Scalable and Accurate Forecasting for Smart Cities

Pasan Manura Karunaratne

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

August 2018
Abstract

Cities are getting bigger, better and smarter. The increased connectivity of people and devices and the availability of cheap sensors has led to a surge in public and government interest in smart city initiatives. This public interest, along with the recent increased interest in machine learning techniques has led to growing research focus into the mining and analysis of data in smart city settings.

Much of the analysis in smart city settings is based on forecasting on time series data recorded by smart sensors for planning purposes. For example, utility companies can use electricity load forecasting on smart meter data for capacity planning, and prediction of pedestrian counts and passenger flow in public transportation systems can help in planning to reduce traffic congestion. Though forecasting in smart city settings yields such benefits, it also entails unique challenges, such as challenges related to multi-step prediction, challenges related to low quality training data due to sensors encountering vandalism, malfunction or communication failures, and challenges in maintaining predictive throughput in systems involving increasingly larger numbers of smart sensors.

Improving accuracy is a primary goal in any forecasting task, which is especially challenging in multi-step prediction scenarios. We address this challenge by providing new methods to incorporate prior knowledge uniquely relevant to smart cities, such as the periodic behaviour of sensor time series data over the Monday-Friday working week. Specifically, we propose novel kernel function compositions which can incorporate such prior knowledge to kernel-based Bayesian forecasting techniques, with the goal of improving prediction accuracy and robustness to spurious data.

We develop our kernel compositions for the state of the art Gaussian Process Regression technique. The new kernel compositions we develop enable prior knowl-
edge relating to multiple periodic effects of the working week (e.g. daily, weekly, holiday effects) and their interactions to be incorporated in the same model. We also provide methods to mitigate the effects of convergence to local optima in the optimisation process over the hyperparameters used in the Gaussian Process models.

We address the challenges relating to missing training data in smart city settings by making use of data of other related sensors (which may have more complete data) to mitigate the impact the low quality data has on prediction accuracy. To this end, we develop multi-task learning methods (which are able to learn joint representations from multiple sensors) to improve Gaussian Process Regression prediction accuracy with missing training data values. We also provide equivalent expressions to our multi-task learning methods as combinations of commonly used kernel functions in Gaussian Processes. This enables the straightforward implementation of these methods in popular machine learning toolkits.

We address the scalability challenge of large volumes of sensor data in two steps. One, we focus on an interpretable label-based forecasting algorithm which allows for high-throughput predictions due to the minimal number of operations needed to be done in the forecasting stage. We perform numerous enhancements on this algorithm in order to improve its prediction accuracy, including filtering, windowing and ensembling methods as well as methods of incorporating exogenous variables.

Our scalable forecasting methods are then developed using this enhanced base algorithm. We develop methods which enable the initial step of the algorithm to be performed using algorithms developed for stream processing, which not only allows for the algorithm to be parallelised across multiple machines, but also enables it to run on real-time data streams. We address the scalability challenges in scenarios with both a single fast stream and a large number of streams, especially with regard to synchronisation issues between multiple machines.

We demonstrate the effectiveness of our methods on multiple real-world publicly-available datasets to illustrate the potential generalisability of our techniques.
Declaration

This is to certify that

1. The thesis comprises only my original work towards the PhD,

2. Due acknowledgement has been made in the text to all other material used,

3. The thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices

Pasan Manura Karunaratne, August 2018
Acknowledgments

First of all I would like to thank my supervisors, especially my principal supervisor Prof Shanika Karunasekera for her constant guidance in helping me develop as a researcher and for being very supportive throughout my PhD candidature. I would also like to thank my co-supervisors Dr Aaron Harwood and Dr Masud Moshtaghi for their continuous feedback and research ideas, as well as my Committee Chair Prof James Bailey for his support and advice. I am also very thankful to A/Prof Trevor Cohn for the many fruitful discussions and guidance regarding Gaussian Processes.

I am also grateful to the subject coordinators who gave me the opportunity to work as a tutor in the department, including Dr Masud Moshtaghi, A/Prof Udaya Parampalli, A/Prof Michael Kirley, Dr Chien Aun Chan, Dr Chris Culnane, Mr Jeremy Nicholson and Prof Rao Kotagiri, and to A/Prof Colin Burvill for giving me the opportunity to work as a mentor to postgraduate students undertaking internships. I would also like to express my gratitude to the Big Data team at Telstra Corporation for providing me the opportunity to undertake an internship, and to the Australian Mathematical Sciences Institute (AMSI) for facilitating the internship. The financial support of the IEEE Communications Society and IEEE Big Data 2017 in the form of travel grants is also much appreciated.

Pursuing a PhD would not be exciting if there were no one to share experiences with. I would like to thank all my friends in the department, with special mention to Sameendra Samarawickrama, Gitansh Khirbat, Doan Minh Tuan, Han Li, and Nicholas Downing for making my stay a very enjoyable one.
A PhD is a long journey of persistence, of both triumphs and setbacks. I am eternally grateful to my parents for supporting me during this journey, for being there at the best of times and at the worst. To them I dedicate this thesis.
List of Publications

This thesis is composed of four main core chapters which are based on the following four published or accepted papers. I am the main author of these papers and have contributed more than 50%.

1. Pasan Karunaratne, Masud Moshtaghi, Shanika Karunasekera, Aaron Harwood, Trevor Cohn. ‘Modelling the Working Week for Multi-Step Forecasting using Gaussian Process Regression’, International Joint Conference on Artificial Intelligence (IJCAI) 2017

CORE Rank A* (https://goo.gl/kPiuRf)*


Not listed on CORE

3. Pasan Karunaratne, Masud Moshtaghi, Shanika Karunasekera, Aaron Harwood. ‘PSF+ - Fast and Improved Electricity Consumption Prediction in Campus Environments’, 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)

*CORE (Computing Research and Education Association) provides a ranking of publication venues ranging from A* (flagship conference), A (excellent conference), B (good conference), and C (other).
Not listed on CORE


CORE Rank A* (https://goo.gl/CY3KdL)
To my parents
## Contents

1. **Introduction**  
   1.1 Background ................................................. 1  
   1.2 Problem Statement ........................................... 5  
   1.3 Organisation of Thesis ................................. 11  

2. **Literature Review**  
   2.1 Introduction ................................................ 16  
   2.2 Time Series Forecasting Algorithms ......................... 17  
   2.3 Bayesian Forecasting ....................................... 21  
   2.4 Label-based Forecasting ................................... 30  
   2.5 Scalable Forecasting ........................................ 35  
   2.6 Conclusions .................................................. 41  

3. **Bayesian Forecasting: Kernel Compositions to Improve Accuracy**  
   3.1 Introduction to Gaussian Process Regression .................. 51  
   3.2 Methodology ................................................. 55  
   3.3 Experiments and Results ..................................... 64  
   3.4 Conclusions .................................................. 72  

4. **Bayesian Forecasting: Improved Accuracy with Low-quality Data**  
   xi
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Gaussian Processes - Multi-task Learning</td>
<td>79</td>
</tr>
<tr>
<td>4.2 Methodology</td>
<td>83</td>
</tr>
<tr>
<td>4.3 Experiments</td>
<td>88</td>
</tr>
<tr>
<td>4.4 Results</td>
<td>95</td>
</tr>
<tr>
<td>4.5 Conclusions</td>
<td>103</td>
</tr>
<tr>
<td>5 Label-based Forecasting</td>
<td>107</td>
</tr>
<tr>
<td>5.1 The PSF Algorithm</td>
<td>112</td>
</tr>
<tr>
<td>5.2 Proposed Technique</td>
<td>115</td>
</tr>
<tr>
<td>5.3 Experiments and Results</td>
<td>121</td>
</tr>
<tr>
<td>5.4 Conclusions</td>
<td>131</td>
</tr>
<tr>
<td>6 Scalable Label-based Forecasting</td>
<td>134</td>
</tr>
<tr>
<td>6.1 Background - The CluStream Algorithm</td>
<td>139</td>
</tr>
<tr>
<td>6.2 Scenario I - Multiple Stream Scalability</td>
<td>146</td>
</tr>
<tr>
<td>6.3 Scenario II - Single Stream Scalability</td>
<td>153</td>
</tr>
<tr>
<td>6.4 Conclusions</td>
<td>178</td>
</tr>
<tr>
<td>7 Conclusions</td>
<td>182</td>
</tr>
<tr>
<td>7.1 Kernel compositions to incorporate prior knowledge</td>
<td>183</td>
</tr>
<tr>
<td>7.2 Multi-task learning for robust predictions</td>
<td>185</td>
</tr>
<tr>
<td>7.3 Label-based forecasting</td>
<td>187</td>
</tr>
<tr>
<td>7.4 Scalable label-based forecasting</td>
<td>189</td>
</tr>
<tr>
<td>7.5 Limitations and Future Work</td>
<td>191</td>
</tr>
</tbody>
</table>

References 217
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Campus Smart Meter - Output Variance Hyperparameter</td>
<td>68</td>
</tr>
<tr>
<td>3.2</td>
<td>Pedestrian - Output Variance Hyperparameter</td>
<td>68</td>
</tr>
<tr>
<td>3.3</td>
<td>Campus Smart Meter - Noise Hyperparameter</td>
<td>70</td>
</tr>
<tr>
<td>3.4</td>
<td>Pedestrian - Noise Hyperparameter</td>
<td>70</td>
</tr>
<tr>
<td>4.1</td>
<td>Kronecker product between $B$ and $k_{data}$</td>
<td>86</td>
</tr>
<tr>
<td>4.2</td>
<td>Markov chain error generation process</td>
<td>93</td>
</tr>
<tr>
<td>4.3</td>
<td>Example run of bursty and random errors</td>
<td>94</td>
</tr>
<tr>
<td>4.4</td>
<td>Smart meter per sensor data comparison of errors</td>
<td>97</td>
</tr>
<tr>
<td>4.5</td>
<td>Pedestrian data per sensor comparison of errors</td>
<td>100</td>
</tr>
<tr>
<td>4.6</td>
<td>Multi-task GPR - % improvement over next best techniques with increasing burst length (smart meter data)</td>
<td>100</td>
</tr>
<tr>
<td>4.7</td>
<td>Multi-task GPR - % improvement over next best techniques with increasing burst length (pedestrian data)</td>
<td>100</td>
</tr>
<tr>
<td>4.8</td>
<td>Parameter sensitivity - error rate with changing $k$</td>
<td>101</td>
</tr>
</tbody>
</table>
6.14 Throughput with number of micro-clusters ............... 176
6.15 Throughput with parallelism ................................ 176
6.16 Throughput with synchronisation period ................... 176
List of Tables

3.1 Initial hyperparameter values chosen ............................. 59
3.2 Error rates for different kernel combinations .................... 68
4.1 No data removal results ............................................. 95
4.2 Data removal results ................................................ 96
5.1 Error Rate Comparison with Other Methods .................... 126
5.2 Percentage improvements over PSF ............................... 127
5.3 Computational Efficiency of Techniques ......................... 130
1

Introduction

1.1 Background

Cities are getting bigger, better and smarter. The increased connectivity of people and devices has lead to a surge in public and government interest in smart city initiatives. For example, the Government of Singapore recently launched a smart city
initiative which aims to address the urban challenges of aging population, urban density and energy sustainability [63]. Among many other similar initiatives, notable programs have been launched in New York City [75], Dublin [38] and Melbourne [14]. This public interest, along with the recent increased interest in machine learning techniques has lead to growing research focus into the mining and analysis of data in smart city settings. [43, 53, 66, 91].

Much of the data collected in smart city settings is time-series data recorded by sensors. Urban management can benefit from predicting the future values of these time series. For example, for utility companies, electricity load forecasting on smart meter data enables capacity planning, including optimal commitment of generators, purchasing power from energy markets and planning for peak daily power demand. Electricity load forecasting would also enable analysis in retrospect, especially in relation to anomalies seen in actual electricity consumption. Prediction on traffic conditions, pedestrian counts and passenger flow in public transportation systems can help in planning to reduce traffic congestion. Forecasting related to the taxi and ride-sharing domains can also help in this regard. Further examples include forecasting of water supply needs which can be used to optimise operations of pumping stations and to determine doses of disinfectants needed, and forecasting of particulate matter in the atmosphere which can help in planning to reduce environmental pollution.
Therefore, with regard to time-series data, forecasting is widely used, and is arguably the canonical machine learning technique applied on such data. However, despite the potential real-world benefits of forecasting, it remains an inherently difficult task. Machine learning is an exercise in generalisation on training data; in forecasting this generalisation is made into an uncertain future. In addition to this challenge of extrapolating into the future, forecasting in smart city settings presents further challenges. Many of these challenges in smart city settings relate to the real-world implications of a large number of sensors being deployed across a sprawling metropolis.

One of the main challenges associated with forecasting in smart city settings is the quality of data; smart city sensor data in most cases is low quality data. For example, the sensors deployed in smart city settings may encounter vandalism, or they may malfunction or suffer communication failures. These conditions will result in low quality training data, especially with regard to missing data. Maintaining predictive accuracy even with low quality data is a challenge that needs to be addressed in smart city scenarios.

Another challenge that needs to be addressed is that of predictive throughput. Smart city applications nowadays increasingly involve a large number of smart sensors recording data at fine time granularities. This is especially true given the recent popularity of the Internet of Things and its related applications. Therefore, forecast-
ing algorithms should be able to scale to the vast volume of data generated by a large number of sensors.

We also need to address the challenges in maintaining the primary goal of forecasting accuracy, which is especially important in the real-world such as in smart city scenarios. Inaccurate predictions may lead to either the forecasts not being utilised by decision makers due to being unreliable, or lead to sub-par decisions being made. Accuracy is more challenging to maintain in multi-step prediction, as it is often performed in real-world scenarios due to its increased utility over single-step prediction, and due to the potential for error propagation between steps.

Therefore, we also focus on maintaining predictive accuracy by making use of characteristics of the data that are unique to smart city scenarios. For example, most smart city data relates to human behaviour, especially to how human behaviour varies over the 9-5 working day and Monday to Friday working week. Addressing the challenges involved in incorporating this prior knowledge on human worker behaviour can allow us to address the intrinsic difficulty in forecasting multiple steps into the uncertain future in smart city settings. Though many habits of human behaviour exist, we focus on periodic effects in this work due to it having reasonable parallels with existing models, and since modelling such periodic effects can generalise to other problems that are not purely influenced by human behaviour.
In summary, we note that the forecasting problem in smart city settings needs to be optimised with respect to two different aspects: predictive accuracy and predictive throughput. Inaccurate predictions will not yield the benefits of forecasting, and at worst can be counter-productive. Accurate predictions need to be made, and they need to be made under the realistic setting of low-quality data by utilising all possible prior information. Predictive throughput is also uniquely important in the smart city space as the number of sensors deployed in a metropolis can be significant. Processing the large number of sensor data streaming in will require efficient algorithms which can be horizontally scaled across multiple processing nodes.

This thesis will present work on improving forecasting algorithms with respect to the aspects of accuracy and throughput for smart city scenarios. Chapters 3 and 4 will provide methods to improve accuracy, while Chapters 5 and 6 will focus on throughput.

1.2 Problem Statement

1.2.1 Improving prediction accuracy by incorporating prior data

Forecasting the future accurately is a non-trivial task, especially when performing multi-step prediction in real-world scenarios. One method of improving the accu-
racy of machine learning algorithms is by incorporating prior knowledge to the model.

The incorporation of prior knowledge, of course, is not a novel concept, and is the linchpin of Bayesian learning. However, the translation of expert human intuition to certain effective machine learning models often is far from straightforward. Indeed, there is an abundance of work in the literature where the main contribution is the specification of methods to incorporate prior knowledge to a particular narrow problem domain (e.g. [57] [83] and [92]).

The motivation of incorporating prior knowledge is to improve the accuracy of predictions and their robustness to spurious data. Bayesian prediction techniques allow probabilistic prediction, and kernel-based Bayesian techniques allow the specification of very flexible priors. Therefore, we focus on the hugely popular kernel-based Bayesian method of Gaussian Process Regression, and provide kernel-composition methods to incorporate prior knowledge of working-week effects on smart city time-series data.

We focus on these aspects in Chapter 3 and answer the question:

Q1. How can we improve multi-step prediction accuracy by incorporating prior knowledge about the working week in Gaussian Process Regression for smart city data?
1.2.2 Dealing with low-quality data

Low-quality data is a common problem faced when using machine learning techniques in real-world settings. We focus on this aspect of missing data in smart city data forecasting in order to make our work more applicable to real-world scenarios.

Ideally, any loss of data is identified early on and the root cause is resolved in order to prevent further data loss. Indeed, the author spent a research internship focused on the early detection of low-quality data. Unfortunately, the more likely scenario is that the generation of low-quality data goes unnoticed for significant lengths of time, only to be discovered once analysis and forecasting on the data needs to be performed.

Therefore, we develop methods to mitigate the presence of missing training data values in time-series forecasting. We make use of the fact that in most smart city applications, there exist multiple sensors that exhibit similar behaviour. More specifically, the time-series data generated by these sensors would be highly correlated. For example, two lecture theatres in the same department in a university could have similar attendance and similar daily power consumption profiles.

In order to take advantage of correlated sensors to address the challenge of missing values, we develop multi-task learning methods which allow the mitigation of the effects of missing data on prediction accuracy. This is done by jointly learning models
on multiple correlated sensors rather than on each sensor independently.

In Chapter 4 we focus on these aspects and answer the question:

Q2. How can we mitigate the effects of low quality data on prediction accuracy in smart city scenarios?

1.2.3 Dealing with high-velocity data

Smart cities are related to the Internet of Things, which is often associated with information explosion. Smart cities require methods that are able to deal with this vast volume of data that is streaming in at a very fast pace. Further, we would prefer methods that are able to perform machine learning tasks in real-time while maintaining a constant space requirement.

We address this challenge in two steps. One, we focus on a label-based prediction technique that has high predictive throughput due to the minimal number of operations needed to be done in the prediction stage. However, since this prediction algorithm is not that accurate, we develop methods to improve its predictive accuracy using numerous methods, including filtering, windowing, and ensembling methods as well as methods of incorporating exogenous variables such as temperature to the label-based forecasting method.

Further, we develop methods to perform the initial part of the algorithm using
algorithms for stream processing, so that the algorithm can not only be run in real-time, but can also be parallelised across multiple machines in order to gain a further throughput boost. The forecasting algorithm we use was not developed in parallelisation in mind, which makes it non-trivial to implement on scalable streaming data frameworks (e.g. such as those based on a directed acyclic graph based paradigm). In addition, there are also challenges with regard to trade-offs between performance and accuracy that need to be addressed in the development of such a system.

In Chapter 5 we address the question:

Q3: How can we improve prediction accuracy and incorporate exogenous variables to label-based forecasting?

and in Chapter 6 we address the question:

Q4. How can we scale label-based prediction algorithms for a large number of sensors and for fast data streams?

1.2.4  Multi-step Forecasting

Most forecasting tasks focus on predicting one step at a time. For example, in a forecasting problem with training data recorded each hour we may choose to forecast the value for the next hour.
Though one-step forecasting is a relatively simple process, it has limited utility. For example in the smart city space, if we had hourly data, we would much rather forecast twenty four steps (i.e. twenty four hours into the future) in order to gain an overall understanding of the predicted behaviour of the time series over the entire next day.

Increased prediction horizons, despite being useful, are associated with a unique set of problems. For one, predicting further away from the present involves increased difficulty in forecasting due to increased uncertainty. This requires predictions that are more robust, which makes incorporating prior knowledge as mentioned in Section 1.2.1 all the more important. A problem more unique to multi-step prediction is that of propagating errors. For example, if the prediction for the next hour is used in the prediction for two hours into the future, the errors in the first prediction may carry over to the prediction for hour two as well.

Therefore, all the methods we develop in this thesis are formulated with the following question in mind:

Q5. How can we effectively perform multi-step prediction and minimise error propagation?
1.3 Organisation of Thesis

1.3.1 Chapter 2 - Literature Review

This chapter surveys the literature relating to forecasting in smart cities, with a focus on kernel-based Bayesian and scalable methods.

1.3.2 Chapter 3 - Bayesian Forecasting: Kernel Compositions to Improve Accuracy

This chapter focuses on improving prediction accuracy by providing methods to incorporate prior knowledge to kernel-based Bayesian forecasting models.

Since Gaussian Process Regression is widely considered to be the state of the art in regression for forecasting, it is used as the base model in this chapter. Gaussian Processes, being a Bayesian method, have the capacity for quite flexible prior specifications. However, the specification of a suitable prior in Gaussian Processes which improves prediction accuracy is a non-trivial task. Therefore, in the literature it is common to observe either:

1. Gaussian Processes being used for regression with quite simple priors, which do not fully capture the prior knowledge of the problem domain [47] [58]
2. The encoding of prior knowledge being performed by experts. There exists much work in the literature where an entire piece of work is based on providing methods for encoding expert intuition in the form of prior information for a specific application \([57][83]\) and \([92]\).

In this chapter we provide methods to incorporate prior knowledge of the working week in Gaussian Processes. We achieve this by providing novel kernel compositions which allow the encoding of this knowledge. We therefore improve upon the specification of naive priors, and unlike other related work in this space such as those mentioned in the second point above, our kernel-compositions are not constrained to a narrow problem domain.

The publication arising from this work is:

Chapter 4 - Bayesian Forecasting: Improved Accuracy with Low-quality Data

This chapter focuses on mitigating the impact of missing training data on prediction accuracy.

We achieve this prediction error minimisation for sensors with missing training data by providing a multi-task learning method, which allows the joint learning of models from sensors which are highly correlated to the sensor with low quality data. We provide representations which allow for this inductive knowledge transfer, and propose a multi-task kernel which allows for a configurable weighting of intra and inter-task dependencies. We also provide a formulation of the multi-task kernel as a linear kernel combination in order to facilitate straightforward implementation in popular machine learning toolkits.

The publication arising from this work is:

1.3.4 Chapter 5 - Label-based Forecasting

This chapter focuses on developing the base algorithm for scalable high-throughput forecasting.

As mentioned previously, we need to address the challenge of high velocity data when forecasting in smart city settings. Our scalable solution makes use of Pattern Sequence-based Forecasting (PSF) [71] which has very low computational requirements as the base algorithm. We improve on the relatively low accuracy of this label-based forecasting algorithm by adding core enhancements, ensembling techniques, and by providing methods to incorporate exogenous variables.

The publication arising from this work is:

Pasan Karunaratne, Masud Moshtaghi, Shanika Karunasekera, Aaron Harwood. ‘PSF+ - Fast and Improved Electricity Consumption Prediction in Campus Environments’, 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm).

1.3.5 Chapter 6 - Scalable Label-based Forecasting

This chapter focuses on scalable high-throughput forecasting.

In this chapter we develop methods to deal with the issue of scalability relating
to a large number of sensor streams, as well as relating to a single fast stream and address the challenges unique to each scenario. Our methods are based on a fast streaming data summarising algorithm based on micro-clusters (CluStream [5]), where the micro-clustering algorithm is used in concert with PSF to perform forecasting. We implement our methods on the stream processing framework Apache Storm.

The publication arising from this work is:

2.1 Introduction

Given the focus on forecasting in this thesis, we start our survey of the literature with an overview of the time series forecasting algorithms most related to our work and which have found popular use in the smart city space. We then provide a more
detailed overview of the algorithms which are more intimately related to our work.

Our work relates to improvements to Bayesian Forecasting using Gaussian Process Regression (Chapter 3 incorporating working week prior information and Chapter 4 forecasting with low quality data) and Scalable Forecasting (Chapter 5 developing base algorithm and Chapter 6 developing distributed scalable solution). In this respect, we provide a review of work on Gaussian Process Regression, especially in smart city application contexts, and also relating to kernel compositions and multi-task learning which are closely related to the work presented in this thesis. We also present prior work on the label-based forecasting algorithm we use to develop a scalable forecasting solution. Our scalable forecasting method is founded on novel distributed stream clustering algorithms we develop, and hence this literature review chapter concludes with a review on stream clustering and distributed stream clustering algorithms in the literature.

2.2 Time Series Forecasting Algorithms

Time series forecasting is a discipline with a long and rich history. We detail the work that is most related to our own work in this section. For a more in-depth exploration of the discipline, we note some notable surveys on the topic, including [35] which provides a long term overview of the developments in the discipline of time series
forecasting, and [41] which serves as a comprehensive survey of time series data mining in general.

We first survey simple straightforward models. Though these models do not provide high accuracy in predictions, their popularity due to their simplicity warrants mention. Of such methods, Exponential Smoothing is particularly well known. Exponential Smoothing is used in [29] for forecasting water distribution in cities, and for predicting safety values in cities in [16], where it was seen in both works to be outperformed by ARIMA and Neural Network models.

Neural network models are commonly used in forecasting for smart city scenarios. For example, atmospheric particulate matter forecasting using neural network models has been performed in [33] and [77], human activity recognition in [116], and urban traffic flow forecasting in [74] [55] and [117].

However, these models often require large volumes of data to be effective and tend to be sensitive to the parameter values and tuning methods used. Neural networks in some scenarios may perform poorly on multi-step forecasts though providing good performance on single step (e.g. one hour prediction horizon) predictions [73]. Methods that enable neural networks to learn long-term dependencies for multi-step prediction may require large volumes of data. Multi-step prediction could be achieved either by building multiple models for each timestep, or by using
the predictions for the immediate time step as input to predicting timesteps further in to the future. Usage of separate models can be effective in situations where there is a limited recency effect in the time series, and reusing predictions as training data for predictions further into the future can be effective in situations with low probability of errors being compounded. However, such issues are alleviated in models which are able to encode the periodicity of the data, thereby enabling easier multi-step extrapolation. Further, given the low interpretability of neural network models, they may not be the best fit in real-world scenarios, where understanding the model to facilitate planning may be as important as the predictions generated from the model itself.

Regression models are also quite popular in forecasting scenarios, especially in early work. For example, [79] and [23] used regression for short-term electricity load forecasting, and [48] used it for peak load forecasting. In more recent work, [56] used a regression model to evaluate the effects wind power forecasts had on electricity spot prices, with [99] and [52] using it for traffic flow forecasting.

Among regression techniques, the ARIMA technique is probably the most widely used. Due to its popularity we often use the ARIMA technique as a benchmark in our experiments, along with other more advanced techniques. ARIMA models also have seen particularly wide use in smart-city based applications. For example, electric-
ity demand forecasting using ARIMA has been performed in [28] [88] [80], with electricity price forecasting done in [31]. A method to forecast fossil fuel production was developed in [40], and a derivative of ARIMA was used to make predictions of traffic conditions of transportation network links in [104].

In our work we use a more sophisticated model of regression, Gaussian Process Regression, which allows for forecasts with higher accuracy. Gaussian Process Regression also allows for probabilistic prediction, the specification of highly flexible priors and automatic feature selection. We provide a comprehensive overview of prior work related to Gaussian Process Regression in Section 2.3 of this chapter.

Gaussian Process Regression also falls under the category of kernel-based methods. Kernel-based methods have received much attention in the literature of forecasting. Chief among other kernel-based methods is Regression using Support Vector Machines (Support Vector Regression), which is grounded in statistical learning theory and has been proven to outperform other non-linear techniques [90]. We provide comparisons to Support Vector Regression in our work. Support Vector Machines have found application in smart city scenarios in parking availability prediction in [118], production output forecasting of industry in [78], travel time prediction in [109] and electricity price forecasting in [89].

In addition to these commonly used methods, other techniques seen include Ge-
netic Algorithms and Particle Swarm Optimisation models. Though being effective in their own niche, these methods have not seen wide adoption in smart city forecasting scenarios compared to the other work mentioned above.

In the next section we extend our survey of prior work to provide deeper insight into work that is more closely related to the work presented in this thesis.

2.3 Bayesian Forecasting

Chapters 3 and 4 relate to our work on Bayesian forecasting. We focus on incorporating prior information related to working-week data to Bayesian forecasting in Chapter 3, and in Chapter 4 we provide methods to perform forecasting in scenarios with low quality data. We use the state-of-the-art Gaussian Process Regression model as the baseline technique.

2.3.1 Gaussian Process Regression Applied

Using Gaussian Processes for regression was popularized in the machine learning community by [106] [84], after they were introduced to the statistics community in [76]. Gaussian Process Regression is a highly flexible method, and consequently has been used in a wide variety of forecasting problems in the literature which relate to real-world problems. For example, [25] forecasts power generation in wind
farms, [19] makes predictions on the effects of combinations of drugs, and [98] [105] build models to represent the kinematics and adaptive control functions to be used by robotic manipulators. Work related to robotic mapping has also been undertaken using spectral analysis to model periodic environment processes [60].

Gaussian Processes have also found application in a number of disciplines which are directly related to smart cities. For example, [66] uses Gaussian Process Regression to estimate traffic flow in order to perform routing in smart cities. They also predict future traffic conditions using a Spatio-Temporal Random Field approach, and combine the two prediction components to create a trip planner. Their use of Gaussian Process Regression is quite straightforward, where they simply use a general Laplacian kernel function since they lacked prior information on preferred routes. Our work in Chapters 3 and 4 make use of elaborate kernel compositions to incorporate far more prior information with the goal of improved accuracy.

Traffic prediction in Dublin city centre is also performed in [91]. They also mention the potential of leveraging more information such as the street network in their discussion. This common focus on incorporating prior information in work on Gaussian Process Regression which relates to real scenarios illustrates the importance of our work in Chapter 3, which provides novel methods to incorporate prior information relating to the working week.
Gaussian Processes are used for probabilistic travel time prediction in [53] on realistic data. Again, similar to our work on novel kernel compositions to incorporate prior information, one of their key contributions relates to kernel functions. In this work, a kernel function for encoding the similarity of a sequence of symbols first introduced in bioinformatics (spectrum kernel [65]) is repurposed for the purpose of encoding similarity between trajectories.

In [43] the authors develop an early warning system against malicious attacks on smart grid communications. This work is also a rather generic application of Gaussian Process Regression, though it relates to a work in a real-world scenario where they endeavour to solve the novel problem of abnormal modes of operation in smart grids.

In relation to the specific application domains we focus in this thesis, Gaussian processes have been used in noteworthy work such as [47] [64] to predict electricity load data. However, they either do not consider the periodic structure in the data, or do not explicitly incorporate weekend and weekday information in their models. Further, prior beliefs on parameter values are not encoded which fails to utilise a source of information that could help with the optimisation process.

For example, [47] performs generic multi-step Gaussian Process Regression, with genetic algorithms used to tune hyperparameters of the kernel functions. They sim-
ply use multiple Gaussian Process models to perform multi-step prediction, which in contrast to our work in Chapters 3 and 4 would not allow the encoding of periodicities and would be less than ideal in terms of computational expense. An elaborate model is proposed in [64] to forecast on Irish load data by making use of the modular nature of kernel functions as we do in our work, though they still do not explicitly address working week effects or make use of priors over hyperparameters.

In relation to the application domain of pedestrian data, little work exists in the literature. The work in [36] performs anomaly detection on pedestrian data, but to the best of our knowledge no attempt at predicting future counts has been done.

In summary, there exist numerous work which use Gaussian Process Regression in the context of smart cities, which further indicates that this is an important and relevant area of research. However, there is a lack of work which looks at the periodic structure of data as it relates to the working week and on adding prior belief on hyperparameters in order to perform multi-step prediction efficiently. We address this gap in the literature in the work in Chapter 3.

2.3.2 Gaussian Process Regression: Kernel Compositions

The first part of our work in relation to Bayesian forecasting (Chapter 3) is incorporating prior information relating to working week data. This work would help in any
scenario with working-week type patterns, and therefore could be especially helpful in improving prediction accuracy in smart city scenarios. We make use of novel kernel combinations to encode this information. There is much work in the literature where the major contribution is the provision of novel kernel combinations to model a specific application domain. Though our work is more general than these works that are highly specialised, work with similar contributions of analysis of kernel combinations in different application domains include identifying the effects of additive errors in control systems [57], forecasting the number of tweets with a particular hashtag [83], and modelling the propagation of seasonal influenza [92].

The kernel composition in [57] used to identify effects of additive errors in control systems is a quite straightforward multiplication of two popular kernels (squared exponential and periodic). This produces a locally periodic kernel which is used for improved prediction of effects due to imprecision in gears and drives in control systems.

A custom-built ‘Periodic Spike’ kernel is introduced in [83] to more accurately represent the nature of Twitter data volume for a given hashtag which exhibits patterns of abrupt periodic peaks. They do also explore kernel combinations (e.g. Squared Exponential*Periodic Spike, Periodic Spike + Linear), though in their scenario the combinations did not yield significant benefits, especially given the drastically increased
model complexity and computational time.

A prime example of kernel compositions can be seen in [92], where the spatio-temporal nature of the propagation of seasonal influenza is modelled using three separate kernels for time, space, and the interplay between space and time. The kernel for time itself is formed through the addition of three separate kernels, one each for periodicity, short term, and long-term trends, whereas the space kernel is an averaging kernel. The kernel for space-time is formed by the product of a squared exponential kernel with the time kernel.

It is also worth noting the extensive emphasis placed on the custom construction of kernel compositions in the authoritative text on Gaussian Processes [85]. The text models the air CO2 concentrations recorded at the Mauna Loa Observatory in Hawaii. The kernel composition allows for the modelling for a smooth trend (squared exponential kernel), a seasonal component (periodic kernel) and medium term irregularities (rational quadratic kernel).

In summary, there exists a reasonable volume of work that investigates incorporating prior information in the form of kernel compositions in order to improve prediction accuracy, though none that look at incorporating prior information of working week periodicities in order to improve multi-step prediction accuracy as is done in Chapter 4.
2.3.3 Gaussian Process Regression: Multi-task Learning

Though the work mentioned above utilise Gaussian Processes in smart city application domains, they do not utilise knowledge transfer between various sensors, nor do they address the key real-world problem of data with gaps in the input. This limitation provides the motivation to the second part of our work relating to Bayesian forecasting (Chapter 4), which proposes a method to perform accurate predictions based on training data with missing values using multi-task Gaussian Processes.

Multi-task learning is a general concept which enables better models to be built for a given task by making use of information contained in related tasks. The earliest canonical reference to multi-task learning is the dissertation [22]. The work in [100] is also an early foundational reference which explores how the concept of multi-task learning relates to lifelong learning by humans, where humans exploit experiences from related learning tasks when faced with learning a new task. Noteworthy work on multi-task learning include [30], which uses a neural network trained jointly on the tasks of predicting part-of-speech tags, chunks, named entity tags etc, and an elegant approach to multi-task learning proposed in [42] based on Support Vector Machine fundamentals.

A method of multi-task learning is proposed in [111], where the commonality between tasks is captured by assuming that the parameters of the separate individual
models for tasks are drawn from a common (hyper)prior. Their work is unique in that compared to other work on hierarchical modelling which addresses parametric modelling, they focus on the nonparametric domain of Gaussian Processes.

Modelling the commonality between tasks in multi-task learning in Gaussian Processes is approached in [62] by using a shared covariance function between the tasks. This approach is commonly used, with [20] being another noteworthy example, where inter-task dependencies are effectively learnt without the need for a large amount of training data.

There exists a reasonable amount of work on multi-task Gaussian Processes in various application domains in the literature. These include applications as diverse as personalised age estimation[114], modelling of robot dynamics [105] and learning effects of drug interactions for HIV therapy screening [19].

A variant of Gaussian Processes called warped Gaussian Processes is used in [114] to perform automatic age estimation from facial images. In their context of multi-task learning, each task corresponds to a person. Their multi-task warped Gaussian Process allows for the model to learn commonalities between different tasks (i.e. persons), while allowing for unique features of each person to be learnt as well. The use of multi-task learning in this domain also helps solve the problem of data sparsity, since in most facial image datasets there exists only a very limited number of training
examples per person.

An oft-cited example of multi-task learning with Gaussian Processes is \cite{105}, which models inverse robot dynamics. A robotic manipulator needs to compute the torques needed to drive it along a given trajectory. This is referred to as the inverse dynamics problem, and the work focuses on learning these inverse dynamics. The multiple tasks relate to the different loadings under which the robotic manipulator needs to operate. The authors use of multi-task learning is to exploit the commonality in these tasks (i.e. different loadings) in order to improve performance rather than learning separate models for each task.

The therapy outcomes of combinations of drugs for HIV is predicted in \cite{19}. This is an elegant practical use of multi-task learning. It directly addresses the problem of sparse data in the problem domain, which does not allow the training of separate models for each combination. Multi-task learning allows the prediction for drug combinations which have little to no training examples, and the authors demonstrate that it allows for substantially improved overall prediction accuracy as well.

In summary, there exists work in highly varied application domains which use multi-task learning for knowledge transfer. However these works do not use multi-task Gaussian Process for handling noisy data and work with complete data. Though \cite{97} does discuss prediction using noisy heart-rate training data, they do not directly
address the problem of gaps in the data. We address this limitation in our work in Chapter 4 which proposes a multi-task learning method to solve the problem of missing training data in Bayesian forecasting.

2.4 Label-based Forecasting

We focus in Chapter 5 on a fast forecasting algorithm (Pattern Sequence Based Forecasting (PSF) [71]) which will be suited to develop to a scalable solution. PSF is an improved version of the work presented in [70]. PSF is particularly suited for the purposes of fast forecasting on high velocity streams. An extensive discussion of the merits of PSF can be found in the introduction to Chapter 5. We provide improvements in Chapter 5 to the PSF algorithm to improve its accuracy, especially where forecasting needs to be done on an individual sensor level.

Much work has been done on the analysis of smart-meter data, though a lot of this work is on data obtained through utility companies for a larger number of electricity consumers and not on an individual sensor level. For example, [102] presents methods to segment consumers using load information for 5000 consumers in Ireland obtained from the Irish Commission for Energy Regulation. This same dataset was used in [103] in a ‘Cluster-based Aggregate Forecasting’ method to forecast residential electricity demand. The occurrence of outliers in time series is predicted in [69],

30
where it uses three consumer data sets from the Spanish, New York and Australian energy markets to demonstrate their results. The work which presents the algorithm that is the basis of Chapter 5 [71] also performs the evaluation of the algorithm on data obtained from the three national energy markets mentioned above. An ensemble model for forecasting is developed in [93], where the authors report results on data from the three energy markets mentioned above as well as from Ontario’s Independent Electricity System Operator.

Our requirement for PSF is to perform prediction on an individual sensor level, which presents a greater challenge in terms of accuracy and makes the enhancements we perform in Chapter 5 vital. Though we did observe reasonable performance when predicting in aggregate, similar to the works cited, our goal as stated is to make predictions for each individual sensor. Given the higher variability of individual sensor data this proved to be a challenging problem. Therefore the methods proposed in these prior work in the literature (which either predict in aggregate or make novel use of PSF for purposes other than prediction) cannot be directly used to improve the accuracy of the PSF algorithm.

Similar work has been done on analysing data from smart meters in campus environments, which is one problem domain we focus on as well. For example, [11] presents work on prediction of reduced energy consumption on 32 buildings in a cam-
pus micro-grid. Reduced consumption prediction is important to estimate potential reduction of energy use during a Demand Response event, and is distinct from the more general problem of forecasting energy use for all time periods within a campus. A method of anomaly detection is presented in [44] on data for two selected buildings from the University of California Berkeley and the University of Tokyo. Work on analysing the nature of power consumption in campus micro-grids have also been performed, with [3] performing analysis on energy usage in the University of California San Diego campus. A similar study on six weeks of data obtained from 39 power meters in a commercial (as opposed to an educational) campus is performed in [18].

Methods to build separate models for each cluster of smart meters, and then using the sum of the predictions output from those smart meters as the prediction for global power usage across all sensors is common in the literature. A method of forecasting using incremental clustering of customers using the Los Angeles Power Grid is presented in [94]. Interestingly, they also identify the problem of increased variability in individual-sensor predictions resulting in lower accuracy levels, and use clustering to form ‘virtual consumers’ to reduce the prediction error at an aggregate level. Kernel spectral clustering is used in [10] to cluster smart meters in order to derive a more accurate prediction for the global power usage. A similar methodology is followed in [103], in that they perform an initial clustering step and then use the
aggregate of the forecasts output by the separate models to provide a global forecasting estimate. However, the problem we investigate is to provide forecasts to each individual sensor in the network. Therefore these methods which only investigate forecasting the aggregate demand is not directly applicable to the problem we solve in our methods that allow the prediction methods to scale.

The PSF algorithm has found much use in the literature. For example, PSF is used to forecast the possibility of outliers in [69]. The methodology to potentially detect outliers is similar to the general forecasting algorithm - however outlier detection adds an additional step where label sequences preceding known anomalies in the training data will be stored. If a data point is preceded by such a motif, the algorithm will determine that data point to be highly likely to be an outlier. Though we do highlight the potential utility of using forecasts as a method of retrospectively identifying anomalies, we do not investigate the possibility of detecting anomalies in advance. Though the forecasting of anomalies is quite a different area, this work demonstrates the utility of the PSF algorithm and forecasting in general.

PSF is used in a forecasting scenario in [68], where the energy consumption at electric vehicle charging stations in the campus of UCLA is predicted. The authors proposed a modified PSF algorithm, which is a combination of k-Nearest Neighbours and PSF. The authors claim that considering only the most recent match in PSF pro-
duces better accuracy results compared to conventional PSF.

PSF is combined with neural networks in [59], again for forecasting purposes. The work refines the predictions output by PSF using a neural network. The neural network also makes use of additional features such as values from the previous day, previous week or both. The authors perform their evaluation on electricity demand in the state of New South Wales in Australia.

We make use of the exogenous variable of temperature in our enhancements to the PSF algorithm. A similar interesting application of exogenous variables is given in [12]. Under the guise of ‘indirect indicators’, the authors use the exogenous variables of weather conditions, building conditions (such as gross area, net area in use, year of construction) and occupancy patterns to improve energy use forecasts. Though these variables were not available to us, such variables could be included using similar methods as those developed to incorporate temperature to the label-based forecasting method. This possibility has been further outlined in Chapter 7.

An ensemble method is proposed in [93] for performing forecasting, with vanilla PSF being one of the algorithms in the ensemble. We use a method of ensembling within the PSF algorithm (described in Chapter 5) as an improvement to PSF itself. This is quite different to the cited work, where no improvements are done on PSF, and where the ensembling of PSF with other models is the primary contribution.
Energy outlier detection in ‘smart environments’ is performed in [24]. Their work is similar to our work on PSF, in that they also perform an initial step in transforming the data (into a suffix tree data structure), and then they use a clustering algorithm to detect energy pattern outliers (an outlier being a data element that falls far from cluster centroids). The manner in which a more efficient representation was chosen to be built in a preprocessing step, and that the validation was done on real data collected from a smart apartment test bed, makes this work similar in spirit to ours. However, the goal and methods they develop are to detect outliers, and not to perform forecasting.

In summary, PSF has been used in a wide variety of works, and its use has not been restricted to only forecasting. In this work we improve on the vanilla PSF algorithm to improve predictive accuracy, especially in cases where prediction needs to be done on an individual sensor level, in order to develop a scalable forecasting solution with PSF as the base algorithm.

2.5 Scalable Forecasting

Predictive analytics is a well-studied area, though less so relating to scalability of forecasting [94].

A neural network technique is proposed in [45] that selects the best prediction
algorithm for each customer in an attempt to improve scalability. Their main contribution is also related to scale, in that it is impossible to run every prediction algorithm for all the customers. Therefore, they use a neural network in order to select the potential best possible prediction algorithm for a given customer.

Though the work makes a noteworthy attempt to address the scale problem, the work depends on the underlying assumption that a single machine can handle the data streams using one algorithm. Their argument is that a single machine cannot run multiple algorithms due to scale problems, therefore they provide a neural network technique to cheaply select the one potentially best algorithm to perform the analysis. Though this method will improve scalability, it will only improve scalability by a small constant factor (i.e. equal to the number of potential different forecasting algorithms considered). Our methods allow the increasing of the scale of the prediction system by the number of processing units available to be utilised, as is generally expected out of a scalable system.

Clustering is used in [94] to estimate the global energy consumption by aggregating the predictions of smaller consumer clusters, which improves scalability as well as accuracy in aggregate forecasts (whereas we require scalable algorithms for individual forecasts). This work is interesting in that they develop a novel similarity metric, where the consumers that fall into a particular cluster are those that reduce the pre-
diction error for the cluster. It is noteworthy that unlike traditional applications of clustering, it is not necessary for the consumers in a particular cluster to have similar behaviour to each other - the key requirement is that the addition of a customer to a cluster should result in reduced prediction error.

A scalable system for forecasting for low voltage grids is proposed in [86]. The authors use a weighted kernel density estimation method as the baseline forecasting algorithm, and use Gearman * to distribute among multiple nodes. Though the combination of kernel density estimation with a scalable framework is noteworthy, not much contribution is made in relation to scaling other than the use of off-the-shelf software, which is not possible with algorithms which are not inherently embarrassingly parallel. The work also does not support real-time stream forecasting.

We develop a scalable forecasting method (using PSF as the base model) in Chapter 6 by performing the labelling in label-based forecasting using stream clustering methods that we have enhanced to run on distributed platforms. We present an overview of work in the literature relating to stream clustering algorithms in Sections 2.5.1 and 2.5.2.

*gearman.org
2.5.1 Stream clustering algorithms

There are numerous stream clustering algorithms that have been proposed in the literature, each with its own niche of improvement. For example, [26] and [21] were developed to be able to find clusters of arbitrary shapes, and to eliminate the need to fine-tune various parameters. The work in [1] claims to offer better quality clusterings, though it is admitted that the algorithm runs slower than other stream clustering algorithms.

Though stream clustering algorithms differ significantly from each other, a large number of stream clustering algorithms share a common aspect in terms of the summary data structure that they maintain. Many algorithms use the micro-cluster summary data structure, which was derived from the feature vector data structure first introduced in [113]. The micro-cluster data structure found its first use in [5]. Thereafter, it has been used in [21] in order to provide more robust clusterings in the presence of outliers in the data stream. Micro-clusters have also been used in [61] to provide anytime data clustering regardless of the stream data rate, and [7] uses the data structure in order to perform clustering on uncertain data streams. Micro-clusters also have found their use in data stream classification scenarios [6]. Given this wide use of micro-clusterings in a variety of work, the micro-cluster was the summary data structure chosen to be focused on in developing distributed stream clustering algo-
2.5.2 Distributed clustering algorithms

Most of the work related to distributed clustering make use of a central coordinator in order to maintain a global picture of the changes in the local nodes. The literature focuses greatly on limiting the communication overhead between the local nodes and the coordinator both by limiting the amount of information sent to the coordinator in each pass, as well as by minimising the frequency of such updates.

For example, in [119] a central coordinator is used to keep track of the models at each local node and to merge all these models to a global model. They claim reduced communication overhead due to transmitting only model parameters and transmitting only when the data distribution changes at each node. In [32], too, the local nodes cooperate with a distinguished node known as the coordinator. A hierarchical approach to the problem is used in [112], with summaries of data computed at local nodes being merged at higher levels. When all summaries reach the root, a k-median clustering procedure is performed. Other work that use this coordinator concept are [87] [54] [49].

Work that does not use a central coordinator approach do exist for clustering. For example, [17] propose a peer-to-peer version of k-means, though qualifying the ben-
efits of the algorithm by stating that it was not designed to deal directly with continuous data streams. A generic local algorithm for k-means monitoring is presented in [108], but not for the actual clustering process itself. There also exist work on parallel versions of traditional batch clustering algorithms such as [115].

There is, however, limited work on clustering streams of data in a parallel manner for the specific purpose of providing performance gains, as opposed to being motivated by inherently distributed data. The SAMOA [34] library includes an implementation of the micro-cluster maintenance algorithm CluStream that aims to provide a distributed version of the algorithm. However, this implementation does not explicitly consider the issues of synchronisation between nodes running in parallel.

Another related work is found in [46], where they present an algorithm for distributed clustering of social media data, specifically Twitter data, in a streaming fashion. However, the algorithm employed is specifically built for the Twitter domain, with preprocessing steps that operate on Twitter-specific constructs. Our distributed architecture and the implementation are for a well established, generic algorithm for clustering which can be applied to any domain, including the Twitter domain as we have demonstrated in Chapter 6. The work in [39] [15] involves processing streaming graphs in parallel as opposed to the clustering of streaming feature vectors that our work focuses on. Further, [110] presents a simplified clustering algorithm which
does not deal with synchronization among the parallel clustering workers.

2.6 Conclusions

We provided an overview of the developments in Bayesian, label-based as well as scalable forecasting methods, as well as general forecasting methods as they relate to the contributions we make in this thesis.

The use of kernel-based methods (e.g. Support Vector Regression) in prior work on smart city scenarios was illustrated, along with work that used the kernel-based method of Gaussian Process Regression which we use as the base model in our work. The advantages of Gaussian Process Regression were outlined along with work that made use of these benefits, with an emphasis on prior work which related to smart cities. We discussed the limitations of prior work (e.g. using simple formulaic kernels, focusing on a narrow problem domain), and also describe work similar to ours (e.g. those that propose novel kernel compositions to encode various types of prior information.) We provided an outline of prior work that use multi-task learning, and discussed the limitation of these works not directly addressing issues with missing training data impacting prediction accuracy. In relation to the scalable forecasting algorithms we develop, we discussed prior work similar to the base forecasting algorithm we use (PSF) and prior work with similar enhancements to those we make to
the algorithm. We also provided a detailed overview of stream clustering algorithms, as distributed stream clustering is the main component in our forecasting methods that can be run in parallel on multiple machines and in real-time on data streams.

We will build on the prior work in the literature in the methods we present in the chapters that follow in this thesis.
Bayesian Forecasting: Kernel Compositions to Improve Accuracy

Accuracy is a primary goal in any forecasting task. This is especially true in real-world scenarios, where sub-par predictions can at best lead to the models being eventually disregarded by decision makers, and at worst contribute to faulty decisions with un-
happy consequences. Therefore, our focus in this chapter is on this primary goal of maintaining predictive accuracy.

Maintaining accuracy in multi-step prediction, as we focus on in this chapter, is even more challenging compared to the more commonly addressed single-step forecasting task. Though multi-step prediction has increased real-world utility, it also involves predicting further away from the present, which involves increased uncertainty and increased potential for errors. A further challenge is that these errors are also at risk of propagating between steps in multi-step prediction.

Common methods used to address the challenges in multi-step prediction include Single Value Prediction methods [27] [47], as well as incorporating external information [92] [11]. Though these are legitimate methods of addressing the challenge (and indeed we make use some of these methods in our work as well), in this chapter we explore further and focus on incorporating even more information to the model, especially information of which we have high confidence and which are closely related to the behaviour of cities.

Cities have periodic behaviour. Indeed, this is arguably the most common complaint of the regular office worker - that the work is routine. This periodic nature of work is often reflected in smart sensor recordings, since the recordings are influenced

---

Pasan Karunaratne, Masud Moshtaghi, Shanika Karunasekera, Aaron Harwood, Trevor Cohn. ‘Modelling the Working Week for Multi-Step Forecasting using Gaussian Process Regression’, International Joint Conference on Artificial Intelligence (IJCAI) 2017
by the behaviour of the workers. Many real-world smart city applications involve data which follows a periodic working-week structure, with weekends exhibiting substantially different behaviour to weekdays.

This periodic behaviour of smart city time series provides us with an opportunity when dealing with the difficult task of performing multi-step prediction. The *multiple* periodic effects (e.g. daily, weekly, holiday effects) commonly observed in smart city data, and how these multiple periodic effects interact is a rich source of prior information that can be used to add to the information the model derives from training data. Explicitly incorporating this prior knowledge in the form of periodic effects of the working week would likely improve predictive accuracy, especially in performing multi-step prediction in smart city settings.

In addition to potential improvements in prediction accuracy, another reason to consider incorporating more prior knowledge in smart city settings is to guard against sudden changes. Though the work may be routine, the prediction still relates to a real-world scenario: therefore it might be the case that we receive anomalous information, either due to spurious data or due to a real change in the environment. We need to guard against prediction algorithms being unduly influenced by such temporary changes. Again, if we have highly informative priors about which we have strong confidence, they will make the prediction algorithm resilient to any temporary
changes while being adaptable to a real concrete change in environment.

Incorporating prior information is the central tenet of Bayesian methods, and are a good fit to the task at hand. Among Bayesian forecasting methods, Gaussian Processes are widely held to be the state of the art method to perform regression, and are popular for their support for intrinsic feature selection and their ability to model complex functions. In addition to being a cutting-edge and popular method, they are especially suited to our task due to their versatility in defining remarkably flexible priors.

Their capability of incorporating flexible priors has seen their use in adding side information to other models as well [2]. These priors are interpretable, and we can sample from the defined prior to get an intuition about how the functions drawn from those priors will behave and the assumptions of the model. Further, no matter how flexible the kernels that define these priors get, we can always learn the kernel parameters directly from the data. These advantages make Gaussian processes uniquely appropriate to add prior information to a model. However, incorporating prior information to Gaussian Process Regression is not a straightforward exercise.

Prior information is encoded in Gaussian Process Regression models through the specification of covariance functions. A simple formulaic specification of a common covariance function will fail to fully capture the unique characteristics of the data
in most real-life scenarios. Such generic model specifications will fall short in prediction, since the prior knowledge incorporated by the covariance functions chosen in Gaussian Processes plays a major role in extrapolation, and thereby their prediction accuracy [83].

On the other hand, though more elaborate models potentially could model the data more closely, the wide variety of combinations possible and the specificity of Gaussian Process model specifications to the given problem domain results in this model specification being done by experts. Notable examples of work in which the main contribution involves experts encoding structure relevant to a given problem by using kernel combinations include [57] [83] and [92]. Too often though, simple models are built which do not fully utilize the flexibility of Gaussian Processes [47] [58]. This is particularly apparent and understandable given the difficulty involved in creating elaborate models with multiple kernels, and also the associated challenges where the high number of parameters increases computational complexity as well as the likelihood of reaching local optima in the optimisation process.

We do not shy away from these challenges involved in a more accurate modelling, and focus on modelling data in smart city scenarios more closely with the end goal of improved accuracy. In smart city scenarios, modelling the behaviour of the Monday-Friday working week is especially important given the periodic nature of work. Specif-
ically, modelling multiple periodic effects would enable a closer modelling of the data. For example, we would like to encode the information that sensor recordings on Mondays tend to be similar to each other (weekly periodicity), as well as incorporate the knowledge that the sensor recordings within a given week would have similarities (daily periodicity.) Modelling weekly periodicities will help in differentiating between weekends and weekdays, and also provide more information to the model about a given type of weekday (such as Mondays). Modelling daily periodicities will help when for example the days in a given week have relatively higher (or lower) values than the days in prior weeks, and will also encode a recency effect to the model.

In order to perform such a close modelling of working-week data, in this chapter we solve the challenging problem of creating new kernels which are able to incorporate information regarding multiple periodic effects to the Gaussian Process model. As mentioned earlier, the formulaic model specification involves using a commonly used off-the-shelf kernel, which is the approach taken in prior work, including [64] which uses a single periodic kernel, and [47] which uses the popular squared exponential kernel. However, the use of such simple models that do not explicitly model the working week will fail to incorporate a significant source of information that can be invaluable in forecasting scenarios.
Therefore, we go beyond the common straightforward model specifications present in prior work, and address the problem of modelling multiple working week periodicities in the same model in order to improve predictive accuracy. For example, we come up with novel kernel compositions that are able to model both daily (for recency effects within a given week) and weekly (for recency effects across weeks) periodicities in the same model. Performing this combination between the multiple periodicities effectively so that the end goal of improved predictive accuracy is achieved is also a challenge, for which we provide multiple methods. These include methods that enable automatic switching between the kernels in the composition based on the time instant being modelled, and also methods which enable the weighting between the multiple kernels to be automatically learnt from data. Such modelling of multiple periodic effects inevitably leads to more elaborate models, which creates the challenge of dealing with a higher number of parameters over which we need to optimise. In this chapter we also provide methods to address this challenge of a harder optimisation problem which is more prone to converge to local optima.

We provide a brief summary of our contributions below.

1. We provide novel kernel compositions to explicitly encode weekday and weekend effects in a working week for improved accuracy in multi-step prediction. These methods are flexible enough to be easily extended to model holiday ef-
2. We suggest methods to mitigate the effects of convergence to local optima in the optimisation process over hyperparameters, which is especially important in the case of low volume of training data or complex models.

3. We illustrate the effectiveness of our approaches on two real-world time-series datasets relating to electricity consumption and the counts of pedestrians in a city.

Our results showed consistent and significant accuracy improvements when using the hyperpriors proposed in this chapter to improve the optimisation process (e.g. error metric improvements from 36% to 17% for daily kernel vs daily kernel with hyperprior). Further accuracy improvements were seen when using all kernel compositions proposed in this chapter (e.g. error metric improvements from 17% to 8% for daily kernel with hyperprior vs kernel composition I).*

We also demonstrate that our methods provide better accuracy across a range of parameter values, regardless of the dataset or hyperparameter being varied.

We begin the discussion of our methods with an introduction to Gaussian Process Regression, which is used in this chapter as well as Chapter 4.

*The error metric used is expressed as a percentage value as given in equation 3.11
3.1 Introduction to Gaussian Process Regression

3.1.1 Gaussian Processes

A Gaussian Process is formally defined as a collection of random variables where any subset of the random variables taken together jointly form a (multivariate) Gaussian distribution. A useful intuition is to view a Gaussian Process as defining a distribution over functions (function-space view [85]).

Since a Gaussian Process is a distribution over functions, sampling from a Gaussian Process results in the draw of a single function. It is possible to specify prior belief in the general properties we expect to see in these functions drawn (e.g. whether the function is continuous). Further, Gaussian Processes follow the Bayesian paradigm of updating prior beliefs based on observed data to form posterior distributions. In the case of Gaussian Processes, these prior and posterior distributions are distributions over functions, and therefore the Bayesian inference that takes place occurs in function space.

A Gaussian Process is fully defined by its mean function \( m(t) \) and covariance (kernel) function \( k(t, t') \) (where \( t \) and \( t' \) are two separate input vectors). Therefore we can write the Gaussian Process as \( f(t) \sim GP(m(t), k(t, t')) \).

In our tasks our data consists of \( n \) pairs \( D = (t_i, y_i) \), where \( y_i \) is the value of the time
series at time $t_i$. We define without loss of generality a prior mean function of $0$ and covariance function $k(t, t')$, with the resulting general form of the Gaussian Process $f(t) \sim GP(0, k(t, t'))$.

### 3.1.2 Regression

Our goal is to predict values $y_*$ at time $t_*$ given training data $D$. We assume a latent function $f$, which provides the values for each data point according to $y_t = f(t) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$ is Gaussian noise. Noting that we perform Bayesian inference with regards to functions $f$, we can write the posterior predictive distribution as follows:

$$p(y_* | t_*, D) = \int p(y_* | t_*, f) \cdot p(f | D)$$  \hspace{1cm} (3.1)

In the case of Gaussian Processes, we are able to solve for this predictive posterior analytically with the solution given below:

$$y_* \sim N(k_*^T (K + \sigma^2 I)^{-1} \mathbf{y}, k(t_*, t_*) - k_*^T (K + \sigma^2 I)^{-1} k_*)$$  \hspace{1cm} (3.2)

where $k_* = [k(t_*, t_1)...k(t_*, t_n)]^T$ are the kernel evaluations between the test point and all the training points, $K = [k(t_i, t_j)]_{i=1}^{n}$ is the covariance matrix formed by the evaluations of the kernel function between all pairs of training points, and $\mathbf{y}$ is the vector
Therefore, it can be seen that the posterior includes both the mean and the variance of the response at the required prediction time points. In this chapter the expected value (i.e. mean) of the response is considered as the forecast of the prediction.

3.1.3 Covariance (Kernel) Functions

Covariance (kernel) functions are central to Gaussian Processes. The kernel function is one of two components (the other being the mean function) that fully defines a Gaussian Process. Since in most scenarios the mean function is set to zero (without any loss of generality), the kernel function represents all of our prior belief of what the functions sampled from the Gaussian Process will look like.

Kernel functions define the similarity between any two input points. That is, a kernel function outputs a real-valued similarity score for any given pair of input points. Evaluating the kernel function over all pairs of input points results in a conventional covariance matrix.

The choice of kernel function determines the overall structure of the functions drawn from the Gaussian Process. For example, consider the popular Squared Expo-
nential (SE) kernel function defined below:

$$k_{SE}(t, t') = \sigma^2 \exp \left( -\frac{(t - t')^2}{2l^2} \right)$$  \hspace{1cm} (3.3)

A kernel function defines the covariance or similarity between pairs of points. In this example, when the two points $t$ and $t'$ are close to each other they will have high covariance compared to points far apart. This encoding of higher similarity to points closer to each other results in smooth functions being drawn from the Gaussian Process defined by this kernel. We are further able to change the properties of these smooth functions by changing the values of the hyperparameters of the kernel $\sigma$ (output variance) and $l$ (length scale).

3.1.4 Combinations of Kernel Functions

Combinations of kernel functions can be used to model a number of effects together. For example, a squared exponential kernel coupled with a periodic kernel can induce a recency effect to the periodic kernel.

Kernels may be combined by either addition or subtraction. The addition of two kernels can be thought of as a logical OR operation, in that the final value of the addition will be high if either one of the two kernels being added outputs a high value. Similarly, the multiplication of two kernels is similar to a logical AND operation,
where the final value is high only if both the two base kernels output a high value.

3.1.5 Optimisation over Hyperparameters

The nature of the functions drawn from the Gaussian Process depends on the type of kernel chosen, as well as the values of the kernel hyperparameters. These hyperparameters are optimised to fit the training data.

The marginal likelihood of a model provides a measure of how likely the data was generated by the given model. Therefore, maximising the marginal likelihood of the model would improve the fit of the model to the data. In Gaussian Processes, it is possible to formulate an analytic solution to the marginal likelihood. Therefore, the Type II maximum likelihood estimate of the marginal likelihood is found by using gradient ascent with respect to the model hyperparameters in the training process. It is important to note that the marginal likelihood is non-convex in the hyperparameter values, and therefore the optimisation process is at risk of converging to local optima.

3.2 Methodology

We detail the process we followed in building the Gaussian Process models to perform multi-step prediction in smart city settings. This includes the choice of the na-
ture of the covariance functions, the setting of the hyperparameter values of the chosen covariance functions, specifying hyperpriors on the hyperparameters to control the optimisation process, and finally our methods to explicitly model the working-week characteristics in the data.

3.2.1 Choice of Kernel Function

We use periodic kernel functions in this work, since our focus is on time-series data that exhibit some degree of periodicity. Periodic kernels allow the modelling of functions that repeat themselves exactly. The distance between the repetitions can be set by setting the periodicity hyperparameter in the kernel to the required value.

The periodic kernel is defined as:

\[
k_{\text{Per}}(t, t') = \sigma^2 \exp \left( -\frac{2 \sin^2 (\pi \frac{|t - t'|}{p})}{l^2} \right)
\]

where \(\sigma\) is the output variance and \(l\) is the length-scale (just as in the Squared-Exponential kernel), and \(p\) is the periodicity hyperparameter.

For example, if we have data sampled at an hourly frequency, and we wish to enforce a daily repetition structure in the functions drawn from the Gaussian Process, we can achieve this by choosing a periodic kernel and setting its periodicity hyperparameter value to 24. Similarly, we can choose a periodic kernel with a periodicity
value of 168 (i.e. $24^7$) to encode a weekly repetition structure in data sampled hourly.

We investigate periodic kernels with periodicity values set to both 24 (referred to as the ‘daily kernel’) and 168 (referred to as the ‘weekly kernel’). We utilise plain-vanilla versions of these kernels for prediction, as well as more elaborate combinations (Section 3.2.4) to model periodic effects that are more complicated than simple repetition, with one or more of these periodic kernels in the mix. †

### 3.2.2 Setting of Hyperparameters

The optimal values of hyperparameters which effectively model the given time series are found by minimising the *negative* log marginal likelihood (NLML).

The likelihood is, as always, the probability of seeing the data given the hypothesis/parameters, i.e. $p(\mathbf{y}|\mathbf{f}, \mathbf{X})$. The marginal likelihood is found by forming a (probability weighted) sum over all possible function values $\mathbf{f}$, i.e. integrating or marginalising over all possible function values as shown below.

$$
\int p(\mathbf{y}|\mathbf{f}, \mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f} = p(\mathbf{y}|\mathbf{X})
$$

(3.5)

In the Gaussian Process model the prior for functions is a (multivariate) Gaussian,

---

†In the case of non-periodic data a periodic kernel is obviously not the proper choice. Though the choice would be data-dependent, a squared exponential kernel to model a smooth function, coupled possibly with a linear kernel could be more appropriate.
and the likelihood is a factorized Gaussian, which allows the derivation of the log marginal likelihood [85]:

\[
\log p(y|X) = -\frac{1}{2} y^T(K + \sigma_n^2 I)^{-1}y - \frac{1}{2} \log |K + \sigma_n^2 I| - \frac{n}{2} \log 2\pi
\] (3.6)

The choice of the type of kernel, and the hyperparameter values of the chosen kernel, represent the prior on the functions that will be generated. A different set of functions will be generated for different hyperparameter values set in the kernel. The marginal likelihood represents the probability of getting the observations from the set of functions generated for a given hyperparameter setting.

The negative marginal likelihood will be minimised for a given choice of kernel, and the minimising will happen on the hyperparameter values. The minimising of the negative marginal likelihood represents comparing all the hyperparameter values for the chosen type of kernel, and picking the hyperparameter values that would most likely result in functions that approximate the data seen.

However, given that the marginal likelihood objective function is commonly non-convex, this optimisation procedure is sensitive to the initial starting values of the hyperparameters provided. Therefore, different starting values for the hyperparameters were experimented with, along with different numbers of iterations for the optimisation process. Reasonable initial values were found by using a manual grid-
Table 3.1: Initial hyperparameter values chosen

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length Scale</td>
<td>0.03</td>
</tr>
<tr>
<td>Periodicity</td>
<td>24 or 168</td>
</tr>
<tr>
<td>Output Variance</td>
<td>0.1</td>
</tr>
<tr>
<td>ARD Bias $a_1$ and $b_2$</td>
<td>0.9</td>
</tr>
<tr>
<td>ARD Bias $a_2$ and $b_1$</td>
<td>0.1</td>
</tr>
<tr>
<td>Noise Variance</td>
<td>0.01</td>
</tr>
</tbody>
</table>

search-like process. The initial parameter values chosen are given in Table 3.1.

3.2.3 Specifying a Hyperprior

We observed that the parameter values arrived at after optimisation of the initial parameter values did not necessarily turn out to be values which would help reduce prediction error. This was especially apparent in the case of the periodicity hyperparameter, which was ‘tuned’ to values far away from 24 (daily) or 168 (weekly) initially set. However, visual inspection of the time-series plots as well as knowledge of the problem domain gives us strong confidence that the periodicities should indeed be either 24 or 168. Therefore, we decided to encode our strong prior belief by putting a prior over the hyperparameter values (a hyperprior), in this case by clamping the value at either 24 or 168 and preventing the optimisation process from modifying these values.
3.2.4 Modelling Combined Weekday and Weekend Effects

It is frequently required to model daily and weekly variations in a single model. Most data exhibits correlations on a weekly periodicity; for example the values for Wednesday this week would be highly correlated to that of the Wednesday last week. In addition, especially in cases where the response variable values might have different levels in different weeks, the values of recent previous days would carry valuable information.

Naive approach

A naive approach to model both these correlations would be to combine two periodic kernels, one with a daily periodicity and the other with a weekly periodicity using either addition or multiplication of the kernel functions. However, this poses the problem of Mondays being correlated to Sundays and Saturdays being correlated to Fridays. We propose a method to solve the problem of combining periodicities elegantly, and with additional scope for more flexible models.

New features

We enhance the feature representation of the time series as follows. Represent the current representation as \((t, y)\), where \(t\) is time and \(y\) is the corresponding output.
value. We augment this representation with two features which represent whether the date \( t \) is a weekday \((w)\) or a weekend \((n)\), resulting in the representation \((t, w, n, y)\). For example, a value of 554.2 on Monday at 11 am might be represented by \((143, 1, 0, 554.2)\) and a value of 742.1 on Sunday at 10 pm might be \((183, 0, 1, 742.1)\).

Consider two days represented by \((t, w, n, y)\) and \((t', w', n', y')\). The dot product \(w \cdot w'\) would be one if both the days \( t \) and \( t' \) are weekdays, and would be zero if either of the days are weekends. Conversely, considering the dot product \(n \cdot n'\), this would be one if both days are weekends, and zero if either of the days are weekdays. We are able to calculate dot products by utilising linear kernels, which enables the elegant expression of the mathematical operation in terms of kernel functions. For example, \(\text{covLIN}(w, w')\) is equivalent to the dot product between \(w\) and \(w'\) \((w \cdot w')\).

**Kernel combinations**

Now consider the expression:

\[
k(t, t') = \text{covLIN}(w, w') \text{covPer}_{\text{daily}}(t, t') + \text{covLIN}(n, n') \text{covPer}_{\text{weekly}}(t, t')
\]  

(3.7)

It can be seen that the values generated by \(\text{covPer}_{\text{daily}}\) will be activated only when both \(t\) or \(t'\) are weekdays (i.e. \(w = w' = 1\)), and the values generated by \(\text{covPer}_{\text{weekly}}\) will be activated only when both \(t\) or \(t'\) are weekends (i.e. \(n = n' = 1\)). Therefore, scenarios
similar to Fridays being considered correlated to Saturdays will not arise. This simplest example of combining the two kernels with different periodicities essentially provides a switching mechanism between either $covPer_{daily}$ or $covPer_{weekly}$ based on the two dates $t$ or $t'$ under consideration.

More elaborate and flexible models may also be constructed. For example, we are able to provide a configurable level of influence of each kernel function as follows:

$$k(t, t') = (a_1covLIN(w, w') + b_1covLIN(n, n'))covPer_{daily}(t, t')$$
$$+ (a_2covLIN(w, w') + b_2covLIN(n, n'))covPer_{weekly}(t, t') \quad (3.8)$$

In this expression the weights $b_1$ and $a_2$ would be close to zero to represent the low level of influence we desire from the daily kernel on weekends and the weekly kernel on weekdays. What is noteworthy is that we do not need to completely discard one type of kernel, but can instead weigh the kernels according to our requirements.

This method can be equivalently expressed using a linear Automatic Relevance Determination (ARD) kernel, which provides the weighted sum of the outputs of
the regular linear kernels when applied separately on each dimension.

\[
k(t, t') = \text{covLINard}([w; n], [w'; n']) \text{covPer}_{\text{daily}}(t, t') \\
+ \text{covLINard}([w; n], [w'; n']) \text{covPer}_{\text{weekly}}(t, t') \tag{3.9}
\]

This allows the use of the regular optimisation mechanism to tune the weights in accordance with our data.

A further variation on this theme is the expression:

\[
k(t, t') = \text{covLIN}(w, w') \text{covPer}_{\text{daily}}(t, t') + \text{covPer}_{\text{weekly}}(t, t') \tag{3.10}
\]

which prevents the weekly periodic kernel being suppressed to 0 for weekdays, but does so for the daily periodic kernel for weekends.

This model is also further extensible to seamlessly model public holidays to the model with a feature representing whether the date is a public holiday which could be combined with a yearly periodic kernel.
3.3 Experiments and Results

3.3.1 Datasets

The data used for electricity load forecasting comes from smart meters installed in the Parkville campus of the University of Melbourne\(^\text{‡}\). This data consists of the electricity load recorded in kWh at each point in time in the smart meters of 21 buildings. The data is available at a granularity of 15-minutes, which was aggregated to be hourly data. We perform 24-hour ahead predictions of the electricity load values in each of the 21 buildings.

The second dataset we used was a pedestrian data set was obtained through the Open Data initiative of the City of Melbourne\(^\text{§}\). The data is the output of a 24-hour system which monitors pedestrian movement at key locations in Melbourne, Australia and provides hourly pedestrian counts for each day. We use our methods to perform 24-hour ahead predictions of pedestrian counts at 10 key locations in the City of Melbourne.

\(^{\text{‡}}\)http://sustainablecampus.unimelb.edu.au

3.3.2 Error Metric

We use as our metric the mean absolute error normalised by the average magnitude of the actual load / prediction count values for the prediction period, and expressed as a percentage. The normalisation step is required to compare error rates between different buildings / sensors, which would have different levels of electricity usage / pedestrian counts. In general form this metric (Mean Error Relative to $\bar{y}$ - MER) is expressed as follows:

$$\text{Error(MER)} = 100 \cdot \frac{1}{N} \sum_{b=1}^{N} \frac{|\hat{y}_b - y_b|}{\bar{y}}$$  \hspace{1cm} (3.11)

where $\hat{y}_b$ is the predicted value at hour $b$, $y_b$ is the actual value at hour $b$, $\bar{y}$ is the mean consumption in the period considered (a day in this context), and $N$ is the number of hours predicted (24 in this scenario).

3.3.3 Experimental Setup

All models discussed were implemented using the GPML Matlab Toolbox\footnote{http://www.gaussianprocess.org/gpml/code/}. Initial hyperparameter values were set to values that were observed to provide reasonable prediction accuracy and did not result in pathological cases. These values are out-
lined in Table 3.1. Hyperparameter optimisation was done via conjugate gradient ascent on the log marginal likelihood function with a maximum iteration limit of 100. Predictions were made for the month of June 2014 for the electricity load forecasting data and the month of October 2016 for the pedestrian forecasting with two weeks of data used for training.

We also compare the results of Gaussian Process Regression with ARIMA, which is the most widely-used forecasting technique based on regression. The Box-Jenkins methodology was used in the tuning of parameters, which resulted in an ARIMA(1,1,0) (number of time lags, degree of differencing and order of moving average model respectively) model being selected.

It is important to note that the kernel composition methods do not make use of new features - the GPR models still operate on the same univariate time series data. The kernel composition methods enable the modelling of increasingly elaborate priors that model the data more closely. However, this close modelling is still based on the same data as is used in other models used for comparison. Therefore, the comparison with other models is appropriate since all models use the same features. The Gaussian Process Regression models have simply been enhanced to have priors that more closely model the data - they do not have any advantage in terms of feature values.
3.3.4 Results

*Periodic kernels*

We begin our exploration with a periodic daily kernel. However, as we see in Table 3.2 the daily kernel does not provide us with an acceptable degree of prediction accuracy. One reason for this low prediction accuracy is that there is not much of a daily signal in the data. Further, just using a daily kernel means that the function is constrained to have a constant pattern across weekends and weekdays, as well as across weeks that may have different value levels. Therefore, this low accuracy is unsurprising. It is interesting to observe, even in this case, that adding a hyperprior helps with accuracy, providing notable improvements in accuracy especially for the campus smart meter dataset (error metric improvement from 36.05% to 17.12% for daily kernel).

We observe better accuracy with the use of a periodic kernel with a weekly periodicity (36.05% vs 17.12%), and much better accuracy when the weekly kernel is coupled with a hyperprior (17.12% vs 9.33%). This agrees with our intuition that this kind of data will have a strong weekly signal. The weekly kernel with the hyperprior also turns out to be the simplest model that improves on the predictive accuracy of the benchmark ARIMA model.
### Table 3.2: Error rates for different kernel combinations

<table>
<thead>
<tr>
<th>Kernel Combination</th>
<th>Smart Meter Error (MER)</th>
<th>Pedestrian Count Error (MER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daily</td>
<td>36.05</td>
<td>72.63</td>
</tr>
<tr>
<td>Weekly</td>
<td>22.10</td>
<td>63.65</td>
</tr>
<tr>
<td>Daily with Hyperprior</td>
<td>17.12</td>
<td>70.30</td>
</tr>
<tr>
<td>Weekly with Hyperprior</td>
<td>9.33</td>
<td>17.98</td>
</tr>
<tr>
<td>Daily<em>LIN + Weekly</em>LIN (Combination I)</td>
<td>8.73</td>
<td>20.68</td>
</tr>
<tr>
<td>Daily*LIN + Weekly (Combination II)</td>
<td>9.26</td>
<td>16.53</td>
</tr>
<tr>
<td>Daily<em>LINard + Weekly</em>LINard (Combination III)</td>
<td>9.06</td>
<td>18.29</td>
</tr>
<tr>
<td>ARIMA</td>
<td>16.40</td>
<td>40.11</td>
</tr>
</tbody>
</table>

### Figure 3.1: Campus Smart Meter - Output Variance Hyperparameter

### Figure 3.2: Pedestrian - Output Variance Hyperparameter
Adding hyperpriors

It can be seen in Table 3.2 that adding a hyperprior improves accuracy drastically in all the combinations under consideration. Especially in the case of pedestrian count prediction, it is seen that when coupled with a hyperprior the use of the weekly kernel in place of the daily kernel provided a very substantial improvement. This could be attributed to the fact that the pedestrian count data has even less of a daily periodicity compared to the smart meter data. Further, fixing some parameters reduces the search space and makes it more likely that good values are found for the parameters being optimised over.

Figures 3.1-3.4 show the accuracy changes across a range of parameter values, which vary across a number of orders of magnitude, when predicting with and without using a (hyper)prior on the periodicity hyperparameter. It is seen that the prediction accuracy is significantly better when using a hyperprior across all parameter values. This is true regardless of the dataset or the hyperparameter being varied.

Further, it is worth noting that in most cases, even the parameter values that yield the worst accuracy when used with a hyperprior are still better or close to the level of accuracy gained with the hyperparameter values that perform best without using a hyperprior. For example, in campus dataset, the absolute worst prediction accuracy using a hyperprior is 22.35%, compared with the absolute best prediction accuracy
without using a hyperprior being 19.31%.

**Modelling weekday and weekend effects**

Our initial attempt of modelling weekday and weekend effects was Combination I:

\[
k(t, t') = \text{covLIN}(w, w')\text{covPer}_{\text{daily}}(t, t') + \text{covLIN}(n, n')\text{covPer}_{\text{weekly}}(t, t')
\]

It is seen in Table 3.2 that this combination results in a noticeable improvement in the electricity load prediction scenario, but not so for the prediction of pedestrian counts. This can be attributed to the fact that the pedestrian count dataset has a strong weekly signal, both for weekdays as well as for weekends. The Combination I model forces the weekly kernel to zero for weekdays, which prevents the modelling...
of the weekly effect for weekdays.

We therefore modified the model specification to Combination II:

\[
k(t, t') = \text{covLIN}(w, w')\text{covPer}_{\text{daily}}(t, t') + \text{covPer}_{\text{weekly}}(t, t')
\]

(3.12)

which prevents the forcing to zero of the weekly kernel for weekdays (i.e. the weekly kernel is always used in the mix) while still allowing the separate modelling of weekdays and weekends. This modified model specification resulted in an improvement in performance as seen in Table 3.2 for the pedestrian count prediction. This is in line with our previous observation where the pedestrian counts had a high weekly periodic signal compared to the daily signal. This model allows us to utilise the weekly periodicity in the prediction of weekdays as well, which is what would have resulted in the increased accuracy.

The model specification in Combination III:

\[
k(t, t') = \text{covLINard}([w; n], [w'; n'])\text{covPer}_{\text{daily}}(t, t') + \text{covLINard}([w; n], [w'; n'])\text{covPer}_{\text{weekly}}(t, t')
\]

allows the setting of relative weights for the daily and weekly kernels. Given that this model specification would not set the kernels to exactly zero or one, and given the
strong weekly signal present in both datasets under consideration, it is unsurprising
that the accuracy of this model is between the Combination I and Combination II
models for both datasets. This version of the model would be most useful in a dataset
which might have varying daily and weekly signal strengths for weekends and week-
days.

Other observations

For all the kernel combinations discussed above, experiments were run with and with-
out adding a noise kernel to the combination. Adding a noise kernel did not have
any significant effect. Further, though the product of a squared exponential kernel
with a periodic kernel would result in a recency effect being modeled along with the
periodic effect, we did not observe improvements in prediction accuracy. This could
be attributed to the modeled recency effect not being significant enough to yield pre-
diction accuracy gains, especially when burdened with the need to now optimise over
a higher number of hyperparameters.

3.4 Conclusions

We provide multiple mechanisms to model the weekend and weekday effects in pe-
riodic time series data using Gaussian Processes. These include methods that enable
an intelligent switching mechanism between daily and weekly periodic kernels, as well as methods that learn the weighting between kernels through the data itself. We also provide methods to constrain the optimisation process by adding hyperpriors to improve convergence.

Our results indicate significant improvements when hyperpriors are introduced to both daily and weekly kernels. These improvements were seen across multiple hyperparameter values, indicating the robustness of this approach and its applicability to a wide range of scenarios where there exists strong confidence about periodicity. Further improved predictions were observed using mechanisms introduced to model daily and weekly periodic kernels in concert. We also observed improved prediction accuracy using the methods which allowed automatic tuning of weights through learning the weights through data, which is particularly promising as this further automates the forecasting process. All results were obtained on two real-world publicly available datasets with widely different applications. This is highly suggestive of the potential generalisation of the methods to other use cases with time series data with similar working-week structure.

Encoding the spatial relationships between sensors was briefly explored though the results were not encouraging. It was seen that spatial information slightly improved accuracy when the sensors were very close to each other (e.g. Bourke Street
South vs Bourke Street North, which are two sensors that are across the street from each other), though other spatial relationships did not yield any consistent accuracy improvements (e.g. sensors that were only a block away from each other did not improve prediction accuracy).

Gaussian Process Regression, though providing high levels of accuracy and flexibility, suffers from scalability issues due to being of complexity class $O(n^3)$. It is unavoidable, therefore, that our elaborate models (especially the ones with automatic tuning of weights) will also be affected. As a solution to this limitation, approximation methods for Gaussian Processes may be used which provide a trade-off between accuracy and scalability. Of such approximation methods, reduced rank approximations [107] [95] [37] are commonly used. Reduced rank approximation methods assume that the covariance matrix is of a lower rank than the number of rows (and columns) of the matrix in order to speed up the matrix inversion process. The accuracy of the predictions that are the result of a reduced rank approximation technique will hinge on the degree to which this approximation holds, which would depend on the specific dataset the method is applied to. A survey of other approximation methods for large datasets can be found in [85].
Bayesian Forecasting: Improved Accuracy with Low-quality Data

In real-world systems such as cities with a network of smart sensors, perfect data is a rarity. Smart city initiatives involve sensors which are spread across a metropolis, and which are exposed to the elements and possible vandalism, which results in low qual-
ity and especially missing data being a common problem in this space. These data quality issues have a significant impact on accuracy in prediction scenarios, where the difficult task of extrapolation needs to be performed, and is especially true in relation to the even harder task of multi-step prediction. Therefore, missing data points in training data is a challenge that needs to be addressed when attempting to forecast in real-world scenarios to maintain prediction accuracy.

The work in this chapter shares the overarching goal of improving prediction accuracy similar to the previous Chapter 3, but in this case we aim to provide methods that explicitly address the challenges associated with unclean data. Though much work exists that deals with prediction in smart city scenarios, these almost invariably work under the assumption of clean data. Of the work that addresses the low quality data problem, [51] does tackle the problem of missing data, though relating to logistic regression, with [67] addressing the problem in sparse linear regression. In [97], which uses Gaussian Process Regression for heart rate data modelling, only the problem of noisy data, and not that of missing data, is addressed.

In order to make prediction systems robust to missing training data two approaches may be considered:

1. Do the best that we possibly can with the data that we already have at our dis-
posal. That is, we make do with the data with missing values. This calls for effective prediction methods. Incorporating prior belief can help these prediction methods to become more robust in the face of incomplete data.

2. Use other, related data. This is the approach taken in this chapter. Though we may not have complete data for a particular sensor, given that there will be a large number of sensors in the sensor network, there will be sensors with more complete data which have similar behaviour to the sensors which have incomplete data.

A naive method of using this related data is to use it to supplement the missing values of the incomplete data sensor. This indeed is a legitimate, albeit naive, method of attacking the problem. We do look into this straightforward method of data replacement and compare it to the superior method we propose in this chapter to address the problem of low quality data.

In the smart city space, though we have the aforementioned challenge of missing data, we also have the opportunity of a large number of sensors. Given the diverse and large number of sensors in a typical smart city setting, it is likely that there would exist sensors with similar behaviour. The models that are built for these sensors with similar behaviour could potentially be improved if knowledge from similar sensors could be incorporated to improve generalisability. More importantly, in the context
of this chapter, such knowledge transfer between sensors could also help in the situation where a particular sensor had data missing. For example, the loss in accuracy due to missing training data in a particular sensor could be mitigated with methods which allowed learning a joint model with a related sensor which had a more complete set of the data.

This need for knowledge transfer between sensors lends itself well to the concept of multi-task learning. Multi-task learning enables knowledge transfer between different related tasks. These tasks in the smart city scenario would be different sensors. Improving the predictive performance of a task by utilising the knowledge from other tasks has been explored in the literature in applications as diverse as personalised age estimation [114], modelling of robot dynamics [105], and learning effects of drug interactions for HIV therapy scanning [19]. Our work also improves predictive performance by utilising knowledge from highly correlated sensors. However, in contrast to prior work, we also focus on and demonstrate the ability to improve predictive accuracy in the face of missing data using the multi-task learning methods we propose for Gaussian Processes.

Our contributions with respect to using related data to mitigate the effects of missing training data on prediction accuracy are listed below:

1. We provide a multi-task learning method which minimises the effect of missing
data on multi-step prediction accuracy.

2. We provide an equivalent expression of this multi-task learning formulation as a combination of commonly used kernels in Gaussian Processes. This will allow straightforward implementation of the multi-task learning scheme using popular machine learning toolkits.

3. We demonstrate how to perform multi-step prediction effectively using Gaussian Process Regression and Support Vector Regression. In particular, we provide mechanisms to perform multi-step prediction on data with periodic behaviour without propagating errors between forecasting steps.

4. We validate our methods on real-world and publicly available datasets and demonstrate significant improvements over competitive baselines and other comparable methods.

4.1 Gaussian Processes - Multi-task Learning

Multi-task learning [22] involves improving the performance of a predictive task by using information in other related tasks. In our context, we will be using the information available in highly correlated sensors in order to improve the prediction accuracy of sensors which have data missing. It should be noted that a task in the
general sense of multi-task learning would correspond to an individual sensor in the smart city environment.

Given an input $x$, GP regression assumes that the response variable values $y$ are the result of the evaluation of a latent function plus the addition of a white noise component $\varepsilon$. (i.e. $y = f(x) + \varepsilon$). We may consider multi-task learning to be learning multiple latent functions $f_m(x)$ for each task $m \in 1, \ldots, M$. These functions can be considered to be generating the response variable values for the input values available for a particular task. For multi-task learning to be effective, the values of these functions should be correlated for the same input values. That is, there should be similarities enforced between the function values $f_m(x)$ and $f_{m'}(x)$, where $m$ and $m'$ are different tasks (sensors). This is achieved by developing shared representations across related tasks.

Multi-task learning has a resemblance to the popular field of transfer learning. Though similar, it should be noted that the two techniques are different and solve different problems. Transfer learning is commonly used in problems for which 1) there does not exist much training data and 2) there exists a closely related problem which has a large amount of training data. Transfer learning can be achieved by first (pre)training a model for the related problem using the large amount of training data available. Then this pretrained model can be adapted to the actual problem by fur-
ther training on the (small amount of) training data of the actual problem.

Transfer learning involves a sequential process where the knowledge learned in the first task (usually large amounts of training data) is transferred to a second task (small training data). For example, transfer learning is popular in deep learning contexts for image processing and natural language processing tasks where neural networks trained on massive amounts of data are made available by large research organisations [50][81]. These pretrained networks can then be further trained with the training data for the specific image processing or natural language processing task at hand. In contrast, multi-task learning learns one common joint model using all data in related tasks. This is done in the hope that learning the common model will help the model performance for all tasks since knowledge can be shared between tasks. We demonstrate how we model the correlations between the functions of the different tasks in Section 4.2.3.

4.1.1 Support Vector Regression

Support Vector Machines are a popular method used for classification and regression purposes with good theoretical guarantees. Regression using Support Vector Machines is a highly competitive contemporary method which is used for comparison purposes in this work.
Given training data \( \{(x_i, y_i), \ldots, (x_l, y_l)\} \subset \mathbb{R}^d \times \mathbb{R} \), where \( d \) is the number of features, the goal is to find a function \( f \) that provides a mapping between the feature values and the target values.

\[
f(x) = w \cdot x + b \quad \text{with} \quad w \in \mathbb{R}^d, b \in \mathbb{R}
\]  

(4.1)

In Support Vector Regression, the goal is to find a function that produces values that have at most \( \varepsilon \) difference from the actual values, while being as ‘flat’ as possible. That is, we do not care about the errors as long as they are less than \( \varepsilon \). The objective is to minimise \( w \) (be as flat as possible). This objective can be realised by minimising the norm of \( w \). The problem can be formulated as a convex optimisation problem as shown below [96]:

\[
\begin{align*}
\text{minimise} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad y_i - w \cdot x_i - b \leq \varepsilon \\
& \quad w \cdot x_i + b - y_i \leq \varepsilon 
\end{align*}
\]

(4.2)

In this work, the Support Vector Regression (SVR) models are optimised using the widely used method of Sequential Minimal Optimisation[82]. Separate SVR models are built for each of the steps (i.e. hours) in the multi-step prediction (Indepen-
dent Value Prediction[27]) in order to prevent error propagation.

4.2 Methodology

This section outlines our multi-task learning method which enables accurate multi-step prediction with missing sensor data using Gaussian Process Regression.

4.2.1 Incorporating prior knowledge

Data relating to smart cities tend to have an inherently periodic nature which relates to the working week. This weekly periodicity is observed in the data under consideration (electricity smart meter data and pedestrian count data) as well. To incorporate this prior knowledge of the periodic structure of the data into the model, we choose to use a periodic kernel as our base kernel function. The base kernel $k_{\text{data}}(t, t')$ relates to the modelling of the functions for each individual sensor separately.

The periodic kernel has the form given below:

$$k_{\text{Per}}(t, t') = \sigma^2 \exp \left( -\frac{2 \sin^2(\pi \frac{|t - t'|}{l})}{l^2} \right)$$

(4.3)

where $\sigma$ is the output variance and $l$ is the length-scale (just as in the Squared-Exponential kernel), and $p$ is the periodicity hyperparameter. Therefore, the base kernel $k_{\text{data}}$ takes
the form $k_{\text{per}}$ given in Equation 4.3.

The periodicity hyperparameter allows us to specify our prior belief of weekly periodicity. If we have data that has been sampled each hour, we can choose to set the periodicity hyperparameter $\rho$ to 168 (i.e. $24 \times 7$) in order to draw functions which have weekly periodicity. The actual function will be learnt from the data, subject to this constraint of periodicity. Encoding this prior belief of periodicity greatly helps in minimising error propagation when performing multi-step prediction.

4.2.2 Representation of data for inductive knowledge transfer

We require a representation to enable the transfer of information from a related sensor which has complete information to a sensor which has training data with missing values. We provide a novel multi-task learning method to provide such a representation to allow inductive knowledge transfer between models. We describe our proposed representation of this joint model in the form of a composite kernel in this section.

Multi-task learning may be either heterotopic or isotopic in nature[8]. In isotopic learning, each data point would be associated with a vector of outputs, the length of which would be equal to the number of tasks. For example, if we were performing multi-task learning over three sensors, a given data point $x_i$ (e.g. a time instant) would
relate to a vector of length three \(< y_1, y_2, y_3 >\). However, it should be noted that in our scenario where we are dealing with missing values, we might not have values for some sensors at a given point in time (i.e. for any given data point \(x\), we might be missing some \(y_i\) values.)

Therefore in this work we follow a heterotopic\(^8\) approach to multi-task learning. In this approach, each data point is augmented by a task identifier \(d\). Such augmented data points would now be paired with one \(y\) value. For example, consider the same example as earlier, but now with the \(y\) value for sensor 2 (\(y_2\)) is missing. We may represent the data we have available to us as \(< x_i, d_i, y_i >\) and \(< x_i, d_j, y_j >\). Once the data is expressed in this manner, it allows us to integrate the data into a Gaussian Process model which will allow us to run standard inference algorithms.

### 4.2.3 Multi-task kernel

In addition to providing a method as detailed above to represent multi-task data with missing values, we also need a mechanism to encode knowledge transfer between the different tasks (i.e. the core multi-task learning model). We propose a separable multi-task kernel \([9, 20]\) of the following form:

\[
k((x, d), (x', d')) = B_{dd'} \otimes k_{\text{data}}(X_0, X_0)
\]  \hspace{1cm} (4.4)
where $\otimes$ is the Kronecker product. The construction of $B$ can enforce no correlation, anti-correlation or positive correlation between tasks. We propose a formulation of $B$ which allows for configurable weighting of these task dependencies. This would relate to a weighted combination between a fully independent learning, (where each task is considered separate from each other and no data-based transfer happens), and of a pooled type of learning (where each task is considered equivalent to each other and the task identity is ignored.) This is possible to be achieved through setting $B = 1 + kI$, with the all-ones matrix added to the product of $k$ and $I$, where $I$ is the identity matrix and $k$ is a configurable hyperparameter.

Figure 4.1 graphically illustrates this operation. The first matrix (grayscale) represents $B$ above, with the coloured second matrix representing $k_{data}$. White represents larger values, while gray values are closer to zero. The diagonal elements represent the intra-task dependencies, and the off-diagonal values represent inter-task dependencies. In our weighted representation, the off-diagonal (gray) values will be small, but non-zero, representing non-zero information transfer between tasks. In the final re-
sult matrix, it is seen that the intra-task blocks are preserved. The off-diagonal blocks are a weighted, less strong version of the original $k_{data}$, representing the information transfer between tasks.

A numerical example of the operation is shown below:

\[
\begin{bmatrix}
5 & 1 \\
1 & 5
\end{bmatrix}
\otimes
\begin{bmatrix}
112.5 & 70.2 & 20.1 \\
70.2 & 112.5 & 18.1 \\
20.1 & 18.1 & 112.5
\end{bmatrix}
\]

\[
\begin{bmatrix}
5 \ast 112.5 & 5 \ast 70.2 & 5 \ast 20.1 & 1 \ast 112.5 & 1 \ast 70.2 & 1 \ast 20.1 \\
5 \ast 70.2 & 5 \ast 112.5 & 5 \ast 18.1 & 1 \ast 70.2 & 1 \ast 112.5 & 1 \ast 18.1 \\
5 \ast 20.1 & 5 \ast 18.1 & 5 \ast 112.5 & 1 \ast 20.1 & 1 \ast 18.1 & 1 \ast 112.5 \\
1 \ast 112.5 & 1 \ast 70.2 & 1 \ast 20.1 & 5 \ast 112.5 & 5 \ast 70.2 & 5 \ast 20.1 \\
1 \ast 70.2 & 1 \ast 112.5 & 1 \ast 18.1 & 5 \ast 70.2 & 5 \ast 112.5 & 5 \ast 18.1 \\
1 \ast 20.1 & 1 \ast 18.1 & 1 \ast 112.5 & 5 \ast 20.1 & 5 \ast 18.1 & 5 \ast 112.5
\end{bmatrix}
\]

4.2.4 Formulation of multi-task kernel as linear kernel combination

We provide the following kernel combination expression which will allow the straightforward implementation of the multi-task kernel in Equation 4.4 in machine learning toolkits.

\[
[k_{\text{Conc}}(t, t') + k_{\text{Linear}}(t, t')] \ast k_{\text{Per}}(t, t')
\]  \hspace{1cm} (4.5)
In this equation, $k_{\text{Const}}(t, t')$ is a kernel function which produces a covariance matrix with a constant value in all elements. The value of the constant is controlled by a hyperparameter, which is set to one to emulate the function of the identity matrix in the original expression for $B$. The input data points are augmented with the task identity using a one-hot encoding of the tasks, over which $k_{\text{Linear}}(t, t')$ operates. $k_{\text{Per}}(t, t')$ is the regular kernel function on the input data (which has the form given in Equation 4.3.) This formulation provides the exact covariance matrix as that output in Equation 4.5. Proof of the equivalence of our kernel combination to the Kronecker product in Equation 4.5 is provided in the Appendix.

In summary, this kernel combination allows us to specify a separable multi-task kernel which allows for the weighted transfer of information between tasks. By changing the value of the hyperparameter $k$, we are able to control how similar the model believes the tasks to be, and therefore how dependent the predictions of one task will be on the other.

4.3 Experiments

We use the same datasets and error metric as used in Chapter 3, of which we provide a brief introduction here as well for the sake of completeness.
4.3.1 Datasets

The pedestrian dataset used in the experiments is a publicly available dataset made available through the Open Data initiative of the City of Melbourne*. It consists of hourly counts of pedestrians at key locations in the city, recorded by sensors which are operated 24 hours a day.

The smart meter data is also a publicly available dataset of the electricity usage of buildings in the Parkville campus of the University of Melbourne†. The data is available at a granularity of 15-minutes, which were aggregated to be hourly data.

We perform 24-hour ahead (multi-step) predictions of the electricity load values in each building, and pedestrian count values at each sensor. We use methods developed in this work to mitigate the effects of missing values in the smart meter and pedestrian count sensor data.

4.3.2 Error metric

The error metric used is the same as used in Chapter 3, of which a description we include here as well for the sake of completeness.

The error metric used is the mean absolute error, which we normalise by using

†http://sustainablecampus.unimelb.edu.au
the average magnitude of the actual load / prediction count values for the prediction period. This ratio is expressed as a percentage to arrive at the final metric value. This metric (Mean Error Relative to $\bar{y}$) has the following form:

$$\text{Error}(MER) = 100 \cdot \frac{1}{N} \sum_{h=1}^{N} \frac{|\hat{y}_h - y_h|}{\bar{y}}$$

(4.6)

where $\hat{y}_h$ is the predicted value at hour $h$, $y_h$ is the actual value at hour $h$, and $\bar{y}$ is the mean consumption in the period considered (a day in this context), and $N$ is the number of hours predicted (24 in this scenario).

4.3.3 Experimental setup

Multi-step forecasting was done for the time series datasets mentioned in Section 4.3.1, both in the case of complete data and missing data. Traditional Gaussian Process Regression and Support Vector Regression were used as the base models.

All preprocessing and model building was done using Matlab. The Gaussian Process Models were implemented using the GPML Toolbox‡, and the Support Vector Regression models were built using methods provided in the Statistics and Machine Learning Toolbox of Matlab.

‡http://www.gaussianprocess.org/gpml/code/
Preprocessing of data for multi-task learning

Since models need to be learnt jointly between sensor data, the sensor data of different sensors needs to be comparable. Therefore, the training data of each sensor is normalised to lie between zero and one. The predictions output by the models will also lie on this scale, which will be scaled up by the same factor post prediction to arrive at the final prediction values.

Choosing sensors for combining

We calculate the cross-correlation values between each pair of sensors. We then select pairs of sensors with cross-correlation values greater than 0.95 as candidates for combination through multi-task learning. For each sensor, predictions were made in combination with each of the other sensors which had a higher than 0.95 cross-correlation with it. We provide the mean of the prediction errors of all such combinations, as well as the confidence intervals of the prediction errors in the results.

Multi-step predictions

For each sensor, in each dataset, we aim to make predictions of the time series for the next 24 time steps. That is, our goal is to predict the pedestrian counts at each sensor for the next day in the pedestrian dataset, and to predict the electricity load
in each building for the next day in the smart meter dataset. The predictions for the pedestrian sensors were made for the days in the month of October 2016, and those for the electricity load data based on the days in the month of June 2014 with two weeks of data being used for training the models.

These multi-step predictions are made for two scenarios: 1) the scenario with complete data, and 2) the scenario with missing values in the data. The missing values for scenario 2 are generated through the error generation process outlined in Section 4.3.3.

*Error generation process*

In order to perform an accuracy evaluation we need to simulate the process of error generation. This is required to have ground truth so that we can compare the prediction accuracy in the scenario with errors against the scenario without errors.

Errors occurring in real-world scenarios tend to either occur as once-off individual errors, or may manifest themselves as bursty errors where the system suffers from a continuous length of errors. We model this real-world error generation process by a Markov chain with two states. State 1 corresponds to generating random solitary errors, with State 2 representing the bursty error generation process.

Since we require a long length of errors in bursty errors, the transition probability
Random errors start with $P(error) = 0.05$

Bursty errors start with $P(error) = 1$

Figure 4.2: Markov chain error generation process

from State 2 to itself is set to a high value (i.e. once a bursty error is generated, the system will continue to generate errors with high probability until the system transitions to State 1). The error creation probability while in State 2 is also set to a high value to enable continuous error generation.

Random errors are generated while the system is in State 1. The errors will be generated with a low probability since these are solitary once-off errors. The system may transition to State 2, which would create bursty errors as mentioned above, with low probability.

Figure 4.2 illustrates the Markov chain error generation machine described above. An example run of errors generated through this two-state Markov chain is shown in Figure 4.3, where a one denotes the presence of an error and a zero represents a correct value.
4.3.4 Techniques used to make predictions

**Multi-step prediction with complete data**

For the scenario with complete data, we perform prediction using three different methods: 1) Traditional Gaussian Process Regression, 2) Our multi-task learning method to mitigate missing data, and 3) Support Vector Regression (for comparison).

**Multi-step prediction with missing data**

As in the scenario with complete data described above, we perform prediction using conventional Gaussian Process Regression, our multi-task learning approach and Support Vector Regression in the missing data scenario as well.

Since missing data is the primary focus of this work, we also use a stronger baseline technique for comparison. In this technique, we replace the missing values of the sensor with values from a highly correlated sensor, chosen as described in Section 4.3.3. We then run both the Gaussian Process Regression and Support Vector Regression.
4.4 Results

4.4.1 Prediction accuracy

We present the results for the smart sensor datasets under consideration in two tables. Table 4.1 presents results for perfect data, when no data removal has been performed. Table 4.2 presents the results for predicting using training data which has undergone the data removal process described in Section 4.3.3.

**No data removal**

In Table 4.1 we observe comparable performance in both the plain-vanilla Gaussian Process Regression (GPR) and Support Vector Regression (SVR) models. This illustrates that the models are indeed comparable since they provide similar levels of
accuracy on the datasets.

A further observation that can be made in Table 4.1 is that prediction accuracy improves when our multi-task learning method is used even in the complete data scenario. It is seen that GPR with multi-task learning improves accuracy from 8.77% to 7.49% for the smart meter data and from 16.28% to 16.12% in the pedestrian data. These improvements would be due to the improved generalisability of the model due to the transfer of information from a correlated sensor.

**Data removal - prediction using traditional techniques**

The results obtained in the scenario with missing data is given in Table 4.2. It can be seen that the plain vanilla GPR suffers significantly in accuracy compared to the full data case when data is removed. We observe a 31% loss in accuracy in the smart meter data set and a 26% loss in accuracy in the pedestrian data set compared to the no data removal scenario discussed in the previous section. This loss in accuracy is

<table>
<thead>
<tr>
<th>Technique</th>
<th>Smart Meter Error Rate</th>
<th>Pedestrian Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR</td>
<td>12.71 ± 2.34</td>
<td>21.98 ± 1.64</td>
</tr>
<tr>
<td>GPR with data replacement</td>
<td>10.62 ± 0.66</td>
<td>21.23 ± 0.96</td>
</tr>
<tr>
<td>Multi-task GPR</td>
<td>8.38 ± 0.58</td>
<td>18.28 ± 0.79</td>
</tr>
<tr>
<td>SVR</td>
<td>11.48 ± 0.79</td>
<td>43.27 ± 2.48</td>
</tr>
<tr>
<td>SVR with data replacement</td>
<td>11.05 ± 0.71</td>
<td>26.49 ± 1.16</td>
</tr>
</tbody>
</table>
unsurprisingly seen in the SVR results as well.

These similar results for both GPR and SVR illustrate that the loss in accuracy is primarily due to the low quality of the training data provided to both models. They also further demonstrate the significant impact missing values have on extrapolation accuracy.

*Data removal - prediction using traditional techniques with data replacement*

This method serves as a stronger baseline for the data removal case. We observe reasonable improvements to the prediction accuracy when the missing values are filled with the values obtained at the same time in the other highly correlated sensor.

The values in Table 4.2 show that the prediction accuracy improves in both models, though the error rates are still not very close to those that we obtained in the no data
removal scenario. (for example, for the smart meter data the error rate for GPR with data replacement is 10.62%, which is lower than the 12.71% for vanilla GPR, but still higher compared to the 8.77% achieved when we had full data). Therefore, we see that this naive technique to mitigate missing data does provide us with a reasonable improvement and thus serves as a competitive baseline.

\textit{Data removal - prediction using multi-task Gaussian Processes}

It is seen in Table 4.2 that the best improvement in accuracy by a considerable margin is provided by our multi-task learning method proposed in this work. We observe that the error rates are now far closer to the error rates with full data. In the case of the smart meter data set the error rate for the data removal scenario (8.38%) is even less that for the full data vanilla GPR (8.77%) (though obviously not lower than full data multi-task learning (MTL) (7.49%)). Therefore we see the effectiveness of GPR with multi-task learning both in comparison to the baseline of a simple replacement of data and also against the comparative technique of SVR.

\textit{Per sensor prediction accuracy}

We provide in this section a more granular view of the relative accuracies of the techniques used in the data removal scenario. Error rates for each sensor using each modelling technique have been graphed in Figures 4.4 and 4.5 for the smart meter and
pedestrian datasets respectively.

In Figure 4.4 (smart meter data), it is seen that except for a handful of cases that SVR is by far the weakest performer of the group for all sensors. However, it also benefits significantly by being supplemented by data replacement, with the error rates going down to be comparable, though slightly higher than that of GPR with replacement. The benefits to GPR through replacement are less clear cut, with accuracies higher and in some cases slightly lower than vanilla GPR across sensors. GPR with multi-task learning is seen to be the strongest performer, outperforming all models consistently and by a considerable amount.

In Figure 4.5 (pedestrian data) we see a similar trend, only that in this dataset SVR performs more poorly compared to GPR, and gets an even bigger boost in accuracy through the data replacement technique.

4.4.2 Parameter sensitivity

Figure 4.8 depicts the change in error rate with changing $k$. As indicated in Section 4.2.4, $k$ is the hyperparameter that specifies the degree of the transfer of learning between sensors. A larger $k$ would mean that a higher weight would be given to the data of the sensor for which predictions are being made, and lower weight for the sensor of which the data we are ‘combining’ to help with the prediction. It can be
Figure 4.5: Pedestrian data per sensor comparison of errors

Figure 4.6: Multi-task GPR - % improvement over next best techniques with increasing burst length (smart meter data)

Figure 4.7: Multi-task GPR - % improvement over next best techniques with increasing burst length (pedestrian data)
seen that for both datasets that there is an initial slight improvement in the accuracy (seen as a decrease in error rate in the graph) with increasing $k$, which then remains stable. We see that the error rate is not sensitive to the parameter $k$ in all scenarios, which allows for the use of the technique without undue focus needing to be made on picking an optimised value of $k$.

### 4.4.3 Computational cost

We illustrate the relative computational cost of each modelling technique as used on the low quality data in Figure 4.9. When comparing the traditional models, it is seen that SVR incurs a much higher computational cost compared to GPR. It can be seen that the data replacement step does not cause much increase in computation time in
either GPR or SVR. GPR with multi-task learning does suffer from a reasonably higher cost compared to conventional GPR. However, given its superior prediction accuracy we feel that this increased cost is well justified.

4.4.4 Different error combinations

We investigate the accuracy of the multi-task GPR method for different error schemes in Figures 4.6 and 4.7.

The figures show the percentage improvement over the next best methods to multi-task GPR against increasing error burst length. These next best methods as was seen in Table 4.2 are traditional GPR with data replacement and traditional SVR with data replacement. The actual values on the x axis represent the probability of transitioning back to the burst error state Markov chain. As this probability increases, so does the burst rate. Therefore greater values on the x-axis represent highly bursty errors, and lower values represent a comparatively higher proportion of random errors.

It can be seen that the percentage improvement in all cases is above zero. Therefore we can clearly see that multi-task GPR outperforms the next-best methods (GPR with replacement and SVR with replacement) comfortably. We also observe a downward trend in the percentage improvement gained using multi-task GPR with in-
creasing burst length. This suggests that a higher relative accuracy improvement may be gained in cases of lower burst length. However, it can still be seen that even at the highest burst lengths (corresponding to a transition probability of around 90%) that multi-task GPR still outperforms the other methods by considerable margins.

4.5 Conclusions

In this chapter we address the challenges related to the real-world problem of multi-step prediction using low quality training data with missing samples.

To tackle the problem of propagating errors when performing multi-step prediction, methods are provided for both Gaussian Process and Support Vector Regression models. These include encoding prior information of periodicity (GPR) and utilising independent value prediction (SVR).

To resolve the problem of missing training data samples, we develop a multi-task learning method which allows the transfer of learning from other related sensors which have more complete data. Our methods are based on a heterotopic representation of data and a separable multi-task kernel which allows a configurable level of influence of intra-task and inter-task dependencies. We express this multi-task learning formulation as a combination of commonly used Gaussian Process kernels to enable straightforward implementation using popular machine learning toolkits.
We evaluate our methods using publicly available real-world datasets related to smart cities. In order to have ground truth for evaluation of methods, missing data was generated using a Markov chain which simulated the real-world scenario of both random and bursty errors. Multi-step 24-hour predictions are made on incomplete training data as well as on complete training data for comparison.

It was seen that the multi-task learning method for Gaussian Process Regression provided significant improvements over the baselines considered as well as over the comparable method of Support Vector Regression. Further, it was seen that our methods did not increase the computational expense significantly, and that the improvements in accuracy seen were consistent across datasets, error-generation scenarios, parameter values and across the individual sensors in the datasets as well.

The use of the kernel compositions to incorporate prior knowledge about the working week were not seen to perform well in concert with the multi-task learning approaches. This is possibly due to the increased complexity of the model with both the elaborate kernel compositions for the prior knowledge as well as multi-task learning transfer making generalisation without overfitting more difficult, in addition to the increased computational expense for the now more demanding optimisation process not converging to effective minima. These issues could potentially be resolved by using approximate methods for Gaussian Processes to enable more training data.
to be utilised with the models, though these approximate methods themselves would have trade-offs with accuracy versus computational expense.

A limitation of our model is that it does not take spatial data into consideration. Adding information on spatial relationships could result in better decisions made about which sensors to combine resulting in potentially improved accuracy figures. Further, using spatial information could be utilised to perform predictions at locations where sensors are not installed, and also be used to inform decisions on locations at which to place new sensors. One approach might be to place sensors where the variance of prediction accuracy is greatest.

Our work focuses on missing data from sensors, and therefore the identity of the sensor with low quality data would be straightforward to identify. This contrasts with scenarios where low quality data could manifest in the form of spurious data, where it would not be immediately obvious that the data of that particular sensor should not be trusted. Therefore, in order to use our methods in situations with such possible invalid data (as opposed to missing data), a step would needed to be added to identify sensors with incorrect data. Anomaly detection algorithms could be considered to perform this sensor identification process.

On a similar note, it is worth mentioning that the accuracy of the prediction would be dependent primarily on the similarity of the time series of the sensors. If the other
sensors (with similar behaviour and complete data) had spurious data this would affect the prediction of the sensor with missing data. The probability of multiple sensors having low quality data, though possible, is not likely enough for multi-task learning to not be considered.

More future work could be done in relation to a hierarchical multi-task learning model, where we build multiple multi-task learning models. We could have multiple multi-task models built at different levels of correlation between sensors (e.g. 0.9, 0.8,...) and weight the predictions of each of the multi-task models corresponding to the level of correlation (i.e. models with high correlation would be weighted higher in the final combined model). These weights would preferably be defined as parameters of the model that could be learnt from the data, effectively providing us with an ensemble of multi-task learning models.
With the advent of cheaper sensors and the Internet of Things paradigm, smart cities nowadays involve increasingly larger numbers of sensors. This results in scenarios where data is being generated very fast, and which consequently require algorithms that are able to keep up with the high rate of data ingestion. The requirement for high performance algorithms applies even to situations where the actual forecasting
task does not need to be performed in real-time, since preprocessing steps will need to be performed online on the fast moving data. Addressing the scalability challenge of such fast moving data provides a more complete solution to the general problem of forecasting in smart city scenarios, complementing the challenges of improving accuracy and robustness of prediction accuracy with low quality data that we have addressed in previous Chapters 3 and 4 respectively. In the next two chapters we focus on scalability of prediction to address the challenge of increasing data volume. In this chapter we address the problem of a base algorithm on which to build the scalable algorithm, and our contributions lie in improving the prediction algorithm of that base algorithm without sacrificing prediction throughput. In our use case of performing predictions in smart city scenarios, we desire certain characteristics in such a base algorithm for developing a scalable forecasting solution.

Given the large volume of data that is generated through the numerous smart sensors in a city, a method that performs fast prediction and does not require a long time for training is paramount. Further, in regard to training, a method which does not require the tuning of too many parameters would be particularly suited for this scenario. For example, autoregressive (e.g. ARIMA) methods require the setting of multiple parameters to suit the particular dataset at hand, and frequently require
visual inspection of auto-correlation plots and the examining of the stationarity of the data. Such parameter setting would be incredibly cumbersome given the wide variety of sensors for which we need to make predictions.

The use of Deep Neural Networks has attracted considerable attention of late, and at first glance would be an alternative to consider. However, it is to be noted that Deep Neural Networks commonly require 1) A large amount of data to be effective and 2) Long training times that can be in the order of days or weeks. Further, models based on Deep Neural Networks have limited interpretability, which is a characteristic that is important in smart city scenarios where understanding the logic underpinning a prediction may be as important as the accuracy of the prediction itself.

Given the overall focus on speed of prediction, the Pattern Sequence-based Forecasting (PSF) algorithm[71] was seen to be uniquely suited for this purpose. PSF is a general-purpose time series forecasting algorithm, and its effectiveness in predicting electricity time and price series was demonstrated in [71], outperforming many autoregressive and neural network approaches.

Though the results of the original PSF algorithm work were demonstrated on electricity time series data, its benefits are not due to being tailored to this use case and are more generally applicable. Indeed, the authors of the original algorithm state that ‘The PSF algorithm aims to be a general-purpose forecasting procedure’[71], an eval-
uation that is borne out by its performance in the other use cases on which we tested
the algorithm. The unique advantages of the PSF algorithm are elaborated below
where it can be seen that the benefits are generally applicable. In addition to the the-
oretical properties of PSF yielding a number of benefits, we also demonstrate in our
results that PSF is orders of magnitudes faster than other commonly used forecasting
approaches, mainly due to the unique discretisation step of the algorithm.

This discretisation step relates to the major challenge of dealing with high-dimensionality
of time series data. For example, data which is recorded each hour would result in
a 24-dimensional vector being created for each day. PSF addresses this problem of
high-dimensionality by performing the aforementioned discretising step on the time
series. Instead of recording each and every value, PSF will instead replace those val-
ues with a single label - hence the name ‘label-based’. This allows PSF to operate on
data of much lower dimensionality, which is one contributing factor to its speed.

In addition to these benefits of speed, we observed that the algorithm was espe-
cially effective in multi-step prediction as is required in this scenario with minimal pa-
rameter tuning. Another advantage of PSF which is relevant to the smart city setting
is that the computation complexity can be easily controlled by varying parameters.
This tunable complexity is also attractive in a setting where scalability is important
as the algorithm can be tuned to the computational resources at hand.
PSF is also an algorithm that is interpretable and is not too much of a black box. Therefore, the result of any changes that are made to the algorithm are more predictable. This predictability to perturbation is advantageous in a scenario where parallelising of an algorithm is involved, since it can be expected that numerous changes will need to be made in order to adapt an algorithm to run on multiple machines.

Vanilla PSF was also shown to perform competitively in terms of accuracy against other techniques in the original PSF paper[71]. However, we found that the accuracy levels observed when we ran it on our real-world data in certain circumstances were not entirely satisfactory. For example, we often observed high variability in prediction accuracy between weekends and weekdays. We were able to isolate these causes of prediction inaccuracies due to the inherent interpretability of the model. This identification of limitations of the algorithm motivated us to perform the enhancements to the algorithm to improve on its accuracy without compromising on predictive performance as detailed in this chapter. These enhancements, the rationale for each of these enhancements, and the resulting accuracy improvements have been detailed in this Chapter.

We make the following contributions in this chapter:

- We improve upon the label-based forecasting technique proposed in [71] (PSF)
by the addition of multiple enhancements to the core mechanism of the algorithm.

- We propose a method to incorporate external data (temperature in this case) to label-based forecasting, and demonstrate improvements of between 27% to 41% over the original label-based forecasting technique.

- We build a predictor ensemble which exploits temporal locality of the time-series data to significant effect. Our ensembling technique reduces the error rate of the predictions by between 37% to 47% over the original technique.

- We compare our enhanced forecasting technique (PSF+, which includes all improvements above) to other competitive linear and non-linear forecasting methods to demonstrate the effectiveness and robustness of our method across multiple real-world datasets.

We now provide a description of the PSF algorithm.

5.1 The PSF Algorithm

The PSF algorithm performs prediction in three steps:
5.1.1 PSF Step 1 - Label Generation

Label generation is the first step of the PSF algorithm. The input to the algorithm would be a set of periodic (say hourly) sensor data $H_d$ for each day $d$. At the end of the first step, each vector $H_d$ will be represented by a single label.

These representative labels for each day are generated by clustering the days in the training time period to similar groups of days using k-means clustering, with the Euclidean distance between the day vectors $H_d$ used as a similarity measure. For example, if we are looking at a period of one year, and choose five clusters, then we would assign each of the 365 days into one of the five clusters based on the (normalised by the maximum usage of the day) hourly usage of each day. Therefore each of the 365 days will fall into one of the five clusters, and the cluster ID will serve as the label for that day.

After the first step of the PSF algorithm is complete, we will have a label for each day which is simply the cluster ID of that day. The algorithm no longer looks at the individual time series values for each day and exclusively uses these cluster IDs or labels until the very last stage in the prediction process.
5.1.2  PSF Step 2 - Equivalent Date Matching

This step identifies days that can be considered ‘equivalent’ to the day to be predicted. This equivalence is determined based on the labels of the days immediately prior to the day to be predicted.

An initial matching window size is set, and the labels of the days immediately preceding the day to be predicted which are within this window are noted. For example, if the labels of the days immediately preceding 2016-04-15 are 4,2,5,1,5,5,4 (oldest first), and the matching size is 3, then the labels considered in the window would be [5,5,4]. The algorithm then searches backwards through time for a match of the label sequence [5,5,4]. If the algorithm finds such matches, then the days immediately following each of the matches are added to the set of all days equivalent to the day to be predicted (ED). That is, the days immediately following the matched window are treated as ‘equivalent days’ to the day to be predicted.

If at least one match is found for the initial match size, the algorithm moves straight on to generating the prediction. However, if no matches were found for the initial match size, the match size is decreased by one, and the search repeats. In our example, the algorithm would now seek matches for the label sequence [5,4]. This process continues until at least one day is added to ED.
5.1.3 PSF Step 3 - Generating Prediction

In the prediction generation stage, we have at least one (and preferably multiple) equivalent days in $ED$. The final prediction is calculated as the average across all days in this set for each hour. For example if we had two equivalent matched days with load values $<23.2, 45.2, \ldots, 19.4>$ and $<11.4, 42.4, \ldots, 18.7>$, the final prediction would be $<(23.2 + 11.4)/2, (45.2 + 42.4)/2, \ldots, (19.4 + 18.7)/2>$.

In general form, the final prediction for day $d$, $\hat{X}(d)$ is calculated as:

$$\hat{X}(d) = \frac{1}{|ED|} \sum_{j \in ED} X(j)$$

(5.1)

where $X(j) \in \mathbb{R}^{24}$ is the 24 hourly load values of day $j$.

The PSF process is illustrated in Figure 5.1.

5.2 Proposed Technique

Our technique (PSF+) is based on the following enhancements to the original PSF algorithm. These enhancements were made to improve the accuracy of the PSF, and are not meant to improve the inherent scalability characteristics of the algorithm.
5.2.1 Filtering and Windowing

It is seen that in the original PSF algorithm, ‘equivalent days’ to the date to be predicted are selected *solely* based on the labels of the days that precede that day. For example, if we wish to predict hourly load values for the date 2016-04-15, and the labels of the days preceding 2016-04-15 are 4,3,1 (match size 3), then *any* day in history which has days with the labels 4,3,1 preceding it will be selected as an equivalent day to 2016-04-15. This essentially blind acceptance of historical days as equivalent to the day to be predicted presents a couple of problems:

- The equivalent day selection method treats all days equally. It was observed that weekends were selected as equivalent to weekdays and vice versa. Even
though the time series profiles for weekends would be significantly different to that of weekdays, it is entirely possible that the profiles of the days leading up to a particular weekend were similar to the profiles of the days leading up to a particular week day, which results in this non-optimal selection of equivalent days.

- The equivalent day selection method does not take into account the recency of the data. A matched day in 2014 for date 2016-04-15 might well not be a worthy representative date.

We tackle these problems by: 1. Filtering - Filtering out weekend matches when a weekday was to be predicted and filtering out weekday matches when the day to be predicted was a weekend, and 2. Windowing - Defining a window \( W_d \) past which equivalent days are not considered (i.e. dates that are older than \( W_d \) days in the past are not considered in the matching process). Windowing prevents old outdated data from affecting the prediction. It is similar in its goal to a time based (exponential) discounting scheme, though it was found that this simpler windowing scheme produced higher improvements in prediction accuracy. Forecasting accuracy improves as it is less likely that irrelevant data can affect the prediction when the windowing step is used.

These quite straightforward techniques address a fundamental limitation of the
original PSF algorithm, and were seen to noticeably improve prediction accuracy.

5.2.2 Match Size Averaging

The original PSF algorithm selects a particular match size, and then searches for matches based on that match size. For example, if the starting match size chosen is five, and there is at least one matching day in history for this match size, then the time series values of that day are used as the predicted time series values for the date to be predicted. The important point to note here is that other match sizes that are lesser than 5 are completely disregarded in this scheme. Match sizes smaller than 5 are only considered if there happens to be zero matches with a match size of 5.

This enhancement involves finding equivalent days using multiple match sizes. We start with a large initial match size, and search for equivalent days using this initial match size. We then also search for equivalent days using all match sizes lower than the initial match size. It is important to note that we search for equivalent days for lower match sizes regardless of whether a larger match size resulted in a match or not. At the end of this process we would have a set of equivalent days

\[ ED^* = ED_{wL} \cup ED_{wL-1} \cup ... \cup ED_1, \]

where \( w_L \) is the initial large match size. The final prediction is the average of all the days in the set \( ED^* \).
5.2.3 Temperature Equivalence

Given the unpredictability of sensor data, there is an ultimate limit of accuracy for even a perfect predictor which utilises only the sensor readings as training data. Therefore, to improve accuracy above what is possibly using solely sensor recordings, we further utilise external information in the form of the temperature of the day to be predicted. Temperature is a factor that can highly influence response variable values of a real-world time series (e.g. heating / cooling requirements change with temperature which would influence electricity consumption), and therefore can be hypothesized to have reasonable predictive power. We incorporate temperature information to our model as follows.

*Temperature of day taken as average of minimum and maximum temperature of day due to lack of granularity in historical temperature data
In addition to equivalent days as selected by the PSF algorithm, we also select ‘temperature equivalent days’. A temperature equivalent day is a day which is of the same day of week (ex - some Tuesday in the past if the date to be predicted is a Tuesday), and which has a temperature close to the forecast temperature of the day to be predicted. Temperature equivalent days are searched for from the most recent week backward, until a prespecified number of equivalent temperature days are found or a configured number of weeks have been searched. This search procedure for temperature equivalent days is shown in Figure 5.2. The final prediction is then computed as the average of the load values of the equivalent days and of the load values of the temperature equivalent days.

5.2.4 Ensemble (Temporal Locality)

The original PSF algorithm does not take into account the temporal locality of the time-series data into consideration. Incorporating temporal locality in the prediction process would be beneficial in improving prediction accuracy as real-world time-series data frequently exhibit similarity in values close to each other in time. It should be noted that this is distinct from preventing old values from tainting the results as was done in the windowing enhancement in 5.2.1. We seek to proactively incorporate the values of recent days with high importance in this step.
We took advantage of temporal locality as follows. We created a predictor ensemble of two predictors, one being the PSF algorithm with all enhancements 5.2.1, 5.2.2, 5.2.3 above, and the other being a predictor which output as its prediction the load values of the same day of the previous week. For each day, the final prediction was the prediction of one of the two predictors. The ‘better’ predictor for each day was chosen based on the error rate for each predictor for the past five days.

For example, if the enhanced PSF predictor had a lower error rate over the past five days than the previous week predictor, the values as output by the enhanced PSF predictor would be chosen as the final prediction values.

The steps of the PSF+ algorithm as compared to the vanilla PSF algorithm are shown in Figure 5.3.

5.3 Experiments and Results

Consider the smart-city environment which is represented as a set of sensors $S = \{S_1, S_2, ..., S_n\}$, where for each sensor $S_i$ we have been provided data for each day in $[t_m, t_n]$. The temperature values $[T_{m}, T_{m+1}, ..., T_n]$ for each day in the time period considered are also provided.

For each sensor $S_i$, we make predictions for the sensor data values for each day $d$ in the test data starting at day $t_j$ and ending at day $t_k$. 

121
Figure 5.3: PSF+ vs PSF
\{d \in [t_j, t_k] \mid t_j > t_m, t_k \leq t_n\}

and record the average error \(e_i\) for the predictions across all the days in \([t_j, t_k]\). This error \(e_i\) is the representative error for a sensor. We report the average of all these sensor-level averages as the average error \(E\) for the dataset. The average error \(E\) for each of the datasets used for evaluation in this paper represents the accuracy of the prediction method under consideration.

Since we are predicting over multiple days for evaluation purposes, after observing the actual time series values of each day \(d\), the day is labelled according to the closest cluster centroid to its time-series vector \(H_d\). This enables the day \(d\) to be used in the matching process when predicting for days \(d + 1\) and beyond. The initial clusters are formed using data reserved as training data.

5.3.1 Data

We use the data obtained through smart meters installed in the campuses of the University of Melbourne\(^\dagger\) and pedestrian count sensors\(^\ddagger\) as mentioned in previous chapters.

\(^\dagger\)http://sustainablecampus.unimelb.edu.au
\(^\ddagger\)http://pedestrian.melbourne.vic.gov.au
5.3.2 Predictions

We reserved the first two months of data in each dataset as training data to use as an initial basis for creating the initial clusters to use for matching and prediction in the rest of the data. We predict sensor data values (i.e. electricity consumption values and pedestrian count values) 24 hours in to the future based on the data of all days in to the past. Errors for the prediction for each day are calculated, and the average of the errors across all days predicted are reported as the error of prediction for that particular sensor ($e_i$).

5.3.3 Error Metric

The Error Metric is the mean absolute error normalised by the average magnitude of the actual load values for the prediction period, and expressed as a percentage as used in previous chapters.

Error rates are provided for each separate improvement proposed, and their relative accuracy improvements are discussed relative to the original PSF algorithm. The error rates are also compared against a baseline of using the load values of the same day last week as the prediction for the day to be predicted as a sanity check.
5.3.4 Results

PSF+ Accuracy

We provide a comparative evaluation of our PSF+ method, which consists of the multiple improvements made to the original PSF algorithm. Comparisons are made to both competitive linear (ARIMA) and non-linear (Neural Network) methods in Table 5.1.

In the implementation of ARIMA, an independent value prediction [27] method was used, where separate models are trained to predict each hour of the next day in order to prevent the cascading of errors in multi-step prediction. The Box-Jenkins methodology was used in the tuning of parameters, which resulted in an ARIMA(1,1,0)
(number of time lags, degree of differencing and order of moving average model respectively) model being selected. A Non-linear Autoregressive Neural Network with tapped delay lines was used in order to make the network time-shift invariant. The neural network consisted of one hidden layer with 10 neurons (sigmoid activation function) and one output layer (linear activation function).

It can be seen in Table 5.1 that PSF+ outperforms both the linear and non-linear methods in both datasets, demonstrating the competitiveness and robustness of our methods.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PSF+ (%)</th>
<th>ARIMA (%)</th>
<th>Neural Network (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Melbourne University</td>
<td>7.29</td>
<td>15.25</td>
<td>12.43</td>
</tr>
<tr>
<td>Pedestrian</td>
<td>16.30</td>
<td>38.54</td>
<td>26.35</td>
</tr>
</tbody>
</table>

Figure 5.5: Pedestrian Comparison of Errors

Table 5.1: Error Rate Comparison with Other Methods
Table 5.2: Percentage improvements over PSF

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Filtering (%)</th>
<th>Match Size Averaging (%)</th>
<th>Windowing (%)</th>
<th>Temperature Equivalence (%)</th>
<th>Ensemble (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Melbourne University</td>
<td>14.13</td>
<td>7.44</td>
<td>34.34</td>
<td>40.93</td>
<td>46.64</td>
</tr>
<tr>
<td>Pedestrian</td>
<td>19.54</td>
<td>32.23</td>
<td>32.22</td>
<td>27.57</td>
<td>37.38</td>
</tr>
</tbody>
</table>

We provide an analysis of the performance of each of the constituent improvements of PSF+ in relation to the original PSF algorithm in the following sections.

Filtering, Match Size Averaging, and Windowing

We see in the ‘Filtering’ column of Table 5.2 that the straightforward enhancement of discarding weekend matches for days to predict which are weekdays and vice versa results in healthy improvements in prediction accuracy for both datasets. This result is encouraging as not only is this step easy to implement, but the reason for the improvements is logical and easily interpretable.

The ‘Match Size Averaging’ column of Table 5.2 presents the performance improvement obtained through averaging across multiple match sizes. By not simply stopping once at least a single match has been found for some match size, and finding matches for every match size smaller than an initial large match size we are able to incorporate more information from more equivalent days to our prediction. It also prevents a single ‘match’ which in reality may not be that representative of the date
to be predicted from affecting the result too much. We see that this averaging pro-
cess does result in a further improvement from the filtering step for the Pedestrian
dataset.

The windowing step also results in significant improvements for both datasets as
seen in the ‘Windowing’ column of Table 5.2.

The error rates when utilising all of the three enhancements of filtering, match size
averaging and windowing combined are shown by red cross marks in Figures 5.4, 5.5,
and for each individual sensor. It is seen that the algorithm with these three enhance-
ments beats the original PSF algorithm comfortably for almost all sensors in the two
datasets. This shows that the enhancements also yield consistent prediction accuracy
improvements across all sensors, in addition to the respectable improvement in ag-
gregate that was described above.

**Temperature Equivalence (Incorporating External Information)**

The ‘Temperature Equivalence’ column in Table 5.2 presents the accuracy improve-
ments obtained through using the external temperature information. In this method
the values of temperature equivalent days are also considered in addition to the nor-
mal equivalent days found through the original PSF algorithm, and the final predic-
tion is the average across both temperature equivalent and regular equivalent days.
The use of exogenous features helps with prediction when the response values of the time series is highly dependent on the added feature. For example, it is expected that electricity consumption would be affected by temperature due to increased / decreased heating / cooling needs in buildings, though temperature would have less of an impact on pedestrian counts. This expectation of the influence of the exogenous feature (temperature in this case) on prediction accuracy is observed in the results, with the temperature feature improving the prediction accuracy in the smart city data set significantly compared to the pedestrian data.

On the sensor-level scale, as shown by the yellow squares in Figures 5.4, 5.5, it is seen that the accuracy improvement for most sensors is modest. However, it is seen that the addition of the temperature data seldom results in lesser accuracy, and therefore we see an improvement of accuracy in the aggregate results. Other similar exogenous features that may be considered include humidity values of the day and occupancy levels of buildings which could be inferred based on network traffic data.

*Computational Efficiency*

PSF is unique in that its pattern matching technique allows exceedingly fast performance compared to other techniques. For example as shown in Table 5.3 the original PSF algorithm easily outperforms the competing ARIMA and neural network
Table 5.3: Computational Efficiency of Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Computation Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>185</td>
</tr>
<tr>
<td>Original PSF</td>
<td>191</td>
</tr>
<tr>
<td>Filtering + Match Size Averaging + Windowing</td>
<td>216</td>
</tr>
<tr>
<td>Temperature Equivalence</td>
<td>260</td>
</tr>
<tr>
<td>Ensemble</td>
<td>339</td>
</tr>
<tr>
<td>ARIMA</td>
<td>221 220</td>
</tr>
<tr>
<td>Neural Network</td>
<td>149 70</td>
</tr>
</tbody>
</table>

techniques. Fast prediction is one of the key advantages of forecasting using pattern matching, and this speed is largely unaffected even with the enhancements made on PSF. Though the enhancements do slightly increase computation time, it is seen that the increases are not that significant except in the case of ensembling. The increase in computation time when ensembling is to be expected due to two different types of predictors being run. Nevertheless, it is seen that the PSF+ algorithm which comprises all the proposed enhancements (including ensembling) performs far better in terms of computational efficiency in comparison to the ARIMA and neural network methods.

**Ensembling**

The ensembling method, which allowed us to use the principle of locality on the time series data was what provided us with the most impressive predictive accuracy
improvements. The ensembling method makes an intelligent choice of prediction method based on the recent error rates of each method.

Our experiments showed that neither of the prediction methods in the ensembles dominates in the selection process, with each method being picked more or less half of the time (Method 1: 46%, Method 2: 54%). This indicates that the improvements in prediction accuracy are due to ensembling resulting in the appropriate predictor being selected to predict each day, and not due to any inherent superiority of any of the methods. It can be seen in Table 5.2 that the ensembling method results in an improvement of the error rate in prediction by 47% for the University of Melbourne and 37% for the Pedestrian data.

This improvement can be seen on a sensor level in Figures 5.4, 5.5, where the blue diamonds representing the ensembling method are consistently, and for many sensors significantly, placed lower than all other methods.

5.4 Conclusions

This Chapter focused on the popular label-based PSF forecasting algorithm. PSF has merits in terms of speed and interpretability which are important in a large scale smart city prediction scenario. The accuracy of PSF is higher in cases of aggregate prediction (e.g. prediction for a whole city) compared to prediction for individual
sensors. The enhancements to PSF we proposed to improve accuracy as well as the unique advantages of the algorithm in terms of scalability make it a highly appropriate choice to develop a distributed forecasting system as proposed in Chapter 6.

We enhance the PSF algorithm through filtering and windowing techniques, and demonstrate that these enhancements improve the predictive accuracy of all sensors present in two real-world smart city datasets. We further improve on our results from the techniques above by providing methods to incorporate external information (temperature in this case). Further significant improvements were made on the predictive accuracy by using an ensembling method which helped the algorithm utilise temporal locality to improve its predictions. Our techniques resulted in an improvement in prediction accuracy of up to 47%.

A limitation of our enhancements is in the ensembling and temperature equivalence steps, which sacrifice on computational expense. However, as seen in the results this does not result in too much of a decline in performance, and these enhancements can be switched off in a scenario with a higher emphasis on predictive performance compared to accuracy. A further improvement for future work could involve a mechanism to perform such adjustments dynamically based on the current rate of ingestion of data, and possibly also based on the resource usage at that point in time.

In the next chapter we discuss scalability in prediction algorithms, and solve the
associated challenges using PSF+ as the base algorithm.
With the explosion of the Internet and the number of devices connected to it, the amount of data being generated nowadays has reached unprecedented levels. The increased connectivity of devices, along with the availability of cheap sensors, has resulted in an expansion of the volumes of data generated in smart city scenarios as well. Therefore, in this chapter we build on our work in Chapter 5 on improving the
accuracy of a high performance algorithm (PSF), and use it as the basis to develop scalable forecasting algorithms.

Forecasting algorithms are frequently applied to data at rest - or in scenarios where the inflow of data is slow enough to allow the use of traditional batch processing algorithms. However, it is often the case that batch processing is neither feasible nor desirable. This is especially the case in smart city settings, where smart sensors are recording information from a dynamic environment. Therefore, machine learning methods used in smart city settings need to be adaptable to dynamic or streaming type data, along with being able to deal with large volumes of data. In this chapter we look at the scalability aspect of forecasting with a focus on streaming data analysis in two types of scenarios.

6.0.1 Scenario I - Multiple Stream Scalability

The first scalability challenge arises due to multiple sensor streams, where each stream may be of low or medium velocity. This scenario is very likely to happen in a smart city setting where we may have a large number of sensors installed.

Dealing with the problem of many data streams involves dividing up these many streams across machines. In this chapter we develop methods where we can perform

---


---

135
prediction on many different individual streams using multiple machines. The basis of these methods will be an adaptation of the PSF algorithm discussed in Chapter 5.

We use the micro-clustering technique [5] in order to adapt the PSF algorithm to run better on data streams, where micro-clustering performs the initial labelling step of PSF. Using micro-clustering for the labelling step also provides the added benefit of serving as a summary of the data stream. This micro-clustering summary has the advantage of being able to be stored in constant space and processed in constant time.

6.0.2 Scenario II - Single Stream Scalability

The second scalability scenario is when we have a single stream of data that is too fast to be processed using a single machine. This is often seen in real-time social media analytics such as analysing Twitter streams which can provide insights into the state of a city. Therefore, we focus on methods that will allow us to process a single such high velocity stream.

The challenge in this scenario relates to dividing up this single fast stream into multiple separate streams, and then processing each separate stream individually in different machines. Since we are dividing up a single stream, and the machines are processing the resultant multiple streams in parallel and independently from each other, the resulting clustering will not be the same as if we had processed the single
fast stream in a single machine.

Therefore, much of the focus of the work in distributing a single fast stream across multiple machines will be on ensuring that the processing nodes stay as synchronised with each other as possible. This synchronisation is needed to ensure that the final clustering result in the distributed version will not deviate too much from the final result achieved if the clustering was done on a single machine. We develop two different synchronisation methods, one with a global in-memory store and one where a peer-to-peer like architecture is proposed for the synchronisation updates.

The methods developed in this chapter enables micro-clustering at scale on a high velocity data stream. These micro-clusterings generated can be used in the initial stage of the scalable PSF algorithm described in Scenario I above.

6.0.3 Micro-clustering motivation

We employ micro-clustering in our solutions in both scenarios. When we need to process fast individual streams, micro-clustering is a perfect fit as it was originally built for summarisation purposes. Not only does micro-clustering perform summarisation in constant time, but it also provides methods to analyse how these micro-clusters changed over time based on user-defined time-ranges, thereby allowing the user to observe how the distribution of data changed over time in the data stream. This
ability to observe concept drift in streaming data is considered especially valuable.

Micro-clustering can be used for performing predictions on multiple streams as well, since it can be used as the initial clustering stage of the PSF algorithm. This allows the PSF algorithm to perform better under streaming scenarios, as well as provide the summarisation and data distribution change monitoring functionalities of micro-clusters. Therefore, we focus on scalable methods that use micro-clusters in this chapter.

6.0.4 Contributions

In this chapter we make contributions in solving the scalability problem in both scenarios with multiple streams and a single fast stream. We modify the PSF algorithm to be able to run in a distributed system and handle a large number of multiple real-time streams, and demonstrate minimal forecast accuracy loss with increased throughput. The synchronisation problem that needs to be solved when dealing with one fast stream divided to multiple machines is solved for the micro-cluster maintenance algorithm. We solve this synchronisation problem of micro-clusters using two approaches: 1. An in-memory global store and 2. A peer-to-peer solution.

Our results demonstrate that our algorithms based on these synchronisation methods can be run on high-throughput streams with large numbers of micro-clusters
compared to the centralised streaming case (linear scalability in many cases). We also illustrate the relative advantages and drawbacks of each of the two distributed algorithms proposed for micro-cluster maintenance and provide guidelines to practitioners on the choice of algorithm and algorithm parameters. We implement our scalable methods on the popular Apache Storm architecture, which is a purpose-built framework for low-latency stream processing.

In the next sections we provide an introduction to the micro-clustering algorithm CluStream [5], and to the Apache Storm architecture.

6.1 Background - The CluStream Algorithm

The CluStream algorithm was introduced in [5], in order to address the fact that existing stream clustering algorithms were blind to the evolution of data. Compared to other algorithms that assumed that the clusters would be computed over the entire data stream, the CluStream algorithm provided the flexibility to investigate clusters over user-defined time periods.

The CluStream algorithm divides the clustering process into two components in order to provide this flexibility - an online component which periodically stores summary statistics, and an offline component which uses these summary statistics along with user input to provide the ability to explore the nature of evolution of the clus-
The online component of the algorithm determines the stream rates that the algorithm will be able to handle, and therefore will be the component on which improvement efforts will be focused on in this chapter. The online component is responsible for maintaining summary statistics of the data instances seen in the stream in a data structure named the ‘micro-cluster’.

The micro-cluster is a method of summarising the data points seen in a data stream. A micro-cluster that serves as a summary of an \( n \) number of instances (each instance with a number of features of dimensions) seen in a stream consists of the following components:

1. The dimension-wise sum of the data points in the stream
2. The dimension-wise squared sum of the data points in the stream
3. The number of data points summarised by the data structure.
4. Sum of squares of timestamps \( T_i \) to \( T_n \)
5. Sum of timestamps \( T_i \) to \( T_n \)

The maintenance of the sum and squared sum of the data points enables the computation of the mean and standard deviation of the data points summarised.
edge of the mean and standard deviation of the data points provides a reasonable understanding of the distribution of the data. The final clustering will be produced by running a traditional clustering algorithm such as $k$-means on demand offline, with the algorithm considering the micro-clusters as pseudo-points.

The micro-clustering can also be intuitively thought of as a very fine granularity clustering as compared to the final clustering that we obtain. A clustering process can be thought of as a process of summarisation of the objects considered into similar groups. Micro-clustering also forms groups of objects in order to provide a summarisation of the data points streamed in. In essence, micro-clusters are stored in memory instead of storing each and every data point in memory as the micro-cluster summarisation can be done in constant space. The salient differences between regular clustering and micro-clustering are that 1. Micro-clusters can be formed on the fly and can be updated for every data point with constant cost and 2. Micro-clusters are more abundant in number as they represent a summary of the data points and therefore lead to a loss in information - storing more micro-clusters results in a lesser loss of information.

For example, consider a scenario where 1000 micro-clusters are maintained. These 1000 micro-clusters would form a constantly updated summary of the streaming data points received by the algorithm. When the final clustering is to be formed, the cen-
troids of these 1000 micro-clusters are treated as the points to be clustered. That is, if a final traditional clustering algorithm such as k-means is employed to find 5 clusters, the micro-cluster centroids would serve as the points to be clustered in place of the much higher number of raw data points that would have been ingested. Therefore, 1000 pseudo-points (that provide a summary of a far larger number of points) would be clustered into five final clusters.

The algorithm used to update the micro-clusters in order to form an accurate representation of the data points seen is outlined in Algorithm 1.

Algorithm 1 Online micro-clustering algorithm

On receipt of new data instance from stream

Find closest micro-cluster \( M \) to the data instance

Check if point falls within maximum boundary of \( M \)

\( if \) (Data instance falls within maximum boundary of \( M \))

Add data instance to micro-cluster \( M \)

\( else \)

Create a new micro-cluster containing only the new data instance

The online micro-clustering algorithm would keep track of a constant (and user-defined) \( q \) number of micro-clusters. On the receipt of each data stream instance, the algorithm calculates the distance between each micro-cluster centroid and the
stream data instance, and decides which out of the \( q \) micro-clusters lies closest (and therefore is most similar) to the data stream instance. This closest micro-cluster then becomes the candidate micro-cluster which may be used to summarise the incoming data instance.

However, though a stream data instance might be closest to a given micro-cluster, it might not be sufficiently close enough to merit being summarised by that micro-cluster. Therefore, the algorithm performs a check to determine whether the distance between the arriving data stream instance and the candidate micro-cluster is lesser than the ‘maximum boundary’ of the candidate micro-cluster.

This maximum boundary is determined by the current distribution of points within the micro-cluster and a user-defined radius factor. Specifically, the ‘radius’ of a micro-cluster is calculated by taking the mean of the standard deviations across all dimensions. That is, each micro-cluster is composed of a number of data points, each of \( d \) dimensions. The standard deviation is calculated for each of these \( d \) dimensions (for the data points in that micro-cluster), and the average of these standard deviations is the radius of that micro-cluster. The maximum boundary will be this radius multiplied by a user-configurable radius factor. If the data instance is sufficiently close (i.e. falls within the maximum boundary of the closest micro-cluster), the algorithm summarises the data point by adding it on to the micro-cluster. Else, a new micro-cluster
with only the single incoming instance is created.

There does not exist a theoretical upper limit to the number of micro-clusters that can be maintained. Micro-clusters preferably are maintained in memory, therefore the upper bound would depend on the amount of physical memory on the node. The maximum number of micro-clusters would also depend on the amount of latency that can be tolerated since for each data point that streams in an exhaustive search should be made across every micro-cluster that is maintained. The higher the number of micro-clusters that is maintained, the higher the time taken for this search.

6.1.1 Background - The Apache Storm platform

Apache Storm is an open-source distributed computation system for real-time streaming data. It has attracted considerable attention from companies such as Twitter and Groupon, and has been used in recent academic work [13] [4] [101] [72]. It is horizontally scalable, while being fault tolerant and providing absolute guarantees of successful message processing. Storm is also programming language agnostic, with application logic being able to be written in any language.

The main abstraction of an application in Storm is the topology, which in most cases is a Directed Acyclic Graph of computation units, or in Storm parlance ‘components’. A topology is formed by the interconnection of several components. These
components may either be spouts or bolts. Spouts provide the source of the stream of data to process, whereas bolts do the actual processing on the data originated from the spout. Components are interconnected through streams, with bolts both subscribing to and emitting streams and spouts only emitting streams. Storm uses tuples as the abstraction for data flow, which is general enough to accommodate any type of data.

Multiple copies of a component which execute the same code can be run on multiple machines with minimal configuration using the Storm framework. Each such instance of a bolt is called a task. The ability of defining multiple tasks of bolts and spouts to run on multiple processes and machines is what allows parallelism of computations in Storm. Storm manages the provisioning of resources and maintaining of connections between these multiple tasks.

The connectivity between the tasks (instances of components) are handled by Storm groupings. A grouping defines the manner in which each data instance is forwarded from the task of one component to a task of the component which is further downstream in the topology. Storm provides six types of groupings - namely Shuffle, Fields, All, Custom, Direct and Global. We have employed the Shuffle grouping (which forwards tuples in round-robin sequence) and the Fields grouping (which forwards tuples based on one or more fields in the tuple (data instance)) in our work.
We now provide descriptions of our work to address the challenges of the two scalability scenarios introduced in this chapter.

6.2 Scenario I - Multiple Stream Scalability

The forecasting algorithm that forms the basis of the scalable prediction solution is PSF+ which was described in Chapter 5. The initial labelling step will be done using the CluStream algorithm, which uses micro-clustering to form summaries of data streams. These micro-cluster numbers will serve as the label values in the PSF+ algorithm in the scalable solution.

In the next section we describe the architecture as implemented on Apache Storm.

6.2.1 PSF-Scaling Architecture

Our architecture for scalable forecasting consists of three separate layers. This three-layered architecture as implemented using Apache Storm is shown in Figure 6.1

In this problem, we are focusing on the problem of a large number of streams. Therefore, we intend for a given stream to be always processed by the same sensor node so that the node will have a global view of that given stream.

In order to ensure each stream is processed by the same sensor node, we first augment the representation of the data elements by adding a label for each stream. The
distribution of the streams to processing nodes will be done according to this label using a fields grouping. As mentioned earlier a fields grouping will ensure that elements are routed according to a given field: if the first element with field value ‘2543’ was routed to bolt task 4, then every subsequent element with the same field value will be routed to the same bolt. Therefore, using a fields grouping on the stream number will ensure that one node will see all the elements in that given stream. This fields grouping applies between all three layers.

The initial spout reads data elements from file and forwards the 24-dimensional vector (in case of hourly data) to the Allocator bolt. The spout is not coupled to the
file on disk, and drop in replacements can be made using popular message queues such as Kafka. Indeed Storm provides purpose-built spout implementations to connect to such message queues. Given that our focus is on exploring methods to achieve the highest possible throughput from the overall topology, the source of the data elements of the spout is less important.

The Allocator bolt is responsible for the discretisation or labelling step done using the PSF+ algorithm. In this scalable implementation for streaming data, we use micro-clustering in order to label the data elements. An initial clustering step is done in order to create the initial cluster centers. Thereafter, the normal micro-clustering procedure is run, where each new 24 dimensional vector (if hourly data) is ‘allocated’ to the nearest micro-cluster (hence the name Allocator Bolt).

The Allocator bolt will forward the label it assigned to the given data element on to the Predictor bolt, along with the 24 dimensional vector. Note that it is required to forward the actual values to the Predictor bolt as well, since though the matching process only uses the discrete labels, the final prediction requires the actual values of the equivalent dates as well. Regardless of the forwarding of the actual values, the simple label-matching process is preserved with only the labels used for matching purposes.

The Predictor bolt will not perform new operations compared to the original PSF
algorithm. The bolt will run the label matching process and output the prediction for the next day to a file for our analysis. In a real-world production scenario the bolt could also output the prediction values to either a message queue (such as Kafka, RabbitMQ) or to an in-memory store such as Redis. This would allow downstream processes to consume the prediction values in order to either perform additional analysis on the data or to visualise the prediction values.

We now present results in order to gauge the performance of the scalable architecture against the single machine scenario. We focus most of our attention on the bottlenecks in the architecture so that practitioners may utilise the resources at hand in the best way possible.

6.2.2 PSF-Scaling Results

*Allocator Bolt Parallelism*

In Figure 6.2 we observe that increased allocator bolt parallelism does not help when the number of spouts is one. In fact, using one spout does not even improve on the throughput we are able to achieve using a single machine. This points to the fact that the spout throughput is not enough to keep up with the processing speed of the allocator bolts, and therefore is the bottleneck in this scenario. For a number of
spouts greater than one, we notice a significant improvement up to 10 allocator bolts and then we see diminishing returns. This is possibly due to a combination of either the number of spouts or the predictor being a bottleneck.

**Predictor Bolt Parallelism**

We see in Figure 6.3 that increasing the number of predictors does not result in a performance improvement. In comparison to allocators, predictors do not perform much work and therefore this result, that predictors are not bottlenecks in the system is understandable. Increasing parallelism helps in situations where the process is CPU-bound (i.e. high Capacity metric of component in Storm). The simplicity of the PSF algorithm means the Predictor bolt is not computationally bound, which resulted in low values of the Capacity metric (< 0.5) seen on Storm.


Spout Parallelism

In Figure 6.4 we see a general improvement in throughput with an increased number of spouts, though the rate of increase decreases past 10 spouts. The slight dip at 30 spouts can be attributed to the fact that the number of tasks exceeded the number of physical machines, which resulted in more than one task being run on more than one machine which resulted in task swapping which resulted in the performance downgrade. It can be seen that an increased number of allocators results in increased throughput for the same spout parallelism, which is in line with the results seen in Allocator Parallelism.
6.2.3 Accuracy Comparison

We present an accuracy comparison of the scalable solution (which uses micro-clusters for labelling) versus vanilla PSF (which uses traditional k-means for labelling). We observe that the use of micro-clusters in the scalable solution does result in slightly lower accuracy in aggregate (17.22% error rate vs 16.30% for vanilla PSF). However, given the throughput advantages this slight lowering can be considered acceptable. Further, it can be seen in Figure 6.5 that the different labelling scheme does not result in large accuracy drops for any individual sensor.

We next present our solution to the single stream scalability problem where we parallelise the micro-clustering algorithm used for labelling to run on multiple machines.
with the single fast stream distributed across the multiple machines. Our focus in this work will be synchronisation methods for the micro-clusterings stored on individual nodes.

6.3 Scenario II - Single Stream Scalability

In this section we investigate distributed versions of the micro-clustering algorithm which are able to run on cloud platforms. Since micro-clustering is not inherently data parallel, it is challenging to run the algorithm in parallel while maintaining clustering accuracy. We present two algorithms which aim to mitigate the non-independent nature of micro-clustering using two different approaches and present the architectures on which the algorithms can be run. Unlike many other stream processing algorithms, our algorithms do not rely on the micro-batching of data. Therefore, our implementation preserves the streaming semantics of the original CluStream algorithm, which was designed to run in a real-time manner for each and every data point. This will allow applications developed using the traditional CluStream algorithm to run essentially unchanged on our system.

We implement the two algorithms using the popular stream processing platform Apache Storm. Our two algorithms solve the challenge of dynamic synchronising of these parallel workers in the Directed Acyclic computation topology of Storm in two
different methods. The principles of these methods can be applied to other similar stream processing problems requiring distributed solutions. We demonstrate that both algorithms are able to handle high throughput streams even with large numbers of micro-clusters as compared to the centralised streaming case with minimal loss in accuracy. We also present the relative benefits and drawbacks of using each of the algorithms, and provide guidelines to the practitioner for the choice of algorithm and algorithm parameters.

Further, the algorithms allow for a configurable level of contention (common-store algorithm) and also an algorithm with no contention (decentralised algorithm with increased message passing). This customisability of the level of contention based on the accuracy required is seldom seen in other proposed algorithms in this space. We have also succeeded in eliminating the effect of contention at the global aggregation stage, which is a very common limitation distributed algorithms face when aggregating the individual results of local nodes. These successes improve upon the state of the art and differentiate and motivate our distributed implementation.

6.3.1 Distributed Algorithms

We present two versions of distributed algorithms, one which runs on a shared-memory architecture using a common store, and the other which runs on a non-shared archi-
tecture in a decentralised fashion. The algorithms have been developed with modern stream processing frameworks in mind which represent computation as a Directed Acyclic Graph (DAG) of different components. It would be instructive for the reader to note that the ‘processing units’ mentioned in the descriptions below would correspond to nodes in this DAG, and that each node would have many parallel instances.

**Shared-memory Algorithm**

In the description of the shared-memory algorithm (Algorithm 2), we identify two different kinds of processing units named allocators and aggregators. The allocator units are those that run the bulk of the clustering algorithm. That is, they perform the assigning of each stream data instance to the micro-cluster closest to it. This means that the allocator processing units would be running the original CluStream algorithm on the data instances that are forwarded on to it. The allocator processing units will be performing the clustering process independently, and only on the subset of data that they receive. However we require a single global clustering based on all the data points as our final result. The aggregator tasks are responsible for forming this global micro-clustering picture based on the local micro-clustering results. The aggregator tasks also update the common store with this global status.

The allocator clustering processing units need to be made aware of this global clus-
Algorithm 2 Shared-memory Algorithm

Data stream instances sent to allocator units in load-balanced manner.

for all allocator units concurrently do
  On receipt of new data instance from stream
  numInstances ← numInstances + 1
  if (numInstances mod syncPeriod == 0)
    Synchronise from common store
    Allocate instance to closest local micro-cluster
    Form and send micro-cluster update message to an aggregator unit

for all aggregator units concurrently do
  if (‘add update’)
    Simple addition of dimension-wise sums and squared sums of instance to micro-cluster
    Write updated micro-cluster to common store
  else
    Merge two closest micro-clusters in common store
    Create new micro-cluster from instance
    Write new micro-cluster to common store

on user request do
  Perform k-means clustering on micro-clusters in common store
tering state periodically in order to prevent accuracy drift with time. This periodic synchronisation is performed as follows. Each allocator unit would maintain a running count of the number of data stream instances that it has received and processed up to the current point in time. This running count is used to determine the time at which to synchronise, where the synchronisation is done whenever the running instance count is a multiple of the user-defined synchronisation period.

It would also be instructive to mention how the passing of information from allocators and aggregators is managed in this algorithm. The step of allocation of data instance to closest micro-cluster happens in the normal manner, based on the CluStream algorithm, but importantly only using local information as mentioned earlier. The changes that happen in the local micro-clustering in each local allocator node are sent to the appropriate aggregator nodes using micro-cluster update messages. A micro-cluster update message would contain the id of the micro-cluster that was modified in the update, and the change in the state of this micro-cluster. This change would be the difference that occurred in the individual components of the micro-cluster (i.e. the five components mentioned in Section 6.1) in this update. Note that each local allocator unit should send their changes on to the appropriate aggregator unit, and that the present local state of the micro-cluster cannot be directly sent onward towards the aggregator. This is because many allocator units may be working
on the same micro-cluster in one update period. Sending just the present local state of the micro-cluster would just result in subsequent overwriting of that micro-cluster state in the aggregator.

The aggregator unit has a straightforward job if the update that occurred was a simple add update. That is, the data stream instance fell within the maximum boundary of the closest candidate micro-cluster that was found. When this is the case, the aggregator units simply use the additive property of micro-clusters in order to summarise the change occurred. However, if a new micro-cluster (containing solely the stream instance just seen) has been created in the previous update, the two closest micro-clusters maintained in the global store are merged in order to make space for the new micro-cluster, as the number of micro-clusters maintained is a constant.

*Non-shared Algorithm*

The non-shared algorithm (Algorithm 3) has only one variety of replicated parallel processing unit, as compared to the two types of ‘allocator’ and ‘aggregator’ units in the shared-memory algorithm described above. Synchronisation of micro-clustering state is performed through exchange of update message vectors. An update message vector is a list of the local micro-clustering updates that occurred during the last update epoch. The non-shared algorithm does share many aspects with the shared-
memory algorithm. However, in this case in order to reduce the overhead of communication between peers, information about the micro-cluster updates that occurred due to several data instances are compiled together in an ‘update message vector’. Each unit’s update message vector is only shared with its peers once the number of instances seen by that peer is a multiple of a user defined update period. Each of the individual micro-cluster update messages that make up this update message vector contain the same information as those that are created in the shared-memory algorithm.

The non-shared algorithm does not hold a continuously updated common view of the micro-clustering as does the shared-memory algorithm. However, a global micro-clustering state is required as input to the final clustering stage which is run on-demand. Therefore, in order to form the final clusters upon user request, a consensus is reached and a common picture formed among all the parallel units as to the global micro-clustering which is to be used as input to the final k-means clustering stage.
Algorithm 3 Non-shared Algorithm

Data stream instances sent to allocator units in round-robin fashion

for all allocator units in parallel do

    on receipt of new data instance from stream
        numInstances ← numInstances + 1

        if (numInstances mod updatePeriod == 0)
            Broadcast update message vector
            Reset update message vector

    Allocate instance to closest local micro-cluster
    Form micro-cluster update message and add to update message vector

on receipt of an update message vector
    Update local copy of micro-clusters

on user request do

    Send synchronisation signal to each task

    Perform $k$-means clustering on micro-clusters in any unit after synchronisation
6.3.2 Architectures

Architecture 1 - High-level description

In this architecture we propose, each individual parallel task will maintain a complete set of micro-clusters. That is, if the number of micro-clusters specified by the user is $n$, each parallel task would maintain $n$ number of micro-clusters. Initially, these $n$ micro-clusters will be initialised to the same values via a separate offline process. Thereafter, as the stream is flowing in, the stream data instances will be distributed in a load-balanced manner among these individual tasks, and the tasks will perform the allocation of the instances to the copy of the micro-clusters that it maintains. The node tasks will perform this allocation step independently, with no reference or communication with the other parallel nodes.

This would mean that each task would be operating based on its own local knowledge of the changes made in the micro-clustering due to the allocation of instances from the data stream to the micro-clusters. Since this local knowledge is imperfect as compared to the global knowledge that the centralised algorithm enjoys, the global state of affairs needs to be communicated to the individual nodes periodically in order to maintain accuracy.

Therefore, though our work aims to perform clustering in a streaming context,
we unfortunately are not able to get around the fact that local nodes are unaware of
global state. Local nodes need to be made aware of the state of the other local nodes,
or else risk continuous degradation in accuracy as the local view of the node deviates
ever more from the global state of affairs. We shall see this effect present itself in the
results section, where an increased synchronisation period results in reduced accuracy.
A method for maintaining the global state across local nodes is very much a necessary
ever evil in the formulation of a distributed version of the CluStream algorithm.

Therefore, in order to maintain the global state of the micro-clusters, in this archi-
tecture we propose the use of a common store such as a database or file system, or
as used in this paper, an in-memory store, which will be accessible by all nodes. The
frequency of access to this common store would depend on the stream speed and the
accuracy required, which is how a practitioner would need to adapt the algorithm to
suit their particular streaming environment.

This common store will not have processing capabilities. Further, as mentioned
above the tasks will not be communicating directly with each other. Therefore, the
need arises for a separate component which will aggregate the changes made in the
micro-clusters in the individual tasks. Though at first glance it appears that this aggre-
gator task would not be able to be parallelised due to its nature of forming one global
micro-clustering using multiple local micro-clusterings, our specific implementation
on Apache Storm is able to achieve parallelisation in this aspect as well.

The final step of the clustering process, which is to use a traditional batch clustering algorithm on the micro-clusters by treating them as pseudo-points, will be run on the global micro-clusters maintained in the common store.

**Architecture 2 - High level description**

In the architecture proposed above, the issue of each task possessing only local knowledge was resolved by maintaining a global picture of the micro-clustering using a store common across all nodes. This common store, however, would be a bottleneck to the overall topology as all parallel bolt tasks would access the same common store to synchronise their local micro-clusters with the global picture. Therefore, we propose an alternate architecture which removes the need for such a global common store.

In common with the previous architecture, we would have multiple tasks which perform the allocation of stream instances to micro-cluster centroids in parallel. Again in common with the previous architecture, these would perform the individual allocation decisions based on local information only. However, compulsory synchronisation of the local state of the nodes in this scenario is done by every node in the architecture acting as a peer, and broadcasting their local state of micro-clustering to
every other node. Though this method will involve higher communication overhead, its benefit is that it removes the need for each node to access a common location to synchronise themselves. This removes the bottleneck of multiple accesses to a single location present in the common store architecture.

6.3.3 Implementation of Architectures

Architecture 1 - Implementation on Storm

Figure 6.6 represents a high-level diagram of the architecture as implemented on the Apache Storm platform.
InstanceSpout
Sends data instances to allocator using ShuffleGrouping (random but load-balanced sending of instances.)

AllocatorBolt
Maintains ‘n’ independent micro-clusters. Performs allocation of data instances to micro-clusters based on local information

AggregatorBolt
Each task will maintain a subset of micro-clusters

Update messages related to a specific micro-cluster will always be forwarded to the task maintaining that micro-cluster

Uses FieldsGrouping on the updated micro-cluster number to achieve this manner of forwarding

Can achieve parallelisation in aggregation step as well since there is no overlap between the subset of micro-clusters managed by each aggregator task

Common Store
A Redis in-memory key-value store is used for performance reasons

Final k-means algorithm is run on micro-clustering maintained in common store.
Architecture 2 - Implementation on Storm

The decentralised architecture described above is not naturally amenable to the Storm platform. The main reason for this is that Storm expects the individual tasks of bolts to be independent, and does not support communication between tasks. This creates a problem since we require our parallel entities to act as peers to exchange synchronization information with each other. In order to work around this issue, we have implemented the individual ‘tasks’ of our architecture as if they are different types of bolts. Since these ‘different’ bolts, which actually run the same code, directly receive tuples from the spout, running the topology in this way also effectively enables parallelisation. This novel manner of setting up a topology in Storm requires setting up the interconnections between components for each level of parallelism. Even though the changes required are minimal, we have anticipated the inconvenience this would pose the practitioner - and therefore the generation of the source code required is able to be performed through a script that we wrote to automate the process.

Figure 6.7 represents the architecture as implemented on Storm and illustrates the point made above.
InstanceSpout
Separate stream connections to each allocator. Round-robin sending of instances.

AllocatorBolt
Maintains ‘n’ independent micro-clusters. Performs allocation of data instances to micro-clusters based on local information.

6.3.4 Results and Discussion

Experimental setup

All experiments on the parallel architectures were run on an 8-node cluster running on OpenStack. Each virtual machine ran on one CPU core and 4 GB of RAM, with 10 GB of disk space. Each of the machines runs CentOS 6.5 and Java 1.6.0_45. The
Apache Storm version used was 0.9.2 and the Redis version used for the common-store architecture was 2.8.17. The nodes were placed in the same availability zone to minimise network communication overhead. In the common-store architecture, a separate node with the same specifications as the clustering worker nodes was utilized to store the common Redis cache.

Performance of the centralised version was observed by running the algorithm separately on a single virtual machine with the same specifications as mentioned above. Two data sets were used in the experiments, one synthetic with 100000 data points and one derived from Twitter data with 5513 unique tweets. The synthetic data set was of three dimensions, with points sampled from a Gaussian distribution to lie around five predetermined points which were to serve as cluster centroids. The Twitter data set was taken from the Sanders Analytics Twitter sentiment corpus which contained tweets relating to five categories, and the vector space model generated from the Twitter data set was of 300 dimensions. These data sets were played as streams for 15 minutes in order to gauge the throughput and accuracy levels obtained. Since Apache Storm works on a pull-based system, the stream rates able to be maintained are the throughput values reported in the results section.
Evaluation Metrics

The full CluStream algorithm can be thought of as having two separate stages - the first stage updates the micro-clustering, which serves as the summary of the ingested data points, and the second stage forms the final clustering using a traditional clustering algorithm by using the micro-cluster centroids as pseudo-points. The running of the second stage is infrequent, on-demand, and is not required to keep up with the stream rate and therefore is not crucial in a performance sense. Therefore, the metrics that we have chosen to evaluate the performance of our distributed algorithm are focused on the online portion of the CluStream algorithm.

The online and first stage of the CluStream algorithm solely deals with updating the micro-clusters. Therefore, any metric that measures performance needs to judge the accuracy that our algorithm is able to provide when performing micro-clustering. The micro-clustering stage works by placing each and every data point into the micro-cluster it is closest to. If the decision of which micro-cluster to place a data point in to is equivalent when running the CluStream algorithm in a single machine and when running in a distributed manner, the micro-clusterings would be equivalent and subsequently any final clustering would be equivalent as well. Unfortunately, this is not the case when the algorithm is distributed, as there can only be limited communication between distributed nodes in order to maintain performance. Therefore, our
distributed CluStream algorithm would make erroneous decisions on which micro-
cluster to assign some data points to due to each node having only a limited view
of the actual global picture of the data. We have chosen the number of such erro-
neous decisions our distributed version of the algorithm makes as compared to the
traditional single-machine algorithm as our accuracy metric as this most closely iden-
tifies how accurate the online stage of the distributed clustering is as compared to the
online stage of the single machine version of clustering.

The throughput of the implementation is measured by the number of stream in-
stances that the topology completes processing per second. This metric is obtained
by recording the number of acknowledge messages that the data originating spout
receives per second. As an acknowledge message received by a spout indicates that
a stream data instance has been fully processed by all computation nodes this serves
as a suitable metric for measuring throughput of the system. We have presented the
raw numbers so recorded at the spout after normalizing by the number of features,
the rationale for which is given below

In a clustering application where the computation of similarity is dependent on
every feature in the feature vector, the throughput will decrease with an increasing
number of features. As our implementation aims to provide a general method of
clustering on the fly, we aim to provide results which would enable a practitioner
to judge how fitting our incarnation of clustering would be to their particular application. As feature vectors differ widely between applications, we have provided throughput scores normalized by the number of features. This is done with the aim of making it easier for a user to ballpark the throughput they could hope to observe based on the number of features employed in their custom application.

**Accuracy comparison - number of micro-clusters**

It is seen in Figures 6.8 and 6.9 that the number of allocation errors made reduces consistently with increasing numbers of micro-clusters in both data sets. This is due to an increase in the number of micro-clusters providing better summarisation of the input stream, along with better input granularity in to the final clustering stage. It is also seen that there is not much separating the two architectures with regards to accuracy with number of micro-clusters, with the number of errors made by the common-store architecture being marginally higher than the number made by the decentralised architecture. Given that the two architectures essentially run the same algorithm, with the most significant difference being in the synchronisation step, this result is as expected.
Accuracy comparison - parallelism

Figures 6.10 and 6.11 illustrate the effect the parallelism level has on the number of allocation errors made by the distributed architectures. A thousand micro-clusters were used in the experiments. When the parallelism is equal to one, the decentralised distributed version effectively reduces to the centralised version, with no effect from the synchronisation step, which means that zero errors are seen when it is compared to the centralised version.

At first glance it would seem that the same should hold for the common-store architecture as well. However, it is to be noted that the aggregation and synchronisation steps are performed regardless of the actual number of nodes employed. Since these only occur periodically, the state of the common store is not a perfect reflection of the changes that were made in the centralised micro-clustering for the same data at all times. These deviations in the structure maintained in the common store from the state that would be present in the centralised case is what results in the non-zero number of errors seen for a parallelism value of one in the common store architecture.

It is seen that the number of allocation errors increases with increasing parallelism values. This is because the major cause of allocation errors - the difference between local information in the individual nodes and global information about the overall
micro-clustering that the centralised version possesses - increases with more parallel nodes. It is seen that the trend is the same in the two data sets, as well as between the architectures, with not much to separate the two with regards to individual values.

**Accuracy comparison - synchronisation period**

It is seen that the number of allocation errors made increases as synchronisation between nodes becomes more infrequent, which is represented by increasing synchronisation period on the x axis in Figures 6.12 and 6.13. It is seen that the number of errors is higher in the decentralised case for the lower dimensional synthetic data, and is higher in the common store architecture for the high dimensional Twitter data. The difference in accuracy levels is seen to become lesser at higher values of synchronisation periods.

**Throughput comparison - number of micro-clusters**

It is seen in Figure 6.14, which shows the results of experiments at a parallelism value of eight and changing numbers of micro-clusters, that at a low number of micro-clusters, the performance of running the micro-clustering algorithm on a single machine eclipses that of the parallel version. This is because at low numbers of micro-clusters, the task to be executed in the micro-clustering algorithm is computationally inexpensive. The overheads incurred in running a distributed version of the
algorithm are therefore much more significant as compared to the computational expense in such a computationally easy task. In this case, the gains of parallelisation are not sufficient to offset the overheads incurred in the parallel execution of the algorithm, which results in better performance being seen in the centralised case.

However, at high numbers of micro-clusters, and correspondingly more involved tasks of execution, it is seen that both architectures comfortably outperform running the algorithm on a single machine. There is not too much separating the performance of the two architectures, however, as both run essentially the same algorithm, and because the number of micro-clusters is not a parameter related to the architecture of the system.

*Throughput comparison - parallelism*

It is seen in Figure 6.15 that the decentralised architecture outperforms the common-store architecture at low levels of parallelism. However, the performance of the decentralised architecture drops off as the parallelism value increases, which eventually leads to the common store architecture bettering the decentralised one. This issue in scalability is due to the increasing level of communication that has to be maintained between nodes in the decentralised case. The communication overhead would
increase with quadratic complexity in the number of nodes, which is why we see a sharp drop off of performance after the threshold of four nodes. The performance in the common store architecture, on the other hand, increases steadily with the number of bolt tasks.
However, it is also seen in Figure 6.15 that increasing parallelism does not yield higher throughput for parallelism values higher than eight. This is because eight virtual machines were employed in the implementation. Increasing the parallelism value above eight would require the running of more than one bolt task on the same virtual machine. This introduces the overhead of context switching between the multiple bolt tasks that now need to run on the same machine. It is seen that this overhead of context switching nullifies any benefit that the raw increase of the number of bolt tasks through increased parallelism provides. Essentially, even though we have more bolt tasks due to an increased parallelism setting, the need to switch between multiple bolt tasks which now have to run on the same machine does not allow us to see increased throughput values.

Throughput comparison - synchronisation period

Increased values of the synchronisation period parameter would mean that the global synchronisation that brings the local information in line with the global state would happen more infrequently.

In the common-store architecture, all allocator tasks access a single common store in order to update themselves. As multiple agents are accessing the same resource, this inevitably leads to contention, which would reduce performance of the system.
Therefore, the more infrequently this costly synchronisation operation needs to happen in the common store architecture, the better the performance would be. We see this effect in Figure 6.16, with the throughput steadily increasing with increased synchronisation period, and then levelling off at higher values where the synchronisation ceases to be a bottleneck.

In the decentralised scenario, contention would not exist as a single resource is not being accessed. Shorter synchronisation periods would result in small batches of information being sent more frequently, and longer periods would mean that information about a higher number of micro-clusters would be sent infrequently. Since the ultimate throughput seen is a function of these two competing effects, we see little variation in the throughput with respect to synchronisation period in the decentralised architecture.

6.4 Conclusions

In this chapter we address the scalability problem when dealing with streaming data. We look at two different versions of the scalability problem, that of a large number of streams as well as of very fast single streams. The PSF algorithm is modified in order to run in a scalable manner for a large number of data streams with minimal forecast accuracy loss with increased throughput. In the scenario of a single fast
stream, we propose methods which divide up the stream among multiple machines and address the challenge of synchronisation between the machines with respect to micro-clusters. We propose both a shared memory and a non-shared (peer-to-peer) algorithm to address the synchronisation problem. Large numbers of micro-clusters are shown to be able to be maintained with these algorithms with linear scalability in many cases. We also provide guidance to practitioners on the choice of algorithm and parameter values by discussing the relative advantages and disadvantages of both methods.

The approach taken to scale the forecasting algorithm is to modify the labelling step of PSF (which involves clustering) with micro-clustering which is able to work on data streams. This approach may be extended to any other algorithm (which may or may not be related to forecasting) which has a preprocessing step which involves clustering. Further, micro-clustering by itself can be used as a stream summarisation technique, and our distributed clustering algorithm may be utilised to provide scalable solutions in such scenarios.

Limitations were seen in our methods when maintaining low numbers of micro-clusters, where the overhead of the distributed system eclipsed any gains of parallelisation due to the low computational expense involved with maintaining the small number of micro-clusters. Other limitations were seen in relation to the number
of allocation errors, which was seen to increase with increasing parallelism values and synchronisation periods. This trade-off would be application dependent and it would be instructive to plot figures which relate errors to parallelism values and synchronisation periods to determine the values that provide the best compromise.

In relation to the comparison between the shared and non-shared algorithms, it was seen that the decentralised non-shared algorithm performed better at relatively lower levels of parallelism, along with being able to handle shorter synchronisation periods (due to being a no-contention algorithm). In both algorithms it was seen that parallelism values above the number of physical machines degraded performance due to the context switching that needs to happen when running multiple tasks on the same machine.

Improvements on our algorithms may be made through an approach which does not require individual allocator tasks to maintain a complete ‘n’ number of micro-clusters. In this approach, individual allocator tasks would hold a subset where the centroids of micro-clusters in a particular subset are close to each other. Since each node only holds a subset of micro-clusters, it is important that data points be forwarded to nodes that hold micro-clusters that are at least approximately close to the data point. Therefore, it is proposed to execute a Locality Sensitive Hash search on the data point to find the subset which is most likely to contain the closest micro-
cluster to it, and forward the data point to the task managing that micro-cluster subset. The reduction of both memory usage and micro-cluster search space due to maintaining only a subset of micro-clusters in each allocator task could conceivably result in throughput improvements in the algorithm.

Further improvements may be made on the communication overhead that is present in the decentralised algorithm for synchronisation, which is a trade-off for eliminating the single point of contention. This improvement may be made in the form of a message passing scheme based on gossiping in order to reduce the messaging complexity. What we propose is a change to the nature of message passing: in the present scheme the message passing occurs from each task to every other task; in gossiping the message passing would be based on a peer-to-peer scheme. Such a scheme should be possible to be implemented in Apache Storm in a similar manner to that which is proposed in the decentralised architecture, and would improve overall performance characteristics.
Conclusions

In this thesis we have explored the challenges of solving the multi-step forecasting problem in the setting of smart cities. This problem is important because forecasting allows for better city planning, better understanding of the behaviour of the city and also allows for identifying anomalous activities. In order to realise these benefits we provide novel methods to perform multi-step forecasting (i.e. 24 hours into the
future) in a more accurate, robust and scalable manner in this thesis. We summarise our contributions in the next sections.

7.1 Kernel compositions to incorporate prior knowledge

We focused on improving the accuracy of multi-step Bayesian forecasting by providing methods to incorporate prior information important in smart city settings. One such source of prior knowledge is related to the Monday-Friday periodic working week structure. Our work had the goal of modelling multiple such periodic effects and their interactions in the same model. We hypothesized that incorporating both daily and weekly periodic information could potentially result in improved prediction accuracy, and that this was a rich source of prior information that was underutilised in prior work. Further, the incorporating of more prior knowledge would make the predictions more robust against anomalous recordings in the training data.

We used the state-of-the-art Gaussian Process Regression method to perform forecasting, due to its support for intrinsic feature selection, ability to model complex functions, and most importantly to this work their versatility in defining remarkably flexible priors. Incorporating prior information to Gaussian Process Regression is done through kernel functions, and is not a straightforward exercise, with much work in the literature where the main contribution is the specification of kernel com-
positions for a particular problem domain. In this work we developed kernel compositions which allowed the specification of daily and weekly periodicities and the interactions between them in the same model.

We developed a number of such compositions: one which allowed the switching between daily and weekly periodicities, one which emphasised weekly periodicities, and one which learnt the relative weighting between the two kernels through the hyperparameter optimisation process. We demonstrated that our kernel compositions provided much better accuracy compared to the formulaic kernel compositions commonly used.

In this work we also investigated improving the hyperparameter optimisation process by setting hyperpriors on the hyperparameters. We found that the addition of hyperpriors also helped with accuracy, and improved the robustness of the predictions in relation to a range of hyperparameter values. The demonstration of all results was done on publicly available real-world datasets with widely different applications, which is suggestive of the potential generalisation capabilities of our methods to other use cases with similar working-week periodicities.

The work in this thesis focused on the periodicity aspect of human behaviour in a smart city, and provided methods to incorporate information related to this aspect of human behaviour. Future work might possibly involve methods to incorporate
other prior knowledge that is known about human behaviour, through fields such as behavioural science.

7.2 Multi-task learning for robust predictions

In this work we focused on the problem of missing training data in sensors. Missing data is an unavoidable problem in real-world scenarios, and especially so in smart city settings with sensors spread across a metropolis which may malfunction and are vulnerable to vandalism. In order to mitigate the effect of missing training data on prediction accuracy, rather than just making do with the data of that particular sensor, we focus on methods of using data of other correlated sensors in order to improve prediction accuracy. This would be a particularly useful method of attack to the problem in the smart city space, given the abundance of sensors deployed in such applications nowadays making it likely that there would exist sensors with similar time series behaviour to each other.

We provided a representation which allowed for such inductive knowledge transfer from related sensors to sensors with missing data for which we want to perform prediction. We proposed a multi-task learning (MTL) kernel to be used in Gaussian Process Regression which provides a weighted model between inter-task and intra-task learning (where in this scenario a ‘task’ is a sensor). We also express this multi-
task learning kernel as a linear combination of commonly used kernels. This allows our methods to be directly implemented using popular machine learning toolkits which provide these commonly used kernels, as well as methods to implement linear combinations of those kernels.

We observed that our multi-task learning method provided higher accuracy than vanilla Gaussian Process Regression (GPR), and that it performed better compared to other competitive kernel-based methods such as Support Vector Regression (SVR) and canonical methods such as ARIMA. Our comparison baselines also included stronger baselines such as GPR supplemented with a simple replacement of values from other sensors, as well as SVR supplemented in the same manner. MTL was the clear best performer out of all these methods as seen in the accuracy results.

Our demonstration of results was based on real-world data which had missing data simulated in order to have ground truth for evaluation, where the error simulation included generating both random and bursty errors to more closely reflect real data loss scenarios. In addition to demonstrating improved accuracy in data loss scenarios, we also provided results for various error generation scenarios, for various parameter settings and also for relative computational costs for error mitigating techniques. It was seen that our error mitigating methods did not increase the computational expense significantly, and that the gains in accuracy were seen across parameter settings, error
generation scenarios and across individual sensors in the datasets as well.

7.3 Label-based forecasting

Forecasting algorithms for smart city settings need to be scalable to the increasing number of sensors in such settings due to the cheap availability of sensors and the Internet of Things paradigm. We focused on developing algorithms which can forecast on large volumes of data. We first focused on a base algorithm on which to base our scalable algorithm.

Speed was the primary requirement of a base algorithm for prediction in this context. In this regard, it was seen that the PSF [71] algorithm was particularly suited for the purpose with its unique discretisation step, which allows for the algorithm to operate on generated labels instead of the high-dimensional raw time series data. This allows for predictions that are orders of magnitude faster compared to other methods. Further benefits of the algorithm include tunable computation complexity, minimal parameter tuning requirements and interpretability of the model which enabled analysis of the effect any modifications made would have on the algorithm.

However, there still was much to be desired in terms of accuracy, especially relating to the variability in error rates between different days. The interpretability of the model allowed for the identification of the causes of errors, which motivated us to
address the limitations of the algorithm and propose enhancements which would improve the accuracy levels achievable without compromising too much on speed of forecasting.

Our enhancements to the algorithm included filtering and windowing methods, and also methods to incorporate external information (temperature in this case). Further significant improvements were made by using an ensembling technique which allowed the algorithm to utilise temporal locality in the time series.

The experiments were done on the smart city and pedestrian real-world publicly-available time series datasets, where the techniques we proposed resulted in improvements of prediction accuracy of up to 47%. We also provide results for the computational efficiency of techniques, where it is seen that though our enhancements add to the computational cost of the algorithm, this does not result in a significant reduction in performance. This was seen to be a favourable trade-off given the prediction accuracy improvements derived from the proposed techniques.

We next use this enhanced base algorithm in our methods that allow for scalable prediction.
7.4 Scalable label-based forecasting

With the explosion of devices and sensors connected to the Internet, the volume of data being generated nowadays has reached unprecedented levels. This also applies to smart city settings with the advent of cheaper sensors and increased connectivity. Therefore, forecasting methods should be able to manage this increasing amount of data being generated. Another common modern-day requirement in settings of smart sensors recording data from dynamic environments is for the algorithms to be able to operate in a real-time online manner on streaming data.

The associated scalability challenges can take two forms:

1. There is a large number of data streams that need to be processed, and this large number cannot be processed effectively by a single machine

2. A single stream has quite high velocity and therefore cannot be processed effectively without parallelisation

We address both these issues of scalability. The first step of improving scalability in Scenario I was by using a low computation cost prediction algorithm at the outset (PSF), and enhancing the algorithm to provide higher accuracy without too much of a cost sacrifice as described in the previous section. We then use this label-based low
computation cost algorithm as the base predictor algorithm in our scalable system. The initial part of the algorithm (labelling using clustering) was implemented using the micro-clustering technique [5] which was purpose-built for running on data streams in constant space and time. We use micro-clustering to perform the initial labelling step of PSF in our scalable method.

The scalability challenge in Scenario II was solved by dividing up the single stream across multiple processing nodes. Dividing up the stream in this manner results in no single processing node seeing the entirety of the stream. This can result in a divergence of results that we would see compared to the single machine scenario where one machine operates on the entire stream. Therefore the bulk of the work in dealing with Scenario II is the synchronisation of information between the nodes. We solve the synchronisation problem using two methods - one using a global in-memory store, and the other using a peer-to-peer-like communication system. We finally output one final global clustering. Though it might seem at first glance that having one final global result would require the intermediate results to be fed to one single processing node, we develop mechanisms so that even this final global result calculation can be parallelised.

We demonstrate our results on publicly available and real-world datasets in relation to throughput and accuracy levels observed. We provide the results on a com-
ponent level in the scalable architectures, where we saw significant improvements over single machine throughput and in many instances saw close to linear increase in throughput up to the number of physical machines employed in the parallel implementation. We also provide advice to practitioners who may use our algorithms on the numbers of each of the components that should be deployed according to the specific needs of the application.

7.5 Limitations and Future Work

7.5.1 Kernel compositions to incorporate prior knowledge

The most important future work would be related to improving performance, since an inherent limitation of Gaussian Processes is their poor scalability due to the method being of complexity class $O(n^3)$. It is inevitable that our Gaussian Process models would also be affected by this limitation, which would be particularly true for our elaborate models, especially those with automatic tuning of weights. Future work could look into compromises between accuracy and scalability through the use of approximation methods for Gaussian Processes such as reduced rank approximations.

Further future work could be done into the extending of our models by the addition of holiday effects into the mix as well. The extension to holiday effects could
be performed in a manner very similar to that used for the weekday versus weekend
effects (in effect this would require an addition of a holiday feature to the representa-
tion), though the accuracy benefits may or may not be significant based on the
problem domain.

7.5.2 Multi-task learning for robust predictions

Future work in relation to multi-task learning could possibly relate to a hierarchical
method of multi-task learning (MTL), where we would have MTL running across
sensors with different correlation levels, and then weight the output accordingly to
get the final prediction. For example, we could have separate MTL-based algorithms
for correlation levels 0.9, 0.7 ... 0.1, and then output the final prediction value by
weighting the output of prediction based on 0.9 correlation the highest. This would
essentially be an ensemble of MTL-based predictors.

It was seen that our multi-task learning models did not perform well in terms of
accuracy (or performance) when used in combination with our methods to incorpo-
rate prior information related to the working week. This is most possibly due to the
increasing complexity of the model making it harder to generalise accurately and the
increased complexity of the optimisation problem making it harder to find optimum
minima. This limitation could possibly be addressed by using low rank approxima-
tions for Gaussian Processes, which would allow for more training data to be used to offset these challenges, though it should be noted that these approximate methods come with their own tradeoffs as well.

Another limitation of our multi-task learning models is that they do not consider spatial relationships between sensors. Incorporating spatial information of sensors (particularly to the decision process of choosing related sensors to a sensor with missing training data) could possibly lead to improved mitigation of prediction accuracy when challenged with low-quality training data. Incorporating spatial information could also potentially enable the forecasting for locations for which sensors are not currently installed. This has potential applications in the area of city planning, where recommendations can be made on where to install new sensors (installation would be preferred in locations where the variance of prediction is high).

7.5.3 Label-based forecasting

Future work in relation to label-based forecasting could include investigating whether other types of exogenous variables (e.g. occupancy of a building based on WiFi data, humidity values of the day) could help in improving prediction accuracy.

In this work the clustering was done on individual elements on a stream for labelling purposes of each data. Further work can be done in attempting to perform
clustering individual streams themselves. Clustering streams will also reduce the number of elements that a prediction algorithm has to work with, and therefore such an algorithm could be used in combination with almost any general-purpose prediction algorithm. It may also be useful to look at the effects of using correlation-based similarity measures in the clustering process.

In relation to prediction performance, some of our enhancements (e.g. temperature equivalence and ensembling steps) result in a drop in prediction speed. Though these enhancements can be switched off in scenarios with a higher emphasis on speed, a mechanism where such adjustments to the enhancements could be done dynamically based on the current rate of data ingestion and possibly the resource usage of the nodes could be an interesting extension of our work.

7.5.4 Scalable label-based forecasting

Limitations were seen in our methods when attempting to maintain a low number of micro-clusters, where the overhead for running the distributed algorithm were higher than the benefits gained through running the micro-cluster maintenance algorithm in parallel (due to the maintenance being of low computational expense for low numbers of micro-clusters). Further limitations were seen in the increasing number of allocation errors with respect to high synchronisation periods and parallelism
values. This limitation could be addressed by plotting figures which relate the errors to changing synchronisation periods and parallelism values and choosing the values that provide the best compromise between errors and throughput. Further, it was seen at parallelism values greater than the number of physical machines that performance degraded noticeably due to the context switching that needed to happen due to needing to run multiple tasks on the same machine.

Improvements could include methods which do not require each node to maintain a complete set of ‘n’ micro-clusters. Each node running an allocator task could hold a subset of micro-clusters, where the micro-clusters in that node would have centroids close to each other. A Locality Sensitive Hashing algorithm could be used to efficiently search for the subset that would most likely contain the closest micro-cluster centroid to a given data point, and forward it to the node maintaining the subset which contains that closest micro-cluster. This will result in a reduction in the search space for micro-clusters and as well as memory usage on each node, which could improve overall performance characteristics.

Improvements in the decentralised synchronisation could be related to reducing the communication overhead currently present due to the broadcast-style of this communication. Messaging based on gossiping could be investigated to reduce messaging complexity which could potentially result in throughput improvements.
7.5.5 Other

In general other kinds of smart city data could also be looked at. For example, one area could be predicting road traffic, which could be extended to look at potential impacts of traffic interventions or events (e.g. accidents). Predicting the change that can happen due to an intervention can be used to provide automatic warnings to users about the intervention and propose new routes to take. Correlating pedestrian data with social media data (e.g. tweets) could also help uncover useful patterns of behaviour and also aid with predictive analysis.

The methods that are described in this thesis are closely associated with the models that are employed, though the ideas may be used in improving other models as well. For example, incorporating prior periodicity information is done for Gaussian Processes, since Gaussian Processes are a natural choice when elaborate and flexible priors need to be constructed. All such prior information is incorporated in the kernel composition, and therefore could potentially be used in other methods that use kernels. The ideas behind the improvements to the PSF algorithm, chosen due to its unique fast prediction capabilities, could be applicable to improvements to other algorithms. The enhancements proposed are novel solutions to unique problems in smart city prediction scenarios, (for example incorporating prior information, dealing with low quality data, and improving scalability), using the most appropriate
models.

The work in this thesis has mostly focused on forecasting (multiple) future time series values given historical time series data. Other possible areas of work could include forecasting other types of quantities; for example forecasting anomalies could be an avenue worth exploring. Work that had a specific focus on forecasting anomalies could identify situations where intervention might be required in real-world scenarios. For example an abrupt predicted increase in power consumption might require an increase in planned power generation capacity. On a similar theme, as mentioned above, work may also be done on forecasting the effects of these interventions. For example the prediction of effects of a traffic intervention such as closing a road could be particularly useful. Finally, change point detection methods could be used to identify points at which concept drift occurs in time series, which again could inform manual intervention efforts in real-world smart city scenarios.

It would also be interesting to see how an ensemble of the Bayesian and label-based forecasting algorithms would perform. Gaussian Process Regression could potentially be used to incrementally improve upon the forecasts that the label-based forecasting algorithm provides. A scalable rate-adaptive solution would also be interesting to be investigated where the choice of algorithm depends on the current streaming rate.
Appendix

Proof of Kronecker product and composite kernel formulation equivalence

We represent the input data points of the multiple sensors using one-hot encoding ($X$). The response variable values for the sensors are stored in a stacked representation ($Y$), where $y_{ij}$ represents the $j^{th}$ response variable value in the $i^{th}$ sensor.

$$
X = \begin{bmatrix}
0 & 1 & \ldots & 1 \\
0 & 1 & \ldots & t_n \\
\cdot & \cdot & \ldots & \cdot \\
1 & 0 & \ldots & t_n \\
\end{bmatrix} \quad Y = \begin{bmatrix}
y_{11} \\
y_{12} \\
\cdot \\
y_{21} \\
y_{22} \\
\cdot \\
y_{n1} \\
y_{nn} \\
\end{bmatrix}
$$

We develop the covariance matrix $K_{\text{data}}$ shown below by evaluating the kernel function $k_{\text{data}}(t, t')$ on the time values (last column) of every pair of input points in $X$. The resulting covariance matrix is shown below, where $k_{ij} = k(t_i, t_j)$
The first \( n \times n \) block is \( K_{\text{block}} = k_{\text{data}}(X_0, X_0) \) in Equation 4.4. Therefore, for \( B = \begin{bmatrix} k+1 & i \\ i & k+1 \end{bmatrix} \), which encodes a higher intra-task dependence (diagonal \( k+1 \) terms) relative to inter-task dependence (off diagonal terms), the final covariance matrix takes the following form upon the expansion of the Kronecker product:

\[
B \times k_{\text{data}}(X_0, X_0) = \begin{bmatrix}
(k + 1) \times K_{\text{block}} & 1 \times K_{\text{block}} \\
1 \times K_{\text{block}} & (k + 1) \times K_{\text{block}}
\end{bmatrix}
\] (7.2)

In the covariance function formulation, we first evaluate \( k_{\text{Linear}} \), which computes the inner product between all pair of input points (vectors formed out of the first two columns). Every element in the resulting matrix is multiplied by a configurable hyperparameter \( k \), which results in the following matrix:
The addition of the covariance matrix generated using the constant kernel function $k_{\text{const}}$ (with its constant set to 1 to result in the identity matrix) produces the following:

$$
\begin{bmatrix}
  k & k & \ldots & k & o & o & \ldots & o \\
  k & k & \ldots & k & o & o & \ldots & o \\
  . & . & \ddots & . & . & . & \ddots & . \\
  k & k & \ldots & k & o & o & \ldots & o \\
  o & o & \ldots & o & k & k & \ldots & k \\
  o & o & \ldots & o & k & k & \ldots & k \\
  . & . & \ddots & . & . & . & \ddots & . \\
  o & o & \ldots & o & k & k & \ldots & k \\
\end{bmatrix}
$$

It can now be seen that the elementwise multiplication of this matrix with $K_{\text{data}}$ (Equation 7.1) results in the same matrix as in Equation 7.2.
Bibliography


[38] Dublin City Council (2016). *Smart Dublin - Open, Connected and Engaged*. Dublin City Council.


206


