Development of an Autonomous, Self-Optimising Machine Learning Framework for use in Manufacturing Applications

by

Carl Berry

A thesis submitted in partial fulfilment for the requirements for the degree of Doctor of Philosophy at the University of Central Lancashire

October 2022

Declaration

I declare that while registered as a candidate for the research degree, I have not been a registered candidate or enrolled student for another award of the University or other academic or professional institution.

I declare that no material contained in the thesis has been used in any other submission for an academic award and is solely my own work.

Signature of Candidate :	Carl Berry
Type of Award :	Doctor of Philosophy
School :	School of Engineering

Acknowledgements

I would like to give my thanks to my supervisors; Geoff Hall, Lik-Kwan Shark and Bogdan Matuszewski for their tireless help and advice throughout this project, without whom this work would not have been possible to complete.

I would also like to thank my family, Ruth, Andrew and Catherine for their support and understanding whilst I completed this work. And finally, Ruby, the family cat for sitting with me through late nights and long weekends.

Abstract

Traditionally manufacturing processes have used a wide variety of sensors to provide process control, quality and productivity metrics. Industry 4.0 introduces the concept of combining and analysing sensor derived production data to offer greater insights into all aspects of manufacturing operations. A key area of advancement is the ability to process and analyse large volumes of data, in real-time, to provide to process control. The objective of this thesis is to define a framework that enables adaptive, optimised production control via the use of automated machine learning algorithm selection.

The complete framework is capable of adaptively processing streamed production data directly from manufacturing processes along with other appropriate sources. To achieve this, a second longer term channel is used to autonomously evaluate and optimise competing algorithms and data strategies to select the most appropriate solution for near real-time control. A series of experiments demonstrates that the framework is suitable for production control applications which benefit from a focus on single or multiple accuracy metrics.

The framework can automatically switch algorithms via a supervisory mode. This allows it to take into account additional factors such as concept drift, production changes, computational complexity and algorithm stability. Algorithm switching is based on a flexible optimisation strategy for algorithm performance over variable time periods. Experimental results are provided for two optimisation methodologies.

Further development of the framework will involve full automation of the real-time algorithm selection method. This will remove the need for specialist data processing knowledge and hence allow the framework to be deployed in a wide range of existing manufacturing companies.

Contents

Abbreviations Used

ABC	-	Artificial Bee Colony
AI	-	Artificial Intelligence
ANN	-	Artificial Neural Network
ASIC	-	Application Specific Integrated Circuit.
BER	-	Balanced Error Rate.
CASH	-	Combined Algorithm Selection and Hyper-parameter (Optimisation)
EDM	-	Electronic Discharge Machining
FA	-	Firefly Algorithm.
GA	-	Genetic Algorithm.
MLP	-	Multi Layered Perceptron.
MSE	-	Mean Squared Error.
PPM	-	Parts Per Million.
PSO	-	Particle Swarm Optimiser.
ROCAUC	-	Receiver Operating Characteristics Area Under Curve.
SGD	-	Stochastic Gradient Descent.
SME	-	Small to Medium Enterprise.

Chapter 1

Introduction

1.1 Project Background

As manufacturing moves towards the adoption of Industry 4.0 (Koshy 2019, Hiskey 2017) there is a desire to monitor manufacturing processes and make predictions for control and/or predictive maintenance purposes. One such framework developed for this problem is that of the Lambda framework (Marz and Warren 2015). Here a control/prediction engine runs alongside a longer term channel. The channel produces competitive output based on the same inputs and conditions by trying different algorithms and input configurations. In the standard Lambda setup these models are used by data scientists to compare with the existing running model to see if they are a better solution. For many industrial implementations this creates two problems. First is that it is often the case, especially in a small to medium enterprises (SMEs), that there is no-one on the staff qualified to undertake the analysis. This, the lack of qualified staff, is a particular barrier to small manufacturers already facing the daunting prospect of implementing Industry 4.0 technologies for the first time. It is the aim of this work to use a machine learning solution to replace the data scientist / analyst role but this highlights even further the second problem, how do we identify the "best" algorithm?

Combined Algorithm Selection and Hyperparameter Optimisation (CASH) problems are an active area of research (Thornton *et al.*, 2013, Feurer *et al.*, 2015, Golovin *et al.*, 2017). These techniques use Bayesian optimisation to tune algorithms and parameters to produce the best results, for many machine learning implementations. "Best" often means the most accurate and this is often measured in terms of the mean absolute deviation, this being the metric that measures the mean average of the magnitude of the difference between the predicted and observed values, this is often simply termed "accuracy".

The mean absolute deviation can hide a number of factors. For example an algorithm that is consistently wrong by a small factor can score lower using this than an algorithm that swings between being accurate and inaccurate. This may be inappropriate in an industrial context where a large error, even once, could have far reaching consequences. There are a wide range of comparative metrics that can be used to rank regression algorithms and even more if classification is considered.

1.2 Regression vs. Classification

The problem, as set, is a regression problem. The system is attempting to predict the precise value of the Ethylene concentration and as such the five algorithms being tested are all regression algorithms. In most manufacturing processes, however, the goal is usually to keep a value within a given range rather than predict an absolute value. This gives a margin of error for the regression algorithms. After an initial visual inspection of the data three classes were identified:

- Below expected limits (0-7 ppm)
- In tolerance limits (7-17 ppm)
- Above expected limits (17+ ppm)

This sets approximately 40% of the data to be within acceptable limits, it is an entirely artificial set of borders, but it gives a reasonable simulation of a running manufacturing process. So long as the regression estimate is within the same class it is considered "within tolerance". This allows the work to include metrics that are more commonly used to measure classification algorithms rather than regression algorithms. A brief look through the literature will demonstrate that there are far more metrics concerned with classification and that this is a much more widely studied area than regression. For this work the number of classification metrics has been limited to a small number. The reason for this is that the process is still fundamentally a regression problem and if the experimental phase included too many classification measures there is a danger of over-powering the regression metrics and biasing the results in terms of accurate classification over poor regression.

1.3 Optimisation

In order for the framework to select a best performing algorithm configuration over an extended period of time it needs to be able to evaluate a large number of possible combinations. As the number of potential algorithms and hyper-parameter increases this becomes impossible to do exhaustively so it must utilise optimisation techniques to look for a good solution to the problem. There are many such techniques and they often take inspiration from observed animal behaviour in nature, hence techniques such as bee colony (Yuce *et al.*, 2013), firefly (Fister *et al.*, 2013), Particle Swarm Optimisation (Yang, 2021) algorithms along with many others. The goal of these optimisers is to conduct a search of the potential solutions in such a way as to find good solutions in a reasonable amount of time and computation power (this is discussed in section 7.1). The techniques used in this work are the bee colony and firefly optimisation strategies.

1.4 Research Motivation

The motivation behind this work is to help develop the usage of machine learning in the industrial SME sector, where specialist knowledge of data analytics and machine learning is scarce and the techniques found often in academic literature are not yet in general use. In order to do this a framework has been created that will work with a wide variety of algorithms and datasets and that has the capacity to run in a fully autonomous mode once setup has been completed. Many of the systems available for use at present that claim to help users with machine learning tasks either assume a deep knowledge of the subject or make very broad assumptions that could lead to poor performance on behalf of the system in being developed. This work describes a framework that would genuinely allow a user with little to no knowledge of the field to gain real benefits from machine learning.

1.5 Aims and Objectives

The aim of this work is to produce a software framework, suitable for use in the manufacturing industry, that can autonomously evaluate, tune and apply a machine learning algorithm to a process. The framework is designed to remove the need for data specialists, beyond initial setup, to allow for usage by SMEs. The framework considers current and previous algorithm performance combined with environment information and system performance to optimise the autonomous swapping of different algorithms. The system does this to ensure that the optimal algorithm is being applied to the process over a set amount of time.

1.6 Contribution To Knowledge

The work presented here has described and developed a framework for autonomously handling machine learning control of an industrial process including continued development and autonomous changing of controlling algorithms. To that end the work has shown new contributions in the areas of :

- Consideration of a combination of accuracy metrics for tuning performances of a machine learning algorithms to best describe the desirable properties required to maximise the outcome of an industrial process rather than the selection or evaluation of a machine learning algorithm based on a single accuracy measure common in most currently available systems.
- Scalable testing of multiple machine learning algorithms and optimisation of the results for a time sensitive sequence.
- Consideration of factors outside of performance selection of machine learning algorithms that take into account both the wider process under consideration and the drifting of targets along with the impact of changing a controlling / prediction algorithm mid-production and the wider effects this would have.

1.7 Report Structure

This report begins by reviewing the current state of the art surrounding machine learning in industry along with considerations of metrics for measuring accuracy and a discussion of machine learning algorithms. The report continues with a review of current methods of automated machine learning tuning, machine learning applied to distributed processing and finally considers the role of lifelong machine learning to the system. An overview of the system follows exploring the architectures involved in the framework. Ways of measuring the accuracy of the system are then discussed followed by the algorithms used in the experiments. The report then describes a series of experiments run to test the feasibility of the framework and discusses the results. The report then looks at two methods of optimisation that were trialled in order to ensure that the best performing algorithm is being used at any given time. The environment around the process is then considered, discussing topics such as concept drift, the time taking to change algorithms and the benefits of doing so. The report concludes with final conclusions and plans for future work.

Chapter 2

Literature Survey

2.1 Introduction

The work presented here covers a wide range of areas where previous work has already been done, this chapter will review various parts of this and explain where this new work is positioned in relation to the current body of work.

2.2 Machine Learning in Industry

Machine Learning has long been an academic focus but more recently there has been more interest in developing machine learning systems that can be used in a variety of industries using a variety of algorithms. The usage of Genetic Algorithms (GAs) in the design of new products (Hsiao, Chiu and Lu, 2010). Abellan-Nebot and Subiron (2010) conducted a review of areas where AI could be used to monitor systems in manufacturing including sensors, feature extraction, treatment of signals and appropriate models for industry. Deng *et al.*, 2012 developed a framework for processing multi-relational data streams in a manufacturing environment, this has a certain commonality with the work presented here except that Deng's system dealt with data mining from traditional structured data sources and, as most systems here, left the final decisions up to a qualified human, Wang, (2007) also looked into data mining for the manufacturing industry, concluding that this was largely data that companies already had or could collect easily and that there were huge opportunities to improve many aspect of manufacturing by utilising them.

In examples of more specific applications, Ghosh and Sanyal, (2016) looked at using Multi Layered Perceptron (MLP) Artificial Neural Networks (ANN) in Electrical Discharge Machining (EDM) manufacturing. Alfaro-Cortes *et al.*, 2020 looked at using random forests to interpret out of control signals.

Deep learning has gathered a lot of attention in AI research and Arellano-Espitia *et al.*, 2020 looked at using such methodologies for fault diagnosis in electromechanical systems. This system used Deep Neural Networks and similarly Yasutomi and Enoki, 2020 used the same to inspect conveyor belt information in their paper.

In an extensive survey of the usage of data mining in manufacturing (Choudhary, Harding and Tiwari, 2008) it was found at the time that it was being used or had the potential to be used in many manufacturing areas such as quality control, job shop scheduling, fault diagnostics, maintenance, defect analysis, yield improvement, and monitoring of the process itself.

There are examples in the literature of discussions of where machine learning and AI fit into the space occupied by SMEs. An article published by Forbes in 2020 (Forbes, 2020) looked at a number of ways in which small businesses could benefit from machine learning, these included tasks such as data driven decision making, simplifying reporting and forecasting, customer prediction and others.

Despite this a survey amongst UK SMEs in 2017 found that less than a third were using any form of AI in their operation (UKTN, 2018). This was echoed in 2021 in a feature published by ForePaas, a cloud hosting company, (ForePaas, 2021) that identified the need for data scientists and the time taken to build models and infrastructure as barriers to SMEs from adopting machine learning techniques. An article by SME News in January 2022 (SME News, 2022) suggested that whilst upfront costs and the complexity of machine learning systems can be an obstacle to SMEs they felt that due to SMEs smaller sizes they had an advantage in the time taken to actually deploy a system. The SME News article also identified competitiveness and anticipating trends to move ahead of rivals as two of the key advantages of machine learning. In the 2022 O'Reilly survey on adoption of AI in enterprises, in the manufacturing sector less than 20% of respondents had machine learning in production, just over 45% were evaluating systems and 35% had no intention of using AI or machine learning (Loukides, 2022), in comparison the financial sector responded with 35% of respondents had AI in production, 45% were evaluating and around 25% were not considering it.

2.3 Metric Selection

When it comes to evaluating machine learning algorithms there are a number of strategies for gauging the quality of a algorithm in the literature, Caruana and Niculescu-Mizil, (2006), state that ”...different performance metrics measure different tradeoffs...” and that ”...it is possible for for learning methods to perform well on one metric but be suboptimal on other metrics...”, for their study they used a variety of eight different metrics to evaluate a number of different supervised learning methods. Gama *et al.*, (2011), further introduces the idea that streaming algorithms running on real-time data need to be considered in a different perspective to the traditional batch methods of evaluation. They consider the ideas of sliding windows and forgetting mechanisms in order to cope with continual data and how to accurately apply evaluation metrics in an environment where older data may be of diminishing value.

Specifically for regression applications such as the implementation used in this work, there is an interest in improving evaluations of algorithms beyond simple mean squared error (MSE), a number of publications have suggested alternatives, Torgo and Ribeiro, (2009), suggested an adaptation of the classification metrics of precision and recall for regression algorithms. Kaneko, (2017), used the concept of applicability domains, where prediction values are considered to be sufficiently accurate if they exhibit the same level of performance as the training samples did.

2.4 Machine Learning Algorithm Comparisons

There are a number of papers where authors have compared one set of machine learning algorithms with another in a variety of different applications; (Loza, Cisneros and Arreola, 2017) compared ANNs against regression models for river pollution contaminants forecasting. In this work they measured the absolute error of the models along with the mean average absolute percentage error. Sharif *et al.* , (2017) compared a number of regression algorithms (particularly Ridge, Elastic net and Lasso, that were used in this work) to try to predict oilseed rape yield under varying climatic conditions.

2.5 Hyper-parameter tuning and Automated Machine Learning

The need to be able to find good sets of hyper-parameters in machine learning algorithms is well established in the literature, (Drignei, Forest and Nychka, 2008) looked at running surrogate models for computationally intensive regression problems that helped to account for uncertainty in the more complex models parameters. Grömping, (2009) looks at the importance of hyper-parameters in regression algorithms both linear regression and random forests.

There are examples of systems that attempt to automatically tune hyper-parameters of machine learning algorithms. These systems are often termed CASH (Combined Algorithm Selection and Hyper-parameter) optimisation methods when the algorithm is selected as well. Thornton *et al.*, (2013), introduces Auto-WEKA, a system that selects both algorithm and hyper-parameter values using Bayesian optimisation that looks to minimise cross-validation error. AUTO-SKLEARN is introduced in Feurer *et al.*, (2015), and this time the system looks to minimise the balanced classification error rate (BER) which they define as "...the average of the proportion of wrong classifications in each class." again the method used is Bayesian optimisation. Google use an in-house "black-box" optimiser called Vizier (Golovin *et al.*, 2017) that runs on parallel Google data centres to allow for large scale testing, it again uses Bayesian optimisation. The drawbacks of Bayesian optimisation are discussed in section 7.2 of this work but also the author believes that one of the main problems of these systems is the targeting of a single metric to evaluate the success of optimisation.

2.6 Machine Learning and Highly Distributed Processing

The testing of large numbers of algorithms and algorithm configurations lends itself to the ideas of Big Data and large scale distributed processing. This work utilised a small Hadoop cluster to carry out the experimental part of the work but it is envisioned that a full scale industrial implementation of this work would utilise a far larger distributed architecture. There has been a great deal of work carried out to try and make machine learning applications more suitable for this type of environment. Yang, Liu and Fu (2010) looked at using MapReduce to create association rules of a Hadoop architecture, this work utilised the distributed nature of large numbers of processing nodes to cope with the need for large amounts of memory required to run an Apriori algorithm on very large datasets. Wang *et al.*, (2014), similarly used MapReduce and Spark to look at an implementation of the C4.5 decision tree implementation on Big Data architecture. Shallue *et al.*., (2018) studied the effects of data parallelism and its effect on batch size and training time when training Artificial Neural Networks (in this case Stochastic Gradient Descent (SGD) models), the work in this paper was carried out using GPUs and ASICs rather than the Big Data distributed Hadoop model but the reasoning is the same. An overview of regression models using Hadoop was provided by Saritha and Sajimon, (2017) who looked at linear regression models and the effect of running on Big Data architecture.

Other work has focused on the ability of applications such as Apache Spark (Apache Software Foundation, 2018) to run streaming operations across highly distributed systems. Kupisz and Unold, 2015 looked at recommender systems using the Mahout machine learning library (Apache Software Foundation, 2014), this was done to attempt to create a recommendation system for e-commerce applications. Bao *et al.*, (2012) considered the issue of massive amounts of fast moving data being streamed from large numbers of sensors in a manufacturing system. Bao *et al.*'s, (2012) goal was to solve one of the problems around cloud manufacturing where users can select different manufacturing services from different providers but there is an obvious need to track the product across multiple sites and manufacturing processes.

The Lambda architecture, which is discussed in section 3.2 of this work, was the subject of Kiran *et al*'s., (2015) paper which looked at high speed sensor analysis over an Amazon EC2 cloud setup.

2.7 Lifelong Machine Learning

A number of papers have appeared suggesting the idea of lifelong machine learning. This is the idea that a machine learning algorithm continually tries to improve its performance over the entire lifetime of the application, rather than the traditional idea that machine learning algorithms have a training phase and, on conclusion, are then fixed and implemented, Silver, Yang and Li (2013), Ruvolo and Eaton (2013) and Liu (2016) are all examples of this. This idea of continual development fits with the work presented here and is particularly important when considered in light of the ideas such as concept drift covered in section 9.2 where the targets of the system may change over time, leaving previously well performing algorithms failing to take account of changes.

2.8 Round-up

This work addresses the need for more applications of machine learning for industrial processes. To do this it uses a distributed machine learning system that compares multiple algorithms and configurations and selects the best performing using a variety of target metrics rather than just relying on one or two that can be biased in certain circumstances. The system is a lifelong learning application that continually assesses new algorithm configurations to look for better performances or changes in the target data. The system uses a framework that performs all of these operations without human intervention after the initial setup phase.

Chapter 3

System Overview

3.1 Introduction

The framework described in this work is an autonomous software system based on a Lambda architecture. The purpose of the framework is to ensure that at any given time the system being monitored/controlled is being done so by the best performing algorithm configuration available. The framework as described can perform a variety of operations but in this work the example of a regression operation on an industrial process is presented. The framework simultaneously runs a real-time stream and a second stream. The second stream evaluates algorithm configurations by running alternatives, evaluating against a wide range of accuracy metrics and using an optimisation function to select the best performing algorithm configuration. This section will describe the system design of the framework as presented including the architecture it is based on, the change made to the architecture to provide autonomous operation and the software used to achieve this.

3.2 Lambda Architecture

The Lambda architecture is described in Marz and Warren, (2015), the principle behind the architecture is that there are two analytical channels. The first channel is a real-time (or near real-time) streaming layer that is responsible for the control of the process in question. This could be a number of different types of operation in industry ranging from control to monitoring to predictive. The second channel is a batch layer that is responsible for trying to improve the performance of the first channel. This is accomplished by testing different configurations of the algorithm running in the streaming channel or entirely different algorithms that are trying to achieve the same task. The second layer has access to the full range of previous data that has been through the system and runs as a series of batch processes taking as long as is necessary to improve the algorithm results. The speed layer has to deal with data at the speeds necessary to control the process. Data is stored in the batch layer so that the algorithms being tested have access to the data necessary whilst the speed layer does not need to store any of the data going through the system. Outputs from the process (environment readings, process results, etc.) are fed back into the system as inputs. In addition to the two analytical channels there is a third serving layer. The purpose of this third layer is to allow users to view the results of either of the two analytical channels, this would be real-time results for the speed layer and batch results for the batch layer, this may involve different visual tools for each layer. Figure 3.1 shows the system.

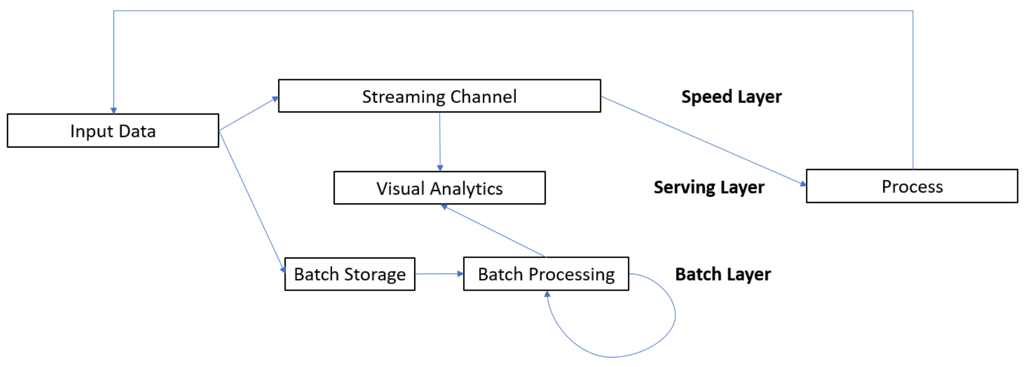

Figure 3.1: *Classical Lambda Architecture*

The classical Lambda architecture uses a Big Data approach to data analytics in the batch layer by utilising highly distributed architectures such as Apache Hadoop to run multiple different configurations and algorithms at once. The number of algorithm configurations that can be checked by the system is only limited by the expandability of the available hardware, and if on-line cloud services are used it is a matter of the monetary budget that the organisation/company involved is willing to spend.

The serving layer allows users to view the results of the analytical layers and make decisions regarding which algorithm configurations to run in the speed layer. The lambda architecture helps to improve process analytics by using current process data to build and refine models for the task at hand but it does have some drawbacks:

- The need for specialist users.
- Manual switching of algorithms.
- Timing of algorithm change over.

Although the Lambda architecture uses a serving layer to produce information regarding the current and candidate algorithm configurations this will generally require a data analyst to interpret the results As outlined earlier many companies, particularly those in the small to medium enterprise (SME) range are unlikely to have dedicated data analysts on staff to carry out out this task. One of the outcomes of this work is to have the system itself analyse its own results and to make intelligent decisions based on the results produced by the two analytical channels.

If there are no specialists available to analyse the results of the data then it is equally unlikely that there would be any dedicated software engineers to deal with changing the algorithm once a necessary change is identified. Due to this the framework is be able to swap the controlling algorithm out for a better performing algorithm configuration. This requires the framework to track more than just the accuracies of the current algorithm and the candidate algorithms, the framework will need to track other metrics to ensure that the changeover is beneficial to the system as a whole and not a very short-term change that will produce difficulties for the process.

Along with being able to decide whether a changeover would be beneficial to the system the framework must also decide when to do the changeover. In the traditional Lambda architecture model this would be determined by data analysts interpreting the information produced in the clearing channel. In the framework described by this work this decision is made autonomously and can therefore be done in a much more timely manner in terms of adapting to changing conditions.

To this end the framework uses a modified version of a Lambda architecture shown in Figure 3.2

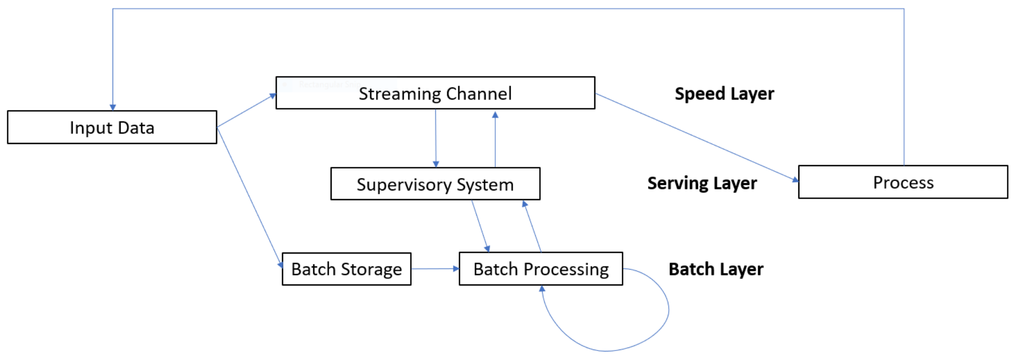

Figure 3.2: *Modified Lambda Architecture*

The changes in the architecture relate to the replacing of the visual analytics section in the serving layer with a supervisory system. The supervisory system (discussed in Chapter 9) is responsible for comparing the results of the candidate algorithm configurations in the batch layer and comparing them to the results from the current algorithm running in the speed layer. The supervisory system is then responsible for making the decision as to whether the algorithms should be switched and if so is tasked with performing this switch. Despite the name the candidate algorithms running in the batch layer are not necessarily running in batch mode, the system could, if it was felt to be beneficial, also run the candidate algorithm configurations in streaming mode. The naming of the batch layer simply indicates that either is feasible and reflects that the system does not require the batch layer candidate algorithms to be running in real-time with the process. The algorithm running in the speed layer must, by necessity, be capable of running and returning results in fast enough real-time to provide valid feedback to the process in question, it is also part of the supervisory algorithm's remit to ensure that this is the case.

3.3 Process Architecture

In the work presented the task used as an example for both the speed and batch layers is that of regression analysis of the chosen dataset. Whilst the speed layer would perform the regression algorithm as expected the mechanisms in the batch and supervisory layer require some explanation. In the batch layer a number of regression algorithms will be tested along with variations in their hyper-parameters (discussed in chapter 5). Each of these are evaluated against a number of accuracy metrics to determine the best performing algorithm configuration. Whilst this is taking place the performance of the current algorithm configuration in the speed layer is also monitored via the supervisory algorithm. The performances of the candidate algorithm configurations are compared alongside the current speed layer algorithm using an optimisation technique to determine which will be the best performing algorithm for the next predetermined amount of time. In this way the optimisation operation (discussed in chapter 7) can be run at predetermined intervals and the best algorithm sequence can be determined until the next result, E.G. If we decided that the optimisation routine should be run every ten minutes then the optimisation algorithm would be set to determine which algorithm configurations would be optimal for the next ten minutes. Once the system has determined the best sequence via the optimisation algorithm it is then left to the supervisory algorithm to handle timing and validity of switching the different algorithm configurations in and out of the speed layer.

In the batch layer it would obviously be beneficial to run as many algorithm configurations as possible. In order to be able to do this the intention is that the framework would utilise a Big Data architecture to have a number of distributed instances of the regression tests running at the same time. In an industrial system this could either be achieved via an on premise cluster, a public cloud service or a hybrid system of the two, depending upon the operators needs, expertise and budget. In the work presented here the prototype system was developed on a small, 4 node, Apache Hadoop cluster.

3.4 Software Architecture

For the prototype system developed for the work presented all code was written using Python3 and utilising the Scikit-learn machine learning library (Pedregosa *et al.*, 2011) for the regression operators. 5GCraft_Firefly (Twain OpenAI Club, 2021) and BeeColPy (Oliveria, 2020) were libraries used for the optimisation functions. The regression tests were conducted on a four node Cloudera CDH6.2.1 installation (Cloudera, 2022) which implements Apache Hadoop (Apache Software Foundation, 2010).

Chapter 4

Metric Measurement

4.1 Difficulties with Accuracy

In many CASH implementations (Thornton *et al.*, 2013, Feurer *et al.*, 2015, Golovin *et al.*, 2017) a significant effort is made to optimise the set of hyper-parameters or algorithm used to produce a classification decision. The method often described in the literature, is that of a Bayesian optimiser (discussed in section 7.2). However, although a single accuracy measure is used to determine the best performing algorithms or set of algorithm parameters, this has limitations when considering a system for use in an industrial process. Under different circumstances any given industrial process may require an algorithm to have a certain type of accuracy and that certain error criteria are more important than others. The most often used metric measure in the literature for classification methods is that of absolute accuracy, this is simply a measure of how often the algorithm correctly predicts an examples class. However a simple accuracy measure tells us nothing about the distribution of the errors or whether particular classes are more often misclassified than others. There are measures to examine these, such as confusion matrices, but in terms of CASH implementations the final decision is based on absolute accuracy.

Whilst most predictive industrial processes problems are classification problems and many more could be reduced to a classification problem there are also a large number of instances where the process is a regression problem and a specific value needs to be predicted. Regression problems are less common in the literature being more complicated than classification problems but the most common metric used in those instances is that of mean squared error (see section 4.2.3) or sometime simply the absolute error, which is simply the magnitude of difference between the expected target value and the observed value ignoring sign. The work presented here is a regression problem as presented but also the data was repartitioned to be useful for a classification problem (see section 5.9). This allows this work to build a hybrid of regression and classification metric measures for experimentation. In an actual industrial environment the process would dictate whether the system would be using regression or classification metrics.

This work uses 10 regression based accuracy measures and 6 classification based metrics, with an additional 5 metrics representing lag for cross correlation prediction.

4.2 Regression Metrics

The 10 accuracy metrics associated with regression accuracy are :

- Current Explained Variance.
- Cumulative Explained Variance.
- Current Maximum Error.
- Cumulative Maximum Error.
- Current Mean Squared Error.
- Cumulative Mean Squared Error.
- Current Median Absolute Error.
- Cumulative Median Absolute Error.
- Current R2 Score.
- Cumulative R2 Score.

4.2.1 Explained Variance

The explained variance is defined as the variance in the model due to included parameters, if this was one parameter it would directly measure the effect altering that one parameter would have on our observed values but more commonly it is the set of parameters that are included in the model. For a model (y) with observed values ($\hat{y}$), with variance Var, explained variance is defined as :

$$1 - \frac{Var(y - \hat{y})}{Var(y)} \tag{4.1}$$

4.2.2 Maximum Error

The maximum error is defined as the largest difference between an observed result and the value predicted by the model. For the same model stated in equation 4.1 the Maximum Error is defined as :

$$max(|y_i - \hat{y}_i|) \tag{4.2}$$

In this case the metric would give some indication of the largest error in the model, a model with low variance may still contain large single errors and it would depend upon the process involved as to whether this was acceptable.

4.2.3 Mean Squared Error

The mean squared error measures the magnitude but not the sign of the error, it is defined bt the following equation:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.3}$$

Where n is the number of observations.

This is a similar situation to the previously described maximum error however in this case the metric tracks errors across a range rather than the single largest error, this gives an indication of the errors as a whole rather than whether there is a single underperforming prediction.

4.2.4 Median Absolute Error

The median absolute error tracks all the errors across the range in a manner similar to the mean squared error but, similarly to the maximum error, tracks a single prediction value. In this case the median vale from the range of absolute errors. It is defined by the equation :

$$(|y_1 - \hat{y}_i|)...(|y_n - \hat{y}_n|) \tag{4.4}$$

This metric gives another view of the data when combined with the maximum and mean squared errors.

4.2.5 R2 Score

The R2 score is closely related to the explained variance, it also measures the variance in the model due to the included parameters. The R2 score is defined as:

$$1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2/n}{Var(y)} \tag{4.5}$$

The difference between explained variance and R2 score is that as part of the sum of squared residuals errors the mean of the errors is subtracted. If the mean of the errors is 0 the both the R2 score and the explained variance will give the same results. However if there is any bias in the data where the model is either over or under predicting observations consistently then the two measures will give different results. By tracking both measures we can learn some information regarding the behaviour of the model and the parameters informing it.

4.3 Current vs. Cumulative Metrics

For each of the metrics listed previously this work tracks two versions, the cumulative version, which is the algorithm using all the data to that point, and the current version, which is the data as a one minute rolling window (previous 6000 data points). The reason for examining different lengths of results is that the industrial process being modelled may be some long standing process, where the conditions do not normally change and the results are consistent, or it may have a rapidly changing environment, where conditions result in large variations. In the former case the model might need to track as many results as possible whilst looking out for slow drift in the conditions leading to slowly rising errors whilst in the latter case it would need to closely track and give emphasis to more recent results to catch errors quickly. In practise the length of time needed to track the metrics for comparison would be dependent on the process itself, the work presented here looks at a short term (1 minute) and long term (all the data as presented at the point of analysis) time scale.

4.4 Classification Metrics

Along with the regression metrics there are also 6 classification metrics. These classification metrics are dependent on the data being split into the three bands described earlier (see section 1.2):

- ROCAUC
- Total Band Accuracy
- Total Tolerance Accuracy
- Total Band 0 Accuracy
- Total Band 1 Accuracy
- Total Band 2 Accuracy

4.4.1 Receiver Operating Characteristics Area Under Curve (ROCAUC)

ROCAUC (Receiver Operating Characteristic Area Under Curve) is a metric that measures the distributions of true positives and false positives at various test values. By sampling across the range of possible values a curve can be constructed by plotting the false positive rate against the true position rate for a classification target, this is the ROC curve. The ROCAUC metric is the area under the ROC curve and this indicates the model's performance across the full classification range. The result of the ROCAUC is a value between 0.0 (a model that always predicts incorrectly) and 1.0 (a model that has a 100% accuracy). The ROCAUC measurement in this study looks at the split between within tolerance and outside tolerance.

4.4.2 Band Accuracy

As described earlier (section 1.2) the entire dataset has been split into three bands; band 0 represents values falling below the permitted range, band 1 represents values falling inside the permitted range and band 2 represents values above the permitted range. A total of five band accuracies are tracked, these are simple total accuracy metrics. The first of these is the total band accuracy, this is the standard accuracy measure of how often the model correctly predicts the band of the observed value against the number of predictions, this is defined in equation 4.6 :

$$\frac{(\hat{y}_0 + \hat{y}_1 + \hat{y}_2)}{n} \tag{4.6}$$

Where $\hat{y}_x$ is the correct number of predictions for group x (for the three bands) and n is the total number of predictions.

The second of the metrics is tolerance accuracy, in this metric the only concern is with whether or not the model can predict whether the value will be in or out of tolerance. The metric is not concerned with getting the band prediction correct. In the case of a tolerance accuracy the algorithm need only predict those in tolerance as band 1 and those out of tolerance as either band 0 or 2, therefore a value of 6ppm (band0) predicted as band 2 would be considered a success as both of these are outside tolerance range. The reason for including both of these measures is that they track different behaviours in the process, in terms of the overall accuracy the system is simply measuring the model's performance at prediction and in many cases this will be the measure that is required, but in a number of other instances the process may not care about the reason for a process failure (either below or above a predetermined measure) just whether or not something will or won't fail. Failure itself may be easier to predict than the exact result of failure, in which case the system should not penalise the model if all that is required is a prediction that a process will result in success or failure.

The final three metrics are related to the model accuracy for predicting the actual specific band for an observation; band 0, 1 & 2 accuracy. In this case the metrics are interested in the model's ability to accurately predict a particular band. This would be of obvious value if there was a particular outcome that we valued over the others, for example if having values in the over tolerance range (band 2) was particularly dangerous the framework could prioritise this measure, band 2 accuracy, over some of the other metrics.

Similarly to the regression metrics this work tracks both the cumulative metrics above as every result in the dataset to date and the value of each of these metrics per minute. Again this is done to have a snapshot of the current performance of an algorithm within the time frame of each test (in this work 1 minute) but also to keep track of how an algorithm is performing over the time that the monitoring is taking place as a whole. By monitoring both the system can make decisions about what is the currently best performing algorithm and set of hyper-parameters but also monitor for configurations that start well but begin to perform poorly as time moves on. This can be an important feature of time series data prediction known as concept drift and will be discussed in section 9.2.

4.4.3 Correlated Metrics

As discussed later (section 5.10) three datasets were created for this work from the one obtained from Fonollosa *et al.* (2015), the dataset as it was obtained (referred to as the raw dataset in this work), the smoothed dataset, that is an average of the previous 5 previous data points and the correlated dataset. In terms of the first two datasets (raw and smoothed) the metrics described above were the complete set of metrics applied, however in the case of the correlated dataset there are an additional 5 metrics tracked :

- Offset
- Total Offset
- Total Absolute Offset
- Average Total Offset
- Average Total Absolute Offset

Every minute (6000 data points) the system calculates the cross correlation value for the data, to find the best fit between the observed results and the target results to calculate the lag between the two data sets. The lag across the dataset was not expected to be uniform as observed by Zhao *et al.*, (2019). The first metric recorded is the offset of the two data sets that produces the best cross correlation, given work in the literature it was expected to see the offset value to be a positive number indicating that the predicted values are lagging behind the target values. However the cross correlation can produce a negative value which represents the predicted values being ahead of the targets, in essence the system would be predicting future values. The second metric is a sum of all the lag values in the signal to the point in time being measured E.G. in the 4^{th} minute of recordings the first metric would give the current best lag value for the 4^{th} minute and the second metric would give the sum of the lags for minutes 1,2,3 and 4. By comparing each individual minute and the total so far in the series it is possible to track both the immediate performance of the algorithm along with the historical performance. The third metric tracks the moving average through the series and the fourth metric does the same but ignores the sign of the offset and is only concerned with the magnitude. These different metrics are useful in different circumstances, a set of predictions that are sometimes very close to being at zero lag but at other times having a high value of lag will have a different pattern in the metrics to one that is consistently a small fixed lag behind the targets, these could however have similar total offsets or average offsets.

4.5 Summary.

The set of metrics being used in this work cover a range of regression and classification measures. They are not an exhaustive list nor necessarily were the metrics chosen to produce the best results when paired with the chosen dataset. The metrics were largely chosen as some of the most common metrics found in the literature. A decision was made to trial both regression and classification metrics in this work, as the framework being presented as the focus of this work is able to be put to use for either classification or regression operations. In a normal operating environment the framework would utilise either a set of regression or classification metrics rather than a combination but here both have been used to show that the framework can cope with both. The multiple metrics replace the more usual practise of a single target metric (often absolute accuracy) and allow the framework to be flexible around different behavioural patterns we may want from the process being modelled. A non-specialist user can adjust the targets by identifying the characteristics of the process, as we know what each of the metrics measure the non-technical user does not need to have a knowledge of the metrics themselves. The importance of each metric is controlled via a weighted vector, the framework uses this during the optimisation phase to determine, via the use of the metrics, which algorithm is performing the best (see chapter 7). Now that the metrics have been defined for the framework these will be used in the next phase of prediction and classification.

Chapter 5

Prediction Algorithms Techniques and Implementation.

5.1 Introduction.

The framework described in this work is to be able to continually evaluate and select machine learning algorithms based upon a set of metrics described in the previous chapter. The evaluation and selection process will be discussed later in chapter 7 and in this chapter the report will discuss the machine learning algorithms being selected. As with the metrics discussed in the previous chapter the algorithms described here are not meant to be an exhaustive list of algorithms. Instead they are a representative selection of potential machine learning algorithms that are likely to be used in industrial processes such as the one being used as an example for this work. The algorithms themselves will be discussed in this chapter along with a discussion on why these particular algorithms were selected and in the next chapter we shall look at the results of the implementations of these algorithms. Again these algorithms are examples of potential solutions and in an industrial setting these could be swapped for completely different algorithms depending on what was deemed most potentially suitable for the process being modelled in question. The goal of the framework being discussed is that it should be agnostic in terms of the dataset being analysed, the machine learning algorithms being used for prediction, classification or control and the optimiser being used for selection.

The five machine learning algorithms selected for testing in this work are as follows :

- Least Absolute Shrinkage and Selection Operator (Lasso)
- Ridge Regression
- Stochastic Gradient Descent (SGD)
- Elastic Net Regression
- Random Forest Regression

The main reason for the selection of these algorithms was that they are very commonly used in the literature when comparing types of regression algorithms, see Sharif *et al.*, 2017 as an example. Linear regression is often used as a predictive model and there are a number of variations, here this work examines four types. The work also includes a non-linear method in the form of random forest regression, which again is common in the literature (Grömping, 2009).

The standard least squares linear model estimates the coefficients of the parameters ($\beta_{0..n}$) using the equation 5.1, many of the regression algorithms used in this work modify this via a tuning parameter.

$$\beta_{i..n} = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_i)^2 \tag{5.1}$$

Each $x_i = (x_{i1}, x_{i2}..., x_{ip})$ is a vector of feature measurements for the ith case

5.2 Least Absolute Shrinkage and Selection Operator (Lasso)

Lasso regression was selected as one of four linear regression algorithms being tested in this work. The Lasso algorithm is described in (Hastie, Tibshirani and Friedman, 2009) as a "shrinkage method", it works by minimizing the function given in equation 5.2 for β values

$$\beta_{Lasso} = \frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j| \tag{5.2}$$

The tuning parameter used by the Lasso algorithm uses the L1 Norm penalty, this uses the magnitude of the coefficients. Due to the shape of the L1 Norm, a diamond (in 2-D, see Figure 5.1) this can lead to coefficients being reduced to 0 (due to the edge of solution spaces hitting the corners of the diamond) and therefore eliminated from the model giving a sparse model. λ is defined to be a complexity parameter with a value greater than 0, the larger the value of the complexity parameter the larger the shrinkage of the coefficients will be.

5.3 Ridge Regression

Ridge regression is another "shrinkage method" (Hastie, Tibshirani and Friedman, 2009) and is related to the Lasso regression algorithm. Again this algorithm works by minimising the β values for equation 5.3.

$$\beta_{Ridge} = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{5.3}$$

As with the Lasso algorithm the λ parameter is a complexity parameter and the bigger the value the smaller the β coefficients will shrink. Unlike the Lasso algorithm the Ridge algorithm uses the square of the coefficient values rather than the absolute value, this is known as the L2 Norm. The shape of the L2 is circular in 2-D (Figure 5.1) which means the solution space for the minimised solution equation will not touch the axis when it meets the L2 contour therefore the β coefficients may become very small but will not be reduced to 0.

5.4 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimiser for training models rather than a machine learning algorithm. Pedregosa *et al.*, 2011 in their documentation describe the function of SGD as "minimising the training error". Equation 5.4 shows for a set of training examples $x_{1..n}$ with targets $y_{1..n}$ to achieve a linear function $f(x) = w^T x + b$, with model parameters w and incept b.

$$E(w, b) = \frac{1}{n}\sum_{i=1}^{n}(L(y_i, f(x_i)) + \alpha R(w) \tag{5.4}$$

Here L is a loss function and R is a regularization term that is implemented to penalise complexity, α is a parameter used to weight the regularisation parameter. There are many choices available when choosing a loss function for L and this will be discussed later in section 5.8, where the report considers implementation of the algorithms. SGD as a method is more efficient than others mentioned here but unlike the previous methods (Lasso and Ridge) has more hyper-parameters that may require tuning. In the previous methods both Lasso and Ridge only had the λ expression that could vary whereas for SGD there are the terms for α and R as well as the choice of loss function.

5.5 Elastic Net

Elastic Net is another shrinkage method, this time it combines both the L1 and L2 Norms as shown in Equation 5.5.

$$\beta_{Lasso} = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_i)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \tag{5.5}$$

The corresponding contour is a combination of L1 and L2 as shown in Figure 5.1, the Elastic Net is an approach to try and avoid some of the issues found in both Lasso and Ridge regression approaches. λ_1 and λ_2 are control parameters to reflect how much influence the L1 and L2 Norms have respectively.

5.6 Comparisons and limitations

All four previously mentioned algorithms and methods are linear regression models, they attempt to fit a hyperplane through the dataset. Each of the methods has a number of advantages and disadvantages that are widely discussed in the literature, see, for example (James *et al.*., 2017, Hastie, Tibshirani and Friedman, 2009, Bishop, 2006). Ridge regression solves some of the issues with ordinary least squares estimation (multicollinearity, where some of the coefficients are correlated and are not independent which can cause high variance), but because of the shape of the contour associated with the L2 Norm the Ridge approach does not reduce the complexity of

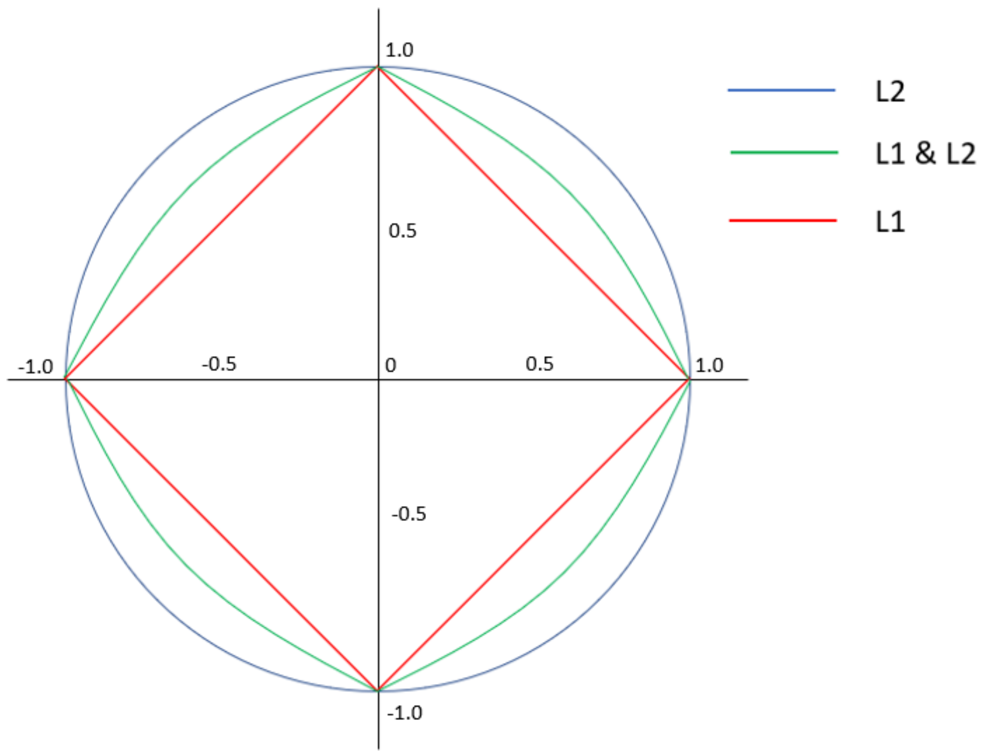

Figure 5.1: *Contours for Linear Regularisation terms as a 2-Dimensional example*

the model and maintains all of the associated coefficients. Lasso attempts to address some of the problems of Ridge by allowing the coefficients to reduce to 0, creating a sparse version of the solution. Lasso can have issues if coefficients are correlated leading to the method concentrating on one of the correlated variables at the expense of others. Elastic Net attempts to compensate for this by using the combination of Ridge and Lasso's penalties with the disadvantage of an increase in the computational cost. SGD, as noted earlier, is an efficient algorithm at the cost of needing to tune more hyper-parameters than the other three presented linear regression methods.

One disadvantage shared by all the methods so far described is that they are all linear regressors, as such they are limited to attempting to fit a linear hyperplane through the dataset to predict values. If the pattern (if a pattern exists) that describes the relationship between the coefficients is non-linear then none of the methods will be able to produce satisfactory results. One approach that can model non-linear patterns in a regression environment is that of the fifth approach detailed in this work, random forests.

5.7 Random Forests

Random Forests as described in James *et al.*, 2017 are an ensemble machine learning method that use a number of individual decision trees and produce a result via collective decision making. If they are being used to perform classification then a vote is taken across the trees and the class that is selected most often has the highest number of votes and is selected as the result. In the case in this work where the goal is a regression prediction the output from the numerous trees is taken and the mean from the predictions is taken as the result.

When building the decision trees for the forest the random forest algorithm will select a random sample of possible predictors and choose from those candidates as the decision split. This allows a greater number of candidates to be considered across the forest and prevents a strong predictor from dominating (as any individual predictor will not be in most of the split considerations). Each individual tree is a weak regressor / classifier compared with a traditionally built decision tree such as a C4.5 / J48 tree but the strength of the approach is in the collective decisions of the multiple trees. The major strength of the random forest approach compared with the previously detailed approaches is that it is not dependent on the prediction pattern being linear and can therefore approximate more potential patterns than the previously mentioned approaches. Random forests do have several drawbacks in that they are computational expensive, can take longer to make a prediction (many trees have to be considered and the outputs averaged rather than just a single function as in the previous methods) and can have a tendency to over fit training data if training isn't carried out carefully. The other drawback, that random forests traditionally have in some applications, is that they are harder to interpret how the process arrived at the target value. Due to this they are often considered black box solutions, in the environment described in this work this is not necessarily a drawback that would hinder their usage. Although many applications find the ability to dive deeper into the workings of the decision engine essential the type of system described in this work is meant to be used specifically by a non-specialist in machine learning. Given that the user is not of a nature to example the algorithms finding in depth and is instead only interested in the results, whilst the tuning and measurements are left to the autonomy of the system it is not a drawback for the inner workings of the prediction engine to be inaccessible or difficult to access for a normal user.

5.8 Implementation

For this work all the above algorithms have been implemented for testing using Python and the Scikit-Learn machine learning library (Pedregosa *et al.*, 2011). Each implementation has a number of options and hyper-parameters that are detailed below along with the values used in this work.

Lasso - The Scikit-learn implementation of Lasso has a number of options including normalisation of the values before regression, maximum number of iterations and some parameters for pre-computation to speed up calculations. The main parameter as far as this work is concerned however is that of the λ parameter from equation 5.2, Scikit-learn refers to this as the α parameter and the default value is 1.0, this is a common starting point for implementations of the Lasso algorithm. It is noted that if α is set to 0 then the algorithm is the same as performing ordinary least squares. For the purposes of the work undertaken the parameter was left at the default value of 1.0.

Ridge - Again the Scikit-learn implementation has a number of parameters, the most important for this work are the λ parameter from equation 5.3 (again referred to in the Scikit-learn documentation as α) and the type of solver used in the computation of the algorithm. Scikit-learn has six different settings for the solver:

- auto
- svd

- cholesky
- sparse
- lsqr
- sag

As the name suggests 'auto' attempts to select the best of the other solvers based on the data it is presented with, 'svd' uses Singular Value Decomposition. 'Cholesky' uses a standard function within Scikit-learn's library to obtain a closed form solution. 'Sparse' uses conjugate gradient solver also native to the Scikit-learn library. 'lsqr' uses dedicated least squares and 'sag' is a stochastic average gradient descent method. More details on these options can be found in the Scikit-Learn documentation (Pedregosa *et al.*, 2011). For the purposes of this work an example of a Ridge regression solver was required and the default value of 1.0 for λ was selected once more and Cholesky as the solution solver as this is the default solver in the library for most general conditions (the auto feature will generally select Cholesky except under specific circumstances).

SGD - A mentioned previously the SGD method has more hyper-parameters than any of the other linear regression models used here and here the values need to be set for the weight, α, the regularisation term, R and the loss function , L as shown in equation 5.3. The α parameter was left at its default setting (as were many parameters in these tests) of 0.0001. R term has the settings of 'none', 'L2', 'L1' or 'elasticnet', for the purposes of this work 'L2' was selected (again this is the default setting in the library). With the regularisation term as an L2 norm we can directly compare the result of SGD with that of our other L2 Norm regressor, Ridge. The loss function, L, has the settings of:

- squared loss
- huber
- epsilon_insensitive
- squared_epsilon_insensitive

For these loss functions, 'squared loss' is the ordinary least squares fit (and is the default value), 'huber' is based on ordinary least squares but tries to deal with outliers by switching to a linear loss past a set distance ϵ, 'epsilon_insensitive' ignores any errors less than ϵ and is a linear loss function past that distance. Lastly 'squared_epsilon_insensitive' again ignores any errors less than ϵ but is a squared loss function past this value. If anything other than squared loss is used then a value for ϵ also needs to be set, the default value for this parameter is 0.1. In this work the loss function was again set to the default value of 'squared loss'.

Elastic Net - Elastic Net shares many parameters with the related Lasso and Ridge algorithms and the most important settings for this implementation are the settings for λ_1 and λ_2, in equation 5.5, that control the amount of influence that the L1 and L2 Norm component have. Scikit-learn sets these using a parameter that sets the value for both parameters and then adjusting the influence of the L1 Norm parameter in relation to the L2 Norm parameter. The default setting, and the one used for this work, is to set the parameters both equal to 1.0 and have the L1 and L2 Norms have equal influence.

Random Forest - The random forest regressor being a more complex regression algorithm has more parameters to set. The following parameters are ones that are key to the purpose of this work:

- n_estimators
- criterion
- max_depth
- min_samples_split
- min_samples_leaf
- min_weight_fraction_leaf
- max_features
- max_leaf_nodes
- min_impurity_decrease
- min_impurity_split

As with many of the other algorithms in this work many of the settings have been set to the default library settings, this report will briefly discuss what effect each of these settings have what they relate to. 'n_estimators' is the number of trees created to produce the forest, the default value for this is 100. 'criterion' selects how the algorithm chooses the split at each node by setting the measurement criteria, the options here are mean squared error and mean absolute error, the default being mean squared error. 'max_depth' is the maximum depth (I.E. number of splits) each tree can expand to, the default setting for this parameter is max, this allows the tree to keep expanding until the leaf nodes are either pure or contain less than the minimum number of samples per split. 'min_samples_split' this is used for the previous setting max_depth, default setting is 2. 'min_samples_split', this is the minimum number of values any node can contain after a split, if a split would lead to the creation of a node with less than this number it is not performed, the default is 1. 'min_weight_fraction_leaf', this is an optional parameter that, if samples have been assigned weights, tracks the minimum amount of weighted samples that need to be in a leaf node after a split, in the work presented here samples are not weighted and this matches the default value of this parameter which is set to 0. 'max_features', this parameter checks the maximum number of features (inputs) considered for each split from the total number of features, there are a number of options for this setting, it can be a fixed integer value, it can be a fixed float value (which is then used as a proportion of the total number of features, E.G. 0.5 would be half of the total number of features), auto sets the value to the same as the number of features, sqrt is the square root of the number of features, log2 is the log2 of the number of features and finally none which, again, sets the number of features equal the total number of features. The algorithm will only restrict the number of features being considered once all other criteria for producing a valid split have been satisfied, I.E. if the parameter is set to 4 then the 4 features being considered will all have been checked to produce valid splits by all other criteria. The default for this parameter is auto. 'max_leaf_nodes' is a parameter that allows the setting of the maximum number of generated leaf nodes, these are created in 'best first' order. If the default, none, is selected (as it was here) then no restriction is place on the number of nodes. 'min_impurity_decrease', this is the minimum decrease in impurity (or gain in purity) a split must obtain in order to be considered, default value is 0. The impurity of a node is the ratio of classes within that node so a node containing just one class will have minimum impurity (maximum purity) and a node containing equal numbers of all classes will have maximum impurity (minimum purity). 'min_impurity_split' is a parameter for early stopping of tree generation where a node will be considered a leaf (final end node) if its impurity is below a given value (default is 1e-7).

5.9 Training Strategies and Replication

For each algorithm discussed above there were also a number of variations created regarding the training data. Algorithm testing was carried out using several sizes of training set. Randomised entries were removed from the full dataset and put aside into a training set. Each algorithm was then trained using the training set and the remaining data was presented as a time series to the algorithm. The goal of this was to preserve the order of data from the original dataset when it came to testing, there were gaps between test samples where training data had been removed, but the order of observations was preserved. The sizes of training sets tested were 1000, 5000, 10000, 50000, 100000, 500000, 1000000, and 2000000. This represents a range from just under 0.025% to just over 50% of the dataset being used as training examples. Each algorithm was run 3 times for each training level against each of the three variations of the dataset (raw, smoothed and correlated, see section 5.10).

5.10 Dataset

To conduct this study the experimental phase required a large, temporal dataset that had features that were relevant or similar to those that might be found in a manufacturing dataset. The dataset selected was from (Fonollosa *et al.*, 2015) and is available from the UCI machine learning repository. The dataset is a mixture of gas concentrations (ppm), the set used was Ethylene & Carbon Monoxide (CO) and the dataset consists of Ethylene and CO concentration levels along with readings of 16 gas sensors. The readings were taken over a unbroken 12 hour period at a frequency of 100Hz with the gas mixture concentration being changed randomly every 80-120 seconds. The dataset consists of approximately 4.2 million samples and the defined goal for the experiment was to predict the Ethylene concentration level using the CO level and the 16 gas sensor readings as inputs.

Randomised entries were removed from the full data set and put aside into a training set. The algorithm was then trained on the training set and the remaining data was presented to the algorithm as a time series set for purposes of testing. The goal here was to preserve the order of data from the original set when it came to testing, there are gaps between test data samples where training data has been removed but the order of observations has been preserved. Sizes of training sets tested were 1000, 5000, 10000, 50000, 100000, 500000, 1000000, and 2000000. This represents a range from just under 0.025% to just under 50% of the data set being used as training examples.

Previous work (Zhao *et al.*, 2019) noted that the sensors are prone to lag in detection, drift in characteristics and to occasionally produce false readings. To account for these effects three versions of the dataset were produced; the original "raw" dataset, a smoothed data set and a correlated dataset. The original dataset was presented to the algorithms as it stood with no pre-processing, this represents the data coming live from monitored machinery. The second data set was a smoothed version, in this case the value presented to the algorithms was the average of the previous five data points. This allows the framework to attempt to reduce single outlying values without falling too far behind in terms of batching readings.

In the case of the detectors lagging this will obviously produce a delayed effect in the predictions produced by the algorithms. In order to look at this a third data set was produced, in this case the "raw" data set was pre-processed by calculating the cross correlation over each minute. This set is referred throughout this work as the correlated dataset. The lag across the data set is not expected to be uniform as observed by (Zhao *et al.*, 2019). The sensors in the original experiment struggled to cope, due to this a new lag value is calculated for each minute of the experiment. Each minute's worth of data (6000 points) is compared with the predictions and offset according to the best calculated cross correlation fit. This method does introduce significant processing overheads and requires the full minutes' worth of data to be collected. In this manner it would appear difficult to envisage this method being used in a production system. It would be possible to track previous minutes and adjust for lag changes if sensor error drift was found to be gradual but in cases where change is sudden and occur for a short period of time before self-correcting (as is often the case with this dataset) this would be of limited use. In the experiments for this work this was calculated as a comparison with the other sets to see if there was a significant amount of lag happening in the regression predictions.

5.11 Summary

In terms of the work presented here it would have been a simple task to generate large numbers of variations for each of the regressors, particularly for the SGD and random forest that have larger numbers of parameters. This would have been the approach if the goal had been to achieve the best results for the dataset involved, however as previously mentioned this is not the goal of this work. The exercise here was to produce a number of competing algorithms that would produce differing levels of differing accuracies (see section 4.1) across the range of the dataset in order to study how the system might select a best performing predictor and how to alter this autonomously during the framework's runtime. For that reason having selected a number of different prediction algorithms it was not felt necessary to expand this further by using different versions with altered hyper-parameters. In a production system it would be desirable to maximise the number of both algorithms and sets of hyper-parameters that the system architecture could cope with in order to produce the best result possible for the industrial system. Such a scenario is discussed in further depth in section 10.1 when the report looks at scalability of the system.

Chapter 6

Prediction Algorithms Results.

6.1 Introduction.

The goal of the prediction phase in this work was not to necessarily produce the best quality predictions based on the dataset but to produce several sets of results that could be analysed in the next phase, the optimisation phase, to select a best performing algorithm and hyper-parameter selection. In this section this report will look at some of the results from the experimental prediction stage to show that the resultant prediction dataset is varied enough for use with the optimisation phase. The results seen here are the results of the many variations detailed in chapter 5 and this report will discuss the nature of the results produced along with some characteristics of interest from the dataset and how these might affect the framework and its treatment of datasets and prediction results in general.

6.2 Prediction Results

Figures 6.1, 6.2 and 6.3 show an unsophisticated unweighted sum of rankings for each of the five algorithms. For each algorithm the same 500,000 points were used for training and predictions made for the remaining points. For each prediction, the result from each algorithm was compared with the known result and the algorithms were ranked according to the various metrics outlined in chapter 4.

The method for ranking compared each algorithm at each minute for each accuracy metric. For example the five algorithms in this work (Lasso, Ridge, SGD, Elastic and Random Forest) are compared to each other at minute one for the explained variance measure, the best performing algorithm is awarded 1 ranking point the next 2 and so forth. If two or more algorithms have the same ranking then they are assigned the same ranking points. Once this has been done, this is then repeated for the next accuracy measure (E.G. Maximum Error) and this is repeated until all 11 accuracy measures have been ranked. The ranks for all the accuracy measures at minute one are then summed to produce an overall rank score for each algorithm after the first minute. With 11 accuracy measures being summed the best possible score an algorithm could have at each minute would be 11 (11*1) meaning that it had performed the best across all metrics, similarly the worst score an algorithm could have would be 55 (11*5) which it would achieve if it was the worst ranked algorithm across all metrics. In reality, as this report shows, no algorithm scored perfectly across all metrics not was any algorithm the worst across all metrics at any time during the tests (see Figure 6.4). This was then repeated for the next minute and so forth.

The ranking gives an overview of the performance of each algorithm through the time of the experiment compared against the other algorithms, this work is concerned in the relative performance of each algorithm rather than an absolute performance rating. In this work it is assumed that all the metric measures are of equal importance. In reality this would be unlikely in an industrial setting, in which case the unweighted sum of rankings shown here could be replaced with a weighted ranking favouring the metrics that produce the desired behaviour in the process. Using a weighted ranking to influence the selection of an algorithm based on a particular priority in terms of accuracy metric is discussed in more detail in chapter 7.

The report will discuss these performance rankings next. Four figures were produced (figures 6.1 to 6.4) to look at different aspects of the results. The rankings can be considered in two ways; either as a single snapshot of the algorithms at a particular minute in time, or the cumulative ranking score of an algorithm throughout the experiment up to that point. These cumulative and non-cumulative rankings can show different patterns in the behaviour of the algorithms, figures 6.1, 6.2 and 6.3 consider the algorithms as they progress using the cumulative ranks whilst figure 6.4 is a snapshot of each algorithm at a particular point in time.

Figure 6.1 displays the first 100 minutes of the experiment, in this figure it can be seen that there is very little separation in the 5 algorithms (the lower the rank the better performing the algorithm is as discussed previously), the only noticeable trend is that the Random Forest algorithm seems to have the most variation. The random forest implementation begins as the worst performing algorithm but by the 55 minute mark has tied with the Lasso implementation as the best performing of the algorithms, before reverting back to being one of the worst performing algorithms by the 100 minute mark.

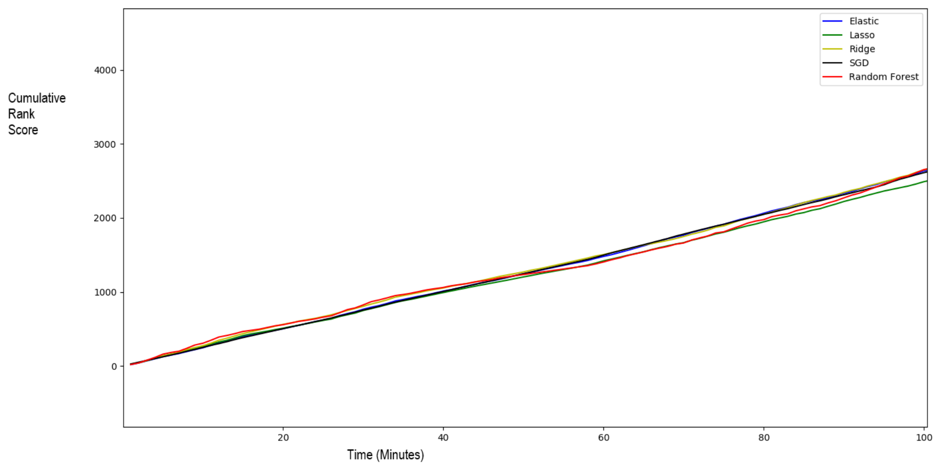

Figure 6.1: *Cumulative Rank Score of 5 Algorithms for first 100 minutes*

Figure 6.2 shows the algorithms after 140 minutes until 240 minutes and we can see that there is beginning to be some separation in the algorithm's performances with the algorithms swapping over less.

Figure 6.3 shows that by the time the full experiment has run there are obvious differences between the algorithms' performances, however although

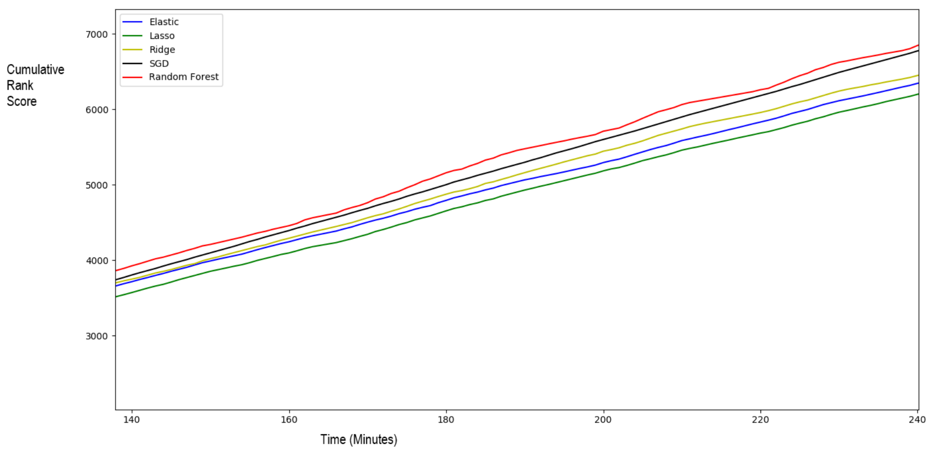

Figure 6.2: *Cumulative Rank Score of 5 Algorithms for minutes 140-240*

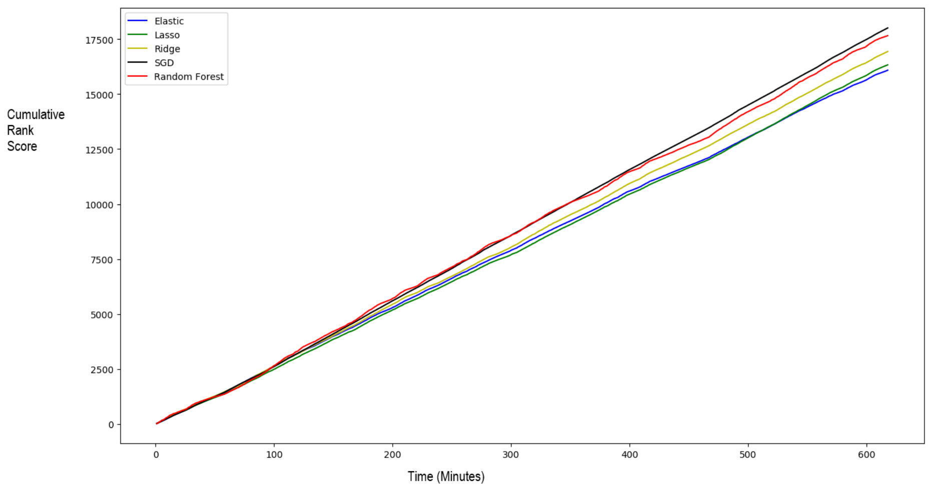

Figure 6.3: *Cumulative Rank Score of 5 Algorithms for minutes 140-240*

there is some clear separation between groups of algorithms (SGD / Random Forest vs. Elastic / Lasso) there is clear overlap between individual algorithms (Elastic and Lasso overlap across the timeline) where for a time one algorithm outperforms the other but the situation is reversed later on.

The three Figures 6.1, 6.2 and 6.3 all show cumulative error ranks but we can look at the data from minute to minute without taking into consideration any previous results.

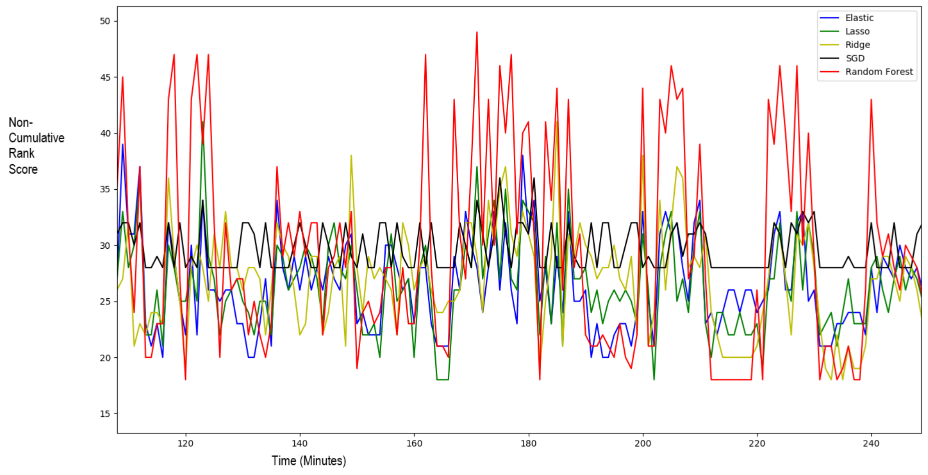

Figure 6.4: *Non-cumulative rank score for all 5 algorithms between 100 and 260 minutes*

Figure 6.4 shows the ranks of the five algorithms during minutes 100 to 260, in this case this is the non-cumulative version which demonstrates the amount of change between the algorithms for any minute to minute change, in particular we can see that Random Forests in this instance often have both very high and very low ranking scores meaning that within a few minutes they can be considered the most accurate and least accurate algorithm for predictions. This demonstrates the problem the framework has trying to decide which algorithm is actually producing the best results, in many instances (such as the example here) there is not going to be an obvious candidate that does not resort to changing the algorithm every minute or so, this is where the framework looks back at the cumulative results but this in itself will have issues. Running with the cumulative results from the beginning of the process can hide a declining pattern in an algorithm whilst it can take an initially poorly performing algorithm some time to catch up, in this case it may be appropriate to use some form of windowed timescale that allows previous results to become less prevalent as time goes on. Again what size of window would depend on the process being modelled with processes that have output that react rapidly to changes of environment where significant changes can happen quickly benefiting from shorter time frame windows. The length and shape of these windows will then become another parameter that would need to be determined by experimentation.

The variation in relative performances between two of the algorithms more clearly in Figure 6.5. Figure 6.5 ranks the value (lower is better performing) of the Random Forest implementation versus the Lasso implementation over the first 100 minutes of the experiment. It is interesting to note that in Figure 6.1 it can be seen that during the first 100 minutes of the experiment using the cumulative errors the Lasso implementation out performed Random Forest implementation for most of the duration except for a time period in the middle (between, approximately 50 and 75 minutes). However looking at this figure of non-cumulative results it can be seen that the pattern is not so simple and that there is a constant swapping over of the two algorithms throughout the runtime. The peaks and lows of Random Forest are hidden in our initial examination when we consider the algorithm just in terms of ongoing and historical performance rather minute to minute evaluations.

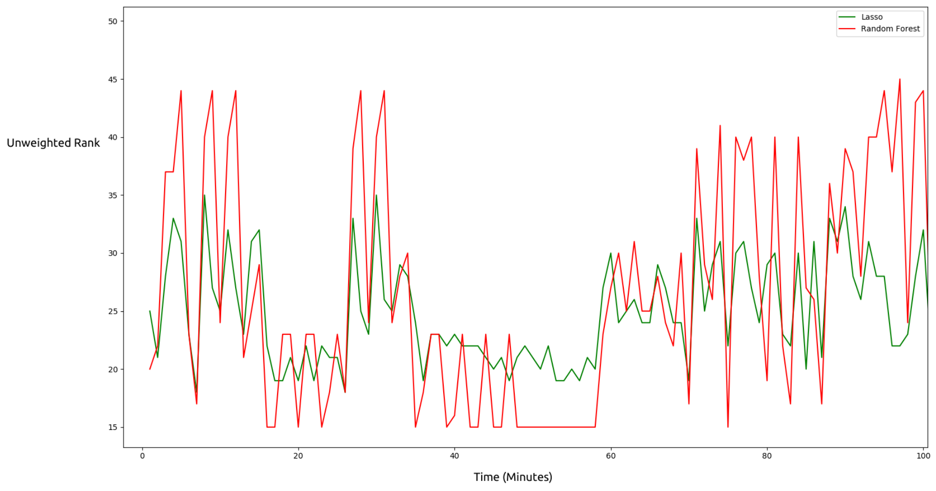

Figure 6.5: *Unweighted Ranking of Random Forest Algorithm vs. Lasso Algorithm over First 100 minutes*

Figure 6.6, shows the best lag time calculated up to 60000 seconds, this was done by the absolute error as a quick measure of how well predications where aligned against outcomes. The chart shows that the data has a large degree of variance with large amounts of lag across some minutes and very little lag in others, this is in keeping with the results found by previous work (Dominguez-Pumar *et al.* 2016, and Ai *et al.*, 2018) found that the gas dataset is an extremely messy dataset large amounts of drift and lag between sensor readings, due to the sensors being slow to recognise changes in gas concentration both for initial detection of ethylene and the resetting of the sensor readings once the gas has cleared. The unreliability in the sensors mean the dataset, in this case, can not reliably be said to have a set time lag with regard to predictions and in reality many real world datasets will have substantial flaws along these lines. However in certain circumstances it could still be beneficial to use a correlated dataset, to examine this type of relationship between predictions and outcomes where one can reliably lead the other.

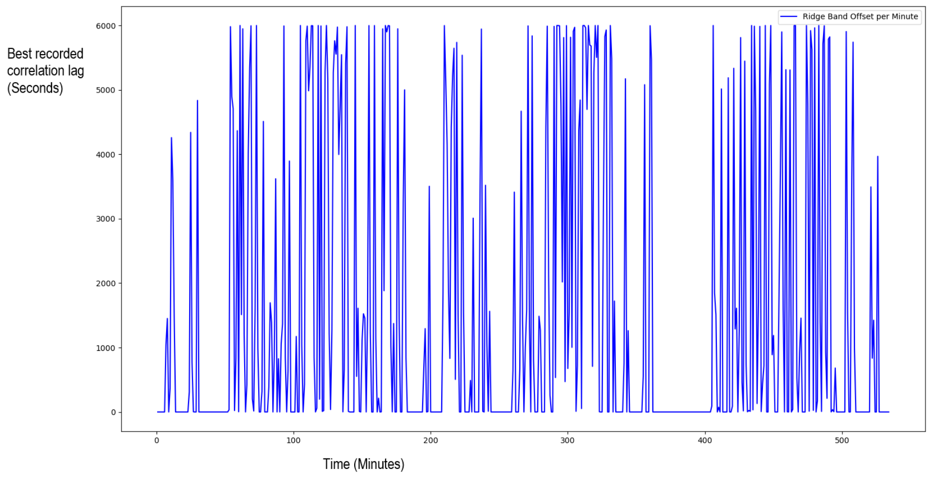

Figure 6.6: *Best fit lag on correlated signal for Ridge Regression (1000000 training points)*

Figure 6.7 shows the effect of using the smoothing operation on the dataset. As previously described the smoothed dataset is an average of the previous 5 readings rather than a single reading. In this case the predictions made by using the Elastic Net algorithm after training the model on 1000000 data points is being examined. Figure 6.6 shows the maximum error in parts per million (ppm) between the predictions and the results. Because the maximum error calculation looks at the maximum error for the last minute we see a very similar progression for both sets of data except that in this case the smoothed dataset has a higher error throughout the experiment. At first

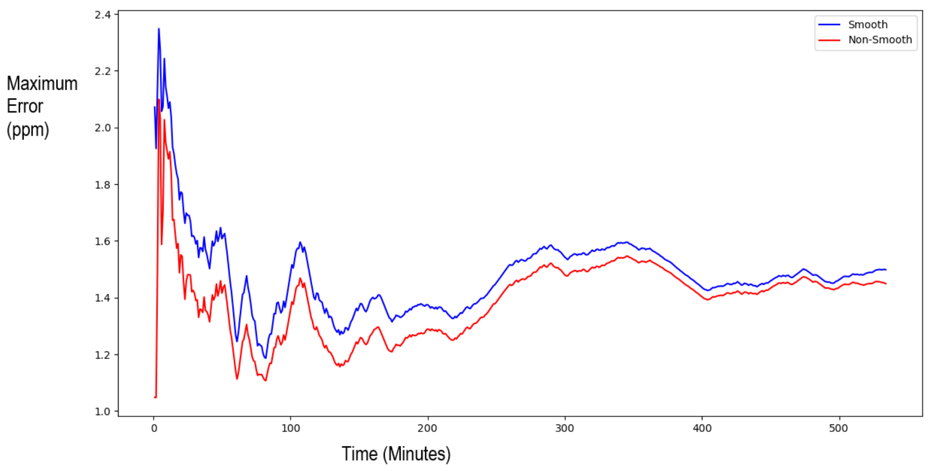

Figure 6.7: *Smoothed vs. non smoothed data for Elastic Net (1000000 training points) based on maximum error per minute*

this may appear counter intuitive as the point of producing the smoothed dataset was to reduce the influence of spikes from outlying results but in some of the metrics, such as this, we can see that smoothing the data just prolongs an outlying results influence on the outcomes. An anomalous, single, large outlying result is only used once in the non-smoothed dataset where in the smoothed dataset it is included for five predictions. The effect of smoothing the data can help for some of the metrics, such as explained variance or median absolute error (see section 4.2.1 & 4.2.4 respectively), but it must be highlighted that every change made to the dataset can have a different influence depending on which metric is being examined, this further shows the need to not rely on a single accuracy metric for evaluations.

Figure 6.8 details an investigation into the differing training levels used in the experimentation phase. In this chart one of the algorithms (Elas-

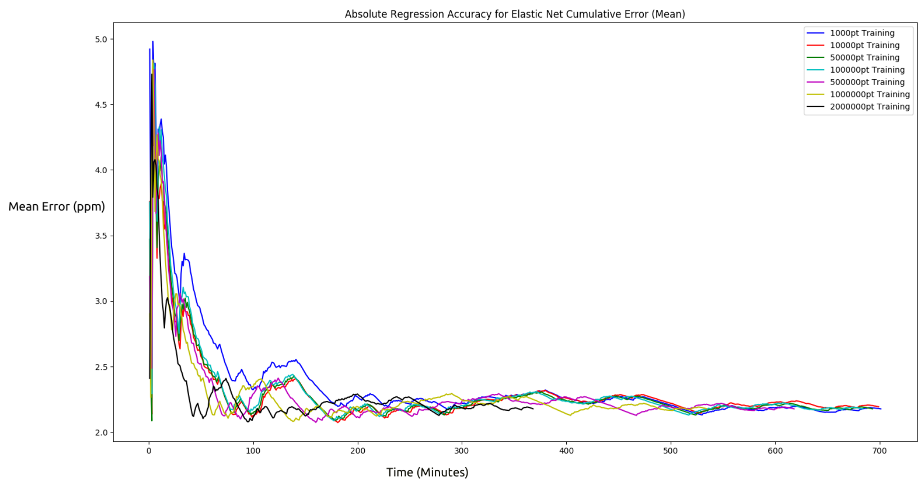

Figure 6.8: *Absolute Regression Accuracy for Elastic Net Cumulative Error (mean) for 7 Training Point Values*

tic Net) has been selected and the error between the predicted value in ppm for the gas concentration and the actual gas concentration for that time has been measured. The error is the arithmetic mean of three training runs. The figure shows that the size of the training set has a marked effect on the cumulative error at the beginning of the experiment, as might be expected the experiments with larger training sets perform much better at the start of the experiment. However the figure shows that as the experiment progresses the training size has less of an influence on the cumulative error, all the experiments converge to a similar sort of mean error value, with the higher number of training points experiments just doing it quicker. Note that the experiment lasts longer for the smaller training set data as obviously there are more data points left in the test set to run against.

6.3 Conclusion

The results produced from the prediction experiments show a wide variety of features, there are numerous instances of algorithms switching places in the ranking order, this behaviour is what was required in order to test out the features of the next phase. On the behaviour of the algorithms tested it is interesting to see that the Random Forest algorithm in particular varies quite substantially in the raking order from minute to minute. It must be remembered that, in these tests, many of the hyper-parameters for the algorithms have been left at their default settings for Scikit-Learn. In an industrial setting it is, of course, these hyper-parameters that we would wish to change and tune as part of the process of generating predictions. Given capacity there would be an opportunity to test out many variations of these parameters to try and achieve the best from the predictive algorithms in question. This report must be careful in this investigation not to try and draw too many conclusions and strategies from the predictions, these are unique to the algorithms and the particular dataset being examined and the treatment of it. For example in this study the correlated dataset did not appear to significantly improve development of the models, despite there being evidence in previous studies of lag being a feature in the dataset (Dominguez-Pumar *et al.* 2016, and Ai *et al.*, 2018). For other processes and datasets lag could have been a key factor and the purpose of the framework is to support a process that is flexible enough to cope will any type of dataset, metric targets or set of prediction algorithms.

The results generated in this section will now be used as inputs for the optimiser described in chapters 7 and 8.

Chapter 7

Optimisation Techniques and Implementation.

7.1 Introduction.

Once the ranking of the metric measures for the prediction algorithms has been completed the system is potentially left with a very large matrix of results. Finding the best performing algorithm at a given instance is a fairly trivial matter and the system can do this with a simple weighted vector sum (discussed in section 7.8). However once we require finding the best performing sequence of algorithms over a set period of time this becomes a much more difficult challenge and one that potentially has too many potential combinations to check every possible configuration. It is for this reason that the next stage of the framework is to use an optimisation algorithm to find a near optimal sequence. Optimisation strategies are used when the problem involved is too large to check all or most of the possible solutions. The goal of an optimisation algorithm is to intelligently search through the potential solutions (feature space) in a manner that finds near optimal solutions.

7.2 Bayesian Optimisers

Many of the CASH implementations discussed earlier in section 2.5 (Thornton *et al.* 2013, Feurer *et al.* 2015, Golovin *et al.* 2017) use Bayesian optimisers to find the value for the hyper-parameters of algorithms. Bayesian optimisers (Frazier 2018) evaluate the function of hyper-parameters without calculating derivatives of the function, which separates them from methods such a gradient descent learners. Bayesian optimisers use a surrogate model of the function being optimised by sampling values in the original function. Points on the surrogate model are then evaluated and points that perform well are sampled around to find better points. The surrogate model is then updated to reflect the new samples. The choice of points to evaluate are determined using an acquisition function, there are a number of options for the choice of acquisition function including knowledge gradient and entropy search but the most common is expected improvement (see Frazier, 2018 for details).

Although currently commonly used Bayesian optimisers do have some drawbacks, it is generally accepted that they scale badly with the number of dimensions in the function to be optimised, with 20 being the most quoted maximum, they also make assumptions that the membership of parameters to the potential solutions is easy to evaluate and that noise is not present in the evaluations. When these assumptions are not met the problem is termed an exotic Bayesian optimisation problem and solving these is an ongoing are of research.

In this work the framework does not use Bayesian optimisation as a solution to the optimisation issue, instead we have opted to examine a pair of bio-inspired optimisers as a general optimisation solver. Such optimisers make no assumptions regarding the solution space and place no limits on dimensions of the function being optimised (other than the the obvious assumption that the greater number of dimensions the longer the optimiser may potentially take to find a solution). Generally most bio-inspired (algorithms that look at natural phenomena for inspiration) optimisation algorithms operate through the same basic set of functions:

- Generate an initial set of solutions (sometimes referred to as an initial population).
- Evaluate the solutions using an evaluation or fitness function.
- Rank the solutions based on the results of the evaluation function.
- Keep some portion of the better performing solutions.
- Use the remaining solutions to generate a new set of solutions.
- Repeat until some stopping conditions are met.

Depending on the type of optimiser we will often see particular terms that perform the same function given different names E.G. evaluation functions and fitness functions achieve the same purpose of rating potential solutions, initial starting solutions can be termed as populations, locations, positions etc. depending on the algorithm.

The variation in optimisation algorithms is down to how these particular steps are accomplished. There are a large number of different approaches to optimisation but for this work the author has chosen to examine two in particular; Artificial Bee Colony algorithm (ABC) and the Firefly algorithm (FA).

7.3 Artificial Bee Colony

The Artificial Bee Colony (ABC) Algorithm is described in Pham *et al.*, 2006 and Yuce at al., 2013 and is inspired by the way in which it is believed honey bees find their food sources. In nature honey bees send workers to search for potential food sources (in the algorithm this is assumed to be a random flight path). Once a bee has found a food source they will return back to the hive and perform a "waggle" dance. The "waggle" dance is a series of figure of eight movements that, it is assumed, communicates to other bees the direction, distance and quality of the food source discovered. After this information has been passed on the hive collectively decides how many bees to send to this newly discovered food source, the bees will constantly update the hive on their return of the current state of the source so that the hive can decide when to abandon the source as exhausted or move to a relative better source (closer to the hive, better quality food etc.). In terms of how this is implemented as an optimisation algorithm the honey bee food search is mapped onto the algorithm search of the solution feature space.

The steps involved in the basic ABC algorithm are as follows:

- Step 1 - Randomly allocate bees at locations in the solution feature space.
- Step 2 - Until stopping criteria are met perform the following.
- Step 3 - Evaluate the randomly selected locations using the evaluation function.
- Step 4 - Rank the locations.
- Step 5 - Select locations that have the highest rank from the evaluation function.
- Step 6 - For each location allocate a proportion of the remaining bees not allocated to high ranked sites randomly to locations within a patch region around the high ranked sites.
- Step 7 - For each location and the surrounding patch region evaluate all the bees and keep the highest evaluated bee at each location.
- Step 8 - Recentre the location and patch region on the highest ranking bee.

- Step 9 - Allocate any remaining bees not assigned to locations and patch regions to a random location within the feature space.
- Step 10 - If stopping conditions have not been met return to step 5.

There are a number of parameters required for the ABC algorithm; the total number of bees being used (n), the number of locations (sites) that will be selected as viable solutions per iteration (m), number of locations within the patch size of each site (e) and initial size of each patch (s). At step 1 a decision is taken as to the size of m, and a bee is allocated to each of these sites, this leaves the algorithm with (n-m) bees for steps 6 and 9. The locations in the solution feature space are initially randomly selected, these locations are then presented to an external evaluation function, in this step the evaluation function takes the vector representing the location (which is a combination of the input variables of the function being optimised) and fits them to a model/algorithm to produce a single value. Taking the value from the evaluation function the ABC then ranks the locations. The ranking can either be in the form of a minimisation or maximisation effort depending on what the evaluation function represents. The ABC then enters the iterative portion of the technique, it will check against certain stopping criteria whether to continue with the algorithm. The stopping criteria are typically; a maximum number of iterations (maximum generations), the return value from the evaluation function reaching a maximum or minimum value, new iterations of the algorithm not producing better candidates than the previous iterations or some combination of these.

Whilst the selected stopping criteria has not been met the algorithm will next select the m locations that are the highest ranked by the evaluation function, at each of these locations e locations are selected at random within the patch size, s, of the central location. This is done for each of the high ranking m sites. After this operation the algorithm will have a total of n-(m*e) bees remaining that have not been allocated to locations within the feature space, these will be used at step 9. To this end when selecting the total number of bees being used in the algorithm, the number of high ranking sites retained and the number of extra locations within each patch the the algorithm must ensure that n > (m*e). For each location and the locations in the surrounding patch s the framework uses the evaluation function to find the highest ranking site. After finding the highest ranking site the algorithm re-centres the patch on that site (or if the original location is still the highest ranking site the patch remains in its current location). This operation represents a local search close to the best currently known sites, it is the fine tuning part of the search as the algorithm attempts to locate the best solution nearby. The bees that remain unallocated are now sent to random locations within the search space in a manner similar to the very first starting point. This operation represents an attempt by the algorithm to avoid local minima. Local minima occur when the algorithm finds a location that is locally the best solution but where a better solution exists globally in the feature space. Many machine learning algorithms struggle with local minima and include strategies to prevent the algorithm from concentrating on the local best solutions too quickly, often, as is the case with ABC, this involves a random search elsewhere in the feature space for optimisation algorithms. Once this step has been completed the algorithm will have bees at the best performing m sites along with a number of randomly selected new sites and it will begin the iteration again provided none of the selected stopping criteria have been met.

7.4 Firefly Algorithm

The Firefly Algorithm (FA) is described in Yang, 2010 and Fister *et al.*, 2013, this is another optimisation algorithm inspired by nature but this time by the mating habits of fireflies. In nature fireflies attract mates by producing a light in their abdomen through chemical bioluminescence. In terms of the algorithm the light given off by each firefly attracts others to them hence the individual fireflies move through the feature space to eventually converge at a point which it is hoped is a good solution. The steps involved are as follows:

- Step 1 - Randomly allocate fireflies at locations in the solution feature space.
- Step 2 - Until stopping criteria are met perform the following.
- Step 3 - Evaluate the randomly selected locations using the evaluation function.
- Step 4 - Rank the locations.
- Step 5 - Adjust each firefly's attractiveness based of the evaluation rank.

- Step 6 - For each firefly move it closer to every other firefly based on the attraction between them.
- Step 7 - if stopping conditions are not met return to step 3.

Similarly to the ABC algorithm there are a number of variations on the basic firefly algorithm, the standard model as described in Fister *et al.*, 2013 follows the following procedures. In similar fashion to the ABC the FA randomly allocates a number of fireflies around the feature space but where the ABC reserves some bees for redistribution later in the algorithm the FA distributes all the fireflies in the first instance. The FA then enters the interaction loop, again it will stay in this loop until some stopping criteria is met, typically for the FA this will be a maximum number of generations but the algorithm could use other checks in a similar way to the ABC algorithm. The algorithm uses an external evaluation function in the same manner as the ABC algorithm to rank all the current positions of the fireflies. The algorithm then uses the ranking of the position to calculate each firefly's light intensity value (I), different implementations of the algorithm use variations of this relationship but the most simple is given in equation 7.1 where it is seen that the light intensity is set to the outcome of the evaluation function (f).

$$I(x) = f(x) \tag{7.1}$$

Other variations of this equation use a form that takes into account the fixed light absorption coefficient (γ), this will also be used later when determining firefly movement. For each pairing of fireflies move the firefly with the lowest light intensity value towards the firefly with a higher light intensity (this represents fireflies moving towards attractive mates in nature). The operation for moving one firefly towards another is given by the equation 7.2 below:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} + \alpha \epsilon_i \tag{7.2}$$

The movement of the firefly composes of 3 sections. The β_0 term is the attractiveness parameter, this is set at the beginning of the process and according to Yang, 2010 is often set, initially, to a value of 1. The next section $e^{-\gamma r_{ij}^2}$ represents the light intensity (and therefore the attractiveness) lessening over distance due to light intensity dispersing in the air. This is represented by the parameter γ, which is the absorption coefficient (again Yang, 2010 states this often initially begins with a value of 1) that represents how much light intensity is lost and term r_{ij}^2 which is the euclidean distance between the two fireflies (i and j) given by the equation in 7.3.

$$r_{ij} = ||s_i - s_j|| = \sqrt{\sum_{k=1}^{k=n} (s_{ij} - s_{jk})^2} \tag{7.3}$$

Here n denotes the dimensionality of the feature space, s_i and s_j are the positions of the two fireflies in question.

The final part of the firefly movement term is $\alpha\epsilon_i$, this term is a random walk term that controls the step size with α being a random parameter between [0,1] and ϵ_i being another random number drawn from a probability distribution (usually Gaussian). The random walk part of the movement is the local minima avoidance function for the FA, by moving fireflies in a route that isn't directly at the most attractive firefly but allowing them to deviate slightly this tries to sample parts of the feature space that otherwise would not be visited. The top ranked firefly in each iteration will not find a firefly whose attractiveness is above their own, in this circumstance the random walk part of the movement function is the only part that takes place, again this is a strategy to avoid local minima by not having fireflies simply converge on the best location found in the first iteration of random location selections.

In terms of stopping criteria the common ones to use for the FA are either a maximum number of iterations or the algorithm will converge as fireflies move towards each other. If convergence is selected then a small threshold can be used to prevent the final stages of the algorithm from not terminating due to small oscillations.

7.5 Evaluation/Fitness Function

In both of the optimisation methodologies there is a requirement to rank the solutions, in order to do this each method needs to pass the solutions to an external evaluation / fitness function that can produce a single value for the solution and allow the method to rank them in terms of the best performing solution. This is often the bottleneck in terms of efficiency with optimisation methods, we also have to take into account here that both the ABC and FA are swarm methods that use multiple agents (locations) and we have a trade-off to make between the amount of time taken by the evaluation function to rank each agent and the accuracy and speed to convergence gained by using more agents.

In the case study we are using for this work our evaluation function is a combination of the results passed out of the prediction methods that show the ranking of each prediction algorithm / hyper-parameter setup and a weighted vector representing the relative importance of each of our metric measures. The particulars of this will be discussed in section 7.8 when this report examines the implementation of these methods.

7.6 Comparison of ABC, FA and to other optimisation methods

In terms of the framework itself there is no need for more than one optimisation method, this work trialled two to show that the choice of optimisation algorithm can be changed and customised to better suit the particular industrial process that is being modelled. Comparing the two methods used in this work there are differences between the ABC and FA methods that may lead to one choice over the other. In terms of complexity the ABC method is a more complex method with a larger number of adjustable parameters, 7 (number of bees, number of sites, number of bees allocated to best sites, number of best sites, patch size, and number of bees allocated to random location not best sites.) compared to the 4 (number of fireflies, attractiveness parameter, absorption coefficient and randomisation parameter) for FA.

This leads to the ABC method being more flexible in terms of adjustments to better fit the application but does lead to an increased difficulty in terms of fully optimising the optimiser.

As noted earlier either of these two methods could be replaced in the framework by a different optimiser and there are a great many different methods described in the literature, two of the most common are Genetic Algorithms (GAs) and the Particle Swarm Optimiser (PSO). Pham *et al.*, 2006 concluded that under a number of circumstances the ABC method outperformed that of GAs, again the drawback found was that the ABC method has a large number of hyper-parameters that need to be tuned compared with the much simpler GA method. In the work carried out here GAs were decided to be an unsuitable method for testing as the default method requires the inputs being considered be binary. Given that what is being optimised in this instance is the ranking of particular regression algorithms and their variations this would require an extra step of processing that isn't necessary for both the ABC and FA methods. In terms of the PSO method this is very closely related to the FA method with both methods using almost the exact same ideas and implementation. It is the case that if the FA algorithm is used with an absorption co-efficient (γ) of 0 then both methods behave in exactly the same way.

7.7 Implementation of Optimisation Algorithms

In the work presented here two Python libraries have been used for the implementations of ABC and FA. For ABC, the library used was BeeColPy (Oliveria, 2020) and the for FA, 5Gcraft_FireflyAlgorithm (Twain OpenAI Club, 2021). Both of these libraries are available under the MIT license.

The BeeColPy library is an implementation of the simple ABC algorithm and takes as hyper-parameters the evaluation function, the boundaries and dimensions of the feature space, the colony size, the number of scouts for each patch, the maximum number of iterations and whether the evaluation function is treated as a maximise or minimise function. The number of scouts is taken as a value between 0 and 1 and represents the percentage of bees from the main colony that will be allocated to the best performing patches. The initial number of patches is always set to 50% of the total colony size. For the initial trials this work began with the default settings of a colony size of 40 bees, scouts set to 0.5, and a maximum of 50 iterations. In terms of the boundary sizes of the feature space the experiment carried out for this work uses the results of the previous experiments with the five prediction algorithms so the boundary sizes were from 0 to 4 (one for each of the ranking values).

The 5Gcraft_FireflyAlgorithm is an implementation of the firefly algorithm and takes parameters for the number of fireflies, maximum number of iterations, α (randomisation parameter), β (attractiveness parameter), γ

(absorption co-efficient) and the boundaries of the feature space. Initially the parameters used for this work were left with the default settings for α (0.5), β (0.2) and γ (1.0) and the settings for the number of fireflies and maximum iterations were set to match the settings used for the ABC algorithm (40 and 50 respectively). Again the boundary sizes were the same as the ABC algorithm due to both methods being supplied with the same inputs.

7.8 Implementation of Evaluation Function

Both the ABC and FA methods, in keeping with most optimisation methods, require an external evaluation function, this function takes each solution presented to it by the optimisation method and produces a single numerical rank value which the optimisation algorithm will then use to rank the solutions depending on whether the goal is is maximise or minimise the output. In the work presented here the evaluation function used is a combination of the results obtained from the previous prediction rankings and a weighted vector of the relative importance of the metric measures discussed earlier. The ranking is calculated using equation 7.4 below.

$$Rank_i = \sum_{j=1}^{n} (p_{ij} * w_j) \tag{7.4}$$

Where:

i is the time period being ranked.

j is the jth metric ranking.

n is the total number of individual prediction rankings being considered per time period.
p is the matrix of prediction result rankings.
w is the weighted relative importance vector.

In the work here, in the previous chapter the system considered 1 minute intervals of predictions, this means that in our matrix of prediction result rankings each row represents 1 minute of predictions. If the system now wishes to optimise for 5 minute intervals it would need to consider the ranking for five rows of combinations. The weighted relative importance vector is a series of values that map directly to the row rankings in the prediction matrix. By adjusting the values in this vector it is possible to increase the emphasis placed on each of the metrics. A value of 0 in the importance matrix would mean that particular metric would be ignored by the evaluation function whilst any negative value would mean that having a poor metric score would be beneficial to the ranking. It is not anticipated that a negative ranking would ever be used in practice. In this work all values were used between 0 and 1, although as this is purely a ranking system the values could be adjusted to any range of positive values.

.

The setting of the weights in the the weighted vector can either be done directly, by a user that understand what each of the metrics represents, or in the case of a non-specialist user it would be a simple matter to ask a series of questions to guide the weighting of the metrics; is the process particularly sensitive to large variations in accuracy? In which case the framework could adjust the weights to favour metrics that penalise high variance (such as maximum error in this work), or if the process can cope with a few, occasional, large inaccuracies provided the majority are within a prediction range then the framework could prioritise metrics that favour more accurate predictions (such as mean squared error in this work).

7.9 Summary

In this chapter the report has detailed the two bio-inspired optimisation methods used as part of this work, in the final framework there would only be the need for a single optimisation method and the selection of this would need to be determined by the suitability of the method to the application being modelled. The next chapter will detail the results of the trials performed using the two chosen methods.

Chapter 8

Optimisation Results and Conclusion

8.1 Results.

The initial set of tests were run against the default settings for the BeeColPy ABC algorithm and 5Gcraft_FireflyAlgorithm, the number of agents (bees for ABC and fireflies for FA) and maximum iterations were varied in order to gauge what influence changing these parameters had. Iteration values of 50 (default setting), 100, 500, 500 were used in combination with 40 (default setting), 100 and 500 agents. In terms of statistical significance a one tailed ANOVA analysis was run against the various experiments shown here to determine how different each of the sets of runs actually were, the results are given in table 8.1.

The results here show varying levels of variance across the runs but with

Figure	Method	No. of Iterations	No. of Agents	F-Value	P-Value
8.1	ABC	50	40	0.452181	0.715895
8.2	FA	50	40	2.922748	0.033796
8.3	ABC	100	40	0.663841	0.574680
8.4	FA	100	40	2.047476	0.106689
8.5	ABC	500	40	1.634690	0.180772
8.6	FA	500	40	5.291889	0.001377
8.7	ABC	5000	40	0.601309	0.614474
8.8	FA	5000	40	7.513479	6.663673e-5
8.9	ABC	100	100	0.288718	0.833552
8.10	FA	100	100	0.123331	0.946310
8.11	ABC	500	100	0.374625	0.771356
8.12	FA	500	100	0.486184	0.692063
8.13	ABC	5000	100	0.085960	0.967703
8.14	FA	5000	100	5.844240	0.000649
8.15	ABC	500	500	0.560239	0.641548
8.16	FA	500	500	0.642246	0.588224
8.17	FA	5000	500	0.239569	0.868736

Table 8.1: One-tailed ANOVA results for agent\iteration experiments F-Value is the ratio of the Mean Squares Treatment vs. Mean Squares Error, the larger this statistic the greater the variation between sample means relative to the variation within the samples.

P-Value is measure of statistical significance of the group variation. Typically values of less than 0.05 are considered of interest and may be statistically significantly different.

very few exceptions show no statistical significance, the three that are worth singling out are the tests at figure 8.6 (FA, 500 iterations, 40 Fireflies), figure 8.8 (FA, 5000 iterations and 40 fireflies) and figure 8.14 (FA, 5000 iterations and 100 fireflies), these three have very low P-values that suggest these are significantly different sets of results. These tests have in common that they are all from the FA method.

With very little variation observed in changing the iterations and number of agents the next set of tests were run against the default settings for the BeeColPy ABC algorithm, the number of agents (bees for ABC and fireflies for FA) and maximum iterations were replicated for the 5Gcraft_FireflyAlgorithm (40 agents and 50 iterations), whilst all other settings were again left at the default values for the library implementations (see previous chapter). Each trial was run four times to see the spread of results for each group of settings. With any optimisation algorithm there is no guarantee that the algorithm will find the best solution, this can only be made certain by testing every possible solution in the feature space, which is what the optimisation strategy is trying to avoid. In some circumstances there can be more than one solution that produces the same ranking score and in order to see the variability of the optimisation methods each experiment was run multiple times. The result set being used from the prediction algorithms is the uncorrelated, smoothed set using 500000 training examples (see chapter 6 previously for details). For all of the figures that follow the prediction algorithms listed on the Y axis are as follows:

- 0 - Elastic Net.
- 1 - Lasso
- 2 - Ridge (Ridge)
- 3 - Ridge (SGD)
- 4 - Random Forest

Figures 8.1 and 8.2 show the results of the ABC and FA optimisation algorithms respectively. In each case the experiment was repeated four times and the first 100 values (minutes) were used for the optimisation.

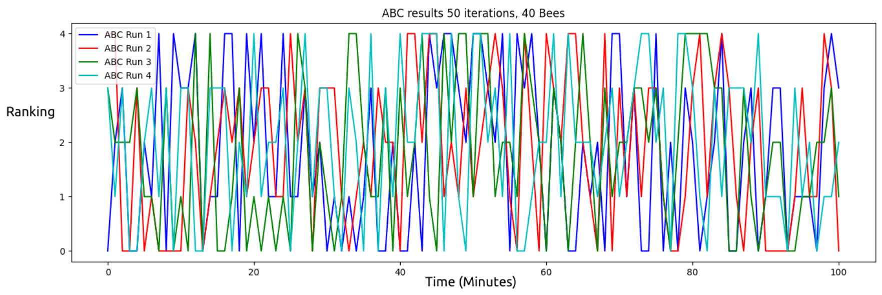

Figure 8.1: *First 100 minutes of prediction results optimised using ABC algorithm*

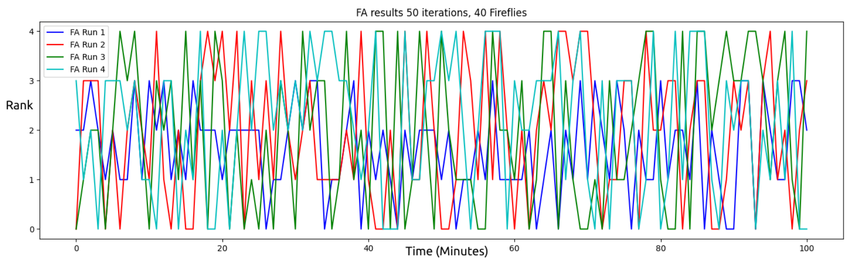

Figure 8.2: *First 100 minutes of prediction results optimised using FA algorithm*

For both of the methods it can be seen from figures 8.1 and 8.2 a large amount of variation in the results, neither method has managed to produce a consistent series of prediction results. Figures 8.3 and 8.4 show the same results after an increase for both methods of the maximum number of iterations from 50 to 100.

Figure 8.3 shows that for the ABC algorithm increasing the number of

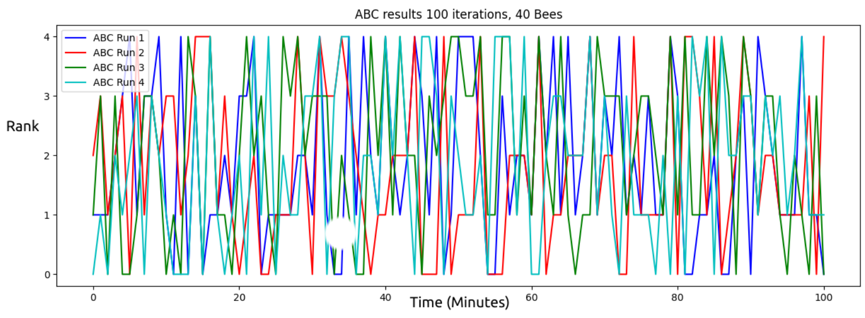

Figure 8.3: *First 100 minutes of prediction results optimised using ABC algorithm*

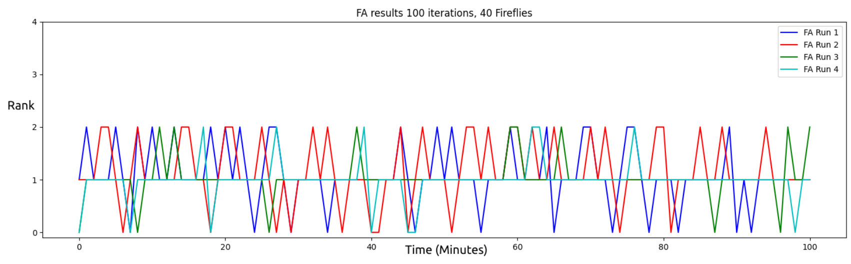

Figure 8.4: *First 100 minutes of prediction results optimised using FA algorithm*

iterations has very little effect on the results produced, there is still a great deal of variation in the results produced. Figure 8.4 shows an interesting feature of the FA method, in the four runs produced here FA never selected either the Ridge (SGD) or Random Forest methods. Whilst it is possible that neither of these prediction methods were the best choice for that particular minute it is unlikely, given results seen in other runs, that this was the case throughout the entire first 100 minutes. What is more likely to have happened is that no solution containing those two algorithms ranked well in the first round of evaluations. The progression for the FA algorithm then moves solutions towards each other, but if no attractive solution exists at the edges of the feature space then there is a chance that those particular areas will not be explored. The FA tries to mitigate this situation by having the fireflies random walk component, the $\alpha\epsilon$ term discussed in the previous chapter, however with this being a purely random factor that adjusts the fireflies movement it is still possible for these edge areas to not be explored. The ABC method is somewhat less prone to this problem as the random local minima avoidance strategy for this method is not linked to the locations of current solutions. The ABC minima avoidance strategy is purely random in nature not having solutions moving away from certain areas does not decrease the probability of those areas being selected as random solutions in future iterations.

In the next set of experiments the iterations are again increased, this time to 500.

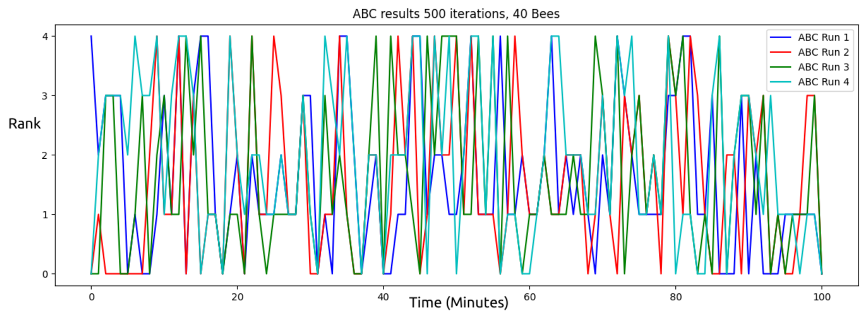

Figure 8.5: *First 100 minutes of prediction results optimised using ABC algorithm*

Figure 8.5 shows that there is a decrease in the variation of the solutions, increasing the number of iterations allows the ABC optimiser to move closer

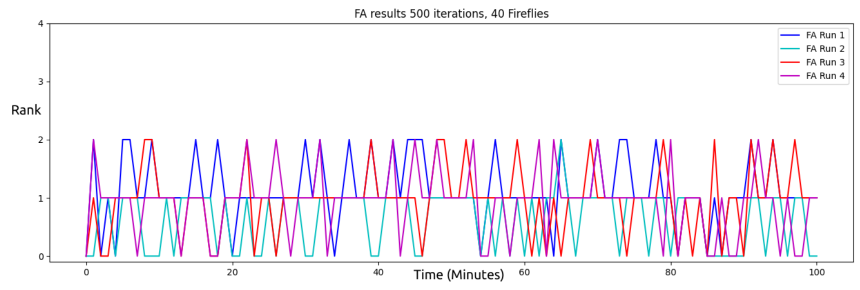

Figure 8.6: *First 100 minutes of prediction results optimised using FA algorithm*

to convergence. Figure 8.6 Again shows a situation were the FA method has not selected either Ridge (SGD) or Random Forest algorithms. At this level for iterations the FA method is not showing reduced variation in the same manner as ABC.

Figures 8.7 and 8.8 are the final experiments with raising iterations at 5000 maximum iterations each.

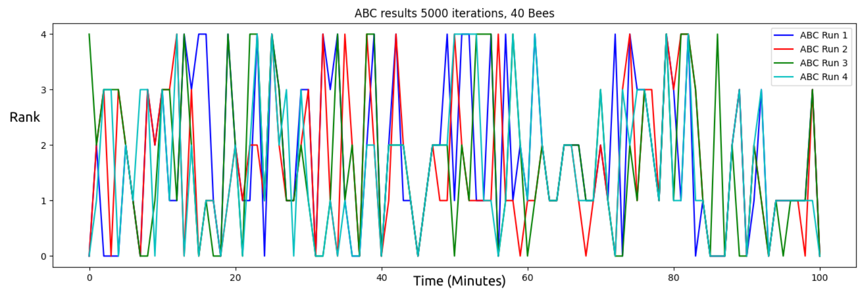

Figure 8.7: *First 100 minutes of prediction results optimised using ABC algorithm*

After 5000 maximum iterations, for this particular dataset the variations in solutions in the ABC method have been reduced, by increasing the max-

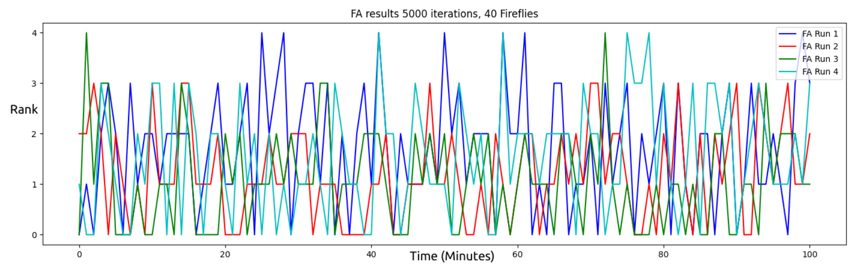

Figure 8.8: *First 100 minutes of prediction results optimised using FA algorithm*

imum iterations the effect on the ABC optimisation method is a reduction in the number of possible solutions delivered by the method. The effect of increasing iterations on the FA method is less noticeable and whilst there is a definite difference between the first and last of these experiments (50 and 50000 iterations) the effect is not as noticeable as we might expect and certainly less so than the ABC method. Some of this may be explained by the FA method not selecting certain results in earlier experiments which of course naturally limits variation.

After adjusting iterations of each methods the next variable to try was to adjust the number of agents in combination with the iterations. To this end the experiment increased the number of agents to 100 and again varied the number of maximum iterations, figures 8.9 and 8.10 show the ABC and FA methods respectively using 100 agents (bees and fireflies) at 100 maximum iterations.

Comparing figure 8.3 and 8.9 it can be seen that this increase in the number of agents for the ABC method appears to have very little effect using

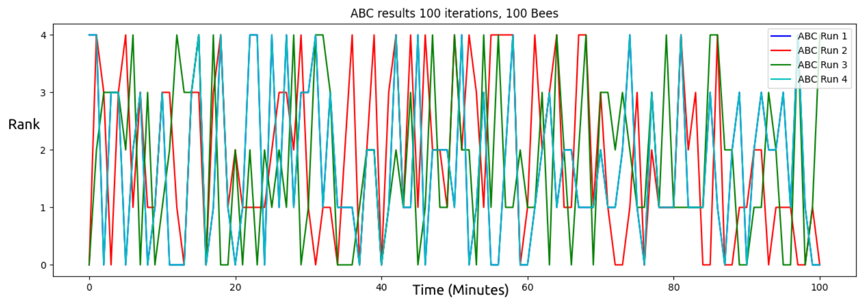

Figure 8.9: *First 100 minutes of prediction results optimised using ABC algorithm*

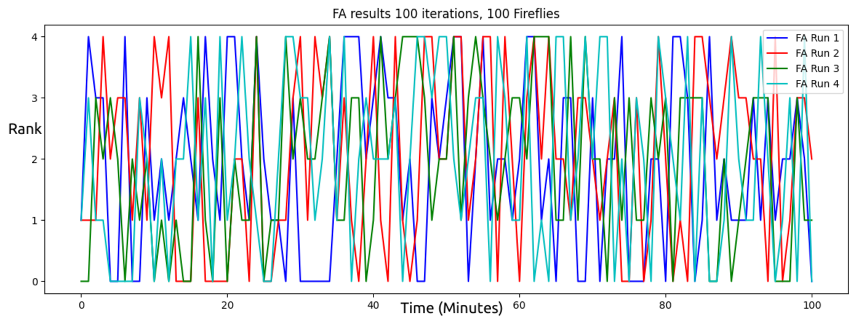

Figure 8.10: *First 100 minutes of prediction results optimised using FA algorithm*

this dataset. Comparing figures 8.2 and 8.10 again there is very little change in the variability of the optimised solutions. Next the experiment tries increasing the number of iterations to 500 along with 100 agents

Again the effects seen for the ABC method are not obvious, in terms of the FA algorithm there is again a situation where the method did not choose two of the prediction algorithms. Next the experiment tried 100 agents at 5000 iterations.

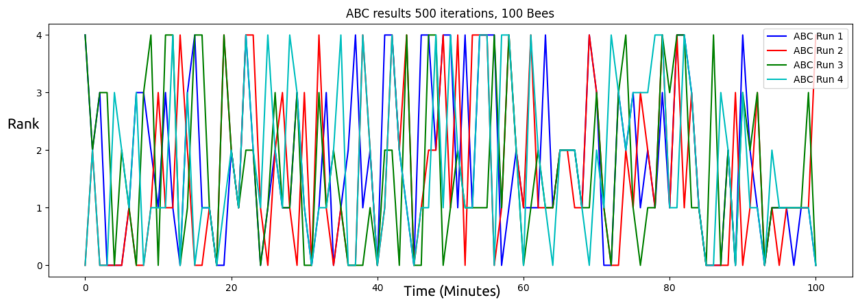

Figure 8.11: *First 100 minutes of prediction results optimised using ABC algorithm*

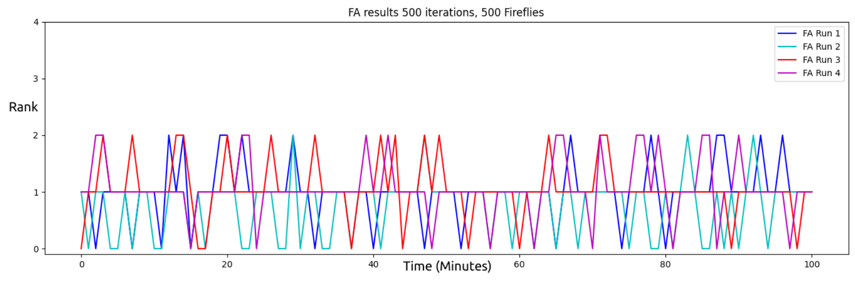

Figure 8.12: *First 100 minutes of prediction results optimised using FA algorithm*

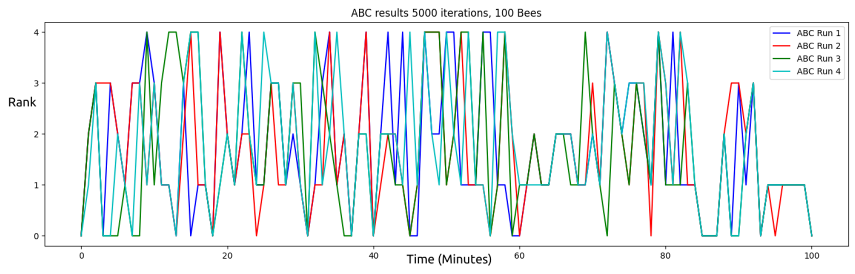

Figure 8.13: *First 100 minutes of prediction results optimised using ABC algorithm*

Again with a large number of iterations it can be seen that the ABC algorithm has reduced the amount of variation whilst the FA method is still quite varied in its range of results. Once again this is highlighted more than

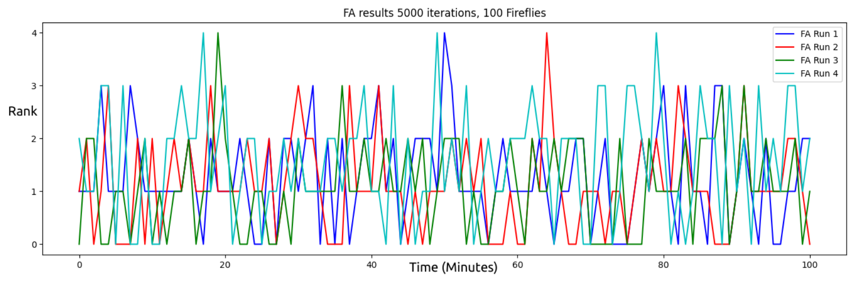

Figure 8.14: *First 100 minutes of prediction results optimised using FA algorithm*

the previous experiment as the algorithm has used the full range of prediction algorithms in its selections. Next the experiment looks to see if a large increase in the agent numbers has an effect on its own by running 500 agents against 500 iterations.

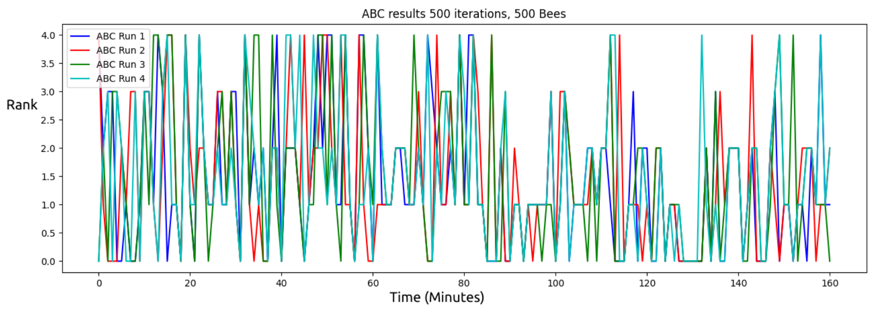

Figure 8.15: *First 100 minutes of prediction results optimised using ABC algorithm*

Comparing the results of the ABC method here with 500 bees and 500 iterations with the previous experiment in figure 8.5 where there had been 40 bees and 500 iterations it can be seen only a slight decrease in variability of the solutions, especially when compared to the previous experiment (figure 8.15) which had only 100 bees but 5000 iterations and far less variation in

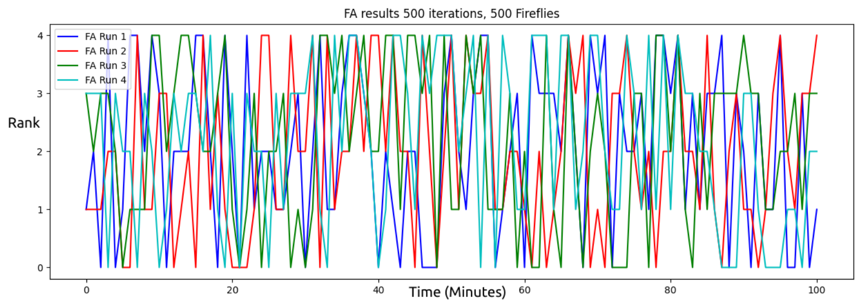

Figure 8.16: *First 100 minutes of prediction results optimised using FA algorithm*

results. It would appear that increasing the number of bees does not immediately lead to the method reaching a stable position. In terms of the FA method when comparing figures 8.16 and 8.14 it can be seen that there is no increase in stabilisation that can be attributed to agent increase.

With the ABC method responding to an increase of iterations over agents a final experiment to look at the effect of these two parameters was attempted for the FA method to see if a large increase in both would produce any effect. Figure 8.17 shows the FA method with 500 agents (fireflies) and 5000 iterations.

This final experiment for the FA method again does not select either the Ridge (SGD) or Random Forest methods, however if this is compared to a similar experiment (figure 14, 500 fireflies and 500 iterations) there is not a noticeable level of change in the variation, from this it can be concluded that, for this dataset, the FA method is much less quick to converge on a set of solutions than the ABC method.

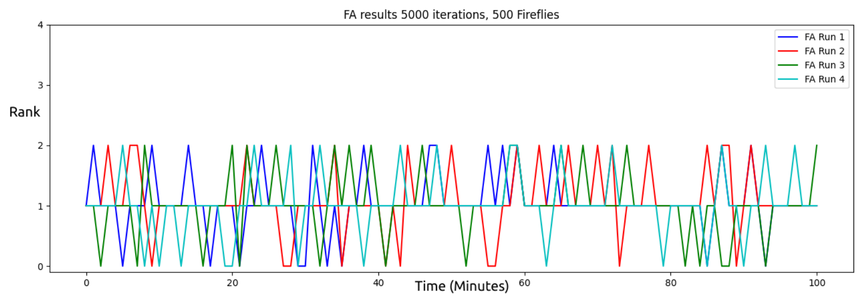

Figure 8.17: *First 100 minutes of prediction results optimised using FA algorithm*

The final experiment undertaken was a demonstration of the way in which a set of solutions could be affected by the choice of accuracy metric and discussed in chapter 5, the goal of the weighted vector in the evaluation function is to be able to bias the results towards particular types of accuracy measure. By adding weights to the prediction results the framework can optimise the solutions towards the behaviour it wishes the process being modelled to exhibit. In the example in figure 8.18 the weight vector has been altered to favour metrics that result in a lower variance of the prediction results. To that end the weights for the Expected Variance, R2 Score and Maximum Error have been left as 1 and the values for all other metric tests have been reduced to 0.1 for the 3rd and 4th runs. The first two runs have been left with all the weights equal. 100 prediction values have again been used but this time they were selected from the 2nd set of values from minutes 100 to 200. For this demonstration the ABC method was selected with 100 bees at 5000 iterations.

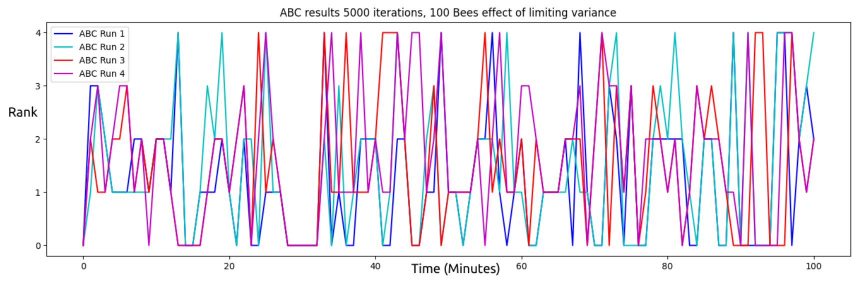

Figure 8.18: *Minutes 100 to 200 of prediction results optimised using ABC algorithm, with weight vector adjustments*

Figure 8.18 shows that by altering the weight vector two different sets of solutions are produced. Although there is still variation between all of the results it is clear that runs 1 and 2 have more common selections with each other than 3 and 4 and similarly the other way around.

8.2 Conclusion

The work presented here uses an optimisation method to produce an acceptable sequence of prediction algorithms using a combination of the results and a weighted vector to prioritise certain aspects of the accuracy measurement metrics. There are many different types of optimisation method and this is still very much an active research area producing new methods and variations on older methods all the time. The framework described in this work would use one such optimisation method to take future predictions and the weight vector to guide the process through which regression algorithm would be the best performing at any given time and when to switch over from one to another. The work here looked at two such methods the Artificial Bee Colony (ABC) and the Firefly Algorithm (FA) and various tests where undertaken to look at the effects of the number of agents used and iterations of the algorithms. The experiments that were carried out demonstrate that some of the effects of changing parameters are common to all optimisation algorithms. Each optimisation method on its own also has a number of hyper-parameters that affect the performance of that particular type of optimiser (these were discussed in the previous chapter). The work presented here did not spend time looking at trying to tweak these hyper-parameters as, has already been stated, the framework itself is not tied to a particular method and the choice of optimiser would be something that would be explored based on matters such as available hardware / software requirements and the features of the process being modelled.

The examples shown here are simple demonstrations of what would be a very small set of results in an actual industrial process, given the small number of prediction variations (5) it could be possible to brute force an actual 'best' answer for the most accurate prediction algorithm at each minute in time. However in a situation where there are potentially tens or hundreds of possible regression models and associated hyper-parameters the need for optimisation as an automated process becomes clear. There is an obvious problem of how, having selected a preferred optimiser, we would optimise the hyper-parameters of the optimiser (or even choose between several optimisers), this becomes a recursive problem (an optimiser for the optimiser) and at some point the framework will have to settle for either a "good enough" default set of parameters or accept that some amount of experimentation would be necessary. By the nature of optimisers they will very often produce a different set of results every time they are run, this is due to them containing random elements to help search the feature space for solutions. The results presented here show that there can be quite a variety of different solutions produced for the same set of parameters and inputs.

The set of experiments run for the purposes of demonstration here show some interesting behaviour in the two selected optimisers. The ABC seems to respond quite quickly to an increase in iterations and this can be a way to try and move towards convergence if the process needs to produce the optimal or very near optimal solution. Of course increasing the number of iterations also has a large effect on runtime and increasing the iterations for the ABC algorithm in this study greatly increased the amount of time it took to run the process. The FA method here showed a much slower move towards convergence with increased iterations than the ABC method, of course given a large enough iteration size all methods would reach convergence as, at the worst, each method would check every possible solution. Part of the reason may be down to the default values used in the libraries used in this presented work and an adjustment of the parameters can lead to faster convergence. There has been an assumption made that the library authors used typical generic values for their implementations but it may simply be the case that those particular sets of hyper-parameters are unsuitable for this particular set of results. Again there is little to be gained in spending time exploring these parameters to produce an optimal set of hyper-parameters for this particular set of results from this particular data set. The goal of this particular section of the framework is to demonstrate where an optimiser fits into the framework in general, why one would be used and what some of the issues are. One of the observations that can be noticed from these results is that both the optimisers do swap over the regression algorithms quite often and do not hold on to using the same algorithm for extended periods of time, this could have some consequences in an industrial environment and this will be discussed in the next chapter.

Chapter 9

Supervisory System

9.1 Introduction

The goal of the optimisation algorithm is to produce a decision stating which of the potential prediction algorithms (and hyper-parameters) are considered to be performing best over the selected time period. However just because an algorithm is currently evaluated as the best performing option this does not necessarily follow that it makes sense in an industrial environment to change to that particular configuration and algorithm. There are a number of other factors that need to be taken into consideration. For this the framework uses a top-level supervisory system that would manage the exchange of algorithm from the test to live control system.

Factors that have not been considered to this point when selecting the best algorithm include:

- Concept Drift.

- Switching Time.
- Algorithm Stability
- Cost Benefit Analysis of Switching.

9.2 Concept Drift

Concept drift is described in Gama, 2012 as "...the concept about which data is obtained may shift from time to time, each time after some minimum permanence." this means that for a continuous stream the underlying distribution of data changes over time. There could be many reasons for this in an industrial setting E.G. a machine tool that is slowly drifting out of alignment will start to produce a different profile of parts. In many instances these are conditions that an industrial system would be trained to identify and this would be the classification targets. In many circumstances drift can occur and the only symptom would be an increasingly inaccurate prediction system E.G. if the system was trained to predict the position accuracy of a hole being drilled a wearing drill bit could cause the position to vary leading to increasingly inaccurate estimations of positions. The system needs to look for this concept drift by monitoring the inputs to look for changes in the underlying distribution. One of the most used systems for this is that of the Page-Hinkley test described in Gama, 2012. The Page-Hinkley test looks for changes in the data which it assumes to be Gaussian in nature (this may or may not be true for any given industrial process but will be for a good number of them) and monitors if the mean of a distribution exceeds some threshold. The metric m_t is calculated as shown in equation 9.1:

$$m_T = \sum_{t=1}^{T} (x_t - \hat{x}_T - \delta) \tag{9.1}$$

Here t is the index of the observed value x, δ is a value used to control the amount of change that will be allowed and $\hat{x}_T$ is calculated using the equation given in 9.2 :

$$\hat{x}_T = \frac{1}{T} \sum_{t=1}^{T} x_t \tag{9.2}$$

In order to calculate the Page-Hinkley metric to determine whether drift has occurred, the metric M_T is calculated as shown in equation 9.3:

$$M_T = min(m_t, t = 1..T) \tag{9.3}$$

and finally detect for drfit using the equation in 9.4:

$$PH_T = m_T - M_T \tag{9.4}$$

This final equation in 9.4 is compared to a threshold value λ, if $PH_T > \lambda$ then it is assumed concept drift has occurred in the observations.

In the equations given T is the maximum period under consideration, the length of T corresponds to the maximum number of observations until a change in distribution is detected, at which point a new sequence is started

(t and T are reset to 0). The values used for the two values of δ and λ in (Gama, 2012) are 0.1 and 100 respectively. To demonstrate this the values from sensor 1 in the dataset were tracked, figure 9.1 shows the CO (ppm) value detected by sensor 1 over the lifetime of the experiment:

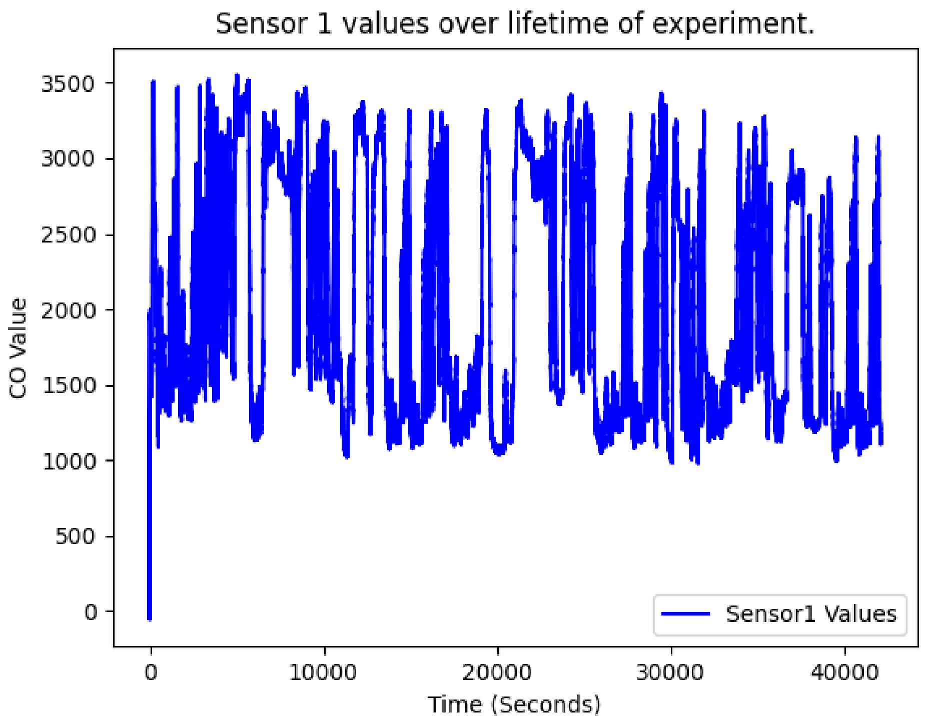

Figure 9.1: *Sensor 1 values over the lifetime of the experiment*

Figure 9.1 shows a great deal of variation in signal, which is to be expected given the nature of the data and the random levels of CO throughout the experiment. It is expected that the dataset will show an amount of concept drift, particularly at the beginning of the experiment when the sensors are first calibrating, but for this exercise was completed to see if the random gas concentrations would mean that the data significantly drifted away from a Gaussian distribution. The next figure (Figure 9.2) shows the m_t values throughout the experiment as calculated by equation 9.1.

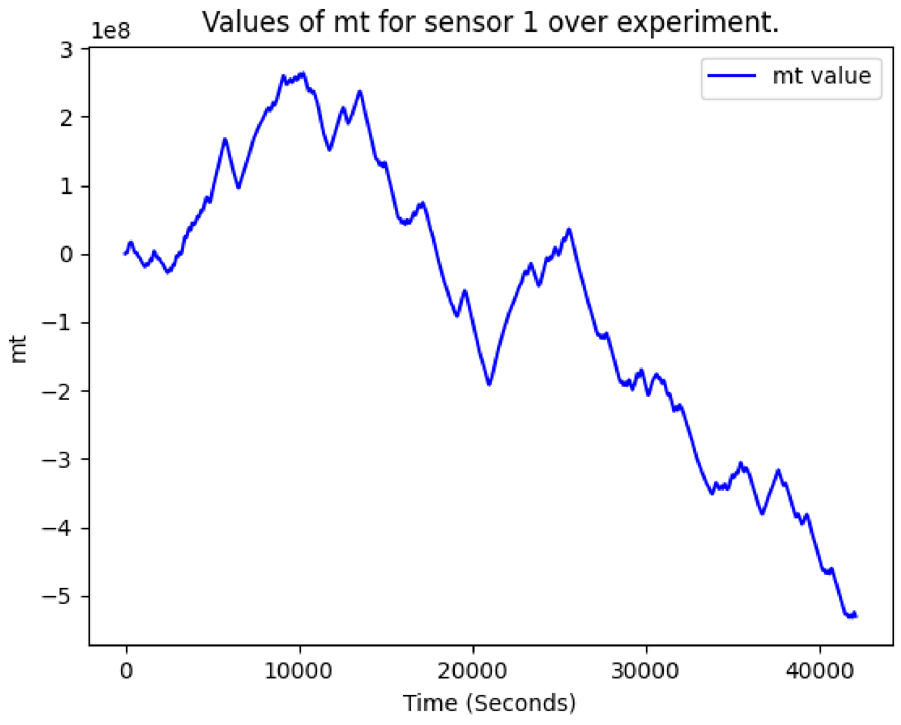

Figure 9.2: *Values of m_t throughout the experiment*

Then we apply the Page-Hinkley test from Equations 9.3 and 9.4 was applied to detect when the values from sensor 1 drift from a one distribution to another.

Figure 9.3 shows a large number of drift detections, figure 9.4 shows a smaller section during the 10000 and 20000 seconds range, as a typical mid-experiment time frame.

It is easier to see what is happening in Figure 9.4, there are sections of the experiment where it takes the sensor data an amount of time to settle into a new distribution, during this time it can be seen that the Page-Hinkley test

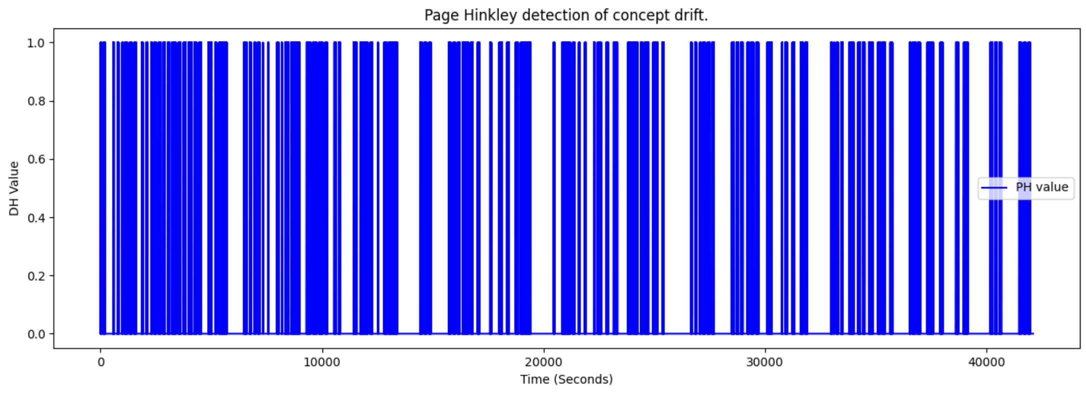

Figure 9.3: *Page-Hinkley detection of Concept Drift*

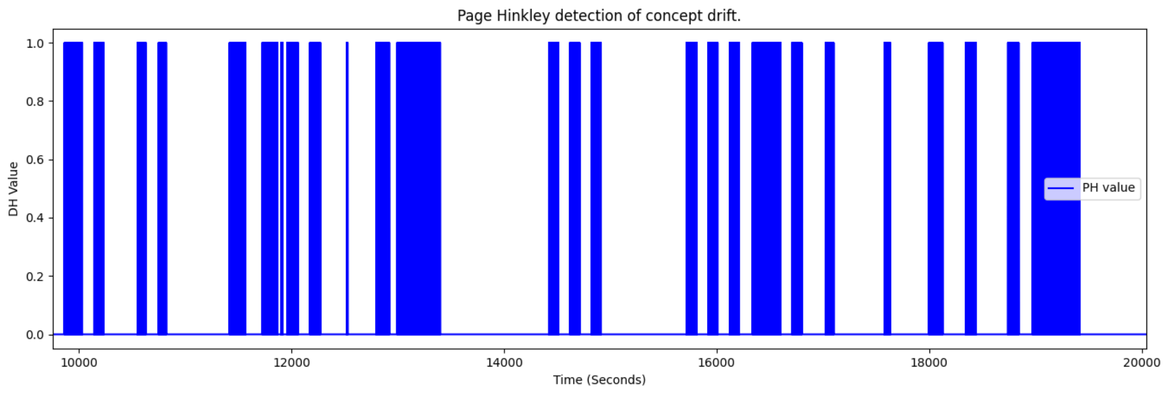

Figure 9.4: *Page-Hinkley detection of Concept Drift (mid experiment)*

registers that concept drift is occurring (shown as a value of 1.0 in Figures 9.3 and 9.4). As previously stated the particular dataset being used for this work was expected to show a number of drift detections due to the random nature of the gas additions in the original experimental design, in other applications, such as an industrial process producing parts, we would expect to see much less variation from the process, a breach of the Page-Hinkley threshold here would constitute a genuine drift in the data. The reason why the system needs to monitor for concept drift is that once drift has been detected any models using previous data from the system are likely to be invalid for future predictions, the system needs to be aware that previous results are invalid and that all models being evaluated need to considered on data from the point of concept drift detect into the future. At the point of concept drift detection the models need to be reset.

9.3 Switching Time.

Once an algorithm, or new set of hyper-parameters for the currently selected algorithm, have been identified the system needs to be able to factor in the amount of time it will take to actually switch over from using the current model to the newly selected one. The process of swapping between algorithms will not be instantaneous in most real world instances, with the need to load the new algorithm onto the live control system taking time. In terms of an industrial process it is likely that we would want to have both the old and new configurations running in parallel for a short period of time in order to confirm a successful transfer had taken place. This transfer period may be as short as keeping the old configuration running until the first observation from the new configuration can be seen. The physical time period needed to swap out the old configuration and substitute it with the new one will be architecture dependent and would need to be established via experimentation and piloting of an actual installation.

The reasoning for tracking a switching time for the algorithms is that it was seen previously in the prediction results that some algorithms managed to perform well but for brief periods of time. If swapping between configurations was instantaneous it might be beneficial to change over every time the system believed an algorithm would be the best performer, however in the cases where this may only be for a brief period of time the system needs to take into consideration the swap time so it does does not end up with an autonomous system entering into thrashing behaviour trying to replace algorithm configurations.

9.4 Algorithm Stability

Along with the time to switch over the system needs to consider how long an algorithm configuration will remain the best performing. If an algorithm configuration is switching between high and low performance constantly due to environmental or process conditions the system may need to not use the configuration unless it settles down into a more regular pattern regardless of it being the best performing algorithm at a particular time. In the experiment carried out here the tested configuration for Random Forest showed a large amount of variations, in practise this configuration would likely be rejected as too unstable and the system would be testing other Random Forest configuration looking for a more consistent set of results.

9.5 Cost Benefit Analysis of Switching

The supervisory system needs to establish what the benefits of switching the algorithm or configuration are. This requires examining a previous set of results. For most of the selection and optimisation process it was sufficient to rank the candidate algorithm configurations and choose the best performing. In the supervisory system it will be necessary to look back at the raw results from the prediction algorithms. In terms of switching between algorithm configurations it is desirable to know how much the system is gaining in terms of increased prediction accuracy. By referring back to the raw results the system can look at the actual difference between two sets of algorithm configurations. The system needs to make a decision between how much accuracy is gained and how often switching occurs. If two configurations are likely to be switching quite frequently then there would have to be a significant gain in accuracy to make each switch worthwhile for the system given the cost associated with switching. In a similar situation to the optimisation process there would be a need to use the weighted vector from the optimisation process to rank the importance of the metric results. The specifics of this would be related to the physical switching time (as previously stated this would be dependent on the hardware and software configuration on which the framework would ultimately be deployed on) but the system could be configured to only allow switching if a switch had exceeded a certain percentage accuracy increase since being tracked in testing stream (See Equation 9.5)

$$\sum_{t=1}^{T} Alg1_t - Alg2_t > S \tag{9.5}$$

Where Alg1 and Alg2 are the algorithm configurations being considered for switching, and S is some predetermined threshold set by the user. This process would only need to track the two top ranked algorithm configurations, if the two configurations being tracked changed before the threshold was tripped then the process would begin again with the newer algorithm selections. Once the threshold is exceeded the system would be clear to make the exchange (provided all other supervisory criteria where met) and the process would begin again monitoring the next best challenging algorithm configuration.

9.6 Conclusion

This section details the additional checks required by an overseeing supervisory system in order to ensure that an autonomous system selecting and switching out algorithm configurations maintains feasible operation. Although the previous chapters have detailed the processes of identifying and selecting a best performing algorithm configuration, in this chapter we also considered practicalities of this in an actual industrial system along with the limitations that the implementation has to work under. Concept drift looks for changes in the dataset that render previous observations obsolete. Switching time considers the physical limitations of the architecture that the system is running on. Algorithm stability looks at how long an algorithm configuration is likely to be useful for. Cost benefit of switching tries to determine whether the act of switching algorithm configurations is likely to bring any real benefits to the system. For the system to perform at an acceptable level the supervisory system needs to be able to confirm that all of these additional concerns are met before changing the prediction algorithm currently running the process.

Chapter 10

Conclusions and Future Work

10.1 Conclusion

The work presented has met the goals set at the beginning of the work with the one exception of not implementing a final switching mechanism in the test, this would need to be developed before the framework could be made available to SMEs. The goal of the project was to develop an algorithm, data set and platform neutral architecture that would be highly adaptive to any number of industrial processes that wished to use an autonomous set of machine learning techniques, and in this case this has been accomplished. The framework should not be confused with the experimental system that was created to demonstrate and test the specifics of the framework. That a regression task to predict the CO levels from a gas data set was developed using Python3 and a particular set of machine learning libraries was not the goal of the work, it was never the goal of this project to optimise this particular task but to develop a framework under which such a task could be done. The framework itself can be used to guide any number of industrial projects towards a wider usage of machine learning in industrial, and in particular manufacturing, sector where take up of these tools has remained slow for SMEs.

10.2 Future Work

The work presented is capable of testing multiple regression algorithms and using optimisation techniques to find near optimal sequences of the algorithms. The work has also demonstrated the ability to identify concept drift in the target signal and the ability to consider information outside of the optimisation results such as algorithm stability, cost benefit of switching the configurations and time taken to switch. The work stopped short of actually implementing the autonomous switching function, this would be the most obvious next step as this was one of the major goals of the work, depending on the hardware available the recommendation would be to run the previous selected configuration and the new configuration in parallel briefly until it could be established (again automatically by the system) that the newer configuration was up and running, this would eliminate the potential for gaps in process control.

The prototype framework developed in this work was done so using the Python3 language, this was used due to the availability of libraries for the various regressions and optimisation algorithms used during the tests, Python is well known as a language that is well supported in the machine learning field. Once the system built using the framework moves beyond the lab into an industrial environment it is probable that a different language would be more suitable, the prototype system run of a small Hadoop cluster for the regression tests and a stand alone PC for optimisation operations but it is anticipated that for a full industrial system this would be expanded to a much larger cloud or on-premise Hadoop installation. Such a system would allow for far more regression tests to take place simultaneously, in the experiments in this work we restricted our tests to one set of hyper-parameters per algorithm but in a full industrial system we would want to test far more than these. Although Python is a supported language for Big Data and Hadoop development it is not considered one of the faster languages and in a system tracking real-time events we would likely need to develop programs in either Java or more likely Scala as this is better supported under the current streaming environments such as Apache Spark Streaming, Apache Storm or Apache Flink. The framework as presented also left the storage of results and data to the discretion of the user, it would be necessary in a fully autonomous system to develop solutions for this. The work as it stands is fully scalable to whatever level we could allocate resources to, the main benefit from increased scale would be being able to check increasing numbers of algorithm configurations.

One of the strengths of the framework presented is that it is not tied to any particular language, architecture, data store or set of algorithms. It would be useful, long term, if this framework was to be offered to industrial partners as a template for developing autonomous machine learning system to develop a variety of case studies using different algorithms for the prediction / control channel (E.G. Try different classification algorithms rather than regression algorithms), different optimisation techniques and different data sets. This would allow us to have a suite of systems built around the framework presented here that would be available for any number of industrial processes, Figure 12.1 shows a potential architecture for an industrialised version of the framework, showing some potential tools that could be used for storage and development.

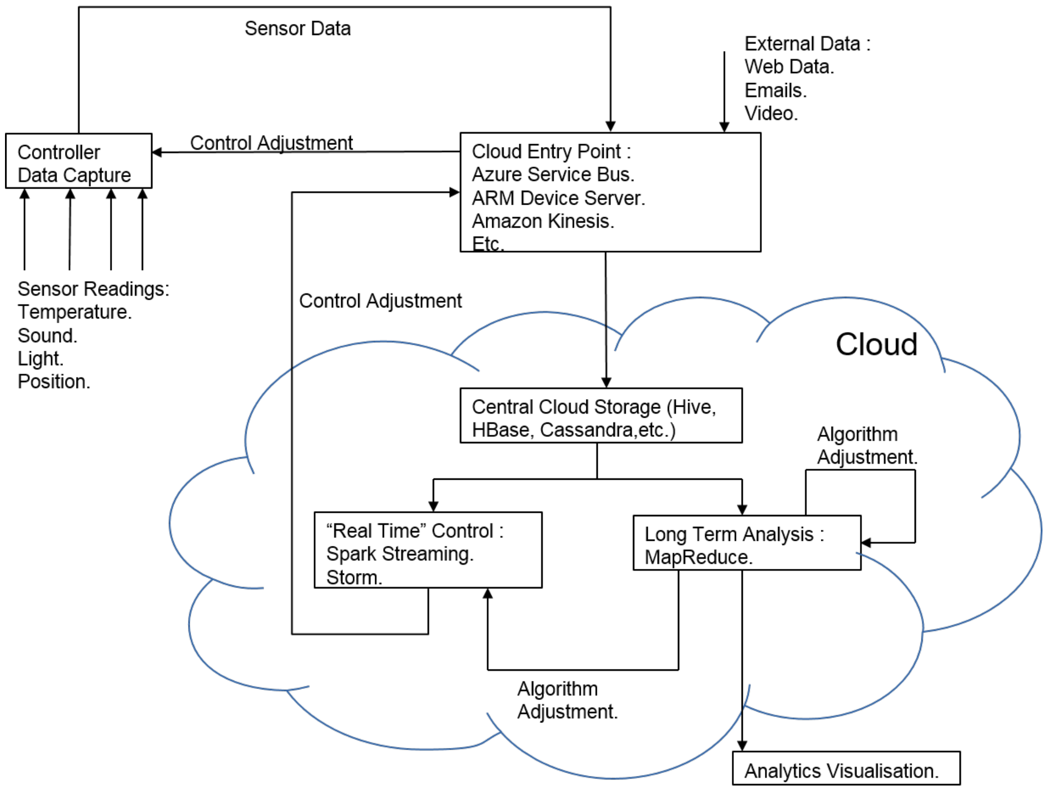
Sensor Data
External Data : Web Data. Emails. Video.
Controller Data Capture
Control Adjustment
Cloud Entry Point : Azure Service Bus. ARM Device Server. Amazon Kinesis. Etc.
Sensor Readings: Temperature. Sound. Light. Position.
Control Adjustment
Cloud
Central Cloud Storage (Hive, HBase, Cassandra,etc.)
Algorithm Adjustment.
"Real Time" Control : Spark Streaming. Storm.
Long Term Analysis : MapReduce.
Algorithm Adjustment.
Analytics Visualisation.

Figure 10.1: *Potential Industrialised Version of The Framework*

References.

Ai *et al.*, 2018 – Mingyao Ai, Jun Yu, Huiming Zhang, Hai Ying Wang, Optimal Subsampling Algorithms for Big Data Generalized Linear Models, arXiv:1806.06761v1 [stat.ME] 18th June 2018.

Abellan-Nebot and Subiron, 2010 - Jose Vicente Abellan-Nebot, Fernando Romero Subiron, A Review of Machining Monitoring Systems based on artificial intelligence process models, International Journal of Advanced Manufacturing Technologies (2010) 47:237-257

Alfaro-Cortes *et al.*, 2020 - E. Alfaro-Cortes, J. Alfaro-Navarro, M. Gamez, N. Garcia, Using random fores to interpret out-of-control signals,Acta Polytechnica Hungarica, 17 (6), 2020

Apache Software Foundation, 2010 - Apache Software Foundation, Apache Hadoop, https://hadoop.apache.org, 2010, accessed Oct. 2022

Apache Software Foundation, 2014 - Apache Software Foundation, Apache Mahout, https://mahout.apache.org, 2014, accessed Oct. 2022

Apache Software Foundation, 2018 - Apache Software Foundation, Apache Spark, https://spark.apache.org, 2018, accessed Oct. 2022

Arellano-Espitia *et al.*, 2020 - Francisco Arellano-Espitia, Miguel Delgado-

Prieto, Victor Martinez-Viol, Juan Jose Saucedo-Dorentes and Roque Afredo Osirnio-Rios, Deep-Learning Based Methodology for Fault Diagnosis in Electromechanical Systems, Sensors Vol. 20, Issue 14, 10.3390 , July 2020.

Bao *et al.*, 2012 - Yuan Bao, Lei Ren, Lin Zhang, Xuesong Zhang, Yongliang Luo, Massive Sensor Data Management Framework in Cloud Manufacturing Based on Hadoop. IEEE International Conference on Industrial Informatics (INDIN), 2012.

Bishop, 2006 - Christopher M. Bishop, Pattern Recogntion and Machine Learning, Springer 2006.

Caruana and Niculescu-Mizil – Rich Caruana and Alexandru Niculescu-Mizil, An Empirical Comparison of Supervised Learning Algorithms, Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh PA, 2006.

Choudhary, Harding and Tiwari, 2009 - A.K. Choudhary, J.A. Harding, M.K. Tiwari, Data Mining in Manufacturing: A Review Based on the Kind of Knowledge, Journal of Intelligent Manufacturing, 2009, 20:501-521.

Cloudera, 2022 - Cloudera Inc, www.cloudera.com, 2022., accessed Oct. 2022

Deng *et al.* ,2012 - Hong Deng, Ying-long Wang, Jun Yang, Liangqing Feng, Framework of Service-Oriented Manufacturing Based on Multi-Relational Data Stream Mining. International Conference on Computer Science and Service System, 2012.

Dominguez-Pumar *et al.*, 2016 – M. Dominguez-Pumar, L. Kowalski, R. Calavia, E Llobet, Smart control of chemical gas sensors for the reduction of their time response., Sensors and Actuators B: Chemical 229 (2106)

Drignei, Forest and Nychka, 2008 - Dorin Drignei, Chris E. Forest, Doug Nychka, Parameter Estimation for Computationally Intensive NonLinear Regression with an Application to Climate Modeling, The Annals of Applied Statistics, 2008 Vol. 2 No. 4

Feurer *et al.*, 2015 – Matthias Feurer, Jost Tobias Springenberg, Aaron Klein, Manuel Blum, Katharina Eggensperger, Frank Hutter, Efficient and robust automated machine learning, Advances in Neural Information Processing Systems 28 (NIPS 2015).

Fister *et al.*, 2013 - Iztok Fister, Iztok Fister Jr., Xin-She Yang, Janez Brest. A comprehensive review of firefly algorithms, Swarm and EVolutionary Computation 13 (2013) 24-46, Elesevier.

Fonollosa *et al.*, 2015 – Jordi Fonollosa, Sadique Sheik, Ramón Huerta, Santiago Marco, Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring, Sensors and Actuators B: Chemical 215 (2015) 618-529, Elsevier B.V.

Forbes, 2020 - Forbes Technology Council, 14 Smart Ways To Leverage Machine Learning For Small Business, Forbes Magazine, Sep 24 2020

ForePaas, 2021 - ForePaas (no author), A Machine Learning Practical Guide for SMEs, https://www.forepaas.com/en/blog/ml-practical-guide-for-smes/ , July 2021, accessed Oct. 2022.

Frazier, 2018 - Peter I. Frazier, A Tutorial on Bayesian Optimization, arXiv.1807.02811v1 [stat.ML], 10 July 2018.

Gama, 2012 - João Gama, Raquel Sebastião and Pedro Pereira Rodrigues, On evaluating stream learning algorithms, Machine Learning (2013) 90:317-346

Ghosh and Sanyal, 2016 - Indranil Ghosh, M.K. Sanyal, Machine Learning for Predictive Modeling in Management of Operations of EDM Equipment Product, 2016 Second International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)

Golovin *et al.*, 2017 – Daniel Golovin, Greg Kochanski, Benjamin Solnik, John Karro, Subhodeep Moitra, D. Sculley, Google Vizier : A service for black-box optimization, KDD 2017 Applied Data Science Paper, August 13-17, 2017.

Grömping, 2009 - Ulrike Grömping, Variable Importance Assessment in Regression: Linear Regression versus Random Forest, The American Statistician, 63:4 308-319, DOI 10.1198/tast.2009.08199.

Hastie, Tibshirani and Friedman, 2009 - Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning : Data Mining, Inference and Prediction 2nd Edition, Springer Series in Statistics, Springer 2009.

Hiskey, 2017 – Terri Hiskey, Industry Week, Preparing for Manufacturing's Future with Industry 4.0, www.industryweek.com/technology-and-iiot/article/22017916/preparing-for-manufacturings-future-with-industry-40 , 26/05/17, accessed 11/07/20.

Hsiao, Chiu and Lu, 2010 - Shih-Wen Hsiao, Fu-Yuan Chiu, Shu-Hong Lu, Product-form design model based on genetic algorithms, International Journal of Industrial Ergomomics 40(2010) 237-246

James *et al.*, 2017 - Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, An Introduction to Statistical Learning with Applications in R, Springer Texts in Statistics, Springer 2017.

Kaneko, 2017 - Hiromasa Kaneko, A new measure of regression model accuracy that considers applicability domains, Chemometrics and Intelligent

Systems 171 (2017) 1-8.

Kiran *et al.*, 2015 - Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, Sartaj Singh Baveja, Lambda Architecture for Cost-effective Batch and Speed Big Data processing. 2015 IEEE International Conference on Big Data.

Koshy, 2019 – Kevin Koshy, Capgemini, Why is Industry 4.0 important in manufacturing?, www.capgemini.com/gb-en/2019/06/why-is-industry-4-0-important -in-manufacturing/ , 13/06/19, accessed 11/07/20.

Liu, 2017 - Bing Liu, Lifelong machine learning: a paradigm for continuous learning, Front. Comput. Sci. 2017, 11(3): 359-361.

Loukides, 2022 - Mike Loukides, AI Adoption in the Enterprise 2022, O'Reilly Media, March 2022, https://www.oreilly.com/radar/ai-adoption-in-the-enterprise-2022 , accessed Oct. 2022

Loza, Cisneros and Arreola, 2017. - Jose Miguel Moran Loza, Marco Antonio Perez Cisneros, Alicia Garcia Arreola, Artificial Neural Networks vs Regression Techniques in the forecasting of contaminants in the Santiago River, based on the sample of a pollutant, through Data Fusion, 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon).

Marz and Warren 2015, - Nathan Marz James Warren, Big Data : Principles and Best Practices of Scalable Real-Time Data Systems, Manning Publications 2015.

Oliveria, 2020 - Samuel Oliveira (renard162), BeeColPy, http://github.com/renard162/BeeColPy , V1.1 uploaded 30th May 2020, accessed Oct., 2022

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

Pham *et al.*, 2006 - D.T.Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, M. Zaidi, The Bees Algorithm - A Novel Tool for Complex Optimisation Problems, Intelligent Production Machines and Systems: 2nd I*PROMS Virtual International Conference 3-14 July 2006, pp 454-459.

Ruvolo and Eaton - Paul Ruvolo, Eric Eaton, Active Task Selection for Lifelong Machine Learning, Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence.

Saritha and Sajimon, 2017 - Saritha K, Sajimon Abraham, Prediction with Partitioning: Big Data Analytics Using Regression Techiques, 2017

International Conference on Networks & Advances in Computational Technologies (NetACT).

Shallue *et al.*, 2018 - Christopher J. Shallue, Jaehoon Lee, Joe Antohnini, Jascha Sohl-Dickstein, Roy Frostig, George E. Dahl., Measuring the Effcets of Data Parallelism on Neural Network Training, arXiv:1811.03600v1 [cs.LG] 8 Nov 2018.

Sharif *et al.*, 2017 - Behzad Sharif, David Makowski, Finn Plauborg, Jørgen E. Oleson, Comparison of regressison techniques to predict response of oilseed rape yield to variation in climatic conditions in Denmark, European Journal of Agronomy 82(2017) 11-20

SME News, 2022 - SME News (no author), Is the Value od AI & Machine Learning Out of Reach for SMEs?, www.sme-news.co.uk/is-the-value-of-ai-machine-learning-out-of-reach-for-smes, Jan 2022, accessed Oct. 2022.

Silver, Yang and Li, 2013 - Daniel L. Silver, Qiang Yang and Lianghao Li, Lifelong Machine Learning Systems: Beyond Learning Algorithms, Lifelong Machine Learning: Papers from the 2013 AAAI Spring Symposium.

Thornton *et al.*, 2013 – Chris Thornton, Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Auto-WEKA : Combined selection and hyperparameter optimization of classification algorithms, KDD '13 Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and

Data Mining, August 2013 pp 847-855.

Torgo and Ribeiro, 2009 - Luis Torgo and Rita Ribeiro, Precision and Recall for Regression, Discovery Science DS 2009, Lecture Notes in Computer Science Vol 5808. Springer, Berlin, Heidelberg.

Twain OpenAI Club, 2021 - Twain OpenAI Club (OpenAIorg), 5Gcraft_FireflyAlgorithm, github.com/OpenAIorg/5Gcraft_FireflyAlgorithm, uploaded 17th June 2021., accessed Oct. 2022

UKTN, 2018 - Salavtore Minetti, SMEs and AI: Fortune favours the early adopters, UK Tech News, 13th April 2018.

Wang, 2007 - Keshend Wang, Applying Data Mining to Manufacturing: The Nature and Implications, Jounal of Intelligent Manufacturing 18:487-495, 2007

Yang, 2010 - X.-S. Yang,"Firefly algorithm, Lévy flights and global optimization", in: Research and Development in Intelligent Systems XXVI (EdsM. Bramer, R. Ellis, M. Petridis), Springer London, pp. 209-218 (2010).

Yang, Liu and Fu, 2010 - Xin Yue Yang, Zhen Liu, Yan Fu, MapReduce as a Programming Model for Association Rules Algorithm on Hadoop, Information Sciences and Interaction Sciences (ICIS) 2010, 3rd International Conference. Chengdu, China 23-25 June 2010. pp99-102

Yang, 2021 - Xin-She Yang, Particle Swarm Opimization, Nature-Inspired Optimization Algorithms 2nd Edition, Elsevier, 2021.

Yasutomi and Enoki, 2020 - A. Yasutomi, H. Enoki, Localization of inspection device along belt conveyors with multiple branches using deep neural networks, IEEE Robotics and Automation Letters, 5 (2) (2020)

Yuce *et al.*, 2013 - Baris Yuce, Michael S. Packianather, Ernesto Mastrocinque, Duc Truong Pham and Alfredo Lambiase, Honey Bees Inspired Optimization Method: The Bees Algorithm, Insects, 2013,4,pp 646-662

Zhao *et al.*, 2019 – Xiaojin Zhao, Zhihuang Wen, Xiaofang Pan, Wenbin Ye and Amine Bermak, Mixture Gases Classification Based on Multi-Label One-Dimensional Deep Convolutional Neural Network, IEEE Access 2019.2892754