Anomaly detection on single variate time series with concept drifts

P. P. B. Perera

Index No. : 15001008

Supervised by

Dr. M.I.E.Wickramasinghe

Submitted in partial fulfillment of the requirements of the
B.Sc. in Computer Science (Hons) Final Year Project in Computer Science
(SCS4124)

University of Colombo School of Computing
Sri Lanka
July 17, 2020

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate: P. P. B. Perera

.....
Signature of Candidate
July 17, 2020

This is to certify that this dissertation is based on the work of Mr. P. P. B. Perera under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Supervisor: Dr. M.I.E.Wickramasinghe

.....
Signature of Supervisor
July 17, 2020

Abstract

This study introduces an anomaly detection method on time series data. An ensemble LSTM CNN network with normalization and regularization of time series data incorporated with a concept drift adaptation technique is used for profiling the normal of a time series. Dynamic Time Warping(DTW) algorithm is used to generate warp distance between observed time window and profiler output to be used as a distance measure. Anomalies are detected when the distance between observed time window and profiler output exceeds a pre-defined threshold value. This study compares performance of few profiling methods on Numenta Anomaly Detection benchmark data set. Anomaly detection method implemented using ensemble LSTM CNN network with DTW introduced in this study performs better that baseline anomaly detection method implemented using ARMA with DTW.
Keywords: anomaly detection, time series, LSTM, CNN, ensemble networks, concept drift, dynamic time warping

Acknowledgement

I would like to express my sincere gratitude to my research supervisor, Dr. M.I.E.Wickramasinghe, senior lecturer of University of Colombo School of Computing for providing me continuous guidance and supervision throughout the research.

I would also like to extend my sincere gratitude to Dr. Kasun de Zoysa, senior lecturer of University of Colombo School of Computing and Dr. Kasun Gunawardana, senior lecturer of University of Colombo School of Computing for providing feedback on my research proposal and interim evaluation to improve my study and Dr. Kasun Jayathunga, research collaborator of University of Colombo School of Computing for providing guidance. I also take the opportunity to acknowledge the assistance provided by Dr. H E M H B Ekanayake as the final year computer science project coordinator.

Many thanks to my beloved mother and my dear father for always being my strength, showing me the correct direction, and making me who I am today. This thesis is dedicated to all my family members, to primary and secondary school teachers, to lecturers of University of Colombo School of Computing and to anyone who supported even by words to make my dreams come true.

Contents

List of Figures

List of Tables

Listings

Acronyms

LSTM	Long Short Term Memory cell network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DDM	Learning with Drift Detection
NAB	Numenta Anomaly Benchmark
MLP	Multilayer Perceptron
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TPR	True Postive Rate
PVV	Precision
F-1500	F 15000 score
LSTMCNNnet	Ensemble LSTM CNN neural network with time series data normalization and regularization
LSTMCNNcda	LSTMCNNnet combined with concept drift adaptation technique
LSTMCNNkeras	Ensemble LSTM CNN neural network
ARMA	Auto Regressive Moving Average

Chapter 1

Introduction

This research project is a further improvement of the research project "LSTM based Framework for Time Series Anomaly Detection" [1] done by N.A.A.H.Eranga under the supervision of Dr.M.I.E.Wickramasinghe.

Eranga has used ensemble Long Short Term Memory cell network)LSTM) Convolution Neural Network(CNN) with time series data normalizing and regularizing techniques to profile normal of a time series, which will be called LSTMCNNnet thenceforth. He has used LSTM to capture long term memory CNN to capture short term features. He has used Dynamic Time Warping(DTW) to generate warp distance between observed time window and profiler(LSTMCNNnet) output to detect anomalies when the warp distance reaches a predefined threshold.

In this study we have explored Eranga's [1] work and discovered that LSTMCNNnet does not adapt to many concept drifts due to it's time series data normalization and regularizing steps, although ensemble LSTM CNN does adapts to concept drifts. Since data normalization and regularizing steps are required to generalize the method introduced in Eranga's work we have incorporated a Concept Drift Adaptation method [2] to LSTCNNnet in order to fix the shortcomings of LSTMCNNnet. Our extended version of LSTMCNNnet will be called LSTMCNNcda henceforth.

Motivation Motivation for this research project was the promising results shown by LSTMCNNnet. Eranga states in his thesis [1]:

> "The previous detection accuracy obtained for the NAB is 75.2%.Using our model we were able to get an accuracy of 79.13% for the dataset. The detection rate of these anomalies depend on the threshold values used in DTW similarity value. By using this threshold we can further increase the accuracy but it will result in increase of the fault detection rate."

1.1 Background to the Research

1.1.1 Anomalies and Time Series

Anomalies

An anomaly is an observation that has not been observed before or not observed in normal conditions. Anomalies are also known as outliers, or novelty points. .Anomalies are divided into three main types in the literature [3], point anomalies, group anomalies, and contextual anomalies.

Time Series

A time series can be considered as a set of data points with a timestamp. With this simple definition of time series. We can classify anomalies that occur in time series as contextual anomalies [3]. Contextual anomaly cannot be identified by just observing a single point at a time. To identify a contextual anomaly we have to consider the context of the observation i.e the past data points from current observation.

According to Box et al. [4] "A time series is a sequence of observations taken sequentially in time". They mention that an intrinsic feature of a time series is that adjacent observation is dependant on previous observations, and time series analysis is about analyzing this dependence. Hyndman et al. [5] mentions a mathematical model of time series consisting three main components as shown in (1.1) where y_t is the observed value, S_t is the seasonal component, T_t is the trend-cycle component and R_t is the remainder component at time t. Trend-cycle consists of Trend and Cycle components as shown in (1.2). A time series is called "stationary" if it's statistical properties(mean, standard deviation and other metrics of observed values) are constant. If a time series is "stationary" we can guarantee that the Trend component of the time series is zero but vise-versa is not true [5].

$$y_t = S_t + T_t + R_t \tag{1.1}$$

$$T_t = Trend + Cycle \tag{1.2}$$

Another salient property of time series in the real world are concept drifts. Real concept drifts occur when the underlying features of a phenomenon that is under observation changes [6]. In practical applications in most scenarios a time series is generated using measurements from physical sensors. When such physical sensor deteriorates concept drifts may occur in the observed time series generated by sensor readings. For an example Oxygen density sensor in a combustion engine may deteriorate with time and sensor reading of normal Oxygen density level may change drastically with time. Figure 1.1 illustrates a concept drift in a time series.

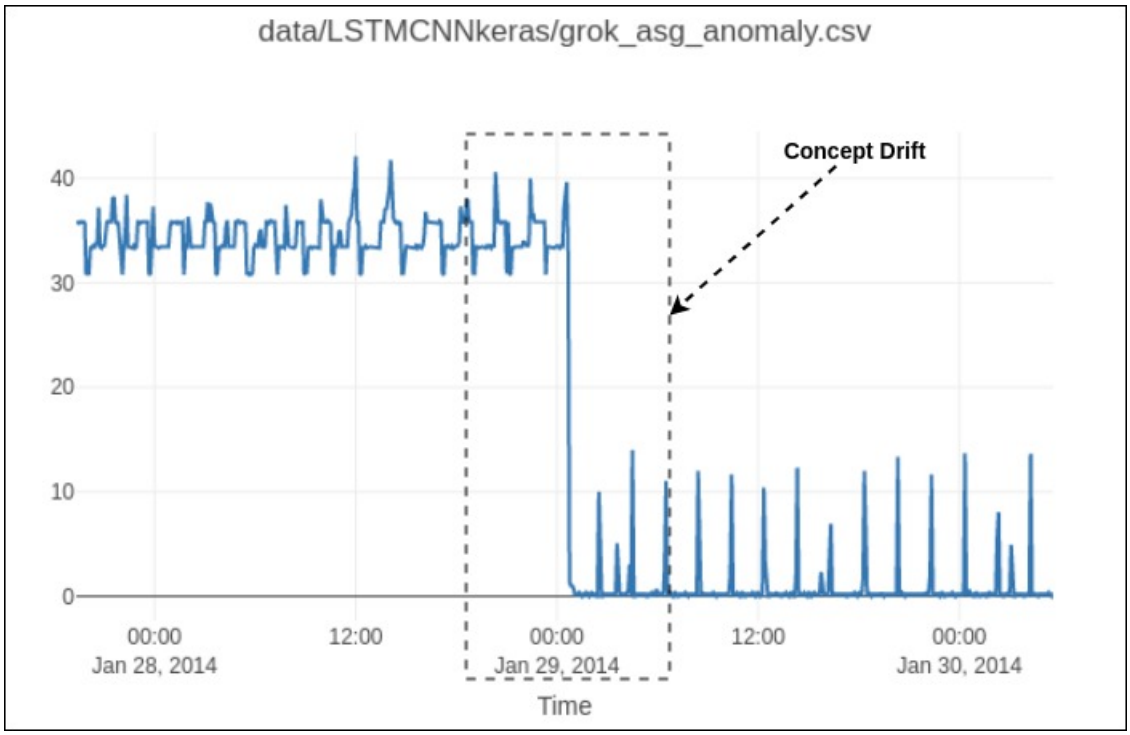

Figure 1.1: Time series plot from Numenta Data Set [7] with a concept drift.

1.1.2 Evaluation techniques for anomaly detection methods

Confusion matrix in the context of anomaly detection

Confusion matrix also know as error matrix is a widely used tabular method of evaluation used to evaluate classifiers. In the anomaly detection context anomaly detection methods classifies data points in to two classes as “anomaly” or “normal”. Confusion matrix provides four principle metrics which are shown in the Fig. 1.2. These are True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN). Lets assume that a anomaly detection method outputs “Positive” if a data point is an anomaly and “Negative” if a data point is normal. When a data point truly is an anomaly and the anomaly detection method outputs “Positive”, such output is considered as TP. If a data point is normal and the anomaly detection method outputs “Negative”, output is considered as TN. If a data point is an anomaly and the anomaly detection method outputs “Negative” it is considered as a FN. Finally if a data point is normal and the anomaly detection method outputs “Positive” output is considered as FP.

In the process of anomaly detection in time series, a manual examination part can be included. Once an automated anomaly detection method signals the presence of an anomaly a user can check weather it is truly an anomaly or not. But if there was a true anomaly and the automated anomaly detection method does not

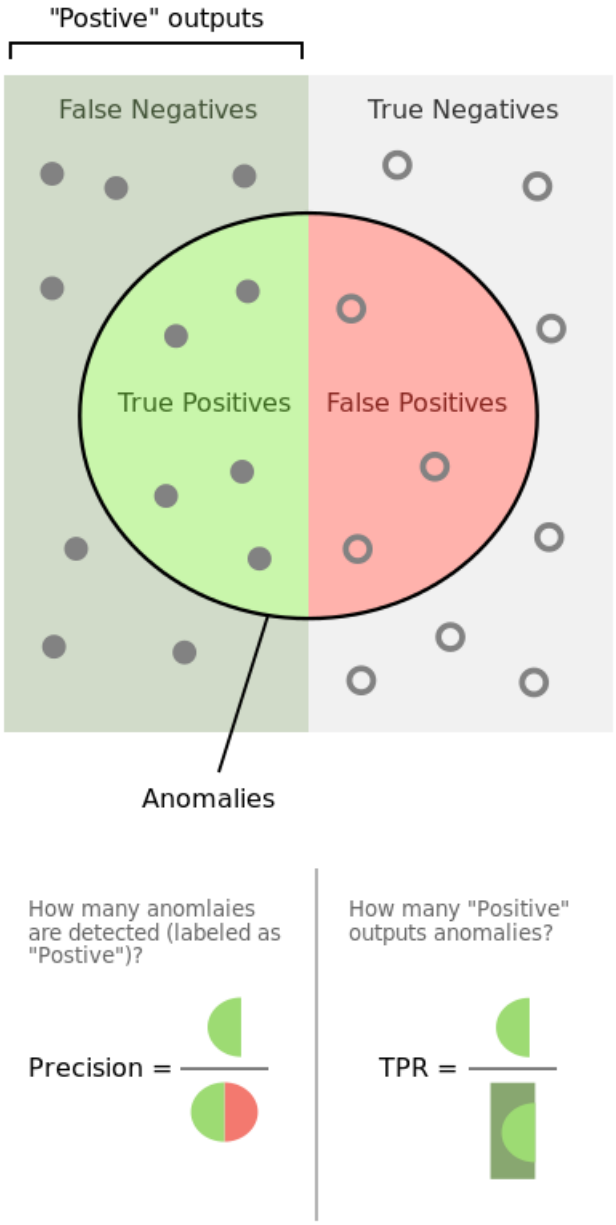

Figure 1.2: Illustration of confusion matrix metrics. Denotes the four principle metrics of confusion matrix True Positive, True Negative, False Positive and False Negative in the Venn diagram. Two derived metrics Precision(PVV) and True Positive Rate(TPR) is represented below the Venn diagram. This image is taken and modified from `en.wikipedia.org/wiki/Precision_and_recall`

signal any presence of an anomaly there might be severe consequences. Therefore it is important that an anomaly detection method detects almost all the anomalies in a data set. Metric used to measure this property is "TPR" or "Recall". Derivation of this metric is given in (1.3) and a pictorial representation of this metric is shown in Fig. 1.2.

$$Recall = \frac{True\ Positives}{All\ anomalies} = \frac{True\ Positives}{True\ Positives + False\ Negative} \tag{1.3}$$

But any anomaly detection method can achieve $TPR = 1$ i.e. achieve perfect Recall easily by classifying all the data points as "anomalies". Therefore to compare anomaly detection methods a metric that measures false alarm rate is suitable. Precision is a metric that is derived from principal metrics of the confusion matrix that measures the false alarm rate of an anomaly detection method. Equation 1.4 shows the derivation of Precision and Fig. 1.2 show Precision with respect to a Venn diagram of principle metrics of the confusion matrix.

$$Precision = \frac{True\ Positives}{\text{“}Positive\text{”}\ outputs} = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{1.4}$$

To combine Precision and Recall we can use the weighted harmonic mean of these two measures(harmonic mean is taken as Precision and Recall are measures of rates). $F_\beta - score$ is the harmonic mean of Precision and Recall. β is the weight associated with Recall, higher β value will give higher weight to Recall. Equation 1.5 represents the $F_\beta - score$.

$$F_\beta - score = \left(1 + \beta^2\right) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{1.5}$$

1.2 Research Problem and Research Questions

Research Aim Aim of this research is to solve the machine learning problem of anomaly detection.

Research Question Previous research done by Eranga [1] has a similar research aim to this research and as a result of their research LSTMCNNnet has been implemented. This research is proposed to cover the research gap of the LSTMCNNnet for some extend.

Time series profiling method LSTMCNNnet (ensemble LSTM, CNN network with time series data normalizing and regularization steps) fails to adapt to some concept drifts. Which will result in anomaly detection method generating a high

number of false alarms when there is a concept drift. This research project addresses this issue.

Main research problem of this research is "Can anomaly detection on time series method based on ensemble LSTM CNN network with DTW implemented in LSTMCNnet be improved using concept drift adaptation techniques ?".

Project objectives Objectives of this research project are aligned to archive the aim of this research by providing a solution for the research question. Below are the list of objective:

1. Understand the construct of concept drift in time series and explore the literature on time series concept drift. Understand the fundamentals of concept drift and types of concept drifts in time series.

2. Improve methodology used in LSTMCNNnet for anomaly detection in time series by considering concept drift in time series.

3. Implement a framework to develop algorithms for anomaly detection on time series.

1.3 Justification for the research

Anomaly detection has been used extensively in the past for intrusion detection, fraud detection, fault detection, system health monitoring, event detection in sensor networks, and detecting ecosystem disturbances [3]. There has been many software tools developed for anomaly detection such as SAS[1] , Rapid-Miner[2] , Oracle[3] , etc.

Anomalies are the first indication of a system failure in any system. Since modern society depends on may digital and analogue systems it is important to build and develop reliable systems. To build reliable systems there must be a mechanism to detect system failure before hand, to detect such failures anomaly detection is a must.

Disaster management systems use anomaly detection on time series sensor data to warn about disasters before they come. Resource management systems use novelty detection systems same as anomaly detection systems to prepare to dispatch resources which helps to maximize resource utilization. Finance and banking sector use anomaly detection to detect credit card fraud. Public surveillance systems use anomaly detection to detect public threats and to mitigate crimes. Social media

[1] `https://documentation.sas.com/?docsetId=vdmmlref&docsetTarget=p1paf478re5i4qn1aie1h4ycyi8a.htm&docsetVersion=8.3&locale=en`

[2] `https://rapidminer.com/resource/fraud-detection-prevention/`

[3] `https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/anomalies.htm`

networks and use anomaly detection to detect popularization of fake news. Cyber-security sector use anomaly detection to detect network attacks and to identify attempts on unauthorized access to networks and computer systems. Aeronautical industry use anomaly detection to detect mechanical faults on passenger air-crafts. Space exploration industry uses anomaly detection systems to predict critical failures in rocket launches and predict system failures in space shuttles. Manufacturing industry uses anomaly detection to predict mechanical failures that will yield to production halts. Anomaly detection is used in many applications. Many of the applications of anomaly detection are in technologies we use daily but they are not visible to us.

This research contribute to the computer science community studying anomaly detection on time series data by implementing a framework for developing algorithms for anomaly detection on time series data in C++ and by improving the time series anomaly detection method implemented in [1] LSTNCNNnet.

1.4 Methodology

We have explored anomaly detection on time series using ensemble LSTM CNN neural networks as profiling normal of a time series, Since LSTM can be used to recognize long term patterns in a time series [8] [9] [10] and CNN can be used to capture local features of a time series and effectively recognize and forecast shot term patterns [11]. By combining LSTM and CNN parallely taking the weighted sum of outputs we can optimize weights to yield better performance than just using LSTM or CNN. And we have used Dynamic Time Warping(DTW) [12] as the distance measure to get the distance between observed predefined time window and profile output to detect anomalies. Since DTW distance measure have less effect on stretching and warping of two time series, thereby giving an accurate interpretation of distance than distance measures like Euclidean when comparing time series data.

We have taken Eranga's [1] implementation of ensemble LSTM CNN network with time series data normalization and regularization steps for normal profiling and DTW distance measure for anomaly detection method(LSTMCNNnet) and explored his results on Numenta Anomaly Benchmark [7] data set. We have identified that LSTMCNNnet does not adapt to some concept drifts in time series, therefore LSTMCNNnet does not perform well when compared with baseline method of time series normal profiling method Auto Regressive and Moving Average(ARMA) [4] method. But ensemble LSTM CNN network without any time series data pre-processing and post-processing steps(LSTMCNNkeras) for time series normal pro-

filing performs better than ARMA method.

Since time series normalization and regularization is required to generalize the profiling method in LSTMCNNnet in order to overcome the issue of inability to adapt to some concept drifts in LSTMCNNnet we have combined LSTMCNNnet with a concept drift detection and adaptation method [2] (LSTMCNNcda). Finally results yielded by our experiments suggest that out method LSTMCNNcda improves the performance of LSTMCNNnet but does not reach the performance of ensemble LSTM CNN network without any time series data pre-processing and post-processing steps.

1.5 Outline of the Dissertation

Chapter 2 contains a detailed literature review on various methods of anomaly detection on time series, statistical methods, neural networks methods, etc. Chapter 3 contains details of design of LSTMCNNnet and design of Concept Drift Adaptation methods introduced in [2]. Chapter 4 contains details about a C++ framework we developed for anomaly detection on time series called "Sherlock" and implementation detains of various profiling methods with Sherlock framework for anomaly detection on time series data. Chapter 5 contains results of experiments we have carried out using profiling methods implemented on Sherlock framework and Numenta Anomaly Benchmark(NAB) data set. Chapter 6 contains the conclusion of this study from the observations of experiments carried out.

1.6 Delimitations of Scope

Scope of this project is to improving LSTMCNNnet with concept drift adaption method to perform better on time series with gradual and rapid concept drifts, but not with chaotic or long-term recurrent concept drifts.

1.7 Conclusion

This chapter laid the foundations for the dissertation. It introduced the research aim, research problem, and objectives if this research. Then the research was justified, the methodology was briefly described and justified, the dissertation was outlined, and the limitations were given. On these foundations, the dissertation can proceed with a detailed description of the research.

Chapter 2

Literature Review

In the modern context data is abundant and collection of data is growing in an exponential rate. Presence of abundant data enables one to use machine learning for anomaly detection. Sensor network data, real time market data and network activity data are some examples of time series data that is collected in an exponential rate, with the growing complexity of modern society.

Fox [13] has introduced additive and innovation (type I and type II) outliers in 1972. Additive (type I) outliers are global outliers which are point anomalies, innovation (type II) outliers are conditional outliers which are context anomalies. According to Guta et al. [3] after introduction of point and contextual anomalies by Fox [13] have been introduced many prediction model based anomaly detection methods on time series data. Prediction model based anomaly detection methods on time series data consists of two main components, a prediction model and a detection model. Prediction model predicts the next data point of a sequence of data points and the detection model compares the predicted data point and the real data point to detect an anomalous data point.

Section 2.1 discusses about prediction models based on Auto-regression in statistics. Section 2.2 discuses use of one class support vector machines in anomaly detection on time series by transforming time series into phase space. Section 2.3 discusses use of Neural Networks in anomaly detection on time series data and time series forecasting. Section 2.4 discuses about use of Hierarchical Temporal Memory(HTM) in anomaly detection.

2.1 Statistical methods

A lot of work has been carried out in modeling stationary signals in statistics [4],a signal can be interpreted as a time series. These statistical models with distance measures have been used to detect anomalies on time series data [13]. These mod-

els include Auto Regressive Moving Average(ARMA). Integrated Auto Regressive Moving Average(ARIMA), which is an extension of ARMA for non-stationary time series data. Integrated Auto Regressive Moving Average with Exogenous variables(ARIMAX), which is an extension of ARIMA with support to model effects of exogenous variables on a non-stationary process.

Concept of stationarity was briefly described in Section 1.1.1. Differancing is the process of obtaining difference between consecutive observations. By differencing a non-stationary time series we can obtain a stationary time series. For an example time series obtained by recording velocity of a moving object under constant gravity is non-stationary, we can obtain a stationary time series by taking the difference of velocity under constant gravity which is acceleration of the object and it is equal to constant gravity.

Auto Regression(AR) models are used for forecasting stationary time series data. Equation 2.1 describes AR model of order p.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t \tag{2.1}$$

where:

y_t is Time point at time t

c is Bias

ϕ_p is Weight given to time point $t - p$

ε_t is White noise term

Moving average(MA) model uses past forecast error in a regression-like model rather than using regression on past value of the forecast variable to model a time series [4]. Equation 2.2 describes MA model of order q.

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q} \tag{2.2}$$

where:

y_t is Time point at time t

c is Bias

ε_t is White noise at time t

θ_q is Weight given to white noise at $t - q$

When AR and MA is combined with weights for each component we obtain the Auto Regressive Moving Average(ARMA) model. Integrated Auto Regressive Moving Average(ARIMA) is obtained by taking the difference of a non-stationary time series to form a stationary time series and applying ARMA to model that series [4]. In ARIMA "integration" is the reverse of difference . Equation 2.3 describes ARIMA(p,d,q) model where p is the order of the Auto-regressive part, d is the degree of first difference involved and q is the order of the Moving average part. ARIMA models can be applied for non-stationary time series.

$$y'_t = c + \phi_1 y'_{t-1} + ... + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + ... + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{2.3}$$

where:

y'_t is Time point at time t of the lagged series of y_t

c is Bias

ϕ_1 is Weight given to lagged time point $t-1$

θ_1 is Weight given to white noise at $t-1$

ε_t is White noise at time t

Box and Tiao in 1981 [14] proposed an extension of the ARIMA model to incorporate several time series or an "explanatory variable" to forecast one time series. Which will add another compound term an "explanatory variable" to the ARIMA model. Motivation of such model is to capture the dependency of a time series on another time series. For example ambient temperature might depend on wind speed. Figure 2.1 illustrates how AR and MA model is combined and created ARIMA model and how ARIMA model is extended to ARIMAX model.

2.2 One Class Support Vector Machines

J. Ma and S. Perkins [15] has proposed a novel algorithm for time series novelty detection based on one-class support vector machines(SVMs). They have chunked a time series into time windows and transformed them to the phase space and applied One Class Support Vector machine to classify the vectors transformed into phase space as anomalous or normal. They have tested their method on synthetic time series (sinusoidal signals with additive noise) and on Santa Fe Institute Competition (SFIC)[1] data set. They have concluded that their method has promising results on those data sets.

[1] `https://physionet.org/content/santa-fe/1.0.0/`

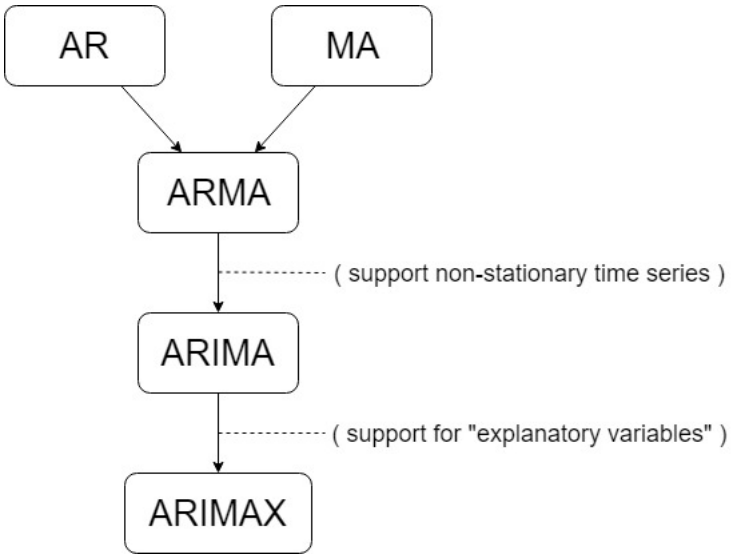

Figure 2.1: Illustrates the progression of Auto Regressive models.

2.3 Neural Network Methods

Most anomaly detection methods use a predetermined time window in a time series to compare features with a normal time window. Generally, the performance of these methods depend on predetermined time window parameter. Vig and Agarwal [8] has proposed Long Short Term Memory(LSTM) Networks to model non-stationary time series for anomaly detection. LSTM allows long term memory storage therefore long term correlations in a sequence can be learned [16]. Therefore LSTM solves the issue of depending on predefined time window in traditional methods. In their study they have used a stacked LSTM model to model complex time series which may consist of more than one underlying process generating a non-stationary time series.

They have shown that stacked LSTM models can capture features of a complex time series at different scales, one LSTM layer captures features at a weekly time-scale and another LSTM layer captures features at a more refined time-scale.

They has used stacked LSTMs to model the normal behavior(time series) and taken the instances that vastly deviates from the model as anomalies. Their stacked LSTM neural network predicts several time steps to ensure the network capture the temporal structure of the sequence.

L. Bontemps et al [10] has used LSTM for detecting collective anomalies in time series data. D. T. Shipmon et al [9] has compared Deep Neural Networks(DNN), Recurrent Neural Networks(RNN), LSTM and a Fourier model in anomaly detection on time series data and concluded that Fourier and RNN were more effective forecasting models and DNN and LSTM produce more false alarms at peaks of time series.

2.3.1 Recurrent neural networks

Goodfellow et al. in their book [17] "Deep Learning" mansions that Recurrent Neural Networks(RNN) [18] are a category of neural networks for processing sequential data, like Convolutional Neural Networks(CNN) are a category of neural networks specialized for processing a grid of values.

Graves in his book [19] mentions that if cyclical connections are allowed in a Multi-layer Perceptron(MLP), Recurrent Neural Networks(RNN) can be obtained. Even though this difference between MLP and RNN is trivial, MLP can only map its inputs to outputs but RNN can map its entire history of inputs to its outputs in principle. RNN allows a "memeory" of previous inputs to be kept in the internal state of the network which enables the previous inputs to influence the current output of the network. Figure 2.2 illustrates the structure of an RNN.

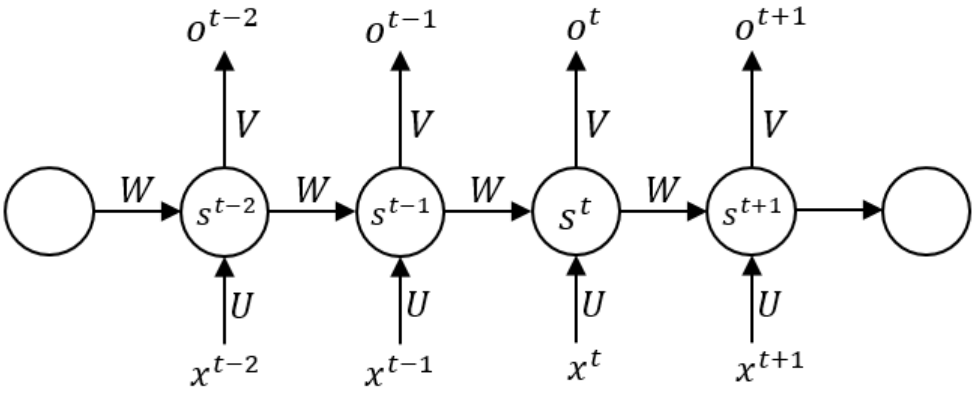

Figure 2.2: Illustrates the structure of an RNN. x^t is the input at time t, o^t is the output at time t and s^t is the hidden state at time t of the RNN. U, V and W are weight matrix of the corresponding edges of the network as shown in the figure. Image taken from [20]

Graves in his book [19] also says that important benefit of RNN is that their ability to use contextual information in their mapping. But the range of context that can be stored in the internal state of the network is limited. Since an input is cycled around the network from its recurrent connections there could be an exponential decay or exploding gradient problem which affects the internal state(Hidden Layer) of the network. Figure 2.3 illustrates the issue of vanishing gradients in RNNs and how it affects the output of the network.

Graves [19] says Long Short Term Memory(LSTM) Cell neural networks consists of recurrently connected sub-nets called memory blocks. Each block can contain one or more self connected memory cell having three multiplicative units: input, output and forget gates. Which is analogous to write, read, and rest operations

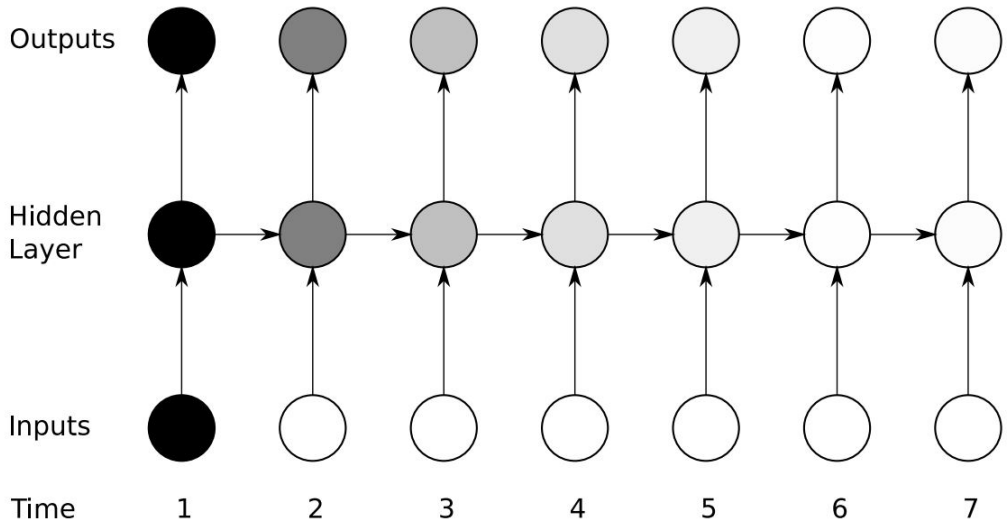

Figure 2.3: Illustrates how the effect of Inputs at $Time = 1$ vanishes from the Hidden Layer and the Outputs as time progresses. Image taken from [19]

in a memory chip of a digital computer. Multiplicative gates in the LSTM cell allows access to information stored over long period of time without the problem of vanishing or exploding gradients in RNN. For e.g if the multiplicative input gate is closed(activation is near zero) then the internal state is not overwritten by the new inputs, and information on internal state can be preserved for a longer period. And the influence of information on internal state on current output can be controlled by the activation value on the output gate. Forget gates activation value can be used to mutate or deviate the information in the internal state without the intervention of input. Figure 2.4 Illustrates the structure of a LSTM cell. Figure 2.5 illustrates how vanishing and exploding gradient issues is addressed by LSTM cells.

2.3.2 Convolutional Neural Networks

Goodfellow et al. in their book [17] "Deep Learning" says Convolutional Neural Networks(CNN) are a category of neural networks used to process data that has a grid-like topology. For an e.g time series, which is one dimensional grit taking data points at regular time intervals, and an image with 2 dimensional grid of pixels.

Convolutional neural networks use the convolution with respect to image processing. Kim and Casper [21] says "Convolution can be intuitively described as a function that is the integral or summation of two component functions". For an e.g taking two single variable function convolution at a point of these two functions are the product of values at that point of the two functions. Figure 2.6 illustrates convolution of two functions in single dimension.

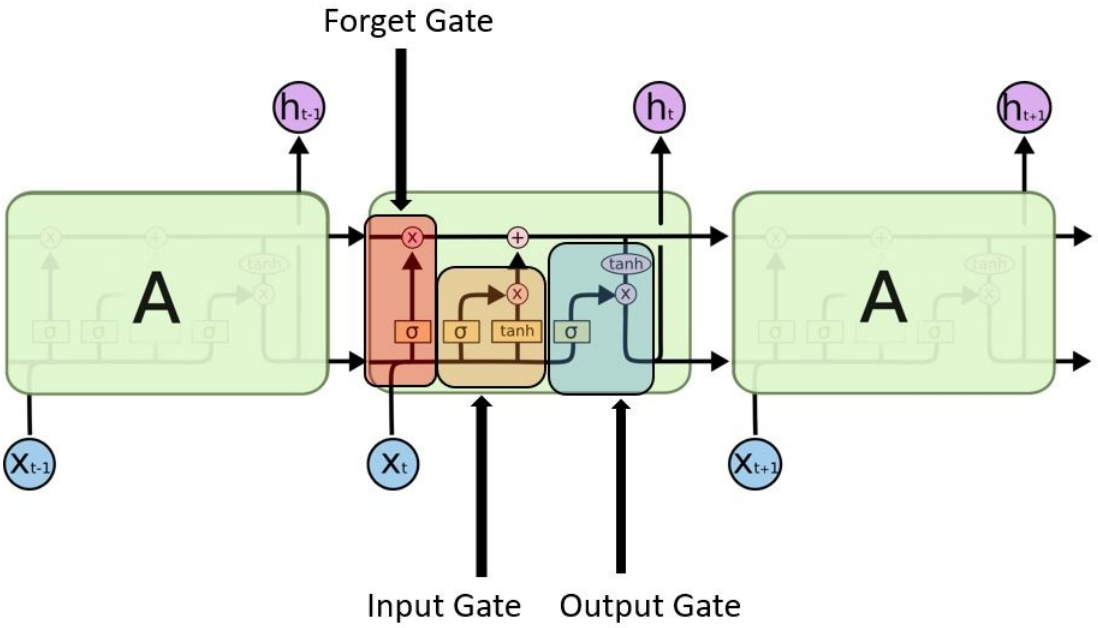

Figure 2.4: Illustrates the structure of a LSTM cell, horizontal axis represents the time dimension of the same LSTM cell. Image taken from `hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4`

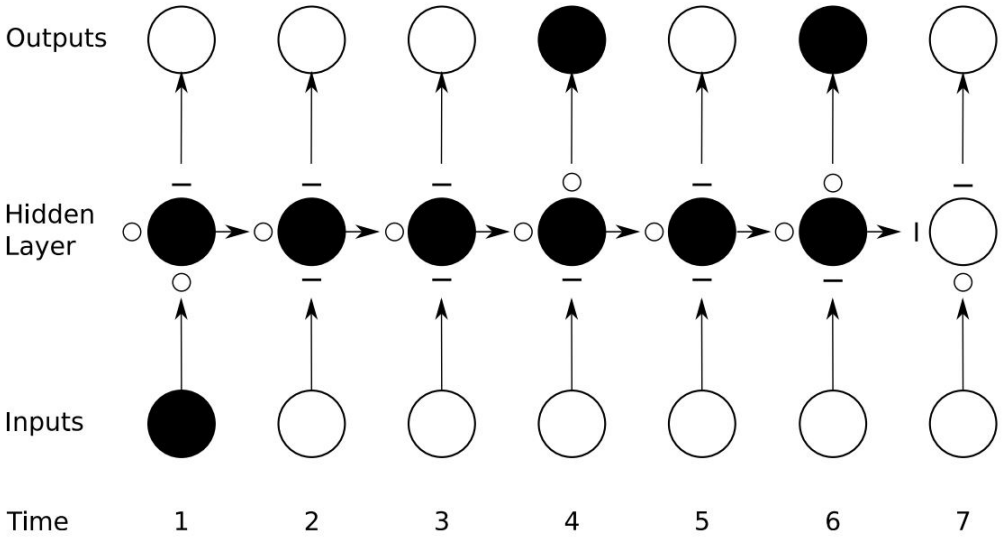

Figure 2.5: Illustrates how the hidden state in the Hidden Layer is controlled by input and forget gates and how the effect of hidden state on output is controlled by the output gate. Image taken from [19]

Zheng et al. [11] has used CNN to classify time series data in their work. In which they have used CNN as a feature extraction layer on time series and a Multi-layer Perception(MLP) layer for classification of time series windows.

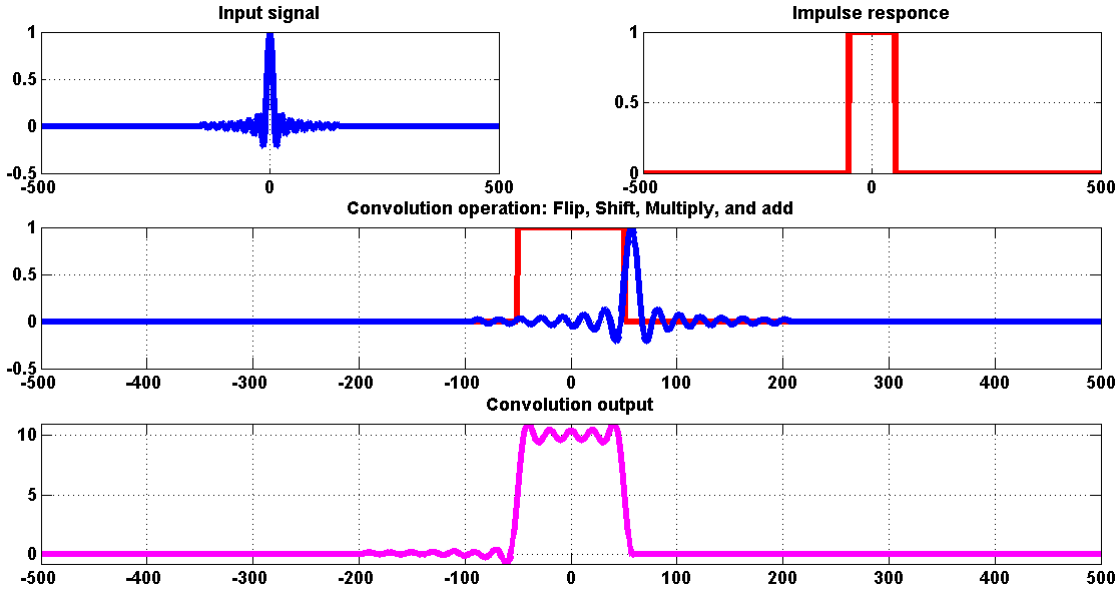

Figure 2.6: Illustrates an Input signal and an Impulse response under convolution operator.image taken from `www.mathworks.com`

2.3.3 LSTM in modeling time series

In anomaly detection on time series by comparing the features of normal behavior with observed time series features, it is important to capture the features of the normal behavior with a high accuracy. Laptev, N. et al. [22] has used LSTM networks with auto-encoders for feature extraction for accurate time series forecasting during high variance segments in a time series.

They have proposed a novel end-to-end recurrent neural network architecture that outperforms the current state of the art event forecasting methods on Uber data and generalizes well to a public M3 data set[2] used for time-series forecasting competitions. Their architecture leverages an auto-encoder for feature extraction. They mention that this problem is challenging because extreme event prediction depends on numerous external factors that can include weather, city population growth or marketing changes (e.g., driver incentives). They have shown that a vanilla LSTM model perform worse than their model.

They have concluded that a single generic neural network model is capable of producing high-quality forecasts for heterogeneous time-series relative to specialized classical time-series models.

[2] `https://forecasters.org/resources/time-series-data/m3-competition/`

F. Gers et al [23] has used LSTM network as an auto-regressive model for forecasting time series and compared it with multi-layer perceptron(MLP) model. They have observed that MLP yields better performance and suggested that LSTM's core strength is to learn and remember single events for long periods.

2.4 Hierarchical Temporal Memory in anomaly detection

Hierarchical Temporal Memory(HTM) is an online machine learning method that was inspired by the structure of human brain. Ahmad et al. [24] has used HTM on anomaly detection on time series data. Ahmad et al. [24] says that Hierarchical Temporal Memory (HTM) structure is based on know properties of cortical neurons in human brains. HTM has been used for sequence learning [25]. HTM has shown good results for prediction tasks [26, 27]. HTM models the spatiotemporal characteristics of the input sequence which can be used to forecast a time series.

Ahmad et al. [24] has used HTM as a prediction model in their work. They have used a probabilistic model of prediction error and they have detected anomalies using likelihood function and a predefined threshold value on the probabilistic model of prediction error. Figure 2.7 Illustrate the structure of HTM.

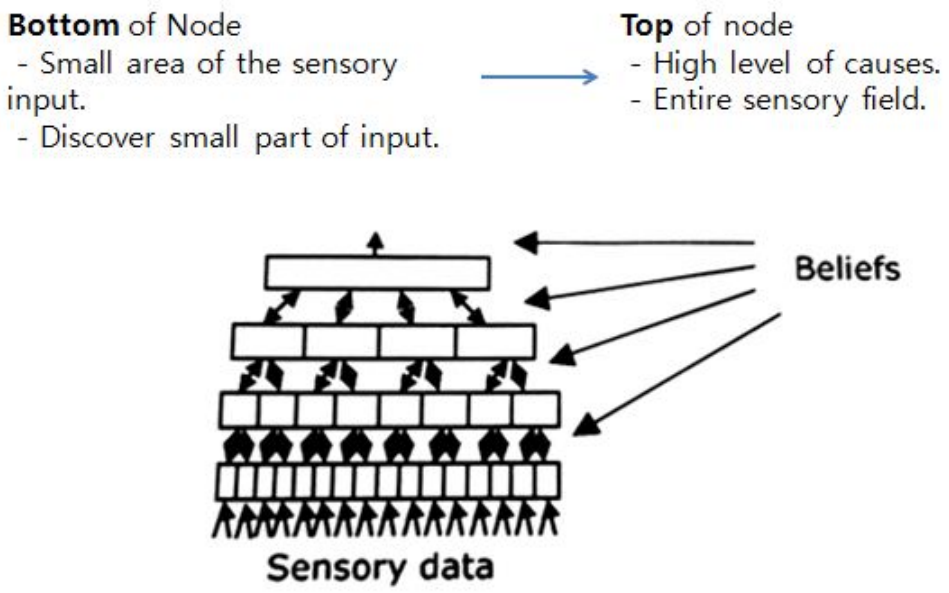

Figure 2.7: Illustrates Hierarchical Temporal Memory(HTM). Image taken from `ecl.unist.ac.kr/2010/09/htmhierarchical\_temporal\_memory`

2.5 Conclusion

It is important to notice that most anomaly detection methods on time series only observe a short time window to detect anomalies, but it is vital to have a long term memory of the patterns observed in a time series to identify some anomalies.

[10] [11] [8] [9] have explored using LSTM in their work for anomaly detection and shown mostly positive results. F. Gers et al [23] has suggested that LSTM's core strength is to learn and remember single events for long periods.

CNN has been used by Y. Zheng [11] for time series classification, since CNN captures short term features of a time series.

It is also important to notice that since most times series observed in practice undergo concept drifts [6] [28] it is vial for an anomaly detection method to differentiate concept drifts from anomalies and adapt to such concept drifts. And there have been many studies on identifying concept drifts on time series [29] [2].

Chapter 3

Design

A common way of anomaly detection in time series is profiling normal of a time series and comparing the profiled normal with observer data points with a distance measure and if this distance reaches a certain threshold detector outputs a positive detection [3].

In the methodology proposed in this study we have used an ensemble LSTM and CNN network with time series data normalization and regularization steps combined with concept drift adaptation technique [2](LSTMCNNcda) for profiling normal of time series and fast Dynamic Time Warping(fastDTW) [30] for measuring the distance between profile normal time series and observed data points. This anomaly detection method is explored because of the promising results shown by Eranga in his thesis [1].

It was observed that when exploring Eranga's [1] implementation of ensemble LSTM and CNN neural network with time series data normalization and regularization steps for profiling method that it does not adapt to concept drifts in time series. It was identified this inability to adapt to concept drifts occur due to the time series data normalization and regularization steps in his implementation of ensemble LSTM and CNN neural network. But ensemble LSTM and CNN neural network without these time series data normalization and regularization steps do adapt to concept drifts as observed in our experiments. But these time series data normalization and regularization steps used by Eranga [1] has to be used to ensure LSTM and CNN neural network performs well in general. Since Sigmoid activation functions used in LSTM CNN ensemble network will yield unexpected results without data normalization in some scenarios.

As a solution for this issue of inability to adapt for concept drifts a concept drift detection and adaptation method is introduced to the profiling method implemented by Eranga [1].

3.1 LSTMCNNnet

When profiling a time series it is important to capture features of long term and short term patterns of a time series. Recurrent Neural Networks(RNN) has been used in the literature [8] [22] [10] [23] to capture features of a series of data points. Since training RNN with back-propagation leads to diminishing and exploding gradients issue Long Short Term Memory(LSTM) Cell was introduced by S. Hochreiter and J. Schmidhuber [16]. LSTM Cell consists of a hidden state as in RNN and updating and forgetting the memory in the hidden state is controlled by gates.

Y. Zheng et al [11] has used Multi-channel deep CNN to predict multivariate time series. LSTM and CNN ensemble neural networks prediction model has been used by T. N. Sainath et al [31] for various vocabulary tasks and T. Lin et al [32] has used LSTM and CNN hybrid neural network to predict trend and slope of time series data to forecast time series in their forecasting model TreNet.

In Eranga's implementation(LSTMCNNnet) he has used LSTM and CNN networks combined in parallel as illustrated in Fig. 3.1 using a feature combination layer. Feature combination layer combines LSTM output and CNN output by taking the weighted sum of outputs with predefined weights as parameters.

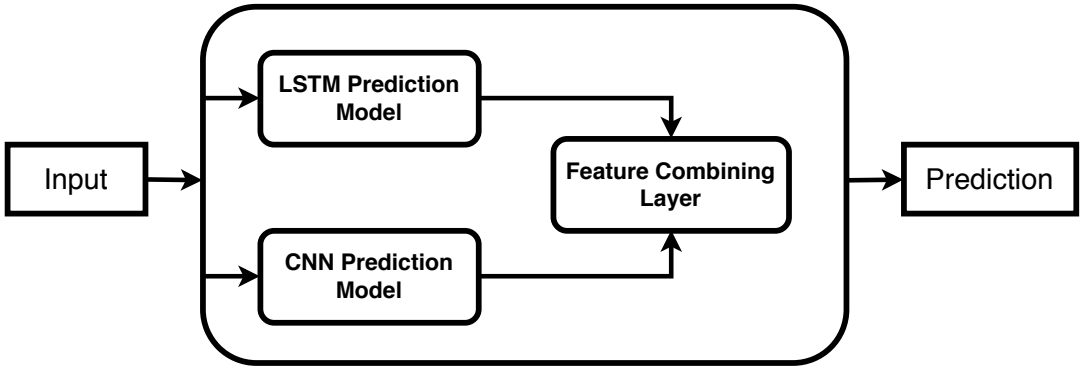

Figure 3.1: LSTM CNN ensemble architecture with feature combination layer taken from Eranga's thesis [1]

3.1.1 LSTM Implementation

In LSTM network implementation a single LSTM layer is present containing memory blocks. Each memory block contains a single memory cell. The structure of the memory cell is shown in Fig. 3.2 where Sigmoid activation function is used for input and output gates and Tanh activation function is used for forget gate. Input for the memory cell is a vector containing input at time (t) and output from cell at time (t-1). Current cell state is affected by the current input values and previous

output of the cell and the output is affected by the current cell state. Intuition behind having a cell state is to preserve long term memory.

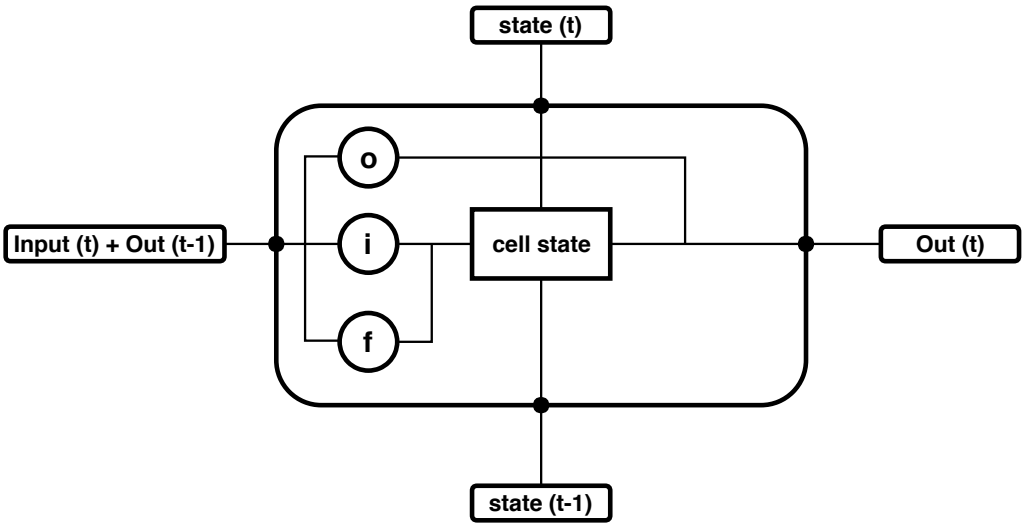

Figure 3.2: LSTM cell structure taken from Eranga's thesis [1]

Figure 3.3 illustrates the structure of LSTM neural network. Input layer is fully connected to LSTM layer and output is given by a fully connected layer. Number of memory blocks in LSTM layer, Learning rate, and the number of time steps unfolded can be adjusted as parameters.

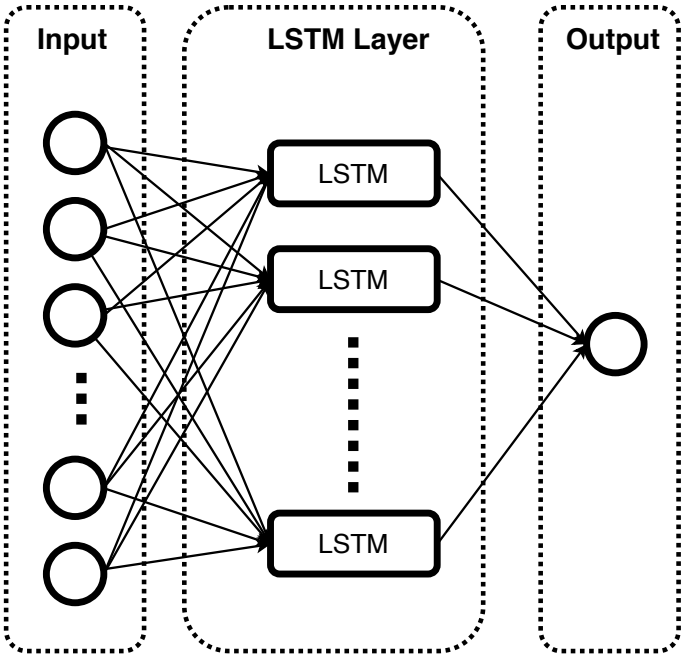

Figure 3.3: LSTM network structure taken from Eranga's thesis [1]

For the LSTM neural network time series is pre-processed by taking the input time window as a vector and normalizing it to produce a unit vector by dividing

the vector from its magnitude. And output is regularized by multiplying it by the previous calculated magnitude. This normalization and regularization steps where added since the activation functions used in the network (Sigmoid function) yields better results for normalized inputs since Sigmoid function will map values greater than 4 to 1 and lower than -4 to 0.

Equation (3.3) shows the normalization step, and (3.4) shows the regularized step.

$$input\ vector(v) = (t_1, t_2, t_3, t_4) \tag{3.1}$$

$$magnitude(m) = \sqrt{t_1^2, t_2^2, t_3^2, t_4^2} \tag{3.2}$$

$$normalized\ input = (\frac{t_1}{m}, \frac{t_2}{m}, \frac{t_3}{m}, \frac{t_4}{m}) \tag{3.3}$$

$$regularized\ output = output \times m \tag{3.4}$$

3.1.2 CNN implementation

CNN implementation contains an input layer, convolution layer, pooling layer and fully connected layers as illustrated in Fig. 3.4. Convulsion and pooling layers together extracts features from the time series. These extracted features are fed into a deep neural network with fully connected layers to learn patterns in the time series.

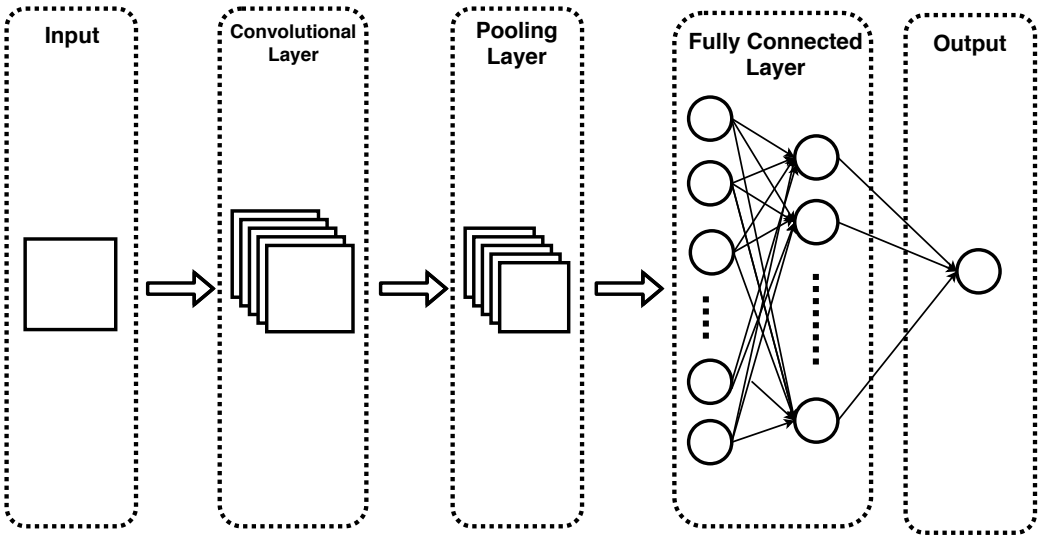

Figure 3.4: CNN structure taken from Eranga's thesis [1]

In the convolution layer strides and filter size, number of filters can be adjusted as parameters. Input matrix height and width of convolution layer can be adjusted as parameters. Time series window size will be taken as the product of input

matrix height and width. Pooling layer height and width can be adjusted as parameters. Number of fully connected layers and size in the deep neural network can be adjusted as parameters. Learning rate of the CNN can adjusted as a parameter.

Time series data is pre-processed as in the LSTM network. Data points from input time window are taken as a vector and normalized to a unit vector by dividing its magnitude (3.3). Post processing is done by min-max feature scaling rather than multiplying the output by previous magnitude. Equation 3.5 shows how the post processing is done on output.

$$O_F = (O - O_{max}) \times \left[\frac{R_{max} - R_{min}}{O_{max} - O_{min}}\right] + R_{min} \tag{3.5}$$

where

O_F is final output

O is current output

O_{min} minimum output

O_{max} maximum output

R_{min} minimum observed value

R_{max} maximum observed value

3.2 Dynamic Time Warping distance measure

Salvador, S. and Chan, P. [30] has introduced an extension of Dynamic Time Warping(DTW) [12] distance measure, they have named their algorithm as fastDTW. FastDTW is an approximation DTW. DTW has a quadratic time and space complexity and fastDTW has a linear time and space complexity. Dynamic time warping (DTW) finds an optimal alignment between two time series in which one time series may be "warped" non-linearly by stretching or shrinking its time axis. This alignment can be used to determine similarity of two time series(where stretching and shrinking will have minimum effect). Similarity is taken as the warp path distance which is a scalar value. FastDTW initially samples down a time series to a low resolution. Warp path found on low resolution is projected to a higher resolution and new warp path is searched only in the projected warp path found in the lower resolution. FastDTW keeps projecting and searching for a new warp path until the time series in the original resolution. Figure 3.5 shows how projection and searching is done.

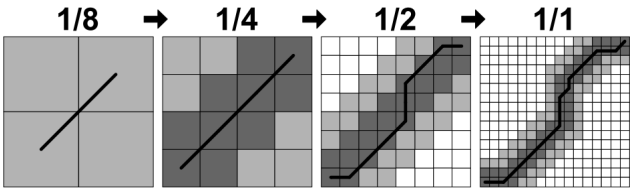

Figure 3.5: Projection and searching process of fastDTW algorithm taken from [30]

3.3 LSTMCNNcda

LSTMCNNcda is an extension of LSTMCNNnet which incorporates the concept drift adaptation method [2].

3.3.1 Concept Drift Detection

Webb et al [6] has introduced formal quantitative analysis of concept drift in the context of time series. This analysis can be applied to different contexts other than time series as well. For classification learning problem on a data stream we can model a process generating a stream of data points as a random variable Z. An instance z can be drawn from Z, z represents a pair (x, y). Where x, y are instances of X, Y variables. X models the random variable over vectors of attribute values of data points. Y models the random variable over class labels. Random variable Z can be modeled as the joint distribution of XY.

A machine learning algorithm takes training data as input to model a process(function or mapping). In classification problems, $P(Y)$ denotes the prior probability distribution over class labels, $P(X)$ denotes the prior probability distribution over data points. $P(X, Y)$ denotes the joint probability distribution over data points and class labels. $P(Y \mid X)$ denotes the probability distribution of observing data point X given that class label is Y and $P(X \mid Y)$ denotes the probability of data point being labeled as Y given that X data point is observed. A data stream is a data set in which the data elements have time stamps. Therefore in a series, in order to reference the probability distribution at a particular time we can add a time subscript such as $P_t(Z)$, to denote a probability distribution at time t.

3.3.2 Learning with drift detection

Gama, J. et al [29] has compared eight concept drift detection methods used on time series. They have chosen the concept drift detection methods in the literature with highest number of citations, provided that there is a freely available implementation or a detailed algorithm is given in the literature. They have concluded

that Learning with drift detection(DDM) by Gama, J et al [2] is the best method.

Gama, J. et al [2] "Learning with drift detection" paper introduce a concept drift detection method for gradual and abrupt concept drifts. Their method can be used to control the error rate of an online learning algorithm. Their method will be referred to as DDM henceforth.

In an online learning scenario data points are presented as a sequence. They have defined context as a set of data points where $P(X \mid Y)$ probability distribution is stationary. Statistical theory guarantees that while the distribution is stationary, the error of classification will decrease as the number of data points with class labels increase. When the distribution changes, the error will increase. For a context DDM defines a warning level, and a drift level. A new context is declared, if the classification error increases reaching the warning level at data point T_W, and the drift level at data point T_D. Online learning algorithm learns a new model using only the data points from T_W (new concept).

For an example Suppose a sequence of data points, in the form of pairs (x_i, y_i). For each data point, the actual decision model predicts (z_i) , that can be True $(z_i = y_i)$ or False $(z_i \neq y_i)$. For a set of data points the error is a random variable from Bernoulli trials. The Binomial distribution gives the general form of the probability for the random variable that represents the number of errors in a sample of n examples.

For each data point i in the sequence, the error-rate is the probability of observing False$(z_i \neq y_i)$ is P_i , with standard deviation given by $s_i = \sqrt{P_i(1 - P_i)/i}$. Binomial distribution is closely approximated by a Normal distribution with the same mean and variance(when $n \geq 30$). When Considering that the probability distribution is unchanged when the context is static, then the $1 - \alpha/2$ confidence interval for P is approximately $P_i \pm \alpha \times s_i$. The parameter α depends on the confidence level.

DDM keeps two registers during the training of the learning algorithm, P_{min} and s_{min}. Every time a new example i is processed those values are updated when $P_i + s_i$ is lower than $P_{min} + s_{min}$.

Warning level defines the optimal size of the context window. The context window will contain the data points that are on the new context and a minimal number of data points on the old context.

Suppose that there is a data point i with correspondent P_i and s_i . Confidence level for warning has been set to 95%, that is, the warning level is reached if $P_i + s_i \geq P_{min} + 2 \times s_{min}$. The confidence level for drift has been set to 99%, that is, the drift level is reached if $P_i + s_i \geq P_{min} + 3 \times s_{min}$. Suppose a sequence of data points where the error of the actual model increases reaching the warning level at

data point K_W,and the drift level at data point K_D. This is an indication of a change in the distribution of the data points. A new context is declared starting from data point K_W, and a new decision model is trained using only the data points starting from K_W.

It is possible to observe an increase of the error reaching the warning level, followed by a decrease. Such situations corresponds to a false alarm of context drift, which may be an anomaly. Figure 3.6 illustrates how concept drift window size is identified by DDM.

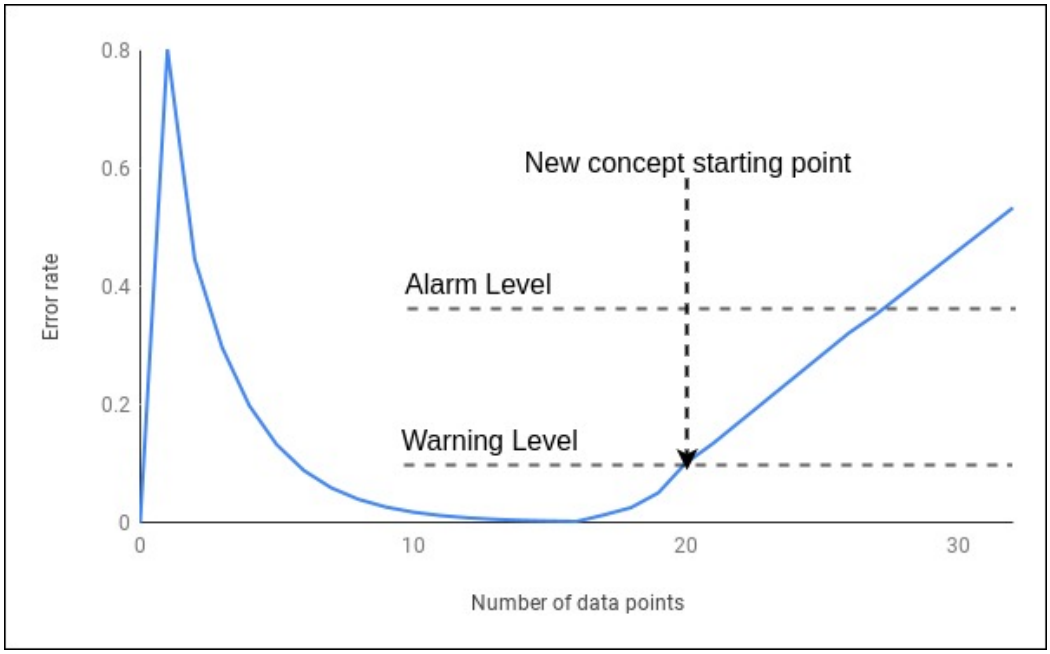

Figure 3.6: How concept drift window size is identified by DDM [2]

Chapter 4

Implementation

4.1 Sherlock Framework

Shipmon et al [9] presents a comparison of DNN, RNN, LSTM and a Fourier model for anomaly detection on time series data. They have concluded that RNN and Fourier models were effective predictive models and DNN and LSTM generates false alarms in peaks of a time series. But P. Malhotra et al [8] in their study of anomaly detection on time series using LSTM has concluded that LSTM performs better on anomaly detection on time series. These contradictory conclusions may be a result of data set biases of the two studies since they use different data sets in different scenarios. From these two studies we can infer that different approaches of anomaly detection on time series have unique advantages for different scenarios. Therefore it is preferred to implement a framework which can be used to implement different anomaly detection methods on time series. Such a framework would drastically reduce repetitive work to be done by a research in this domain. For this reason we have introduced a framework for anomaly detection on time series named "Sherlock" written in C++.

Structure of Sherlock framework is based on the generic schema provided by J. Gama et al [28] for online adaptive learning algorithms in their survey on concept drift adaptation. Figure 4.1 illustrates this schema. A system may consist of a memory module which decides how and which data is presented to the learning algorithm. And a loss estimation model to tack the performance of learning algorithm and send information to change detection module to update the learned model if required.

Sherlock framework consists of three main modules, Profiling module, Anomaly detection module and Concept Drift Detection module. There is a singleton class "SharedMemory" for communication in between these modules and it consists of a buffer to store past data points observed and other metrics. Modules have different

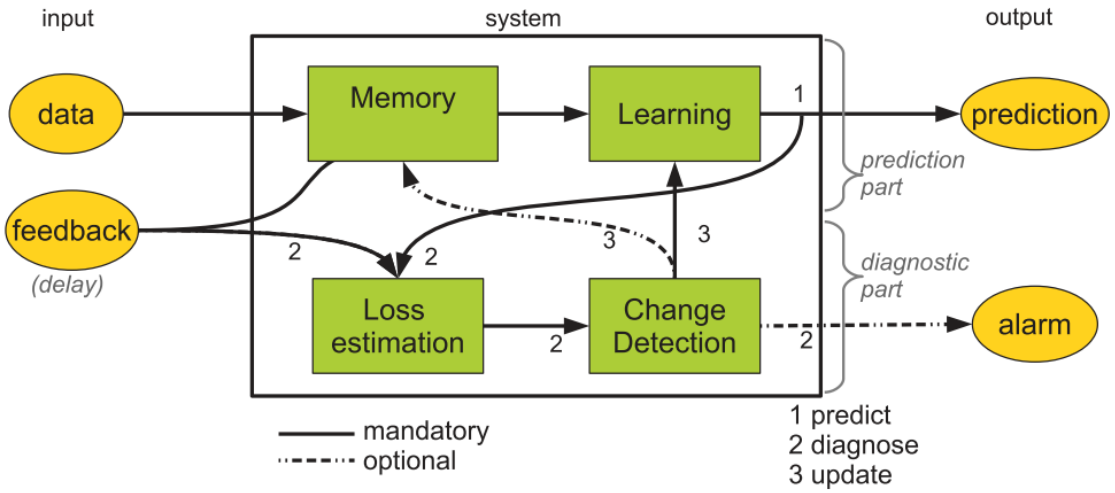

Figure 4.1: Generic schema provided by J. Gama et al [28] for online adaptive learning algorithms

abstract classes with interfaces that can be used to implement different methods to be used by those modules, which can be used to test mixtures of different methods for profiling, detecting anomaly and concept drifts without implementing same method again and again. A singleton class "MainLoop" handles the control of the framework. Figure 4.2 illustrates the high level architecture of Sherlock framework.

For this study we have implemented LSTMCNNnet and LSTMCNNcda with fastDTW [30] on Sherlock framework.

4.1.1 Profiling module

Profiling module consists of one abstract class "Profiler" with two interfaces called "profile" which should return the expected value for current observed data point and "init" to run initializing operations. Profiling module can access previously observed data points and shared flags set by other modules from "SharedMemory". "Profiler" class has a reference to "SharedMemory" which has to be set by the constructor of an implementing class. Other methods like "train" to train a profiler can be implemented as private methods in an implementing class. Listing 4.1 contains the source code of "Profiler" abstract class.

Listing 4.1: "Profiler" abstract class

```
1 #ifndef _PROFILER_H_
2 #define _PROFILER_H_
3
4 #include <string>
5 #include "SharedMemory.h"
```

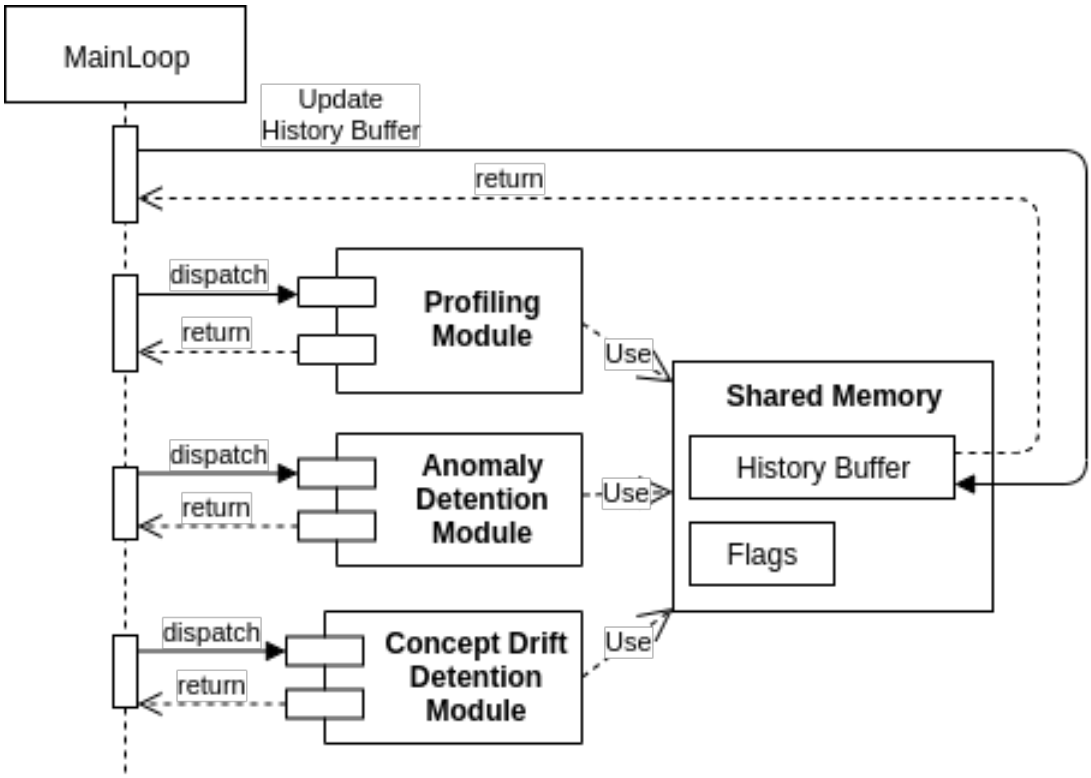

Figure 4.2: Architecture of Sherlock framework

```
6
7  class Profiler
8  {
9  private:
10
11 protected:
12     std::string identifyer;
13     SharedMemory *sharedMemory;
14
15 public:
16     // Destructor
17     virtual ~Profiler() {
18         sharedMemory = NULL;
19     };
20
21     // methods
22     void setSharedMemory(SharedMemory *sharedMemory){
23         this->sharedMemory = sharedMemory;
24     }
25
```

```
26      std::string getIdentifier(){
27          return this->identifyer;
28      }
29
30      // virtual methods
31      virtual void init() = 0;
32
33      virtual double profile() = 0;
34
35  };
36
37  #endif // _PROFILER_H_
```

4.1.2 Anomaly detection module

Anomaly detection module consists of three abstract classes "AnomalyDistanceMeasure", "AnomalyThresholdSetter", and "AnomalyDetector". "AnomalyDistanceMeasure" provides "measureDistance" interface which should return a double value. "AnomalyThresholdSetter" provides two interfaces "getWarningThreshold" and "getAlarmThreshold" which returns double values. These interfaces are used by the "AnomalyDetector" class to signal an anomaly detection or warning."AnomalyDetector" class provides "detectWarning" and "detectAlarm" which return boolean values. All of these abstract classes has a reference to "SharedMemory" object which has to be set in the constructor of an implementing class. And these classes can access anomaly distance, anomaly detection and warning signal history from the history buffer in "SharedMemory". Listing 4.2 contains the source code of "AnomalyDistanceMeasure" abstract class. Listing 4.3 contains the source code of "AnomalyThresholdSetter" abstract class. Listing 4.4 contains the source code of "AnomalyDetector" abstract class.

Listing 4.2: "AnomalyDistanceMeasure" abstract class

```
1  #ifndef _ANOMALYDISTANCEMEASURE_H_
2  #define _ANOMALYDISTANCEMEASURE_H_
3
4  #include <string>
5  #include "SharedMemory.h"
6
7  class AnomalyDistanceMeasure
8  {
```

```
9  private:
10
11 protected:
12     std::string identifyer;
13     SharedMemory *sharedMemory;
14
15 public:
16     // Destructor
17     virtual ~AnomalyDistanceMeasure() {
18         sharedMemory = NULL;
19     };
20
21     // methods
22     void setSharedMemory(SharedMemory *sharedMemory){
23         this->sharedMemory = sharedMemory;
24     }
25
26     std::string getIdentifier(){
27         return this->identifyer;
28     }
29
30     // virtual methods
31     virtual void init() = 0;
32
33     virtual double measureDistance() = 0;
34
35 };
36
37 #endif // _ANOMALYDISTANCEMEASURE_H_
```

Listing 4.3: "AnomalyThresholdSetter" abstract class

```
1  #ifndef _ANOMALYTHRESHOLDSETTER_H_
2  #define _ANOMALYTHRESHOLDSETTER_H_
3
4  #include <string>
5  #include "SharedMemory.h"
6
7  class AnomalyThresholdSetter
8  {
```

```
9  private:
10
11 protected:
12     std::string identifyer;
13     SharedMemory *sharedMemory;
14
15 public:
16     // Destructor
17     virtual ~AnomalyThresholdSetter() {
18         sharedMemory = NULL;
19     };
20
21     // methods
22     void setSharedMemory(SharedMemory *sharedMemory){
23         this->sharedMemory = sharedMemory;
24     }
25
26     std::string getIdentifier(){
27         return this->identifyer;
28     }
29
30     // virtual methods
31     virtual void init() = 0;
32
33     virtual double getWarningThreshold() = 0;
34
35     virtual double getAlarmThreshold() = 0;
36
37 };
38
39 #endif // _ANOMALYTHRESHOLDSETTER_H_
```

Listing 4.4: "AnomalyDetector" abstract class

```
1 #ifndef _ANOMALYDETECTOR_H_
2 #define _ANOMALYDETECTOR_H_
3
4 #include <string>
5 #include "SharedMemory.h"
6
```

```
7  class AnomalyDetector
8  {
9  private:
10
11 protected:
12     std::string identifyer;
13     SharedMemory *sharedMemory;
14
15 public:
16     // Destructor
17     virtual ~AnomalyDetector() {
18         sharedMemory = NULL;
19     };
20
21     // methods
22     void setSharedMemory(SharedMemory *sharedMemory){
23         this->sharedMemory = sharedMemory;
24     }
25
26     std::string getIdentifier(){
27         return this->identifyer;
28     }
29
30     // virtual methods
31     virtual void init() = 0;
32
33     virtual bool detectWarnning() = 0;
34
35     virtual bool detectAlarm() = 0;
36
37 };
38
39 #endif // _ANOMALYDETECTOR_H_
```

4.1.3 Concept Drift detection module

Concept drift detection module consist of similar abstract classes to Anomaly detection module “ConceptDistanceMeasure”, “ConceptThresholdSetter” and “ConceptDriftDetector”. All these classes also has a reference to “SharedMemory”

where implementing class should set “SharedMemory” object in its constructor. concept distance history can be accessed from the history buffer in “SharedMemory”. “ConceptDriftDetector” class has a single interface “detect” which returns a boolean value. When a concept drift is detected a flag in “SharedMemory” is set and data points in the old concept is flushed from the history buffer leaving data points from the new concept so that the Profiling Modules “Profiler” can train the profiler for the new concept. Listing 4.5 contains the source code of “ConceptDistanceMeasure” abstract class. Listing 4.6 contains the source code of “ConceptThresholdSetter” abstract class. Listing 4.7 contains the source code of “ConceptDriftDetector” abstract class.

Listing 4.5: “ConceptDistanceMeasure” abstract class

```
1  #ifndef _CONCEPTDISTANCEMEASURE_H_
2  #define _CONCEPTDISTANCEMEASURE_H_
3
4  #include <string>
5  #include "SharedMemory.h"
6
7  class ConceptDistanceMeasure
8  {
9  private:
10
11 protected:
12     std::string identifyer;
13     SharedMemory *sharedMemory;
14
15 public:
16
17     // Destructor
18     virtual ~ConceptDistanceMeasure() {
19         sharedMemory = NULL;
20     };
21
22     // methods
23     void setSharedMemory(SharedMemory *sharedMemory){
24         this->sharedMemory = sharedMemory;
25     }
26
27     std::string getIdentifier(){
```

```
28          return this->identifyer;
29      }
30
31      // virtual methods
32      virtual void init() = 0;
33
34      virtual double measureDistance() = 0;
35
36  };
37
38  #endif // _CONCEPTDISTANCEMEASURE_H_
```

Listing 4.6: "ConceptThresholdSetter" abstract class

```
1  #ifndef _CONCEPTTHRESHOLDSETTER_H_
2  #define _CONCEPTTHRESHOLDSETTER_H_
3
4  #include <string>
5  #include "SharedMemory.h"
6
7  class ConceptThresholdSetter
8  {
9  private:
10
11 protected:
12      std::string identifyer;
13      SharedMemory *sharedMemory;
14
15 public:
16      // Destructor
17      virtual ~ConceptThresholdSetter() {
18          sharedMemory = NULL;
19      };
20
21      // methods
22      void setSharedMemory(SharedMemory *sharedMemory){
23          this->sharedMemory = sharedMemory;
24      }
25
26      std::string getIdentifier(){
```

```
27          return this->identifyer;
28      }
29
30      // virtual methods
31      virtual void init() = 0;
32
33      virtual double getWarningThreshold() = 0;
34
35      virtual double getAlarmThreshold() = 0;
36
37  };
38
39  #endif // _CONCEPTTHRESHOLDSETTER_H_
```

Listing 4.7: "ConceptDriftDetector" abstract class

```
1  #ifndef _CONCEPTDRIFTDETECTOR_H_
2  #define _CONCEPTDRIFTDETECTOR_H_
3
4  #include <string>
5  #include "SharedMemory.h"
6
7  class ConceptDriftDetector
8  {
9  private:
10
11 protected:
12      std::string identifyer;
13      SharedMemory *sharedMemory;
14
15 public:
16      // Destructor
17      virtual ~ConceptDriftDetector() {
18          sharedMemory = NULL;
19      };
20
21      // methods
22      void setSharedMemory(SharedMemory *sharedMemory){
23          this->sharedMemory = sharedMemory;
24      }
```

```
25
26      std::string getIdentifier(){
27          return this->identifyer;
28      }
29
30      // virtual methods
31      virtual void init() = 0;
32
33      virtual int detect() = 0;
34
35  };
36
37  #endif // _CONCEPTDRIFTDETECTOR_H_
```

4.2 Profiling methods implementation

Auto-Regressive and Moving Average (ARMA) Profiling model is implemented as the baseline method to compare the performance of proposed profiling model from here on wards we will call this implementation as "arma" model.

LSTM and CNN ensemble neural network implementation is taken from Eranga's work [1], which is described in Section 3.1, this implementation is integrated with Sherlock framework and it is called "LSTMCNNnet" model.

Since anomaly detection performance of LSTMCNNnet is inferior when compared with arma model to validate the use of LSTM and CNN ensemble neural network for profiling method an LSTM CNN ensemble neural network was implemented using Keras[1] without any pre-processing or post-processing steps. This implementation is called "LSTMCNNkeras" model.

LSTMCNNkeras model has shown an improvement in certain metrics and drawbacks in some metrics when compared with LSTMCNNnet model. We have identified the reason behind the improvement was that LSTMCNNkeras model adapts to certain concept drifts in time series when predicting and LSTMCNNnet does not. Reason for this difference in results is the presence of time series data normalization and regularization steps in LSTMCNNnet. But these time series data normalization and regularization steps are required to generalize our profiling model for data sets with different feature scales. Although in a time series scale of the features may change as new data points are generated. This is a known issue in this model.

[1] https://keras.io/

Further we have implemented another model by extending LSTMCNNnet with a concept drift detection and adaptation method desctibed in Section 3.3.2. This implementation will be called "LSTMCNNcda"(cda stands for Concept Drift Adaptation) model.

4.2.1 ARMA profiling model implementation

Arma model is implemented using python Statsmodels library[2]. Arma model has two parameters p and q. p is the number of previous time points taken into the Auto Regressive(AR) model. q is the number of errors associated with previous time points in the Moving Average(MA) model. Training data set is used to find the arma(p,q) model that yields the minimum mean square error when predicting the time series. Error is taken as the euclidean distance between the observed time series and predicted time series. Parameter q and p varied from zero to three.

4.2.2 LSTMCNNnet profiling model implementation

Details of LSTMCNNnet implemented on Sherlock framework is provided in Section 3.1. Parameters of feature combination layer weight of LSTM and weight of CNN are set for different data sets giving higher weight for LSTM if anomaly detection is higher with just LSTM profiler compared to CNN profiler and vise versa. Learning rate is set arbitrary in between 0.1-.0001 for different data sets which are taken from Eranga's [1] work. And in the LSTM network number of LSTM cells and in the CNN matrix height, matrix width number of fully connected layers and their size are taken from Eranga's work for different data sets. In the CNN pooling layer height and weight are set to one, Convolution layer strides, filters are set to one and filter size is set two for all data sets. Training iterations are set to ten and number of forecasting(profiling) points are set to one for all data sets.

4.2.3 LSTMCNNkeras profiling model implementation

LSTMCNNkeras was implemented using python Keras framework with the same structure and parameters of LSTMCNNnet except the pre-processing and post-processing steps in LSTM network and CNN in LSTMCNNnet, which are the time series data normalization and regularization steps. There are no any pre-processing or post-processing steps in the LSTMCNNkeras implementation.

[2] `https://www.statsmodels.org/stable/index.html`

4.2.4 LSTM and CNN profiling models implementation

LSTM network and a CNN network are implemented separately in python Keras framework with same structure and parameters of LSTM network and CNN in LSTMCNNnet except the pre-processing and post-processing steps. These two profiling models were implemented to compare performance increase of ensemble LSTM and CNN network (LSTMCNNkeras) when compared with vanilla version of LSTM and CNN.

4.2.5 LSTMCNNcda profiling model implementation

LSTMCNNcda is implemented on Sherlock framework by extending the LSTM CNNnet implementation on Sherlock framework with Concept Drift Detection method introduced by Gama, J. et al [2] in their paper "Learning with Drift Detection". Details of this method is mentioned in Section 3.3.2. For the observation time window for concept drifts we have used 30 data points.

4.3 Anomaly Detection method implementation

For anomaly detection we have used an implementation of fastDTW distance measure introduced by Salvador, S. and Chan, P. [30]. Details of their algorithm is mentioned in Section 3.2. For the time window size to calculate warp distance we have used the window size used by Eranga [1] in his work for different data sets. If the distance between profiled time window and the observed time window exceeds a predefined threshold, that data point is marked as an anomaly.

4.4 Threshold setting mechanism

As mentioned above a threshold value is set to detect anomalies using DTW distance measure between observed time series and profiling model output. To set this threshold value a threshold setting process is used for the last one third of training data available in a time series data set. As the threshold setting process we have taken the maximum distance observed and multiplied it with a constant. For our experiments we have selected two as this constant.

Chapter 5

Results and Evaluation

All experiments were carried out on a laptop PC with Intel Core i7-7700HQ processor with eight virtual cores and 2.80 Ghz clock speed. And 16 GB RAM on Ubuntu 10.04 LTS Operating system.

5.1 Data set

For all experiments Numenta Anomaly Benchmark (NAB)[1] data set was used. NAB contains data sets of 58 different anomaly detection scenarios in seven categories. All data sets have anomalies labeled by a team of observers [2]. One category is "realAWSCloudwatch", which contains AWS server metrics collected by AmazonCloudwatch service, metrics include CPU Utilization, Network Bytes In, and Disk Read Bytes. Another category is "realAdExchange" , which contains metrics such as cost-per-click (CPC) and cost per thousand impressions (CPM). Another category is "realKnownCause" , which contain data from many domains where the anomaly cause is known and anomalies are not hand labelled. Another category is "realTraffic" , which contain data collected from the Minnesota Department of Transportation in city metro areas of Minnesota, which includes occupancy, speed, and travel time captured by specific sensors. Another category is "realTweets" , which contain collection of Twitter mentions of a publicly traded companies such as Google and Amazon. Value observed is the number of mentions for given 5 minute time window. Another category is artificially generated data with anomalies. And the final category is artificially generated data without anomalies.

Each data set is divided into two main parts: the first part of the time series is taken for training and the second part is taken for testing. As default, first 15%

[1] https://github.com/numenta/NAB/tree/master/data

[2] https://drive.google.com/file/d/0B1_XUjaAXeV3dW1kX1B3VkYwOFE/view

of the time series is taken as training data, but if there are anomalies in the first 15% of the time series data points up to the first anomaly is taken for training data. First two third of the training data is taken for profiling model training and other one third is taken for threshold setting process. Figure 5.1 illustrates training data is selected.

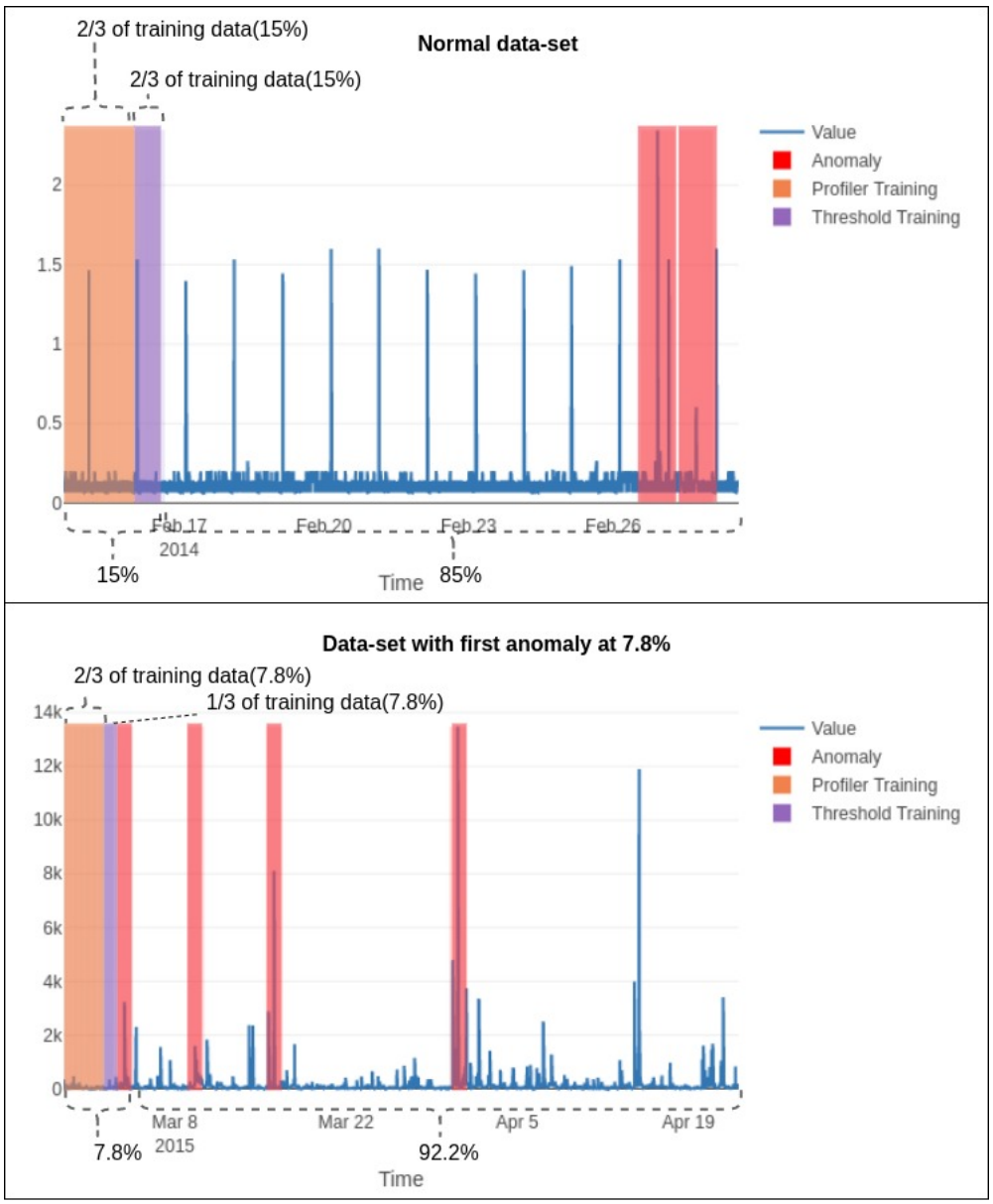

Figure 5.1: How training data is selected. In the top figure a first 15% of the data set is taken as training data. In the bottom figure first 7.8% of the data-set is taken as training data since there is an anomaly at the 7.8% percentile of the data set.

5.2 Results

Metrics used to compare anomaly detection performance with different profiling models are True Positive Rate(TPR) which represents how many anomalies are detected from all anomalies , Precision(PVV) which represents how many positive detections are correct, and Weighted Harmonic Mean of TPR and PVV with 1500:1 weight ratio for TPR:PVV. Harmonic weights are used since TPR and PVV are measures of rates and different profiling methods have the best value for TPR and PVV. We have given a very high weight for TPR since we assume that in many scenarios detecting an anomaly is has much greater importance than reducing false alarms. Mean Square Error for forecasting time series for different profiling models are presented to infer how efficiently profiling model forecasts time series data.

We have taken all the positive detection in an anomaly window as one True Positive(TP) and all positive detections outside anomaly window as False Positives(FP). If there were no positive detections in an anomaly window we count it as a False Negative(FN). And all other points as True Negatives(TN). Figure 5.2 illustrates how TPs and FPs are counted.

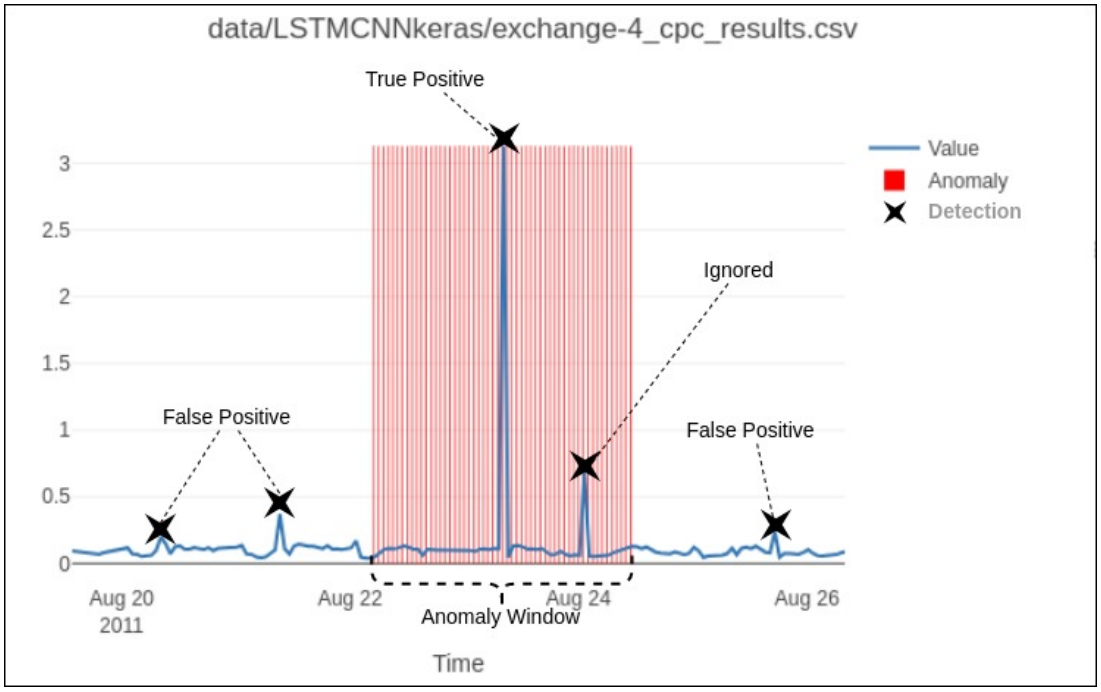

Figure 5.2: How True Positives(TP) and False Positives(FP) are counted.

Table 5.1 represents the number of True Positives, False Positives, False Negatives and True Negatives generated by anomaly detection using each profiling method ARMA, CNN, LSTM, LSTMCNNnet, LSTMCNNkeras, and LSTMCNNcda. Highest number of True Positives and lowest number of False Negatives were obtained by LSTMCNNkeras profiling method. From observing these metrics

we can infer that LSTMCNNkeras model is better.

Table 5.1: Raw confusion metrics obtained for anomaly detection with ARMA, CNN, LSTM, LSTMCNNnet, LSTMCNNkeras, and LSTMCNNcda profiling methods.

	TP	FP	FN	TN
ARMA	72	4494	44	327569
CNN	73	13194	43	318869
LSTM	64	13461	52	318602
LSTMCNNnet	51	**1542**	65	**330521**
LSTMCNNkeras	**77**	12924	**39**	319139
LSTMCNNcda	54	5719	62	326344

For further comparison we have calculated TPR, PVV and F-1500 in Table 5.2. LSTMCNNkeras has got the highest TPR which can be inferred as the profiling method with most anomalies detected. LSTMCNNnet has the highest PVV which can be inferred as the profiling method with least false alarms in anomaly detection. F-1500 score is highest in the LSTMCNNkeras. From these metrics we can infer that LSTMCNNkeras has better ability to detect all anomalies but LSTMCNNnet has a better precision where less false alarms are generated, both of these features are preferable in anomaly detection. We can observe that difference of anomaly detection metrics from Fig. 5.3 and forecasting metrics from Fig. 5.4. LSTMCNNkeras method has the lowest mean square error for forecasting time series.

Table 5.2: TPR, PVV, and F-1500 metrics multiplied by 100 for anomaly detection with ARMA, CNN, LSTM, LSTMCNNnet, LSTMCNNkeras, and LSTMCNNcda profiling methods.

	ARMA	CNN	LSTM	LSTMCNNnet	LSTMCNNkeras	LSTMCNNcda
TPR	62.06	62.93	55.17	43.96	**66.37**	46.55
PVV	1.57	0.55	0.47	**3.2**	0.59	0.93
F-1500	60.52	58.51	51.22	43.59	**61.8**	45.08
MSE	1.71E+15	1.43E+16	1.94E+15	8.68E+15	**3.12E+13**	1.24E+16

Few plots of time series data sets are discussed below. Plots contain Warp distance(in green), Value(in blue) and Anomaly window(highlighted in red) of time series data with concept drifts of LSTMCNNkeras, LSTMCNNnet, and LSTMCNNcda are compared.

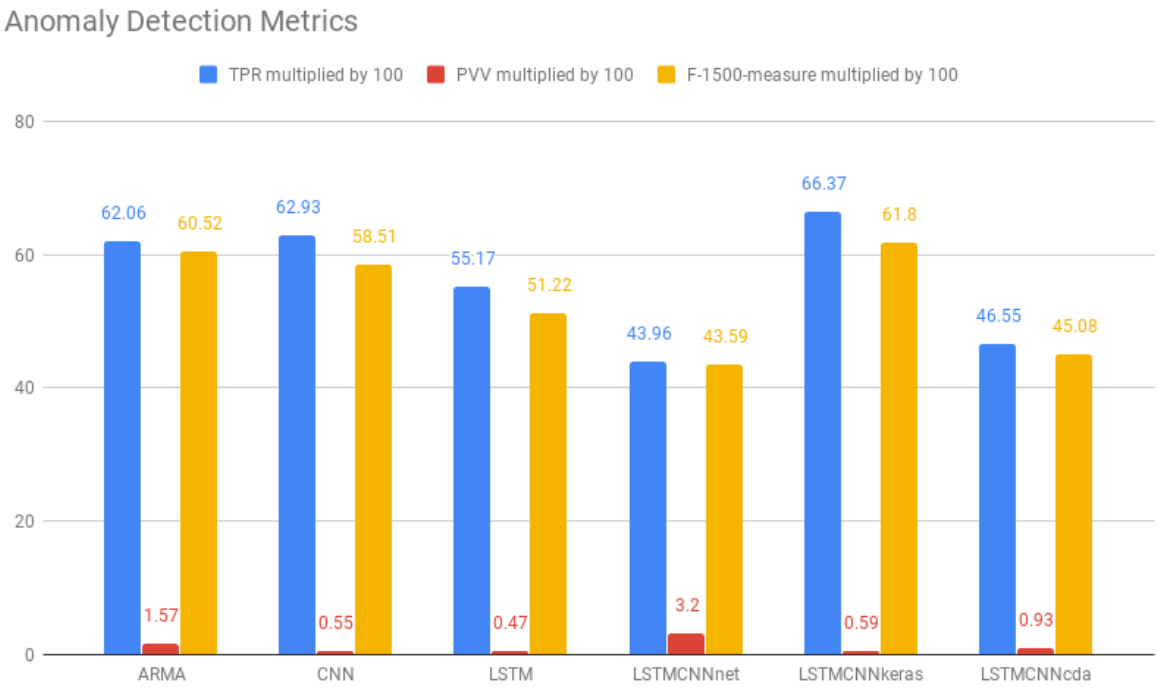

Figure 5.3: Bar chart of TPR, PVV, F-15000 metrics for anomaly detection using ARMA, CNN, LSTM, LSTMCNNnet, LSTMCNNkeras, LSTMCNNcda profiling methods.

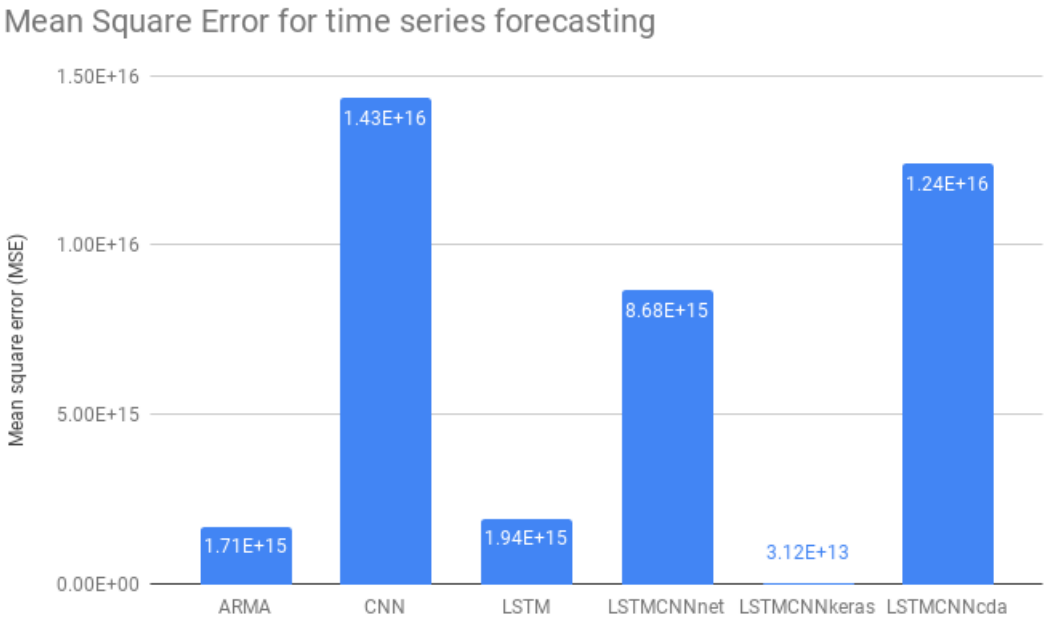

Figure 5.4: Bar chart of mean square error of forecasting for ARMA, CNN, LSTM, LSTMCNNnet, LSTMCNNkeras, LSTMCNNcda profiling methods.

From Fig. 5.5 we can observe that LSTMCNNkeras profiling method adapts to a concept drift in a very short period. But LSTMCNNnet does not adapt to the concept drift at all. LSTMCNNcda has small adaption to the concept drift yielding a better result that LSTMCNNnet.

From Fig. 5.6 we can observe that LSTMCNNkeras profiling method adapts to a concept drift quickly. But LSTMCNNnet does not adapt to the concept drift. At the beginning of the concept drift LSTMCNNcda yields worst forecasting performance than LSTMCNNnet but gradually it adapts to the concept drift yielding better performance.

From Fig. 5.7 we can observe that LSTMCNNkeras profiling method adaptor to two concept drifts relatively closer to each other quickly. LSTMCNNnet does not adapt to this concept drift. LSTMCNNcda tries to adopt for the 1st concept drift when it is at the starting stages of the adaption concept again shifts to the old one where LSTMCNNcda yields worst forecasting performance than LSTMCNNnet.

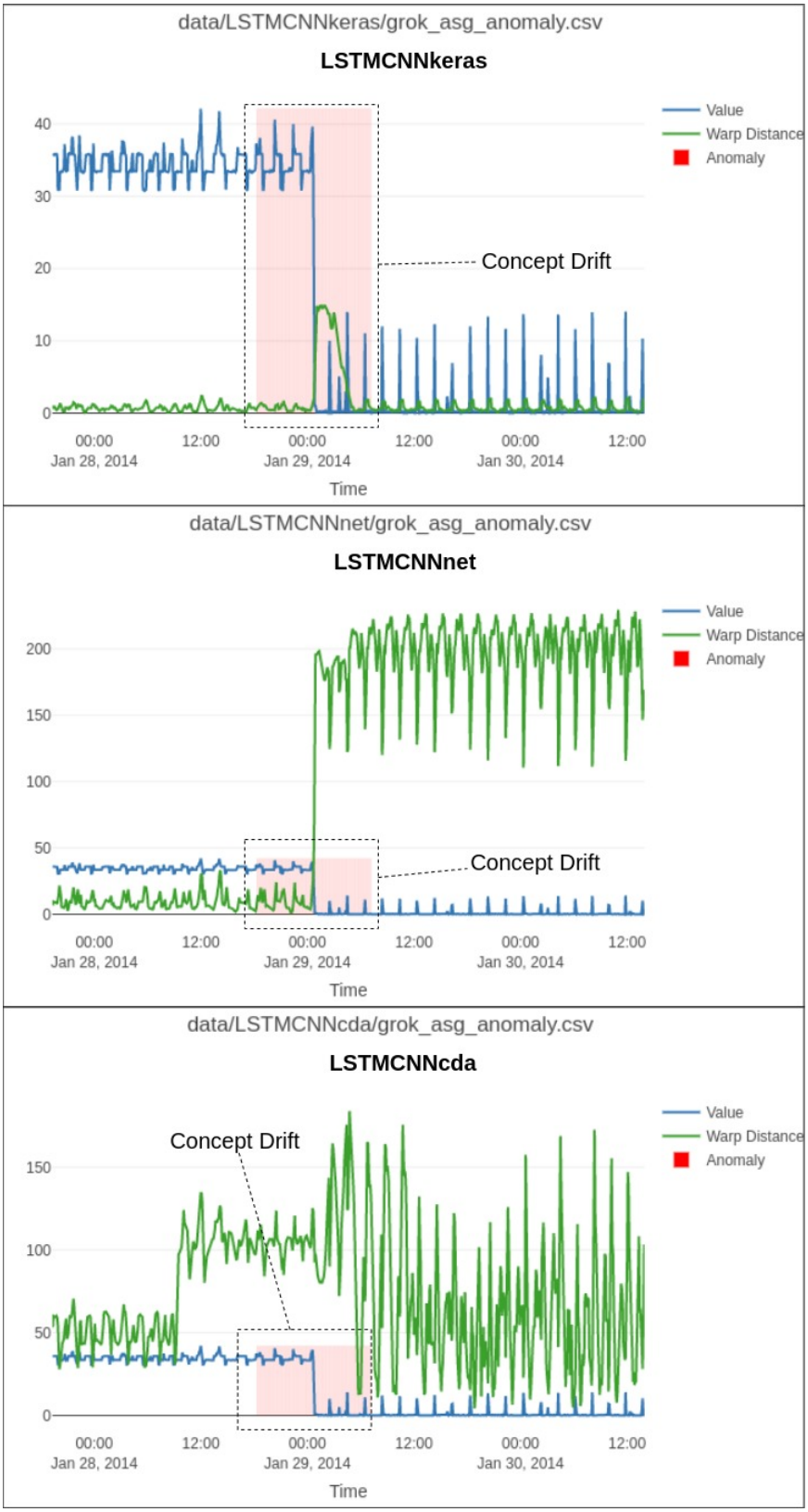

Figure 5.5: Illustrates how LSTMCNNkeras, LSTMCNNnet, and LSTMCNNcda behave when there is a concept drift in "grok_asg_anomaly" data set. Value, Warp distance, and Anomaly window is shown in blue, green, and red respectively.

Figure 5.6: Illustrates how LSTMCNNkeras, LSTMCNNnet, and LSTMCNNcda behave when there is a concept drift in "rds_cpu_utilization_e47b3b" data set. Value, Warp distance, and Anomaly window is shown in blue, green, and red respectively.

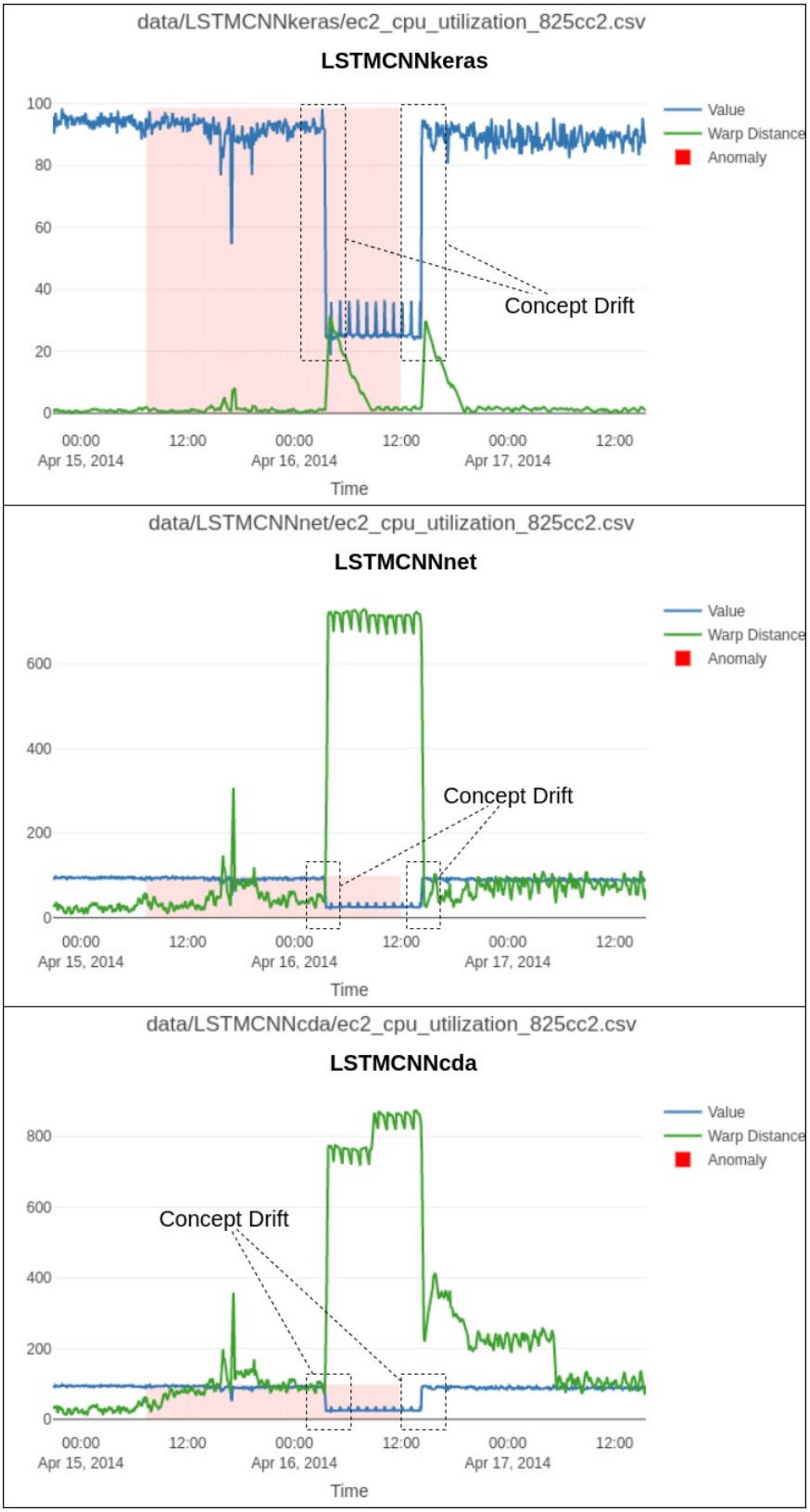

Figure 5.7: Illustrates how LSTMCNNkeras, LSTMCNNnet, and LSTMCNNcda behave when there is a concept drift in "ec2_cpu_utilization_825cc2" data set. Value, Warp distance, and Anomaly window is shown in blue, green, and red respectively.

Chapter 6

Conclusions

6.1 Introduction

In conclusion we can say that ensemble LSTM CNN network for normal profiling method with dynamic time warping for distance measure for anomaly detection on time series performs better that baseline profiling method ARMA.

But when ensemble LSTM CNN network is introduced with data normalization and regularization steps for generalizing the model it's performance drops since it fails to adapt to concept drifts. When concept drift detection and adaptation method introduced in [2] is incorporated with LSTM CNN ensemble profiling method with normalization and regularization steps there is a slight improvement but it does not reach the performance of LSTM CNN ensemble profiling method without normalization and regularization steps.

6.2 Conclusions about research questions (aims/objectives)

We can conclude that ensemble LSTM CNN profiling method with DTW for anomaly detection on time series performs better than baseline profiling method ARMA. But LSTMCNNnet does not perform better than the baseline profiling method ARMA.

We can also conclude that performance gap between LSTMCNNnet and LSTM-CNNkeras is because LSTMCNNnet fails to adapt to certain concept drifts in time series. from our observations in Chapter 5

6.3 Conclusions about research problem

We can conclude that LSTMCNNnet performance is improved when Concept Drift Adaptation [2] method is incorporated with LSTMCNNnet from the F-1500 metric observed in Chapter 5. And in most cases of concept drifts LSTMCNNcda performs better than LSTMCNNnet.

6.4 Limitations

Problem of input normalization for time series data is a complex issue which is persistent in other LSTM and CNN based prediction models used for time series forecasting. Since normalizing a time series is difficult because maximum and minimum values of range is not know in most scenarios.

Threshold setting mechanism on distance measure to deferential anomalies from normal observations used in our implementation is not an optimum solution.

6.5 Implications for further research

Further research can be carried out on time series data normalization and regularization techniques or finding better concept drift adaptation method to be coupled with LSTMCNNnet.

A dynamic threshold setting mechanism can be used to further improve performance of proposed method.

References

[1] N.A.A.H.Eranga and M.I.E.Wickramasinghe, "Lstm based framework for time series anomaly detection," erangamx@gmail.com, 1 2019, submitted in partial fulfillment of the requirements of the B.Sc. (Hons) in Computer Science Final Year Project (SCS 4124).

[2] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 286–295.

[3] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, Sep. 2014.

[4] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control.* Wiley, 2016.

[5] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts, 2018.

[6] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, Jul 2016. [Online]. Available: https://doi.org/10.1007/s10618-015-0448-4

[7] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec 2015, pp. 38–44.

[8] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *ESANN*, 2015.

[9] D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, and S. T. Edwards, "Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data," *CoRR*, vol. abs/1708.03665, 2017.

[10] L. Bontemps, V. L. Cao, J. McDermott, and N.-A. Le-Khac, "Collective anomaly detection based on long short-term memory recurrent neural networks," in *Future Data and Security Engineering*, T. K. Dang, R. Wagner, J. Küng, N. Thoai, M. Takizawa, and E. Neuhold, Eds. Cham: Springer International Publishing, 2016, pp. 141–152.

[11] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *Web-Age Information Management*, F. Li, G. Li, S.-w. Hwang, B. Yao, and Z. Zhang, Eds. Cham: Springer International Publishing, 2014, pp. 298–310.

[12] *Dynamic Time Warping*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 69–84. [Online]. Available: https://doi.org/10.1007/978-3-540-74048-3_4

[13] A. J. Fox, "Outliers in time series," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 34, no. 3, pp. 350–363, 1972. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1972.tb00912.x

[14] G. C. Tiao and G. E. P. Box, "Modeling multiple times series with applications," *Journal of the American Statistical Association*, vol. 76, no. 376, pp. 802–816, 1981. [Online]. Available: http://www.jstor.org/stable/2287575

[15] J. Ma and S. Perkins, "Time-series novelty detection using one-class support vector machines," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 3, July 2003, pp. 1741–1745 vol.3.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.173

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[19] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2

[20] E. (https://stats.stackexchange.com/users/190470/ethan), "Recurrent neural network (rnn) with multiple outputs," Cross Validated, uRL:https://stats.stackexchange.com/q/321635 (version: 2018-01-04). [Online]. Available: https://stats.stackexchange.com/q/321635

[21] S. Kim and R. Casper, "Applications of convolution in image processing with matlab," *University of Washington*, 2013. [Online]. Available: www.math.washington.edu/~wcasper/math326/projects/sung_kim.pdf

[22] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," 2017.

[23] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Artificial Neural Networks — ICANN 2001*, G. Dorffner, H. Bischof, and K. Hornik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 669–676.

[24] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[25] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, p. 23, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2016.00023

[26] D. E. Padilla, R. Brinkworth, and M. D. McDonnell, "Performance of a hierarchical temporal memory network in noisy sequence learning," in *2013 IEEE International Conference on Computational Intelligence and Cybernetics (CYBERNETICSCOM)*, Dec 2013, pp. 45–51.

[27] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Computation*, vol. 28, no. 11, pp. 2474–2504, 2016, pMID: 27626963. [Online]. Available: https://doi.org/10.1162/NECO_a_00893

[28] J. a. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014. [Online]. Available: https://doi.org/10.1145/2523813

[29] P. M. Gonçalves, S. G. de Carvalho Santos, R. S. Barros, and D. C. Vieira, "A comparative study on concept drift detectors," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8144 – 8156, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414004175

[30] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, Oct. 2007.

[31] L. Vinet and A. Zhedanov, "A 'missing' family of classical orthogonal polynomials," *Journal of Physics A: Mathematical and Theoretical*, vol. 44, no. 8, p. 085201, jan 2011. [Online]. Available: https://doi.org/10.1088%2F1751-8113%2F44%2F8%2F085201

[32] T. Lin, T. Guo, and K. Aberer, "Hybrid neural networks for learning the trend in time series," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 2273–2279. [Online]. Available: https://doi.org/10.24963/ijcai.2017/316

Appendices

Appendix A

Code Listings

Listing A.1: Main method for Sherlock framework to implement and run LSTMC-NNcda on NAB data set

```
1  #include <stdlib.h>
2
3  #include <iostream>
4  #include <iomanip>
5  #include <fstream>
6  #include <thread>
7  #include <mutex>
8  #include <memory>
9  #include <sys/types.h>
10 #include <unistd.h>
11 #include <sys/mman.h>
12 #include <stdio.h>
13 #include <json.hpp>
14 #include <iomanip>
15 #include <dirent.h>
16 #include <sys/types.h>
17 #include <sys/stat.h>
18 #include <math.h>
19
20
21 #include "MainLoop.h"
22 #include "HistoryBuffer.h"
23 #include "SharedMemory.h"
24
25 #include "Profiler.h"
```

```
26 #include "DefaultProfiler.h"
27 #include "LSTMCNnetProfiler.h"
28
29 #include "AnomalyDistanceMeasure.h"
30 #include "FastDTWAnomalyDistanceMeasure.h"
31
32 #include "AnomalyThresholdSetter.h"
33 #include "TrainingMaxAnomalyThresholdSetter.h"
34 #include "ConfidenceIntervalAnomalyThresholdSetter.h"
35
36 #include "AnomalyDetector.h"
37 #include "DefaultAnomalyDetector.h"
38 #include "ConfidenceIntervalAnomalyDetector.h"
39
40 #include "ConceptDistanceMeasure.h"
41 #include "FastDTWConceptDistanceMeasure.h"
42
43 #include "ConceptThresholdSetter.h"
44 #include "ConfidenceIntervalConceptThresholdSetter.h"
45
46 #include "ConceptDriftDetector.h"
47 // #include "NoConceptDriftDetector.h"
48 #include "DynamicWindowConceptDriftDetector.h"
49
50
51 void runLSTMCNnet(std::string inputFileName, std::string
      outputFileName);
52 void processConfigJSON(std::string fileJSON);
53 void processDataSet(std::string filePath, std::string
      fileName);
54 void configureMainLoop();
55
56 HistoryBuffer *historyBuffer;
57 Profiler *profiler;
58 SharedMemory *sharedMemory;
59 MainLoop *mainLoop;
60 ModelStruct *modelStruct;
61
62
```

```
63 std::string inputFileName = "../data/nab_tuned/test/
      Twitter_volume_AAPL.csv"; // set by args
64 std::string configurationFileName = "../data/nab_tuned/test
      /Twitter_volume_AAPL.json"; //set by args
65 std::string outputFileName = "../data/nab_tuned_results/
      test1/Twitter_volume_AAPL.csv"; // set by args
66
67 bool verbose = false;
68 int processedCount = 0;
69 int processesLimit = -1;
70 double predictionTrainingRatio = 0.1; // set by args
71 double thresholdTrainingRatio = 0.045; // set by args
72 double thresholdMaxMultipler = 2.0; // set by args
73
74 struct ROW {
75     std::string timestamp;
76     double value;
77     bool prediction_training;
78     double prediction;
79     bool label;
80     double warp_distance;
81     bool threshold_training;
82     double distance_threshold;
83     bool positive_detection;
84
85     ROW(){
86         value = 0.0;
87         prediction_training = true;
88         prediction = 0.0;
89         label = false;
90         warp_distance = 0.0;
91         threshold_training = true;
92         distance_threshold = 0.0;
93         positive_detection = false;
94     }
95 };
96
97
98
```

```
 99
100 int main(int argc, char** argv)
101 {
102
103     if(argc < 7){
104         std::cout << "[main] Provide arguments : '$Sherlock
                inputFileName outputFileName
               configurationFileName predictionTrainingRatio
               thresholdTrainingRatio thresholdMaxMultipler'"
              <<std::endl;
105         return 0;
106     }
107     inputFileName = argv[1];
108     std::cout << "[main] inputFileName = " << inputFileName
          << std::endl;
109     outputFileName = argv[2];
110     std::cout << "[main] outputFileName = " <<
          outputFileName << std::endl;
111     configurationFileName = argv[3];
112     std::cout << "[main] configurationFileName = " <<
          configurationFileName << std::endl;
113
114     predictionTrainingRatio = atof(argv[4]);
115     std::cout << "[main] predictionTrainingRatio = " <<
          predictionTrainingRatio << std::endl;
116     thresholdTrainingRatio = atof(argv[5]);
117     std::cout << "[main] thresholdTrainingRatio = " <<
          thresholdTrainingRatio << std::endl;
118     thresholdMaxMultipler = atof(argv[6]);
119     std::cout << "[main] thresholdMaxMultipler = " <<
          thresholdMaxMultipler << std::endl;
120
121     std::cout << "[main] Starting" <<std::endl;
122
123
124     processConfigJSON(configurationFileName); // it
          configures MainLoop too
125     runLSTMCNnet(inputFileName, outputFileName);
126
```

```
127
128     std::cout << "[main] Ended" << std::endl;
129     return 0;
130 }
131
132 void processConfigJSON(std::string fileJSON){
133
134     std::ifstream fileJSONStream(fileJSON);
135     nlohmann::json JSONDocument;
136     fileJSONStream >> JSONDocument;
137
138     // Initializing the structure
139     modelStruct = new ModelStruct();
140
141     int input_size = (int)JSONDocument["input_size"];
142
143     modelStruct->learningRate = (double)JSONDocument["
            prediction_model"]["learningRate"];
144     std::cout << "[main:processConfigJSON] Got learningRate
            from : "<< fileJSON << " : " << modelStruct->
            learningRate << std::endl;
145
146     modelStruct->trainingIterations = (int)JSONDocument["
            prediction_model"]["trainingIterations"];
147     std::cout << "[main:processConfigJSON] Got
            trainingIterations from : "<< fileJSON << " : " <<
            modelStruct->trainingIterations << std::endl;
148
149     modelStruct->numPredPoints = 1;
150
151     // LSTM parameters
152     modelStruct->memCells = (int)JSONDocument["
            prediction_model"]["model"]["LSTM"]["memCells"];
153     std::cout << "[main:processConfigJSON] Got LSTM params
            from : "<< fileJSON << std::endl;
154
155     // CNN parameters
156     modelStruct->matWidth = (int)JSONDocument["
            prediction_model"]["model"]["CNN"]["matWidth"];
```

```
157     modelStruct->matHeight = (int)JSONDocument["
            prediction_model"]["model"]["CNN"]["matHeight"];
158     modelStruct->targetC = (int)JSONDocument["
            prediction_model"]["model"]["CNN"]["targetC"];
159
160     std::vector<nlohmann::json> ConvolutionLayers;
161     JSONDocument["prediction_model"]["model"]["CNN"]["
            ConvolutionLayers"].get_to(ConvolutionLayers);
162
163     struct::ConvLayStruct *CLs = new struct::ConvLayStruct
            [(int)ConvolutionLayers.size()];
164     for(int i=0; i<(int)ConvolutionLayers.size(); i++) {
165         CLs[i].filterSize = (int)ConvolutionLayers[i]["
                filterSize"]; // filter size: N x N
166         CLs[i].filters = (int)ConvolutionLayers[i]["filters
                "]; // No of filters
167         CLs[i].stride = (int)ConvolutionLayers[i]["stride"
                ];
168     }
169
170     // Pooling layers
171     std::vector<nlohmann::json> PoolingLayers;
172     JSONDocument["prediction_model"]["model"]["CNN"]["
            PoolingLayers"].get_to(PoolingLayers);
173
174     struct::PoolLayStruct *PLs = new struct::PoolLayStruct
            [(int)PoolingLayers.size()];
175     for(int i=0; i<(int)PoolingLayers.size(); i++) {
176         PLs[i].poolH = (int)PoolingLayers[i]["poolH"];  //
                pool size: N x N
177         PLs[i].poolW = (int)PoolingLayers[i]["poolW"];
178     }
179
180     // Fully connected layers
181     std::vector<nlohmann::json> FullyConnectedLayers;
182     JSONDocument["prediction_model"]["model"]["CNN"]["
            FullyConnectedLayers"].get_to(FullyConnectedLayers);
183
```

```
184     struct::FCLayStruct *FCLs = new struct::FCLayStruct[(
            int)FullyConnectedLayers.size()];
185     for(int i=0; i<(int)FullyConnectedLayers.size(); i++) {
186         FCLs[i].outputs = (int)FullyConnectedLayers[i]["
                outputs"];  // neurons in fully connected layer
187     }
188
189     // Setting layer order
190     std::vector<nlohmann::json> layerOrderJSON;
191     JSONDocument["prediction_model"]["model"]["CNN"]["
            LayerOrder"].get_to(layerOrderJSON);
192
193     char *layerOrder = new char[(int)layerOrderJSON.size()
            ];
194     std::string layer;
195     for(int i=0; i<(int)layerOrderJSON.size(); i++) {
196         layer =  (std::string)layerOrderJSON[i];
197         layerOrder[i] = layer[0];  // neurons in fully
                connected layer
198     }
199
200     std::cout << "[main:processConfigJSON] Got CNN params
            from : "<< fileJSON << std::endl;
201     modelStruct->netStruct.layers = (int)layerOrderJSON.
            size();
202     modelStruct->netStruct.layerOrder = layerOrder;
203     modelStruct->netStruct.CL = CLs;
204     modelStruct->netStruct.PL = PLs;
205     modelStruct->netStruct.FCL = FCLs;
206
207
208
209     std::cout << "[main:processConfigJSON] Done building
            model in file : "<< fileJSON << " !" <<std::endl;
210
211
212     modelStruct->trainDataSize = (int)(input_size*
            predictionTrainingRatio) + 1 - (int)(modelStruct->
            matHeight*modelStruct->matWidth);
```

```
213
214     // # History Buffer
215     historyBuffer = new HistoryBuffer();
216     int historyBufferSize = modelStruct->trainDataSize+(
            modelStruct->matHeight*modelStruct->matWidth);
217     double *bufferHistory = new double[historyBufferSize];
218     historyBuffer->setSize(historyBufferSize, bufferHistory
            );
219
220     double lstmW = (double)JSONDocument["prediction_model"
            ]["model"]["lstmW"];
221     std::cout << "[main:processConfigJSON] Got LSTM Weight
            from : "<< fileJSON << " : " << lstmW << std::endl;
222
223     double cnnW = (double)JSONDocument["prediction_model"][
            "model"]["cnnW"];
224     std::cout << "[main:processConfigJSON] Got CNN Weight
            from : "<< fileJSON << " : " << cnnW << std::endl;
225
226     // # LSTMCNnetProfiler
227     profiler = new LSTMCNnetProfiler("lstmcnnnet 1",
            modelStruct, lstmW, cnnW);
228
229
230     // # Shared Memory
231
232     // ## profiler
233     int profilerSize = (int)JSONDocument["dtw_window"];
234     ProfilerMemory *profilerMemory = new ProfilerMemory(
            profilerSize);
235     profilerMemory->inWindowSize = modelStruct->matWidth*
            modelStruct->matHeight;
236     profilerMemory->OutWindowSize = 1;
237     profilerMemory->minTrainingWindowSize = modelStruct->
            trainDataSize + (modelStruct->matHeight*modelStruct
            ->matWidth);
238
239     // ### anomaly detector
```

```
240     int anomlayDetectorInWindowSize = (int)JSONDocument["
            dtw_window"];
241     int anomalyDistanceSize = (int)((input_size - (
            modelStruct->trainDataSize+(modelStruct->matHeight*
            modelStruct->matWidth)))*thresholdTrainingRatio) -
            anomlayDetectorInWindowSize;
242     int anomlayWarrningSize = 1;
243     int anomalyAlarmSize = 1;
244     AnomalyDetectorMemory *anomalyDetectorMemory = new
            AnomalyDetectorMemory(anomalyDistanceSize,
            anomlayWarrningSize, anomalyAlarmSize);
245     anomalyDetectorMemory->inWindowSize =
            anomlayDetectorInWindowSize;
246
247     // ### concept drift detector
248     int conceptDetectorInWindowSize =
            anomlayDetectorInWindowSize;
249     int conceptDistanceSize = anomalyDistanceSize;
250     int conceptWarningSize = 1;
251     int conceptAlarmSize = 1;
252     ConceptDriftDetectorMemory *conceptDriftDetectorMemory
            = new ConceptDriftDetectorMemory(conceptDistanceSize
            , conceptWarningSize, conceptAlarmSize);
253     conceptDriftDetectorMemory->inWindowSize=
            conceptDetectorInWindowSize;
254
255     sharedMemory = new SharedMemory(profilerMemory,
            anomalyDetectorMemory, conceptDriftDetectorMemory);
256     sharedMemory->history = new Queue<double>(
            historyBufferSize);
257
258     configureMainLoop();
259 }
260
261 void configureMainLoop(){
262
263     mainLoop = new MainLoop(verbose);
264
265     // set historyBuffer
```

```
266     mainLoop->setHistoryBuffer(historyBuffer);
267
268     // ## Anomaly Detection
269     // setting anomaly distance measure
270     AnomalyDistanceMeasure *anomalyDestanceMeasure = new
            FastDTWAnomalyDistanceMeasure("fastDTW 1");
271     mainLoop->setAnomalyDistanceMeasure(
            anomalyDestanceMeasure);
272
273     // setting anomaly threshold setter
274     double maxMultiplierAlarm = thresholdMaxMultipler;
275     double maxMultiplierWarning = 1.5;
276     AnomalyThresholdSetter *anomalyThresholdSetter = new
            TrainingMaxAnomalyThresholdSetter("
            maxAnomalyThrehsolder 1", maxMultiplierWarning,
            maxMultiplierAlarm);
277     mainLoop->setAnomalyThresholdSetter(
            anomalyThresholdSetter);
278
279     // setting anomaly threshold setter
280     AnomalyDetector *anomalyDetector = new
            DefaultAnomalyDetector("defaultAnomalyDetector 1");
281     mainLoop->setAnomalyDetector(anomalyDetector);
282
283     // ## Concept Drift Detection
284     // setting concept distance measure
285     ConceptDistanceMeasure *conceptDestanceMeasure = new
            FastDTWConceptDistanceMeasure("fastDTW Concept1");
286     mainLoop->setConceptDistanceMeasure(
            conceptDestanceMeasure);
287
288     // setting concept threshold setter
289     double alphaWarrning = 2;
290     double alphaAlarm = 3;
291     ConceptThresholdSetter *conceptThresholdSetter = new
            ConfidenceIntervalConceptThresholdSetter("Confidence
             Interval Concept Threshold Setter", alphaWarrning,
            alphaAlarm);
```

```
292     mainLoop->setConceptThresholdSetter(
            conceptThresholdSetter);
293
294     // setting concept drift detector
295     // ConceptDriftDetector *conceptDriftDetector = new
            NoConceptDriftDetector("NoCDD");
296     ConceptDriftDetector *conceptDriftDetector = new
            DynamicWindowConceptDriftDetector("CDD");
297     mainLoop->setConceptDriftDetector(conceptDriftDetector)
            ;
298
299
300     mainLoop->setProfiler(profiler);
301
302     mainLoop->setSharedMemory(sharedMemory);
303 }
304
305 void runLSTMCNnet(std::string inputFileName, std::string
        outputFileName){
306     /**
307      * Processes input files in NAB format.
308      * CSV files with header "timestamp,value,anomaly_score
             ,label,S(t)_reward_low_FP_rate,S(t)
             _reward_low_FN_rate,S(t)_standard"
309      *
310      */
311
312     // File pointer
313     std::fstream inputStream {inputFileName, std::ios::in};
             // open inputFileName
314     std::fstream outputStream {outputFileName, std::ios::
            out}; // open inputFileName
315
316     // Read the Data from the file
317     // as String Vector
318     std::vector<std::string> row;
319     std::string line, word, temp;
320     int lineNumber = 1;
321     bool correctHeader = false;
```

```
322
323     ROW writeROW;
324
325     while (!inputStream.eof()) {
326
327         row.clear();
328
329         // read an entire row and
330         // store it in a string variable 'line'
331         std::getline(inputStream, line);
332         // used for breaking words
333         std::stringstream stringStream(line);
334
335         if(line != ""){
336             // read every column data of a row and
337             // store it in a string variable, 'word'
338             while(std::getline(stringStream, word, ',')) {
339                 // add all the column data
340                 // of a row to a vector
341                 row.push_back(word);
342             }
343
344             // check if header it is in correct format
345             if(lineNumber == 1){// header
346                 if(row[0] == "timestamp" &&
347                     row[1] == "value" &&
348                     row[2] == "anomaly_score" &&
349                     row[3] == "label"){
350                         correctHeader = true;
351                     std::cout << "[main:runLSTMCNnet] Input
                            file header is correct !" << std::
                        endl;
352                 } else {
353                     std::cout << "[main:runLSTMCNnet] Input
                            file header is Incorrect !" << std
                        ::endl;
354                 }
355                 outputStream << "timestamp,value,
                        prediction_training,prediction,label,
```

```
                    warp_distance,threshold_training,
                    distance_threshold,positive_detection"
                    << std::endl;
356                 lineNumber++;
357             } else if(correctHeader){
358                 writeROW.timestamp = row[0];
359                 writeROW.value = std::stod(row[1]); // get
                        double
360                 writeROW.label = std::stoi(row[3]); // get
                        int
361                 lineNumber++;
362
363                 // Run MainLoop Tick
364                 historyBuffer->writeSafe(writeROW.value);
                        // wtite to history
365                 mainLoop->tick();
366                 historyBuffer->setTickedPrevious(true);
367
368                 // Get data and write to outputfile
369                 outputStream << writeROW.timestamp << ",";
370                 outputStream << writeROW.value << ",";
371
372                 writeROW.prediction_training = sharedMemory
                        ->profiler->training;
373
374                 // outputStream << std::setprecision(1);
375                 outputStream << (double)writeROW.
                        prediction_training << ",";
376
377                 if(sharedMemory->profiler->profile->index <
                        0){
378                     writeROW.prediction = 0.0;
379                 } else {
380                     writeROW.prediction = sharedMemory->
                            profiler->profile->data[sharedMemory
                            ->profiler->profile->index];
381                 }
382                 outputStream << (double)writeROW.prediction
                        << ",";
```

```
383                 outputStream << ((double)writeROW.label) <<
                        ",";
384
385                 if(sharedMemory->anomalyDetector->distance
                        ->index < 0){
386                     writeROW.warp_distance = 0.0;
387                 } else {
388                     writeROW.warp_distance = sharedMemory->
                            anomalyDetector->distance->data[
                            sharedMemory->anomalyDetector->
                            distance->index];
389                 }
390                 outputStream << (double)writeROW.
                        warp_distance << ",";
391
392                 writeROW.threshold_training = sharedMemory
                        ->anomalyDetector->training;
393                 if(sharedMemory->profiler->training){
394                     writeROW.threshold_training = false;
395                 }
396                 outputStream << (double)writeROW.
                        threshold_training << ",";
397
398                 writeROW.distance_threshold = sharedMemory
                        ->anomalyDetector->thresholdAlram;
399                 outputStream << (double)writeROW.
                        distance_threshold << ",";
400
401                 if(sharedMemory->anomalyDetector->alarm->
                        index < 0){
402                     writeROW.positive_detection = false;
403                 } else {
404                     writeROW.positive_detection =
                            sharedMemory->anomalyDetector->alarm
                            ->data[sharedMemory->anomalyDetector
                            ->alarm->index];
405                 }
406                 outputStream << (double)writeROW.
                        positive_detection ;
```

```
407
408                     outputStream << std::endl;;
409                     if (isnan(writeROW.prediction)){
410                         break;
411                     }
412                 }
413             }
414         }
415         std::cout << "[main] Done Processing file !" << std::
              endl;
416 }
```

Listing A.2: JSON format used to store parameters for each NAB data set used by Sherlock implementation of LSTMCNNnet, LSTMCNNcda and python implementation of ARMA, CNN, LSTM, and LSTMCNNkeras

```
1  {
2      "prediction_model": {
3          "learningRate": 0.003,
4          "model": {
5              "CNN": {
6                  "PoolingLayers": [
7                      {
8                          "poolH": 1,
9                          "poolW": 1
10                     }
11                 ],
12                 "matWidth": 30,
13                 "FullyConnectedLayers": [
14                     {
15                         "outputs": 40
16                     },
17                     {
18                         "outputs": 20
19                     },
20                     {
21                         "outputs": 1
22                     }
23                 ],
24                 "LayerOrder": [
```

```
                        "C",
                        "P",
                        "F",
                        "F",
                        "F"
                    ],
                    "matHeight": 2,
                    "targetC": 1,
                    "ConvolutionLayers": [
                        {
                            "stride": 1,
                            "filterSize": 2,
                            "filters": 1
                        }
                    ]
                },
                "lstmW": 0.5,
                "LSTM": {
                    "memCells": 10
                },
                "cnnW": 0.5
            },
            "trainingIterations": 10,
            "trainDataSize": 200,
            "numPredPoints": 1
        },
        "input_size": 1624,
        "dtw_window": 8
}
```

Listing A.3: LSTMCNNnet profiling method implemented Sherlock as a Profiler in Sherlock

```
#include "LSTMCNNetProfiler.h"

#include <iostream>
#include <algorithm>
#include <vector>
#include "LSTMCNNet.hpp"

```

```
 8 #include <iostream>
 9
10 // Constructors
11 LSTMCNnetProfiler::LSTMCNnetProfiler(std::string identifyer
       , ModelStruct *model, double lstmWeight, double
       cnnWeight)
12 {
13     this->identifyer = identifyer;
14     std::cout << "[LSTMCNnetProfiler] Constructing {" <<
          identifyer << "}" << std::endl;
15
16     this->modelStruct = model;
17     lstmW = lstmWeight;
18     cnnW = cnnWeight;
19
20     std::cout << "[LSTMCNnetProfiler] Done Constructing
          using given LSTMCNnet ModelStruct {" << identifyer
          << "}" << std::endl;
21 }
22
23 // Destroying
24 LSTMCNnetProfiler::~LSTMCNnetProfiler()
25 {
26     std::cout << "[LSTMCNnetProfiler] Destroying {" <<
          identifyer << "}" << std::endl;
27     delete predictionModel;
28     delete modelStruct;
29 }
30
31 void LSTMCNnetProfiler::init()
32 {
33     std::cout << "[LSTMCNnetProfiler] Initializing {" <<
          identifyer << "}" << std::endl;
34
35     if(modelStruct->numPredPoints == -1){
36         modelStruct->numPredPoints = sharedMemory->profiler
              ->OutWindowSize;
37     }
38
```

```
39
40      this->predictionModel = new LSTMCNNFCPredictionModel(
            modelStruct);
41
42      std::cout << "[LSTMCNnetProfiler] Done Initializing {"
            << identifyer << "}" << std::endl;
43  }
44
45  double LSTMCNnetProfiler::profile()
46  {
47      double profile = 0;
48      if(modelStruct->trainDataSize < modelStruct->matHeight*
            modelStruct->matWidth){
49          std::cout << "[LSTMCNnetProfiler:ERROR!] Not enogh
                training data to train model ! {" << identifyer
                << "}" << std::endl;
50      };
51      if( (modelStruct->trainDataSize + (modelStruct->
            matHeight*modelStruct->matWidth-1)) > sharedMemory->
            history->index){
52          if(verbose)
53          std::cout << "[LSTMCNnetProfiler] Collecting data
                to train ! {" << identifyer << "}" << std::endl
                ;
54          sharedMemory->profiler->trainingDataCollected += 1;
55      } else {
56          if((modelStruct->trainDataSize + (modelStruct->
                matHeight*modelStruct->matWidth-1)) >
                sharedMemory->profiler->trainingDataCollected){
57              if(verbose)
58              std::cout << "[LSTMCNnetProfiler] Collecting
                    data to train for new concept ! {" <<
                    identifyer << "}" << std::endl;
59              sharedMemory->profiler->trainingDataCollected
                    += 1;
60          } else if(sharedMemory->profiler->training) {
61              std::cout << "[LSTMCNnetProfiler] Done
                    Collecting data to train. Training ! {" <<
                    identifyer << "}" << std::endl;
```

```
62
63              // Training the networks in the model
64              predictionModel->train(sharedMemory->history->
                    data, sharedMemory->history->index);
65              sharedMemory->profiler->training = false;
66
67              std::cout << "[LSTMCNnetProfiler] Done Training
                    ! {" << identifyer << "}"  << std::endl;
68          }
69
70
71              // parameters for model outputs
72              int predictions = 1;
73              int abs = 1;
74
75              double *inWondow = new double[sharedMemory->
                    profiler->inWindowSize];
76
77              for(int i=sharedMemory->profiler->inWindowSize
                    -1; i >= 0; i--){
78                  inWondow[sharedMemory->profiler->
                        inWindowSize -1 - i] = sharedMemory->
                        history->data[sharedMemory->history->
                        index - i];
79              };
80
81              // getting predicted time series data points
82              profile = predictionModel->predict(predictions,
                    inWondow, sharedMemory->profiler->
                    inWindowSize,lstmW, cnnW, abs)[0];
83
84              delete inWondow;
85              sharedMemory->profiler->profiledCount++;
86
87      }
88
89      return profile;
90  }
```

Listing A.4: FastDTW implementation

```
1  #include "DTW.hpp"
2
3  DTW::DTW() { }
4
5  DTW::~DTW() { }
6
7  double DTW::getSimilarity(Eigen::VectorXd inputVector1,
       Eigen::VectorXd inputVector2) {
8
9      Eigen::MatrixXd matrix(inputVector1.size(),
           inputVector2.size());
10     matrix(0,0) = 0;
11     for (int i = 1; i < inputVector1.size(); i++) {
12         matrix(i,0) = std::abs(inputVector1(i)-inputVector2
               (0)) + matrix(i-1,0);
13     }
14     for (int j = 1; j < inputVector2.size(); j++) {
15         matrix(0,j) = std::abs(inputVector1(0)-inputVector2
               (j)) + matrix(0,j-1);
16     }
17     for (int i = 1; i < inputVector1.size(); i++) {
18         for (int j = 1; j < inputVector2.size(); j++) {
19             matrix(i,j) = std::abs(inputVector1(i)-
                   inputVector2(j)) + getMinimum(matrix(i-1,j),
                    matrix(i,j-1), matrix(i-1,j-1));
20         }
21     }
22
23     double warpDistance = 0;
24     double minimum, a, b, c;
25     int i = inputVector1.size()-1;
26     int j = inputVector2.size()-1;
27
28     warpDistance += matrix(i,j);
29     while((i!=0) || (j!=0)) {
30
31         if ((i-1) >= 0 ) a = matrix(i-1,j);
32         else a = std::numeric_limits<double>::max();
```

```
33          if ((j-1) >= 0 ) b = matrix(i, j-1);
34          else b = std::numeric_limits<double>::max();
35          if (((i-1) >= 0 ) && ((j-1) >= 0 )) c = matrix(i-1,
               j-1);
36          else c = std::numeric_limits<double>::max();
37
38          minimum = getMinimum(a,b,c);
39          warpDistance += minimum;
40          if (minimum == a) i--;
41          else if (minimum == b) j--;
42          else if (minimum == c) { i--; j--;}
43
44      }
45
46      return warpDistance;
47  }
48
49  double DTW::fastDTW(Eigen::VectorXd inputVector1, Eigen::
       VectorXd inputVector2, int radius) {
50
51      int minimumSize = radius + 2;
52      if ((inputVector1.size() <= minimumSize) || (
           inputVector2.size() <= minimumSize)) {
53          // Terminiation step
54          return getSimilarity(inputVector1, inputVector2);
55      } else {
56          // Recursive step
57          int tmpSize1 = inputVector1.size()/2, tmpSize2 =
               inputVector2.size();
58          Eigen::VectorXd tmpVector1 = inputVector1.head(
               tmpSize1);
59          Eigen::VectorXd tmpVector2 = inputVector2.head(
               tmpSize2);
60          double lowPath = fastDTW(tmpVector1, tmpVector2,
               radius);
61      }
62
63      return 0.0;
64  }
```

```
65
66 double DIW::getMinimum(double x1, double x2, double x3) {
67     return std::min(std::min(x1, x2), x3);
68 }
```

Listing A.5: Concept drift detection method [2] implemented on Sherlock as ConceptThresholdSetter

```
1  #include "ConfidenceIntervalConceptThresholdSetter.h"
2
3  #include <iostream>
4
5  // Constructors
6  ConfidenceIntervalConceptThresholdSetter::
       ConfidenceIntervalConceptThresholdSetter(std::string
       identifyer)
7  {
8      this->identifyer = identifyer;
9      std::cout << "[ConfidenceIntervalConceptThresholdSetter
           ] Constructing {" << identifyer << "}" << std::endl
           ;
10 }
11
12 ConfidenceIntervalConceptThresholdSetter::
       ConfidenceIntervalConceptThresholdSetter(std::string
       identifyer, double warningAlpha, double alarmAlpha){
13     this->identifyer = identifyer;
14     this->warningAlpha = warningAlpha;
15     this->alarmAlpha = alarmAlpha;
16
17
18     std::cout << "[ConfidenceIntervalConceptThresholdSetter
           ] Constructing {" << identifyer << "}" << std::endl
           ;
19 }
20
21 // Destroying
22 ConfidenceIntervalConceptThresholdSetter::~
       ConfidenceIntervalConceptThresholdSetter()
23 {
```

```
24      std::cout << "[ConfidenceIntervalConceptThresholdSetter
            ] Destroying {" << identifyer << "}"  << std::endl;
25  }
26
27  void ConfidenceIntervalConceptThresholdSetter::init()
28  {
29      std::cout << "[ConfidenceIntervalConceptThresholdSetter
            ] Initializing {" << identifyer << "}"  << std::endl
            ;
30
31      windowSize = sharedMemory->conceptDriftDetector->
            distance->size - 1;
32
33      warningThreshold = -1;
34      alarmThreshold = -1;
35
36      std::cout << "[ConfidenceIntervalConceptThresholdSetter
            ] Done initializing {" << identifyer << "}"  << std
            ::endl;
37  }
38
39  double ConfidenceIntervalConceptThresholdSetter::
        calculateMean()
40  {
41   // calculates Mean for distance [0 n-1 , n] 0 to n-1, n is
         left out
42      double mean = 0;
43      for(int i=0; i < this->windowSize; i++){
44          mean += sharedMemory->conceptDriftDetector->
                distance->data[sharedMemory->
                conceptDriftDetector->distance->size -2 -i];
45      }
46      mean = mean/this->windowSize;
47      return mean;
48
49  };
50
51  double ConfidenceIntervalConceptThresholdSetter::
        calculateSD(double mean)
```

```
52 {
53  // calculates Standard deviation for distance [0 n-1 , n]
       0 to n-1, n is left out
54
55      double finalSD = 0;
56      double SD;
57      for(int i=0; i < this->windowSize; i++){
58          SD += sharedMemory->conceptDriftDetector->distance
                ->data[sharedMemory->conceptDriftDetector->
                distance->size -2 -i];
59          SD = mean - SD;
60          if(SD < 0){
61              SD = 0 - SD;
62          }
63          finalSD += SD;
64      }
65      finalSD = finalSD/this->windowSize;
66      return finalSD;
67 };
68
69
70 double ConfidenceIntervalConceptThresholdSetter::
       getWarningThreshold()
71 {
72      //for distance [0 n-1 , n] 0 to n-1 used for
           calculateing minSD and minMean, n is left out
73      double SD;
74      double mean;
75      firstMeanAndSD = sharedMemory->conceptDriftDetector->
           training;
76      if((!sharedMemory->profiler->training) &&
77          (sharedMemory->profiler->profiledCount +
              sharedMemory->conceptDriftDetector->inWindowSize
              ) >= windowSize &&
78          sharedMemory->conceptDriftDetector->distance->index
               +2 >= windowSize){
79
80          mean = this->calculateMean();
81          SD = this->calculateSD(mean);
```

```
82
83              if(firstMeanAndSD){
84                  this->minMean = mean;
85                  this->minSD = SD;
86                  this->firstMeanAndSD = false;
87              }
88              if(mean < this->minMean){
89                  this->minMean = mean;
90              }
91              if(SD < this->minSD){
92                  this->minSD = SD;
93              }
94
95              alarmThreshold = this->minMean + (this->alarmAlpha
                    * this->minSD);
96              warningThreshold = this->minMean + (this->
                    warningAlpha * this->minSD);
97
98
99              sharedMemory->conceptDriftDetector->training =
                    false;
100
101         }
102
103
104         return warningThreshold;
105 }
106
107 double ConfidenceIntervalConceptThresholdSetter::
        getAlarmThreshold()
108 {
109     return alarmThreshold;
110 }
```

Listing A.6: Concept drift detection method [2] implemented on Sherlock as ConceptDriftDetector

```
1 #include "DynamicWindowConceptDriftDetector.h"
2
3 #include <iostream>
```

```
4
5  // Constructors
6  DynamicWindowConceptDriftDetector::
       DynamicWindowConceptDriftDetector(std::string identifyer
       )
7  {
8      this->identifyer = identifyer;
9      std::cout << "[DynamicWindowConceptDriftDetector]
           Constructing {" << identifyer << "}" << std::endl;
10 }
11
12 // Destroying
13 DynamicWindowConceptDriftDetector::~
       DynamicWindowConceptDriftDetector()
14 {
15     std::cout << "[DynamicWindowConceptDriftDetector]
           Destroying {" << identifyer << "}" << std::endl;
16 }
17
18 void DynamicWindowConceptDriftDetector::init()
19 {
20     std::cout << "[DynamicWindowConceptDriftDetector]
           Initializing {" << identifyer << "}" << std::endl;
21
22     windowSize = sharedMemory->conceptDriftDetector->
           distance->size - 1;
23
24     std::cout << "[DynamicWindowConceptDriftDetector] Done
           initializing {" << identifyer << "}" << std::endl;
25 }
26
27 double DynamicWindowConceptDriftDetector::calculateMean()
28 {
29  // calculates Mean for distance [0 n-1 , n] 1 to n, 0 is
       left out
30     double mean = 0;
31     for(int i=0; i < this->windowSize; i++){
32         mean += sharedMemory->conceptDriftDetector->
               distance->data[sharedMemory->
```

```
                conceptDriftDetector->distance->size -1 -i];
33      }
34      mean = mean/this->windowSize;
35      return mean;
36
37  };
38
39  double DynamicWindowConceptDriftDetector::calculateSD(
        double mean)
40  {
41   // calculates Standard deviation for distance [0 n-1 , n]
        1 to n, 0 is left out
42      double finalSD = 0;
43      double SD;
44      for(int i=0; i < this->windowSize; i++){
45          SD += sharedMemory->conceptDriftDetector->distance
                ->data[sharedMemory->conceptDriftDetector->
                distance->size -1 -i];
46          SD = mean - SD;
47          if(SD < 0){
48              SD = 0 - SD;
49          }
50          finalSD += SD;
51      }
52      finalSD = finalSD/this->windowSize;
53      return finalSD;
54  };
55
56
57  int DynamicWindowConceptDriftDetector::getNewConceptCount()
58  {
59      if(!sharedMemory->conceptDriftDetector->warning->data[
            sharedMemory->conceptDriftDetector->warning->index])
            {
60          return -1;
61      }
62      int newConceptCount = 1;
63      int i = sharedMemory->conceptDriftDetector->warning->
            index;
```

```
64      while(i >= 0 &&
65          sharedMemory->conceptDriftDetector->warning->data[i
                ]){
66              newConceptCount += 1;
67              i -= 1;
68          }
69      return newConceptCount;
70  }
71
72  int DynamicWindowConceptDriftDetector::detect()
73  {
74      int newConceptCount = -1;
75      double mean = this->calculateMean();
76      double SD = this->calculateSD(mean);
77      double metric = mean + SD;
78
79      if(metric >= sharedMemory->conceptDriftDetector->
            thresholdAlram){
80          // Alarm
81          newConceptCount = this->getNewConceptCount();
82          sharedMemory->conceptDriftDetector->warning->
                enQueue(true);
83          sharedMemory->conceptDriftDetector->alarm->enQueue(
                true);
84      } else if(metric >= sharedMemory->conceptDriftDetector
            ->thresholdWarning){
85          // Warnning
86          sharedMemory->conceptDriftDetector->warning->
                enQueue(true);
87          sharedMemory->conceptDriftDetector->alarm->enQueue(
                false);
88      } else {
89          // None
90          sharedMemory->conceptDriftDetector->warning->
                enQueue(false);
91          sharedMemory->conceptDriftDetector->alarm->enQueue(
                false);
92
93      }
```

```
94
95      return newConceptCount;
96  }
```

Listing A.7: ARMA implementation in python

```
1  import model_helpers
2  import numpy as np
3  import statsmodels.api as sm
4  import warnings
5
6  class arma:
7      def __init__(self, data, training_ratio, ar_max=3,
           ma_max=3):
8          self.ar_max = ar_max # maximum AR parameter
9          self.ma_max = ma_max # maximum MA parameter
10         training_data_end = int(len(data)*training_ratio)
11         testing_data_start = training_data_end
12         self.training_data = data[:training_data_end].
               astype('float64')
13         self.testing_data = data[testing_data_start:].
               astype('float64')
14         self.params = None
15
16     def train(self):
17         ar_max = self.ar_max
18         ma_max = self.ma_max
19         params = np.zeros((ar_max+1, ma_max+1))
20         y = self.training_data
21
22         for ar in range(0,ar_max+1):
23             for ma in range(0,ma_max+1):
24                 try:
25                     with warnings.catch_warnings():
26                         warnings.simplefilter("ignore")
27                         arma_model = sm.tsa.ARMA(y, order=(
                               ar,ma))
28                         arma_result = arma_model.fit(trend=
                               'c', disp=-1)
29                     y_prediction = arma_result.predict()
```

```
30                      mse = model_helpers.MSE(y,y_prediction)
31                  except ValueError:
32                      mse = float("inf")
33                  except np.linalg.LinAlgError:
34                      mes = float("inf")
35                  params[ar][ma] = mse
36                  print("("+str(ar)+","+str(ma)+") : " +str(
                        mse))
37
38          minAR, minMA = model_helpers.get_min_matrix(params)
39          print("Best model : ("+str(minAR)+","+str(minMA)+")
                | with MSE = "+str(params[minAR][minMA]))
40          self.params = params
41          self.ar = minAR
42          self.ma = minMA
43          return minAR,minMA
44
45      def get_output(self):
46          ar = self.ar
47          ma = self.ma
48          y = self.testing_data
49          with warnings.catch_warnings():
50              warnings.simplefilter("ignore")
51              try:
52                  arma_model = sm.tsa.ARMA(y, order=(ar,ma))
53                  arma_result = arma_model.fit(trend='c',
                        disp=-1)
54              except ValueError:
55                  print("model : ("+str(ar)+","+str(ma)+")
                        failed ! getting next best model")
56                  self.params[ar][ma] = float("inf")
57                  minAR, minMA = model_helpers.get_min_matrix
                        (self.params)
58                  print("Best model : ("+str(minAR)+","+str(
                        minMA)+") | with MSE = "+str(self.params
                        [minAR][minMA]))
59                  self.ar = minAR
60                  self.ma = minMA
61                  return self.get_output()
```

```
62                 except np.linalg.LinAlgError:
63                     print("model : ("+str(ar)+","+str(ma)+")
                           failed ! getting next best model")
64                     self.params[ar][ma] = float("inf")
65                     minAR, minMA = model_helpers.get_min_matrix
                           (self.params)
66                     print("Best model : ("+str(minAR)+","+str(
                           minMA)+") | with MSE = "+str(self.params
                           [minAR][minMA]))
67                     self.ar = minAR
68                     self.ma = minMA
69                     return self.get_output()
70
71             y_prediction = arma_result.predict()
72             return ar, ma, np.array(y_prediction)
```

Listing A.8: CNN implementation in python

```
1  import numpy as np
2  from keras.models import Sequential
3  from keras.layers import Dense, Conv1D, MaxPooling1D,
       Flatten
4  from keras import optimizers
5
6  class cnn:
7      def __init__(self, data, epochs, batch_size,
           training_ratio, sequance_length,CL1filters=1,
           CL1kernal_size=2, CL1strides=1, PL1pool_size=1,
           DL1units=20, DL2units=5, DL3units=1, learningRate
           =0.001):
8          self.CL1filters = CL1filters
9          self.CL1kernal_size = CL1kernal_size
10         self.CL1strides = CL1strides
11         self.sequance_length = sequance_length
12         self.PL1pool_size = PL1pool_size
13         self.DL1units = DL1units
14         self.DL2units = DL2units
15         self.DL3units = DL3units
16         self.epochs = epochs
17         self.batch_size = batch_size
```

```
18          self.learningRate = learningRate
19          training_data_end = int(training_ratio*len(data))
20          testing_data_start = training_data_end -
                sequance_length
21          training_data = data[:training_data_end]
22          testing_data = data[testing_data_start:]
23
24          self.training_feature_set = []
25          self.labels = []
26          for i in range(sequance_length, len(training_data))
                :
27              self.training_feature_set.append(training_data[
                    i-sequance_length:i])
28              self.labels.append(training_data[i])
29          self.labels = np.array(self.labels)
30          self.training_feature_set = np.array(self.
                training_feature_set)
31          self.training_feature_set = np.reshape(self.
                training_feature_set, (self.training_feature_set
                .shape[0], self.training_feature_set.shape[1],
                1))
32
33          self.testing_feature_set = []
34          for i in range(sequance_length, len(testing_data)):
35              self.testing_feature_set.append(testing_data[i-
                    sequance_length:i])
36          self.testing_feature_set = np.array(self.
                testing_feature_set)
37          self.testing_feature_set = np.reshape(self.
                testing_feature_set, (self.testing_feature_set.
                shape[0], self.testing_feature_set.shape[1], 1))
38
39      def train(self):
40          self.model = Sequential()
41          self.model.add(Conv1D(filters=self.CL1filters,
                kernel_size=self.CL1kernal_size,strides=self.
                CL1strides, input_shape=(self.sequance_length,
                1)))
42          self.model.add(MaxPooling1D(pool_size=self.
```

```
            PL1pool_size))
43          self.model.add(Flatten())
44          self.model.add(Dense(units = self.DL1units))
45          self.model.add(Dense(units = self.DL2units))
46          self.model.add(Dense(units = self.DL3units))
47
48          adam = optimizers.Adam(lr=self.learningRate)
49
50          self.model.compile(optimizer = adam, loss = '
                mean_squared_error', metrics=["mse"])
51          self.model.fit(self.training_feature_set, self.
                labels, batch_size=self.batch_size, epochs=self.
                epochs)
52
53      def get_output(self):
54          predictions = self.model.predict(self.
                testing_feature_set)
55          ret_prediction = np.zeros(predictions.shape[0])
56          for i in range(0, predictions.shape[0]):
57              ret_prediction[i] = predictions[i][0]
58          return ret_prediction
```

Listing A.9: LSTM implementation in python

```
1  import numpy as np
2  from keras.models import Sequential
3  from keras.layers import Dense, LSTM
4  from keras import optimizers
5
6
7  class lstm:
8      def __init__(self, data,  epochs, batch_size,
           training_ratio, sequance_length, lstmCells=10,
           learningRate=0.001):
9          self.lstmCells = lstmCells
10         self.sequance_length = sequance_length
11         self.epochs = epochs
12         self.batch_size = batch_size
13         self.learningRate = learningRate
14         training_data_end = int(len(data)*training_ratio)
```

```
15          testing_data_start = training_data_end -
               sequance_length
16          training_data = data[:training_data_end]
17          testing_data = data[testing_data_start:]
18          self.training_feature_set = []
19          self.labels = []
20          for i in range(self.sequance_length,len(
               training_data)):
21              self.training_feature_set.append(training_data[
                   i-self.sequance_length:i])
22              self.labels.append(training_data[i])
23          self.labels = np.array(self.labels)
24          self.training_feature_set = np.array(self.
               training_feature_set)
25          self.training_feature_set = np.reshape(self.
               training_feature_set, (self.training_feature_set
               .shape[0], self.training_feature_set.shape[1],
               1))
26
27          self.testing_feature_set = []
28          for i in range(self.sequance_length, len(
               testing_data)):
29              self.testing_feature_set.append(testing_data[i-
                   self.sequance_length:i])
30          self.testing_feature_set = np.array(self.
               testing_feature_set)
31          self.testing_feature_set = np.reshape(self.
               testing_feature_set, (self.testing_feature_set.
               shape[0], self.testing_feature_set.shape[1], 1))
32
33      def train(self):
34          self.model = Sequential()
35          self.model.add(LSTM(units=self.lstmCells))
36          self.model.add(Dense(units=1))
37
38          adam = optimizers.Adam(lr=self.learningRate)
39
40          self.model.compile(optimizer = adam, loss = '
               mean_squared_error', metrics=["mse"])
```

```
41          self.model.fit(self.training_feature_set, self.
               labels, epochs = self.epochs, batch_size = self.
               batch_size)
42
43
44      def get_output(self):
45          predictions = self.model.predict(self.
               testing_feature_set)
46          ret_prediction = np.zeros(predictions.shape[0])
47          for i in range(0, predictions.shape[0]):
48              ret_prediction[i] = predictions[i][0]
49          return ret_prediction
```

Listing A.10: LSTMCNNkeras model implementation in python

```
1  import numpy as np
2  from keras.models import Model
3  from keras.layers import Dense, Input, Add, Layer, LSTM,
      Conv1D, MaxPooling1D, Flatten
4  from keras import optimizers
5
6  # Define custom layer for weighted sum
7  class WeightedSum(Layer):
8      def __init__(self, weight1, weight2, **kwargs):
9          self.weight1 = weight1
10         self.weight2 = weight2
11         super(WeightedSum, self).__init__(**kwargs)
12     def call(self, model_outputs):
13         return self.weight1 * model_outputs[0] + (self.
               weight2) * model_outputs[1]
14     def compute_output_shape(self, input_shape):
15         return input_shape[0]
16
17 class LSTMCNNkeras:
18     def __init__(self, data,  epochs, batch_size,
           training_ratio ,sequance_length, lstmCells=10,
           CL1filters=1, CL1kernal_size=2, CL1strides=1,
           PL1pool_size=1, CNNDL1units=20, CNNDL2units=5,
           CNNDL3units=1,lstmWeight=0.5, cnnWeight=0.5,
           learningRate=0.001):
```

```
19          self.lstmCells = lstmCells
20
21          self.CL1filters = CL1filters
22          self.CL1kernal_size = CL1kernal_size
23          self.CL1strides = CL1strides
24          self.PL1pool_size = PL1pool_size
25          self.CNNDL1units = CNNDL1units
26          self.CNNDL2units = CNNDL2units
27          self.CNNDL3units = CNNDL3units
28
29          self.lstmWeight = lstmWeight
30          self.cnnWeight = cnnWeight
31
32          self.learningRate = learningRate
33
34          self.sequance_length = sequance_length
35          self.epochs = epochs
36          self.batch_size = batch_size
37          training_data_end = int(len(data)*training_ratio)
38          testing_data_start = training_data_end -
               sequance_length
39          training_data = data[:training_data_end]
40          testing_data = data[testing_data_start:]
41          self.training_feature_set = []
42          self.labels = []
43          for i in range(self.sequance_length,len(
               training_data)):
44              self.training_feature_set.append(training_data[
                   i-self.sequance_length:i])
45              self.labels.append(training_data[i])
46          self.labels = np.array(self.labels)
47          self.training_feature_set = np.array(self.
               training_feature_set)
48          self.training_feature_set = np.reshape(self.
               training_feature_set, (self.training_feature_set
               .shape[0], self.training_feature_set.shape[1],
               1))
49
50          self.testing_feature_set = []
```

```
51          for i in range(self.sequance_length, len(
                testing_data)):
52              self.testing_feature_set.append(testing_data[i-
                    self.sequance_length:i])
53          self.testing_feature_set = np.array(self.
                testing_feature_set)
54          self.testing_feature_set = np.reshape(self.
                testing_feature_set, (self.testing_feature_set.
                shape[0], self.testing_feature_set.shape[1], 1))
55
56      def train(self):
57
58          input_shape = Input(shape=(self.sequance_length, 1)
                )
59
60          #lstm
61          lstm = LSTM(units=self.lstmCells)(input_shape)
62          lstmdense = Dense(units = 1)(lstm)
63
64          #cnn
65          cnn = Conv1D(filters=self.CL1filters, kernel_size=
                self.CL1kernal_size,strides=self.CL1strides,
                input_shape=(self.sequance_length, 1))(
                input_shape)
66          cnnpooling = MaxPooling1D(pool_size=self.
                PL1pool_size)(cnn)
67          cnnflaten = Flatten()(cnnpooling)
68          cnndense1 = Dense(units = self.CNNDL1units)(
                cnnflaten)
69          cnndense2 = Dense(units = self.CNNDL2units)(
                cnndense1)
70          cnndense3 = Dense(units = self.CNNDL3units)(
                cnndense2)
71
72          #combinantion layer
73          out = WeightedSum(self.lstmWeight, self.cnnWeight)
                ([lstmdense, cnndense3])
74
75          self.model = Model(input_shape, out)
```

```
76
77          adam = optimizers.Adam(lr=self.learningRate)
78
79          self.model.compile(optimizer = adam, loss = '
                mean_squared_error', metrics=["mse"])
80          self.model.fit(self.training_feature_set, self.
                labels, epochs = self.epochs, batch_size = self.
                batch_size)
81
82      def get_output(self):
83          predictions = self.model.predict(self.
                testing_feature_set)
84          ret_prediction = np.zeros(predictions.shape[0])
85          for i in range(0, predictions.shape[0]):
86              ret_prediction[i] = predictions[i][0]
87          return ret_prediction
```