: Illustration of a feed-forward neural network consisting of four layers — one input, one output and two hidden layers. (This is also an example of a Multi-Layer Perceptron (MLP).)

The NN illustrated in Figure 3.17 is also an example of a Feed-forward Neural Network (FFNN) characterised by having distinct and independent layers (input, output and hidden layers). Each neuron in a layer only has directed connections to neurons in the next layer (closer to the output layer) [37]. FFNNs are good at learning functions that map inputs to a desired output. We will discuss the specifics of what 'learning' refers to in Section 3.8.3.

3.8.2 Sigmoid Neuron

We have already described the step function as an activation function used in perceptrons. We will now describe another popular activation function used in neural networks, namely the *Sigmoid* or *logistic* function. This activation function is given by:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.8.4}$$

By varying z, we get a plot of the sigmoid function that can been seen in Figure 3.18. For large positive values of z the sigmoid function outputs a value close to one and for large negative values, a value close to zero. This is similar behaviour to a perceptron's step function as discussed above.

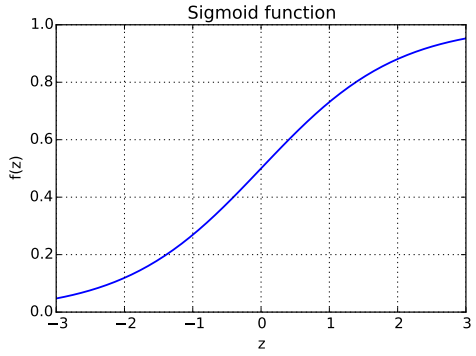


Figure 3.18: The sigmoid function. The output of a sigmoid neuron is a real number between zero and one.

When comparing the step function and the sigmoid function, we note that the sigmoid function has smooth transitions (i.e. no discontinuities) across all possible input values of z. This property is crucial for learning a neural network.

3.8.3 Learning Neural Networks

A neuron, the simplest component of a neural network, takes input values (denoted as $\mathbf{x}_i$) and multiplies each with an associated weight (w_i). Using the activation function $\sigma(z)$ the neuron maps the weighted sum of the input values to an output value y_i. By combining multiple neurons into a network, each computing a single value, NNs are able to learn complex functions that map input values to output values.

Assuming we have a FFNN that has an input layer, an output layer and one hidden layer with all weights and biases initialised, we define a *training set* $\mathbf{T}$ as one that contains *training examples*, i.e. inputs-to-target pairs $(\mathbf{x}_j, d_j)$ with each input vector $\mathbf{x}_j$ having a corresponding desired output value d_j.

According to Rumelhart, Hinton and Williams [57], the aim of *learning* is to find weights and biases by making small changes to each neuron in the network to ensure that for each input vector $\mathbf{x}_j$ the output y_j is the equal or close to the desired value d_j. Figure 3.19 illustrates this schematically. (Note that it is common practice to model the bias term b as a weight and thus it is omitted in Figure 3.19).

The sigmoid function's smooth transitions makes it a prime activation function for learning. Because of its smoothness any small change Δw_j in the weights (and Δb in the biases) will cause a small change $\Delta \hat{y}$ in the output.

This can be approximated to:

$$\Delta y \approx \sum_i \frac{\partial y}{\partial w_i} \Delta w_i + \frac{\partial y}{\partial b} \Delta b \tag{3.8.5}$$

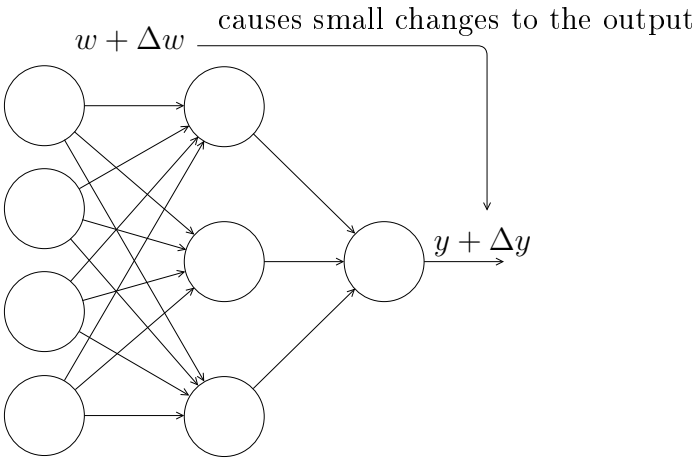


Figure 3.19: A simple illustration of how a Neural Network learns. Small changes to the weights and biases causes small changes to the output. The objective of learning is to bring the output of the Neural Network closer to the desired output for each training example.

where the sum is taken over all neurons i and thus all weights w_i. In short, Δy is a linear function of the changes Δw and Δb in the weights and biases.

To move forward, we define an *error function* (also referred to as an *objective function*) that will aid us in finding weights and biases so that the output of the network y approximates the desired output d_j for each training example $\mathbf{x}_j$ in $\mathbf{T}$. The *Quadratic cost function* is a popular error function to use when learning NNs and it is defined as follows:

$$E = \frac{1}{2}\sum_j (y_j - d_j)^2 \tag{3.8.6}$$

where E denotes the error function, equal to the sum of the squared differences calculated over all training examples in $\mathbf{T}$.

The objective is to minimise E and bring the network output y_j closer to the desired output d_j. Formally, the quadratic cost function is a smooth function with only one global minima. The objective is to find a set of weights that will minimise it to this global minima. *Gradient decent* is the approach used to minimise this error function and it is given by:

$$\Delta w = -\eta \nabla E \tag{3.8.7}$$

where η is a positive value known as the *learning rate* which dictates the speed of learning. Smaller values of η cause the NN to learn slowly and larger values more quickly. Larger learning rates also increase the chance of the network missing (or overshooting) the desired value.

3.8.3.1 Backward-propagation

Backward-propagation is a major improvement in learning NNs that was first proposed in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams [57] in their paper entitled: "Learning representations by back-propagating errors." Instead of taking partial derivatives of the output with respect to the weights (as done in equation 3.8.5). Rumelhart et al. suggested computing the partial derivatives of the Error-function $\frac{\partial E}{\partial w}$ with respect to the weights. For each weight w_i, we compute:

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_j \frac{\partial y_j}{\partial w_i} \frac{dE}{dy_j} \\ &= \sum_j x_i (y_j - d_j) \end{aligned} \tag{3.8.8}$$

where j is a training example in $\mathbf{T}$. Using this equation Rumelhart et al. defined the *Delta rule*, used to calculate the change in each weight δw_i as follows:

$$\begin{aligned} \Delta w_i &= \eta \frac{\partial E}{\partial w_i} \\ &= \eta \sum_j x_i (y_j - d_j) \end{aligned} \tag{3.8.9}$$

where η again is the learning rate.

The Backward-propagation algorithm steps for a single training pair can be summarised as follows:

- **Input:** Each neuron in the input layer calculates its output using the inputs x_i and the sigmoid activation function.
- **Feed-forward:** All hidden layers and the output layer compute their associated $z = \mathbf{x} \cdot \mathbf{w} + b$ and activations $\sigma(z)$.
- **Output error:** Compute the output error $E = \frac{1}{2} \sum_j (y_j - d_j)^2$ for each j output neuron in the output layer.
- **Back-propagate the error:** Starting from the output layer working back, compute $\frac{\partial E}{\partial y_j} = (y_j - d_j)$.
- **Update weights:** Calculate the weight updates Δw_i using equation 3.8.9.

In summary, the Backward-propagation algorithm is a fast learning algorithm for learning neural networks and gives a detailed insight into how changing the weights changes the overall network [48].

3.8.4 Forecasting using Neural Networks

The inputs of NNs are typically values that relate to features of the data being modelled and predicted. When forecasting on time-series data, observations within a pre-defined look-back window (with a length of m observations) are input into the NN. A functional relationship between these input values and desired output values are learnt through training. This forecasting approach is similar to the approach presented in [51].

Figure 3.20 illustrates how forecasting is applied using a FFNN on example data.

Each time-series observation y_t (with $0 \leq t \leq m$) is fed into corresponding neurons in the input-layer, each calculating their activation and feeding this information forward throughout the network. The result produced by the output layer (typically consisting of only one neuron) is the one-step ahead forecast of the NN.

Similar to how multiple values are forecasted, the one-step ahead forecast can be used as a 'true value' when moving the look-back window one step in time.

3.8.5 Recurrent neural networks

In the previous section, we discussed a Feed-Forward Neural Network (FFNN), where information is input and processed serially throughout the network. FFNNs are limited and struggle to perform forecasts on time-series data that contain ordering or state-dependence [20]. As discussed in Section 1.2.4, cloud computing is presented with different types of workloads. Trending and seasonal/cyclic workloads can be viewed as state-dependent processes and thus we investigate a type of NN able to model these.

The purpose of this section is to briefly introduce *Recurrent Neural Networks*, denoted RNNs. Kriesel [37] defines recurrence in neural networks as "the process of a neuron influencing itself through any connection" and states that NNs typically do not have specified input and output layers. Depending on the type of RNN, a network may have the ability to compute complex sequential patterns or as stated by Elman [20] "be able to remember state through recurrent connections."

Figure 3.21 illustrates an example of a RNN with connections feeding information back into the network. At every time step a RNN can be viewed as a FFNN. Thus all the properties of FFNNs holds including the Backward-propagation algorithm.

3.8.6 Elman recurrent neural networks

Following Nae, Iosup and Prodan [45], we investigate the use of a type of RNN referred to as an Elman network for forecasting on cloud resources. Jeffrey

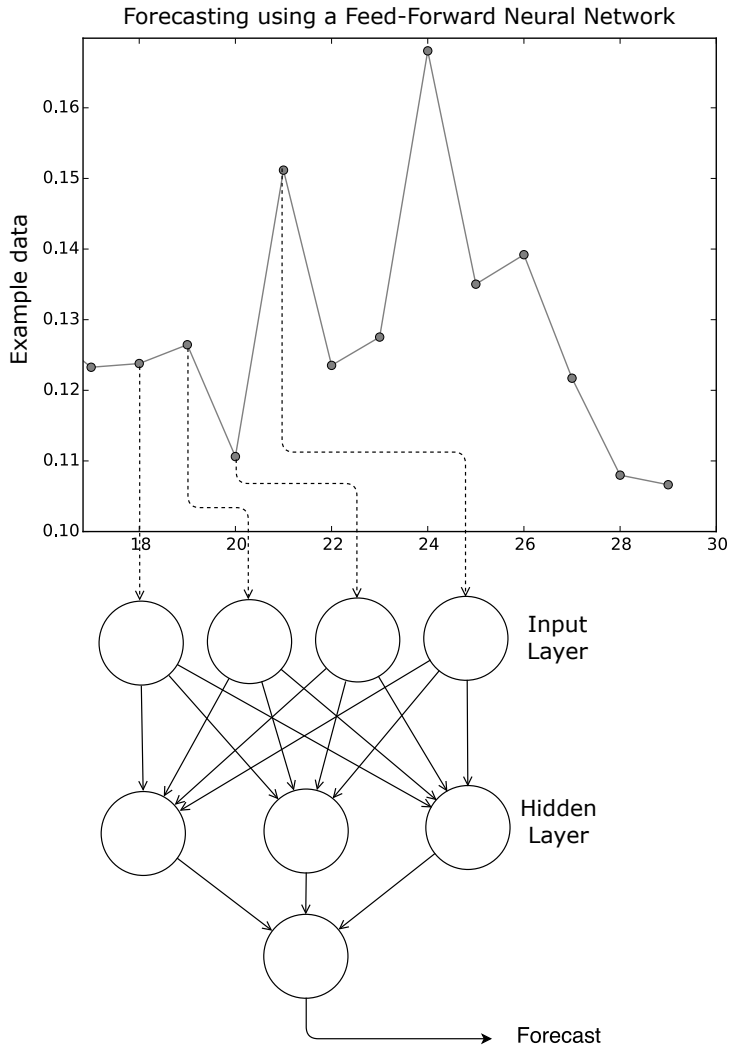


Figure 3.20: An example of how a Feed-Forward Neural Network is used to predict on a time-series. Each observation in the training window can be viewed as a 'feature' of the time-series. The network aims to find a functional mapping of these past values to the future value being predicted.

L. Elman first presented his work in 1990, in a paper entitled "Finding structure in Time" [20]. He based his work on a Jordan network [35], keeping the feed-forward structure of FFNNs but augmenting the network with an additional recurrent layer.

This layer contains neurons termed *context neurons* and is used to remember the 'context' of the task being computed by storing the hidden state of each hidden neuron at every time step. The context neurons are connected with trainable weights to the hidden layer and can be trained using backward-propagation as described previously.

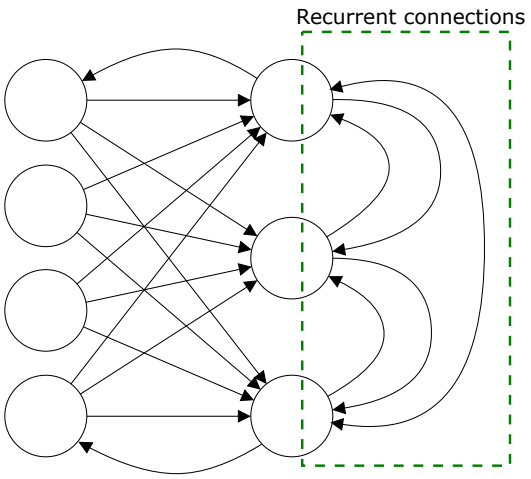


Figure 3.21: An example of a Recurrent Neural Network with recurrent connections in the rightmost layer and connections 'feeding' information back to the layer on the left, which could also be viewed as the 'input' layer.

Elman is able to show that this type of neural network structure performs better than traditional feed-forward neural networks, especially predicting on sequential data and data that is time delayed.

Figure 3.22 illustrates a typical Elman-RNN, its context neurons and the trainable recurrent connections from the hidden layer output to the input of that layer.

3.9 Summary

This chapter described the background and theory required for the development and implementation of the forecasting methods used in this thesis. The chapter started with simpler models, such as Moving Average (MA) which uses a linear combination of equally weighted past observations to predict a smoothed value as an estimate.

Next, the development of three Exponential Smoothing approaches were discussed and the formulation of the equations used in Holt-Winters (HW) method were covered. The HW method employs three smoothing equations to forecast on data that contain trends and seasonal components. The Autoregression (AR) model and the estimation of its model parameters using the Autocorrelation Function (ACF) were discussed and similarities to Linear Prediction Analysis highlighted.

A machine learning method, the Markov Chain model, was discussed in the context of time-series prediction. The estimation of the first- and second-order Markov chains' transition matrices and processes used to perform forecasts

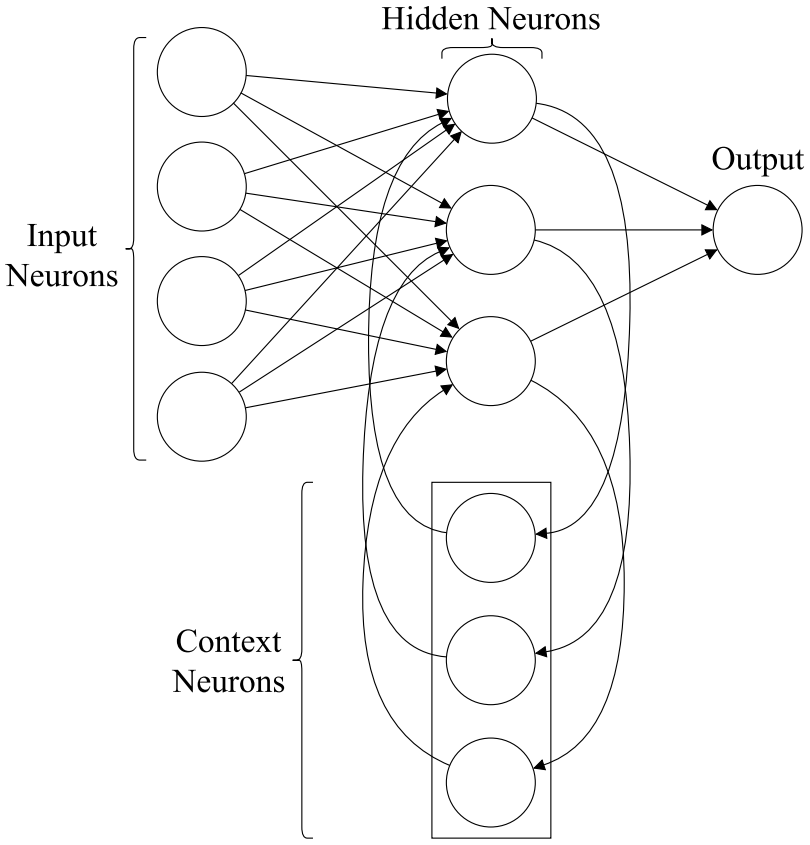


Figure 3.22: An example of an Elman-Recurrent Neural Network (RNN) with context neurons capturing and 'remembering' the state of the output. These context neurons allow the network to remember the last state. Elman RNNs are able to predict on state-dependant workloads.

were described. It was decided to not investigate higher order Markov chain models because of the size of the transition matrix increasing exponentially.

Finally, the necessary theory and intuition for understanding Neural Networks (NNs) were discussed. This includes the basics on neurons, weights, the sigmoid activation function and the Backward-propagation algorithm used for learning NNs. Two types of Neural Networks (NNs) are important for the work in this thesis. They are Feed-Forward Neural Networks (FFNN) and Elman-Recurrent Neural Networks (RNNs).

The next chapter describes the details of implementing these forecasts methods in Python. Discusses the reasoning behind design decisions, outlines a research pipeline and highlights issues encountered and describes how these were handled.

Chapter 4

Implementation of Forecasting Methods

In previous chapters, we identified prominent forecasting methods from literature (Chapter 2) and covered the theoretical development and formulation required to forecast on time series data (Chapter 3).

In this chapter we describe the implementation of these methods in Python 2.7, outline the Resource Forecasting Pipeline we built and highlight the implementation issues encountered. We used the literature as reference as far as possible and followed the descriptions of PRESS and Agile as presented in their respective papers. The design and implementation decisions made with implementing each forecasting method are discussed where relevant.

4.1 Holt-Winters Implementation

In 2013 André Queiroz [52] implemented all three types of Exponential Smoothing algorithms in Python 2.7, namely linear, additive and multiplicative. We based our Holt-Winters implementation on his work and added functionality for refitting a model. Improvements were made to the speed of re-fitting by choosing the previous time step's parameters as starting condition for the BFGS optimisation.

Issues: The Holt-Winters method requires a parameter L, the length of a complete season, in order to perform seasonal estimations. But this parameter is unknown in the resource usage data used. It was decided to use the dominant period (inverse of the dominant frequency) as the length of the season, assuming that this will have the largest contribution to the seasonal estimate. The dominant frequency was extracted by taking the FFT of the total trace using a function provided by the Numpy package. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm from SciPy was used as numerical solver.

4.2 Auto-regression Implementation

According to Kupferman et al. [38], the order of an Auto-regression model, denoted by AR(p), determines how far the model extends into the future and how accurate the model can perform long-term predictions.

Three order selection techniques were discussed in Section 3.6.5 and it was decided to investigate the digital filter design approach. Calculating the power spectrum density (PSD) of our data reveals that the majority of data traces exhibit a spectrum illustrated in Figure 4.1.

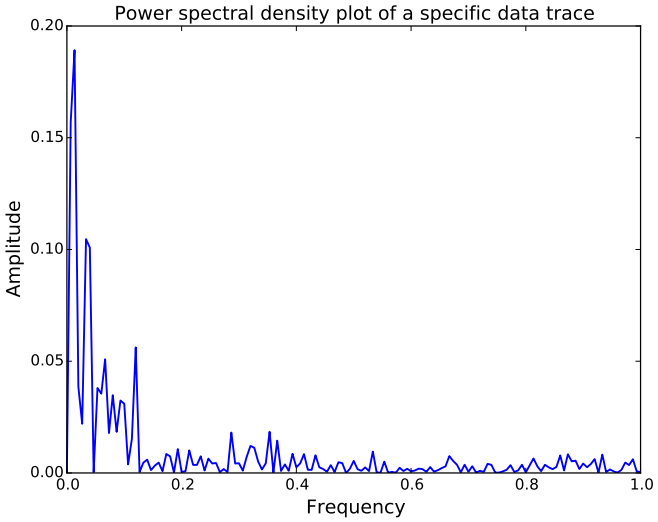
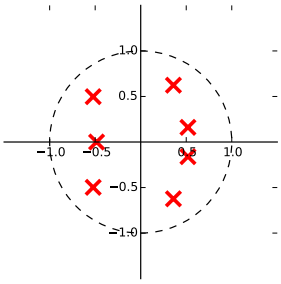


Figure 4.1: The power density spectrum of a CPU usage data trace.

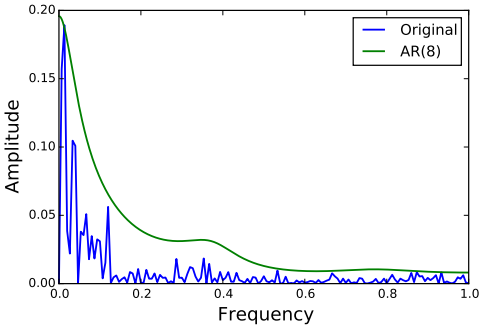
Issues: Performing order selection per data trace would be costly and time consuming. It was decided to rather fit three different AR models, each with a higher order to all data used in evaluation.

Figure 4.2 shows the z-plane pole plots and PSD plots of the three AR models used in this thesis. These AR models are of order 8, 16 and 30. We observe that the highest order model, AR(30)'s envelope matches closely with the data's PSD.

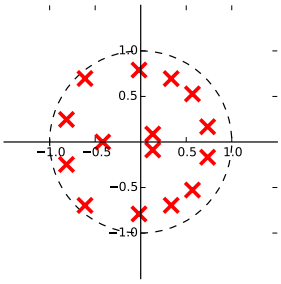
Statsmodels [50], a Python package that provides statistical functionality for working with different data types (including time-series data) was used when fitting our AR model. The auto-correlation function method from Statsmodels produced the ACF coefficients. Using a linear solver from Numpy we calculated the model's auto-correlation coefficients needed for fitting a model.



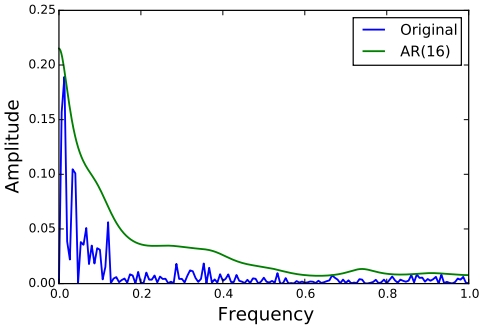
(a) Z-plane of AR(8).



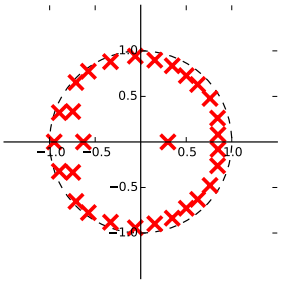
(b) PSD of AR(8).



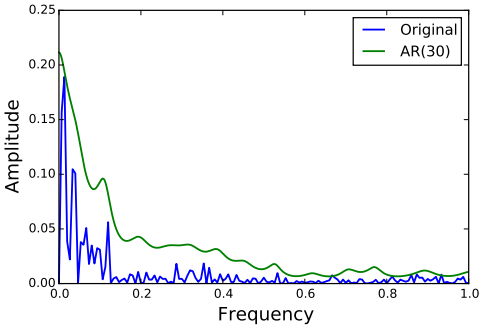
(c) Z-plane of AR(16).



(d) PSD of AR(16).



(e) Z-plane of AR(30).



(f) PSD of AR(30).

Figure 4.2: Comparison of the power spectrum density (PSD) of our data and Auto-regressive models of increasing order. Using this we can perform order selection based on which AR model's PSD fit best to the data's PSD.

4.3 Markov Chain Implementation

We followed PRESS's approach for our Markov chain models using 40 states and digitising the data into equal-sized bins. We implemented a first-order and a second-order Markov model. Because of the dimension of the transition matrix $\mathbf{P}$ increases exponentially with the order p, it was decided not to con-

sider higher order chains than second order. For constructing the transition matrix **P** we used the frequency of state transitions (also referred to as the Frequentive approach), because it is unclear which method PRESS used. Another approach for constructing a transition matrix is using a Bayesian method, also referred to as *Laplace estimates* [14]), but it was found that the Frequentive approach better captures the state-transitions of our data.

Issues: It was found that data with small variations causes the Markov Chain to 'state lock'. We tried different methods for initialising the transition matrix **P**, but this issue was still exhibited in the minority of data traces used.

4.4 PRESS Implementation

As described in their paper, PRESS employs dynamic time warping in their *signature driven* model when the average-pattern does not match the window. The signal processing tool 'warps' one signal onto another so that their lengths agree. For our implementation we used **DTW 1.0**, a python module created by Pierre Rouanet [56] that performs dynamic time warping using the absolute distance between signals.

Issues: We had difficulty identifying every detail of PRESS's design, but implemented our PRESS model as closely to what is described in the author of PRESS's paper. The following issues were found:

1. As mentioned in Section 4.3, our Markov chain model uses the frequency of state transitions to construct the transition matrix **P**, because it is unclear which method PRESS uses.

2. When evaluating our method, we used Google's 29 day dataset compared to the 7 hour dataset used by PRESS, which has more noise and longer trace lengths. Thus we used a mean-ratio constraint of 0.1 instead of 0.05 for similar signature patterns.

3. We performed medium-to-long term forecasts compared to one-ahead forecasts performed by PRESS.

4. The specific order of the AR model used by PRESS as a comparative forecasting method is unknown.

4.5 Agile Implementation

Using Agile's paper as reference, we implemented our own version of Agile as close to their description as possible. We also set our wavelet transform scales to 5 and corresponding forecast window lengths to [1, 2, 4, 8, 16].

Issues: The specific wavelet functions employed in Agile's wavelet transforms are unknown. We therefore decided to use the wavelet functions provided

by the PyWavelet [64] python library. Also, identical to PRESS, the details of the Auto-regression model used to compare Agile's performance is unclear.

4.6 Neural Network Implementation

A Neural Network's accuracy depends greatly on the choice of *hyper-parameters* when training on data. Grid search is a common approach used to find suitable values for these parameters. Bergstra and Bengio [6] suggests rather using *random search* to find 'good' parameters, as it finds good hyper-parameters faster and does not require a grid to be set up beforehand. We follow their suggestion and use the first 1000 samples of each machine to search for hyper-parameters. The best Root-Mean-Squared-Error (RMSE) on a validation set (10% of the 1000 samples) is used to confirm that these hyper-parameters are a good set. This approach finds a 'good' neural network for each machine.

These networks are then trained on the rest of the Training window (2000 samples from the corresponding machine's trace). After each forecasting step the network is updated with the new data.

The python library, PyBrain [58], was used to construct, train and activate the FFNNs and RNNs in this thesis. For constructing FFNNs, the library's `buildNetwork()` function was used. The Elman RNNs where constructed using the NN components and the weighted connections provided by the library.

Issues: It was found that our RNNs presented transient behaviours when refitting and forecasting. It was investigated but the root cause could not be determined and thus it was decided to use the last forecasted value from the previous prediction step as the first forecasted value of the next step. Figure 4.3 illustrates an example plot showing the transient effects and the work-around.

4.7 Resource Forecasting Pipeline

On the basis of the statement made by Lorido-Botrán et al. [42], the motivation of this thesis is to perform a formal investigation in comparing prominent forecasting methods for provisioning and auto-scaling of cloud resources, a Resource Forecasting Pipeline (RFP) was developed and is discussed here.

The motivations for developing the RFP are:

1. Standardise the input of data from pre-processed datasets.
2. Provide an environment for performing repeatable experiments and evaluations.
3. Facilitate the development of forecasting methods through standardised class prototypes.
4. Provide a way to develop and tweak performance and evaluation metrics.

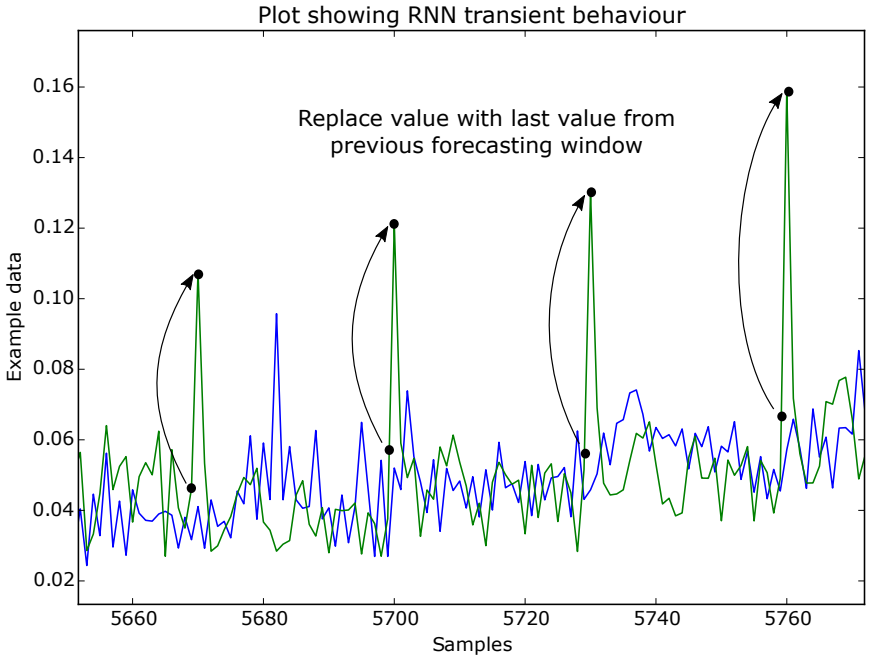


Figure 4.3: Inspecting the RNN transient behaviour when forecasting. The root cause for this behaviour could not be determined, but it was decided to use a work-around: replace the anomaly value with the last value of the previous forecasting window.

The pipeline contains three distinct stages with interfaces between these to facilitate modular design. The three stages are discussed here and outlined in Figure 4.4 on page 58:

- **Data Input:** This stage is used to pre-process cloud resource data which is then processed by the next stages. In this thesis, two evaluation datasets were used. The Data Input stage processed the data into individual files for each data source and data type (e.g. CPU or Memory). Each file then contains the resource usage of a single source spanning the entire length of the trace.
- **Modelling and Forecast:** This stage contains the implementations of the forecasting methods investigated. Separate python classes were used, with each class following the signature as illustrated in Figure 4.4. The signature has the following methods:
  - An `init()` constructor, which takes model specific configurations as arguments.
  - A `fit(y)` method that takes the initial training data as input and estimates the model parameters using the relevant method de-

scribed in Chapter 3.

  - An `update(y)` method used to refit the model on new data and update the model parameters accordingly.
  - A `predict(n)` method that produces n forecasted values using the fitted model and relevant forecasting equation.

- **Evaluation:** Finally, this stage calculates the evaluation metrics, aggregates the results and produces *whisker-box-plots* for each metric and each experiment performed.

4.8 Summary

This chapter covers the implementation details of the forecasting methods investigated in this thesis, the issues encountered and how these were handled.

Prior work done by André Queiroz [52] was used as basis for our Holt-Winters implementation. The dominant period of the data was used as substitute of the L parameter required in modelling seasonal data. This was done under the assumption that this period will have the largest contribution to the seasonality.

PRESS's description on implementing a Markov chain model was followed and it was decided to use a Frequentive approach to estimating the transition matrix in our implementation. Both PRESS and Agile were implemented as closely as possible, but there still exists specifics that are unknown. PyBrain [58], a Python library for building and using Neural Networks (NN), was used for the FFNNs used in this thesis. Functionality for building custom NNs, provided by the library, was used to construct the Elman-RNNs.

Finally, the development of a Resource Forecasting Pipeline (RFP) was discussed. The RFP facilitates data pre-processing, forecasting method modelling and ensures repeatable experiments with custom defined evaluation metrics. An overview of the pipeline was shown in Figure 4.4.

The formal investigation and comparison of the forecasting methods are performed in the next chapter.

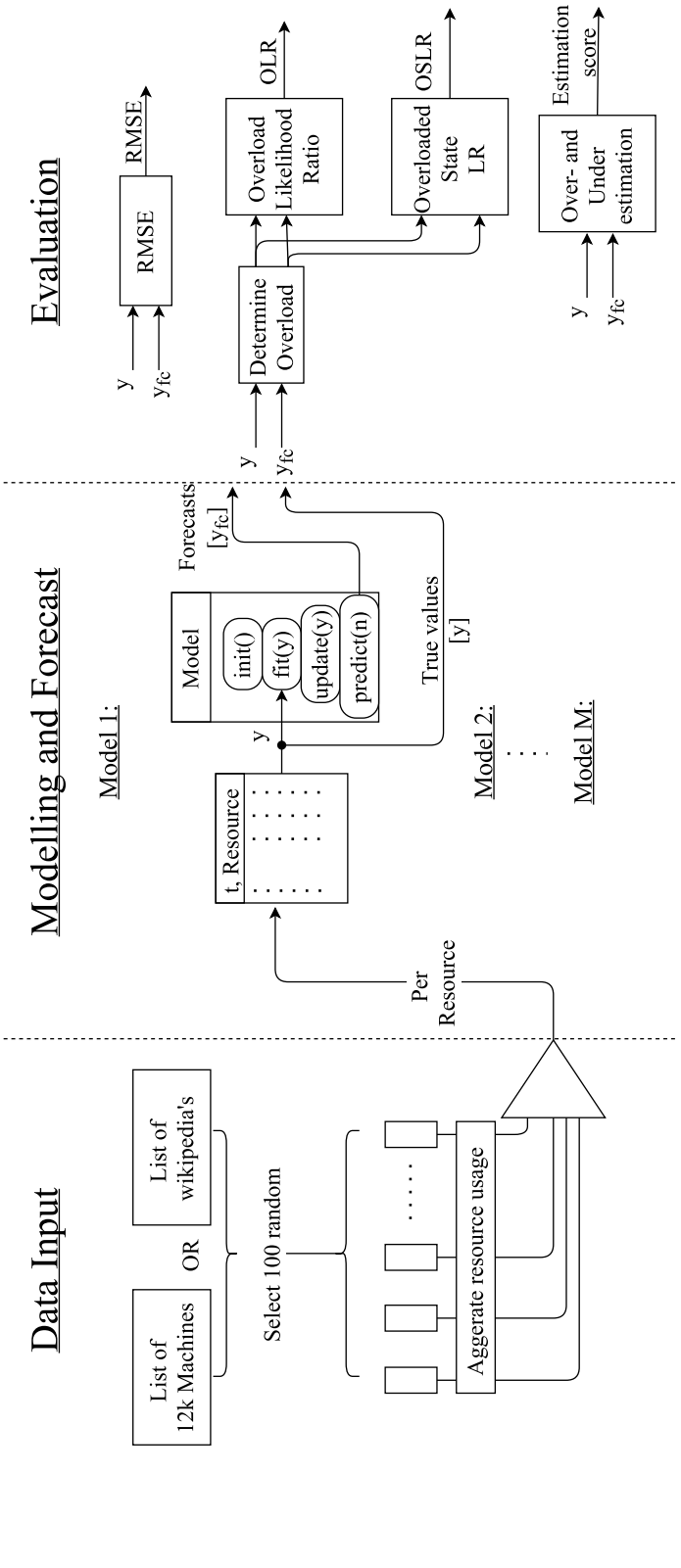


Figure 4.4: The Resource Forecasting Pipeline: the three stages used to pre-process, model and forecast and finally evaluate the forecasting models discussed and implemented in this work.

Chapter 5

Experimental Investigation

In previous chapters, prominent forecasting methods were identified from literature, the theory and formulation of forecasting equations described and the detail of implementation of these methods discussed. The forecasting methods evaluated in this chapter are listed from least complex to most complex as following:

1. MA(1): A naive forecasting method that predicts the current observation as the future forecast. This method serves as the **baseline** method.
2. MA(30): A window-average forecasting method that predicts a future value by taking an average of past observations (30 samples in this case).
3. Holt-Winters: Proposed in [68], a forecasting method that models the level, trend and periodicity of past values.
4. AR(8), AR(16) and AR(30): Three Auto-regressive models of increasing order, forecasting a value using a weighted sum of historical values. These weights are estimated using the auto-correlation function and using Yule-Walker equations.
5. Two Markov chains (first and second order): Probabilistic models that estimate the likelihood of transitioning from one level to another.
6. PRESS: An advance forecasting method that uses signature patterns and Markov chains to forecast a future value.
7. Agile: Another advance forecasting method that uses wavelet transforms to deconstruct historical workloads and forecasts future values using Markov chains.
8. Feed-forward Neural Network (FFNN): A method that learns a functional mapping from historical values to future values.

9. Recurrent Neural Network (RNN): A Elman-type NN that learns state-based functional mappings from sequential observations to future forecasts.

The purpose of this experimental investigation is to evaluate the forecasting methods using five different performance measures collected and adapted from literature. These evaluation metrics aim to measure generic prediction accuracy and forecasting performance that specifically relate to provisioning of cloud resources. We use two evaluation datasets, namely the 2011 Google cluster dataset [67] and Wikipedia's Pageview dataset [66]. Each dataset contains the resource requirements of a real-life cloud environment. The usages recorded include CPU utilisation, Memory usage, number of Page requests and Network load.

Our first investigation aims to reproduce the results obtained by PRESS [24] and Agile [47] in their respective papers. This investigation also aims to identify the order of the Auto-regression model used by both PRESS and Agile.

Next, we investigate different performance metrics used to compare forecasting methods. The first metric used is Root Mean Squared Error (RMSE) which measures a method's forecasting accuracy, by calculating the (square root) distance between the predicted values and the true values. The second metric used in this investigation is the Correct Estimation Rate (CER) which measures the ability of a method to forecast a value within an acceptable distance of the true value. Estimation Score (ES) measures how often a method over- or under-estimates when performing a forecast. The last two metrics used are Overload Likelihood Ratio (OLR) and Overloaded State Likelihood Ratio (OSLR). These measure a method's likelihood of correctly predicting an overloaded observation as well as the likelihood of correctly predicting an overloaded state.

Furthermore, we combine four different forecasting methods using three ensemble model approaches. Each forecasting method addresses a specific characteristic of cloud workloads. The first approach, calculates the average forecast of the four methods. The second approach calculates a weighted average of the forecasts produced by the four methods. Finally, we investigate an ensemble model that combines forecasts using a Feed-Forward Neural Network (FFNN).

In our concluding investigation we investigate the effects on prediction performance when using a shorter forecasting window length.

5.1 Experimental Setup

The purpose of this section is to cover the detail on the evaluation parameters and datasets used across all experiments performed in the chapter. Each experiment reported follows the structure of Motivation, Setup, Results and

Table 5.1: The Evaluation Parameters used across all experiments for fitting a forecasting model, predict using that model and post-process the forecasts.

Parameter	Value
Training window	3000 samples
Forecasting window	30 samples
Negative replacement value	5th percentile

Interpretation. The specifics of the evaluation metric used in the experiment, is given in the Setup section.

All experiments are executed using 5-fold cross-validation similar to the approach used by Agile. A 100 data sources (i.e. *machines* in the Google cluster dataset and *languages* in the Wikipedia Pageview dataset) are selected at random from the corresponding datasets. Each data-trace is split into training and testing parts, and the training data used to estimate the model. Forecasts are produced and tested against the testing data. This operation is performed five times and produces 500 distinct data traces.

The evaluation results are post-processed to reject outliers that fall outside two standard deviations of the median value. These filtered results are plot using whisker-box-plots with the top and bottom of the box representing the 25th and 75th percentile values. The whisker ends represent the 90th and 10th percentile values respectively. The centre line represents the median and the mean value is indicated with a red square. In each plot we highlight the best performing method using a green solid-line and worst performing method using red dashed ellipse.

5.1.1 Evaluation parameters

To ensure a fair comparison between forecasting methods we use the same modelling and evaluation parameters across all experiments. Table 5.1 summarises these evaluation parameters.

Training window: The training window specifies the number of samples (or past observations) used when modelling a forecasting method. Similar to Agile, we use a training window of length equal to 3000 samples (which is about a third of a data trace for both the Google cluster and Wikipedia datasets). We believe that this window length is sufficient for a method to capture the behaviour and characteristics of the workload. Initial investigation has shown that this training window length contain sufficient complete seasons for estimating the Exponential Smoothing model parameters. It also contains a sufficient number of state transitions for estimating the Markov Chain transition matrices.

Forecasting window: Similar to Agile, the forecasting window (i.e. the number of values to forecast) is set to 30 samples which is 1% of the training window size. This produces medium-to-long term forecasts and in terms of cloud provision leads to resources being made available before overload occurs.

Negative replacement value: Forecasting methods may predict negative values when resource usages suddenly drops and this change regressed, thus we follow PRESS's suggestion to replace negative values with the 5th percentile value of the total trace.

5.1.2 Datasets

The 2011 Google cluster dataset

The 2011 Google cluster usage trace dataset [67] contains the resource requirements corresponding to tasks scheduled onto each machine in Google's production cluster cell (containing over 12 000 machines). Data was recorded every 5 minutes (300 seconds) over a period of 29 days. According to Liu and Cho [41] the dataset has been sanitised to obfuscate confidential information, but still gives useful and accurate information on cluster usage and load. This is important for the evaluations performed in this thesis. Refer to Appendix B.1 for the full listing of the resources data captured.

For evaluation purposes we pre-process the dataset by aggregating the resource usages for all tasks running on a given machine within every 5 minute period. This gives the total usage (specifically CPU and Memory usages) for that machine and produces 8352 data points per machine over the 29 days.

The Wikipedia Pageview dataset

The second evaluation dataset was made available by Wikimedia Statistics [66] and published at [65]. This dataset contains the raw page-view count and network traffic for every Wikimedia project. These projects include: Wikipedia, Wikinews, Wiktionary, Wikiquotes, etc. Records were captured hourly and cover the time period from 2007 up to present day.

We make the following assumptions regarding the use of this dataset:

- Data in the Wikipedia pageview dataset is captured hourly, compared to every 5 minutes in the Google cluster dataset. For the purpose of this thesis, the dataset will be used a comparative dataset in which the cloud application is known (or at least the type of application is known). We assume that using the same evaluation parameters will ensure comparable results between datasets.

- The cloud application which produced the Wikipedia dataset, is a wiki-structured website (or series of websites) with static content. Each visit amounts to a measurable load, both on the VM resources hosting the application and the network infrastructure resources. Different to the Google cluster dataset we do not have the load presented to individual VMs and thus assume that each language specific Wikipedia, can be regarded as a datacenter. The total page-views thus corresponds to the load presented to the cloud resources within that datacenter.

For the purpose of this thesis, we limit the evaluation dataset to records from Wikipedia pages per language (approximately 1500 languages) for the year of 2014. The two statistics captured in these records are the number of page views (non-unique views) to each of the Wikipedia's per language and the total network data transferred corresponding to these views. This produces a total of 8714 observations per language, for 2014.

5.1.3 Statistical significance test

When comparing two test results, the **Student's T-test** is used in statistical significance testing, to determine if the means of the two test results differs in a statistically significant way. The purpose of this testing is to ensure that our results are not influenced by the test sample size, but that we may have confidence that the results reflect the true population. An assumption for using the Student's T-test is that the data comes from a t-distribution (with $2(N-1)$ degrees of freedom). In the case for 5-fold cross-validation where each data source (machine or language) is independent and random sampling is performed, this assumption holds.

Let S_1 and S_2 be two result sets to be compared, produced by two different forecasting methods. Let μ_1 and μ_2 be the means and σ_1 and σ_2 the standard deviation of S_1 and S_2 respectively. The results show that on average forecasting method producing S_1 performs better than the method producing S_2. We would like to test this by setting the *Null hypothesis* H_0 and *Alternative hypothesis* H_a as following:

$$\begin{aligned} H_0 &: \mu_1 \leq \mu_2 \\ H_a &: \mu_1 > \mu_2 \end{aligned} \tag{5.1.1}$$

and to reject the null hypothesis with a 95% confidence, we require a p-value less than statistical significance level of $\alpha = 0.05$.

We calculate the p-value by first calculating the t-score using both result sets and the following equation:

$$t_{score} = \frac{\mu_1 - \mu_2}{S_p \sqrt{\frac{2}{N}}} \tag{5.1.2}$$

where N is the number of samples in a results set and S_p is the *scaling parameter* calculated using:

$$S_p = \sqrt{\frac{\sigma_1^2(N_1 - 1) + \sigma_2^2(N_2 - 1)}{N_1 + N_2 - 2}} \tag{5.1.3}$$

where $N_1 = N_2 = N$.

Using the t_{score}, we calculate the probability of $P(T \leq -t_{score})$, where T follows a Student's t-distribution. If this one-sided p-value is less than 0.05 we can reject the null hypothesis with 95% confidence.

Throughout the experiments presented in this chapter we will use the statistical significance test when comparing the best and worst performing forecasting method as well as in situations where results may look similar. Please refer to Appendix C for additional tables containing all statistical significance results relevant to this chapter.

5.2 Investigate PRESS And Agile's Results

5.2.1 Motivation

In their the respective papers, the authors of PRESS [24] and Agile [47], report that their method is able to outperform AR as comparative forecasting method. As stated in Chapter 4, the specific order of the AR model used by both PRESS and Agile is unknown.

The purpose of this evaluation is to investigate which one of the three AR models (namely 8th, 16th or 30th order model) best represent the model used by PRESS and Agile respectively. We conduct two experiments in this evaluation. The first is performed using the 7 hour Google cluster dataset [28] and using the two evaluation metrics (over- and underestimation rates) used by PRESS. The second is performed using the 29 day Google cluster dataset and using the two evaluation metrics namely, true positive rate (TPR) and false positive rate (FPR) proposed by Agile.

5.2.2 Setup

PRESS: It was found that when following PRESS's experimental setup, some issues were revealed. We highlight these here and describe how they were handled:

1. A training window of size 512 samples was used by PRESS. The 7 hour Google cluster dataset follows the same structure as the 29 day dataset, specifically that samples were captured every 5 minutes, this produces 84 samples over the 7 hour period. It is unclear how PRESS uses a large training window of 512 samples. Furthermore, the authors of PRESS

also used a shorter training window of size 32 sample for their comparative methods. We decided to use this as training window size in our experiment.

2. PRESS only used three data traces (out of thousands) for their results on the Google cluster dataset. We believe this to be insufficient representing statistical significant results. It was decided to instead use 100 data traces in this experiment.

In their evaluation, the authors of PRESS performed one-ahead prediction and we do the same in this experiment.

Agile: For the experiment of comparing Agile to our three AR models we follow the experimental setup and evaluation parameters as discussed in Section 5.1. It is important to note that the authors of Agile also compared their method to PRESS. In this experiment will do the same and aim to confirm the relative performance between Agile and PRESS.

5.2.3 Results

PRESS

The results of comparing PRESS to our AR models on CPU and Memory usage data are shown in Figure 5.1.

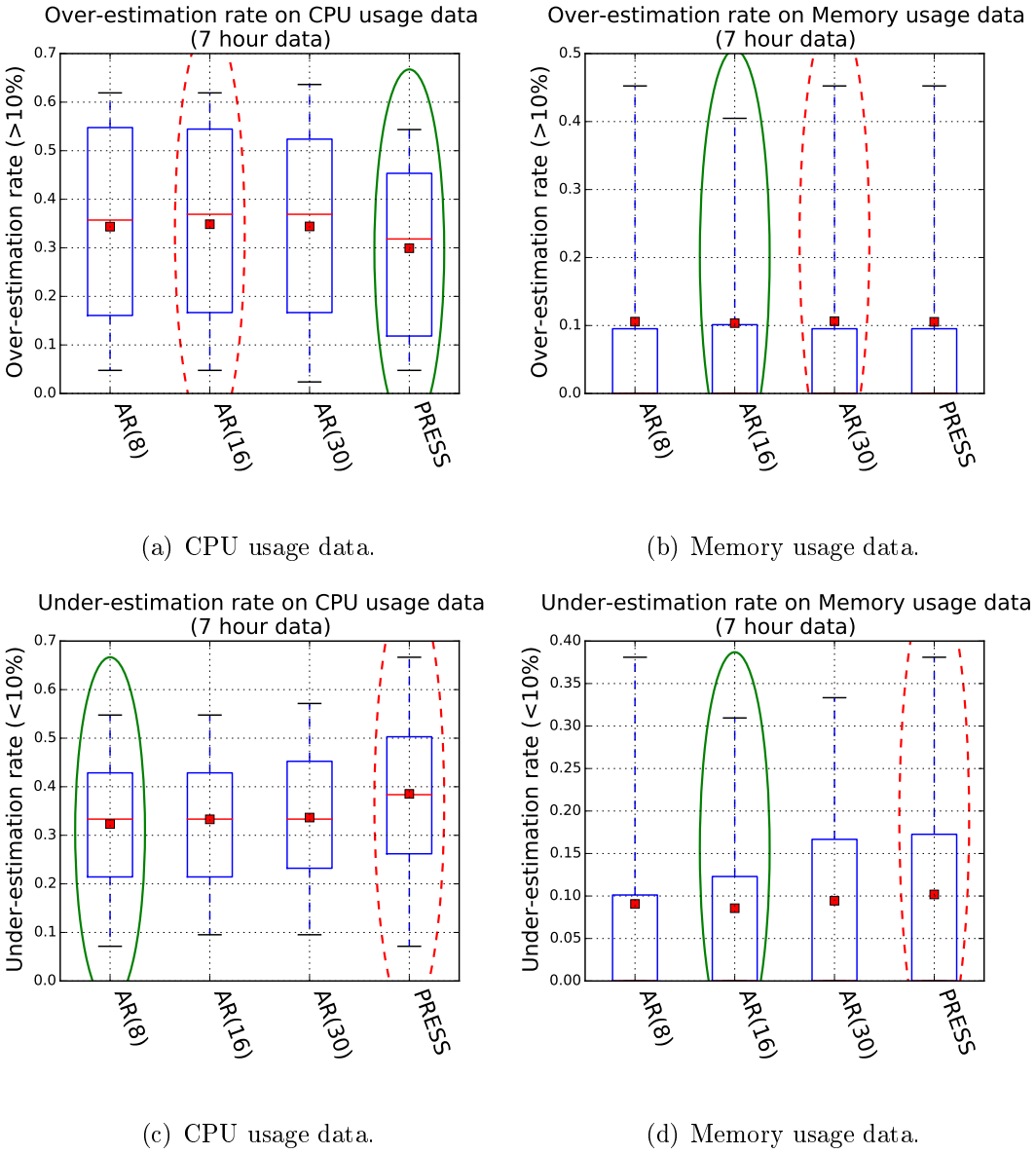
Agile

The results for comparing Agile to the three AR models and PRESS are shown in Figure 5.2.

5.2.4 Interpretation

PRESS: The results presented by the authors of PRESS show that the AR model achieved an over-estimation rate (OER) of less than 1% and that PRESS achieved an OER of 5% on CPU data. Similarly on Memory usage data, the AR model should have an OER of 5% and PRESS an OER of 10%. Both these results do not agree with the results shown in Figures 5.1(a) and 5.1(b). In our results PRESS achieves a better OER on CPU usage and a statistically similar OER to all AR models on Memory usage data. This may be contributed to the added variance when evaluating on more traces compared to the authors of PRESS.

For the results on under-estimation rates (UER) it is reported that the AR model should under-estimate more than 80% of the time compared to PRESS's 5% on CPU usage. AR also under-estimates more than 60% compared to PRESS's 30% on Memory usage data. In our results we see that all three



(a) CPU usage data.

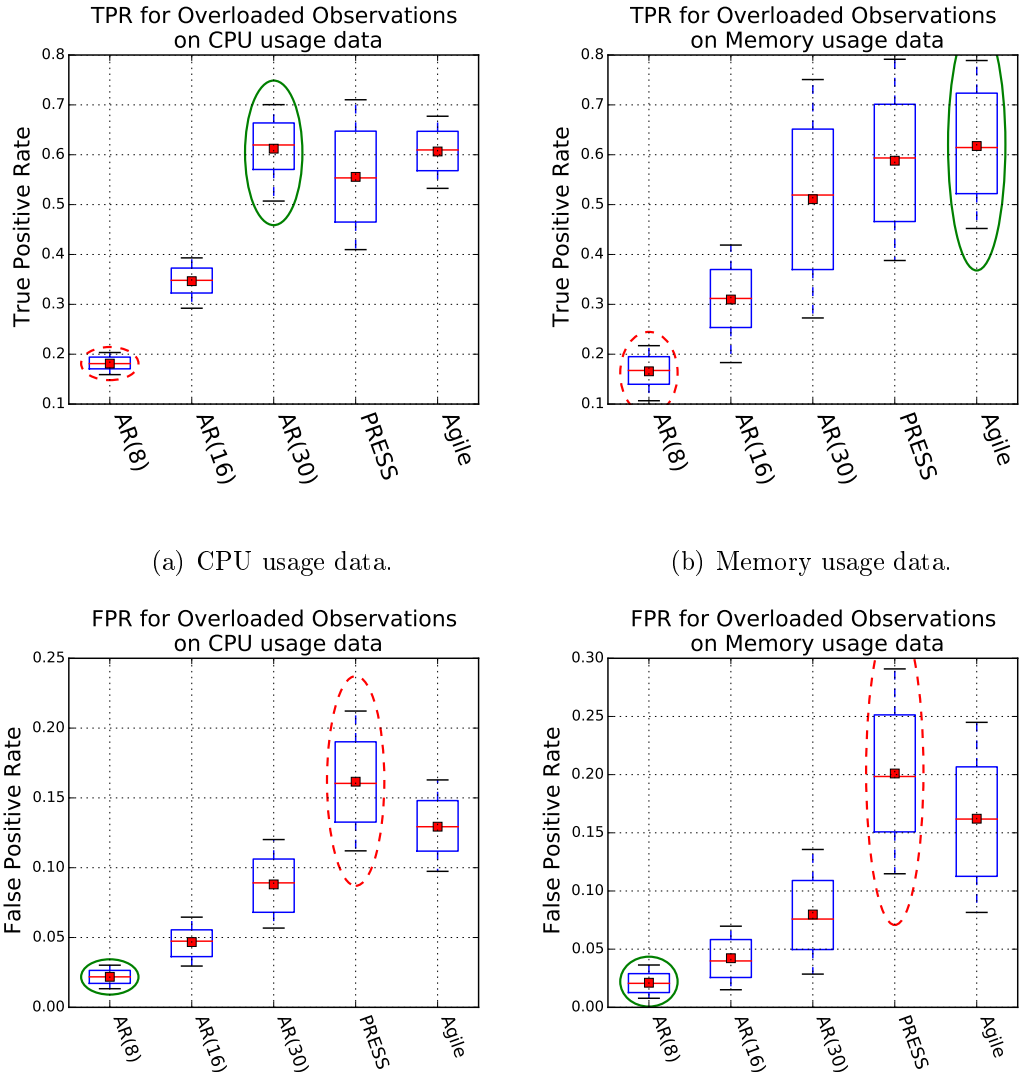
(b) Memory usage data.

(c) CPU usage data.

(d) Memory usage data.

Figure 5.1: Comparison of PRESS and three Auto-regression models of order 8, 16 and 30. The experiment was performed on the 7 hour Google cluster dataset [28].

AR model achieves an UER of above 30% and PRESS performing worse with an UER close to 40% on CPU usage data. From our Memory usage data we observe that all AR model have an UER of 10% on average. Again these results do not agree with those presented by PRESS. This may again be contributed to the larger sample of data traces (100 traces) used or to the discrepancies of the training window size used.



(a) CPU usage data.

(b) Memory usage data.

(c) CPU usage data.

(d) Memory usage data.

Figure 5.2: Comparison of Agile, PRESS and three Auto-regression models (orders 5, 16, 30). The results show that the authors of Agile may have used an AR model of order between 16 and 30.

Agile: In their paper the authors of Agile [47] reported that Agile achieved a TPR of around 90% on CPU usage data and 80% on Memory usage data when evaluated using a 100 data traces from the 29 day Google cluster dataset. It is also reported that PRESS achieved a TPR of 60% and 65% on CPU and Memory respectively, The AR model used by Agile, achieved a TPR of 45% and 40% on the usages respectively.

In our results (see Figure 5.2), we observe that Agile achieves a TPR of 60% on both CPU and Memory usage data and PRESS 55% and 60% on the

two usages. These rates are lower but do still follow a similar trend compared to the results given in [47]. Our results also suggest that the order of the AR model used by the authors of Agile was of order between AR(16) and AR(30).

Interpreting the FPR results from [47], it is reported that Agile achieved FPRs of 20% on CPU and 15% on Memory, PRESS FPRs of 35% and 40% and the AR model achieved FPRs of 25% and 20% on the two usage respectively. In our results all methods achieve lower FPRs (with lower is better) as well as Agile and PRESS achieving similarly relative results. The results produced by the AR models do not completely agree, but AR(30) has the closest resembling to the results reported.

Concluding remarks

The 7 hour results suggest that our implementation of PRESS may not agree with the one presented in [24]. The limiting factors include the details of the Markov Chain implementation and the pre-processing of the dataset itself. It is also interesting to note that higher order AR models do not improve the one-ahead prediction performance.

For Agile, we could not completely confirm the results reported by Agile's authors, but the relative performance of PRESS and Agile do agree. We could also identify that an AR model closely resembling AR(16) and to a lesser extend AR(30) was used.

Throughout the investigations presented in this chapter we will focus specifically on comparing Agile and PRESS to AR(16), but also highlight AR(30)'s results.

5.3 Evaluation Using Root Mean Squared Error

5.3.1 Motivation

In research fields such as statistical modelling and forecasting, Root Mean Squared Error (RMSE) is a popular metric used for comparing forecasting methods. The RMSE gives an indication of the forecasting error made by a specific method [32], and for the purpose of this experiment the RMSE is used as generic evaluation metric and serve as a **baseline** performance metric.

5.3.2 Setup

Scikit-learn [59] defines **RMSE** as the error measure corresponding to the expected value of the quadratic error loss. It is calculated as the square distance

between the predicted values and the true values using the following definition:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{5.3.1}$$

where n is the number of observations used, $\hat{y}_i$ the ith forecasted value and y_i the corresponding true value (or desired value when used in training a model). In terms of provisioning, a method that achieves a low RMSE is an indication of accurate forecasts.

5.3.3 Results

The RMSE results on CPU and Memory usage data are shown in Figures 5.3(a) and 5.3(b) and the Pageview and Network data results are shown in Figures 5.3(c) and 5.3(d).

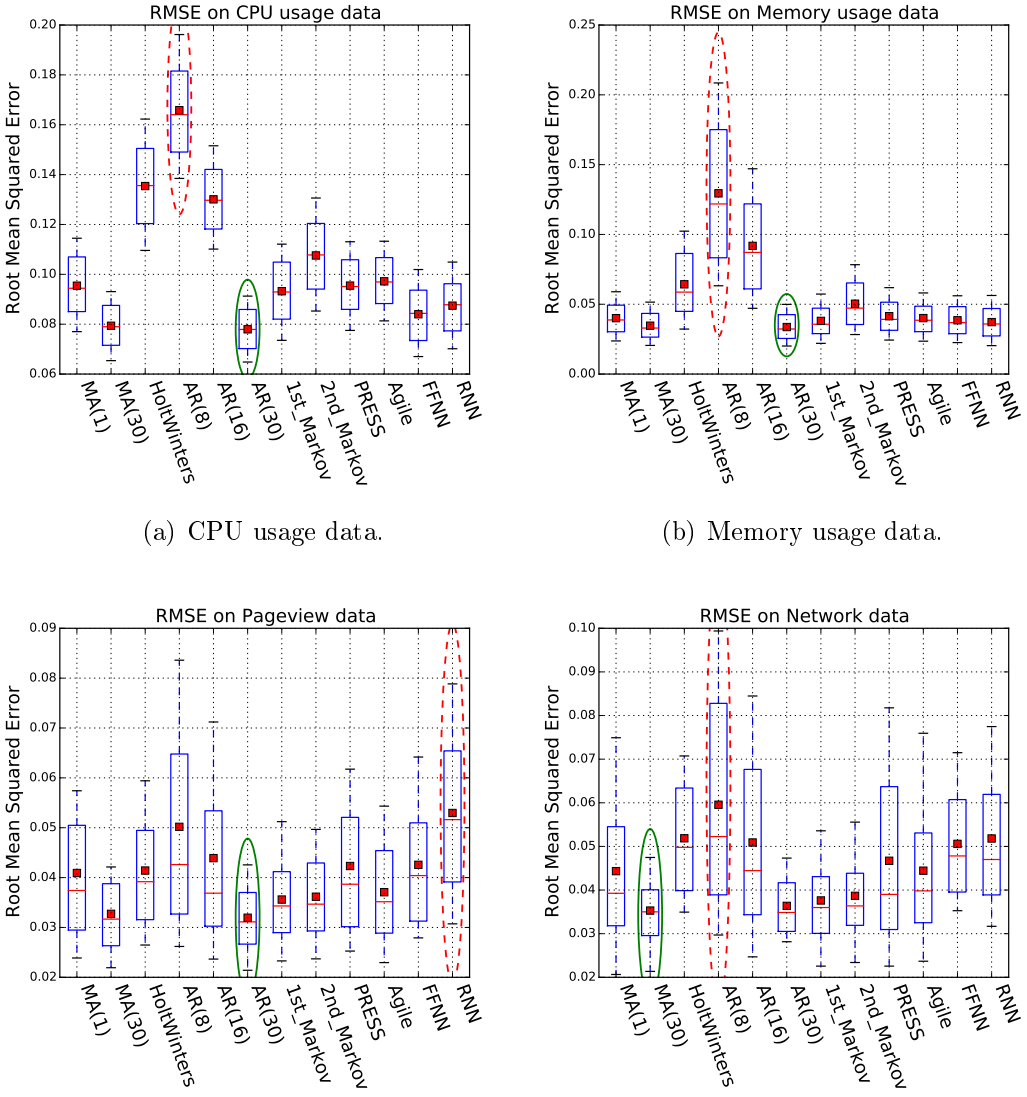
5.3.4 Interpretation

The RMSE results obtained when comparing the forecasting methods on resource data, show that the AR(30) model performs statistically better than all other methods on CPU and Pageview data. On the Memory and Network usage data MA(30) achieves similar results to the AR(30) and with a p-value of 0.1888 we cannot reject the null hypothesis. This means we do not have sufficient evidence that AR(30) is the best forecasting method when evaluated on these usages.

The AR(8) method under-performs in comparison with the other methods, on CPU, Memory and Network usage data and also RNN achieves poor results on Pageview data. A possible explanation for RNN's result is that RNNs rely on seasonality in the data and thus the performance on RMSE may suggest that the pageview data do not contain prominent seasonal components or may have measurement noise which could influence the result.

The RMSE results show that the less complex methods, namely MA(30) and AR(30), outperform the more complex forecasting methods. This suggests that RMSE might not be a sufficient metric to use when comparing more complex forecasting methods. Supporting this, the complex methods achieve statistically similar RMSE results on CPU and Memory usage data.

Interpreting PRESS and Agile's results we see that both methods perform better than AR(16), but it is also important to note that AR(30) achieves a better RMSE across all usages. This may support the theory that a higher order AR model would outperform these PRESS and Agile.



(a) CPU usage data.

(b) Memory usage data.

(c) Pageview data.

(d) Network data.

Figure 5.3: The Root Mean Squared Error results of the forecasting methods evaluated on CPU and Memory usage data from the 2011 Google cluster dataset and on Pageview and Network data from the Wikipedia Pageview dataset.

5.4 Evaluation Using Correct Estimation Rate

5.4.1 Motivation

The RMSE evaluation metric used in Section 5.3 calculated the error distance between the forecasted values $\hat{\mathbf{y}}$ and the true values $\mathbf{y}$. In terms of cloud provisioning, being able to forecast values close to the true values are beneficial but not required. Following PRESS [24], we now define an acceptable error

distance between the forecasted values the corresponding true values. We classify a forecast as a *correct estimation* of the load at that moment in time when it is predicted within the acceptable error distance.

5.4.2 Setup

Similar to PRESS we use an acceptable error distance equal to 10% of the true value. Using this we calculate the **Correct Estimation Rate** (CER) as the ratio of correctly estimated values over the total number forecasts:

$$CER = \frac{\#\text{correct estimations}}{\text{total forecasts}} \tag{5.4.1}$$

A higher CER indicates the model produces accurate forecasts.

5.4.3 Results

Figures 5.4(a) and 5.4(b) show the CERs for all methods on CPU and Memory usage data. Figures 5.4(c) and 5.4(d) show the CERs for Pageview and Network usage data.

5.4.4 Interpretation

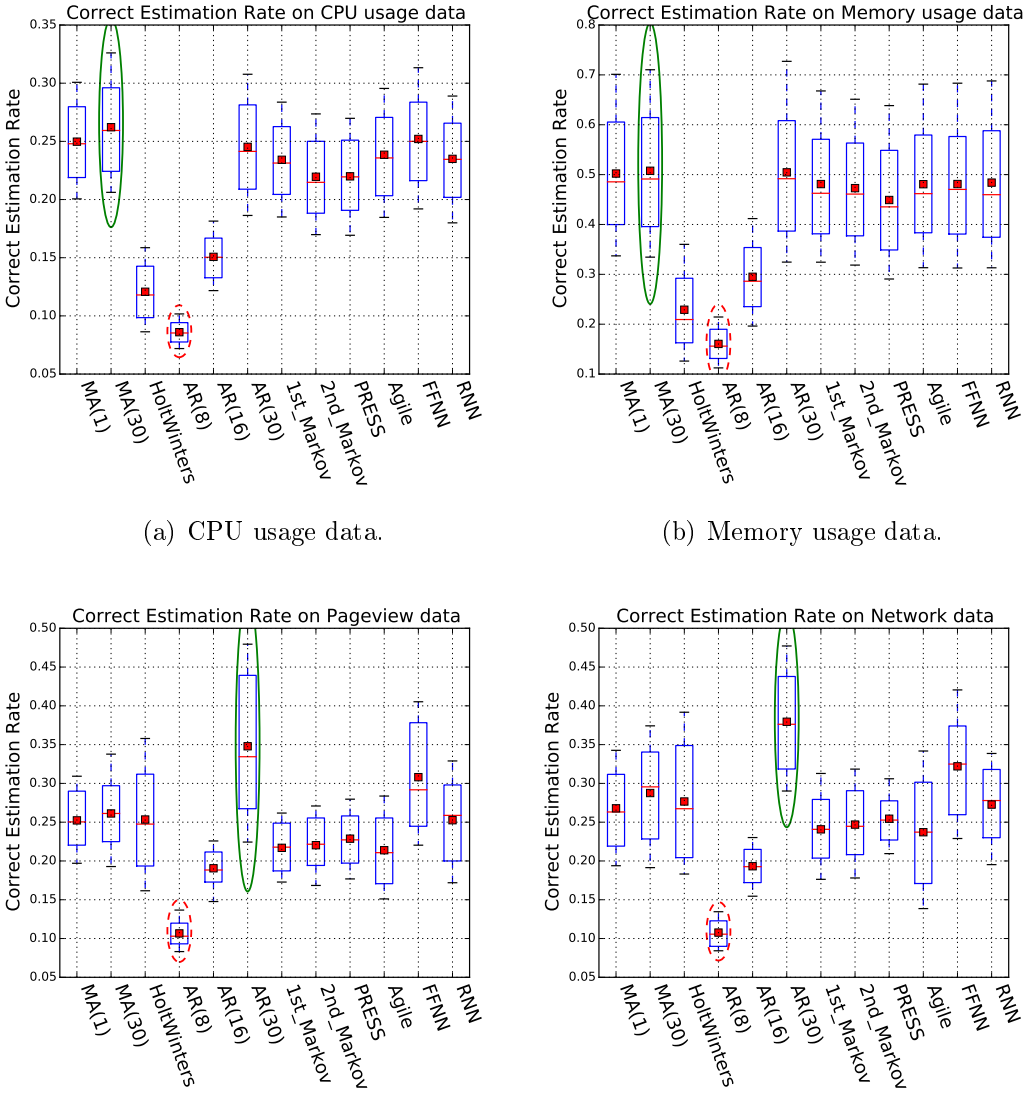
We observe that different to the RMSE results, the MA(30) model performs statistically better than other methods on CPU usage data and with a p-value of 0.3456, performs statistically similar to AR(30) on the Memory usage data. This result may suggest that the AR(30) model forecasts values closer to the true values and thus achieves lower RMSE results. AR(30) may forecast values within the 10% band less frequently compared to both MA models.

In our option, these results may also suggest characteristics of the 2011 Google cluster dataset. MA(30) uses the average of past values to forecast and its result on CPU usage data may suggest that the majority of the CPU data are within 10% of the average utilisation.

On the Pageview and Network data, the AR(30) model outperforms the other forecasting methods with a p-value less than 0.05. Also both PRESS and Agile have a higher CER than AR(16), supporting the results from Section 5.2.

Apart from Holt-Winters, AR(8) and AR(16) which under performs on CPU and Memory usage data, there is no statistically significant evidence which method achieves the lowest CER on these two usages. On Pageview and Network usage data, we see that FFNN is highlighted a second best performing method, but other more complex methods achieve similar results.

The use of correct estimation rate, as evaluation metric for provision of cloud resources may be insufficient for comparing the performance of more complex forecasting methods.



(a) CPU usage data.

(b) Memory usage data.

(c) Pageview data.

(d) Network data.

Figure 5.4: The results of the Correct estimation rates for CPU, Memory, Pageview and Network data. A forecast is a correct estimate of the load, when it is within 10% of the true value and the correct estimation rate is calculated using the ratio of correct estimations over the total forecasts.

5.5 Evaluation Using Estimation Score

5.5.1 Motivation

In cloud resource provisioning, it is less detrimental for cloud providers to over-provision resources compared to under-provisioning [5]. Over-provisioning incurs more costs, but still adhere to SLAs between providers and users.

In previous evaluations, the RMSE calculated the error distance between the forecasted values and the true values, whereby the CER specified an acceptable error distance of 10% of the true values. The purpose of this evaluation is to compare forecasting methods using the Estimation Score (ES), a metric that evaluates performance based on values being over- or under-estimated.

5.5.2 Setup

We define an **Estimation score** (ES) as linear combination of two estimation rates, namely over-estimation rate (OER) and under-estimation rate (UER). We follow PRESS's [24] over- and under-estimation metric, classifying an over-estimation as a value more than 10% of the true value and under-estimation a value less than 10%. The OER is then calculated using the ratio of over-estimated values to total forecasts, whereby the UER calculated using the ratio of under-estimated value to total forecasts.

The following is used to combine the two estimation rates, calculating the ES:

$$\text{ES} = 0.3(\text{OER}) + 0.7(\text{UER}) \tag{5.5.1}$$

A series of operation points exist (i.e. weight combinations for OER and UER) which could be used. These could be selected based on the application or based on the importance of either overload- or underload. In this thesis, we choose an operation point where over-estimation is weighted less than under-estimation and we assume that $(0.3, 0.7)$ will be sufficient weights to this end.

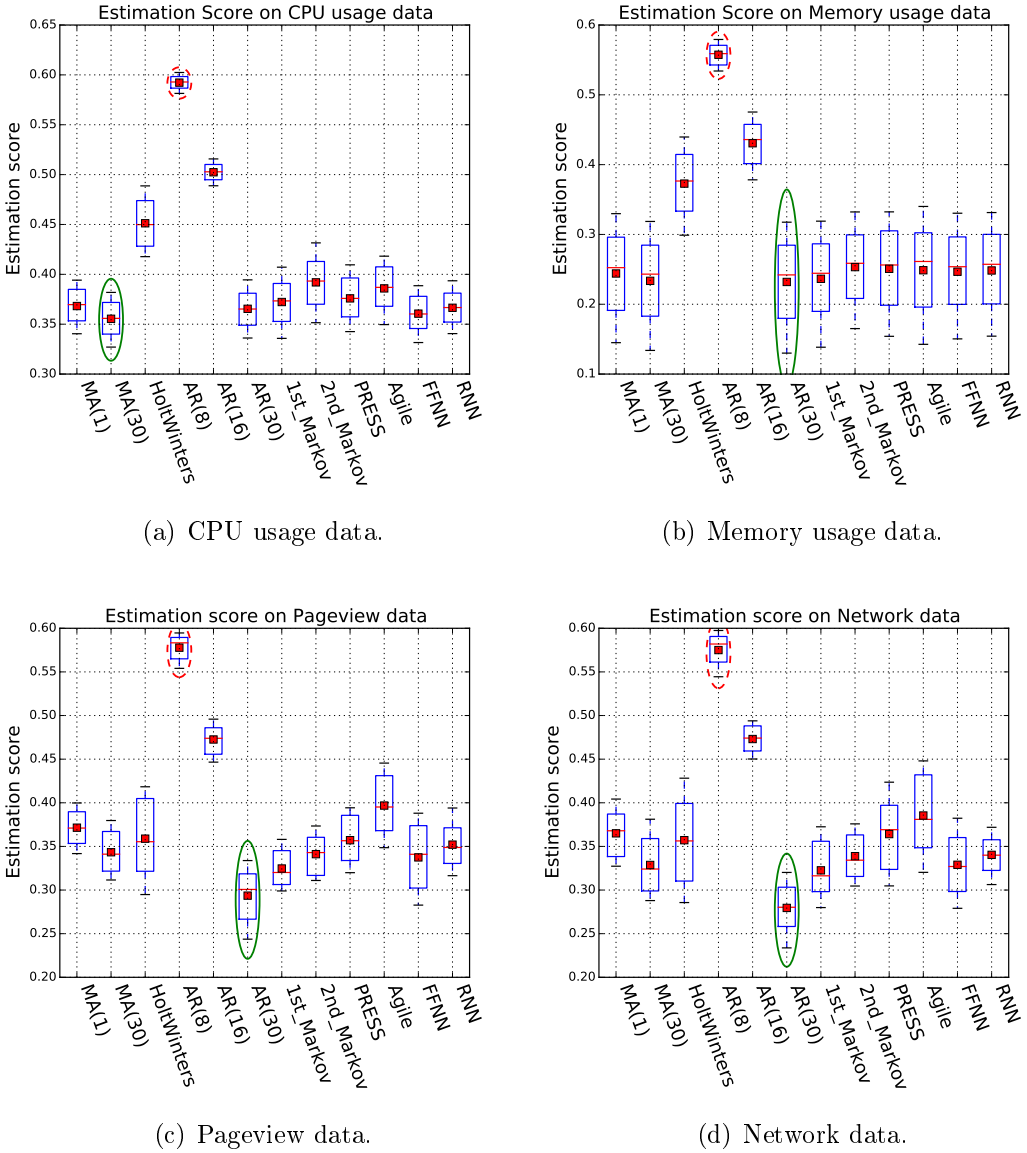
5.5.3 Results

Figures 5.5(a) and 5.5(b) show the Estimation score results for CPU and Memory usage data. Figures 5.5(c) and 5.5(d) show the Pageview and Network data results.

5.5.3.1 Interpretation

From the ES results in Figure 5.5, we observe that on CPU usage data, MA(30) performs statistically better than other methods and FFNN achieving the second best ES. On Memory usage, MA(30) achieves an ES similar to AR(30) (with a p-value of 0.2826) and thus we fail to reject the null hypothesis and can not concisely say that either AR(30) or MA(30) is better. On Pageview and Network data, AR(30) statistically achieves the best performance and AR(8) and AR(16) are highlighted as the methods that perform the worst. Again we observe that both PRESS and Agile outperform AR(16), but compared to the AE(30) method they are both outperformed.

The majority of workloads captured in the Wikipedia dataset are stable and seasonal workloads. An initial explanation of Agile's ES result include,



(a) CPU usage data.

(b) Memory usage data.

(c) Pageview data.

(d) Network data.

Figure 5.5: The Estimation Score (ES) results for all methods evaluated on CPU, Memory, Pageview and Network data. The ES is calculated by combining the over- and under-estimation rates using a weighting skewed to under-estimation. Lower Estimation scores indicate less frequent under- and over-estimations and is thus an indication of a more accurate prediction.

the inherent Markov Chain model used in Agile is unable to learn significant transitions probabilities at different scales, causing the poor ES performance.

Interpreting all results as a whole, we see dissimilarity between methods' Estimation scores on CPU, Pageview and Network data. On Memory usage data the ES is statistically inconclusive on measuring the difference between the more complex methods. All three evaluation metrics used in evaluation

thus far have not shown a significant dissimilarity between the more complex forecasting methods, which may suggest this is attributed to the Memory data itself. We hope to see supporting evidence for this in the next two evaluations.

Finally, the operation point $(0.3, 0.7)$ that was chosen and used in calculating the ES may be a limiting factor. Future investigation should be done using various operation points.

5.6 Evaluation Using Overload Likelihood Ratio

5.6.1 Motivation

When evaluating forecasting performance, the Estimation score (ES) gives us a weighted measure of how often a method under- and over-estimates. Following Agile's proposal, let us now define a specification based on a forecasting method's ability to predict *overloaded* observations.

We define an overloaded observation as one being above a predefined *overload threshold*. Note an overloaded observation is different to an *overloaded state* defined and evaluated in 5.7.2. This specification is similar to classification in the field of machine learning.

Similar to the argument given in Section 5.5.1, we assume that forecasting overload incorrectly (i.e. over-estimating) is less detrimental than predicting not-overloaded incorrectly, possibly breaking SLAs and causing End users to have a poor experience.

The purpose of this experiment is to define an evaluation metric that prioritises predicting overloaded observations correctly.

5.6.2 Setup

Following Agile's approach, the overloaded threshold in this experiment is set to the 70th percentile of the total trace. Liu and Cho reported in their paper "Characterizing machines and workloads on a Google cluster" [41] that the majority (93%) of the machines monitored in the Google cluster dataset have a capacity set to 0.5, which supports AGILE's argument for setting overload at a capacity of 0.7 and higher.

Next, we calculate the True Positive Rate (TPR) and False Positive Rate (FPR) using four standard classification conditions namely, *true positive* (T_p), *false positive* (F_p), *true negative* (T_n) and *false negative* (F_n). A predicted value is deemed a true positive when it is forecasted as overloaded and its corresponding true value is also overloaded. Similar to the true negative, where a forecasted value is less than the overload threshold and its corresponding true value is also less than the overload threshold. For false positive and false negative, the forecasted value is different to its true value.

TPR is calculated using the number of overloaded forecasts that agree with the true overloaded values, divided by the total number of true overloaded observations. FPR is calculated using the number of true non-overloaded values, falsely predicted as overloaded, divided by the total number of true non-overloaded observations:

$$\begin{aligned} \text{TPR} &= \frac{T_p}{T_p + F_n} \\ \text{FPR} &= \frac{F_p}{F_p + T_n} \end{aligned} \tag{5.6.1}$$

where a **high** TPR and **low** FPR indicates a high accuracy.

With these conditions we calculate the Positive Likelihood Ratio (LR+), denoted as **Overload Likelihood Ratio** (OLR) in this thesis. The OLR assesses the likelihood of a model to correctly forecasting overloaded observations compared to falsely predicting an overloaded observation. The OLR is calculated using the following:

$$\text{OLR} = \frac{\text{TPR}}{\text{FPR}} \tag{5.6.2}$$

where an OLR greater than 1 indicates that the model is more likely to correctly predict overloaded samples and OLR less than 1 indicates the model struggled to predict overloaded samples.

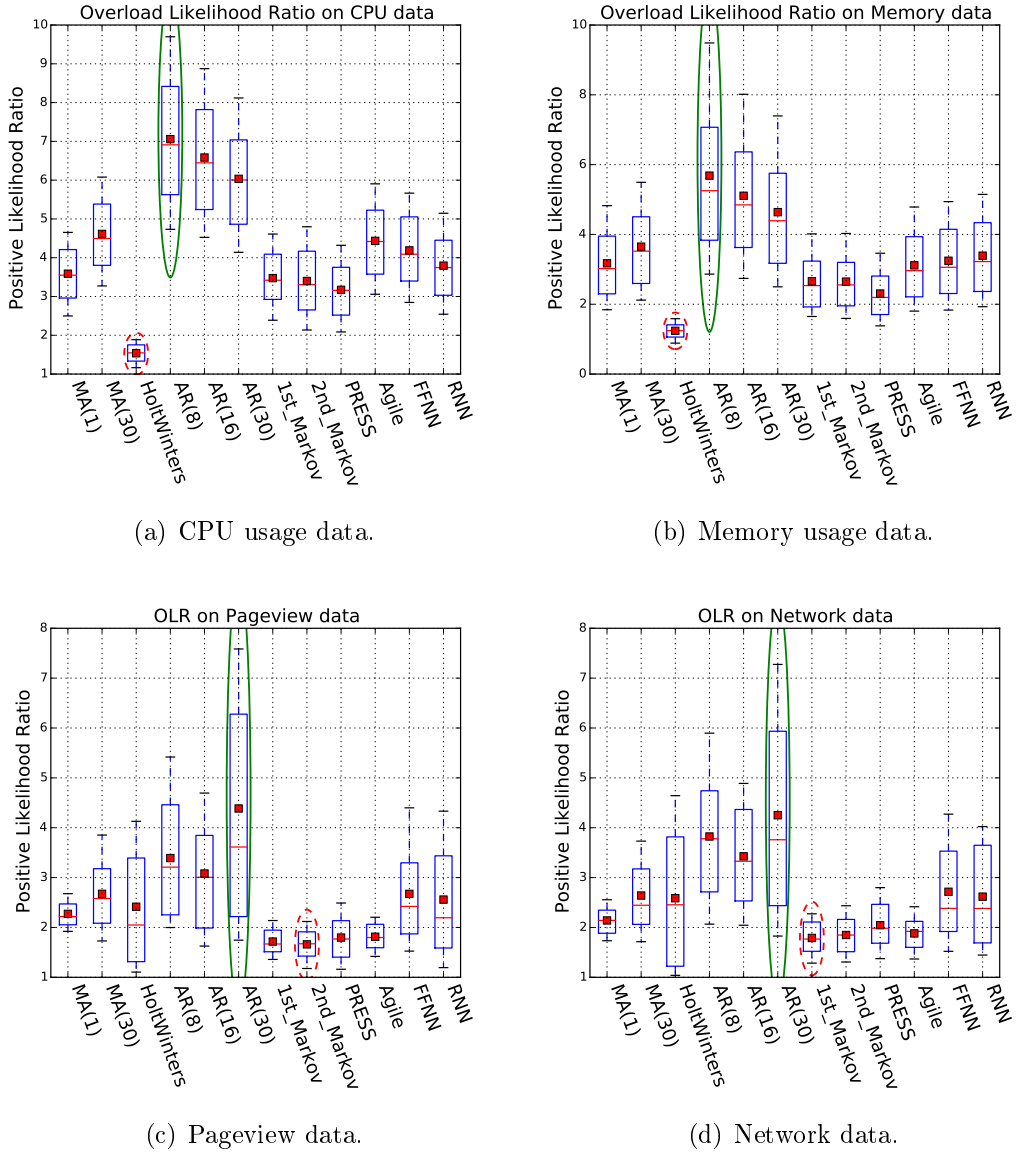
5.6.3 Results

The OLR results for CPU and Memory usage data is shown in Figures 5.6(a) and 5.6(b) and for Pageview and Network shown in Figures 5.6(c) and 5.6(d).

5.6.4 Interpretation

The OLR results suggests that the AR models have the highest likelihood to correctly predicting an overloaded observation. This result agrees with the majority of evaluations already discussed in this chapter. The large differences in performances between methods, may suggest that OLR is a representative evaluation metric that can be used to compare forecast methods. Interesting to note, both neural networks and Holt-Winters achieved significant OLRs, which may suggest that these methods are also viable forecasting methods for predicting overloaded observations. According to [19], OLR values higher than 1 are preferred with values higher than 5 indicating high likelihood for accurately predicting overloaded observations.

The more complex methods achieve an OLR of 3 or 4, indicating that these methods are all moderately likely to correctly forecast the majority of overloaded true observations on these resources.



(a) CPU usage data.

(b) Memory usage data.

(c) Pageview data.

(d) Network data.

Figure 5.6: Combining the True Positive- and False Positive Rates for correct overload predictions, produces the Overload Likelihood Ratio (OLR) results for CPU, Memory, Pageview and Network data. Values higher than 1 are preferred, with OLR values higher than 5 indicating high likelihood for accurately predicting overloaded observations.

On the Pageview and Network data, AR(30) is highlighted as best performing method Apart from the other two AR models, the two NNs are highlighted as promising forecasting methods on the Wikipedia data. Agile and PRESS are again outperformed by AR(30), but Agile is highlighted as good candidate on CPU and possibly Memory usage data.

5.7 Evaluation Using Overloaded State Likelihood Ratio

5.7.1 Motivation

In cloud provisioning, the occurrence of observations being above a predefined threshold may be very sporadic or stochastic, causing the provisioning or auto-scaling algorithms to rapidly switch between overload and non-overload. Because of this, the authors of Agile [47] defined the concept of an *Overloaded state*.

We will follow their definition, whereby Q consecutive observations are above an overload threshold (above the 70th percentile of the data). When evaluating an overloaded state, Agile defines a *grace period* or tolerance in which the forecasted state will still be considered a true positive. Figure 5.7 illustrates the overloaded state graphically.

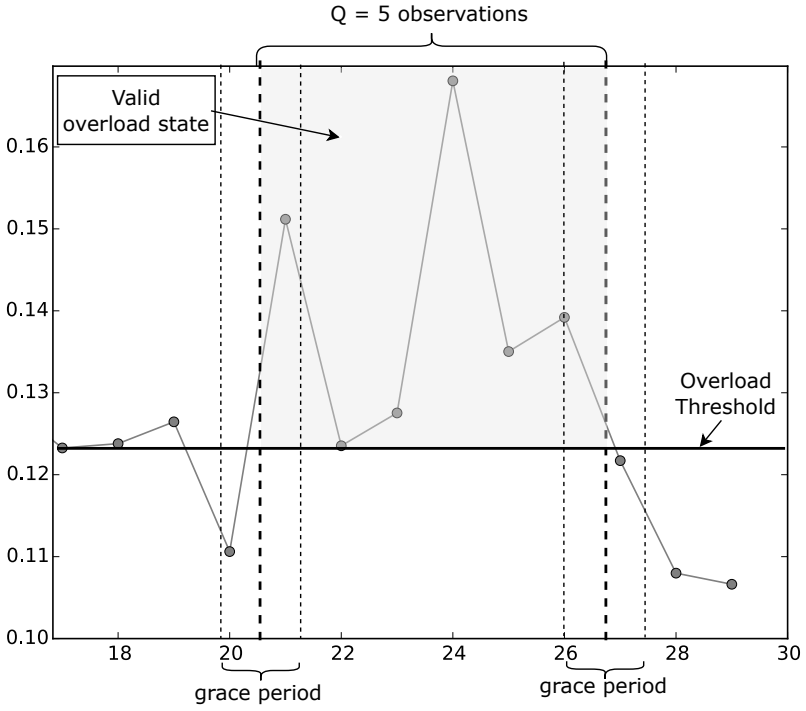


Figure 5.7: An illustration of an overload state. A valid overloaded state is one where Q consecutive observations are above the overload threshold.

The purpose of this evaluation is to compare the forecasting methods using a likelihood of accurately predicting overloaded states and comparing these results to the OLR results in Section 5.6.

5.7.2 Setup

In this evaluation we follow Agile's definition of an overloaded state, Q is set to 5 consecutive overloaded samples. We deem a *true positive* prediction as one that is predicted within the grace period (set to 3 samples) of the starting and the ending of such a state. A *true negative* prediction is one where the end of an overloaded state is predicted within a grace period of 3 samples of the true end of the overloaded state.

We calculate and combine the TPR and FPR to produce a Positive Likelihood Ratio (LR+) for overloaded states, denoted as **Overloaded State Likelihood Ratio** (OSLR) using:

$$\text{OSLR} = \frac{\text{TPR}}{\text{FPR}} \tag{5.7.1}$$

where an OSLR greater than 1 is preferred and values greater than 5 indicating a high likelihood of correctly estimating an overloaded state [19].

5.7.3 Results

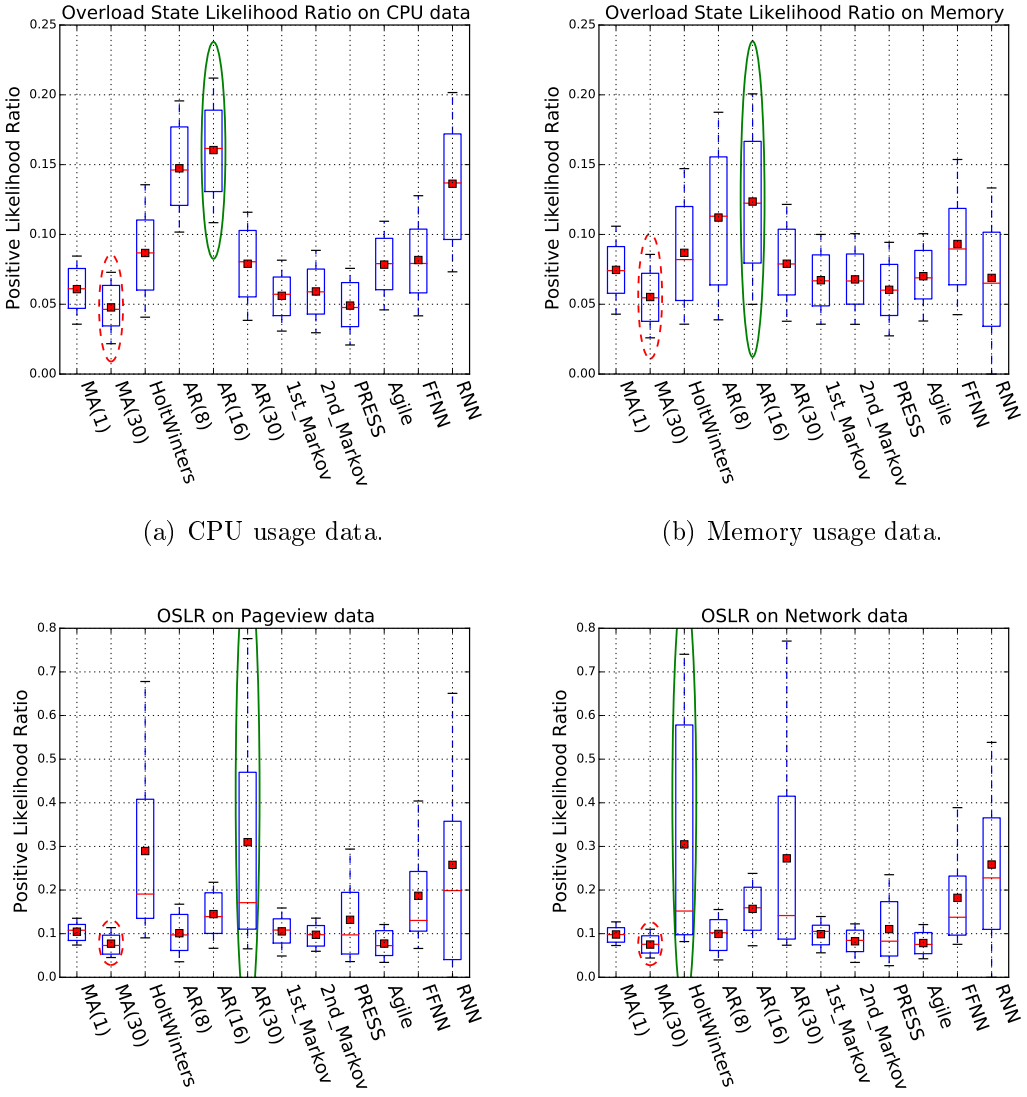
The OSLR results on the Google cluster data are shown in Figures 5.8(a) and 5.8(b) and on the Wikipedia data shown in Figures 5.8(c) and 5.8(d).

5.7.4 Interpretation

From the OSLR results we observe that AR(8) and AR(16) have very low false positive rates which cause these two methods to achieve unexpectedly high OSLRs. We also observe that the RNN achieves a high OSLR which agrees with the characteristic of RNNs, namely the ability to 'remember' state. Our FFNN statistically outperforms the RNN on forecasting accuracy on the Memory data, which is surprising because a FFNN does not have a way to remember state, which we thought would be required for accurate overload state predictions. A possible explanation for this result is that the FFNN model is able to learn a functional mapping of observations representing an overloaded state, instead of 'remembering' the context of that state.

The MA(30) model performs the worst across all usages and this result may be attributed to the MA(30) model being unable to 'remember' state.

The most surprising result is that all forecasting methods perform poorly (with OSLR values less than 0.15) when predicting overloaded states. According to [19] this is a conclusive indication that the methods are unable to accurately predict true overloaded states. These results disagree with the results obtained in the evaluation using OLR (in Section 5.7.3) as evaluation metric. We are unsure on the reason for this, but our initial theory is that granularity of samples captured in the datasets (5 minute or hour intervals) together with measurement noise may be the primary reason why all forecasting methods struggle to accurately forecast 5 consecutive overloaded observations.



(a) CPU usage data.

(b) Memory usage data.

(c) Pageview data.

(d) Network data.

Figure 5.8: Overloaded state likelihood ratio (OSLR) results for CPU, Memory, Pageview and Network data. OSLR values greater than 1 are preferred with values greater than 5 indicating a high likelihood of correctly estimating an overloaded state.

Further investigation will need to be performed in order to confirm (or reject) this theory.

5.8 Ensemble Model Evaluation

5.8.1 Motivation

The purpose of this experiment is to investigate the possible performance gain when combining forecasting methods into ensemble models. Four forecasting methods were selected, each addressing a characteristic of the workloads typically presented to cloud hosted resources:

1. Moving Average (MA(30)) which should be able to accurately model stable workloads where the deviation is close to the mean of past values.
2. Feed-forward Neural Network (FFNN) learns a functional mapping of past values to future values. We believe that FFNNs will be able to model cyclic patterns present in seasonal workloads.
3. Auto-regression (AR(30)) should be able to model trending workloads best.
4. Agile, a Markov chain based model which considers the probability of transitioning from one lower state to another higher state at different scales, and we believe should be able to model bursty workloads best.

5.8.2 Setup

We investigate three approaches to combine the four methods into an ensemble model:

- **Average Model:** Using the simple average of the forecasts produced by each methods and combining these using equal weights.
- **WA Model**: A Weighted Average (WA) model that uses past forecasts to determine different weight values for combining future forecasts.
- **Combo Model:** A 3-layer FFNN (4 input neurons, 2 hidden neurons and 1 output neuron) that determines a functional mapping between forecasts produced by each method and the future true value.

The specifics on these combining approaches are given next. The 'dataset' used in this investigation is the forecasts produced by MA(30), AR(30), Agile and FFNN on both CPU and Memory usage data.

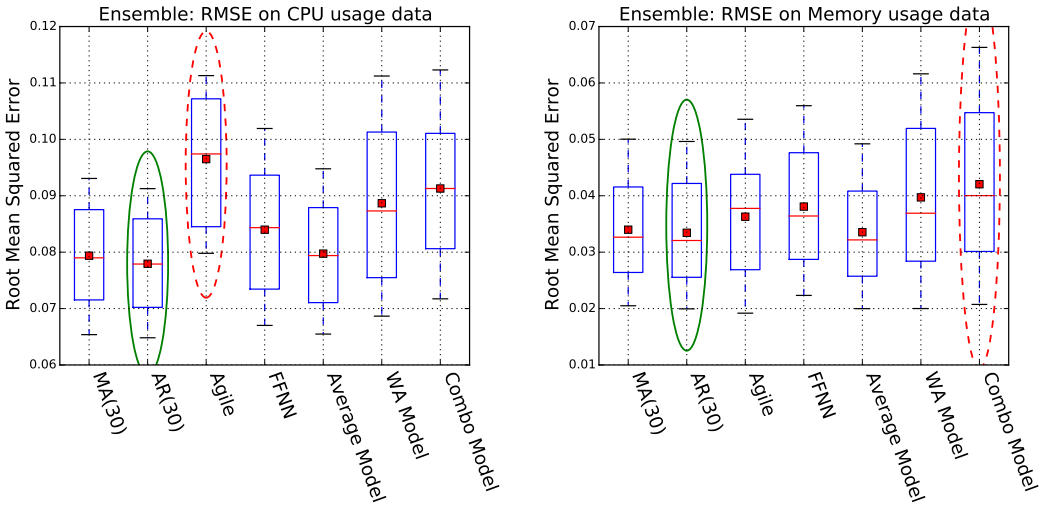
Average Model: For each forecasting window (with length equal to 30 samples) we combine the corresponding forecasts produced by the four methods using an equal weighted average and output a new set of 30 forecasted values.

WA Model: A 2-layer FFNN, with four input neurons and one linear output neuron is trained on the first forecasting window of 30 samples. It learns weights associated with each of the forecasting methods. It then produces new predictions using the four methods' latest forecasts.

Combo Model: The initial 3-layer FFNN parameters are found using random search and then trained on the first forecasting window of 30 samples. For every forecasting step, the network is activated with the forecasted values corresponding to that step and 30 new forecasts are produced. The network is then updated using the forecasts of the four methods and the corresponding true values.

5.8.3 Results

The results for the ensemble model investigation are shown in Figures 5.9 to 5.13 and the statistical significance test results are shown in Table C.2.

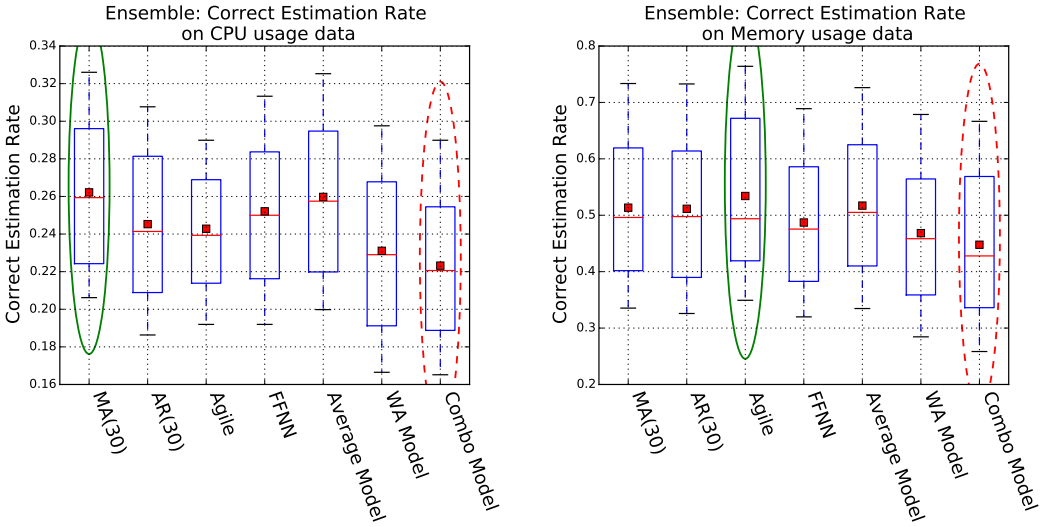


(a) CPU usage data. (b) Memory usage data.

Figure 5.9: Ensemble models: Root Mean Squared Error results.

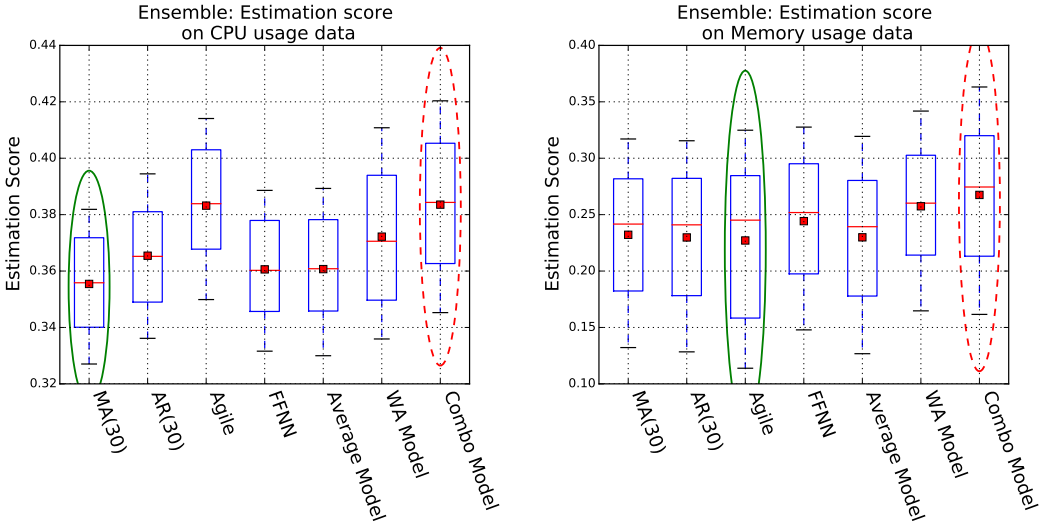
5.8.4 Interpretation

The most important conclusion that can be made from the results presented is that all ensemble models perform statistically worst (or similar) compared to the best performing forecasting method used in the ensemble. This goes against the well known property of ensemble models. In general, combining two or more analytical models will improve the predictive accuracy.



(a) CPU usage data. (b) Memory usage data.

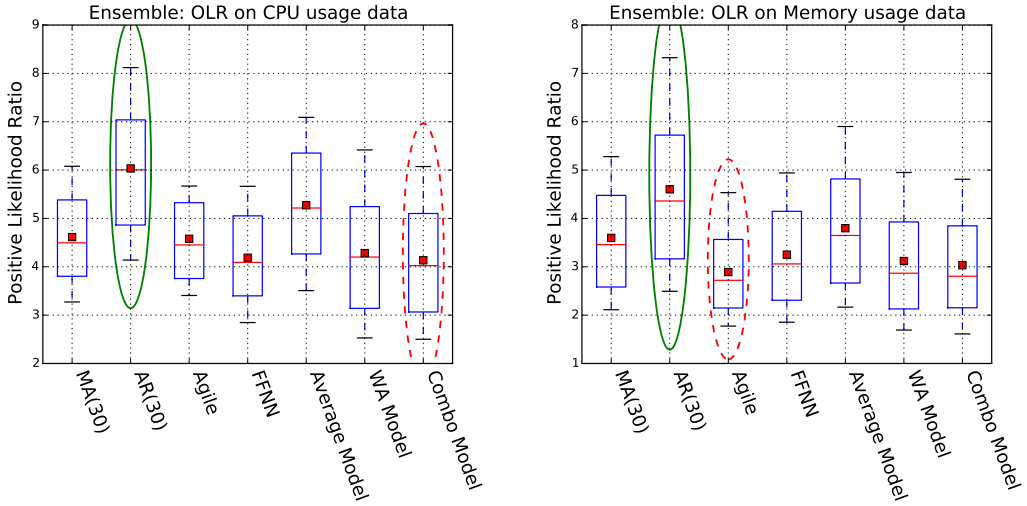
Figure 5.10: Ensemble models: Correct Estimation Rate results.



(a) CPU usage data. (b) Memory usage data.

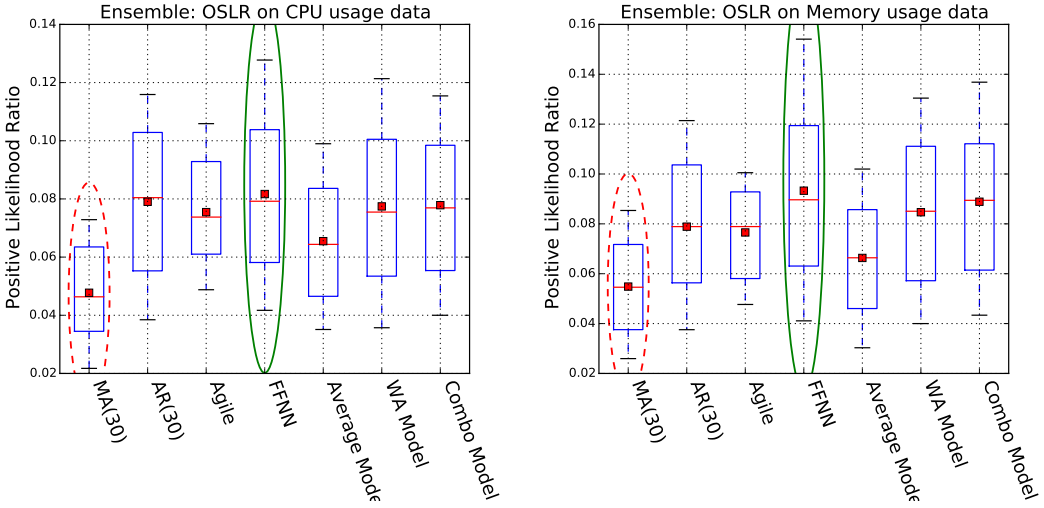
Figure 5.11: Ensemble models: Estimation Score results.

There exist several possible explanations for the unexpected results. The first of these involves the assumption that only combining the previous window's forecasts output by the four forecasting methods, will produce a good ensemble model. This might be a limiting factor, because the previous window may not contain significantly new information about the cloud resource being estimated. Future investigation into this may include using a longer look-back window which and which may possibly lead to better fitting of the ensemble



(a) CPU usage data. (b) Memory usage data.

Figure 5.12: Ensemble models: Overload Likelihood Ratio results.



(a) CPU usage data. (b) Memory usage data.

Figure 5.13: Ensemble models: Overloaded State Likelihood Ratio results.

models.

Secondly, it may be possible that there exists an implementation error within the source code of our ensemble models or that the neural network Python package used in this thesis (PyBrain [58]) is not designed to be used in this manner. Further investigation and re-implementation may aid in determining the existence of an error within the implementation source code or limitations of the Python package usage.

Thirdly, the assumption that just combining the forecasts of four individual forecasts methods will improve accuracy may be flawed. The types of workloads presented to cloud resources may change over time and this may not be as prominent in the forecasts produced by the individual methods. It might be beneficial to design an ensemble model that implements all four types of forecasting methods and based on the workload type and employs the relevant modelling and forecasting procedure.

5.9 Investigate Shorter Forecasting Window

5.9.1 Motivation

The forecasting window length used in the main evaluations performed in this chapter, is set to 30 samples following the authors of Agile [47].

The purpose of this evaluation is to investigate the effects of using a shorter forecasting window length and also verify the best and worst performing methods. Our hypothesis is that we will see an improvement in forecasting performance. This is because the forecasting methods are updated and refit more often and the cumulative error of using forecasted value as 'true values' will be less.

5.9.2 Setup

To investigate the effects of using a shorter forecasting window length, we re-run all forecasting methods on CPU and Memory usage data using a **Forecasting window length** of 15 samples. Evaluations are performed using the five evaluation metrics and these results are compared to the 30-sample forecasting window evaluations.

For this investigation we use data from the 2011 Google cluster dataset and only select a 100 randomly machines/traces, thus implicating that no cross-validation is performed. The average performance change is calculated by comparing the mean values of the 30-sample window results and the 15-sample window results per metric. The average change across all methods are reported. Statistical significance tests (with $N = 100$ and confidence level of 95%) are used to determine if these performance changes are significant or not.

5.9.3 Results

The summarised results for investigating a shorter forecasting window length are shown in Table 5.2. For more detail results please refer to the Appendix C.3.

Table 5.2: Comparing two forecasting window lengths: a 30 sample window and 15 sample window (denoted by *). The best method for each window length is listed and the average increase in forecasting performance shown.

Metric & Data type	30-Samples: Best Method	15-Samples: Best Method	Average gain
RMSE on CPU data	AR(30)	AR(30)*	5.65%
RMSE on Memory data	AR(30)	AR(30)*	5.60%
Correct Est. Rate on CPU data	MA(30)	FFNN*	7.50%
Correct Est. Rate on Memory data	AR(30)	AR(30)*	5.72%
Estimation score on CPU data	MA(30)	MA(30)*	2.66%
Estimation score on Memory data	AR(30)	AR(30)*	5.54%
OLR on CPU data	AR(30)	AR(30)*	11.40%
OLR on Memory data	AR(30)	AR(30)*	12.77%
OSLR on CPU data	RNN	Agile*	47.09%
OSLR on Memory data	RNN	FFNN*	58.46%

5.9.4 Interpretation

The results shown in Table 5.2 reveal that the majority of forecasting methods achieve statistically better performance on all evaluation metrics when evaluated on CPU and Memory usage data. The Average performance gain varies from 2.66% (Estimation score on CPU data) to a surprising 58.46% (OSLR on Memory data). When comparing the best performing method for each evaluation metric, we observe that the majority of results highlight the same method except for CER and OSLR on CPU data and OSLR on Memory data.

The results verify our hypothesis that using a shorter forecasting window does improve the forecasting accuracy. This can be contributed to the fact that the forecasting methods are updated more frequently and less of a cumulative error is made when performing medium-to-long-term forecasts.

In terms of the objective of this thesis, to compare forecasting methods in the same evaluation environment, we see consistency between the 30-sample and 15-sample forecasting window results. The most important result is the large performance gain observed on OSLR. Agile is highlighted as the best forecasting method on CPU usage data, which agrees with the results reported by its authors.

Only evaluating on CPU and Memory usage data and using a specific forecasting window length of 15 samples, may be limiting factors in this evaluation. Future investigation will need to be performed to possibly find the optimum forecasting window length and re-evaluate on data from the Wikipedia

dataset. Investigation on the performance change for the ensemble models using a shorter forecasting window length should also be performed.

5.10 Summary

This chapter described and reported on the formal investigation performed as the main contribution of this thesis. The global objective and evaluation parameters were outlined and information on the two datasets used in evaluation was discussed. Statistical significance tests (using the one tailed Student's T-test) were performed throughout the investigations. We will now summarise each investigation performed:

- Two experiments were executed aiming to confirm the results reported by the authors of PRESS [24] and Agile [47] respectively. These two methods were compared to three Auto-regression models of order 8, 16 and 30. The results reported by PRESS could not be confirmed, but the Agile results agrees in the general trend. From the Agile results we identified that the AR model used was of order between 16th and 30th. Further investigation should be done in order to ensure the versions of implementations match with what was presented by the authors of PRESS and Agile.

- Evaluation done using RMSE as generic and baseline accuracy metric showed that the AR(30) model performed the best and AR(8) the worst, but failed to give a statistically significant difference between more complex forecasting methods.

- The Correct Estimation Rate (CER) results highlighted MA(30) as best method on CPU and Memory usage data. AR(30) was the best method on Pageview and Network data. It was found that CER is also not well suited to differentiating between more complex forecasting methods, especially on Memory usage.

- Evaluations done using Estimation Score (ES) (a weighted combination of over-and under-estimation rates) showed AR(30) to be the best method. Apart from the two lower order AR models, Agile performed the worst on Pageview data which was surprising. A possible limitation of this evaluation was the specific operation point used. Future investigation should be done in finding the optimum ES operation point.

- The evaluation performed using Overload Likelihood Ratio (OLR), a measure of the likelihood that a method will correctly forecast an overloaded observation, has highlighted AR(30) as the best performed method. All methods achieved high OLR values, indicating a moderate likelihood

that they will correctly forecast overload. The OLR as evaluation metric gave a realistic measure of forecasting performance and highlights differences when comparing multiple methods.

- Overloaded State Likelihood Ratio (OSLR) measured a forecasting methods ability to correctly forecast an overloaded state. From the evaluation, RNNs, Holt-Winters and FFNN were highlighted as the best performing methods. All OSLR values were very low which may be suggest that predicting 5 consecutive values is a very difficult task. Also, these low values may have us question the validity of the results.

- The possible increase in performance of combining four forecasting methods was investigated using three ensemble model approaches. The four forecasting methods used were MA(30), AR(30), Agile and FFNN. The three combining approaches were a simple average, a weighted average and a FFNN-based approach. These results showed unexpectedly that all ensemble models performed statistically worse than the single best forecasting method used in the ensemble. A possible explanation for this includes that only combining the forecasts produced by each forecasts method may not be sufficient. A suggestion for future work is to investigate an ensemble model which contains multiple methods and selects an appropriate method for the specific workload presented.

- The effects of using a shorter forecasting window length (of size 15-samples) were investigated. Our hypothesis was that we would see an increase in prediction accuracy and this was confirmed. In terms of comparing forecasting methods, the majority of best performing methods agreed with the results obtained using 30-sample window length. All metrics showed an increase in performance, with notably OLR and OSLR showing the highest increase and also Agile being highlighted as the best method by OSLR on CPU data.

Chapter 6

Conclusions

The objective of the work presented in this thesis was to identify, implement and compare prominent forecasting methods for effective provisioning of cloud hosted resources. After implementing a collection of diverse forecasting methods, we evaluated and compared these methods. Evaluations were performed using the same evaluation parameters, performance metrics and two real-life datasets. The two datasets are the 2011 Google Cluster dataset [67] and Wikipedia Pageview dataset [65].

6.1 Summary of Work

6.1.1 Cloud workloads and forecasting methods

In order to identify, evaluate and compare forecasting methods, we investigated the types of workloads presented to cloud resources. It was found that cloud resources are typically presented with four types of workloads, namely: (1) Stable, (2) Trending, (3) Seasonal/Cyclic and (4) Bursty/Stochastic, each of these associated with a type of web-application or service.

Using these workload types and recent research on the topic of cloud provisioning and scaling, we identified the following prominent forecasting methods: Moving Average, Exponential Smoothing (specifically the Holt-Winters' method), Auto-Regression, Markov Chains, Neural Networks (which includes Feed-Forward Neural Networks and Recurrent Neural Networks) as well as PRESS and Agile.

6.1.2 Evaluation metrics

Formal evaluation metrics were adapted from literature for the purpose of this thesis. These metrics include a generic time-series metric, Root Mean Squared Error (RMSE) as well as four provision specific metrics: Correct Estimation Rate, Estimation score (a weighted combination of over- and under-estimation rates), Overload Likelihood Ratio and Overloaded State Likelihood Ratio. The

two likelihood ratios use accuracy measures similar to those proposed by the authors of Agile [47]. These evaluation metrics allow for a more concise measurement of the performance of athe forecasting method being evaluated.

A Resource Forecasting Pipeline was developed in this work and could serve as basis for a formal testing and comparison framework. It facilitates the use of new cloud resource datasets (pre-processed) as input data. New forecasting methods can be implemented using the class signature illustrated in Section 4.7 and new evaluation metrics added to the final stage of the pipeline.

6.1.3 Experimental investigations

Each forecasting method was evaluated on two datasets namely, the 2011 Google Cluster dataset [67] (containing CPU and Memory usage data) and the Wikipedia Pageview dataset [65] (containing Pageview and Network usage data) and we used the evaluation metrics discussed.

In their respective papers, PRESS and Agile describe their approaches as improvements over other forecasting methods such as Auto-regression (AR) and Weighted Moving Average (WMA). We investigated this by evaluating PRESS on Google's 7 hour cluster dataset and comparing it to our three AR models. We conclude that we could not reproduce PRESS's results and that there is no significant difference between the different order AR models used for one-ahead predictions.

We performed a similar experiment for Agile and found that our Agile results agree relative to what was reported in [47]. We also concluded that the AR model order used by Agile is between AR(16) and AR(30). It is important to note that both PRESS and Agile are still outperformed by AR(30) model. Efforts were made to contact the authors of PRESS and Agile, in regards to accessing their implementations, but no response was received.

Investigations were performed on combining four forecasting methods, each addressing a characteristic of cloud workloads using three ensemble approaches.

Finally, an investigation into using a shorter forecasting window was performed in order to verify the validity of the evaluation results obtained in the first five evaluations. The results showed that the 15-sample forecasting window length achieved better performance but more importantly it agrees with the 30-sample forecasting window results.

6.2 Concluding Perspective

The purpose of this section is to emphasise the most important results and findings of the research done in this thesis.

It was found that there is no single forecasting method that outperforms all other methods, but our 30th order Auto-regressions did achieve high performance results on the majority of metrics and across both datasets. The sec-

ond and third best performing models were Moving Average and Feed-Forward Neural Networks, with the Moving Average method result surprising because of the simplicity of the method. This suggests that less complex forecasting methods are sufficient for accurately forecasting on cloud resources. We believe more effort needs to be put into developing better evaluation metrics of workload modelling schemes.

Assessing the evaluation metrics used in this thesis, we found that there is no single metric that gives a concise performance measure of evaluating provision accuracy. From the metrics used, we believe that the Overload Likelihood Ratio (OLR), a measure of the likelihood that a method will correctly predict overloaded observations, delivers the best comparative difference between the forecasting methods investigated. The method showed a large increase in performance when used to evaluate on the shorter forecasting window.

The poor performance of the ensemble models investigated were unexpected and at present there does not exist a clear explanation of the root cause of these results. We hope that future investigations will reveal more information on this.

The limitations of the research presented in this thesis include, (1) the investigations were only performed using historical datasets of cloud resources and not on a real-life cloud. (2) The data sources used were only pre-processed to facilitate modelling and forecasting and no additional pre-processing methods considered or used, which may have improved the results. (3) The investigation of ensemble models is incomplete and inconclusive, evident from the unexpected results and poor performance.

6.3 Recommendations

- The use of pre-processing methods on the input data, such as smoothing, zero-padding or under-sampling, could present more accurate modelling and forecasting by reducing measurement noise or effects of bursty workloads and should be investigated.

- Investigate the optimum forecasting window length, still considering how this length relates to real-life cloud environments and the limitations on Virtual Machine spin-up or spin-down time.

- Greater effort should to be put into developing evaluation metrics that are specifically tailored to the cloud environment and that are able to measure forecasting accuracy within context of cloud resource management. Possible features to take into account include; Service Level Agreement (SLA) violations and the energy usage of Virtual Machines (VMs) or datacenters.

- Further investigation should be done on the specifics of PRESS and Agile's implementations, as it is still unclear if the results found in this thesis can be contributed to the differences in our implementations or not.
- Investigate model penalising schemes which could increase the forecasting method's Estimation scores by lowering SLA violations.

6.4 Future Work

- In terms of provisioning of cloud hosted resources, we believe the next step is to implement and evaluate the forecasting methods investigated into a closed-loop cloud environment. This involves using a commercial cloud platform, presenting it with live workloads or stimulate it using real world data traces. Have a resource management system provisioning and allocating resources as needed and have an evaluation module record performance metrics (possibly with a focus on energy usage).
- Further research needs to be done on improving the profiling of workloads presented to cloud resources, possibly in real-time. A better understanding of the effects different workloads have on the physical hardware together with additional objectives specified by the cloud user, may lead to improvements to provision. The scheme may learn and adapt faster to changes in the workload.
- With better workload profiling, effort should be put into the development of an adaptive ensemble model that implements various (simpler) forecasting methods and adapts to the type of workload presented at that moment in time.
- The use of deep-learning (such as Convolution nets and Boltzmann machines) or Probabilistic Graphical Models (PGMs) should be investigated and compared to the methods investigated in this thesis. These may be able to learn more complex functions or behaviours inherent to cloud workloads and better map load requirements to resource provisioning and auto-scaling.

Appendices

Appendix A

Derivations

A.1 Yule-Walker Equations

The following derivation taken from literature [21].

The Yule-Walker equations are used to estimate the Auto-regression, AR(p) of order p, model parameters ϕ_i.

The general equation of an AR(p) is given by:

$$y_{t+1} = \phi_1 y_t + \phi_2 y_{t-1} + ... + \phi_p y_{t-p+1} + \epsilon \tag{A.1.1}$$

where ϕ_i is the model parameters and ϵ the model fitting error and the series length is equal to N.

A.1.1 For lag of 1

We will derive the equations using mathematical induction.

1. We start by multiplying both sides of the model by y_t,

$$y_t y_{t+1} = \sum_{i=1}^{p} (\phi_i y_t y_{t-i+1}) + y_t \epsilon \tag{A.1.2}$$

2. then take the expectation $\mathbf{E}[\cdot]$,

$$\mathbf{E}[y_t y_{t+1}] = \sum_{i=1}^{p} (\phi_i \mathbf{E}[y_t y_{t-i+1}]) + \mathbf{E}[y_t \epsilon] \tag{A.1.3}$$

3. note that $\mathbf{E}[y_t \epsilon] = 0$ because the error is uncorrelated with previous values of the series,

$$\mathbf{E}[y_t y_{t+1}] = \sum_{i=1}^{p} (\phi_i \mathbf{E}[y_t y_{t-i+1}]) \tag{A.1.4}$$

4. divide through by $(N-1)$ and use the auto-covariance property that $C_{-t} = C_t$ we get,

$$C_1 = \sum_{i=1}^{p} \phi_i C_{i-1} \tag{A.1.5}$$

5. and the auto-covariance C_t for a stationary time-series has the property,

$$C_t = \sigma^2 \rho_t \tag{A.1.6}$$

ρ_t the auto-correlation of the time-series at t

6. divide equation A.1.5 through by σ^2,

$$\rho_1 = \sum_{i=1}^{p} \phi_i \rho_{i-1} \tag{A.1.7}$$

A.1.2 For lag of 2

1. Multiplying both sides of the model by y_{t-1},

$$y_{t-1}y_{t+1} = \sum_{i=1}^{p} (\phi_i y_{t-1} y_{t-i+1}) + y_{t-1}\epsilon \tag{A.1.8}$$

2. then take the expectation $\mathbf{E}[\cdot]$,

$$\mathbf{E}[y_{t-1}y_{t+1}] = \sum_{i=1}^{p} (\phi_i \mathbf{E}[y_{t-1} y_{t-i+1}]) + \mathbf{E}[y_{t-1}\epsilon] \tag{A.1.9}$$

3. $\mathbf{E}[y_{t-1}\epsilon] = 0$,

$$\mathbf{E}[y_{t-1}y_{t+1}] = \sum_{i=1}^{p} (\phi_i \mathbf{E}[y_{t-1} y_{t-i+1}]) \tag{A.1.10}$$

4. divide through by $(N-1)$ and use the auto-covariance property that $C_{-t} = C_t$ we get,

$$C_2 = \sum_{i=1}^{p} \phi_i C_{i-2} \tag{A.1.11}$$

5. divide equation A.1.11 through by σ^2,

$$\rho_2 = \sum_{i=1}^{p} \phi_i \rho_{i-2} \tag{A.1.12}$$

A.1.3 For lag of k

1. Multiplying both sides of the model by y_{t-k},

$$y_{t-k}y_{t+1} = \sum_{i=1}^{p}(\phi_i y_{t-k}y_{t-i+1}) + y_{t-k}\epsilon \tag{A.1.13}$$

2. then take the expectation $\mathbf{E}[\cdot]$,

$$\mathbf{E}[y_{t-k}y_{t+1}] = \sum_{i=1}^{p}(\phi_i \mathbf{E}[y_{t-k}y_{t-i+1}]) + \mathbf{E}[y_{t-k}\epsilon] \tag{A.1.14}$$

3. $\mathbf{E}[y_{t-k}\epsilon] = 0$,

$$\mathbf{E}[y_{t-k}y_{t+1}] = \sum_{i=1}^{p}(\phi_i \mathbf{E}[y_{t-k}y_{t-i+1}]) \tag{A.1.15}$$

4. divide through by $(N-1)$ and use the auto-covariance property that $C_{-t} = C_t$ we get,

$$C_k = \sum_{i=1}^{p}\phi_i C_{i-k} \tag{A.1.16}$$

5. divide equation A.1.16 through by σ^2,

$$\rho_k = \sum_{i=1}^{p}\phi_i \rho_{i-k} \tag{A.1.17}$$

The same holds for lag of p,

$$\rho_p = \sum_{i=1}^{p}\phi_i \rho_{i-p} \tag{A.1.18}$$

Rewriting the equations in matrix form we get:

$$\mathbf{r} = \mathbf{R}\cdot\Phi$$

$$\begin{bmatrix}\rho_1\\ \rho_2\\ \rho_3\\ \vdots\\ \rho_p\end{bmatrix} = \begin{bmatrix}\rho_0 & \rho_{-1} & \rho_{-2} & \cdots & \rho_{1-p}\\ \rho_1 & \rho_0 & \rho_{-1} & \cdots & \rho_{2-p}\\ \rho_2 & \rho_1 & \rho_0 & \cdots & \rho_{3-p}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \rho_{p-1} & \rho_{p-2} & \rho_{p-3} & \cdots & \rho_0\end{bmatrix}\begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \vdots\\ \phi_p\end{bmatrix} \tag{A.1.19}$$

where $\rho_0 = 1$ and $\rho_{p-i} = \rho_{i-p}$.

Note that $\mathbf{R}$ is full-rank and symmetric thus it is invertible, giving a system of equations for the Auto-regression parameters Φ,

$$\Phi = \mathbf{R}^{-1}\mathbf{r} \tag{A.1.20}$$

Appendix B

Datasets

B.1 2011 Google Cluster Dataset

In the document 'Google cluster-usage traces: format & schema' [53] the authors describe how resource usage data was captured. Measurements are taken at 1 second intervals and then aggregated the values per task, reported every 300 seconds (or 5 minutes). The following resources was captured:

1. Mean CPU usage
2. Maximum CPU usage
3. Memory usage
4. Assigned memory
5. Unmapped page cache memory usage
6. Page cache memory usage
7. Maximum memory usage
8. Mean disk I/O
9. Maximum disk I/O
10. Mean local disk space used
11. Cycles per instruction (CPI)
12. Memory accessed per instruction (MAI)

In this work we only use the Mean CPU and Mean Memory usage rates. The Mean CPU usage is measured in CPU-core seconds per second, whereas the Mean Memory usage is measured as the total canonical memory, user accessible pages and page cache excluding stale pages.

B.2 Wikipedia Pageview Dataset

The official description given by the Wikipedia Analytics team, authors of the info-page "Page view statistics for Wikimedia projects" [65] for the Wikipedia Pageview and Network dataset as follows:

> Each page request to a Wikipedia project page, whether it be for editing, reading or for "special pages" is captured and logged by Wikipedia's squid caching hosts, with internal and non-general page views being filtered out.

The Wikimedia projects are abbreviated as follow:

- wikibooks: ".b"
- wiktionary: ".d"
- wikimedia: ".m"
- wikipedia mobile: ".mw"
- wikinews: ".n"
- wikiquote: ".q"
- wikisource: ".s"
- wikiversity: ".v"
- mediawiki: ".w"

The dataset is currently (2015) being maintained by the Wikipedia Analytics team.

Appendix C

Additional Results

C.1 Statistical Significance Test Results

In the Experiment Investigation Chapter, the statistical significance **One-Tailed Test** was used to determine if two forecasting method results a statistically different with a confidence value of 95% (thus a p-value less than 0.05). The test results for the five evaluations performed using the different evaluations metrics, are shown in Table C.1.

Table C.1: Statistical significance test results for the evaluations performed on the 2011 Google Cluster and Wikipedia datasets.

Metric	Resource	Hypothesis	P-value	Significant?
RMSE	CPU	AR(30) < MA(30)	0.0252	Yes
	Memory	AR(30) < MA(30)	0.1888	No
	Pageview	AR(30) < MA(30)	4.20E-280	Yes
	Network	AR(30) < MA(30)	4.43E-16	Yes
Correct Estimation Rate	CPU	MA(30) > FFNN	0.0014	Yes
	Memory	MA(30) > AR(30)	0.3426	No
	Pageview	AR(30) > FFNN	7.14E-13	Yes
	Network	AR(30) > FFNN	3.68E-20	Yes
Estimation Score	CPU	MA(30) < FFNN	1.35E-04	Yes

	Memory	AR(30) < MA(30)	0.2826	No
	Pageview	AR(30) < Markov1	7.19E-42	Yes
	Network	AR(30) < Markov1	1.84E-34	Yes
Overload Likelihood Ratio	CPU	AR(30) > MA(30)	1.20E-08	Yes
	Memory	AR(30) > MA(30)	0.0386	Yes
	Pageview	AR(30) > FFNN	1.25E-10	Yes
	Network	AR(30) > FFNN	6.25E-05	Yes
Overloaded State Likelihood Ratio	CPU	RNN > HW	3.85E-17	Yes
	Memory	FFNN > HW	0.0018	Yes
	Pageview	AR(30) > HW	7.01E-12	Yes
	Network	AR(30) > HW	2.08E-09	Yes

C.2 Ensemble models: Statistical Significance Test Results

In Section 5.8 we investigated the possible accuracy gain when combining four forecasting methods, each addressing an aspect of workloads typically presented to cloud hosted resources. In Table C.2, we present the statistical significance test results, when comparing each ensemble model with the best performing forecasting method on the specific evaluation metric and data type (either CPU or Memory usage data).

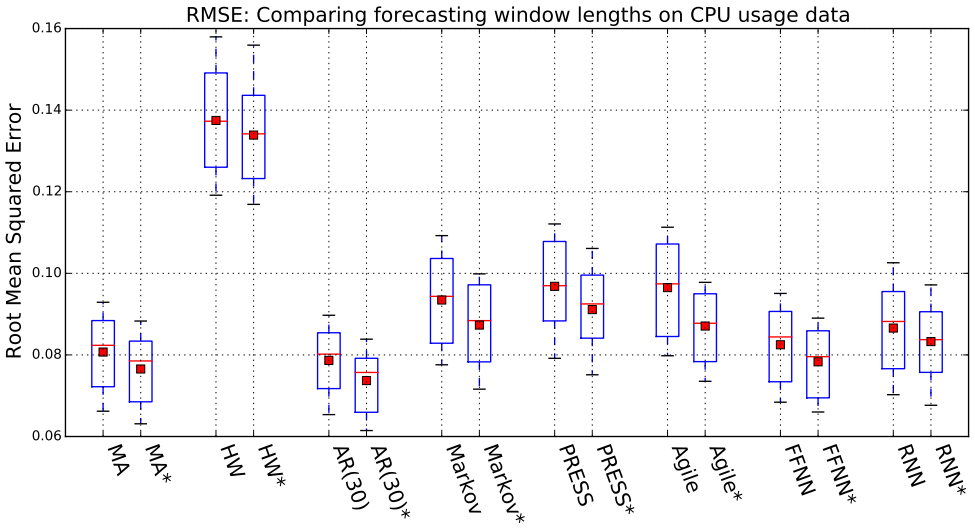
Table C.2: The statistical significance test results for comparing the ensemble models.

Metric & Data type	**Best Method**	**Hypothesis**	**P-value**	**Significant?**
RMSE on CPU data	AR(30)	< Average Model	0.0045	Yes
		< WA Model	1.75E-23	Yes
		< Combo Model	1.32E-35	Yes

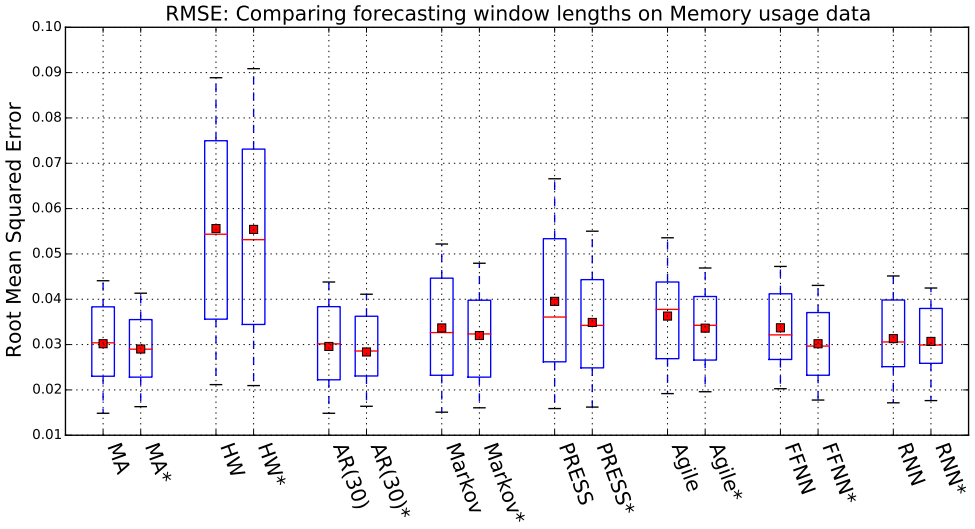
RMSE on Memory	AR(30)	< Average Model	1.13E-06	Yes
		< WA Model	0.0909	No
		< Combo Model	4.45E-04	Yes
Correct Estimation Rate on CPU data	MA(30)	> Average Model	1.73E-06	Yes
		> WA Model	3.25E-30	Yes
		> Combo Model	8.84E-40	Yes
Correct Estimation Rate on Memory data	Agile	> Average Model	1.47E-19	Yes
		> WA Model	5.95E-05	Yes
		> Combo Model	0.0213305	Yes
Estimation Score on CPU data	MA(30)	< Average Model	4.31E-13	Yes
		< WA Model	2.97E-32	Yes
		< Combo Model	6.52E-59	Yes
Estimation Score on Memory data	AR(30)	< Average Model	2.54E-06	Yes
		< WA Model	3.90E-23	Yes
		< Combo Model	2.25E-29	Yes
OLR on CPU data	AR(30)	> Average Model	2.12E-15	Yes
		> WA Model	8.76E-31	Yes
		> Combo Model	1.22E-33	Yes
OLR on Memory data	AR(30)	> Average Model	5.61E-15	Yes
		> WA Model	4.03E-44	Yes
		> Combo Model	8.62E-50	Yes
OSLR on CPU data	FFNN	> Average Model	5.29E-36	Yes
		> WA Model	5.66E-06	Yes
		> Combo Model	0.0061	Yes
OSLR on Memory	FFNN	> Average Model	2.48E-22	Yes
		> WA Model	0.0040	Yes
		> Combo Model	0.0055	Yes

C.3 Investigate Shorter Forecasting Window

The results for the investigation in to using a shorter forecasting window of 15 samples (denoted by *), are shown in Figures C.1, C.2, C.4 and C.5. The statistical significance test results are shown in Table C.3.

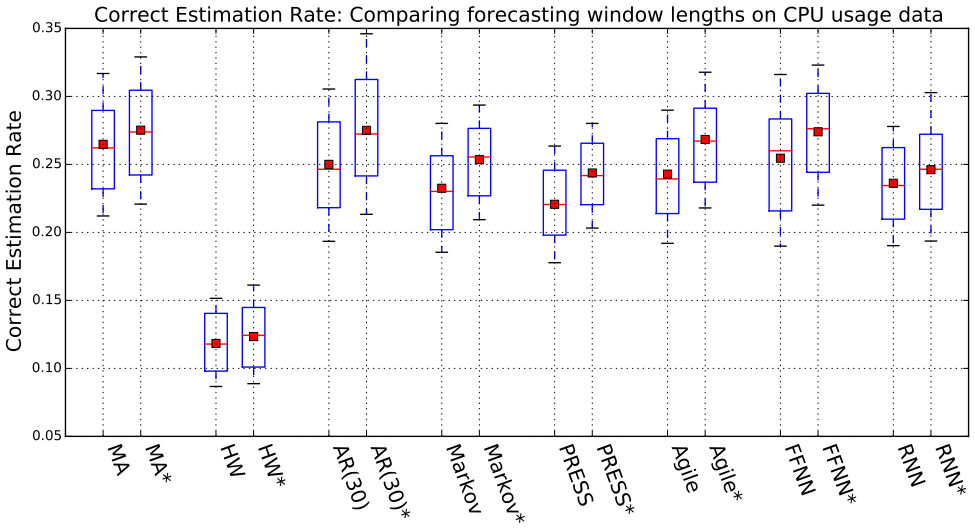


(a) CPU usage data.

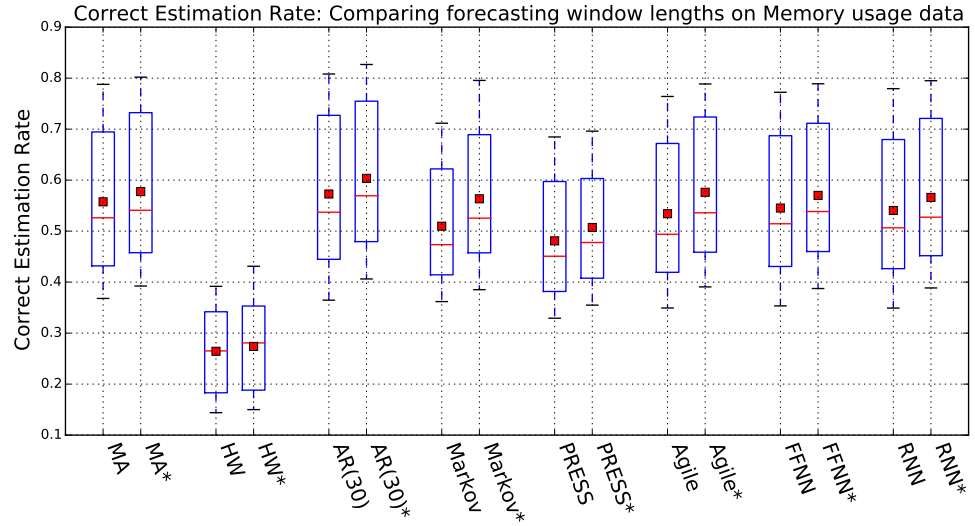


(b) Memory usage data.

Figure C.1: Comparing forecasting window length of 30 samples to a window length of 15 samples (denoted by *) on Root Mean Squared Error.



(a) CPU usage data.



(b) Memory usage data.

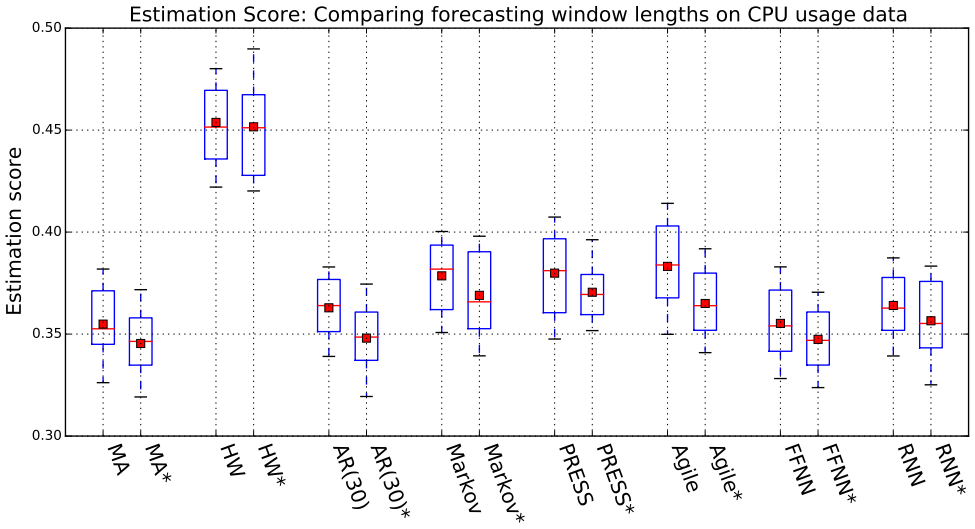
Figure C.2: Comparing forecasting window length of 30 samples to a window length of 15 samples (denoted by *) on Correct Estimation Rate.

Table C.3: The statistical significance test results for the investigation into using a shorter forecasting window length.

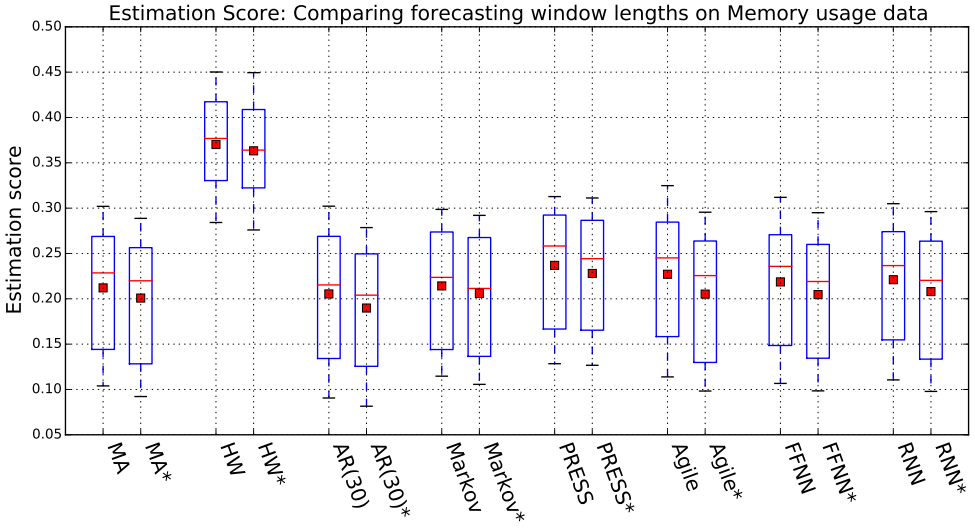
Metric & Data type	**Hypothesis**	**P-value**	**Significant?**
RMSE on CPU data	MA(30)* $<$ MA(30)	1.85E-03	Yes
	HW* $<$ HW	4.09E-02	Yes

	AR(30)* < AR(30)	3.79E-05	Yes
	Markov* < Markov	1.95E-04	Yes
	PRESS* < PRESS	5.48E-04	Yes
	Agile* < Agile	1.47E-08	Yes
	FFNN* < FFNN	1.51E-03	Yes
	RNN* < RNN	2.11E-02	Yes
RMSE on Memory data	MA(30)* < MA(30)	0.208263	No
	HW* < HW	0.479556	No
	AR(30)* < AR(30)	0.196025	No
	Markov* < Markov	0.173971	No
	PRESS* < PRESS	0.025158	Yes
	Agile* < Agile	0.049637	Yes
	FFNN* < FFNN	0.00743	Yes
	RNN* < RNN	0.322387	No
Correct Est Rate on CPU data	MA(30)* > MA(30)	3.84E-02	Yes
	HW* > HW	8.66E-02	No
	AR(30)* > AR(30)	5.21E-05	Yes
	Markov* > Markov	2.83E-05	Yes
	PRESS* > PRESS	8.09E-07	Yes
	Agile* > Agile	4.86E-06	Yes
	FFNN* > FFNN	9.73E-04	Yes
	RNN* > RNN	3.21E-02	Yes
Correct Est Rate on Memory data	MA(30)* > MA(30)	1.82E-01	No
	HW* > HW	2.60E-01	No
	AR(30)* > AR(30)	9.24E-02	No
	Markov* > Markov	4.02E-03	Yes
	PRESS* > PRESS	7.80E-02	No
	Agile* > Agile	2.83E-02	Yes
	FFNN* > FFNN	1.22E-01	No
	RNN* > RNN	1.19E-01	No
Estimation score on CPU data	MA(30)* < MA(30)	6.44E-04	Yes
	HW* < HW	2.72E-01	No
	AR(30)* < AR(30)	3.73E-08	Yes
	Markov* < Markov	1.56E-03	Yes
	PRESS* < PRESS	4.00E-04	Yes
	Agile* < Agile	2.86E-08	Yes
	FFNN* < FFNN	3.12E-03	Yes
	RNN* < RNN	6.43E-03	Yes

Estimation score on Memory data	MA(30)* < MA(30)	1.47E-01	No
	HW* < HW	2.26E-01	No
	AR(30)* < AR(30)	7.36E-02	No
	Markov* < Markov	2.09E-01	No
	PRESS* < PRESS	1.84E-01	No
	Agile* < Agile	2.48E-02	Yes
	FFNN* < FFNN	9.07E-02	No
	RNN* < RNN	1.02E-01	No
OLR on CPU data	MA(30)* > MA(30)	1.05E-04	Yes
	HW* > HW	4.54E-03	Yes
	AR(30)* > AR(30)	1.77E-03	Yes
	Markov* > Markov	2.26E-07	Yes
	PRESS* > PRESS	6.01E-03	Yes
	Agile* > Agile	1.32E-05	Yes
	FFNN* > FFNN	1.40E-05	Yes
	RNN* > RNN	1.77E-03	Yes
OLR on Memory data	MA(30)* > MA(30)	2.35E-02	Yes
	HW* > HW	1.25E-02	Yes
	AR(30)* > AR(30)	1.34E-03	Yes
	Markov* > Markov	1.87E-04	Yes
	PRESS* > PRESS	2.65E-03	Yes
	Agile* > Agile	2.63E-03	Yes
	FFNN* > FFNN	2.04E-02	Yes
	RNN* > RNN	1.53E-02	Yes
OSLR on CPU data	MA(30)* > MA(30)	2.65E-01	No
	HW* > HW	6.69E-06	Yes
	AR(30)* > AR(30)	2.08E-08	Yes
	Markov* > Markov	2.36E-30	Yes
	PRESS* > PRESS	3.19E-20	Yes
	Agile* > Agile	5.02E-31	Yes
	FFNN* > FFNN	3.66E-13	Yes
	RNN* > RNN	5.47E-02	No
OSLR on Memory data	MA(30)* > MA(30)	3.17E-02	Yes
	HW* > HW	5.68E-07	Yes
	AR(30)* > AR(30)	7.19E-14	Yes
	Markov* > Markov	8.30E-32	Yes
	PRESS* > PRESS	6.85E-26	Yes
	Agile* > Agile	2.18E-26	Yes
	FFNN* > FFNN	1.88E-09	Yes
	RNN* > RNN	8.65E-09	Yes

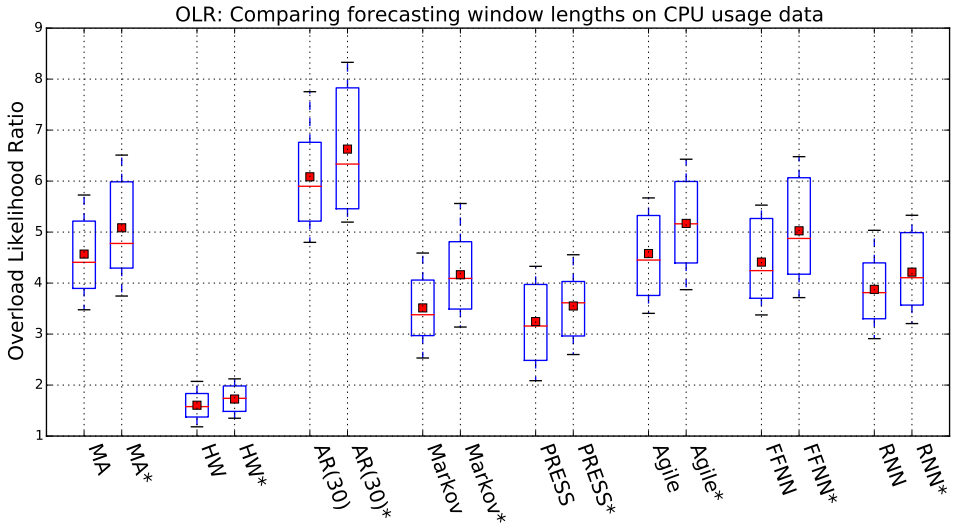


(a) CPU usage data.

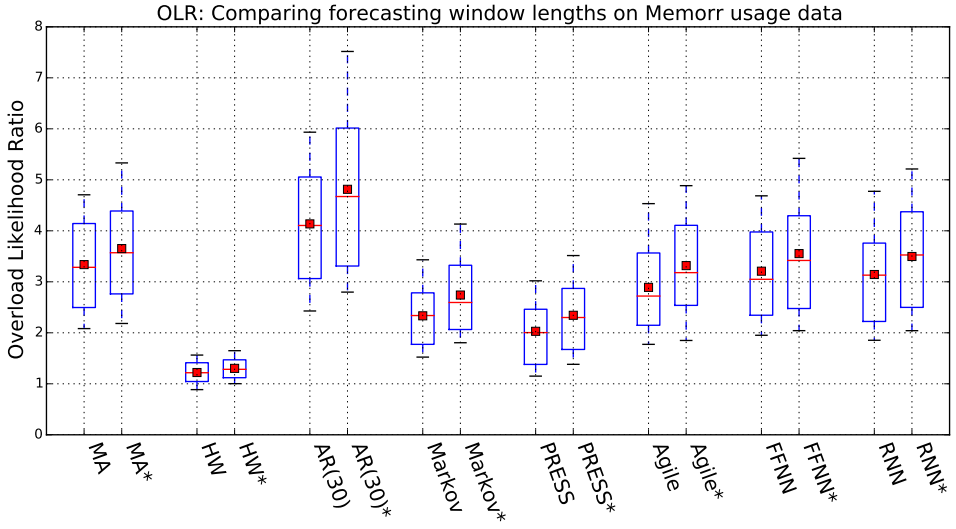


(b) Memory usage data.

Figure C.3: Comparing forecasting window length of 30 samples to a window length of 15 samples (denoted by *) on Estimation score.

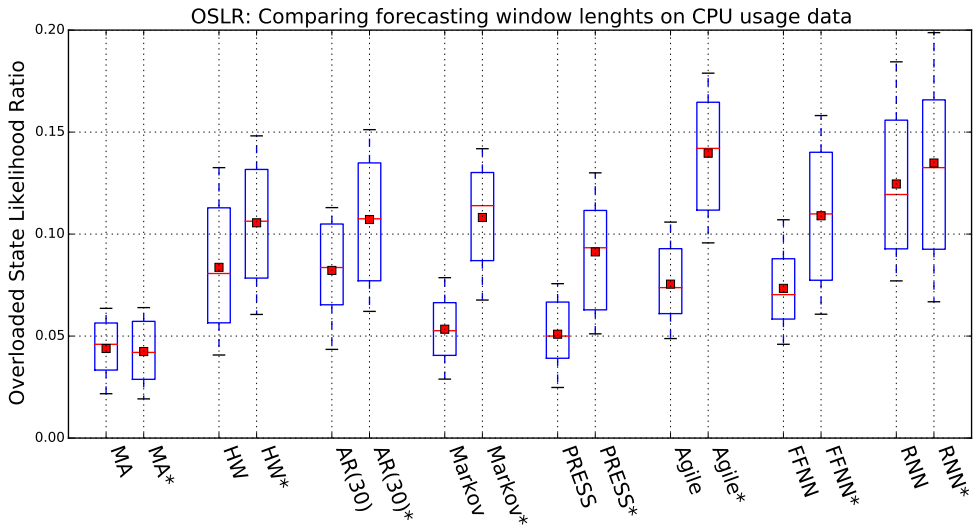


(a) CPU usage data.

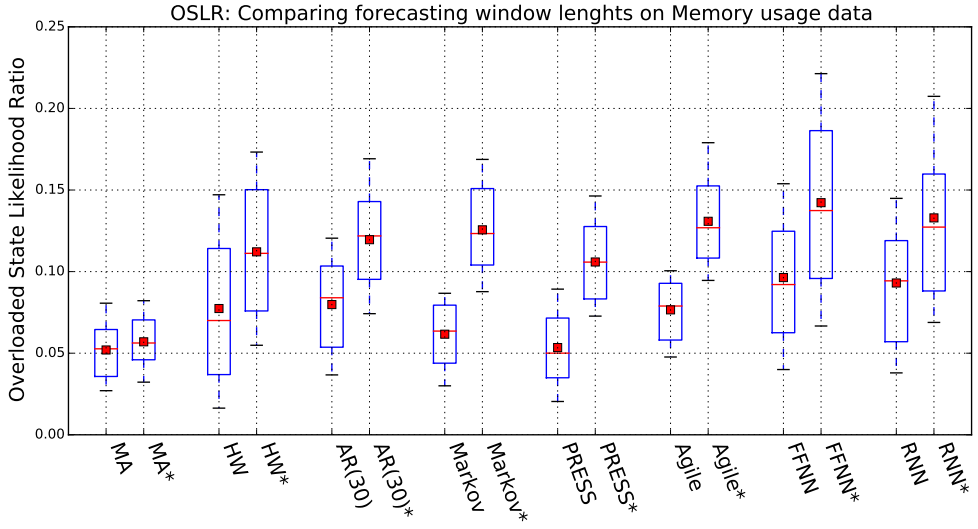


(b) Memory usage data.

Figure C.4: Comparing forecasting window length of 30 samples to a window length of 15 samples (denoted by *) on Overload Likelihood Ratio.



(a) CPU usage data.



(b) Memory usage data.

Figure C.5: Comparing forecasting window length of 30 samples to a window length of 15 samples (denoted by *) on Overload State Likelihood Ratio.

Bibliography

[1] R. K. Agrawal and R. Adhikari, "An Introductory Study on Time Series Modeling and Forecasting," *CoRR*, 2013.

[2] Amazon, "Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting." [Online]. Available: http://aws.amazon.com/ec2/

[3] A. Amies, G. N. Liu, H. Sluiman, and Q. G. Tong, *Developing and Hosting Applications on the Cloud*. IBM Press, 2012. [Online]. Available: http://www.ibmpressbooks.com/bookstore/product.asp?isbn=9780133066845

[4] M. Arlitt and T. Jin, "A workload characterization study of the 1998 World Cup Web site," *IEEE Network*, vol. 14, no. 3, 2000.

[5] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, p. 50, 2010.

[6] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[7] A. Boardman, F. S. Schlindwein, A. P. Rocha, and A. Leite, "A study on the optimum order of autoregressive models for heart rate variability." *Physiological measurement*, vol. 23, pp. 325–336, 2002.

[8] R. G. Brown, *Smoothing, Forecasting and Predicition of Discrete Time Series*, ser. Dover Phoenix Editions. Dover Publications, 1963. [Online]. Available: https://books.google.co.za/books?id=XXFNW_QaJYgC

[9] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and computation: practice ...*, pp. 1–37, 2002.

[10] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for vound constrained optimization," *Society of Industrial and Applied Mathematics*, vol. 16, no. 5, pp. 1190–1208, 1995.

[11] F. Caglar and A. Gokhale, "iOverbook: Intelligent Resource-Overbooking to Support Soft Real-time Applications in the Cloud," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, 2014. [Online]. Available: http://www.dre.vanderbilt.edu/~gokhale/WWW/papers/CLOUD-2014.pdf

[12] J. L. H. Carvalho, A. F. Rocha, I. dos Santos, C. Itiki, L. F. Junqueira, and F. A. O. Nascimento, "Study on the optimal order for the auto-regressive time-frequency analysis of heart rate variability," *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No.03CH37439)*, pp. 2621–2624, 2003.

[13] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '03*, p. 300, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=781027.781067

[14] M. Craven, "Markov Chain Models (Part 1)," p. 8, 2011. [Online]. Available: https://www.biostat.wisc.edu/bmi576/lectures/markov-chains-1.pdf

[15] C. Croarkin and P. Tobias, *NIST/SEMATECH e-handbook of statistical methods*, 2014, vol. 1. [Online]. Available: http://www.itl.nist.gov/div898/handbook/

[16] Dropbox, "Dropbox." [Online]. Available: https://www.dropbox.com/?

[17] R. Durbin, S. Eddy, and A. S. Krogh, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[18] T. Dutoit and F. Marqués, *Applied Signal Processing.* Boston, MA: Springer US, 2009. [Online]. Available: http://link.springer.com/10.1007/978-0-387-74535-0

[19] M. Ebell and H. Barry, "Likelihood Ratios Part 1: Introduction," 2008. [Online]. Available: http://omerad.msu.edu/ebm/Diagnosis/Diagnosis6.html

[20] J. L. Elman, "Finidng structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1402_1/abstracthttp://doi.wiley.com/10.1207/s15516709cog1402_1

[21] G. Eshel, "The yule walker equations for the AR coefficients," *Internet resource*, pp. 1–8, 2003. [Online]. Available: http://www.stat.sc.edu/~vesselin/STAT520_YW.pdf

[22] N. R. Farnum and L. W. Stanton, *Quantitative Forecasting Methods*, ser. Duxbury series in statistics and decision sciences. PWS-Kent Pub. Co., 1990, vol. 41. [Online]. Available: http://books.google.co.za/books?id=9AWYmbMLYmUC

[23] M. Gerolimetto, "Autocorrelation function analysis: Theorethical autocorrelation function," 2010.

[24] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic reSource Scaling for cloud systems," in *Proceedings of the 2010 International Conference on Network and Service Management, CNSM 2010*. Ieee, Oct. 2010, pp. 9–16. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5691343

[25] Google, "Gmail." [Online]. Available: mail.google.com

[26] ——, "Google App Engine." [Online]. Available: https://developers.google.com/appengine/

[27] M. Hamdaqa, T. Livogiannis, and L. Tahvildari, "A Reference Model for Developing Cloud Applications," *CLOSER*, 2011. [Online]. Available: http://stargroup.uwaterloo.ca/~mhamdaqa/publications/AREFERENCEMODELFORDEVELOPINGCLOUDAPPLICATIONS.pdf

[28] J. L. Hellerstein, "Google cluster data," Google research blog, Jan. 2010. [Online]. Available: http://googleresearch.blogspot.com/2010/01/google-cluster-data.html

[29] J. Huang, C. Li, and J. Yu, "Resource prediction based on double exponential smoothing in cloud computing," *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pp. 2056–2060, 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6201461

[30] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2013. [Online]. Available: http://otexts.com/fpp/

[31] R. J. Hyndman, "Forecasting using Exponential smoothing methods," 2013. [Online]. Available: http://robjhyndman.com/talks/RevolutionR/5-ExponentialSmoothing.pdf

[32] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[33] R. Illsley and Rackspace, "The Role of IT Principles in IT Governance," 2014. [Online]. Available: http://www.rackspace.co.uk/sites/default/files/UnlockedNov2014_TheRoleOfCloudInITModernisation_Ovum.pdf

[34] B. Jennings and R. Stadler, "Resource Management in Clouds: Survey and Research Challenges," *Journal of Network and Systems Management*, Mar. 2014. [Online]. Available: http://link.springer.com/10.1007/s10922-014-9307-7

[35] M. I. Jordan, "Serial Order: A Parallel, Distributed Processing Approach," *Advances in psychology*, vol. 121, no. 8604, pp. 471–495, 1986.

[36] P. Kalekar, "Time series forecasting using Holt-Winters exponential smoothing," *Kanwal Rekhi School of Information Technology*, no. 04329008, pp. 1–13, 2004. [Online]. Available: http://www.it.iitb.ac.in/~praj/acads/seminar/04329008_ExponentialSmoothing.pdf

[37] D. Kriesel, "A Brief Introduction to Neural Networks," 2005. [Online]. Available: http://www.dkriesel.com/en/science/neural_networkshttp://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf

[38] J. Kupferman, J. Silverman, P. Jara, and J. Browne, "Scaling into the cloud," *CS270-Advanced Operating Systems*, pp. 1–8, 2009. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Scaling+Into+The+Cloud#0

[39] S. L. S. Lili, Y. S. Y. Shoubao, G. L. G. Liangmin, and W. B. W. Bin, "A Markov Chain Based Resource Prediction in Computational Grid," *2009 Fourth International Conference on Frontier of Computer Science and Technology*, no. 60673172, pp. 0–5, 2009.

[40] T. Liu, "Application of Markov chains to analyze and predict the time series," *Modern Applied Science*, pp. 508–511, 2010. [Online]. Available: http://www.ccsenet.org/journal/index.php/mas/article/view/6040

[41] Z. Liu and S. Cho, "Characterizing machines and workloads on a Google cluster," *Proceedings of the International Conference on Parallel Processing Workshops*, pp. 397–403, 2012.

[42] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling Techniques for Elastic Applications in Cloud Environments," *Technical Report: University of the Basque Country*, pp. 11 – 14, 2012.

[43] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12, 2011.

[44] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST, Tech. Rep., 2009. [Online]. Available: http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf

[45] V. Nae, A. Iosup, and R. Prodan, "Dynamic Resource Provisioning in Massively Multiplayer Online Games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 380–395, Mar. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5444882

[46] R. Nau, "Moving average and exponential smoothing models." [Online]. Available: http://people.duke.edu/~rnau/411avg.htm#SMA

[47] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: elastic distributed resource scaling for Infrastructure-as-a-Service," *10th International Conference on Autonomic Computing (ICAC '13)*, p. 14, 2013. [Online]. Available: http://dance.csc.ncsu.edu/papers/icac2013.pdfhttp://cairo.csc.ncsu.edu/icac13/main_20130306120721_v2.pdf

[48] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/index.html

[49] A. Papoulis and S. U. Pillai, *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 1991.

[50] J. Perktold, "StatsModels: Statistics in Python." [Online]. Available: http://statsmodels.sourceforge.net/

[51] R. Prodan and V. Nae, "Prediction-based real-time resource provisioning for massively multiplayer online games," *Future Generation Computer Systems*, vol. 25, no. 7, pp. 785–793, Jul. 2009. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167739X08001933

[52] A. Queiroz, "Implementation of Holt-Winters algorithms." [Online]. Available: https://gist.github.com/andrequeiroz/5888967

[53] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., ...*, pp. 1–14, 2011. [Online]. Available: http://googleclusterdata.googlecode.com/files/Googlecluster-usagetraces-format+schema(2011.10.27external).pdf

[54] RightScale, "RightScale," 2015. [Online]. Available: http://www.rightscale.com/home-v1?utm_expid=99127306-75.HRZOZscaQUmmbp3-FlBRkg.1

[55] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[56] P. Rouanet, "dtw 1.0 : Dynamic Time Warping Python Module." [Online]. Available: https://github.com/pierre-rouanet/dtw

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[58] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "PyBrain," *Journal of Machine Learning Research*, 2010.

[59] Scikit-learn, "3.3. Model evaluation: quantifying the quality of predictions." [Online]. Available: http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

[60] SciPy.org, "SciPy.org." [Online]. Available: http://www.scipy.org/

[61] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: elastic resource scaling for multi-tenant cloud systems," *Proceedings of the 2nd Symposium on Cloud Computing*, pp. 5:1—-5:14, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2038921

[62] B. Urgaonkar and P. Shenoy, "Dynamic provisioning of multi-tier internet applications," *... Computing, 2005. ICAC ...*, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1498066

[63] G. Walker, "On periodicity in series of related terms," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, pp. 518–532, 1931.

[64] F. Wasilewski, "PyWavelets - Discrete Wavelet Transform in Python." [Online]. Available: http://www.pybytes.com/pywavelets/#

[65] Wikimedia Analytics, "Page view statistics for Wikimedia projects." [Online]. Available: http://dumps.wikimedia.org/other/pagecounts-raw/

[66] ——, "Wikistats: Wikimedia Statistics." [Online]. Available: http://stats.wikimedia.org/

[67] J. Wilkes, "More Google cluster data," Google research blog, Nov. 2011. [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html

[68] P. R. Winters, "Forecasting Sales by Exponentially Weighted Moving Averages," *Management Science*, vol. 6, no. 3, pp. 324–342, 1960. [Online]. Available: http://dx.doi.org/10.1287/mnsc.6.3.324

[69] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, Apr. 2010. [Online]. Available: http://www.springerlink.com/index/10.1007/s13174-010-0007-6