

2021

## Multivariate Time Series Classification of Sensor Data from an Industrial Drying Hopper: A Deep Learning Approach

Md Mushfiqur Rahman  
mr0143@mix.wvu.edu

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>



Part of the [Industrial Engineering Commons](#)

---

### Recommended Citation

Rahman, Md Mushfiqur, "Multivariate Time Series Classification of Sensor Data from an Industrial Drying Hopper: A Deep Learning Approach" (2021). *Graduate Theses, Dissertations, and Problem Reports*. 8309. <https://researchrepository.wvu.edu/etd/8309>

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact [researchrepository@mail.wvu.edu](mailto:researchrepository@mail.wvu.edu).

2021

## Multivariate Time Series Classification of Sensor Data from an Industrial Drying Hopper: A Deep Learning Approach

Md Mushfiqur Rahman

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>



Part of the [Industrial Engineering Commons](#)

---

**Multivariate Time Series Classification of Sensor Data from an Industrial Drying Hopper: A Deep Learning Approach**

**Md Mushfiqur Rahman**

Thesis submitted to the Benjamin M. Statler College of Engineering and Mineral Resources at  
West Virginia University  
in partial fulfillment of the requirements for the degree of

**Master of Science  
in  
Industrial Engineering**

Thorsten Wuest, PhD, Committee Chair

Kenneth Currie, PhD

Behrooz Kamali, PhD

Department of Industrial and Management Systems Engineering

Morgantown, West Virginia

2021

Keywords: Predictive Maintenance, Multivariate Time Series, Classification, Data labeling,  
Imbalanced Data, Deep Learning

**Copyright 2021 Md Mushfiqur Rahman**

## **Abstract**

### **Multivariate Time Series Classification of Sensor Data from an Industrial Drying Hopper: A Deep Learning Approach**

**Md Mushfiqur Rahman**

In recent years, the advancement of industry 4.0 and smart manufacturing has made a large number of industrial process data attainable with the use of sensors installed in the machineries. This thesis proposes an experimental predictive maintenance framework for an industrial drying hopper so that it can detect any unusual event in the hopper which reduces the risk of erroneous fault diagnosis in the manufacturing shop floor. The experimental framework uses Deep Learning (DL) algorithms in order to classify Multivariate Time Series (MTS) data into two categories- failure or unusual events and regular events, thus formulating the problem as binary classification.

As classification is a supervised learning technique, any DL algorithm needs labeled data for classification. Moreover, raw data extracted from the sensors contain missing values. Therefore, necessary preprocessing is performed to make it usable for DL algorithms and the dataset is self-labeled after defining two categories precisely. To tackle the imbalanced data issue, data balancing techniques like Ensemble Learning with undersampling and Synthetic Minority Oversampling Technique (SMOTE) are used. Moreover, along with DL algorithms like Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM), Machine Learning (ML) algorithms like Support Vector Machine (SVM), K Nearest Neighbor (KNN), etc. have also been used to perform a comparative analysis on the result obtained from these algorithms. The result shows that CNN is arguably the best algorithm for classifying this dataset into two categories and outperforms other traditional approaches as well as deep learning algorithms.

## **Acknowledgements**

I would like to express my unbound respect and thankfulness to my research supervisor Dr. Thorsten Wuest for his proper guidance and support during the span of this research. It was him who motivated me to incite my own aptitude for knowing the unknown, encouraged me to the utmost and gave rise to the confidence in me for completing this thesis.

I am also grateful to the committee members Dr. Kenneth Currie and Dr. Behrooz Kamali for their valuable insights and feedback over the course of this thesis.

I wish to thank my parents, without whose prayers and constant support, I could never reach this stage of my life.

## Table of Contents

List of Figures .....	vi
List of Tables .....	viii
1 Introduction.....	1
1.1 General Introduction .....	1
1.2 Background .....	3
1.3 Objectives and Scopes.....	4
1.4 Outline of methodology .....	4
1.5 Organization of the Thesis .....	5
2 Literature Review.....	6
2.1 Traditional Algorithms.....	6
2.2 Deep Learning Approaches .....	8
2.3 Data Labeling .....	11
3 Methodology.....	12
3.1 Data Exploration and Preprocessing .....	12
3.1.1 Handling Missing Values.....	14
3.1.2 Data labeling .....	15
3.1.3 Characteristics of the labelled dataset.....	23
3.2 Solution Approach.....	31
3.2.1 Artificial Neural Network .....	31
3.2.2 Convolutional Neural network (CNN).....	35
3.2.3 Recurrent Neural Network.....	38
3.2.4 Combination of CNN and LSTM .....	42
3.2.5 Machine Learning Algorithms.....	43
3.2.6 K-Nearest neighbor (KNN).....	46
3.2.7 Performance Measure .....	47
3.2.8 System specification .....	48
4 Experimental Results and Discussion.....	50
4.1 Experimental setup.....	50
4.2 Hyperparameter Tuning .....	52
4.3 Result.....	54
4.3.1 Ensemble Learning (CNN) .....	54
4.3.2 Ensemble Method (LSTM) .....	56
4.3.3 Ensemble Learning (CNN-LSTM) .....	58
4.3.4 SMOTE.....	60

## Table of Contents

4.3.5	Machine learning algorithms .....	62
4.3.6	Summary .....	62
4.4	Discussion .....	64
4.4.1	Event definition and subsequence extraction.....	64
4.4.2	Data imbalance issue.....	65
4.4.3	Result interpretation.....	67
5	Conclusion and Future Work.....	69
	References.....	70
	Appendix.....	78

**List of Figures**

Figure 1.1: Temperature Profiles ..... 3

Figure 2.1: Two methods of calculating DTW distance[37] ..... 6

Figure 2.2: Deep learning overview for time series classification[20] ..... 9

Figure 2.3: MDDNN model architecture[61] ..... 10

Figure 3.1: Raw dataset in CSV format ..... 12

Figure 3.2: Temperature profile obtained from primarily preprocessed data..... 13

Figure 3.3: Missing values in the primarily processed dataset ..... 13

Figure 3.4: Missing values in the temperature profile ..... 14

Figure 3.5: Missing value imputation ..... 14

Figure 3.6: Startup Procedure[36]..... 15

Figure 3.7:Cleaning Cycle[36]..... 15

Figure 3.8: Conveying Issue[36]..... 16

Figure 3.9: Event..... 17

Figure 3.10: Variation of an event ..... 17

Figure 3.11: Definition of an event..... 19

Figure 3.12: Labelled Data ..... 21

Figure 3.13: Example of an event and a non-event ..... 21

Figure 3.14: Dataset Statistics..... 23

Figure 3.15: A simple visualization of nullity by column ..... 23

Figure 3.16: Nullity matrix ..... 24

Figure 3.17: Statistical summary of Hopper 1 hopper outlet temperature..... 24

Figure 3.18: Quantile and descriptive statistics of H1HOT..... 24

Figure 3.19: Common values and Extreme values of Minimum and Maximum of H1HOT .. 25

Figure 3.20: Binary classification labeling ..... 25

Figure 3.21: Change of distribution after data normalization..... 27

Figure 3.22: Data normalization ..... 28

Figure 3.23: Undersampling and oversampling[72] ..... 29

Figure 3.24: SMOTE[76]..... 30

Figure 3.25: Ensemble method[77]..... 31

Figure 3.26: Basic Structure of a Neural Network [79]..... 32

Figure 3.27: Activation Function [80] ..... 33

Figure 3.28: MLP with one hidden layer[82] ..... 35

Figure 3.29: Multi channel Deep CNN application on time series [32] ..... 36

Figure 3.30: CNN for time series classification[83] ..... 36

Figure 3.31: Different types of RNN architecture[84]..... 38

Figure 3.32: Computational Graph of RNN [85]..... 39

Figure 3.33: Back propagation through time[87] ..... 39

Figure 3.34: Vanishing and Exploding Gradient [89] ..... 40

Figure 3.35: LSTM structure[31], [89] ..... 41

Figure 3.36: CNN LSTM architecture[91] ..... 42

Figure 3.37: Support Vector Machine [94]..... 44

Figure 3.38: Optimizing hyperplanes [95]..... 44

Figure 3.39: Decision Tree[97]..... 45

Figure 3.40: Random Forest[98]..... 46

Figure 3.41: KNN [100]..... 47



## List of Figures

Figure 3.42: Confusion Matrix .....	48
Figure 4.1: Confusion matrix and classification report of imabalanced dataset (CNN).....	50
Figure 4.2: Hyperparameter tuning.....	53
Figure 4.3: accuracy vs. epoch and loss vs. epoch.....	53
Figure 4.4: CNN framework.....	54
Figure 4.5: Result summary of CNN (Ensemble Learning) .....	55
Figure 4.6: Confusion matrix and classification report of approach 3, run 1 .....	55
Figure 4.7: best undersamples in each experimental run .....	55
Figure 4.8: best learning curves (CNN).....	56
Figure 4.9: LSTM framework and summary .....	57
Figure 4.10: Result summary of LSTM.....	57
Figure 4.11: Confusion matrix and classification report of approach 1, run 7 .....	57
Figure 4.12: Best undersamples in each experimental run (LSTM).....	57
Figure 4.13: Best learning curves (LSTM) .....	58
Figure 4.14: CNN-LSTM framework and summary .....	58
Figure 4.15: Result summary of CNN-LSTM .....	59
Figure 4.16: Confusion matrix and classification report of appr. 1, run 4 and appr. 2, run 1 ..	59
Figure 4.17: Best undersamples in each experimental run (CNN- LSTM) .....	59
Figure 4.18: Best learning curves (CNN-LSTM) .....	60
Figure 4.19: Result summary of SMOTE .....	61
Figure 4.20: Confusion Matrix and Classification report (CNN, run 3).....	61
Figure 4.21: Learning curves (SMOTE).....	61
Figure 4.22: Result summary of ML algorithms .....	62
Figure 4.23: Learning curves with high fluctation during convergence .....	66

**List of Tables**

Table 3.1: Event durations ..... 19  
Table 3.2: Sliding Window Algorithm ..... 22  
Table 3.3: Histograms of the temperature zones ..... 26  
Table 3.4: Python libraries ..... 49  
Table 4.1: Training and Test Examples ..... 50  
Table 4.2: List of hyperparameters ..... 52  
Table 4.3: values considered for hyperparameters ..... 52  
Table 4.4: Initial values for hyperparameters ..... 52  
Table 4.5: Result summary (average result in ten runs) ..... 62  
Table 4.6: Result summary (Best result in ten runs)..... 63

## 1 Introduction

### 1.1 General Introduction

In recent years, the advancement of smart manufacturing – the merger of information technology and operational technology, has made the collection and processing of large number of data industrial process data attainable. These collection of data which is referred as big data is often an interchangeable term with artificial intelligence (AI) as big data uses various type of analytics method of AI, machine learning and deep learning. AI refers to when computer, robot, or other machines exhibit human-like intelligence. By implementing AI, the computer or machine can mimic the capabilities of the human mind by learning from examples and experience. AI is used to recognize objects, understand and respond to commands, make decisions and solve various kinds of problems. Big data are being used to train different AI models. As a result, machines can process large amount data faster than before and we can ask machines to vacuum our floors, finish our sentences while typing and even recommendations what to watch next on TV [1].

Large numbers of sensors installed in various industrial equipment and machine tools on the shop floor have accelerated this development and increased the amount of available data even more. These sensors record the activity of a machine over time. These data sets are referred as time series data, and their analysis has gained popularity over the last few years. The installed sensors in the industrial equipment and machinery assemble various time series information [2] which can be analyzed to obtain meaningful events in smart manufacturing systems. In addition to manufacturing [3], time series data can be found in various other domains such as healthcare [4], climate [5], robotics [6], stock markets [7], energy system [8], and many more. Among these diverse set of domains, the manufacturing domain is in the focus of this paper as the case study is set in the plastic processing industry.

Sensors are one of the key driving forces in the revolution of intelligent and smart manufacturing. In an industrial machine tool, sensors may collect data for different key variables over time which is called Multivariate Time Series (MTS) data. If there is only one variable measured over time, these data are called univariate time series. So we can say, a MTS consists of several univariate time series. This is why MTS analysis is considered more complex than the analysis of univariate time series. One of the main reasons behind this difference is the correlation between the different variables. In this paper, our analysis of time series will be limited to one of the most common machine learning problems, classification. In a nutshell, the goal is to i) identify several key events over the time series and ii) enable the model to identify the class of any key event from those classified and identified events within the dataset.

Classification problems mainly deal with categorical variable where each variable belongs to a specific category and the goal of the classification model is to identify the category of a specific event. For MTS classification, the whole time series is divided into specific segments, each of those segments belong to a category with distinguished patterns.

A number of algorithms have been developed to analyze MTS. Some common approaches used before the evolution of smart manufacturing are simple exponential smoothing [9], dynamic time warping [10], [11], autoregressive integrated moving average [12]. In addition to those traditional approaches several machine learning algorithms like K nearest neighbor [13], decision trees [14], and Support Vector Machine (SVM) [15] were used on multiple occasions. Some authors used combination of k nearest neighbor algorithm with some distance approaches like DTW [16], [17] or Euclidean distance measure [18]. It has been shown that no single traditional approach can outperform the result obtained from the K nearest neighbor algorithm coupled with some distance measures [19]. However, ensemble methods of different discriminant classifiers such as SVM and nearest neighbor with some distance approaches and other machine learning classifiers such as decision tree and random forest can provide better result than nearest neighbor combined with dynamic time warping method (NN-DTW) [10]. Two of the common issues with the traditional methods are that they often fail to locate important features within the time series on their own and cannot identify the correlation between the variables which result in false identification of any categorical event [3]. From this point of view it is evident that for a univariate time series they may provide reasonably good result, but for MTS their efficiency may not be good enough. In addition to that, handling of the massive volume of data is another issue for traditional approaches along with simple machine learning algorithm. This is why deep learning has come into the picture with the capability of handling large amount of data by using a deep neural network in multiple layers to extract meaningful features.

For the last few years deep learning techniques, a variety of neural network algorithms, have been used extensively to deal with time series problems. For MTS, deep learning approaches are of special interest as deep neural networks can learn the pattern of the dataset by understanding the correlation between the variables of interest. It was shown in the literature that deep neural networks can significantly outperform any traditional methods such as NN-DTW [20] for both multivariate and univariate time series. For a small number of variables, for example, in signal processing where two nodes are available and values obtained for those two nodes over time build a MTS with two variables, NN-DTW may provide good result as the distance method needs to deal with only two curves. However, when the number of variables increases, it becomes more complex for NN-DTW. This is why deep neural networks are of special interest for the case study of this paper where twelve distinct temperature zones which means twelve variables are present in the dataset.

The most common two neural networks used over the last few years are Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) and there has been a lot of variations developed to tackle with a variety of problems. CNNs gain much popularity for their contribution to computer vision problems [21]. This is why CNNs have been used extensively in image recognition tasks [22], natural language processing [23]–[25], and speech recognition [26]. The speech recognition and natural language processing both can be seen as some sort of sequential learning problems. This is why although initially developed for computer vision problems, CNN has been one of the most popular deep neural networks for dealing with time series problems especially MTS problems [3], [20], [27]–[33]. Another popular neural network

used recently is RNN which is mainly developed for sequential learning and it performs well for univariate time series, but its use for the classification of MTS is limited [34]. For the time series dataset with missing values it provides reasonably good result [35].

This paper aims to answer the following research questions relevant for the case study presented in the next chapter:

- **RQ1: Can a deep learning approach provide better result than the traditional approaches for this case study?**
  - **RQ1.1: Which deep neural network is the most suitable one for this case?**
  - **RQ1.2: Should we use a combination of several neural networks like CNN and RNN or a single one can provide the best result?**
- **RQ2: How we can deal with the unlabeled data issues?**

## 1.2 Background

In the polymer processing industry dryers are one of the fundamental components for “supplying dry-heated air that is blown upward through the to-be-dried material for several hours, while new undried, cold/moist material is continuously loaded on top of the dryer module, steadily moving downward through the dryer” [36] [37]. The drying hopper has two distinct components, one of which is drying hopper monitor and another one is the regen wheel. Both of these have distinct impact on the overall polymer processing. The drying hopper monitor has eight temperature zones, the regen has three temperature zones and dew point temperature is also measured for delivery air; all these temperatures are measured by temperature sensors. These twelve temperatures are measured using sensors over the period of one year (12 months) for this case study. The final data available is preprocessed with ignoring missing values and outliers or extraneous cases. Overall these data have temperature readings for twelve temperature zones collected over a year with a sampling interval of one minute. Figure 1.1 shows the temperature profiles obtained from the sensors.

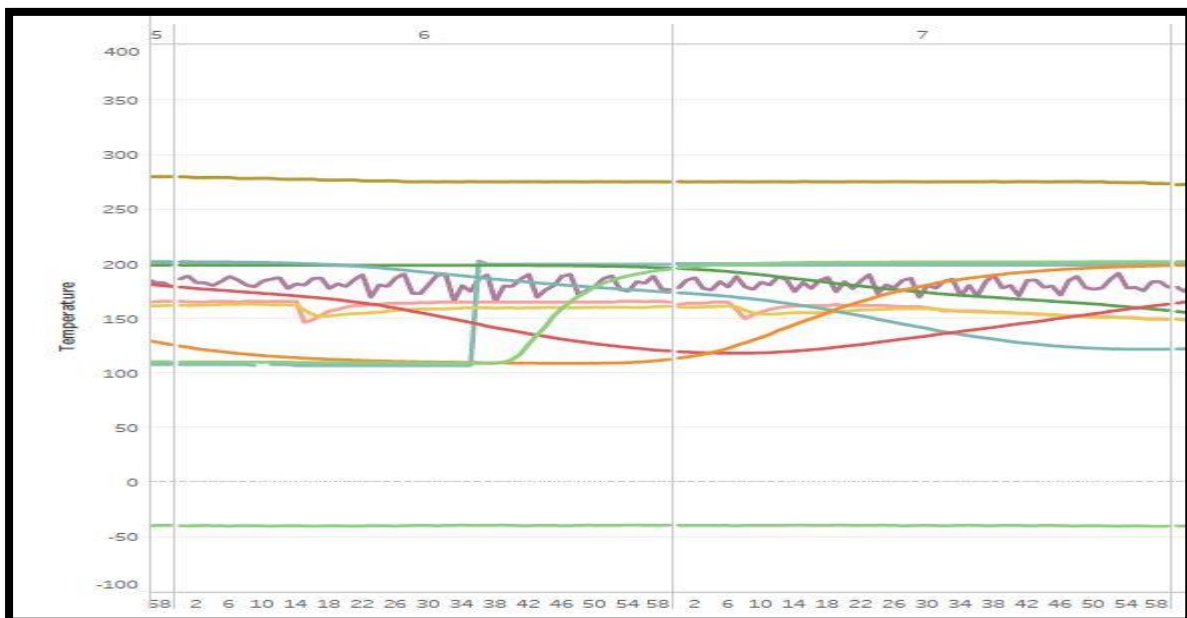


Figure 1.1: Temperature Profiles

As there is a large amount of data available for six main temperature zones in the dryer/ hopper system and six additional temperature zones in the regen and dryer regions like the hopper 1 delivery air temperature etc., it is possible to extract meaningful features from the real time analysis of these data. If real time scenario of the drying hopper can be extracted from the analysis of data, the production planner can determine the type of maintenance necessary.

The main goal of identifying any key event is a part of predictive maintenance so that the operator of the machine can identify any potential hazard in the process. A massive amount of data collected from sensors has made this possible in recent years. These large data sets can be analyzed to identify hundreds of features which can be used to provide meaningful information about the state and condition of the machines. Predictive maintenance can be defined as “the maintenance strategy that employs advanced analytics to predict machine failures is known as Predictive Maintenance” [8].

For this purpose, deep learning algorithm needs to be employed so that using a classifier, machine can automatically detect whether any specific instance belongs to any disruption events, based on which proper initiatives can be taken for the smooth flow of production. A detailed description of the overall process and some distinct events can be found in [36].

### 1.3 Objectives and Scopes

There are several objectives and scopes of this thesis. First, versatility of MTS analysis needs to be studied. Drying hoppers mechanism and patterns of temperature profile understanding is another important objective of this thesis. Afterwards, understanding various parameters related to MTS and applying this understanding to analyze the current material drying process and various events associated with industrial drying hopper is another crucial objective of this thesis. In addition to that, identifying various ways to deal with data labeling and imbalance data issue bring a lot of potential scopes for this specific drying hopper case. Understanding various parameters related to machine learning and deep learning algorithms and employing those algorithms to classify the MTS data is the primary goal of this thesis so that a comparative analysis on the employed algorithms can be performed.

### 1.4 Outline of methodology

In order to carry out the experiment, several steps have been incorporated which are:

Study of the state of art of MTS classification with traditional approaches like machine learning algorithms and deep learnin algorithms

Study of the literature regarding data labeling and imbalanced data issue

Perform necessary preprocessing of the data for using in ML and DL algorithms

Model buildup for this particular case study and implementation in python

Select the best method in terms of different performance measures and provide recommendation

## 1.5 Organization of the Thesis

This thesis is organized into five chapters along with references and appendix at the end. The first chapter provides the general introduction with background of the study. Moreover, objectives of the thesis and outline of methodology are also described in this chapter.

The second chapter provides the current state of the art of the MTS classification. The first two sections describe the traditional approach and deep learning approaches whereas the last section describes the current state of the art for handling unlabeled data issue.

The third chapter highlights the characteristics of the dataset and necessary preprocessing. This chapter also provides various technique to deal with data imbalance issues. Afterwards, solution approach from the view point of various algorithms are described. Moreover, various performance measure and system specification were showcased in this chapter as well.

The fourth chapter describes the results obtained through using different algorithms and provides a summary of the result. This chapter also identifies the best method to use for this specific data case. Afterwards, a detail discussion is performed on the overall issues and results of the thesis

The fifth and final chapter provides the conclusion and potential scopes for future work for this specific case study.

## 2 Literature Review

A variety of algorithms have been applied to solve MTS classification. MTS data have increased in various domains like anomaly detection, clinical diagnosis, weather prediction, stock price, human motion detection, fault detection in manufacturing process and so on. Among these variety of fields, MTS data have become very common in manufacturing industry due to the use of variety of sensors installed at the machineries in the shop floor of a manufacturing plant. This is why MTS analysis like classification has gained extreme popularity among researchers in the manufacturing domain. With the increasing importance in temporal data mining, researchers have continuously been developing variety of algorithms to tackle a variety of problems in this field. Among temporal data mining problems, multivariate analysis provides high complexities with increasing number of variables which might be highly correlated or not. Overall the spatial structure in temporal data, time dependency, correlation among variables etc. need to be carefully handled when dealing with any MTS analysis. In this section, current state of the art of MTS classification will be presented from two point of views; one of them is the traditional approach and the other is the AI approach like deep learning.

### 2.1 Traditional Algorithms

A benchmark algorithm used for classifying MTS is K- nearest neighbor with dynamic time warping. Two approaches can be taken for MTS data according to the authors in [37]. One of them is summing up the univariate time series DTW distances for the dimension of MTS whereas in the other one, distance between two time steps is calculated through summing up distance between each MTS which as shown in Figure 2.1.

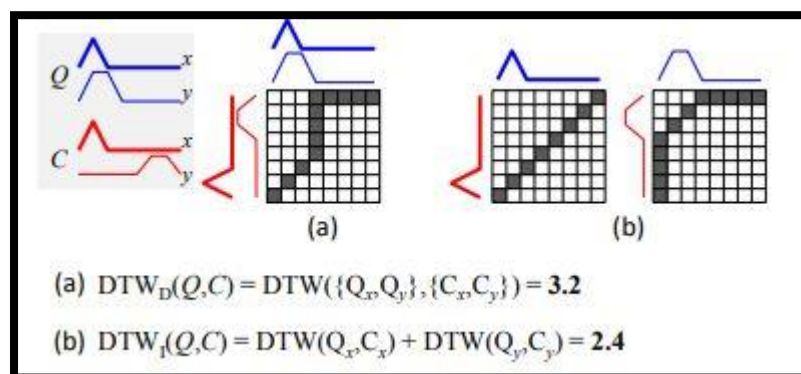


Figure 2.1: Two methods of calculating DTW distance[37]

The author claims that traditional belief which is two methods are equivalent to each other in terms of classification of MTS for a specific case is not really true and these two methods vary from problem to problem. One method might work better in one specific use case whereas the other one might not work well. They tested these ideas on a very extensive set of MTS datasets and justified the reasoning behind using two different DTW approaches.

Using nearest neighbor classifier is very common in MTS datasets. In [38], the authors used large margin nearest neighbor (LMNN) and DTW. Mahalanobis distance based DTW is used to calculate the relations among variables through Mahalanobis matrix and LMNN is used to learn the matrix through minimizing a renewed, non-differentiable cost function by co-ordinate descent method. This method is compared with other similarity measure technique of MTS and the authors claimed the superiority of their proposed method over other techniques. This



technique is also used by the authors in [39]. DTW multivariate prototyping is used in evaluating scoring and assessment methods for virtual reality training simulators. It classifies the VR data as novice, intermediate or expert where 1-NN DTW performed reasonably well, the only better algorithm for this case was RESNET; an advanced version of CNN [40]. Overall, using DTW as a dissimilarity measure among features of time series and adapting the nearest neighbor classifier in temporal data mining was very popular before the evolution of deep learning [41].

A parametric derivative DTW is another variant of the DTW used in temporal data mining. This technique combines two distances which are DTW distance between MTS and the DTW distance between derivatives of MTS. This new distance is used afterwards for classification with nearest neighbor rules [42]. Using a template selection approach based on DTW so that the complex feature selection approach and domain knowledge can be avoided is another approach taken for classifying MTS in [43]. Another variant of DTW is using DTW distance measure with integral transformation. Integral DTW is calculated as the value of DTW on the integrated time series. This technique combines the DTW and integral DTW with the 1-nearest neighbor classifier which shows no overfitting issue [44].

DTW has also been used with hesitant fuzzy sets where time instance segments get more attention than treating MTS data as a whole object or time instance one by one. In this method, alignment between time instance segments is optimized as claimed by the authors in [45]. Their research also showed that this method can be reduced to original DTW by setting scale parameters. Furthermore, this method can balance the time consumption and accuracy of the MTS classification.

Data normalization is a commonly used technique in any temporal data mining problem as different variable have values which might be highly different from one variable to another. But sometimes, normalization might destroy the information existing in the raw data which is why combination of both raw data and normalized data might preserve meaningful information about the data. The authors in [46] used this approach on nearest neighbor with DTW and obtained better classification accuracy. Longest common subsequence method is sometimes incorporated with DTW to provide better classification accuracy [47].

Symbolic representation of MTS is another traditional technique used for MTS classification which considered all elements of the time series simultaneously and symbols are learned through using a supervised learning algorithm. A tree based ensemble is used to detect the interactions between each univariate time series represented as columns with time index. A second ensemble is used to handle high dimensional input through implicit feature selection. These tree learners can efficiently handle nominal and missing values [48]. MrSQL is another technique based on symbolic representation which is used by research that transform time series data in time domain known as symbolic aggregate approximation (SAX) [49] and frequency domain known as symbolic fourier approximation (SFA) [50]. Discriminative subsequences are extracted from this symbolic data and these are used as features for training a classification model [51][52]. Word extraction for time series classification plus multivariate unsupervised symbols and derivatives abbreviated as WEASEL+MUSE also uses SFA transformation to create sequence of words. A feature selection method determines promising features and these features are extracted from all dimensions. Feature selection is performed using a chi-squared model and then logistic regression is used to learn the features [53].

In dealing MTS classification, two major components need to be considered which are approximating sequential dynamics and learning relationship among different variables. In [54], authors used distance based method for approximating sequential dynamics, whereas granger causality is used to learn the relationship among different variables. Sparsity of the learnt time series is constrained to find the focal series.

One of the most extensive research on traditional methods for both MTS and UTS can be found in [10] which highlights almost all of the above mentioned traditional approaches in different categories like whole series similarity, phase dependent intervals, phase independent shapelets, dictionary based classifiers, and combinations of transformations. This paper is a great resource for any time series classification enthusiast to get an overview of all the traditional methods. Another review paper which shows a brief overview of different classification approaches for MTS can be found in [52].

Machine learning algorithms, both nonlinear techniques and ensemble learning techniques have also been applied for time series classification over the year. Traditional classifiers like Naïve bayes, Decision Tree and SVM are the most popular ones. Before using these algorithms MTS data needs to be converted into feature vector format. This is why the authors in [55] segmented the time series for obtaining a qualitative description of each series and determined the frequent patterns. Afterwards the patterns which are highly discriminative between the classes are selected and transformed the data into vector format where the features are the discriminative patterns.

### 2.2 Deep Learning Approaches

With the evolution of deep learning CNN has been used mostly over the years in temporal data mining especially for classification. Moreover, RNN like Long short time memory has also been the example of recent algorithmic advance in time series classification problem. Furthermore, combination of both of these two algorithms which are although developed for different purposes showed extremely good result for time series classification problem. Over the year, a lot of different versions of these algorithms have been proposed by the researchers and those are performing well in different case studies. In this section, several papers which have used these algorithms and their variants will be discussed briefly.

CNN has been adapted to time series classification with 1D filter in the convolutional layer. The reason of its popularity is it can discover and extract suitable internal structure to generate the deep features of the input time series automatically through convolution and pooling operation [56]. This is not really the case in traditional feature extraction method where features need to be extracted manually through feature engineering.

“Deep learning for time series classification: A review” [20] and “The great MTS classification bake off: a review and experimental evaluation of recent algorithmic advances” [57] are the two papers which provided the summary and basics of the recent algorithmic advance in the use of deep learning for MTS classification. Natural language processing (NLP) and Speech recognition (SR) are two fields where RNN and LSTM has been highly successful over the year and recently CNN has also showed high performance in terms of accuracy. NLP and SR both have sequential aspects which is similar to time series analysis. An overview of the deep learning approaches for times series classification taken from [20] is shown in Figure 2.2.

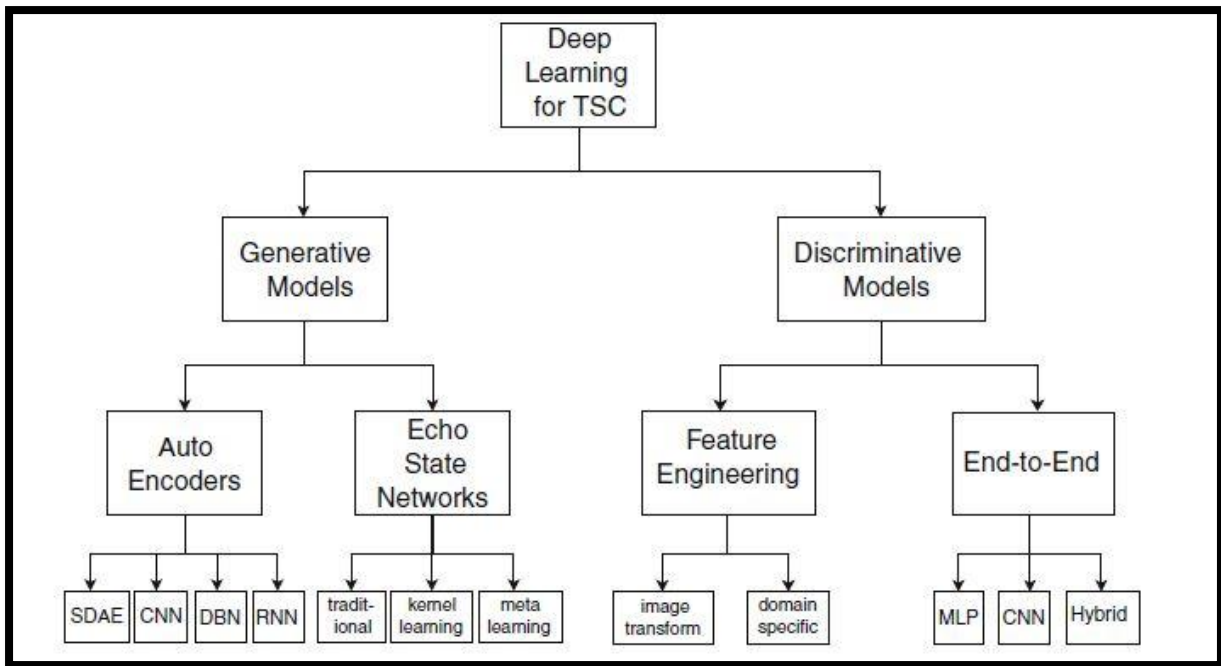


Figure 2.2: Deep learning overview for time series classification[20]

The authors in [27] used a tensor scheme with multivariate CNN for time series classification where the model considers multivariate aspect and lag feature characteristics simultaneously. Four stages were used in CNN architecture which are input tensor transformations stage, univariate convolution stage, multivariate convolution stage and fully connected stage. In this method they have used an image like tensor scheme to encode the MTS data. This approach is taken because of the highly successful nature of CNN in compute vision for image classification.

In addition to using convolution operation, deconvolution has also been applied to time series data mining. In [58], the authors used deconvolutional network along with SAX discretization to learn the representation of MTS. In this way, the authors were able to capture the correlation with deconvolution that forced the pooling operation for dimension reduction along each position of each variable. SAX discretization extracted bag of features and this representation and bag of features improved classification accuracy. Dilated CNN is another version of the CNN applied to time series where MTS is transformed into image, stacks of dilated and strided convolutions are applied for feature extraction across the variables [29]. Among other approaches with CNN, multi-channel deep CNN is another highly used one, where the model learn features from individual time series and combines all channels after the convolution and pooling stage. The combined and learnt features are then fed to a multilayer perceptron (MLP) for final classification [32].

The dataset used in this thesis comes from manufacturing domain where fault detection in an industrial machine has been very common now a days with the installed sensors and high technological advance because of industry 4.0 and artificial intelligence. CNN has been extensively applied in manufacturing time series data obtained from sensors. This predictive maintenance helps detecting fault in a machine before reaching a critical condition. In majority of cases, combination of time series images and CNN have been applied for fault detection in manufacturing. The imaging is used for two reasons; one of them is the highly successful nature of CNN in image processing, the second one is getting an overview of the fault pattern from

the image and identifying the image of time series as production threatening or not. In [59], the authors performed a principal component analysis for feature extraction and reducing the number of MTS variables to two so that they can identify the most useful two components in the machine. The time series are encoded into image using Gramian angular field (GAF) and used the images as input for the CNN. Another similar research can be found in [33] where three techniques of converting MTS data into images have been used and tested which are GAF, Gramian angular difference field (GADF), and Markov Transition field (MTF). It has been found that different approaches of converting MTS into images do not affect the classification performance and a simple CNN can outperform other approaches. In semiconductor manufacturing it has been tested that MTS- CNN can successfully detect the fault wafers with high accuracy, recall and precision [3].

Combining CNN, LSTM and DNN has been another highly used approach over the year. In [60], the authors proposed a combined architecture abbreviated as CLDNN and applied on large vocabulary tasks which outperformed three individual algorithms. Another similar approach named as MDDNN has been used to predict the class of a subsequence in terms of earliness and accuracy. Attention mechanism is incorporated with the deep learning framework in order to identify critical segments related to model performance [61]. The proposed framework as shown in Figure 2.3 used both time domain and frequency domain through fast fourier transformation and merged them together for prediction. Another similar research focused on early classification can be found at [28].

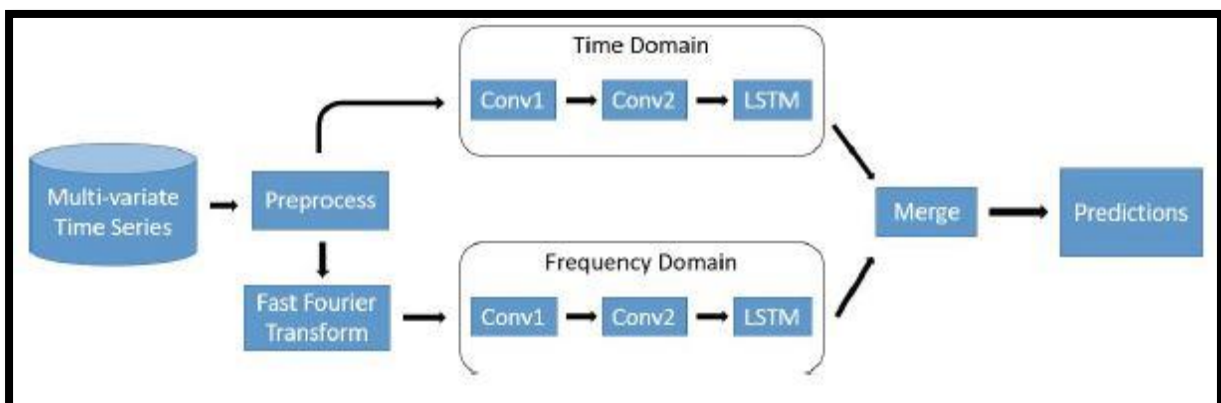


Figure 2.3: MDDNN model architecture[61]

Apart from LSTM, other recurrent network variants like bidirectional RNN (BiRNN), bidirectional Long Short Term Memory (BiLSTM), Gated Recurrent Unit (GRU), Bidirectional Gated Recurrent Unit (BiGRU) have been adapted to use in MTS classification. In [62], the authors used MLSTM- FCN which is the combination of LSTM, squeeze and excitation (SE) block and fully CNN where the SE block is integrated within FCN to leverage its high performance for the MTS classification. The similar approach of using excitation block has also been used in [30].

Multi scale entropy and inceptions structure ideas has been used with LSTM-FCNN model for MTS classification. Subsequences of each variable have been convolved through 1D convolutional kernel with different filter size to extract high level multi-scale spatial features. Afterwards, LSTM has been applied to further process and capture temporal information. Both of these spatial and temporal features are used as input to the fully connected layer [31]. Apart from CNN, Evidence feed forward hidden markov model (EFF-HMM) has been combined with LSTM to classify MTS. According to [63], learning of EFF-HMM is performed based on

the mistakes of the LSTM which outperformed other state of the art in human activity recognition.

### 2.3 Data Labeling

Classification is a supervised learning technique which needs labelled data. But the dataset used in this thesis is not labelled which is why data labeling was the primary concern before using any supervised learning algorithm. In literature, very few works on time series data labeling can be found. The technique researchers often used is known as semi supervised and active learning for univariate time series. In [64], the authors focused on active learning with positive unlabeled data. Their framework proposed a sample selection strategy to find the most informative samples for manual labeling. They introduced two active learning approaches which obtained high confident training dataset for classification.

Another paper addresses the labeling issue and the relevance of self-labeling techniques and semi supervised learning technique for time series classification. An empirical study was performed to compare self-labeled methods and various learning schemes and dissimilarity measure. The authors experimented with 35 different datasets with different percentage of labelled data in order to measure the transductive and inductive classification capabilities of the self labelled data [65].

Semi supervised learning approach has been extensively used in text classification, but in time series domain it has not been used much. In [66], the authors made special consideration to adapt the well-known semi supervised approach into time series domain. Their approach was tested on diverse data sources like electrocardiograms, handwritten documents, manufacturing and video datasets. The results of the experiment showed that only a small amount of labelled data is needed for using the semi supervised approach.

In this chapter, a brief overview of the current state of the art of MTS have been presented. In the next chapter, methodology of this thesis with data exploration as well as necessary preprocessing and various algorithms tested on this dataset will be discussed.

### 3 Methodology

#### 3.1 Data Exploration and Preprocessing

As mentioned in the background section, there are twelve distinct temperature zones. The temperature zones are:

- Delivery Air Dewpoint (DAD)
- Regen Temperature Active Setpoint (RTAS)
- Regen Temperature Wheel Inlet (RTWI)
- Regen Temperature Wheel Outlet (RTWO)
- Hopper 1 Delivery Air Temperature (H1DAT)
- Hopper 1 Hopper Outlet Temperature (H1HOT)
- Hopper 1 Drying Monitor 1 Temperature (Bottom) (H1DM1T)
- Hopper 1 Drying Monitor 2 Temperature (H1DM2T)
- Hopper 1 Drying Monitor 3 Temperature (H1DM3T)
- Hopper 1 Drying Monitor 4 Temperature (H1DM4T)
- Hopper 1 Drying Monitor 5 Temperature (H1DM5T)
- Hopper 1 Drying Monitor 6 Temperature (H1DM6T)

The raw data obtained from the machine is preprocessed to obtain the final data file ignoring missing values and outliers almost in all cases. These twelve temperatures are measured using sensors over the period of one year (12 months) for this case study although the obtained data file contains sensor reading of six months. The final dataset is prepared using a sampling interval of one minute. A chunk of the dataset is shown Figure 3.1 and Figure 3.2.

Unix Time	Actual Time	Delivery Air Dewpoint (F)	Regen Temp Active Setpoint (F)	Regen Temp Wheel Inlet (F)	Regen Temp Wheel Outlet (F)	Hopper 1 Delivery Air Temp (F)	Hopper 1 Hopper Outlet Temp (F)	Hopper 1 Drying Monitor 1 Temp (Bottom) (F)	Hopper 1 Drying Monitor 2 Temp (F)	Hopper 1 Drying Monitor 3 Temp (F)	Hopper 1 Drying Monitor 4 Temp (F)	Hopper 1 Drying Monitor 5 Temp (F)	Hopper 1 Drying Monitor 6 Temp (Top) (F)
1525150860000.00	5/1/2018 5:01:00 AM	-38.08	195.93	196.33	154.08	200	165	203.21	201.99	203.09	202.4	198.97	159.85
1525150920000.00	5/1/2018 5:02:00 AM	-38.01	195.93	195.9	150.81	200	165	203.21	201.99	203.09	202.4	198.91	159.98
1525150980000.00	5/1/2018 5:03:00 AM	-37.99	195.93	195.93	151.38	200	165	203.21	201.99	203.09	202.4	198.95	159.95
1525151040000.00	5/1/2018 5:04:00 AM	-38.13	195.93	196.03	149.61	200	165	203.21	201.99	203.09	202.4	198.93	160.07
1525151100000.00	5/1/2018 5:05:00 AM	-38.26	195.93	195.86	155.1	200	165	203.21	201.99	203.09	202.39	198.92	159.77
1525151160000.00	5/1/2018 5:06:00 AM	-38.36	195.93	195.84	155.03	200	165	203.21	201.99	203.09	202.39	198.97	159.4
1525151220000.00	5/1/2018 5:07:00 AM	-38.38	195.93	195.97	152.37	200	165	203.21	201.99	203.09	202.39	198.96	159.29
1525151280000.00	5/1/2018 5:08:00 AM	-38.4	195.93	195.82	150.79	200	165	203.21	201.99	203.09	202.39	198.96	159.25
1525151340000.00	5/1/2018 5:09:00 AM	-38.49	195.93	196.22	151.05	200	165	203.21	201.99	203.09	202.39	199	159.11
1525151400000.00	5/1/2018 5:10:00 AM	-38.67	195.93	196.06	155.21	200	165	203.21	201.99	203.09	202.39	198.93	159.25
1525151460000.00	5/1/2018 5:11:00 AM	-38.75	195.93	196	154.6	200	165	203.21	201.99	203.09	202.39	198.93	159.18
1525151520000.00	5/1/2018 5:12:00 AM	-38.73	195.93	196.07	152.19	200	165	203.21	201.99	203.09	202.39	198.95	159.3
1525151580000.00	5/1/2018 5:13:00 AM	-38.67	195.93	195.68	152.85	200	165	203.21	201.99	203.09	202.39	198.99	159.5
1525151640000.00	5/1/2018 5:14:00 AM	-38.74	195.93	195.97	149.43	200	165.5	203.21	201.99	203.09	202.39	198.97	159.59
1525151700000.00	5/1/2018 5:15:00 AM	-38.88	195.93	196.24	153.56	200	165	203.21	201.99	203.09	202.39	198.9	159.4
1525151760000.00	5/1/2018 5:16:00 AM	-38.97	195.93	195.97	155.07	200	165	203.21	201.99	203.09	202.39	198.95	159.37
1525151820000.00	5/1/2018 5:17:00 AM	-38.9	195.93	195.83	151.84	200	165.5	203.21	201.99	203.09	202.39	198.94	159.4

Figure 3.1: Raw dataset in CSV format

As the temperature values for all twelve variables are measured over the period of time, this dataset is a MTS. As mentioned before, MTS mainly consists of several univariate time series. A univariate time series has only one variable measured over a certain period of time with a specific time interval. It can be denoted as  $T = \{ t_1, t_2, t_3, \dots, t_n \}$  where  $t_i$  is the measured value at the  $i^{th}$  entry of the time series. A MTS  $X = ( X_1, X_2, \dots, X_m )$  where  $X_m$  is the  $m^{th}$  univariate time series which can be denoted as before where each univariate time series has  $n$  dimension [40]. The dataset can be viewed as an  $n * m$  matrices where  $m$  refers to the no. of univariate time series and  $n$  refers to the length of each time series.

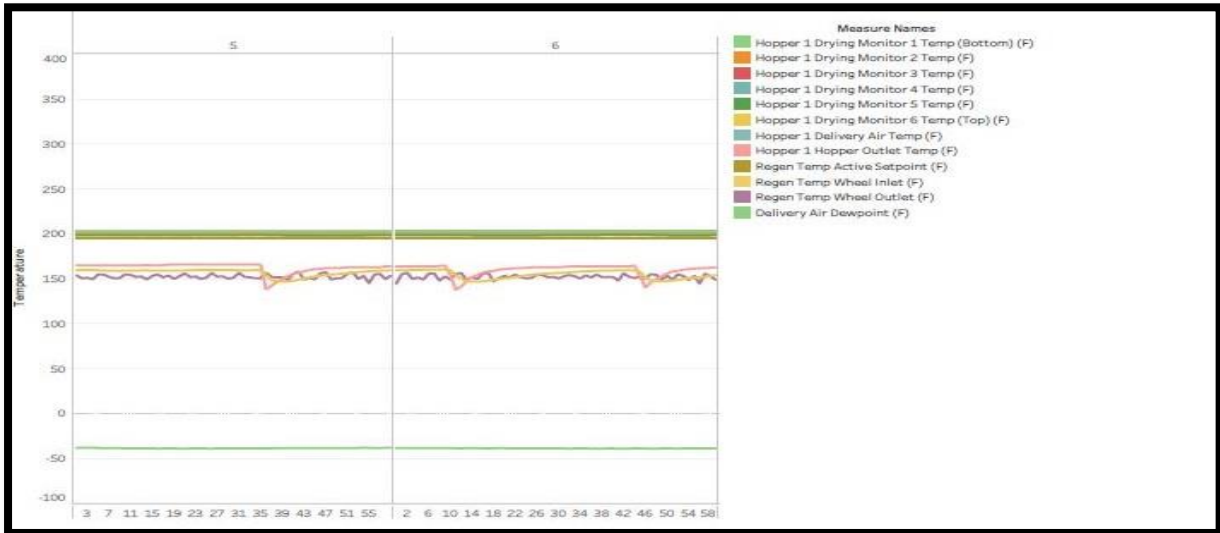


Figure 3.2: Temperature profile obtained from primarily preprocessed data

In Figure 3.1, the first column of the dataset is the time step converted to UNIX time in milliseconds. UNIX time which is also known as Epoch time is a method for describing a point in time. It is the number of seconds that has elapsed since the Unix epoch minus leap seconds; the Unix epoch is 00:00:00 UTC on 1 January 1970 (an arbitrary date); leap seconds are ignored, with a leap second having the same Unix time as the second before it, and every day is treated as if it contains exactly 86,400 seconds [67].

The first time step in the data file is 1525150860000 which can be converted to the real date and time as May 1, 2018 5:01:00 AM. So, the temperature reading starts from May 1, 2018 5:01:00 AM and ends at November 1, 2018 5:00:00 AM. As the sampling interval is one minute, no. of entries in the time series can be calculated in the following way:

No. of entries in the time series dataset:

19 (May 1) + 30\*24 (May 2 – May 31) + 30\*24 (June) + 31\*24 (July) + 31\*24 (August) + 30\*24 (September) + 31\*24 (October) + 5 (November) = 4416 hours = 4416 \* 60 = 264,960 minutes.

But the data file contains 263,476 entries which indicates 264,960 – 263,476 = 1,484 minutes of data are missing. A detail investigation of the temperature profiles reveal those missing values in the dataset. One of those examples are shown in Figure 3.3 and Figure 3.4.

1525415520000.00	5/4/2018 6:32:00 AM	-25.14	375	375.83	209.41	200	143	203.6	202.25	203.3	202.59	198.91	146.11
1525415580000.00	5/4/2018 6:33:00 AM	-25.3	375	375.37	208.61	200	148	203.6	202.25	203.3	202.59	198.8	144.92
1525415640000.00	5/4/2018 6:34:00 AM	-25.06	375	375.03	209.17	200	152	203.58	202.25	203.3	202.59	198.66	145.23
1525415700000.00	5/4/2018 6:35:00 AM	-24.74	375	374.39	206.03	200	155	203.57	202.25	203.3	202.59	198.56	145.86
1525415760000.00	5/4/2018 6:36:00 AM	-24.71	375	374.64	190.69	200	156	203.57	202.25	203.3	202.59	198.54	146.38
1525434120000.00	5/4/2018 11:42:00 AM	-29.03	375	375.11	220.59	200	153	203.53	202.34	203.42	202.68	199.13	157.93
1525434180000.00	5/4/2018 11:43:00 AM	-28.91	375	374.88	194.45	200	148	203.52	202.34	203.41	202.68	199.07	153.5
1525434360000.00	5/4/2018 11:46:00 AM	-29.35	375	374.96	211.06	200	158.5	203.51	202.34	203.41	202.68	198.93	152.76
1525434420000.00	5/4/2018 11:47:00 AM	-29.47	375	375.77	218.38	200	160	203.51	202.34	203.41	202.68	198.93	153.83
1525434480000.00	5/4/2018 11:48:00 AM	-29.26	375	374.55	213.17	200	161.5	203.51	202.34	203.41	202.68	198.84	155.01
1525434540000.00	5/4/2018 11:49:00 AM	-29.11	375	374.16	197.55	200	162	203.51	202.34	203.41	202.68	198.81	156.38
1525434600000.00	5/4/2018 11:50:00 AM	-29.32	375	375.5	209.8	200	163	203.51	202.34	203.41	202.68	198.85	157.66

Figure 3.3: Missing values in the primarily processed dataset

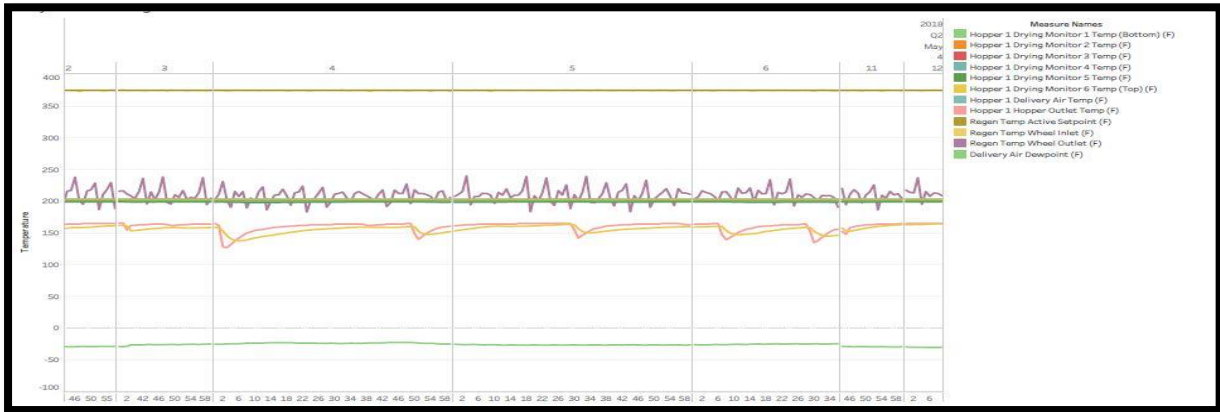


Figure 3.4: Missing values in the temperature profile

### 3.1.1 Handling Missing Values

Missing values have been a common issue in any time series analysis especially in manufacturing domain. The sensors data might be missing for numerous reasons like power outage at the sensor’s node, random occurrences of local interference [68] or data might be missing during data preprocessing steps.

As shown in Figure 3.4, the windows from 6 am to 12 am where a large portion of the dataset is missing which is from 6:37 am to 11:41 am. Although this is an extreme case in this dataset, other missing value instances are not that severe. In most of the cases, 1 or 2 minutes of data are missing. But there are some extreme cases as well as the case in Figure 3.4. Two approaches were taken to tackle the missing value problem which are provided below.

- If the missing values are at those time steps where no event is happening previously or afterwards within the time steps which has the same length of the missing time steps, the missing values will be filled using a moving average. For example, data is missing from 101<sup>th</sup> time step to 109<sup>th</sup> time step, then the first missing value (time step 101) will be filled using the average of the observations from the previous 60 time steps (time step 41 to time step 100, the second missing value (time step 102) will be filled using the average of the observations from time step 42 to time step 101.

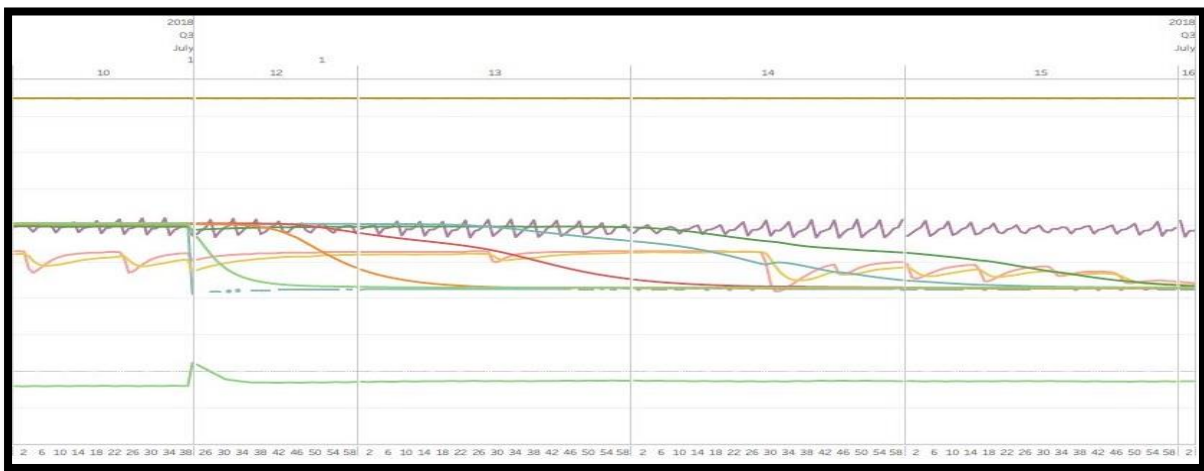


Figure 3.5: Missing value imputation



- If time steps are missing within an event, then the missing values will be imputed using the moving average of the sixty observations which are in the event (either happening before or afterwards of the missing time steps). Figure 3.5 shows such an example, where an event has started at around 10:37 am and data are missing from 10:40 am to 12:24 pm. It cannot be imputed with rolling average of previous observation as an event has already started.

### 3.1.2 Data labeling

Classification is a supervised learning technique which requires labeled data to learn the intrinsic behavior of events. Supervised learning is then used to predict the class of any event which is essential for example in fault diagnostics applications. For this case study, three major events were identified that occur regularly and have an impact on operations: startup procedure, cleaning cycle, and conveying issues [36] which are shown in Figure 3.6, Figure 3.7 and Figure 3.8.

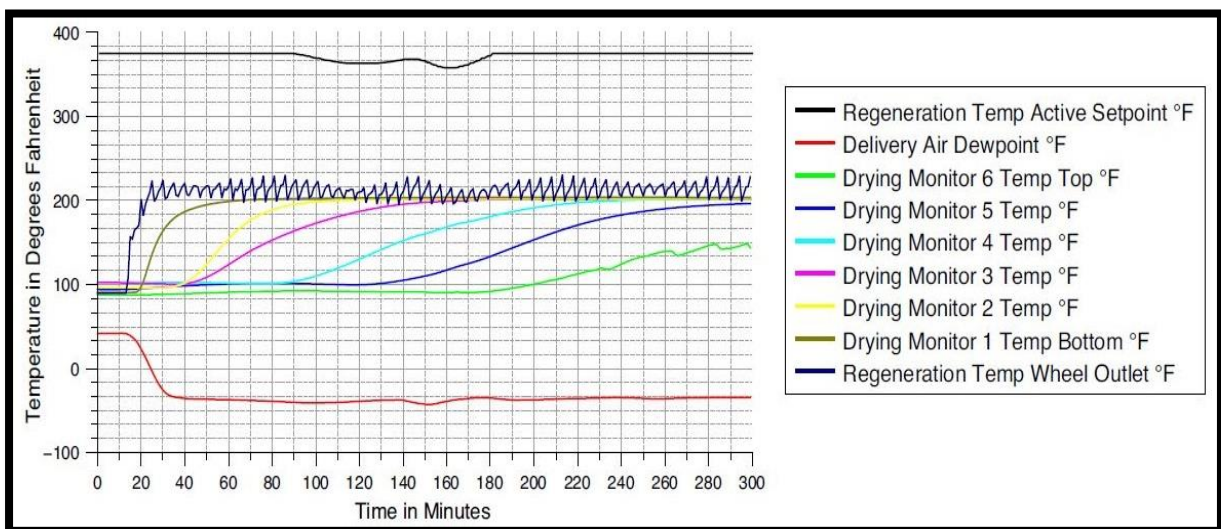


Figure 3.6: Startup Procedure[36]

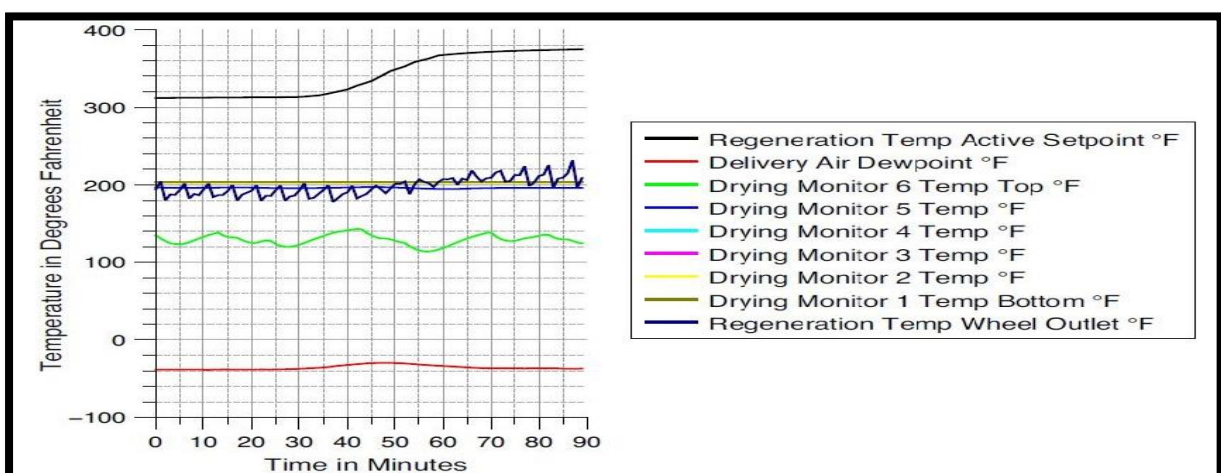


Figure 3.7: Cleaning Cycle[36]

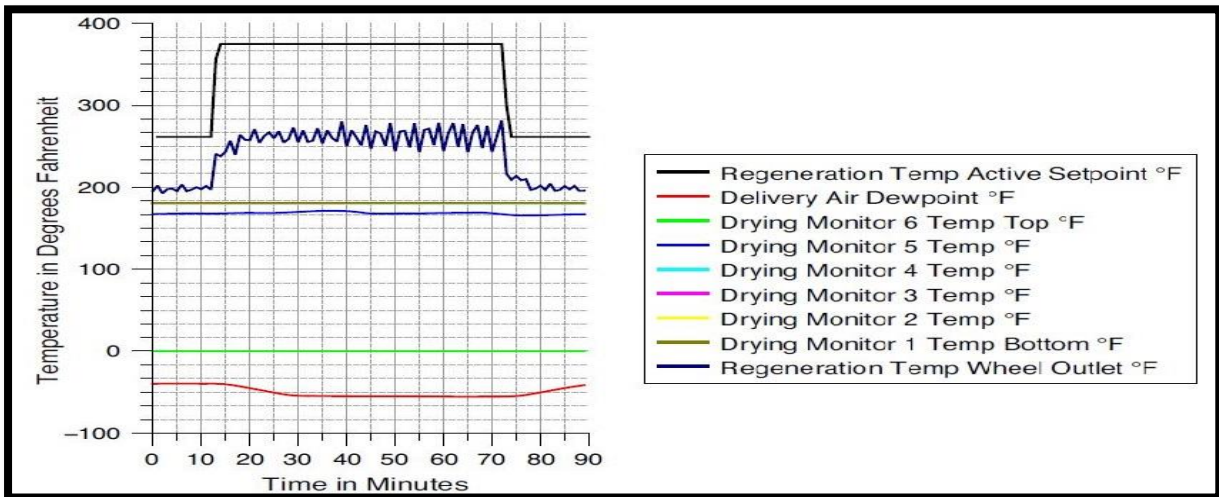


Figure 3.8: Conveying Issue[36]

Although three distinct events are identified, they are hard to define due to a lot of variation of these events. Therefore, the initial target of this thesis is to identify a specific category; either failure (any event which is unusual) or regular rather than building the ability to detect the type of any unusual event other than the usual behavior (steady state) and detect its class. For this purpose, any event which is unusual from the regular case is identified, selected, and labeled as one class and the rest of the events which can be defined as steady state will be a separate class. Assuming this approach is successful for this case study, the subsequent goal will be to identify the class of any unusual labeled event (not steady state). This will enable a variety of value adding applications, including contextualizing the operation for the process planers and operators, predict necessary maintenance steps, and provide input for customized designs of next generation systems.

### 3.1.2.1 Data labeling issues

Selecting a specific event for the initial analysis highlights another challenge associated with the data set. For example the cleaning cycle is a specific type of events, but there are many variations of this event that can be observed over the time series. The challenge is to define the event precisely and still be able to take some variations of the definition into account for the classification. For the analysis, among several events observed in the data set, one event which is shown in the image below is considered for the specific event.

In Figure 3.9, several temperature reading shows deviation from the usual behavior which are described below:

- Regen temperature wheel inlet drop sharply from around 300° F to around 150° F, then with a slight increase within 5 minutes, it starts decreasing again to around 100° F and becomes steady.
- Hopper 1 drying monitor temperature 2 and 5 show almost similar behavior, drop from around 175° F to around 160° F.
- Regen temperature wheel outlet drops from around 175° F after a continuous oscillating behavior to around 100° F within 20 to 30 minutes.
- Hopper 1 drying monitor temperature 1 drops from 175° F to around 125° F.

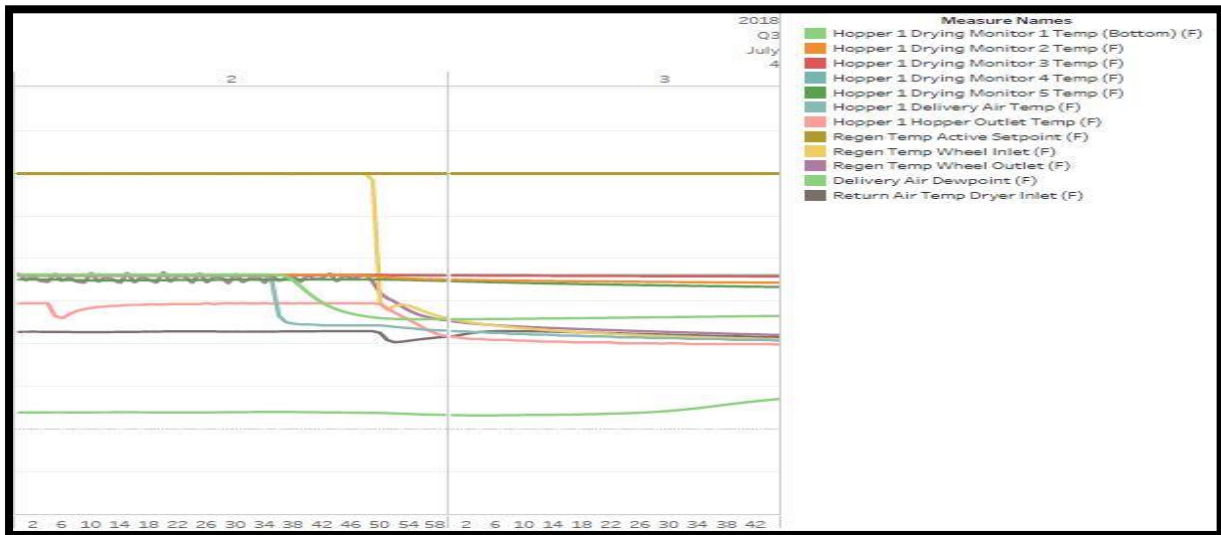


Figure 3.9: Event

- Hopper 1 delivery air temperature sharply drops from 175° F to 125° F, then becomes steady for around 15 minutes, then drops slowly to around 100° F.
- Hopper 1 outlet temperature drops from around 150° F to 100° F within 10 minutes.
- Return air temperature dry inlet shows a small drop 100° F from 115° F, then becomes steady again.
- Delivery air dew point increases from around 10° F to around 40° F. This change can be seen around 25-30 minutes after the other temperature drops occur.

Among 12 temperatures, certain amount of deviations can be observed in the above mentioned 8 temperatures. The deviation for hopper 1 drying monitor temperature 2 and 5 as well as return air temperature dry inlet are minor compared to other 5 temperatures where a significant amount of temperature drop can be observed. When defining the event, the variations of this event need to be considered while the overall scenario might be same with a little variation in temperature drop. Another such event is shown in Figure 3.10.

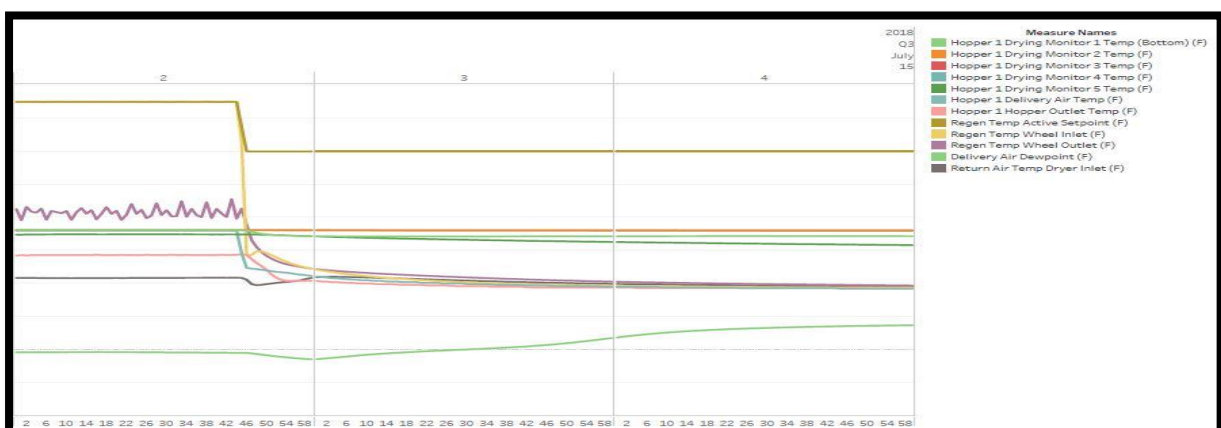


Figure 3.10: Variation of an event

The major variations are the regen temperature active set point dropping sharply to 300° F from around 375° F which was steady in the event shown before. Hopper 1 drying monitor

temperature 1 drops slightly by roughly 5° F from an approximately 175° F initial temperature, which was not the case before and hopper 1 drying monitor temperature 2 remains constant.

It is evident that although the two events are almost similar in fashion, there are some major changes which can be observed in the temperature behavior. So, the event must be defined in such a way that any minor changes from the specified event can also be detected through the classification algorithm. This can be either setting a range for each temperature; for example, the regen temperature drops from 300° F to 100° F, without setting it specifically, there can be a defined limit which can be dropping down from anywhere between 310° F and 290° F to anywhere between 100° F to 120° F or specifying a slope for each temperature zone can be another strategy.

### 3.1.2.2 Data labeling approaches

In order to label the data considering only one main event or all those major events, two approaches can be taken. One of them is manual labeling and the other one is semi supervised learning approach.

Manual labeling is always costly with respect to time needed to go through the data with an expert, especially in cases where the data set spans a long time period as is the case here. It needs significant effort from the expert who has deep understanding of the overall process. So, manual labeling may result in accurate labeling of the data which will assist the classifier to learn the behavior of the data and will be trained accordingly to identify the category.

The second method that aligns with the time restrictions of experts to label data in case of large data set is the semi supervised learning. Due to the large volume, manual labeling technique is not a feasible choice although it may provide the best labeling of the dataset. The semi supervised approach [13], [65] needs a very small amount of labeled dataset. Then this labeled dataset is trained to predict the classes of the rest of the dataset. In this way, the whole dataset is labeled which can be trained again to learn the pattern of the complete dataset to identify the classes of the test set or any future dataset.

For this case study, manual labeling is performed to obtain the finalized labelled dataset. As mentioned earlier, the labelling will be performed in such a way so that the dataset can be converted to a binary classification dataset. In this fashion, all those steady state events or regular events will be treated as one class and the rest of the dataset where any unusual pattern or behavior can be observed are treated as the other class.

### 3.1.2.3 Event identification

Before labelling, one more thing needs to be clarified which is the definition of an event for this dataset. There can be various ways to define an event, define the length of an event with a start time and end time. For example, the event shown in Figure 3.11, starts exactly from 5:04:00 AM, but for simplicity the start time of this event is considered as 5:01:00 AM, so that the hour from 5:01:00 AM to 6:00:00 AM can be considered as an event. In other words, it can be said that a major event (unusual) occurs in this hour. The definition of an event as an hour is considered for the ease of labeling and simplicity in visualizing an event.

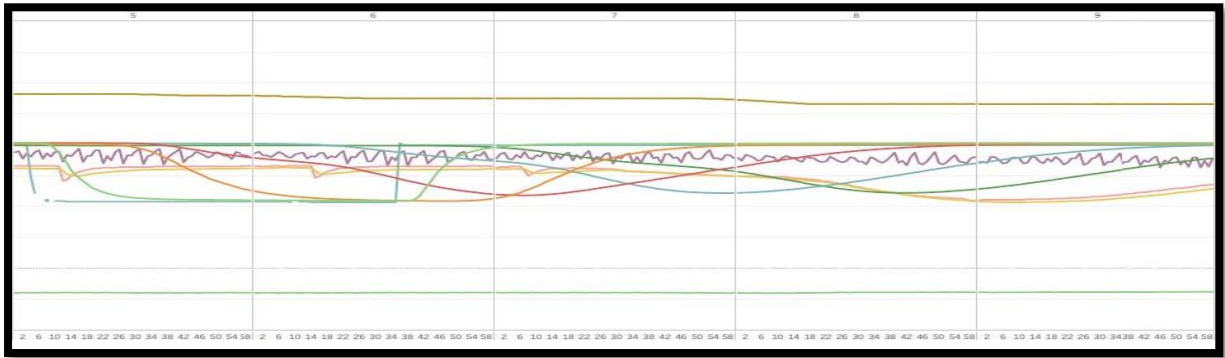


Figure 3.11: Definition of an event

In order to label the dataset, first all those significant events (unusual) are identified and listed in Table 3.1.

Table 3.1: Event durations

Start Time	End Time	Time difference(hour)	Start Time (unix)	End Time (unix)
5/5/2018 5:00:00 AM	5/5/2018 10:00:00 AM	5.00	1525496460000.00	1525514400000.00
5/9/2018 7:00:00 PM	5/10/2018 1:00:00 AM	6.00	1525892460000.00	1525914000000.00
5/10/2018 11:00:00 AM	5/10/2018 1:00:00 PM	2.00	1525950060000.00	1525957200000.00
5/12/2018 3:00:00 AM	5/12/2018 4:00:00 AM	1.00	1526094060000.00	1526097600000.00
5/13/2018 3:00:00 AM	5/14/2018 4:00:00 AM	25.00	1526180460000.00	1526270400000.00
5/19/2018 2:00:00 AM	5/21/2018 4:00:00 AM	50.00	1526695260000.00	1526875200000.00
5/26/2018 2:00:00 AM	5/29/2018 3:00:00 AM	73.00	1527300060000.00	1527562800000.00
6/2/2018 2:00:00 AM	6/4/2018 5:00:00 AM	51.00	1527904860000.00	1528088400000.00
6/9/2018 2:00:00 AM	6/11/2018 7:00:00 AM	53.00	1528509660000.00	1528700400000.00
6/23/2018 1:00:00 AM	6/25/2018 5:00:00 AM	52.00	1529715660000.00	1529902800000.00
7/1/2018 12:00:00 PM	7/2/2018 5:00:00 AM	17.00	1530446460000.00	1530507600000.00
7/4/2018 2:00:00 AM	7/9/2018 4:00:00 AM	122.00	1530669660000.00	1531108800000.00
7/15/2018 2:00:00 AM	7/16/2018 1:00:00 AM	23.00	1531620060000.00	1531702800000.00
7/21/2018 2:00:00 AM	7/23/2018 4:00:00 AM	50.00	1532138460000.00	1532318400000.00
7/28/2018 2:00:00 AM	7/30/2018 2:00:00 AM	48.00	1532743260000.00	1532916000000.00
8/4/2018 3:00:00 AM	8/6/2018 3:00:00 AM	48.00	1533351660000.00	1533524400000.00

8/7/2018 5:00:00 PM	8/7/2018 7:00:00 PM	2.00	1533661260000.00	1533668400000.00
8/8/2018 10:00:00 PM	8/9/2018 2:00:00 AM	4.00	1533765660000.00	1533780000000.00
8/9/2018 7:00:00 PM	8/9/2018 9:00:00 PM	2.00	1533841260000.00	1533848400000.00
8/11/2018 2:00:00 AM	8/13/2018 5:00:00 AM	51.00	1533952860000.00	1534136400000.00
8/22/2018 2:00:00 PM	8/22/2018 4:00:00 PM	2.00	1534946460000.00	1534953600000.00
8/24/2018 7:00:00 PM	8/25/2018 6:00:00 AM	11.00	1535137260000.00	1535176800000.00
8/25/2018 7:00:00 AM	8/25/2018 2:00:00 PM	7.00	1535180460000.00	1535205600000.00
8/25/2018 7:00:00 PM	8/27/2018 2:00:00 AM	31.00	1535223660000.00	1535335200000.00
8/27/2018 12:00:00 PM	8/27/2018 2:00:00 PM	2.00	1535371260000.00	1535378400000.00
8/28/2018 10:00:00 PM	8/29/2018 12:00:00 PM	14.00	1535493660000.00	1535544000000.00
9/3/2018 2:00:00 AM	9/4/2018 2:00:00 AM	24.00	1535940060000.00	1536026400000.00
9/5/2018 6:00:00 PM	9/5/2018 8:00:00 PM	2.00	1536170460000.00	1536177600000.00
9/20/2018 10:00:00 AM	9/20/2018 12:00:00 PM	2.00	1537437660000.00	1537444800000.00
9/21/2018 2:00:00 AM	9/21/2018 11:00:00 AM	9.00	1537495260000.00	1537527600000.00
9/21/2018 8:00:00 PM	9/21/2018 10:00:00 PM	2.00	1537560060000.00	1537567200000.00
10/23/2018 6:00:00 PM	10/23/2018 8:00:00 PM	2.00	1540317660000.00	1540324800000.00
10/23/2018 11:00:00 PM	10/24/2018 2:00:00 AM	3.00	1540335660000.00	1540346400000.00
10/24/2018 4:00:00 AM	10/24/2018 6:00:00 AM	2.00	1540353660000.00	1540360800000.00
10/27/2018 1:00:00 AM	10/28/2018 1:00:00 AM	24.00	1540602060000.00	1540688400000.00
10/28/2018 9:00:00 PM	10/29/2018 10:00:00 AM	13.00	1540760460000.00	1540807200000.00
10/29/2018 12:00:00 PM	10/29/2018 6:00:00 PM	6.00	1540814460000.00	1540836000000.00
		Total hours =841.00		

#### 3.1.2.4 Labeling each row as 0 or 1

After listing the start time and end time of all events, both the original time series data file where the first column is the time step in UNIX time and the event duration data file shown in Table 3.1 are needed to label the time series data. A sample rate of 60,000 is used as UNIX time in the original data file is in millisecond and the sampling used to generate the dataset was 1 minute or 60,000 milliseconds. The time durations from the event duration dataset are used to turn it into a column of milliseconds where each entry is 60,000 milliseconds apart from each other. Afterwards an iterable variable is created which contains the time column of the

original time series data. Afterwards, this variable and the broken down event markings data are used to create a column of 1s and 0s that exactly match the rows of the original dataframe. In short the steps are:

- Prepare a list of events with the start time and finish time.
- Break the start and finish times of the events into a column of milliseconds where each entry is 60, 000 milliseconds or 1 minute apart from each other.
- Convert this list into a column of 1s and 0s.
- Add this column to the original time series data file.

More details of this procedure can be found in [69]. A portion of the final dataset with the labelled column is shown in Figure 3.12.

labels	Time	Delivery Air Dewpoint (F)	Regen Temp Active Setpoint (F)	Regen Temp Wheel Inlet (F)	Regen Temp Wheel Outlet (F)	Hopper 1 Delivery Air Temp (F)	Hopper 1 Hopper Outlet Temp (F)	Hopper 1 Drying Monitor 1 Temp (Bottom) (F)	Hopper 1 Drying Monitor 2 Temp (F)	Hopper 1 Drying Monitor 3 Temp (F)	Hopper 1 Drying Monitor 4 Temp (F)	Hopper 1 Drying Monitor 5 Temp (F)	Hopper 1 Drying Monitor 6 Temp (Top) (F)
0	1525150860000.00	-38.08	195.93	196.33	154.08	200	165	203.21	201.99	203.09	202.4	198.97	159.85
0	1525150920000.00	-38.01	195.93	195.9	150.81	200	165	203.21	201.99	203.09	202.4	198.91	159.98
0	1525150980000.00	-37.99	195.93	195.93	151.38	200	165	203.21	201.99	203.09	202.4	198.95	159.95
0	1525151040000.00	-38.13	195.93	196.03	149.61	200	165	203.21	201.99	203.09	202.4	198.93	160.07
0	1525151100000.00	-38.26	195.93	195.86	155.1	200	165	203.21	201.99	203.09	202.39	198.92	159.77
0	1525151160000.00	-38.36	195.93	195.84	155.03	200	165	203.21	201.99	203.09	202.39	198.97	159.4
0	1525151220000.00	-38.38	195.93	195.97	152.37	200	165	203.21	201.99	203.09	202.39	198.96	159.29
0	1525151280000.00	-38.4	195.93	195.82	150.79	200	165	203.21	201.99	203.09	202.39	198.96	159.25
0	1525151340000.00	-38.49	195.93	196.22	151.05	200	165	203.21	201.99	203.09	202.39	199	159.11
0	1525151400000.00	-38.67	195.93	196.06	155.21	200	165	203.21	201.99	203.09	202.39	198.93	159.25
0	1525151460000.00	-38.75	195.93	196	154.6	200	165	203.21	201.99	203.09	202.39	198.93	159.18
0	1525151520000.00	-38.73	195.93	196.07	152.19	200	165	203.21	201.99	203.09	202.39	198.95	159.3
0	1525151580000.00	-38.67	195.93	195.68	152.85	200	165	203.21	201.99	203.09	202.39	198.99	159.5
0	1525151640000.00	-38.74	195.93	195.97	149.43	200	165.5	203.21	201.99	203.09	202.39	198.97	159.59
0	1525151700000.00	-38.88	195.93	196.24	153.56	200	165	203.21	201.99	203.09	202.39	198.9	159.4
0	1525151760000.00	-38.97	195.93	195.97	155.07	200	165	203.21	201.99	203.09	202.39	198.95	159.37
0	1525151820000.00	-38.9	195.93	195.83	151.84	200	165.5	203.21	201.99	203.09	202.39	198.94	159.4

Figure 3.12: Labelled Data

### 3.1.2.5 Labeling subsequences

In Figure 3.12, it can be noticed that for each row or each minute a label is assigned. But a minute of data cannot really define an event. As mentioned earlier, hours of MTS information will be extracted from the data and each hour will be defined as either event or non-event. Two examples of event and non- event are shown in Figure 3.13.

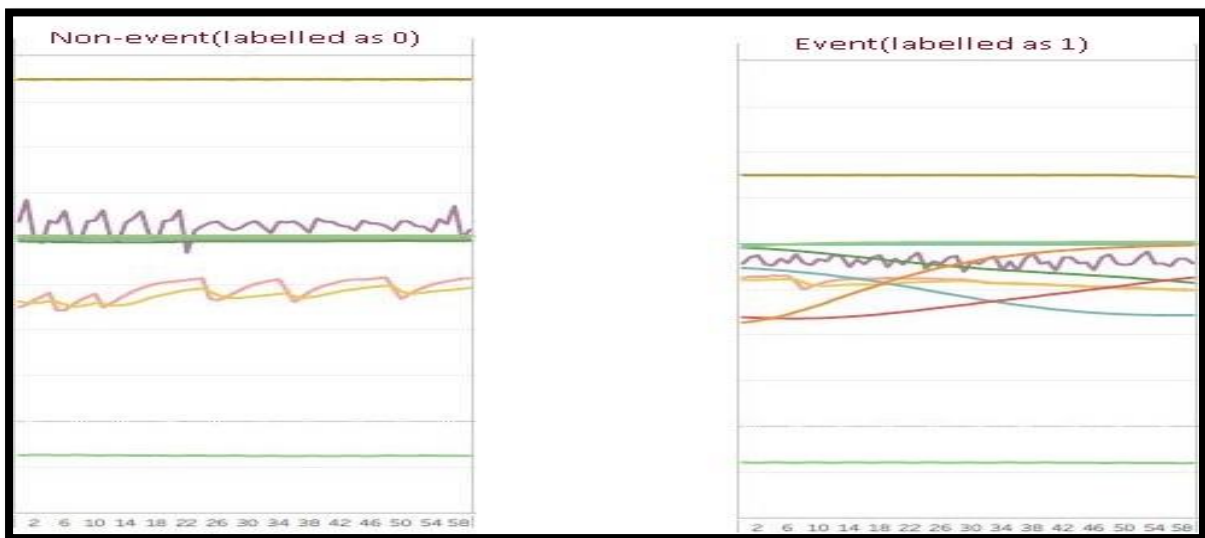


Figure 3.13: Example of an event and a non-event

So, instead of labeling a minute of MTS data or one row as one example (either event or non-event), sixty minutes or one hour of MTS data or sixty rows simultaneously will be considered as an example. Each sixty rows or 1 hour of MTS data will be considered as one subsequence. A subsequence is a piece extracted from a long sequence with a specific length; in this case the length of each subsequence is sixty minutes.

### 3.1.2.5.1 Sliding Window

Sliding window algorithm is a well-known technique to extract subsequences from a long time series. Two parameters need to be defined before using a sliding window which are window length and sliding step. As each example has a length of sixty minutes, window length will be taken as sixty. Sliding step will also be taken as sixty as after picking sixty minutes of data, if we want to move to next example, we have to move sixty minutes forward. Then another subsequence with a length of sixty minutes will be extracted from the long time series and will be labelled.

As mentioned earlier, the events are defined as an hour. When the dataset was labelled by each row or each minute, each minute was assigned a label. Now the goal is to assign a label for each sixty minutes. So, using sliding window algorithm all the subsequences will be extracted and a label will be assigned. As mentioned earlier, the example of events and non-events are defined as an hour. So, in primary labeling, all sixty rows or minutes of each hour are assigned same label. After subsequence extraction, the label of each hour will be assigned according to the labels given to the all sixty rows or minutes of that particular hour.

If a MTS,  $T$  has a length of  $n$ , window size is  $L$  and the sliding step is  $p$ , the number of extracted subsequences,  $m$  can be obtained by using the following formula [32]:

$$m = \left\lceil \frac{n-L+1}{p} \right\rceil \text{ Or } m = \frac{n-L}{p} + 1$$

In our case, the length of the time series,  $n = 264,960$ , window length,  $L = 60$ , and sliding step,  $p = 60$ . So,  $m = (264,960-60)/60+1 = 4,416$ .

So, using a window length of sixty and sliding step of sixty, 4,416 subsequences can be extracted from this time series. The pseudocode for the extraction of subsequences and the labelling of the extracted subsequences are shown in Table 3.2.

*Table 3.2: Sliding Window Algorithm*

<b>Pseudocode of Sliding Window (subsequence extraction and labeling):</b>
1. <b>START</b> the procedure
2. <b>The set of extracted subsequences, <math>X(n, L, p) := 0</math>, labels, <math>Y := 0</math></b>
3. <b><math>i := 0, m := 0</math> where <math>m</math> is the no. of extracted subsequences</b>
4. <b>While (end of a subsequence, <math>j = p*i + L) &lt; n</math> do</b>
5. <b><math>X[m] := T(p*i, \dots, j)</math> and <math>Y[m] := L(j)</math></b>
6. <b><math>i := i+1, m := m+1</math></b>
7. <b>End While</b>
8. <b>End the procedure</b>

After labeling each subsequence, the dataset can be viewed as a three dimensional dataset with dimension  $N*L*M$  where  $N$  represents  $N^{th}$  example or subsequence,  $L$  represents the window length,  $M$  represents number of sensors or input variables of the MTS. Each subsequence has



a dimension of  $L * M$ . In this case,  $L = 60$  and  $M = 12$ , so, each subsequence has  $60 * 12 = 720$  features of the MTS.

### 3.1.3 Characteristics of the labelled dataset

A detailed characteristics of the labelled dataset can be obtained by using pandas profiling tool in python. A basic summary of the labelled dataset is provided in Figure 3.14.

Dataset statistics		Variable types	
Number of variables	13	Numeric	12
Number of observations	264960	Categorical	1
Missing cells	0		
Missing cells (%)	0.0%		
Duplicate rows	1372		
Duplicate rows (%)	0.5%		
Total size in memory	26.3 MiB		
Average record size in memory	104.0 B		

Figure 3.14: Dataset Statistics

#### 3.1.3.1 Data visualization of each variable

In the labelled dataset, there are 13 variables, twelve variables measuring the temperature of twelve zones in the drying hopper are the input variables, X and the other column which is the output variable valued either 1 or 0. In order to classify the data, an efficient algorithm is needed to learn the mapping function from the input to output. In this way the goal is to learn the mapping function as approximately as possible so that when new input data are provided to the algorithm, it can predict the output of the input data.

As the issue of missing values is already taken care of in the previous section, it can be seen from Figure 3.14 that no missing values exist in the labelled dataset, instead there are some duplicate rows which is only 0.5% of the dataset. Existence of no missing values is also evident in Figure 3.15 and Figure 3.16.

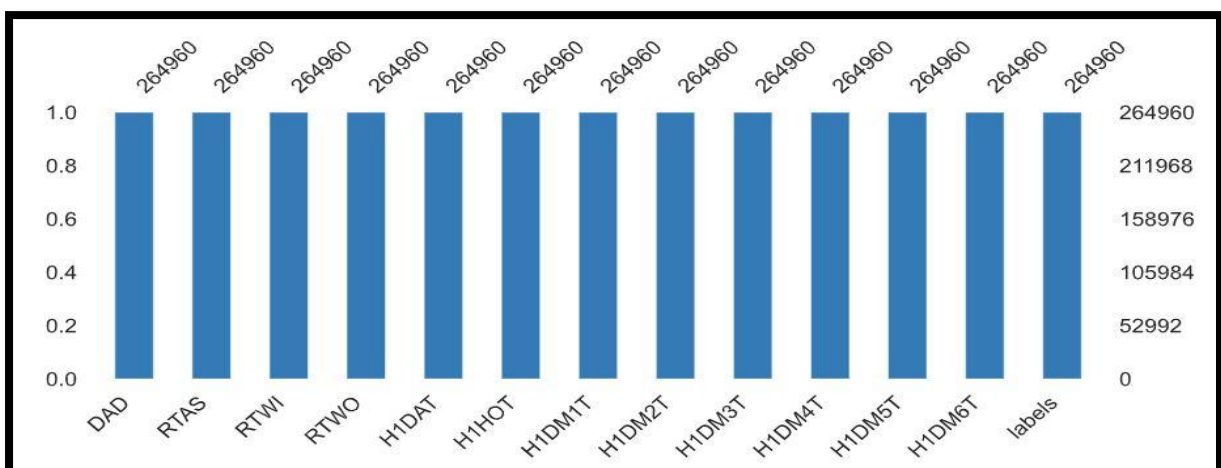


Figure 3.15: A simple visualization of nullity by column

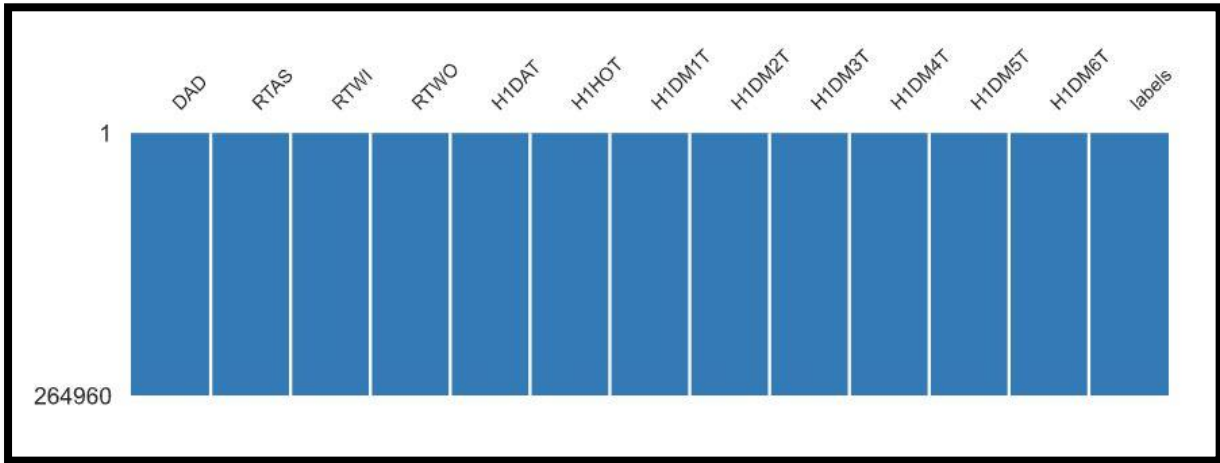


Figure 3.16: Nullity matrix

A detail inspection of all twelve input variables can also be done using pandas profiling. For example, the detailed statistics of the hopper 1 hopper outlet temperature (H1HOT) is shown in Figure 3.17, Figure 3.18 and Figure 3.19. Figure 3.17 shows the distinct value counts and missing value counts with mean, minimum and maximum value of the variable.

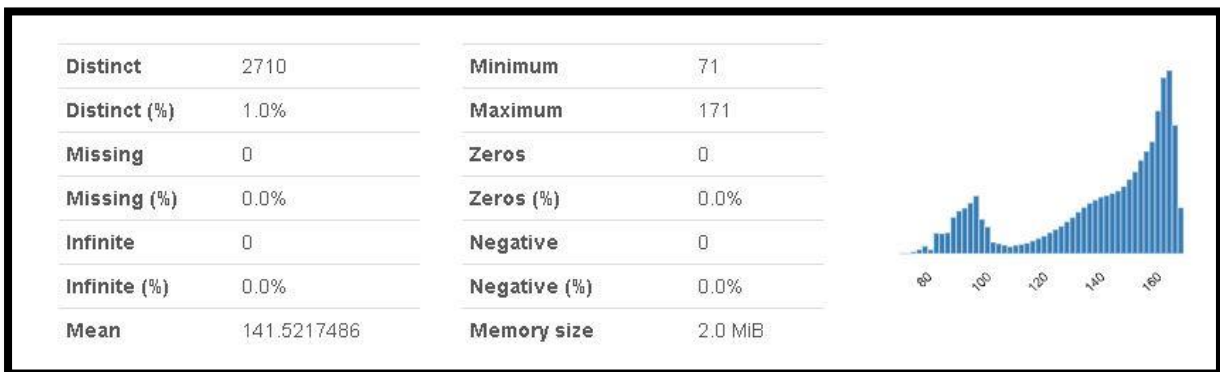


Figure 3.17: Statistical summary of Hopper 1 hopper outlet temperature

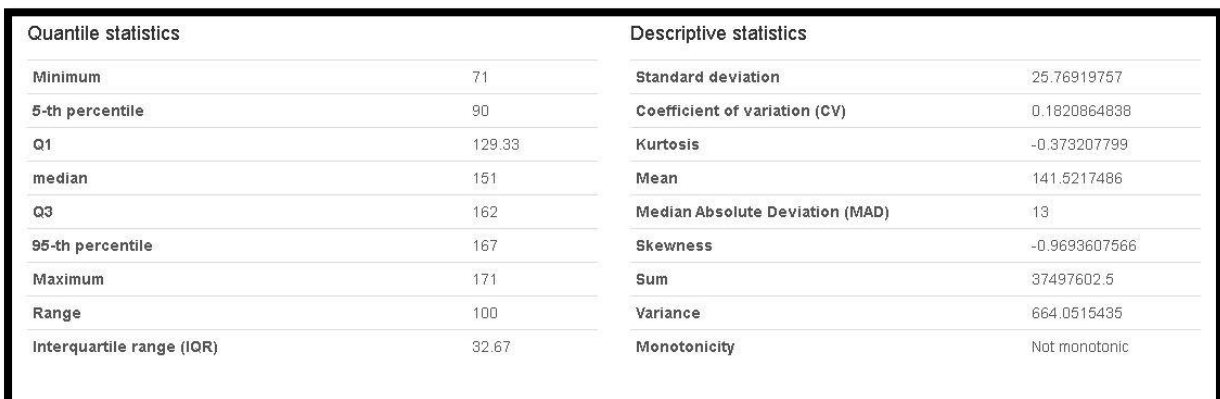


Figure 3.18: Quantile and descriptive statistics of H1HOT

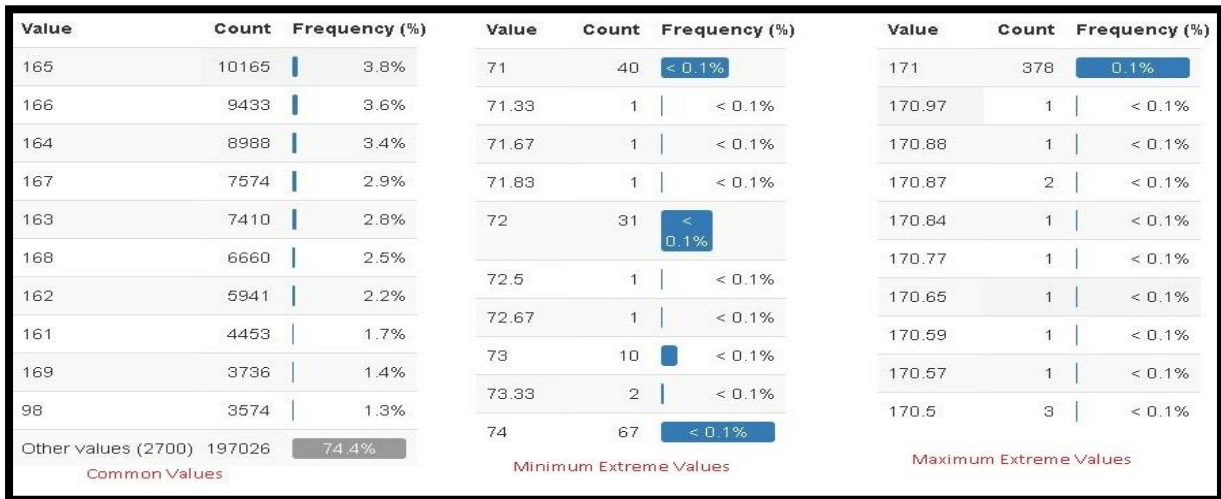


Figure 3.19: Common values and Extreme values of Minimum and Maximum of H1HOT

Figure 3.18 represents the Quantile statistics like percentile, minimum, maximum, median, range as well as interquartile range and descriptive statistics like variance, standard deviation, mean absolute deviation (MAD), skewness, monotonicity and so on. Figure 3.19 represents the common values as well as the minimum and maximum extreme values of hopper 1 hopper outlet temperature. Detailed characteristics of the other eleven variables can be inspected in similar fashion. The frequency distribution of all twelve variables can be obtained from the histogram which are shown in Table 3.3.

### 3.1.3.2 Distribution of the input and output variables

It is clearly evident from the histograms that either there is a skewness or the histogram is bimodal in shape which indicates a clear division in the temperature values of all twelve temperatures zones. For example, H1DM1, H1DM2, H1DM3, H1DM4, H1DM5, H1DAT, RTWI, DAD, and RTAS, all these temperature zones are clearly divided in two regions which indicate the events and non-events. This is also reflected in other histograms like RTWO, H1HOT, and H1DM6 which are more like a bimodal shape. When a failure event occurs, the temperature drops all on a sudden from the steady state values. This phenomenon is clearly reflected in almost all temperature zones. This is why, the approach to treat this problem as a binary classification problem is highly justified. The labeling was done in this fashion that the non-events which are very high in number are labelled as class 0 and the events which are very low in number are labelled as class 1.

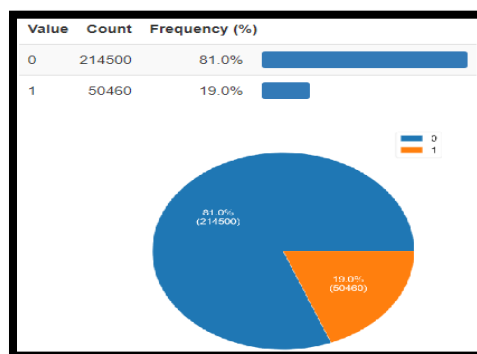
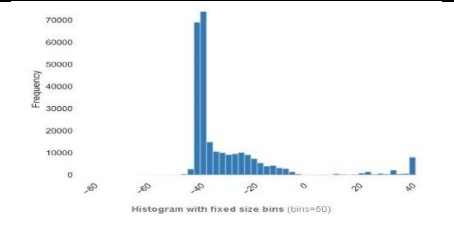
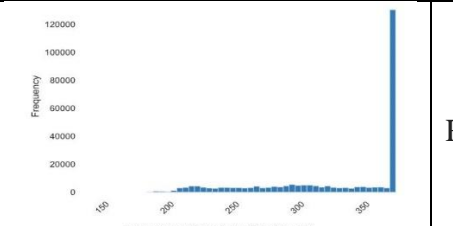
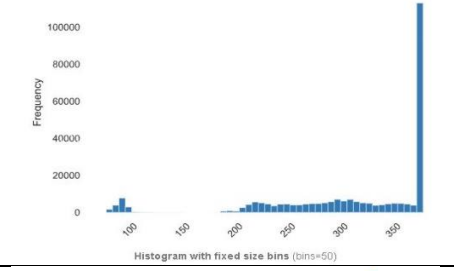
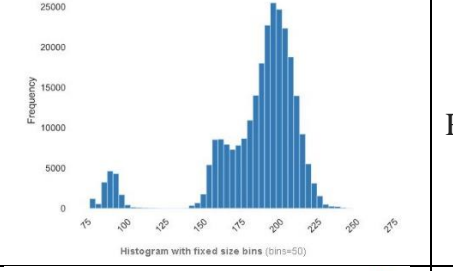
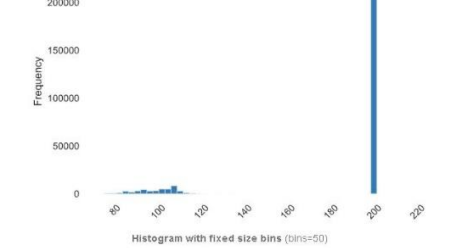
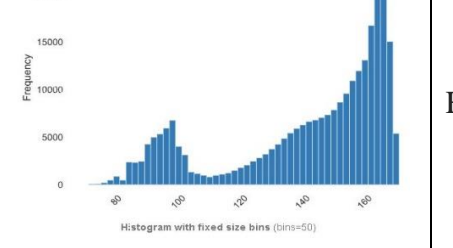
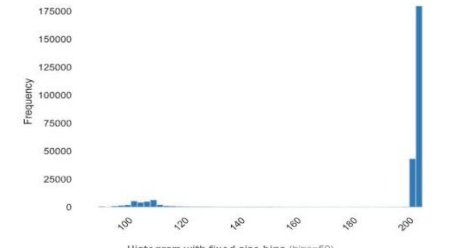
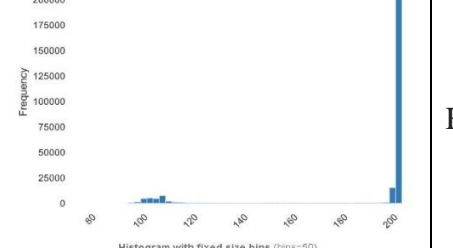
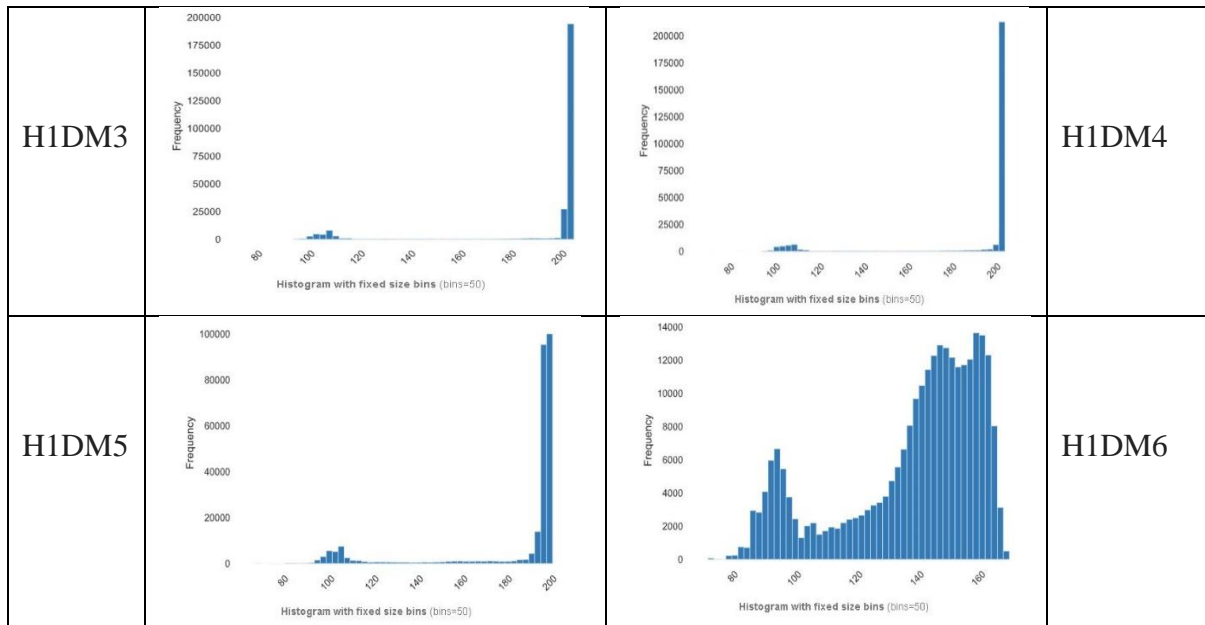


Figure 3.20: Binary classification labeling

The detail of the categorical variable, labels is shown in Figure 3.20. From the pi-chart and histogram it can be seen that only 19.1% of the data belong to the class 1 (events), where 80.9% of the data belong to the class 0 (non-events). This phenomenon further justifies the distribution shown in the histograms of the temperature zones. The value count for class 0 shown in Figure 3.20 is 214,500, as each complete example or subsequence of an event or non-event was defined as an hour previously, the number of subsequences belong to class 0 is  $214,500/60 = 3,575$ . On the other hand, the value count for class 1 is 50,460, so the number of examples or subsequences belong to class 1 is  $50,460/60 = 841$ . The total number of subsequences or examples (both events and non-events) are  $3,575+841 = 4,416$  which matches with the previous result of number of subsequences obtained by using sliding window algorithm.

Table 3.3: Histograms of the temperature zones

Temp. Zone	Histogram	Histogram	Temp. Zone
DAD			RTAS
RTWI			RTWO
H1DAT			H1HOT
H1DM1			H1DM2



### 3.1.3.3 Data transformation (Min-max scaler)

As mentioned in section 3.1.3.2, the dataset is skewed which is why transforming the data using scaler transformation technique like normalization of the input variables is highly recommended for any machine learning and deep learning algorithm which use a weighted sum of input variables and use distance measures between examples like SVM and k nearest neighbor. For example, delivery air temperature is comparatively lower than the other temperatures in drying hopper. So during learning the model might learn very large weight values or very small values. Both of these case is highly sensitive and can result in poor performance of the model during learning. According to the authors in [70], “in practice, it is nearly always advantageous to apply pre-processing transformations to the input data before it is presented to a network. Similarly, the outputs of the network are often post-processed to give the required output values”. Data normalization changes the distribution of the input variables as shown in Figure 3.21.

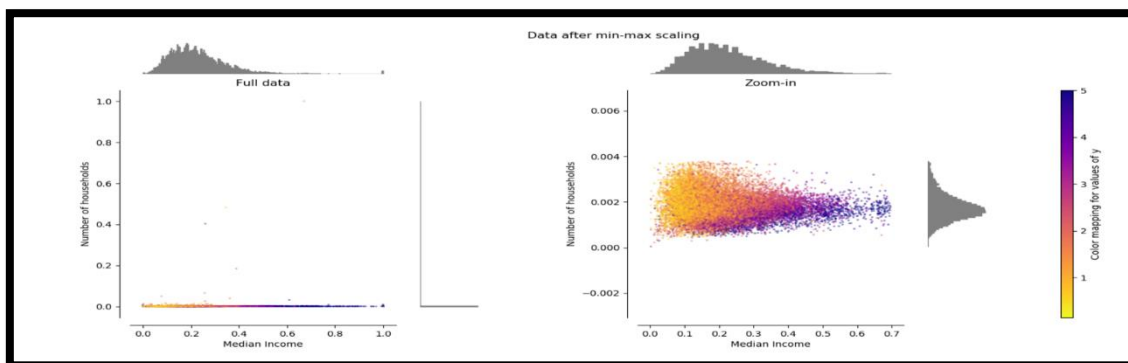


Figure 3.21: Change of distribution after data normalization

The main advantage of scaling is it makes the algorithm’s learning process easier especially deep learning algorithms. Lack of scaling or standardization sometimes result in high error gradient values which changes the updated weight values in an uncontrollable way. The most common method of scaling is data normalization. In this way, data are normalized in a range

from 0 to 1. In order to normalize the data, the maximum and minimum value of each variables needs to be identified. Afterwards a value is normalized as follows:  $\text{Normalized } X = (X - \min) / (\max - \min)$ . In python, this task is done by using MinMaxScaler for a scikit learn object. In this case, after using data normalization, the dataset looks like as shown in Figure 3.22. It can be noticed that the output variable is not used for data normalization as it is a categorical variable.

	DAD	RTAS	RTWI	RTWO	H1DAT	H1HOT	H1DM1T	H1DM2T	H1DM3T	H1DM4T	H1DM5T	H1DM6T	labels
0	0.327778	0.211111	0.395961	0.402019	0.892857	0.94	0.979083	0.980389	0.979835	0.982147	0.984107	0.897415	0
1	0.328367	0.211111	0.394530	0.384913	0.892857	0.94	0.979083	0.980389	0.979835	0.982147	0.983659	0.898738	0
2	0.328535	0.211111	0.394630	0.387895	0.892857	0.94	0.979083	0.980389	0.979835	0.982147	0.983958	0.898433	0
3	0.327357	0.211111	0.394963	0.378636	0.892857	0.94	0.979083	0.980389	0.979835	0.982147	0.983808	0.899654	0
4	0.326263	0.211111	0.394397	0.407355	0.892857	0.94	0.979083	0.980389	0.979835	0.982072	0.983734	0.896601	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
264955	0.323906	0.422997	0.554498	0.478709	0.892857	0.93	0.979865	0.980550	0.979371	0.981849	0.979108	0.847344	0
264956	0.323064	0.422997	0.555164	0.494350	0.892857	0.93	0.979865	0.980550	0.979371	0.981849	0.979705	0.851008	0
264957	0.323653	0.422997	0.554598	0.462021	0.892857	0.93	0.979865	0.980550	0.979371	0.981849	0.980152	0.855893	0
264958	0.324411	0.422997	0.553733	0.458621	0.892857	0.93	0.979778	0.980550	0.979371	0.981849	0.980675	0.858742	0
264959	0.323653	0.422997	0.555297	0.453756	0.892857	0.93	0.979691	0.980550	0.979448	0.981849	0.980675	0.861490	0

Figure 3.22: Data normalization

### 3.1.3.4 Imbalanced Dataset Issue

The division of two class clearly introduces a new issue for this dataset which is the imbalanced classification problem. In real world, imbalanced dataset is not really a surprising issue as perfectly balanced dataset are very rare. Moreover, in manufacturing detecting a failure event will always result in an imbalanced data as the failure event happens very rarely with a reasonably good setup of machineries and maintenances. But the issue is deep learning algorithm is highly ineffective to class imbalance as there is an underlying assumption of balanced data in deep neural network algorithms.

In order to tackle the class imbalance issue, this thesis will go through four major techniques which are Undersampling, Oversampling, Synthetic Minority Oversampling Technique (SMOTE) and Ensemble learning with Undersampling. These techniques are described in the next couple of sections.

#### 3.1.3.4.1 Undersampling

This technique is probably the simplest one, where a portion of the data belonged to the majority class will be dropped to make the dataset a balanced one with respect to both classes for binary classification. In this case, the number of examples belonged to the minority class is 845 and the number of examples in the majority class is 3571. For training purpose, the dataset will be divided in two segments: training examples and test examples. This will be done by using the first 80% of the dataset starting from May 1, 2018 5:01:00 AM to September 9, 2018 10:00:00 AM as training set and the rest of the dataset as test set. After train test split, the number of training examples from minority class will be 791 and the number of training examples from the majority class will be 2742. From 2742 examples of the majority class, 791 examples will be randomly chosen and these  $791 + 791 = 1582$  examples will be used for training the dataset with an efficient classification algorithm. Number of test examples after the train test split will be 883. These 883 examples will be used for the evaluation of classification algorithm.

Although the imbalanced data issue can be solved in this fashion, but a large portion of the data will be lost. This undersampling approach is called random undersampling which is very

simple and effective, but the issue is examples are removed without any concern for how useful or important they might be in determining the decision boundary between the classes. This means it is possible, or even likely, that useful information will be deleted [71]. There are other undersampling techniques which are out of the scope of this thesis.

#### 3.1.3.4.2 Oversampling

The simplest oversampling techniques is duplicating the minority class randomly over and over until the minority class is equal to the majority class and make the dataset balanced. In this case, 791 training examples from the minority class will be duplicated randomly to create 2742 examples of minority class and will be added to training dataset. Figure 3.23 shows the visuals of both undersampling and oversampling.

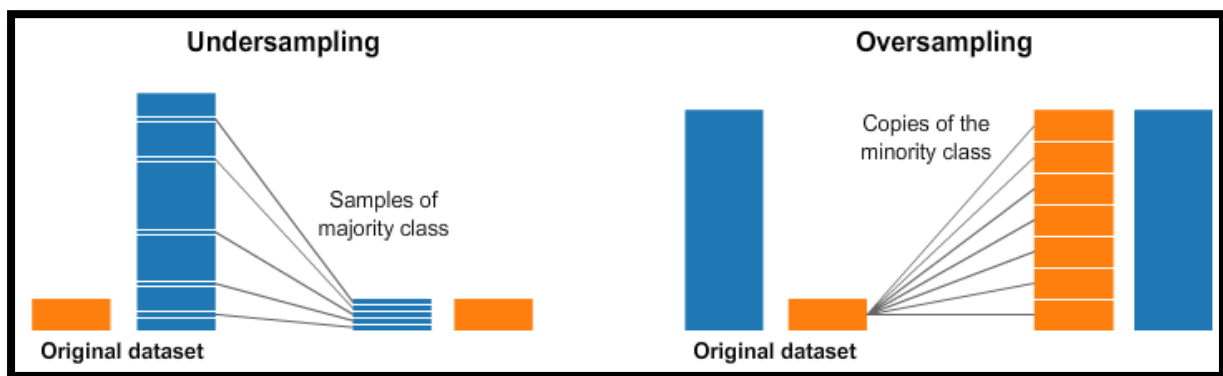


Figure 3.23: Undersampling and oversampling[72]

In this fashion, the number of training examples will be increased from 3533 to  $2747+2742 = 5484$  examples. This technique is highly prone to overfitting and the added examples do not really add any meaning to the dataset. Random oversampling is also a naïve method like random undersampling as it assumes nothing about the data and no heuristics are used [73]. Therefore, instead of duplicating examples randomly, new data can be synthesized from existing examples. This is a data augmentation technique known as Synthetic Minority Oversampling Technique (SMOTE) described in section 3.1.3.4.2.1.

##### 3.1.3.4.2.1 Synthetic Minority Oversampling Technique (SMOTE)

As the name implied, SMOTE synthesizes new examples from the minority class. The technique was first introduced and described in [74]. The underlying theme of the method is selecting examples that are nearest to the feature space. It draws a line between the nearest examples and draws a new sample along that line. According to [75], SMOTE randomly selects a minority class example and determines its  $k$  nearest neighbors. Afterwards, a synthetic example is formed by randomly selecting a neighbor from the neighbors determined previously. A line in the feature space is then used to connect the minority class example and the randomly selected neighbor. The synthetic examples are mainly formed as a convex combination of the randomly chosen minority example and the nearest neighbors. A visual of the steps of SMOTE is shown in Figure 3.24.

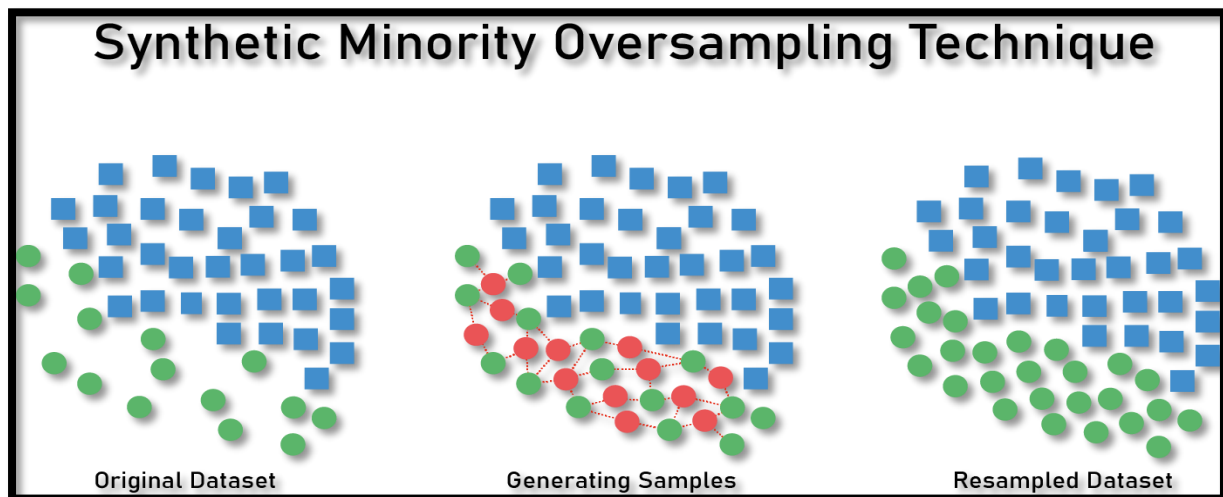


Figure 3.24: SMOTE[76]

The main advantage of this technique is it can produce as many synthetic examples as needed for creating a balanced dataset. There are many ways to use SMOTE. One common technique is to use undersampling first to reduce the number of examples in the majority class and use SMOTE afterwards for oversampling the minority class. The other approach is using the SMOTE only to oversample the minority class to balance the class distribution which will be used in this thesis.

The main reason for which SMOTE works better than random oversampling is the synthetic examples generated by SMOTE which are very reasonable compared to the duplicated examples created by random oversampling as these synthetic examples are very close to the minority examples in the feature space. But there are some drawbacks as well like SMOTE does not really care about the majority class, so it might create some ambiguous examples which cannot really be considered as representatives of the dataset.

#### 3.1.3.4.3 Ensemble Learning

Ensemble learning is a highly useful machine learning technique where multiple learning techniques are used to solve the same problem and then combine the results from all of these techniques using a majority voting technique. The main advantage is it combines the result of several techniques and tries to improve the result obtained from each technique. To deal with the imbalanced data issue, undersampling will be performed by dividing the majority class in certain segments, and each segment of the majority class will be combined with the minority class to train the dataset. A simple example of the ensemble learning with undersampling is shown in Figure 3.25 where out of 4000 training examples, 3000 examples belong to the majority class and the rest 1000 examples belong to the minority class. 3000 majority class examples are divided in three segments where in each segment 1000 examples are selected randomly. Afterwards each 1000 examples from the majority class and all 1000 examples from the minority class are combined and shuffled properly. These three sets of data are used to build a three classifiers each of which will train the dataset and then test on the test dataset to predict a class for the test examples.



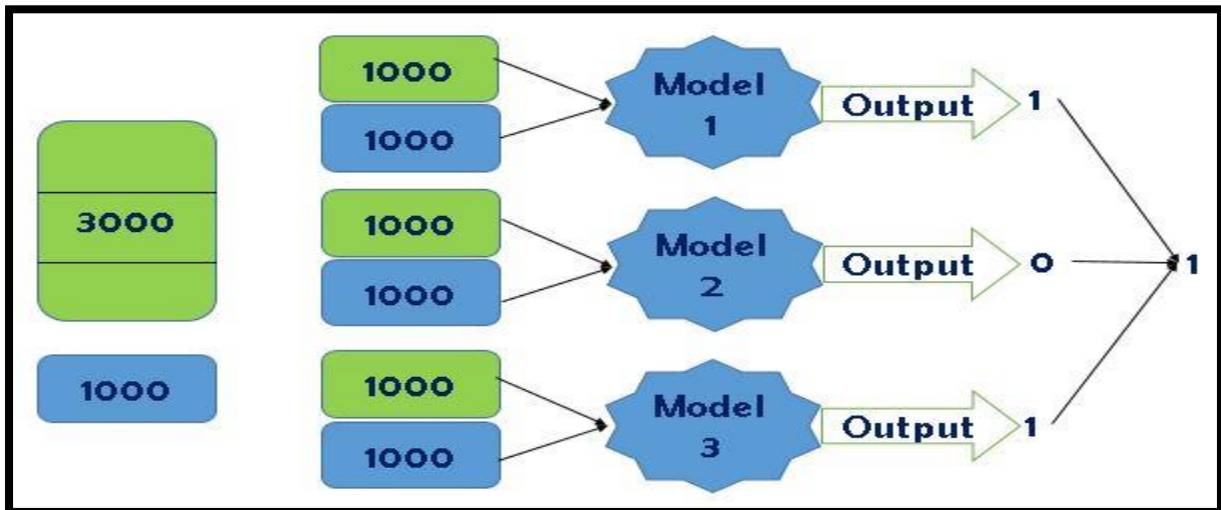


Figure 3.25: Ensemble method[77]

For example, classifier 1 predicts the class of one example as 1, classifier 2 predicts it as class 1 and classifier predicts it as class 0. As the majority of the vote belongs to the class 1, the ensemble learning will identify this example as class 1. For our case, three different approaches were taken for ensemble learning which will be presented in the result section.

### 3.2 Solution Approach

As mentioned before among several traditional approaches to classify MTS, K nearest neighbor combined with dynamic time warping is one of the best methods so far. On the contrary deep learning has gained popularity in recent years. Among several deep learning algorithms CNN has been the state of art now with several variations. Traditional multi-layer perceptron (MLP) of neural network has an issue in long time series as the length slows down the computational speed. Deep learning like CNN and RNN like long short term memory have the ability to learn the features during training and then MLP is used for classification. A nonlinear function like ReLU or Tanh or sigmoid function is used over many layers of neural networks. Each layer takes the output from its previous layer and at the end the probability distribution of each class is obtained which is used to identify the class of that example. A short overview of these algorithms and solution approaches taken for this case study will be presented in the next couple of subsections.

#### 3.2.1 Artificial Neural Network

Artificial neural network has gained immense popularity over the last few years with the advent of artificial intelligence through deep learning. As mentioned earlier, deep learning is a subfield of machine learning. The first machine learning model reference is found in 1957 by psychologist Frank Rosenblatt [78]. At that time it has limited power to capture the insight of a process through the use of a perceptron. But over the year, researchers improved it with many hidden layers and could not get the result they were looking for. The gradient back propagation algorithm has made the multi-layer deep neural network popular and now it has become one of the most advanced tools in data science. Several type of neural networks have been proposed so far like CNN (CNN), RNN (RNN), Generative Adversarial Network (GAN) and so on. The basic architecture of a neural network is shown in Figure 3.26.

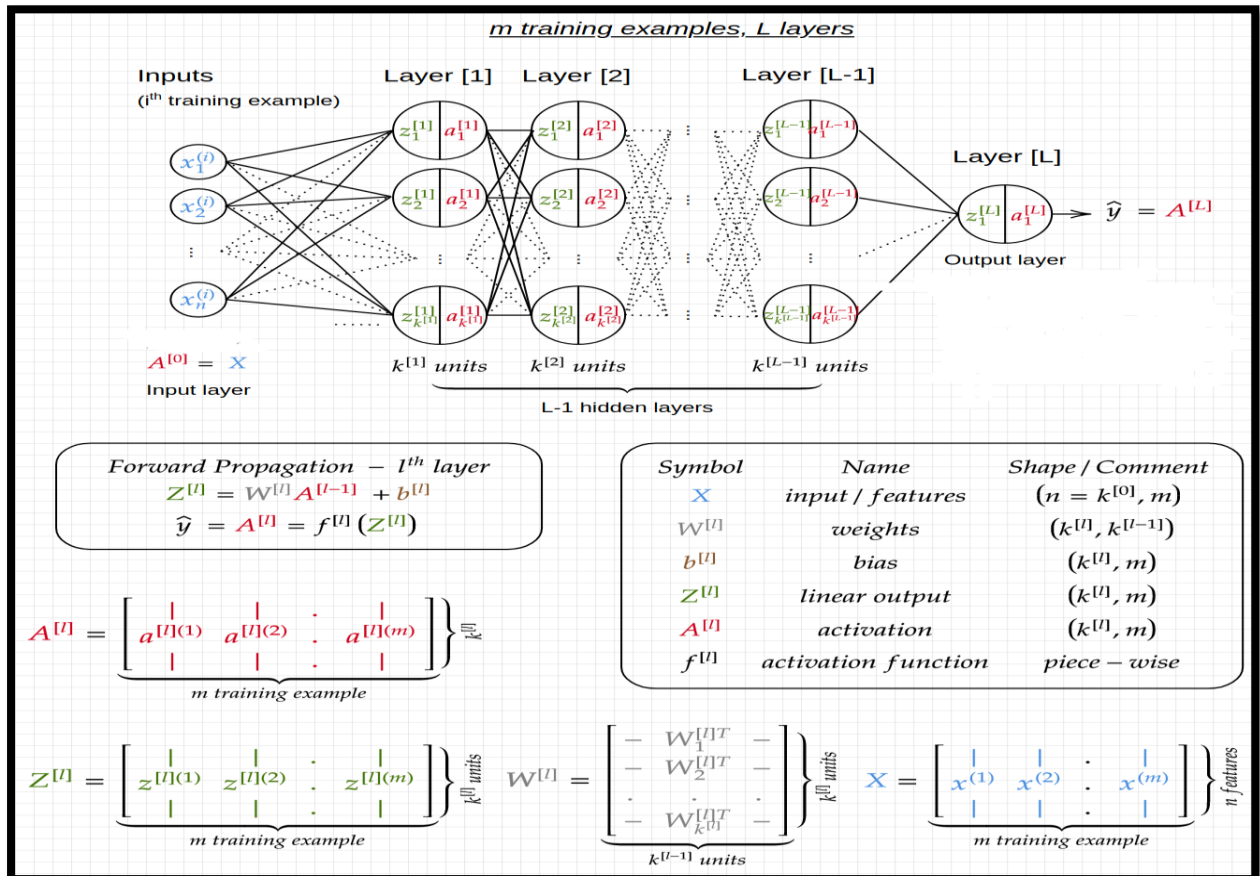


Figure 3.26: Basic Structure of a Neural Network [79]

An artificial neural network consists of several layers with input layer, some hidden layers and output layer. Each hidden layer has a specific number of nodes which build the connection from one layer to another layer. The nodes also known as neurons are modeled by weights which can be of any value. So, all inputs are updated by the weights and summed and this function is modeled from one layer to another layer. The modeling is mainly performed by the linear combination of the weights with the inputs and bias is added to the linear combination. The overall idea is more like a linear regression, but it can be of millions of input nodes or thousands of nodes in the hidden layer. An activation function is used to control the output like tanh, sigmoid or relu.

### 3.2.1.1 Forward Propagation

A typical neural network starts with m examples, each of the examples has n input features. Each layer, L has  $k^{[L]}$  units/ neurons. The first layer has  $k^{[0]} = n$  units. So, the input matrix is of shape  $(n, m)$ . In each layer, two major operations occur in the forward pass which is known as the forward propagation. Those are the linear transformation using the weights associated with each neuron and the bias and the nonlinear transformation with the use of the activation function,  $A^{[L]}$ . The forward propagation step is shown below:

$$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]} \tag{1}$$

$$\hat{y} = \mathbf{A}^{[L]} = \mathbf{f}^{[L]}(\mathbf{Z}^{[L]}) \quad (2)$$

### 3.2.1.2 Activation Function and cost function

The activation functions are mainly used for nonlinear mapping of the input data stream. The non-linearity is needed to create a nonlinear decision boundary through the nonlinear combination of the weights and the inputs. Typically used activation functions are shown in Figure 3.27.

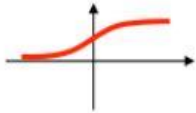
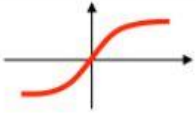
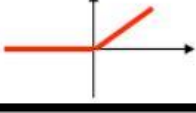
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	

Figure 3.27: Activation Function [80]

The forward propagation step is followed by a cost or loss calculation. Mainly, the idea is the weights are initialized randomly at the very beginning and then the forward propagations are performed. Afterward a log loss cost function is calculated in terms of the actual output value and the predicted value from the forward propagation. The typically used log loss functions are binary cross entropy or categorical cross entropy which are mainly used for classification. For regression problems, MSE is the commonly used measure. A typical log loss cost function is shown below:

$$J = -\frac{1}{m} \sum_{i=1}^m (y_i \log(a^{[L](i)}) + (1 - y_i) \log(1 - a^{[L](i)})) \quad (3)$$

### 3.2.1.3 Backward Propagation

In backward propagation the derivative of the loss or cost function is calculated with respect to the neural network parameters  $\mathbf{W}$  and  $\mathbf{b}$ . These gradients are then used to update weights and biases. There are several optimization algorithms available for the calculation of the optimum weights and biases. Among them gradient descent is perhaps the most popular one. There are various versions of the gradient descent used commonly in numerous applications like RMSprop, ADAM optimization algorithm, gradient descent with momentum. Among them, gradient descent with momentum uses past steps to determine the direction of the gradient descent. ADAM is one of the best optimizers in noisy dataset and has been very popular in recent years over stochastic gradient descent in many cases. There are some regularization techniques to reduce over fitting like L2 regularization where the cost function is penalized to avoid the risk of highly over fitted model and the dropout techniques. Another useful technique to reduce overfitting is dropout. In dropout technique, nodes in a hidden layer are randomly shut down to use random different subsets of the neurons. The main goal is to learn more robust features of the dataset. The basic equations of the backward propagation is shown below:

$$dZ^{[L]} = dA^{[L]} * g^{[L]'}(Z^{[L]}) \quad (4)$$

$$dW^{[L]} = \frac{\partial J}{\partial W^{[L]}} = \frac{1}{m} dZ^{[L]} A^{[L-1]T} \quad (5)$$

$$db^{[L]} = \frac{\partial J}{\partial b^{[L]}} = \frac{1}{m} \sum_{i=1}^m dZ^{[L](i)} \quad (6)$$

$$dA^{[L-1]} = \frac{\partial J}{\partial A^{[L-1]}} = W^{[L]T} dZ^{[L]} \quad (7)$$

After obtaining the derivatives parameter updates are performed in the following way:

$$W^{[L]} = W^{[L]} - \alpha dW^{[L]} \quad (8)$$

$$b^{[L]} = b^{[L]} - \alpha db^{[L]} \quad (9)$$

Where  $\alpha$  = learning rate which is one of the most important hyper parameters to choose during training.

After training the training examples over multiple iterations and when the cost is reduced significantly, it indicates the neural network is learning the parameters effectively. Afterwards, the learned parameters are used on a test set to check the performance of the model. There are several performance measures like accuracy for classification or RMSE for the regression problems.

Neural networks are prone to over fitting and under fitting. There are several techniques to prevent over fitting like L2 regularization or dropout as mentioned before. In order to reduce under fitting several measures like using a bigger network with many hidden layers with many neurons or training a longer period of time.

Two very important hyper parameters in neural network are batch size and no. of epochs. Batch size is the no. of training examples the learning algorithm will use to update the parameters. If batch size is 1, it is called stochastic gradient descent, if it is m (no. of training examples), this is called batch gradient descent and if it is between 1 and n, this is called mini batch gradient descent which is extremely useful in most type of problems. The typical values for the batch size 32, 64, 128 etc. The number of epochs is a hyperparameter that defines the number of times the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset can update the internal model parameters. An epoch is comprised of one or more batches. It can be viewed as a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples [81].

In this thesis, a very basic artificial neural network like multilayer perceptron will be used along with other approaches in search of the best algorithm for this specific drying hopper case. The traditional MLP has a similar kind of structure as shown in Figure 3.26. The input layer has a

shape of (n\*m) where n is the number of features in each example and m is the number of examples which are 720 and 3533 respectively in this case. A typical one MLP with one hidden layer is shown in Figure 3.28.

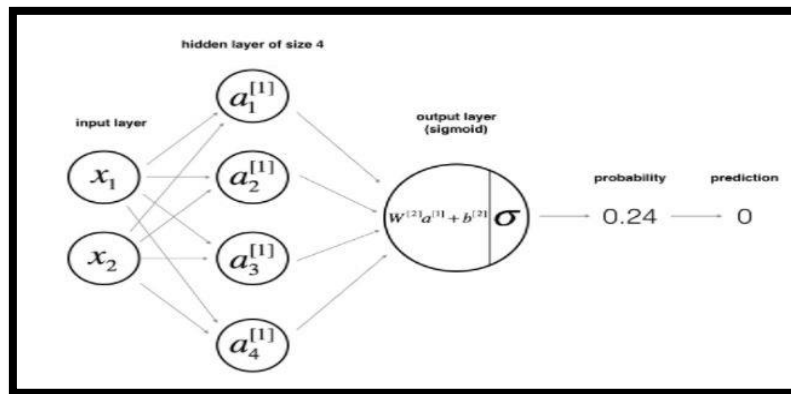


Figure 3.28: MLP with one hidden layer[82]

Here, one important thing needs to be remembered is the training dataset was actually a three dimensional dataset with shape of (3533, 60, 12). For traditional MLP, this dataset is transformed to two dimensional dataset to align with the requirement of the input layer of a MLP. Therefore, the dataset is reshaped from (3533, 60, 12) to (3533, 720). The no. of hidden layers and no. of nodes in a hidden layer, both of these hyperparameters will be finalized during the training period through trial and error. Too many neurons in the hidden layer can cause overfitting and it takes to much time for the processing unit during training period. With the aid of loss vs. epoch and accuracy vs. epoch graph, different set of neuron in the hidden layer can be used for trial and error. Systematic grid search or random search of the hyperparameters can take care of this issue. Before training period, these hyperparameters are tuned with the model to have an overview of the performance of the model in terms of different combination of hyperparameters.

The last layer of the MLP will have only one neuron embedded with sigmoid activation function. The class of a particular example will be obtained in the following way.

$$Y_{predicted\ class} = \begin{cases} 1; & \hat{y} > 0.5 \\ 0; & otherwise \end{cases} \quad (10)$$

### 3.2.2 Convolutional Neural network (CNN)

The most common two neural networks used over the last few years are CNN and RNN and there has been a lot of variations developed to tackle with a variety of problems. CNN gain much popularity for their contribution to computer vision problems. This is why CNN have been used extensively in image recognition tasks, natural language processing, and speech recognition. The speech recognition and natural language processing both can be seen as some sort of sequential learning problems. This is why although initially developed for computer vision problems, CNN has been one of the most popular deep neural networks for dealing with time series problems especially MTS problems. A typical example of the CNN application on time series data is shown in Figure 3.29 and Figure 3.30.

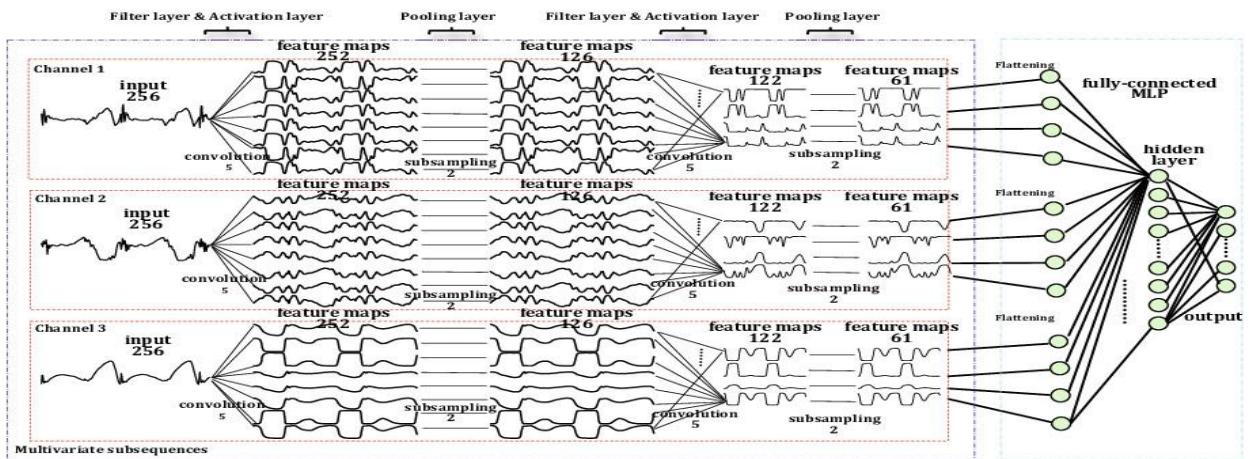


Figure 3.29: Multi channel Deep CNN application on time series [32]

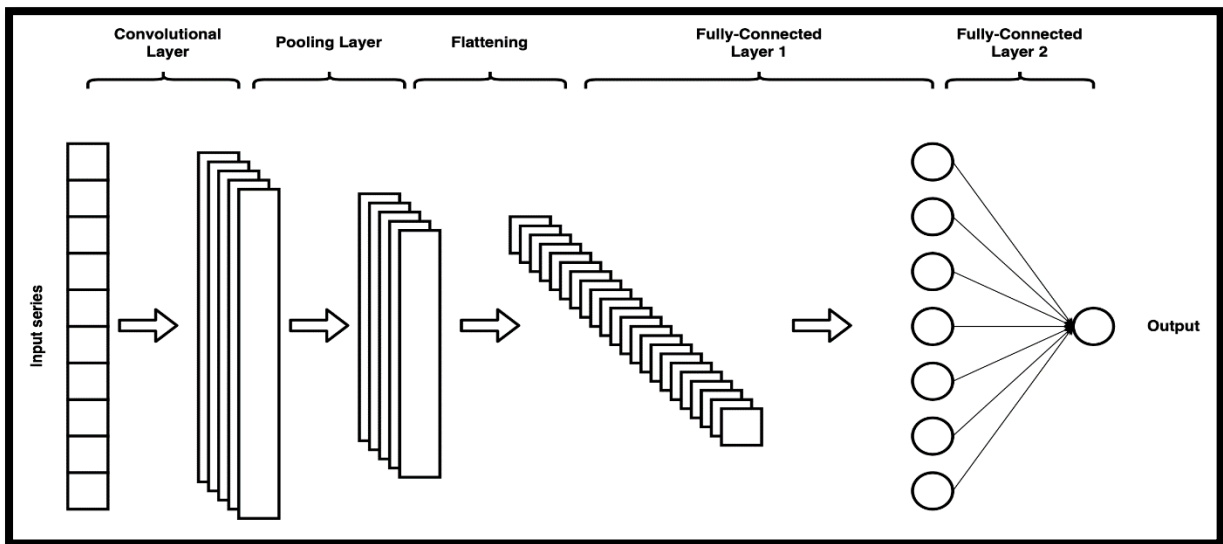


Figure 3.30: CNN for time series classification[83]

Originally, CNN was developed for image processing where the filter matrices are used for the dimensionality reduction of the large 2D image with pixel values. For time series, the filters used are 1D which are used on the subsequences of a long time series. For a MTS, various approaches can be taken for using CNN. As shown in Figure 3.29, each variable can be processed separately using CNN through convolutional or filter layer and pooling layer. In this case, the input subsequence has a length of sixty. For a deep CNN these convolutional layer and filter layers can be used multiple times until the shape of the input subsequence reduced to a desired value without losing any meaningful feature. Generally a convolutional layer is always followed by a pooling layer. The reason is the pooling layer can provide a quick summary of the features which are present in a region of the feature map generated by the convolutional layer. After a certain amount of dimensionality reduction of the input subsequences, all those subsequences with reduced length are flattened and joined together to commence a fully connected layer. From this layer, the network will work like a traditional MLP as discussed in section 3.2.1.

Another approach is rather than processing each variable separately, twelve variables with a subsequence length of sixty will be processed simultaneously and flattened after extracting meaningful features through convolutional layer and pooling layer.

### 3.2.2.1 Convolutional Layer

A CNN mainly consists of three layers which are the convolutional layer, pooling layer and fully connected layer. In convolutional layer the input feature matrix will be used to perform a convolution with a fixed filter. The filter is initially defined randomly which are trained over the training process to obtain the desired values of the filter with the help of a cost function. There are actually two filter parameters, one of them is the no. of filters and the other one is filter size or kernel size. Typical values of kernel size for 1D convolution operation in time series application are 3,5,7 and so on which depends on the length of the input subsequence.

Stride and padding are two most important parameters to be decided during convolution operation. Stride is mainly used to define how many units the filter will shift during convolution operation over the input feature vector. A common scenario in the convolution operation is the input feature vector size will go down continuously over many layers of convolution. The problem is when the size of the input matrix will be reduced, there is a possibility that many important features might be lost. To overcome this, padding operation is performed so that even with a smaller filter than the input vector, the size of it will be the same. It will help the convolution operation to go slowly over the layers of the neural network. So, two types of valid operations exist, one of them is called valid, where padding is performed and it is not very common in time series analysis and the other one is same, where no padding operations is performed, so the input feature vector shape decreases with the convolution operation. A typical example of the convolution operation which is more like a sum product operation is shown below.

For example, the input subsequence is [5, 4, 9, 2, 7, 6] and the filter we are using has a kernel size of 3 which is (2, 3, 1)

So the output subsequence will be [5\*2+4\*3+9\*1, 4\*2+9\*3+2\*1, 9\*2+2\*3+7\*1, 2\*2+7\*3+6\*1] or [31, 37, 31, 31].

Each term of the output subsequence is obtained by using the following formula

$$n^{[L]} = \left\lfloor \frac{n^{[L-1]} - f^{[L]}}{s^{[L]}} + 1 \right\rfloor \quad (11)$$

### 3.2.2.2 Pooling layer

In the pooling layer with the help of a sliding window the dimension reduction of the input feature matrix is performed through the use of max pooling or average pooling. Padding and stride also need to be specified in this layer which are hyper parameters in pooling operation. The previous example can be shown to obtain the output subsequence after the pooling operation by using the same formula. The choice of max pooling or average pooling is also a hyperparameter.

For example, kernel size is 2 for the pooling operation. The possible output subsequences for the pooling layer is given below.

Max pooling: [37, 37, 31] and Average pooling: [34, 34, 31].

### 3.2.2.3 Fully Connected Layer

The fully connected layer is mainly used to use the result of previous two layers to learn the nonlinear features. This step is more like the typical MLP described in the previous section. The remaining steps like forward propagation, backward propagation, and weight updating using gradient descent are the same as a typical neural network.

### 3.2.3 Recurrent Neural Network

CNN has gained immense popularity in computer vision which is because of the strong capturing power of the CNN in terms of spatial feature. In time series classification, the same intuition is also applied where a convolutional and a pooling layer tries to extract the spatial features of a subsequence or the abstract concept of a subsequence. But one shortcoming of CNN is it ignores the temporal information or dependency of one time step on the previous values in a time series. Therefore, for sequence modeling, RNN has been developed and has gained widespread success in many sequence modeling applications like machine translation, natural language processing, time series forecasting or classification and so on.

A traditional neural starts with a fixed size input, so when it comes to a situation when the input can be of variable lengths, traditional neural network does not work. RNN is developed in such a way it can process variable length inputs. There are several type of RNN architectures as shown in Figure 3.31.

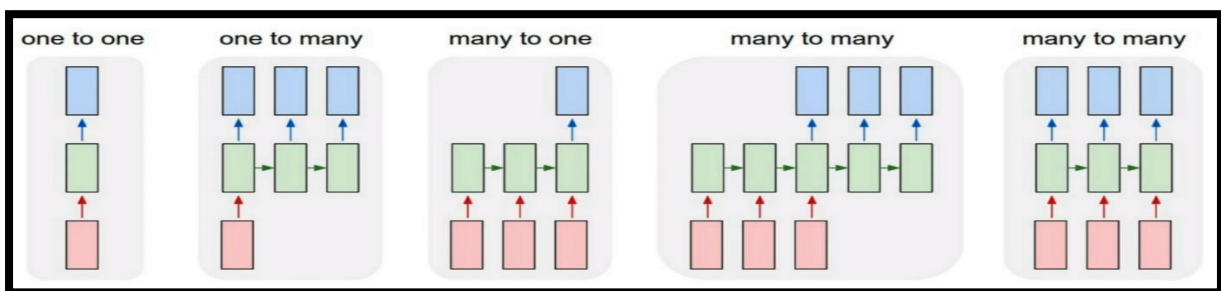


Figure 3.31: Different types of RNN architecture[84]

- The first architecture as shown in Figure 3.31 is the traditional feed forward neural network which is known as one to one architecture with fixed size input and output length.
- The second architecture is one to many where the output can be of variable length. The common example of this architecture is image identification where the input is an image and the output is a text describing the image.
- The third architecture referred to as many to one which is used for sentiment classification or time series analysis like forecasting and classification. In this case, many to one architecture will be used where the input is a subsequence and the output will be the class of that subsequence.
- The last two architectures known as many to many which is commonly used for language translation like Google Translator where the input and output both can be of variable lengths. For time series analysis, this architecture can also be used to forecast time series like forecasting the stock price of next seven days at a time.

A general computational graph of RNN is shown in Figure 3.32.



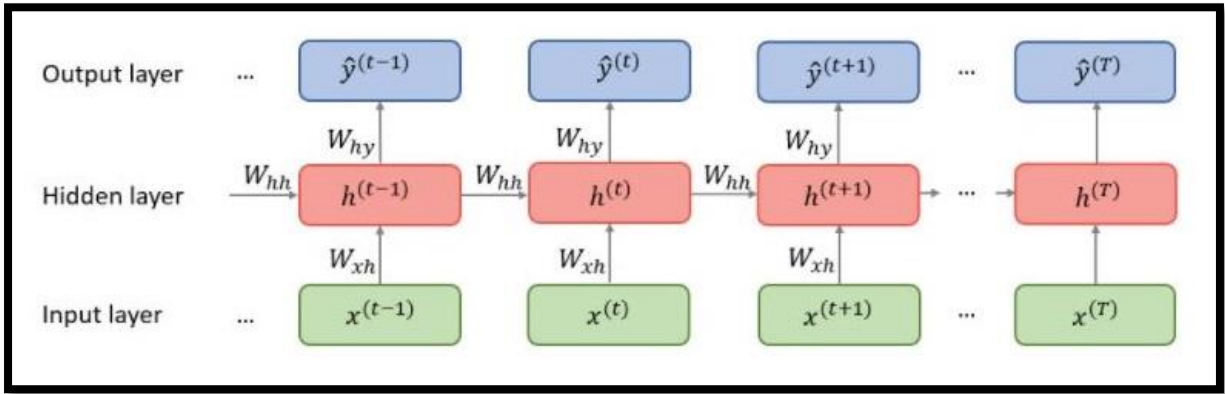


Figure 3.32: Computational Graph of RNN [85]

As shown in Figure 3.32, between input layer and output layer there is a hidden state which is the output of the input state. These hidden states take the information from the previous hidden state as well as the new input and then transmit it to the next hidden state. In this way, a recursive structure with time dependence is generated where each hidden state depends on the previous state. There are three weight matrices used in this architecture which are  $W_{xh}$ ,  $W_{hh}$  and  $W_{hy}$  and the bias terms  $b_h$  and  $b_y$  which are shared temporally. A cell state or hidden state  $h(t)$  is a function with weights of the input  $x^{(t)}$  and the previous hidden state  $h^{(t-1)}$  [86]. This function and the set of parameters are same at every time step and with every time step, hidden state  $h^{(t)}$  is updated as shown in Figure 3.32. The basic set of equations of a RNN are shown below.

$$h^{(t)} = f_w(x^{(t)}, h^{(t-1)}) \tag{12}$$

$$h^{(t)} = g_1(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h) \tag{13}$$

$$\hat{y}^{(t)} = g_2(W_{hy}h^{(t)} + b_y) \tag{14}$$

Back propagation in RNN is different from the traditional feed forward neural network which is known as Back Propagation through Time (BPT). A computational graph of BPT is shown in Figure 3.33 obtained from [87].

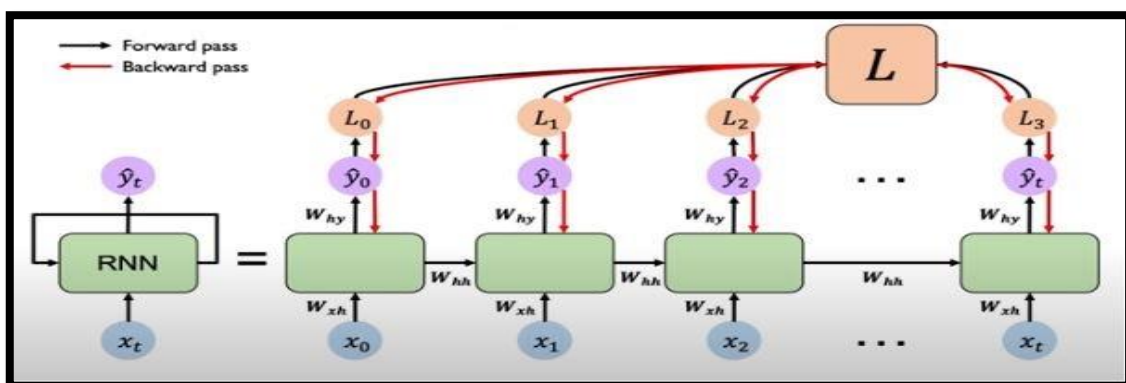


Figure 3.33: Back propagation through time[87]

Back propagation through time is performed at every single time step  $T$ , the basic equation for BPT is shown below [88]. More details on the BPT can be found at [84] where each step of gradient calculation and weight updating can be found.

$$\frac{\partial L^{(T)}}{\partial W} = \sum_{t=1}^T \frac{\partial L^{(T)}}{\partial W_{(t)}} \quad (15)$$

When computing the gradient with respect to  $h_0$  involves many factors of  $W_{hh}$  and consequently there is a long term dependency when handling the repeated gradient computation back in time. The two common problems faced by RNN are vanishing and exploding gradient.

- When gradient becomes too small, updating of the parameters do not add any significant information to the process as it becomes insignificant with not really a major update of the weights. So with respect to the number of layers with long sequence, RNN suffers from long term dependency. This situation of insignificant weight update due to very small values of gradient is known as vanishing gradient.
- The second problem known as exploding gradient happens when gradient becomes too large with exponential growing, the resulting weight after the update becomes too large.

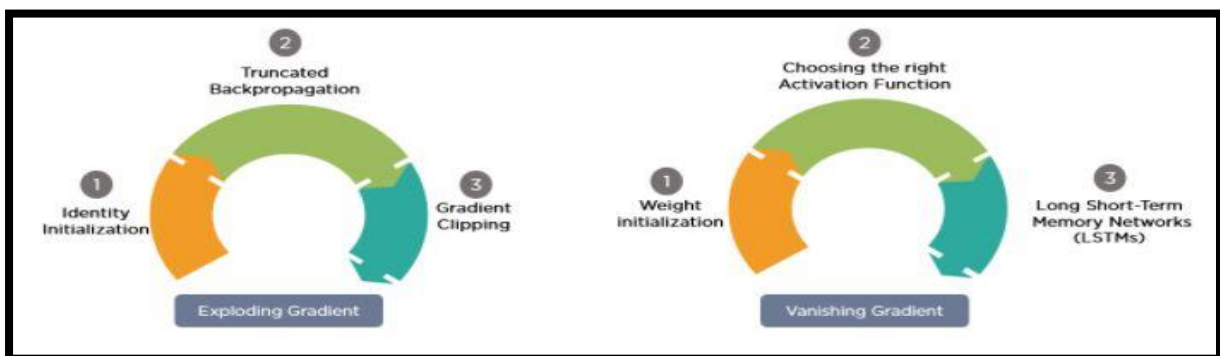


Figure 3.34: Vanishing and Exploding Gradient [89]

The common techniques to overcome these errors are shown in Figure 3.34. In this thesis, the most popular version of RNN, LSTM will be used to get rid of the vanishing gradient problem associated with RNN especially in long time series which is similar to the dataset presented in this thesis. A short description of LSTM will be provided in section 3.2.3.1.

### 3.2.3.1 Long Short Term Memory

Long Short Term Memory (LSTM) is a special kind of RNN which uses a complex recurrent unit with gates so that the gates can control what information will pass through the next step. RNN does not bother whether a particular information is important or not. LSTM relies on a gated cell in order to track information throughout many steps[87]. The gated memory unit controls the information flow and thus carry on with the selective long term information and forgets short term information which the model finds unnecessary. This is how LSTM solves the RNN's problem of exploding gradient and vanishing gradient. LSTM uses repeating chain like structure where each unit is repeated over the architecture, but layers are present in each unit which interact with each other. Figure 3.35 shows how a single unit of LSTM works.

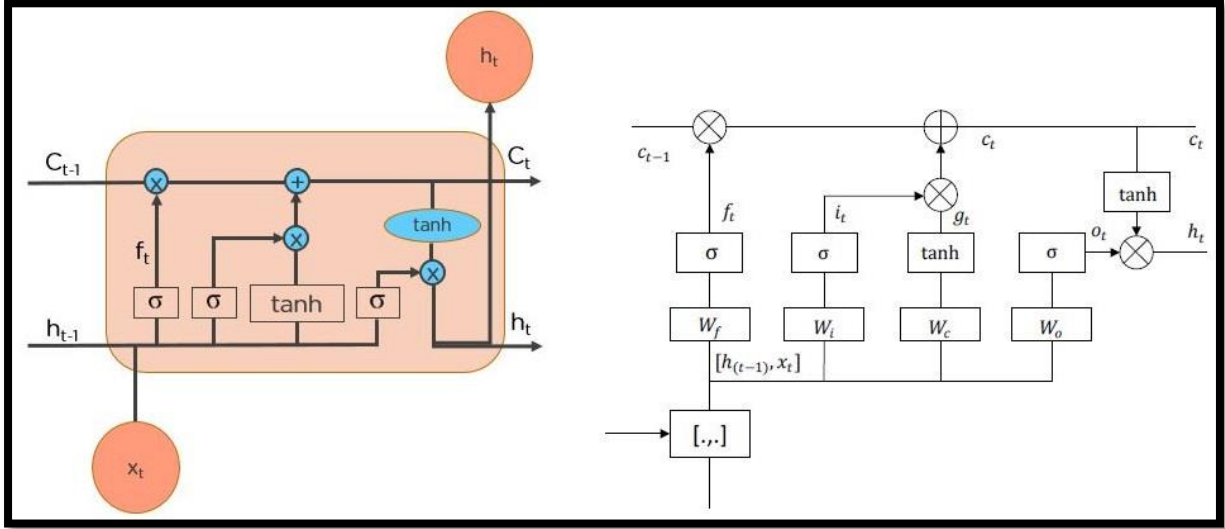


Figure 3.35: LSTM structure[31], [89]

As shown in Figure 3.35, there are three different gates in a LSTM which are forget gate, input gate and output gate.

**Forget Gate:** This gate controls the information flow from the previous timestamp. The basic equation at this gate is provided below.

$$f_t = \sigma(w_{if}x_t + b_{if} + w_{hf}h_{t-1} + b_{hf}) \quad (16)$$

Where  $x_t$  = input timestamp at time  $t$ ,  $w_{if}$  = weight matrix for the input,  $h_{t-1}$  = hidden state at time  $t-1$ ,  $w_{hf}$  = weight matrix for the hidden state;  $b_{if}$  and  $b_{hf}$  are the bias terms associated with input and hidden state respectively. Sigmoid function as shown in Figure 3.27 is used after the linear combination of weights and biases which will convert the value of  $f_t$  between 0 and 1. If  $f_t = 0$ , the network will forget the information and if  $f_t = 1$ , the network will remember the information for further processing.

**Input gate and New Information Processing:** The input gate is used to quantify the information obtained from the newest input [90]. The basic equation is provided below.

$$i_t = \sigma(w_{ii}x_t + b_{ii} + w_{hi}h_{t-1} + b_{hi}) \quad (17)$$

Where  $w_{ii}$  = weight matrix for the input,  $w_{hi}$  = weight matrix for the hidden state;  $b_{ii}$  and  $b_{hi}$  are the bias terms associated with input and hidden state respectively for the input gate. A sigmoid function is used in this gate as well to convert the input timestamp between 0 and 1.

Afterwards, the new information which will be passed to the cell state needs to be processed. The basic equation is shown below.

$$g_t = \tanh(w_{ig}x_t + b_{ig} + w_{hg}h_{t-1} + b_{hg}) \quad (18)$$

$$c_t = f_t c_{t-1} + i_t g_t \quad (19)$$

Where  $w_{ig}$  = weight matrix for the input,  $w_{hg}$  = weight matrix for the hidden state;  $b_{ig}$  and  $b_{hg}$  are the bias terms associated with input and hidden state respectively for the update of new

information,  $c_{t-1}$  = cell state at timestamp t-1. Here tanh activation function as shown in Figure 3.27 is used. The reason is the tanh activation function converts the value of  $g_t$  between -1 and 1. If the value of  $g_t$  is negative, the new information will be subtracted from the previous cell state and if it is positive, it will be added to the new information.

**Output gate:** The basic equations for the output gate is shown below.

$$O_t = \tanh(w_{io}x_t + b_{io} + w_{ho}h_{t-1} + b_{ho}) \tag{20}$$

$$h_t = o_t * \tanh(c_t) \tag{21}$$

The output value used a sigmoid function like the previous gates and turn the output into a value between 0 and 1. Afterwards, the current hidden state will be calculated using the output and the cell state at time  $t$  with a tanh activation function which indicates the hidden state is function of the long term memory  $c_t$  and the current output.

In this thesis, LSTM will be used to capture the time dependency of the events occurred in the drying hopper. For a large sequence, LSTM will figure out which information might be necessary for this case and use that information to maintain the time dependency. In addition to that, as an improvement over RNN, LSTM has the high capturing power in terms of temporal information. But it does not really care about the internal features like spatial feature which is mainly handled by CNN. So, in the next section combination of both LSTM and CNN will be explored.

### 3.2.4 Combination of CNN and LSTM

Combination of CNN and LSTM which is commonly known as CNN LSTM is mainly used for capturing internal features of input like time series or sequence of images through CNN layer and LSTM layer for sequential learning simultaneously. The architecture also includes a deep neural network like MLP at after the CNN and LSTM layer. The architecture of CNN LSTM is shown in Figure 3.36 [91].

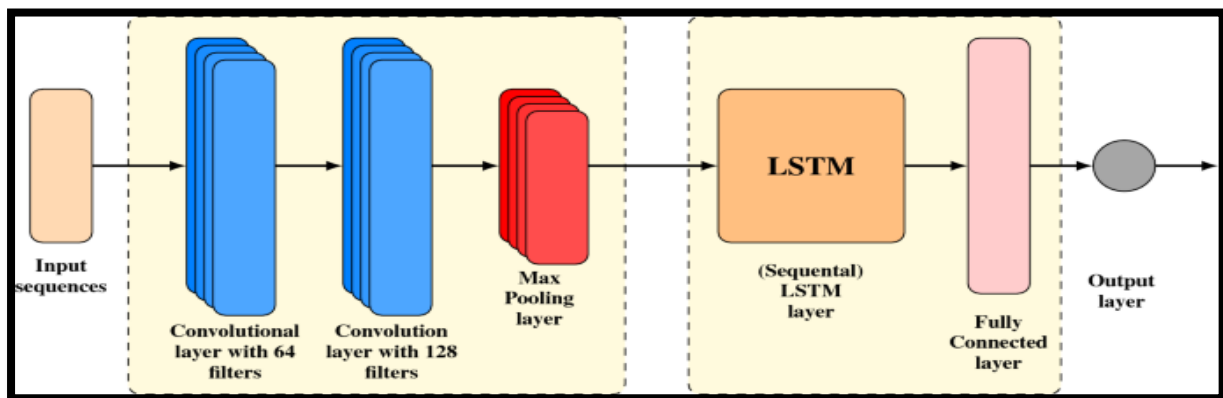


Figure 3.36: CNN LSTM architecture[91]

CNN LSTM has been used for sequence analysis problems in various ways in the literature. In [92], the authors trained three models CNN, LSTM and deep neural network like MLP separately and combined the outputs from these three networks using a combination layer which is similar to ensemble learning.

CNN LSTM works well on dataset which have 2D structure or pixels in an image or 1D structure like words in sentence. In addition to that, the input or output or both has a temporal structure. In this case, drying hopper temperature profile has both spatial and temporal features. The spatial features are for example, the peak temperature value of an event or the specific pattern observed in an event. Moreover, as the drying hopper temperature values are recorded over the period, any event or non-event has temporal dependency. For example, this dataset is processed in such a way that each event or non-event is of one hour length. Each time step in an event is temporally dependent on the previous time step. So, theoretically it makes more sense to use the CNN LSTM model. So in simple block diagrams, the architecture looks like as follows.



A CNN LSTM model uses the input subsequences as blocks, extract features from each block and then uses LSTM to use the flattened extracted features and identify its own features before a final mapping on each class is made [93]. In this case, the input subsequence of length 60 will be divided into four subsequences each of which will have a length of 15 minutes.

### 3.2.5 Machine Learning Algorithms

In this thesis, several machine learning algorithms will be used along with the deep learning techniques to perform a comprehensive evaluation of these algorithms on drying hopper use case dataset. Both non-linear algorithms like k-nearest neighbors, classification and regression tree, SVM and naïve bayes as well as ensemble algorithms like bagged decision trees, random forest, extra trees and gradient boosting will be used to evaluate the performances of these algorithms compared to the deep learning algorithms and traditional methods like dynamic time warping with k nearest neighbor. In the next two subsection, a brief background of SVM and random forest will be provided to have an overview of both non-linear method and ensemble method.

#### 3.2.5.1 Support Vector Machine (SVM)

SVM is a non-linear machine learning algorithm used for supervised learning like classification or regression problem mostly for classification problems. The main idea is each data points of all classes are plotted in an n-dimensional space with n number of features. The final goal is to obtain the hyper plane that separates the classes. A simple example of SVM is shown in Figure 3.37 where three hyperplanes are shown among which optimum hyperplane will be chosen which separates the two classes better.

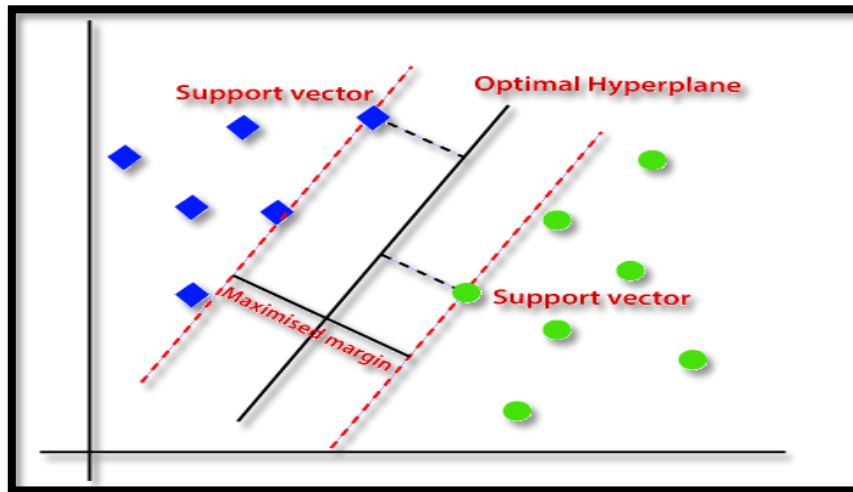


Figure 3.37: Support Vector Machine [94]

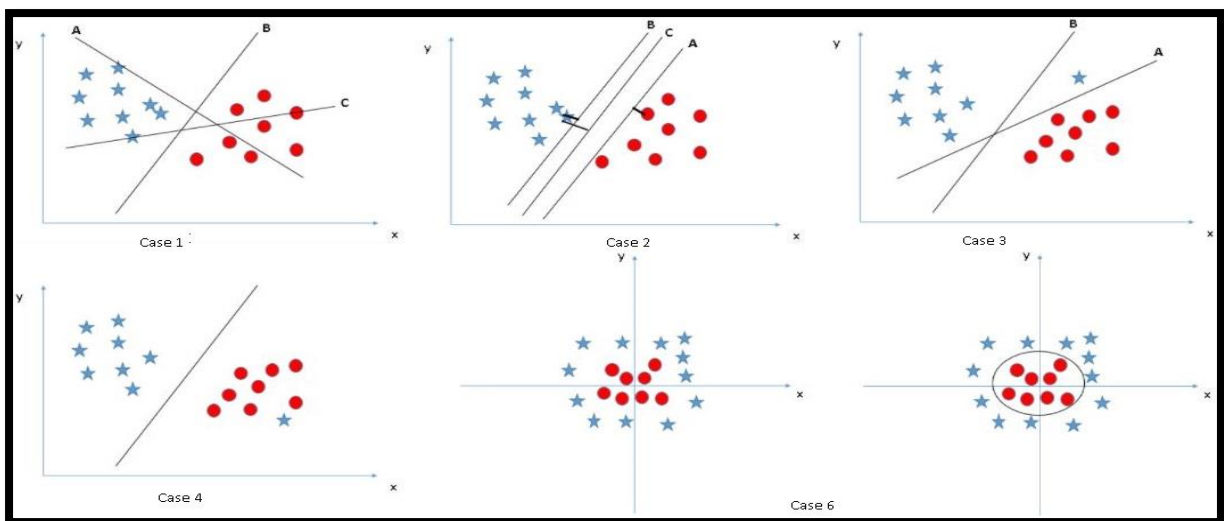


Figure 3.38: Optimizing hyperplanes [95]

In this thesis, the drying hopper temperature values recorded are divided into a number of subsequences, each of which is treated as one example (train or test). Each example has  $60 \times 12 = 720$  features or 60 features at each time stamp for all twelve temperature zones. In order to use SVM, the input must be a two dimensional array. This is why the input array of shape (no. of examples, window length, number of sensors or variables) is reshaped into a shape of (no. of examples, no. of features in one example). Figure 3.37 shows how SVM optimizes hyperplanes in variety of cases.

The general thumb rule is choosing the hyperplane that separates the classes well as in case 1 of Figure 3.38. In case 2, SVM tries to minimize the distance between nearest data points which is known as margin. It gives more priority on separating classes than minimizing margin as shown in case 3. SVM ignores outliers and with a defined feature and tries to optimize the margin (case 4). If the dataset is linearly inseparable, SVM uses non-linear hyperplane by using radial basis function (RBF) kernel or polynomial kernel. Normally in high dimensional space, dataset are more linearly separable [95]. More details on SVM can be found at [96].

### 3.2.5.2 Random Forest

Random forest is an ensemble machine learning algorithm used in variety of cases. The main idea behind random forest is building multiple decision trees and combining them together to provide a better prediction. As mentioned, random forest is an ensemble technique which uses bagging technique where the core idea is the combination of learning models increase the accuracy of the prediction. In section 3.1.3.4.3, ensemble learning through undersampling technique was mentioned which is used in this thesis in order to overcome the issue, imbalanced dataset. The ensemble method was used to improve the classification accuracy of the neural network combining multiple undersampling neural network model and random forest works in similar fashion.

Decision tree is another machine learning algorithm based on which random forest works. Decision tree splits the dataset repetitively using the decision node unless the leaf nodes or terminal nodes are obtained where the best split is found through maximizing the entropy gain. As decision tree is highly sensitive to the training data which might result in high variance with generalizing error. Random forest is developed to overcome this issue through a random collection of trees which is less sensitive to training data.

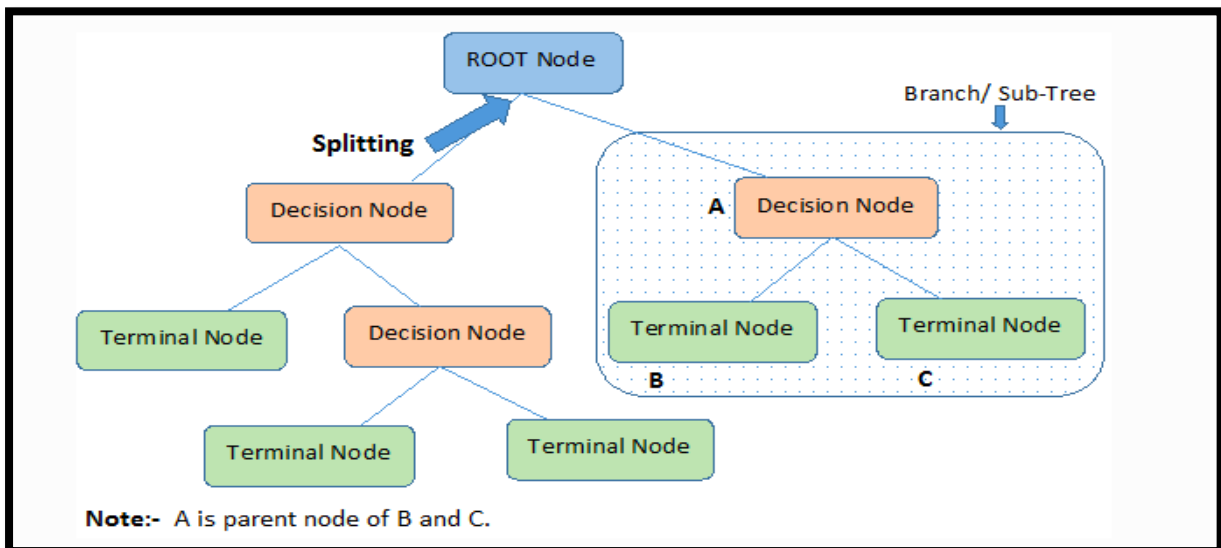


Figure 3.39: Decision Tree[97]

Random forest starts with random sampling with replacement multiple times. For example, our training data has 720 features in each example and there are  $m$  training examples. So out of those  $m$  training examples, it will randomly sample the dataset with replacement, which indicates any row or example can be appeared more than once in the new dataset. This process is known as bootstrapping. Afterwards each bootstrapped dataset will be trained using a decision tree where each dataset will use a subset of features, 720 in this case. For the test set, each example will be passed through each of the newly developed trees and prediction will be made on each tree for the new example. A majority voting system will be used to define the final class of that example. This ensemble technique is known as aggregation. Bootstrapping and aggregation together is known as bagging.

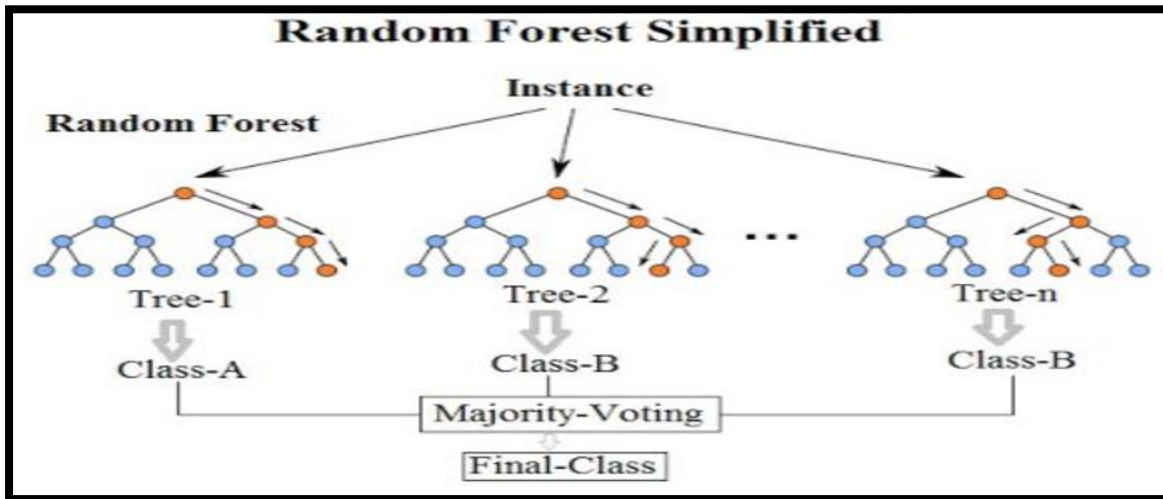


Figure 3.40: Random Forest[98]

The process is called random forest as two operations performed in the algorithm; bootstrapping and random feature selection, both are random process. Bootstrapping ensures that same data are not used in every tree, thus it prevents the sensitivity to original training data problem of decision tree.

Random feature selection helps to reduce the correlation between trees as when all the features are used in bootstrapped data, the trees developed from those data will be similar in nature with same decision nodes. This might increase the variance and will reduce the efficacy of the random forest. The number of features used in each bootstrapped data is normally the log or square root of the total number of features. More details on the theory of the random forest can be found in [99].

### 3.2.6 K-Nearest neighbor (KNN)

K nearest neighbor is another nonlinear machine learning algorithm and is probably the most intuitive algorithm in machine learning. The main intuition behind K- nearest neighbor is the examples which have same labels or similar features should be close enough from each other in an n dimensional space. K nearest neighbor uses a distance approach to calculate the distance between two time series or two MTS. The most commonly used distance method is Euclidean distance method, but it has a disadvantage. When two time series have unequal length Euclidean distance method cannot be used. This is why another distance approach which is highly popular in time series application, dynamic time warping (DTW) is often incorporated with K nearest neighbor to classify time series. Figure 3.41 shows the basic of K nearest neighbor algorithm. The main steps of using a KNN are:

- Calculating the distance between each pair of training examples using distance measures like Euclidean (equal length time series) or DTW (unequal length time series)
- Selecting k nearest examples based on the distance calculated in the previous step.
- Assigning the most common label among k nearest examples to the new example.



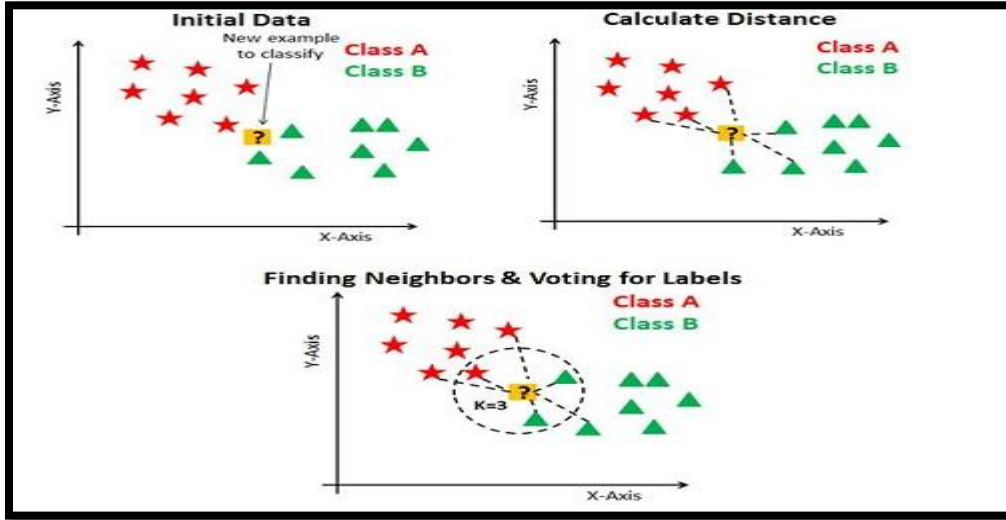


Figure 3.41: KNN [100]

For this thesis, KNN with Euclidean distance approach will be used as the extracted subsequences are of same length. More details about KNN with distance approach can be found in [32], [101].

### 3.2.7 Performance Measure

In this thesis, accuracy, precision, recall and f1 scores will be used as performance measure. In order to calculate these measures, four terms need to be introduced which are true positive (TP), true negative (TN), false positive (FP) and false negative (FN). For the rest of this thesis, non-events will be identified as positive class whereas events will be identified as negative class. These performance measures are described below.

#### Accuracy:

Accuracy is defined as the ratio between correctly classified examples and total number of examples. It can be misleading especially in case of imbalance dataset. It can be used if percentage of examples belonged to the majority class is known before, so that the lower bound of accuracy can be determined.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (22)$$

#### Precision:

Although accuracy is a performance measure for the overall datasets, precision is a performance measure for individual class. Precision is a performance measure which is related to the prediction. It can be defined as the ratio between correctly predicted positive class example and all predicted positive class examples or the definition can be provided in terms of negative class as well. It is a well-known measure to identify percentage of examples from a class which are correctly identified in terms of predicted labels.

$$Precision = \frac{TP}{TP + FP} \quad (23)$$

**Recall:**

Another highly useful performance measure is recall also known as sensitivity which is related to the truth instead of prediction. It can be identified as the ratio between the examples which are actually positive and the examples which are predicted as negative, but actually positive. This definition can be extended to the negative class perspective as well. It is also a well-known measure to identify the percentage of examples from a class which are correctly identified in terms of the actual labels.

$$Recall = \frac{TP}{TP + FN} \quad (24)$$

**F1 score:**

Probably the best performance measure is the f1 score which considers the data imbalance issue. It is a more structured performance measure using precision and recall.

$$F_1 = \frac{2 * precision * recall}{precision + recall} \quad (25)$$

Another useful graphical technique to visualize the model's performance is confusion matrix. A confusion matrix is a matrix with number of dimension equals to the number of classes. A typical confusion matrix for two classes is shown in Figure 3.42.

	P	TP	FN
Truth	N	FP	TN
		P	N
		Predicted	

Figure 3.42: Confusion Matrix

### 3.2.8 System specification

In this thesis, python programming language will be used for data preprocessing, model development, experimental runs and evaluation. A variety of python frameworks like pandas and numpy for data preprocessing and neural network framework like tensorflow and keras will be used for model development and experimentation. The experiment will be run on windows 10, Intel® core™ i5-3337U CPU @ 1.8 GHz. The following table shows the python libraries used for the experimentation.

Table 3.4: Python libraries

<b>Name</b>	<b>Purpose of use</b>
Pandas	Data preprocessing
Numpy	Data preprocessing
Scikit learn	ML model build up and data preprocessing like min-max scaling
imblearn	Data balancing
Matplotlib	Plotting graph
keras	Deep learning model build up
Tensorflow	Backend for keras
Seaborn	Confusion matrix

In the next chapter, the detail of the experimentation on the models described in this chapter, result and evaluation will be highlighted. Afterwards, the discussion regarding the result and limitation of the thesis will be presented.

## 4 Experimental Results and Discussion

The first goal of the previous chapter was to highlight the characteristics of the dataset and various preprocessing steps with data labeling and imbalanced data issue. Furthermore, the next goal was to highlight the solution approaches that will be implemented on this specific drying hopper case study. In this chapter, a detail of the experimental setup and results will be showed. Furthermore, a comparative analysis will be performed on the deep learning algorithms and traditional techniques dedicated to time series.

### 4.1 Experimental setup

One of the common issues regarding the dataset was imbalanced dataset as discussed in the previous chapter. In order to remedy this issue, four techniques were presented which are random undersampling, random oversampling, SMOTE, and ensemble learning using random undersampling. In this section, experimental setup for SMOTE and ensemble learning will be highlighted. Undersampling technique will not be explicitly showed as a separate experiment as it will be used extensively in ensemble learning. Moreover, data balancing techniques will be used only for deep learning algorithms as in predictive modeling, most of the deep learning algorithms are designed with the assumption of equal number of samples in each class. For other traditional approaches, the regular preprocessed dataset will be used.

As discussed in the previous chapter, in order to use ensemble learning with majority voting, the training dataset will be divided into certain segments. In Table 4.1, it can be seen that only 22.39% examples belong to class 1 in the training set which makes the dataset highly imbalanced. When no special consideration was taken for the data imbalance issue, with any one of the deep learning models; CNN, LSTM, MLP, the final test accuracy was found as 94.34%.

Table 4.1: Training and Test Examples

Examples	Test Examples		Training Examples	
	Events (class1)	Non-events (class 0)	Events (class 1)	Non-events (class 0)
# examples	50	833	791	2742
percentage	5.66%	94.34%	22.39%	77.61%

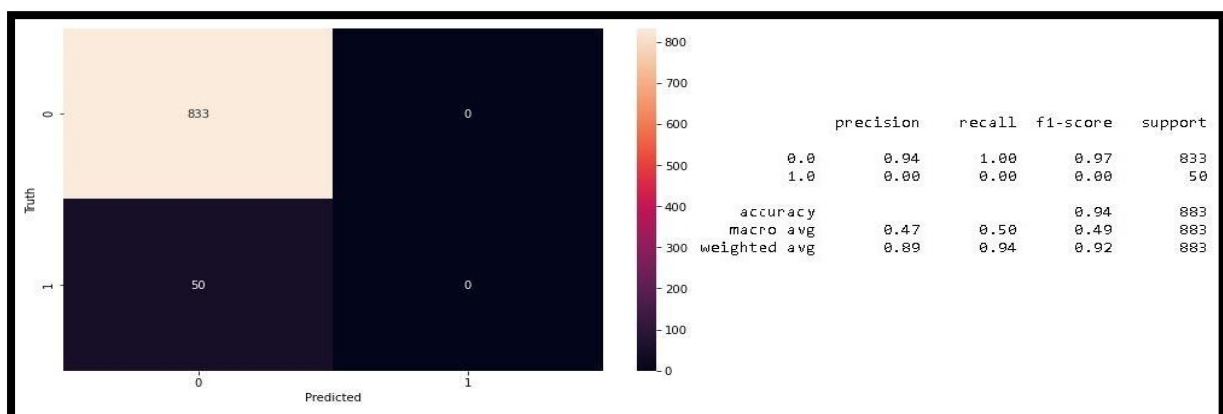


Figure 4.1: Confusion matrix and classification report of imbalanced dataset (CNN)

## Experimental Results and Discussion

Figure 4.1 shows a very high accuracy of the model and can fool anyone who is not aware of the distribution of the two classes in the test set. As shown in Table 4.1, 93.43% test examples belong to class 0. This is why the deep learning model showed an accuracy of 93.43% as it cannot identify any events of class 1. All the events of class 1 were identified as 0 which resulted in the test accuracy of 93.43%. The performance of this model using CNN can be more clearly understood from confusion matrix and classification report with precision, recall and f1 score as shown in Figure 4.1.

As a remedy to this issue, ensemble learning, oversampling and SMOTE have been used. For ensemble learning, three approaches have been taken to evaluate the effectiveness in terms of precision and recall.

**Approach 1:** 2,742 training examples of class 0 are divided into five groups. As 2,742 is not divisible by 5 ( $2,742/5 = 548.4$ ), the first three groups have 548 examples each, other two groups have 549 examples each, combining together five groups have total  $548*3+549*2 = 2742$  examples of class 0. Afterwards, 548 or 549 examples from class 1 were chosen randomly. These 548 examples from class 1 and 548 examples from class 0 are combined and shuffled to obtain one group of dataset for training. In this way, five group of training sets have been generated and each of them is trained using a separate model and majority voting is used at the end in the following way.

$$n_1 = \sum_{i=1}^5 \hat{y}_i \quad (26)$$

$$\hat{y} = \begin{cases} 1, n_1 > 2 \\ 0, otherwise \end{cases} \quad (27)$$

Here,  $n_1$  is the sum of the outputs of any particular test example from five undersampling models which can be maximum of five if all models determined that example as 1 and minimum of 0 if all models determined that example as 0.  $n_1$  can also be defined as number of models which determined the class of that particular example as 1.

**Approach 2:** 2,742 training examples of class 0 are divided into three groups. First two groups have 791 examples each and the last group has  $2,742-791*2 = 1,160$  examples. Afterwards, 791 examples of class 1 will be combined with each group to build three group of datasets. Each group will be shuffled properly before training. In this way, three group of training sets have been generated and each of them is trained under a separate model. Majority voting is used in the following way.

$$n_1 = \sum_{i=1}^3 \hat{y}_i \quad (28)$$

$$\hat{y} = \begin{cases} 1, n_1 > 1 \\ 0, otherwise \end{cases} \quad (29)$$

**Approach 3:** This approach is similar to the previous one. The only difference is 2,742 examples are divided into three equal segments where each of the group has 914 examples. Afterwards, 791 examples are combined and shuffled with each of the three groups. In this

## Experimental Results and Discussion

way, each training dataset has 914 (class 0) +791 (class 1) = 1,705 training examples. The majority voting is used similarly as shown in approach 2.

### 4.2 Hyperparameter Tuning

In order to use deep learning algorithms, hyperparameter tuning is a very important step. A lot of hyperparameters exist in a deep learning algorithm from which the ideal combination needs to be selected for optimal performance. Table 4.2 shows a list of parameters and hyperparameters associated with CNN, LSTM and MLP.

Table 4.2: List of hyperparameters

	Neural Network / Fully connected layer of CNN	CNN and Long Short Term Memory
<b>Parameters</b>	Weights, $W^{[L]}$ , biases, $b^{[L]}$	Filters, $f^{[L]}$
<b>Hyperparameters</b>	<ul style="list-style-type: none"> <li>• # hidden layers, L</li> <li>• # hidden units in each hidden layer, <math>K^{[L]}</math></li> <li>• Choice of activation function (Sigmoid, Tanh, ReLU, Leaky ReLU)</li> <li>• Optimizer (SGD, Adam, RMSprop)</li> <li>• Batch size</li> <li>• # epochs</li> </ul>	<ul style="list-style-type: none"> <li>• Filter size, f</li> <li>• # filters</li> <li>• stride, s</li> <li>• Choice of pooling strategy (max or average)</li> <li>• Pooling or subsampling size</li> <li>• # LSTM units</li> </ul>

As mentioned in the previous chapter, Keras deep learning framework in python is used for the experimentation which runs on top of tensorflow. Apart from the hyperparameters shown in Table 4.2, there are other hyperparameters as well. One of the most important hyperparameters in any deep learning algorithm is the learning rate,  $\alpha$ . The default value of  $\alpha$  in keras is 0.01 with no momentum. The default value of  $\alpha$  was used for the initial test run. First group of samples mentioned in approach 2 was taken for the hyperparameter tuning of CNN and LSTM. The following tables show the summary of the hyperparameters for the initial experiment on hyperparameter tuning.

Table 4.3: values considered for hyperparameters

Hyperparameter	Values considered
# filters	8, 16
Filter size	3, 5, 7
Batch size	32, 64, 128

Table 4.4: Initial values for hyperparameters

Hyperparameter	Initial value
Learning rate	0.01
Activation function in convolutional layers and dense layers	ReLu
Activation function in the output layer	sigmoid
Probability rate in dropout layer	0.5
Subsampling size (pooling size)	2

## Experimental Results and Discussion

# hidden layer in fully connected layer (FCL)	1
# neurons in FCL	200
# epochs	100

Different values for number of filters, filter size and batch size have been used as shown in Table 4.3 and other hyperparameters were kept constant as shown in Table 4.4. 10 experimental runs were performed to figure out the best set of hyperparameters. As neural network is highly stochastic in nature due to random weight initialization, each run obtains different test accuracy. This is why test accuracy average and variability has taken into consideration for choosing a hyperparameter. Figure 4.2 shows the box plots for number of filters, filter size and batch size in terms of test accuracy.

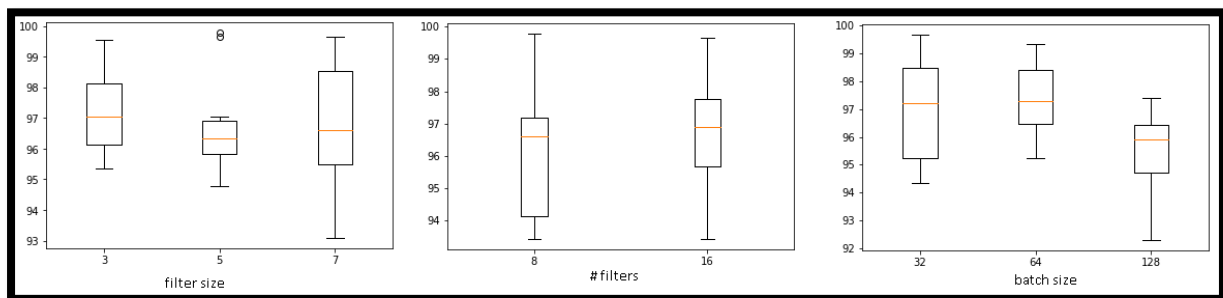


Figure 4.2: Hyperparameter tuning

It can be seen from Figure 4.2, average test accuracy for all three filter sizes show almost similar average test accuracy, but filter size 5 shows less variability in terms of accuracy. Number of filters is chosen as 16 as it has the higher average accuracy and batch size is chosen as 64 for higher average and less variability.

Now a final test run will be performed using the finalized value of filter size, number of filters and batch size as well as other hyperparameters as shown in Table 4.4. In this regard, a validation split of 10% will be used which indicates 10% of the training data will not be used in training, they will be used to validate the model. The following figure shows the graph of accuracy and loss with respect to number of epochs.

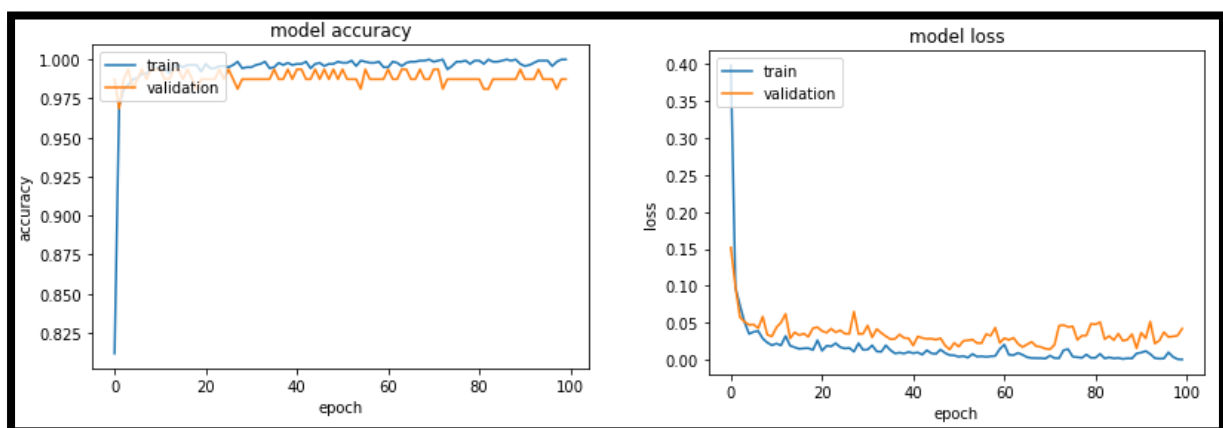


Figure 4.3: accuracy vs. epoch and loss vs. epoch

## Experimental Results and Discussion

From Figure 4.3, it is evident that the model is converging to the optimal solution. So these hyperparameters will be used in the final experiment except no. of epochs. Although the accuracy is good, but it is also evident that the model suffers from overfitting which can be seen in Figure 4.3 that the learning curve is fluctuating continuously. This indicates the model has learned too much and is suffering from generalization error [102]. From the figure, it can be seen that within 10 epochs the model reaches almost zero loss and very high accuracy. This is why number of epochs will be used as 10 in the final experiment.

### 4.3 Result

After performing the hyperparameter search, final experimental runs were performed. The following section provides a summary of the results for all of the considered algorithms and corresponding data balancing techniques.

#### 4.3.1 Ensemble Learning (CNN)

In the final experiment, 10 experimental runs were performed for each of the deep learning and machine learning algorithms. For this specific drying hopper case, the main goal is to capture the events automatically so that a predictive maintenance approach can be taken as a remedy. As the number of non-events are very high, it is natural that if the model can determine the non-events correctly, accuracy will be automatically higher regardless of whether the model is able to identify the events or not as they are very low in numbers. This is why instead of accuracy, precision of class 0 and recall of class 1 are more important in this case as these two depend on how many of events are wrongly identified also known as false positive. For this case study, it is desirable to have precision of class 0 and recall of class 1 as high as possible or number of false positive as low as possible. Figure 4.4 shows the CNN framework used in this thesis.

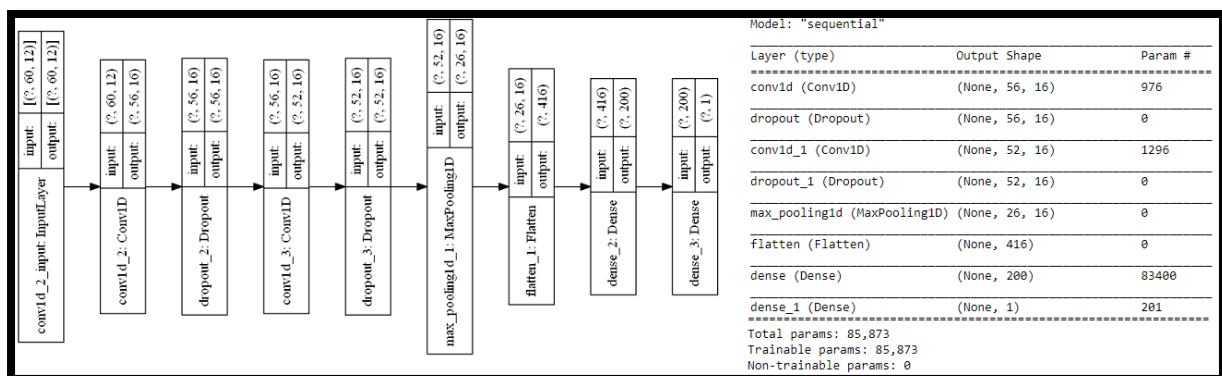


Figure 4.4: CNN framework

Summary of the ten experimental runs using CNN is shown in Figure 4.5 and Figure 4.6. In terms of average accuracy, approach 3 clearly shows best result with 99.30% average accuracy. False positive (events identified as non-events) values vary between 1 and 2 whereas false negative values (non-events identified as events) vary between 0 and 7 in ten experimental runs with two outliers (11 and 19). First experimental run obtained the best result for approach 3 as only 1 example was misclassified as non-event which was actually an event. The following figures show the best result achieved from these three approaches in ten experimental runs.



## Experimental Results and Discussion

Ensemble Learning	Metrics	Experimental Runs (Convolutional neural Network)										Average accuracy
		1	2	3	4	5	6	7	8	9	10	
Approach 1	TP	832	811	832	812	826	817	829	829	811	833	0.9866
	TN	49	48	46	48	49	49	47	48	48	48	
	FP	1	2	4	2	1	1	3	2	2	2	
	FN	1	22	1	21	7	16	4	4	22	0	
	Accuracy	0.9977	0.9728	0.9943	0.9740	0.9909	0.9807	0.9921	0.9932	0.9728	0.9977	
Approach 2	TP	820	833	830	827	833	820	812	831	829	820	0.9900
	TN	49	48	48	49	48	49	49	49	49	49	
	FP	1	2	2	1	2	1	1	1	1	1	
	FN	13	0	3	6	0	13	21	2	4	13	
	Accuracy	0.9841	0.9977	0.9943	0.9921	0.9977	0.9841	0.9751	0.9966	0.9943	0.9841	
Approach 3	TP	833	826	831	833	822	829	832	828	832	814	0.9930
	TN	49	49	49	48	49	49	48	49	49	49	
	FP	1	1	1	2	1	1	2	1	1	1	
	FN	0	7	2	0	11	4	1	5	1	19	
	Accuracy	0.9989	0.9909	0.9966	0.9977	0.9864	0.9943	0.9966	0.9932	0.9977	0.9773	

Figure 4.5: Result summary of CNN (Ensemble Learning)

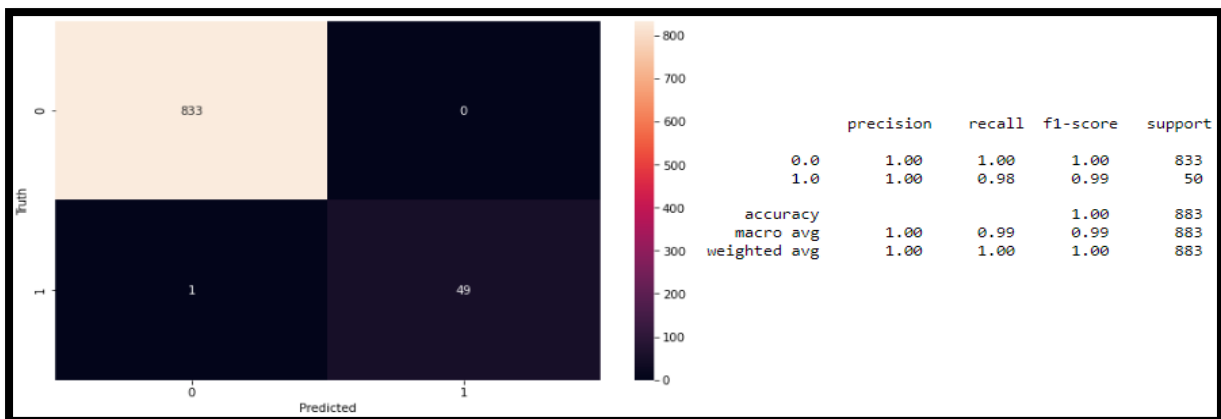


Figure 4.6: Confusion matrix and classification report of approach 3, run 1

As mentioned in the experimental setup, undersampling approaches were combined for ensemble learning and majority voting is used for predicting the class of test examples. Each undersample of all three approaches provided different predictions for the test examples and these are combined later which improved the overall accuracy. The best undersamples of the ensemble learning in each run are shown in Figure 4.7.

Metrics	Experimental Runs (Convolutional neural Network)									
	1	2	3	4	5	6	7	8	9	10
TP	833	833	833	831	833	833	833	831	833	833
TN	49	49	47	49	48	49	49	49	49	48
FP	1	1	3	1	2	1	1	1	1	2
FN	0	0	0	2	0	0	0	2	0	0
accuracy	0.9989	0.9989	0.9966	0.9966	0.9977	0.9989	0.9989	0.9966	0.9989	0.9977
Approach/segment	Approach 3, segment 2	Approach 3, segment 2	Approach 1, segment 4	Approach 2, segment 3	Approach 2, segment 2	Approach 3, segment 2	Approach 3, segment 2	Approach 2, segment 2	Approach 1, segment 2	Approach 1, segment 2

Figure 4.7: best undersamples in each experimental run

## Experimental Results and Discussion

Segment 2 of approach 3 where 914 training examples of class 0 (915 to 1828) and 791 training examples of class 1 were combined, obtained best result in four experimental runs. This evidence clearly indicates these 914 training examples of class 0 provide significant information for training a CNN model. If undersampling is used without any ensemble learning, there is an uncertainty about the significance of that specific segment of the data used for training. Figure 4.8 shows the best learning curves obtained from the best undersampling strategies.

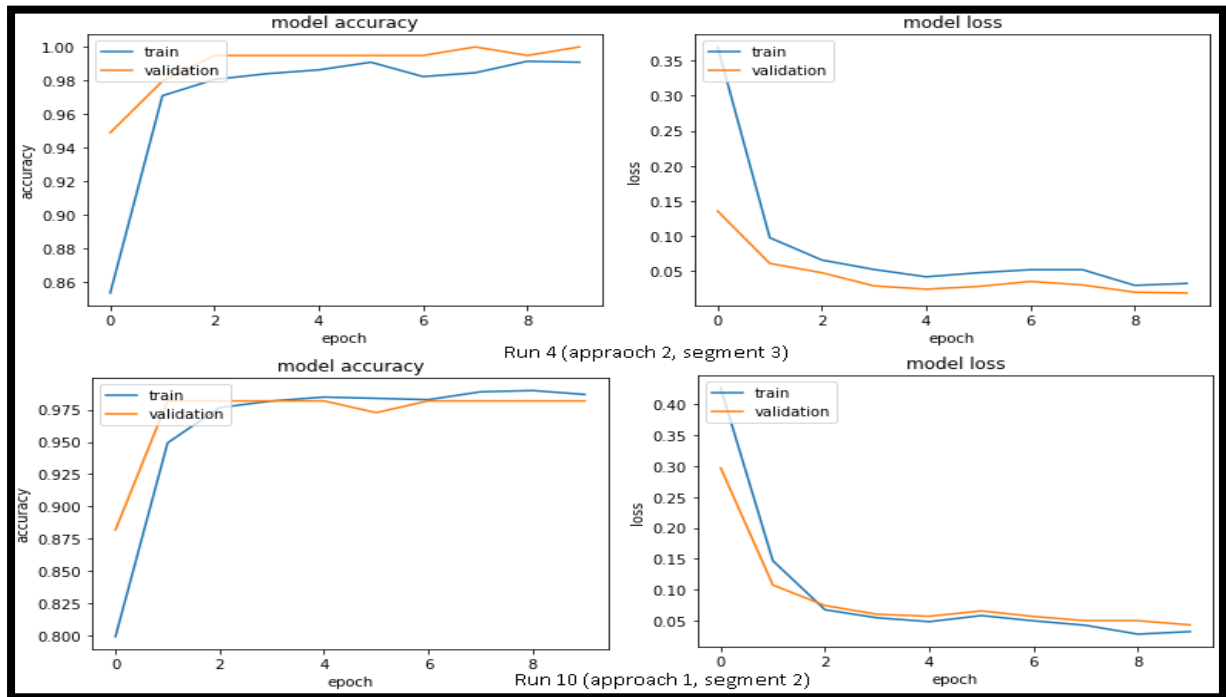


Figure 4.8: best learning curves (CNN)

So, using CNN and ensemble learning no. of false positives can be reduced to 1 and false negatives to 0. The best accuracy obtained using CNN and ensemble learning was 99.30% (approach 3) where one example was wrongly classified in maximum runs.

### 4.3.2 Ensemble Method (LSTM)

For LSTM, hyperparameters setup is exactly the same as in CNN in the common portion like in the fully connected layer. The experiment starts with 200 LSTM units in the LSTM layer after the input layer and obtained significant result. Figure 4.9 shows the LSTM framework used in this thesis. Figure 4.10 provides a summary of the ten experimental runs using Long Short Term Memory network.

In case of LSTM, approach 3 showed the best result as shown in Figure 4.10. It has one outlier in run 3 when it results is 24 false positives which is a highly extraneous case. The result is quite similar as compared to CNN and the three approach produced almost similar result with small variation and approach 3 provides the best average accuracy again. The following figures show the best results obtained in ten experimental runs for LSTM.

# Experimental Results and Discussion

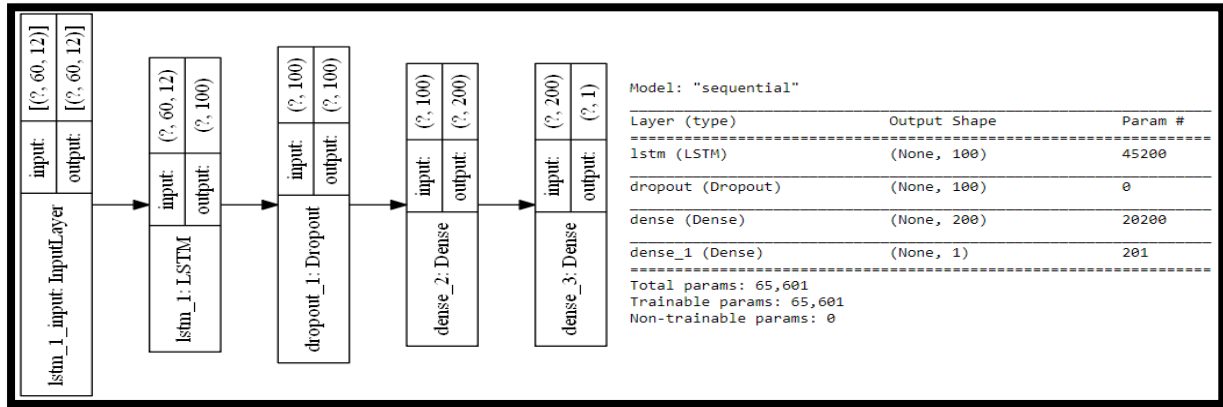


Figure 4.9: LSTM framework and summary

Ensemble Learning	Metrics	Experimental Runs (Long Short Term Memory)										Average accuracy
		1	2	3	4	5	6	7	8	9	10	
Approach 1	TP	811	810	830	811	832	830	833	833	832	832	0.9874
	TN	48	47	47	48	43	46	48	47	43	48	
	FP	2	3	3	2	7	4	2	3	7	2	
	FN	22	23	3	22	1	3	0	0	1	1	
	Accuracy	0.9728	0.9706	0.9932	0.9728	0.9909	0.9921	0.9977	0.9966	0.9909	0.9966	
Approach 2	TP	833	832	802	830	832	833	828	814	806	831	0.9855
	TN	47	48	49	47	48	36	42	47	49	48	
	FP	3	2	1	3	2	14	8	3	1	2	
	FN	0	1	31	3	1	0	5	19	27	2	
	Accuracy	0.9966	0.9966	0.9638	0.9932	0.9966	0.9841	0.9853	0.9751	0.9683	0.9955	
Approach 3	TP	832	833	833	831	814	831	832	831	830	831	0.9905
	TN	44	45	26	48	48	48	46	48	47	48	
	FP	6	5	24	2	2	2	4	2	3	2	
	FN	1	0	0	2	19	2	1	2	3	2	
	Accuracy	0.9921	0.9943	0.9728	0.9955	0.9762	0.9955	0.9943	0.9955	0.9932	0.9955	

Figure 4.10: Result summary of LSTM

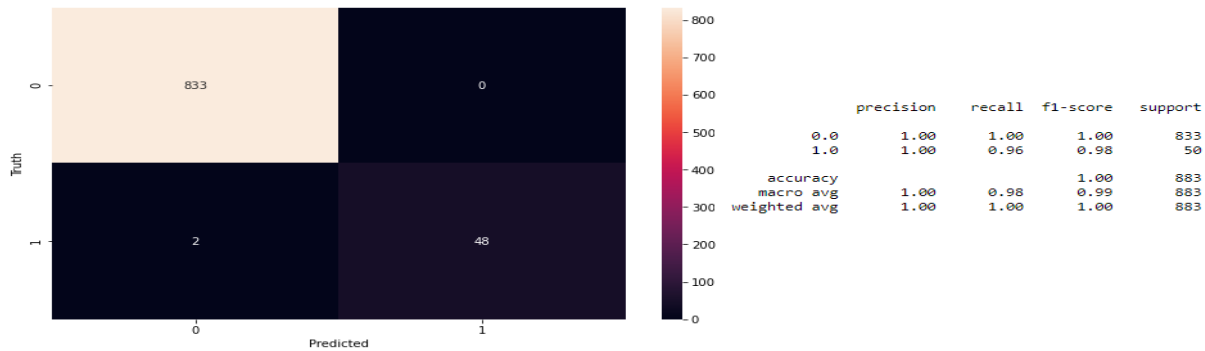


Figure 4.11: Confusion matrix and classification report of approach 1, run 7

The best undersamples of the ensemble learning in each run are shown in Figure 4.12.

Metrics	Experimental Runs (Long short term memory)									
	1	2	3	4	5	6	7	8	9	10
TP	833	831	832	832	833	827	833	833	828	831
TN	47	49	47	48	47	48	48	48	49	49
FP	3	1	3	2	3	2	2	2	1	1
FN	0	2	1	1	0	6	0	0	5	2
accuracy	0.9966	0.9966	0.9955	0.9966	0.9966	0.9909	0.9977	0.9977	0.9932	0.9966
Approach/segment	Approach 2, segment 3	Approach 2, segment 3	Approach 1, segment 3	Approach 3, segment 2	Approach 2, segment 2	Approach 1, segment 3	Approach 1, segment 2	Approach 1, segment 2 and 3	Approach 2, segment 3	Approach 2, segment 3

Figure 4.12: Best undersamples in each experimental run (LSTM)

## Experimental Results and Discussion

Segment 3 of approach 2 where 1160 training examples of class 0 (1583 to 2742) and 791 training examples of class 1 were combined, obtained best result in four experimental runs. This evidence clearly indicates these 1160 training examples of class 0 provide significant information for training a LSTM model. Figure 4.13 shows the best learning curves obtained from the best undersampling strategies of ensemble learning with LSTM.

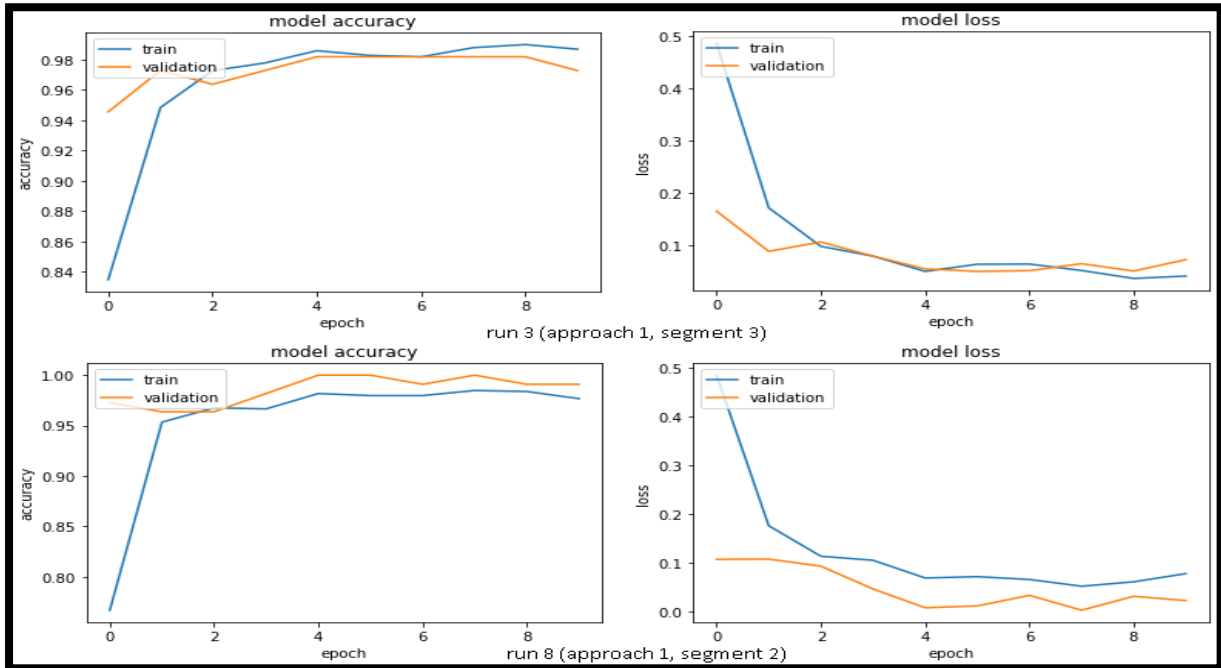


Figure 4.13: Best learning curves (LSTM)

So, using CNN and ensemble learning, no. of false positives can be reduced to 2 and false negatives to 0. The best accuracy obtained using CNN and ensemble learning was 99.05% (approach 3) where two examples were wrongly classified in maximum runs.

### 4.3.3 Ensemble Learning (CNN-LSTM)

Same hyperparameters setup as in CNN and LSTM is used for CNN-LSTM model. Figure 4.14 shows the CNN-LSTM framework.

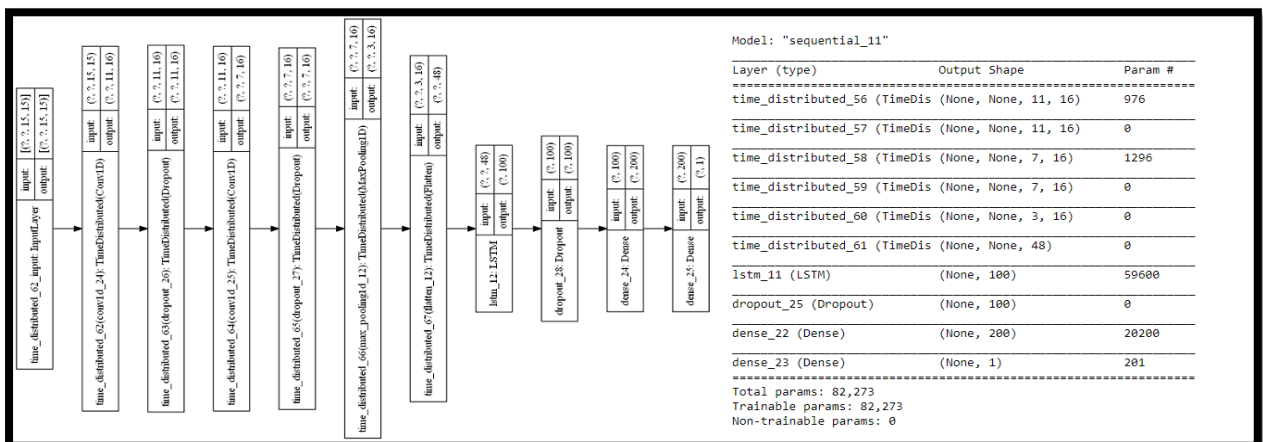


Figure 4.14: CNN-LSTM framework and summary

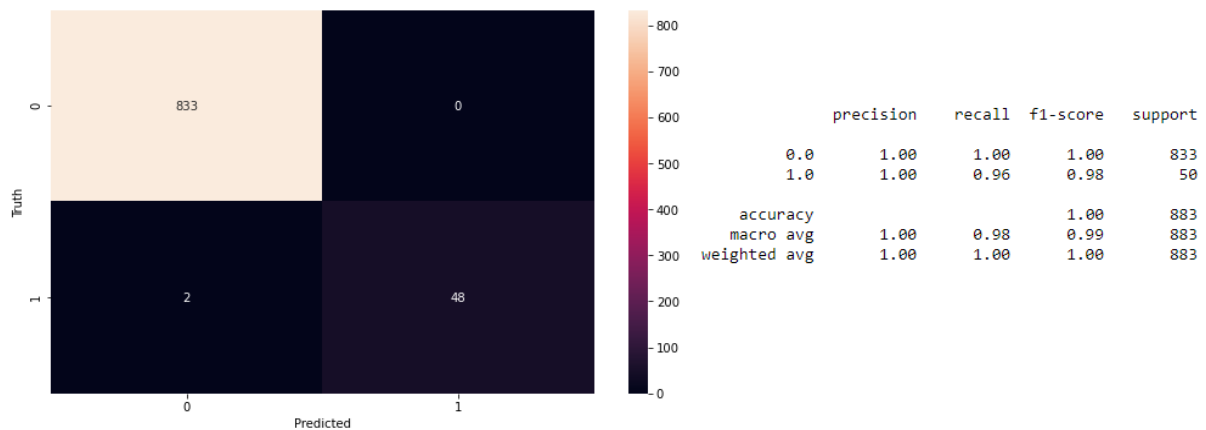
## Experimental Results and Discussion

Figure 4.15 provides a summary of the ten experimental runs using CNN-LSTM.

Ensemble Learning	Metrics	Experimental Runs (CNN-LSTM)										Average accuracy
		1	2	3	4	5	6	7	8	9	10	
Approach 1	TP	832	827	832	833	831	827	832	806	811	831	0.9875
	TN	48	48	47	48	40	45	48	46	39	49	
	FP	2	2	3	2	10	5	2	4	11	1	
	FN	1	6	1	0	2	6	1	27	22	2	
	Accuracy	0.9966	0.9909	0.9955	0.9977	0.9864	0.9875	0.9966	0.9649	0.9626	0.9966	
Approach 2	TP	833	827	832	808	804	807	830	831	802	828	0.9826
	TN	48	47	47	48	48	47	47	46	48	48	
	FP	2	3	3	2	2	3	3	4	2	2	
	FN	0	6	1	25	29	26	3	2	31	5	
	Accuracy	0.9977	0.9898	0.9955	0.9694	0.9649	0.9672	0.9932	0.9932	0.9626	0.9921	
Approach 3	TP	833	825	805	810	820	812	825	816	809	827	0.9807
	TN	47	48	48	48	48	48	47	48	48	48	
	FP	3	2	2	2	2	2	3	2	2	2	
	FN	0	8	28	23	13	21	8	17	24	6	
	Accuracy	0.9966	0.9887	0.9660	0.9717	0.9830	0.9740	0.9875	0.9785	0.9706	0.9909	

*Figure 4.15: Result summary of CNN-LSTM*

Although, approach 3 works best for CNN and LSTM model, for CNN-LSTM approach 1 works well with 98.75% average accuracy across ten runs. Figure 4.16 shows the best results obtained in ten experimental runs for LSTM.



*Figure 4.16: Confusion matrix and classification report of appr. 1, run 4 and appr. 2, run 1*

The best undersamples of the ensemble learning in each run are shown in Figure 4.17. Approach 3, segment 2 appears twice in the list along with approach 1, segment 5 and approach 2, segment 2 and approach 2, segment 3.

Metrics	Experimental Runs (CNN LSTM)									
	1	2	3	4	5	6	7	8	9	10
TP	833	832	832	833	833	832	832	832	833	831
TN	49	48	48	48	48	47	48	46	46	49
FP	1	2	2	2	2	3	2	4	4	1
FN	0	1	1	0	0	1	1	1	0	2
accuracy	0.9989	0.9966	0.9966	0.9977	0.9977	0.9955	0.9966	0.9943	0.9955	0.9966
Approach	Approach 2,	Approach 1,	Approach 1,	Approach 1,	Approach 3,	Approach 2,	Approach 3,	Approach 2,	Approach 2,	Approach 1,
segment	segment 4	segment 1	segment 5	segment 2	segment 2	segment 3	segment 2	segment 2	segment 3	segment 5

*Figure 4.17: Best undersamples in each experimental run (CNN- LSTM)*

## Experimental Results and Discussion

Figure 4.18 shows the best learning curves obtained from the best undersampling strategies of ensemble learning with CNN-LSTM.

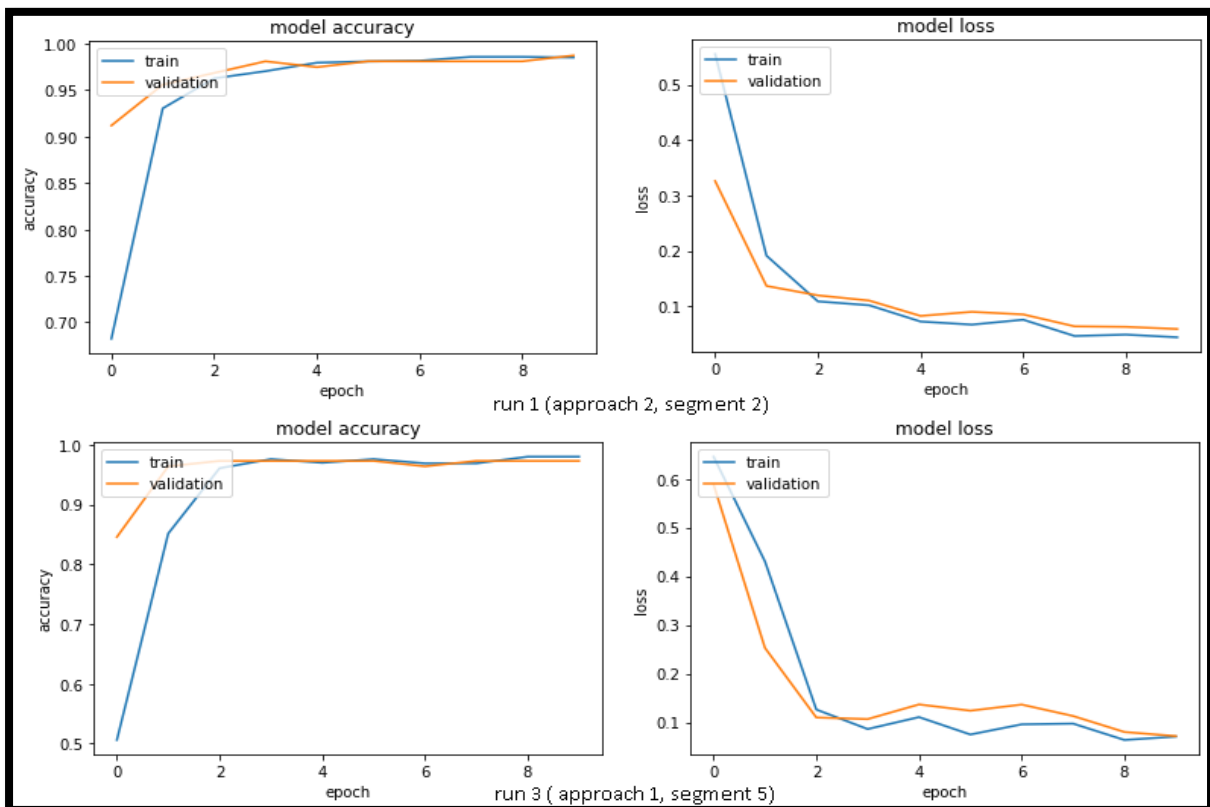


Figure 4.18: Best learning curves (CNN-LSTM)

The result is similar with LSTM as the combination of CNN and LSTM can reduce the number of false positive to as low as 2 and false negatives to 0 as well. Although approach 1 works best in terms of average accuracy, it has some outliers across ten runs which is not the case for approach 2 and 3.

### 4.3.4 SMOTE

In the training set, 2,742 examples belong to class 0 and 791 examples belong to class 1. Using SMOTE, 1,951 more samples were generated from the minority class, so that class 1 has 2,742 examples as well. These  $2,742+2,742 = 5,484$  examples were combined and shuffled properly for training the dataset using CNN, LSTM and CNN-LSTM. The following figure shows the summary of the ten experimental runs.

CNN again performs best when using SMOTE for data augmentation with 99.42% average accuracy with no outlier in the false positive values across ten runs. The best result obtained using SMOTE is shown in Figure 4.20.

# Experimental Results and Discussion

Ensemble Learning	Metrics	Experimental Runs (SMOTE)										Average accuracy
		1	2	3	4	5	6	7	8	9	10	
CNN	TP	831	832	833	827	833	832	832	829	832	815	0.9942
	TN	49	47	49	49	48	47	48	48	48	50	
	FP	1	3	1	1	2	3	2	2	2	0	
	FN	2	1	0	6	0	1	1	4	1	18	
	Accuracy	0.9966	0.9955	0.9989	0.9921	0.9977	0.9955	0.9966	0.9932	0.9966	0.9796	
LSTM	TP	833	822	793	779	832	830	833	821	828	831	0.9830
	TN	46	49	49	50	49	48	42	49	48	48	
	FP	4	1	1	0	1	2	8	1	2	2	
	FN	0	11	40	54	1	3	0	12	5	2	
	Accuracy	0.9955	0.9864	0.9536	0.9388	0.9977	0.9943	0.9909	0.9853	0.9921	0.9955	
CNN-LSTM	TP	832	828	828	832	813	833	812	825	832	831	0.9900
	TN	49	48	48	49	49	41	49	48	48	47	
	FP	1	2	2	1	1	9	1	2	2	3	
	FN	1	5	5	1	20	0	21	8	1	2	
	Accuracy	0.9977	0.9921	0.9921	0.9977	0.9762	0.9898	0.9751	0.9887	0.9966	0.9943	

Figure 4.19: Result summary of SMOTE

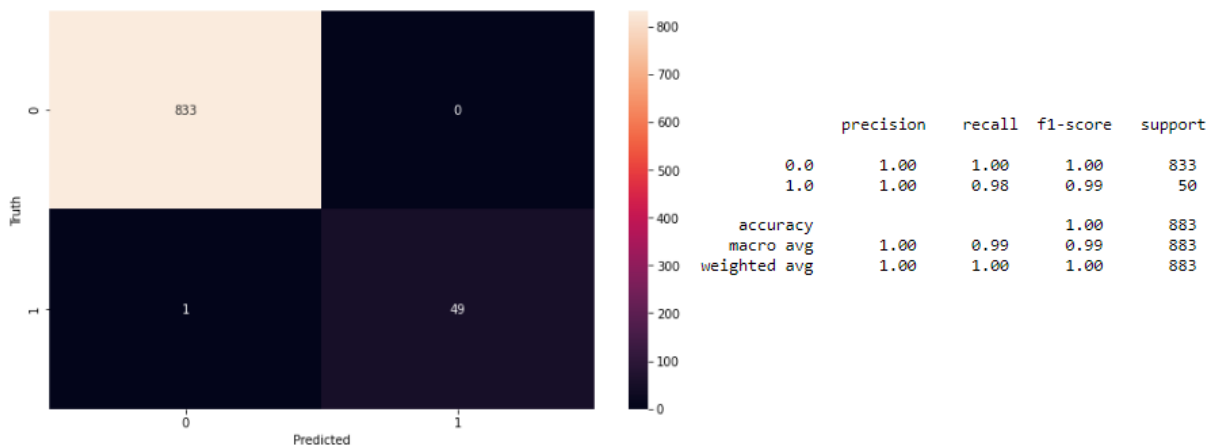


Figure 4.20: Confusion Matrix and Classification report (CNN, run 3)

So using SMOTE, CNN performs best where number of false positive can be reduced as low as 1 and false negative as low as zero. CNN also shows less variability in terms of false positive and false negative with only one outlier in false negative values. The best learning curves are shown in Figure 4.21.

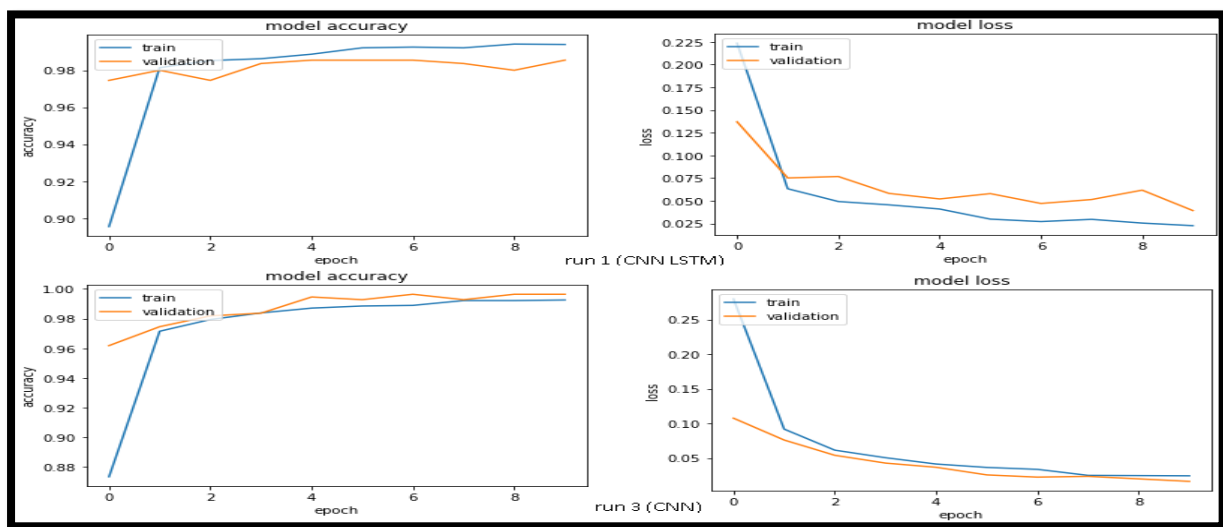


Figure 4.21: Learning curves (SMOTE)

## Experimental Results and Discussion

### 4.3.5 Machine learning algorithms

In order to compare the performance of deep learning with other existing approaches, some machine learning algorithms have been used like k-nearest neighbor, support vector machine, naïve Bayes, random forest and gradient boosting. The following figures summarize the results obtained from machine learning model using the actual dataset and SMOTE dataset.

	Metrics	ML algorithms						Best
		K-nearest neighbor	Support Vector Machine	Naïve bayes	Decision Trees	Random Forest	Gradient Boosting	
Actual Dataset (Best result in 10 runs)	TP	833	822	817	813	820	822	SVM and GB
	TN	38	49	48	50	49	49	
	FP	12	1	2	0	1	1	
	FN	0	11	16	20	13	11	
	Accuracy	0.9864	0.9864	0.9796	0.9773	0.9841	0.9864	
Experimental run	3	1	4	9	2	3		
Average accuracy (10 runs)		0.9742	0.9789	0.9692	0.9621	0.9735	0.9684	SVM
SMOTE(Best Result in 10 runs)	TP	833	818	817	815	815	820	GB
	TN	44	50	48	49	49	49	
	FP	6	0	2	1	1	1	
	FN	0	15	16	18	18	13	
	Accuracy	0.9932	0.9830	0.9796	0.9785	0.9785	0.9841	
	Run	6	2	5	1	8	6	
Average accuracy (10 runs)		0.9813	0.9766	0.9703	0.9643	0.9601	0.9732	KNN

Figure 4.22: Result summary of ML algorithms

### 4.3.6 Summary

The final results from all algorithms using different approaches showed in the previous section can be summarized from two perspectives. One of them is the best result (accuracy, FP and FN) obtained for an algorithm either using ensemble learning, SMOTE or the actual dataset in ten experimental runs. The second one is the average result obtained (accuracy) for an algorithm in ten experimental runs. Table 4.5 and Table 4.6 summarize the result using these two measures.

Table 4.5: Result summary (average result in ten runs)

Method	Ensemble Learning								
	Approach 1			Approach 2			Approach 3		
Algorithm	CNN	LSTM	CNN-LSTM	CNN	LSTM	CNN-LSTM	CNN	LSTM	CNN-LSTM
Average accuracy (10 runs)	0.9866	0.9874	0.9875	0.9900	0.9855	0.9826	0.9930	0.9905	0.9807
Method	SMOTE								
Algorithm	CNN	LSTM	CNN-LSTM	K-nearest neighbor	Support Vector Machine	Naïve bayes	Decision Trees	Random Forest	Gradient Boosting
Average accuracy (10 runs)	0.9942	0.9830	0.9900	0.9813	0.9766	0.9703	0.9643	0.9601	0.9732



## Experimental Results and Discussion

Method	Original dataset					
Algorithm	K-nearest neighbor	Suppor Vector Machine	Naïve bayes	Decision Trees	Random Forest	Gradient Boosting
Average accuracy (10 runs)	0.9742	0.9789	0.9692	0.9621	0.9735	0.9684

Table 4.6: Result summary (Best result in ten runs)

Method		Algorithm	TP	TN	FP	FN	Accuracy
Ensemble Learning	Approach 1	CNN	832	49	1	1	0.9977
		LSTM	833	48	2	0	0.9977
		CNN-LSTM	833	48	2	0	0.9977
	Approach 2	CNN	833	48	2	0	0.9977
		LSTM	832	48	2	1	0.9966
		CNN-LSTM	833	48	2	0	0.9977
	Approach 3	CNN	833	49	1	0	0.9989
		LSTM	831	48	2	2	0.9955
		CNN-LSTM	833	47	3	0	0.9966
SMOTE		CNN	833	49	1	0	0.9989
		LSTM	832	49	1	1	0.9977
		CNN-LSTM	832	49	1	1	0.9977
		K-nearest neighbor	833	38	12	0	0.9864
		Support Vector Machine	822	49	1	11	0.9864
		Naïve bayes	817	48	2	16	0.9796
		Decision Trees	813	50	0	20	0.9773
		Random Forest	820	49	1	13	0.9841
		Gradient Boosting	822	49	1	11	0.9864
Original dataset (No data balancing techniques)		K-nearest neighbor	833	44	6	0	0.9932
		Support Vector Machine	818	50	0	15	0.9830

## Experimental Results and Discussion

	Naïve bayes	817	48	2	16	0.9796
	Decision Trees	815	49	1	18	0.9785
	Random Forest	815	49	1	18	0.9785
	Gradient Boosting	820	49	1	13	0.9841

It is evident from the above shown summary, CNN method works best both in terms of average result and best result in ten experimental runs. For ensemble learning, approach 3 shows the best accuracy with CNN and for SMOTE CNN works best among all algorithms.

### 4.4 Discussion

In this section, major limitations of this thesis as well as understanding of the result and how it can be interpreted from manufacturing perspective are highlighted.

#### 4.4.1 Event definition and subsequence extraction

The purpose of this thesis was to automatically detect any unusual event occurring on the industrial drying hopper installed in the manufacturing shop floor of a polymer manufacturing industry. As the raw dataset obtained from the machine interface was not in ideal structure for using in a ML or DL algorithms, it needed certain preprocessing from the expert. Even after primary preprocessing done by an expert in this field, the dataset still had missing values and no labeling. The major hindrance in using the dataset for event detection was no accurate definition of unusual event through which temperature profile can be visually divided in various classes. This is why, the events needed to be defined at the very beginning of the experiment considering all variations in the unusual events. Several assumptions had to be made in order to maintain consistency in the definition of an event. For example, if there is a certain small peak in any temperature value for three or four minutes, those are not defined as events. There were two limitations in defining an event. First, no physics based model or mechanics of the drying hopper was available from which the events and their structure like temperature profiles can be understood. Second, in all possible scenarios which can be identified as events, there are too many variations of them which can be potential candidates of the events. For example, hopper 1 hopper outlet temperature varies around 150° F. In some time steps, this temperature value goes below 100° F whereas all other sensors reading are normal. Now the question is, whether this should be identified as events or non-events. This type of confusion was universal in almost all cases where temperature is dropping down or rising all on a sudden. So, events were defined based on the visualization of the temperature profile. Whenever something unusual and lasting phenomenon was notified in the temperature profile which is significantly different from the steady state condition were listed as events.

As mentioned in 3.1.2, the previous study on this particular drying hopper case [36] predicts three different type of unusual events; dryer undersize, conveying issue and cleaning cycle. But in the temperature profile, defining this individual event was not straightforward due to the variation of these events. This is why, the goal of this thesis was to identify any unusual event rather than identifying the type of the event. Another assumption had to be made for defining an event which is the start and end time of an event for simplification. The temperature profiles

## Experimental Results and Discussion

were segmented in an hourly basis and each hour there are 60 entries or time steps. So, if something unusual has started happening in an hour, regardless of the actual start time, that particular hour is defined as an example of an event for labeling simplicity.

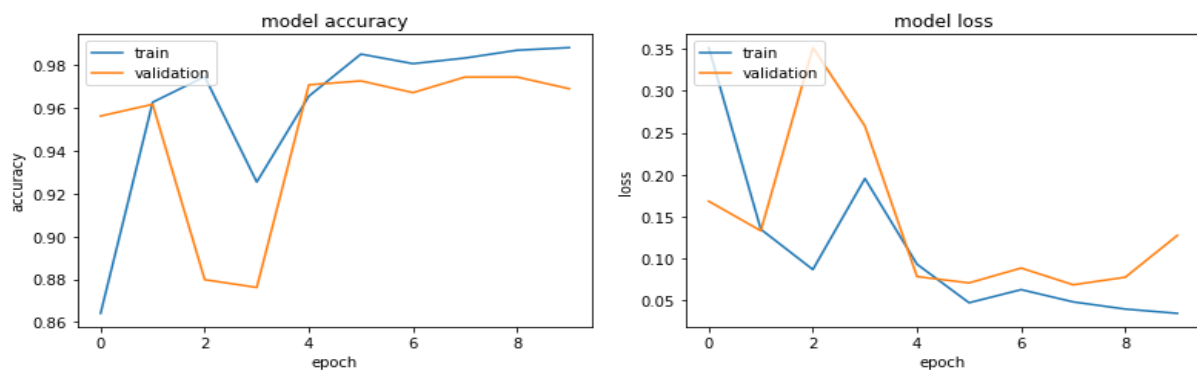
The window approach used in data preprocessing step for extracting subsequences and labeling each subsequence from the labels of each row is also based on some assumptions. For the ease of labeling a window length of sixty minutes and sliding step of 60 minutes were considered for which 4416 examples or subsequences were obtained. As at the beginning each row was assigned a label and events or non-events were defined hourly which indicates in each hour all sixty minutes or rows have same labels. If the start time end time were defined more precisely like the actual start and time instead of hourly basis, then other window approach like different sliding step other than sixty or multiples of sixty can be used. For example, an event starts at 1:26 pm, so the hour from 1:01 pm to 2:00 pm was defined as an event and each row was assigned the same label, 1. But it could have been done otherwise like from 1:01 pm to 1:25 pm, these 25 rows as label 0, remaining 35 rows; 1:26 pm to 2:00 pm as label 1 as the event started from 1:26 pm. In this way, apart from 60 or its multiple, any other values could be used for sliding step. But labeling each subsequence would become more complex from the labelled rows. The reason is not all rows have same labels if this approach is used. For example, if window length is sixty minutes, but sliding step is 1, first subsequence will start from 5:01 am, May 2018, end at 6:00 am, May 1, 2018. But the second subsequence will start from 5:02 am and will end at 6:01 am which was not the same for sliding step of sixty minutes. For sixty minutes sliding step, second event starts at 6:01 am and ends at 7:00 am. Now, when moving the sliding window using sliding step of 1 minute, at one point it will extract a subsequence which starts from 12:39 pm and ends at 1:38 pm. As mentioned above, events started from 1:26 pm precisely, so the rows from 1:26 pm to 1:38 pm are labelled as 1, but rows from 12:39 pm to 1:25 pm are labelled as 0. So, this subsequence has rows with both labels which makes it difficult to define as an event or non-event. If events are defined as any 1 hour where at least one row or minute was labelled as 1, this might lead to a problem. If in an hour at least 30 % rows are labelled as 1, that hour can be defined as event as something unusual is happening in that hour. But if only one row has label 1, rest of the rows have label 0, defining that hour as an event is misleading. The issue is to determine the minimum percentage of rows which has to be labelled as 1 to define that hour as an event. If this problem can be taken care of number of examples would be much higher than the current number. For example, with sliding step of 1 and window length of 60, number of extracted subsequences would be,  $m = (264960 - 60) / 1 + 1 = 264,901$  according to the formula in 3.1.2.5.1, which is 463.88 % more than the current number of examples. As this thesis is dependent on data driven modeling completely with not much input about the physics of the events and non-events with no clear definition, the simplest way to define and visualize the events was chosen for classification.

### 4.4.2 Data imbalance issue

As shown before, training set had 2742 examples from class 0 (77.61%) and 791 examples from class 1 (22.39%) which indicates that the dataset is not balanced. During first experimental trial using simple neural network, all test examples were identified as class 0 as shown in Figure 4.1. From the literature review of the scientific journals and exploration of various data analytics blogs like “towardsdatascience” [103], “analyticsvidhya” [104] and so on, the issue was figured out as imbalance classification issue. Afterwards, CNN, LSTM and CNN-LSTM were applied to the same imbalanced dataset, but resulted same as the simple

## Experimental Results and Discussion

neural network. But machine learning algorithms like SVM, KNN and others were working fine with the imbalanced dataset. This is why data imbalance issue handling techniques like ensemble learning with undersampling and SMOTE as an oversampling technique was used for deep learning algorithms and the result turned out very reasonable. For machine learning algorithms, apart from regular dataset, SMOTE was also used as a data augmentation technique in order to check the performance of ML algorithms. Undersampling was used to combine all undersampling models as an ensemble learner. Without combination of all undersamples, the performance of an algorithm cannot be determined properly using a single undersampling approach. For example, if only one undersample from approach 3 was used for training the algorithm and no ensemble learner was used, there is no certainty that the algorithm will converge to the optimum without overfitting during training with this dataset and will perform well on validation set and test set as shown in Figure 4.23. Even if one undersample performs well on new data, there is still uncertainty regarding the stochastic nature of deep learning algorithm. This particular undersample might work well in one run, but there is no guarantee that it will work well in all experimental runs. This is why ensemble learning with undersampling provides very powerful result as it combines the output from undersamples with majority voting. Through majority voting, even if one undersample out of three undersamples works badly on the new dataset, ensemble learning will take care of this issue by using the prediction from the majority voting. The problem appears when majority of the undersamples perform bad as the voting favors the wrong prediction.



*Figure 4.23: Learning curves with high fluctuation during convergence*

Random oversampling technique was not used as it does not add any value or new information to the dataset. As simple copying of the minority examples oftentimes lead to overfitting and overfitting controlling techniques like L2 regularization and dropout also fails to prevent it. This is why, more structured oversampling approach like SMOTE was used to as data imbalance handling technique.

Apart from these techniques, other data imbalance handling techniques like using class weights to two classes was also tried with this dataset. In this technique, the minority class is assigned higher weight during training so that the convergence is not always inclined to majority class. But this weighting technique did not work at all with this dataset. Even with different weights provided to the minority class, the algorithms were classifying all test examples as majority class. This is why, this thesis explored only ensemble learning with undersampling and SMOTE as oversampling technique.

## Experimental Results and Discussion

### 4.4.3 Result interpretation

Section 4.3 provides the summary of the final result obtained from various algorithms used for classification of the drying hopper MTS dataset. In ensemble learning, the best result for each approach in each run was summarized for all three deep learning algorithms used; CNN, LSTM, CNN-LSTM. In addition to that, best undersampling result for each run was highlighted as well. For SMOTE, average result for three deep learning algorithms and the six ML algorithms were showed and the best results found in ten runs were also highlighted.

The performance measure used in this thesis are precision, recall, f1 score and accuracy. Among them precision, recall and accuracy depends on TP, TN, FP and FN values whereas f1 score is measured from precision and recall. In a manufacturing industry, fault detection is highly important in order to take predictive or preventive maintenance. Now the question is which measure is more important in context of manufacturing. The main goal is to accurately identify events, but if any event is wrongly classified as event that leads to machine failure or other related issue. If this issues is missed by the operator, it might create serious issue depending on the type of failure. This is why, the primary goal is to reduce the FP values (events wrongly identified as non-events) or increase the TN values (events accurately identified as events). The secondary goal is to reduce the FN values (non-events wrongly identified as events) or increase the TP values (non-events accurately identified as non-events). For example, the algorithm identifies a non-event as an event. The operator will check that instance manually and will identify that it is actually a non-event, nothing bad is happening in the machine. If this instance is high in number, it will be waste of time for the operator to check those misleading predictions. But if the algorithm identifies an event as non-event, the operator will not check that prediction as it is predicting non-event. This can lead to serious issue as the operator has no idea that something is happening in the machine. If any event is identified, the operator of the machine has two and a half hours to fix it. The length of each example is one hour, so after training the dataset, any future example of 1 hour length directly extracted from the sensors with primary preprocessing can be used by the algorithm to predict its class and take initiatives accordingly. With industry 4.0 and AI revolution, this type of automation is highly important in any manufacturing plant.

This is why reducing FP values is the primary goal in context of manufacturing. It is also desirable in this experimental setup that across all ten runs number of FP values remain consistent with less outlier for an algorithm with ensemble learning or SMOTE. In each run, the goal is to capture the best model with high accuracy and less FP values. So the best model is selected in the following way: First the model with highest accuracy will be picked as the best model, afterwards second highest accuracy will be picked. If the first one has less number of FP values, this is arguable the best model. This is the general case in almost all experimental runs where the model with best accuracy has least number of FP values as well. But also there are some exceptions. In some cases, the model identified all non-events but missed some events and wrongly classified. But due to the large number of correctly identified non-events the accuracy goes higher. For example, in Figure 4.22, the best result in terms of accuracy and average accuracy across 10 runs, obtained using SMOTE for ML algorithms is K nearest neighbor with best accuracy of 99.32% in run 6 and average accuracy of 98.13% across ten runs. Now the question is whether it is really the best one among ML algorithms when using SMOTE. It is evident it has 6 FP values and 0 FN values whereas the second best, gradient boosting has 98.41% accuracy in run 6 with only 1 FP values, but 13 FN values and 97.32%

## Experimental Results and Discussion

average accuracy across 10 runs. This is why the best method among ML algorithms while using SMOTE is chosen as gradient boosting, not k nearest neighbor as it identifies more events wrongly.

Table 4.5 summarizes the best result in terms of average accuracy of 10 runs. The best average accuracy was found for CNN in SMOTE with 99.42% average accuracy. The FP and FN values are also consistent in ten experimental runs while using CNN with only one outlier where 18 non-events were classified as events (FN). Apart from this instance, number of FP values varies between 0 and 3 whereas number of FN values varies between 0 and 6. The second best model was also found using CNN in ensemble learning approach 3 with 99.30% average accuracy. The FP values are highly consistent across 10 runs and the number varies between 1 and 2. But it has two outliers in the FN values with 11 and 19. The number of FN varies between 0 and 7 which is also higher than the CNN using SMOTE.

Table 4.6 summarizes the best result obtained in 10 runs in terms of accuracy. Again, the best result found in CNN with ensemble learning approach 3 and CNN with SMOTE. Both of these two showed 99.89% accuracy with only one FP value and no FN value. So certainly, CNN is the best algorithm to classify this dataset into two categories with both ensemble learning and SMOTE.

Performance of LSTM and CNN-LSTM is also good enough which is not much lesser than the CNN. But CNN shows not only high accuracy, but also consistency in less number of FP and FN values across ten runs. ML algorithms performance is relatively worse than the deep learning approaches. The maximum average found is 98.13% with KNN in terms of average accuracy across 10 runs when no imbalance data handling technique is used. The second best was SVM with 97.89% average accuracy. In terms of best accuracy among ten runs, KNN provides 99.32% accuracy (no data imbalance technique), but it wrongly classified 6 events. The second best is KNN, SVM and GB (SMOTE) with 98.64% accuracy. KNN again suffers from the high FP values, it has wrongly classified 12 events whereas both SVM and GB wrongly classified only 1 event and 11 non-events. So among ML algorithms, SVM is selected as the best one.

### 5 Conclusion and Future Work

The recent evolution in Industry 4.0, artificial intelligence and internet of things (IOT) have increased the data availability in various domains. This is why data analytics has become highly popular over the year with newer algorithms and techniques being developed regularly for continuous improvement. Among variety of data, time series dataset has become highly available in various domain and various analyses on time series are being performed by researchers frequently. This thesis exploits deep learning algorithms in order to classify MTS data obtained from the sensors installed at the drying hopper in a polymer manufacturing industry. Perfect dataset never really exists in real life scenario which is also the case for the dataset used in this thesis. This is why necessary preprocessing was performed in the dataset to make it usable by deep learning and machine learning algorithms. As classification is a supervised learning approach, any DL or ML algorithms need labelled data. So, the data was also labelled after defining the two categories precisely. To tackle the imbalance data issue, ensemble learning with undersampling and SMOTE as an oversampling technique was explored on different deep learning approaches. The result showed that CNN is arguably the best algorithm for classifying this dataset as events and non-events.

Previously two more research have been performed on this dataset. One of them focused on understanding of the process of an industrial drying hopper [36] and the other one focused on pattern recognition as unsupervised learning [105]. Both of those research made significant contribution towards automatic event detection in the drying hopper. This thesis takes the work from the previous two research to the next step through classifying the dataset into two categories as events and non-events. But there are still other potential future works possible for this specific case which are described below.

- The events need to be defined more precisely with the help of an expert who has the solid understanding of all equipment and sensors of the industrial drying hopper. Afterwards, the events need to be categorized in three different types as mentioned in [36] which might be the hardest part due to too many variations of those events. The next goal should be instead of binary class classification, multi class classification techniques need to be applied if the events can be labelled as various types.
- The sliding window technique used in this thesis used the subsequences extracted as an hour with hourly sliding step. This approach can be further changed into variable windowing approach with different sliding steps to figure out the best window and sliding step size.
- The deep learning approaches used in this thesis are the simplest version of CNN, LSTM and CNN-LSTM. Now a days, lot of variation of these networks have been proposed and those are performing very well in large datasets. For example, residual network which is a variant of CNN has been extremely popular in recent years. Moreover, multi-channel deep CNN, dilated CNN are among other variants of CNN which are used extensively in MTS classification. Similarly, RNN variants like gated recurrent unit (GRU) has also been used for some MTS datasets. Moreover CNN-LSTM has also some established variants like CONV-LSTM which is used oftentimes. For multiclass classification, simple CNN or LSTM model may not work well. So more complex deep learning approaches might be needed to classify the dataset.

## References

- [1] “What is Artificial Intelligence? How Does AI Work? | Built In.” <https://builtin.com/artificial-intelligence> (accessed May 27, 2021).
- [2] E. Oztemel and S. Gursev, “Literature review of Industry 4.0 and related technologies,” *J. Intell. Manuf.*, vol. 31, no. 1, pp. 127–182, 2020, doi: 10.1007/s10845-018-1433-8.
- [3] C. Y. Hsu and W. C. Liu, “Multiple time-series convolutional neural network for fault detection and diagnosis and empirical study in semiconductor manufacturing,” *J. Intell. Manuf.*, vol. 32, no. 3, pp. 823–836, 2021, doi: 10.1007/s10845-020-01591-0.
- [4] S. S. Jones *et al.*, “A multivariate time series approach to modeling and forecasting demand in the emergency department,” *J. Biomed. Inform.*, vol. 42, no. 1, pp. 123–139, Feb. 2009, doi: 10.1016/j.jbi.2008.05.003.
- [5] Z. Du, W. R. Lawrence, W. Zhang, D. Zhang, S. Yu, and Y. Hao, “Interactions between climate factors and air pollution on daily HFMD cases: A time series study in Guangdong, China,” *Sci. Total Environ.*, vol. 656, pp. 1358–1364, Mar. 2019, doi: 10.1016/j.scitotenv.2018.11.391.
- [6] C. Pérez-D’Arpino and J. A. Shah, “Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6175–6182, doi: 10.1109/ICRA.2015.7140066.
- [7] N. Maknickienė, A. V. Rutkauskas, and A. Maknickas, “Investigation of financial market prediction by recurrent neural network,” *Innov. Technol. Sci. Bus. Educ.*, vol. 2, no. 11, pp. 3–8, 2011.
- [8] L. Martín, L. F. Zarzalejo, J. Polo, A. Navarro, R. Marchante, and M. Cony, “Prediction of global solar irradiance based on time series analysis: Application to solar thermal power plants energy production planning,” *Sol. Energy*, vol. 84, no. 10, pp. 1772–1781, Oct. 2010, doi: 10.1016/j.solener.2010.07.002.
- [9] J. F. Muth, “Optimal Properties of Exponentially Weighted Forecasts,” *J. Am. Stat. Assoc.*, vol. 55, no. 290, pp. 299–306, 1960, doi: 10.1080/01621459.1960.10482064.
- [10] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, “The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” *Data Min. Knowl. Discov.*, vol. 31, no. 3, pp. 606–660, 2017, doi: 10.1007/s10618-016-0483-9.
- [11] D. J. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series.,” in *KDD workshop*, 1994, vol. 10, no. 16, pp. 359–370.
- [12] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [13] G. He, Y. Li, and W. Zhao, “An uncertainty and density based active semi-supervised learning scheme for positive unlabeled multivariate time series classification,” *Knowledge-Based Syst.*, vol. 124, pp. 80–92, 2017, doi: 10.1016/j.knosys.2017.03.004.
- [14] M. L. Tuballa and M. L. Abundo, “A review of the development of Smart Grid technologies,” *Renewable and Sustainable Energy Reviews*, vol. 59. Elsevier Ltd, pp. 710–725, Jun. 01, 2016, doi: 10.1016/j.rser.2016.01.011.



- [15] L. Batal, L. Sacchi, R. Bellazzi, and M. Hauskrecht, “Multivariate time series classification with temporal abstractions,” in *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference, FLAIRS-22*, 2009, pp. 344–349.
- [16] K. Yang and C. Shahabi, “An efficient k nearest neighbor search for multivariate time series,” *Inf. Comput.*, vol. 205, no. 1, pp. 65–98, Jan. 2007, doi: 10.1016/j.ic.2006.08.004.
- [17] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, “Classification of time series by shapelet transformation,” *Data Min. Knowl. Discov.*, vol. 28, no. 4, pp. 851–881, 2014, doi: 10.1007/s10618-013-0322-1.
- [18] Y. Chang *et al.*, “A Multi-Task Imputation and Classification Neural Architecture for Early Prediction of Sepsis from Multivariate Clinical Time Series,” *2019 Comput. Cardiol. Conf.*, vol. 45, no. 1, pp. 2–5, 2019, doi: 10.22489/cinc.2019.110.
- [19] J. Lines and A. Bagnall, “Time series classification with ensembles of elastic distance measures,” *Data Min. Knowl. Discov.*, vol. 29, no. 3, pp. 565–592, 2015, doi: 10.1007/s10618-014-0361-2.
- [20] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, “Deep learning for time series classification: a review,” *Data Min. Knowl. Discov.*, vol. 33, no. 4, pp. 917–963, 2019, doi: 10.1007/s10618-019-00619-1.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Adv. Neural Inf. Process. Syst.*, vol. 25, pp. 1097–1105, 2012.
- [22] C. Szegedy *et al.*, “Going Deeper with Convolutions,” 2015.
- [23] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” Sep. 2015, Accessed: May 28, 2021. [Online]. Available: <https://arxiv.org/abs/1409.0473v7>.
- [24] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems*, Sep. 2014, vol. 4, no. January, pp. 3104–3112, Accessed: May 28, 2021. [Online]. Available: <https://arxiv.org/abs/1409.3215v3>.
- [25] A. M. Alayba, V. Palade, M. England, and R. Iqbal, “A combined CNN and LSTM model for Arabic sentiment analysis,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Aug. 2018, vol. 11015 LNCS, pp. 179–191, doi: 10.1007/978-3-319-99740-7\_12.
- [26] T. N. Sainath *et al.*, “Improvements to deep convolutional neural networks for LVCSR,” in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings*, 2013, pp. 315–320, doi: 10.1109/ASRU.2013.6707749.
- [27] C. L. Liu, W. H. Hsaio, and Y. C. Tu, “Time Series Classification with Multivariate Convolutional Neural Network,” *IEEE Trans. Ind. Electron.*, vol. 66, no. 6, pp. 4788–4797, 2019, doi: 10.1109/TIE.2018.2864702.
- [28] H. S. Huang, C. L. Liu, and V. S. Tseng, “Multivariate time series early classification using multi-domain deep neural network,” *Proc. - 2018 IEEE 5th Int. Conf. Data Sci.*

- Adv. Anal. DSAA 2018*, pp. 90–98, 2019, doi: 10.1109/DSAA.2018.00019.
- [29] O. Yazdanbakhsh and S. Dick, “Multivariate Time Series Classification using Dilated Convolutional Neural Network,” 2019, [Online]. Available: <http://arxiv.org/abs/1905.01697>.
- [30] F. Karim, S. Majumdar, H. Darabi, and S. Harford, “Multivariate LSTM-FCNs for time series classification,” *Neural Networks*, vol. 116, pp. 237–245, 2019.
- [31] Z. Guo, P. Liu, J. Yang, and Y. Hu, “Multivariate Time Series Classification Based on MCNN-LSTMs Network,” *ACM Int. Conf. Proceeding Ser.*, pp. 510–517, 2020, doi: 10.1145/3383972.3384013.
- [32] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, “Exploiting multi-channels deep convolutional neural networks for multivariate time series classification,” *Front. Comput. Sci.*, vol. 10, no. 1, pp. 96–112, 2016, doi: 10.1007/s11704-015-4478-2.
- [33] T. C. Images, “Sensor Classification Using Convolutional Neural,” *Sensors (Switzerland)*, no. 1, 2020.
- [34] K. C. Lei and X. D. Zhang, “An approach on discretizing time series using recurrent neural network,” *Proc. - 2018 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2018*, pp. 2522–2526, 2019, doi: 10.1109/BIBM.2018.8621092.
- [35] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, “Recurrent Neural Networks for Multivariate Time Series with Missing Values,” *Sci. Rep.*, vol. 8, no. 1, pp. 1–14, 2018, doi: 10.1038/s41598-018-24271-9.
- [36] J. Lenz, S. Swerdlow, A. Landers, R. Shaffer, A. Geller, and T. Wuest, “Smart services for polymer processing auxiliary equipment: An industrial case study,” *Smart Sustain. Manuf. Syst.*, vol. 4, no. 1, pp. 103–120, 2020, doi: 10.1520/SSMS20200032.
- [37] M. Shokoohi-Yekta, J. Wang, and E. Keogh, “On the non-trivial generalization of Dynamic Time Warping to the multi-dimensional case,” in *SIAM International Conference on Data Mining 2015, SDM 2015*, 2015, pp. 289–297, doi: 10.1137/1.9781611974010.33.
- [38] J. Shen, W. Huang, D. Zhu, and J. Liang, “A Novel Similarity Measure Model for Multivariate Time Series Based on LMNN and DTW,” *Neural Process. Lett.*, vol. 45, no. 3, pp. 925–937, Jun. 2017, doi: 10.1007/s11063-016-9555-5.
- [39] J. Mei, M. Liu, Y. F. Wang, and H. Gao, “Learning a Mahalanobis Distance-Based Dynamic Time Warping Measure for Multivariate Time Series Classification,” *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1363–1374, Jun. 2016, doi: 10.1109/TCYB.2015.2426723.
- [40] N. Vaughan and B. Gabrys, “Scoring and assessment in medical VR training simulators with dynamic time series classification,” *Eng. Appl. Artif. Intell.*, vol. 94, p. 103760, Sep. 2020, doi: 10.1016/j.engappai.2020.103760.
- [41] J. Ircio, A. Lojo, U. Mori, and J. A. Lozano, “Mutual information based feature subset selection in multivariate time series classification,” *Pattern Recognit.*, vol. 108, p. 107525, Dec. 2020, doi: 10.1016/j.patcog.2020.107525.
- [42] T. Górecki and M. Łuczak, “Multivariate time series classification with parametric derivative dynamic time warping,” *Expert Syst. Appl.*, vol. 42, no. 5, pp. 2305–2312,

- Apr. 2015, doi: 10.1016/j.eswa.2014.11.007.
- [43] S. Seto, W. Zhang, and Y. Zhou, “Multivariate time series classification using dynamic time warping template selection for human activity recognition,” in *Proceedings - 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015*, 2015, pp. 1399–1406, doi: 10.1109/SSCI.2015.199.
- [44] M. Łuczak, “Univariate and multivariate time series classification with parametric integral dynamic time warping,” *J. Intell. Fuzzy Syst.*, vol. 33, no. 4, pp. 2403–2413, Jan. 2017, doi: 10.3233/JIFS-17523.
- [45] S. Liu and C. Liu, “Scale-varying dynamic time warping based on hesitant fuzzy sets for multivariate time series classification,” *Meas. J. Int. Meas. Confed.*, vol. 130, pp. 290–297, Dec. 2018, doi: 10.1016/j.measurement.2018.07.094.
- [46] M. Łuczak, “Combining raw and normalized data in multivariate time series classification with dynamic time warping,” *J. Intell. Fuzzy Syst.*, vol. 34, no. 1, pp. 373–380, Jan. 2018, doi: 10.3233/JIFS-171393.
- [47] T. Górecki, “Classification of time series using combination of DTW and LCSS dissimilarity measures,” *Commun. Stat. Simul. Comput.*, vol. 47, no. 1, pp. 263–276, Jan. 2018, doi: 10.1080/03610918.2017.1280829.
- [48] M. G. Baydogan and G. Runger, “Learning a symbolic representation for multivariate time series classification,” *Data Min. Knowl. Discov.*, vol. 29, no. 2, pp. 400–422, Mar. 2015, doi: 10.1007/s10618-014-0349-y.
- [49] J. Lin, E. Keogh, L. Wei, and S. Lonardi, “Experiencing SAX: A novel symbolic representation of time series,” *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 107–144, Oct. 2007, doi: 10.1007/s10618-007-0064-z.
- [50] P. Schäfer and M. Höggqvist, “SFA: A symbolic fourier approximation and index for similarity search in high dimensional datasets,” in *ACM International Conference Proceeding Series*, 2012, pp. 516–527, doi: 10.1145/2247596.2247656.
- [51] T. Le Nguyen, S. Gsponer, I. Ilie, M. O’Reilly, and G. Ifrim, “Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations,” *Data Min. Knowl. Discov.*, vol. 33, no. 4, pp. 1183–1222, Jul. 2019, doi: 10.1007/s10618-019-00633-3.
- [52] B. Dhariyal, T. Le Nguyen, S. Gsponer, and G. Ifrim, “An Examination of the State-of-the-Art for Multivariate Time Series Classification Machine Learning Methods for Text Classification View project Text and Web Mining View project An Examination of the State-of-the-Art for Multivariate Time Series Classification,” doi: 10.1109/ICDMW51313.2020.00042.
- [53] P. Schäfer and U. Leser, “Multivariate Time Series Classification with WEASEL+MUSE,” vol. 11, Nov. 2017, Accessed: Jun. 22, 2021. [Online]. Available: <http://arxiv.org/abs/1711.11343>.
- [54] D. Yang, H. Chen, Y. Song, and Z. Gong, “Granger Causality for Multivariate Time Series Classification,” *Proc. - 2017 IEEE Int. Conf. Big Knowledge, ICBK 2017*, pp. 103–110, 2017, doi: 10.1109/ICBK.2017.36.
- [55] L. Batal, L. Sacchi, R. Bellazzi, and M. Hauskrecht, “Multivariate time series

- classification with temporal abstractions,” *Proc. 22nd Int. Florida Artif. Intell. Res. Soc. Conf. FLAIRS-22*, pp. 344–349, 2009.
- [56] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, “Convolutional neural networks for time series classification,” *J. Syst. Eng. Electron.*, vol. 28, no. 1, pp. 162–169, 2017, doi: 10.21629/JSEE.2017.01.18.
- [57] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, “The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” *Data Min. Knowl. Discov.*, vol. 35, no. 2, pp. 401–449, Mar. 2021, doi: 10.1007/s10618-020-00727-3.
- [58] W. Song, L. Liu, M. Liu, W. Wang, X. Wang, and Y. Song, “Representation Learning with Deconvolution for Multivariate Time Series Classification and Visualization,” *Commun. Comput. Inf. Sci.*, vol. 1257 CCIS, pp. 310–326, 2020, doi: 10.1007/978-981-15-7981-3\_22.
- [59] K. S. Kiangala and Z. Wang, “An Effective Predictive Maintenance Framework for Conveyor Motors Using Dual Time-Series Imaging and Convolutional Neural Network in an Industry 4.0 Environment,” *IEEE Access*, vol. 8, pp. 121033–121049, 2020, doi: 10.1109/ACCESS.2020.3006788.
- [60] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, “Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2015-Augus, pp. 4580–4584, 2015, doi: 10.1109/ICASSP.2015.7178838.
- [61] E. Y. Hsu, C. L. Liu, and V. S. Tseng, “Multivariate time series early classification with interpretability using deep learning and attention mechanism,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Apr. 2019, vol. 11441 LNAI, pp. 541–553, doi: 10.1007/978-3-030-16142-2\_42.
- [62] M. Khan, H. Wang, A. Ngueilbaye, and A. Elfatyany, “End-to-end multivariate time series classification via hybrid deep learning architectures,” *Pers. Ubiquitous Comput.*, 2020, doi: 10.1007/s00779-020-01447-7.
- [63] A. M. Tripathi, “Enhancing Multivariate Time Series Classification Using LSTM and Evidence Feed Forward HMM,” *Proc. Int. Jt. Conf. Neural Networks*, 2020, doi: 10.1109/IJCNN48605.2020.9207636.
- [64] G. He, Y. Duan, Y. Li, T. Qian, J. He, and X. Jia, “Active learning for multivariate time series classification with positive unlabeled data,” *Proc. - Int. Conf. Tools with Artif. Intell. ICTAI*, vol. 2016-Janua, pp. 178–185, 2016, doi: 10.1109/ICTAI.2015.38.
- [65] M. González, C. Bergmeir, I. Triguero, Y. Rodríguez, and J. M. Benítez, “Self-labeling techniques for semi-supervised time series classification: an empirical study,” *Knowl. Inf. Syst.*, vol. 55, no. 2, pp. 493–528, 2018, doi: 10.1007/s10115-017-1090-9.
- [66] L. Wei and E. Keogh, “Semi-supervised time series classification,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 748–753.
- [67] “Unix time - Wikipedia.” [https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time) (accessed Jun. 10, 2021).

- [68] L. Gruenwald, H. Chok, and M. Aboukhamis, “Using data mining to estimate missing sensor data,” in *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2007, pp. 207–212, doi: 10.1109/ICDMW.2007.103.
- [69] “Labelling Time Series Data in Python | by Lucy Rothwell | Towards Data Science.” <https://towardsdatascience.com/labelling-time-series-data-in-python-af62325e8f60> (accessed Jun. 11, 2021).
- [70] J. Chris Bishop, C. Bishop, G. Hinton, and P. Bishop, “Neural networks for pattern recognition. Advanced texts in econometrics.” Oxford: Clarendon Press, 1995.
- [71] “Undersampling Algorithms for Imbalanced Classification.” <https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/> (accessed Jun. 12, 2021).
- [72] “The 5 Most Useful Techniques to Handle Imbalanced Datasets - KDnuggets.” <https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html> (accessed Jun. 12, 2021).
- [73] “Random Oversampling and Undersampling for Imbalanced Classification.” <https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/> (accessed Jun. 12, 2021).
- [74] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2011, doi: 10.1613/jair.953.
- [75] “Imbalanced Learning: Foundations, Algorithms, and Applications - Google Books.” <https://books.google.com/books?hl=en&lr=&id=CVHx-Gp9jzUC&oi=fnd&pg=PT9&dq=Imbalanced+Learning:+Foundations,+Algorithms,+and+Applications+1st+Edition&ots=2iMkJjGobj&sig=ydvxXpVL7gZa66NLKwKctoEwyJw#v=onepage&q&f=false> (accessed Jun. 12, 2021).
- [76] “Bank Data: SMOTE. This will be a short post before we... | by Zaki Jefferson | Analytics Vidhya | Medium.” <https://medium.com/analytics-vidhya/bank-data-smote-b5cb01a5e0a2> (accessed Jun. 12, 2021).
- [77] “(34) Handling imbalanced dataset in machine learning | Deep Learning Tutorial 21 (Tensorflow2.0 & Python) - YouTube.” <https://www.youtube.com/watch?v=JnlM4yLFNuo&t=1914s> (accessed Jun. 12, 2021).
- [78] “Perceptron - Wikipedia.” <https://en.wikipedia.org/wiki/Perceptron> (accessed May 28, 2021).
- [79] “[Memo Sheet] Deep Neural Network. Have you ever dreamed of a place where... | by Harry Pommier | Zenika.” <https://medium.zenika.com/memo-sheet-deep-neural-network-dedcda759d9c> (accessed May 28, 2021).
- [80] “Activation Functions for Artificial Neural Networks - mlxtend.” [http://rasbt.github.io/mlxtend/user\\_guide/general\\_concepts/activation-functions/](http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/) (accessed May 28, 2021).
- [81] “Difference Between a Batch and an Epoch in a Neural Network.” <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/> (accessed May 28, 2021).

## References

- [82] “GitHub - Kulbear/deep-learning-coursera: Deep Learning Specialization by Andrew Ng on Coursera.” <https://github.com/Kulbear/deep-learning-coursera> (accessed Jun. 15, 2021).
- [83] “Convolutional neural networks for time series forecasting - Python for Finance Cookbook.” <https://subscription.packtpub.com/book/data/9781789618518/10/ch10lv11sec63/convolutional-neural-networks-for-time-series-forecasting> (accessed Jun. 15, 2021).
- [84] “Vanilla Recurrent Neural Network - Machine Learning Notebook.” [https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent\\_neural\\_networks](https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks) (accessed Jun. 17, 2021).
- [85] “Chapter 4 Recurrent neural networks and their applications in NLP | Modern Approaches in Natural Language Processing.” [https://compstat-lmu.github.io/seminar\\_nlp\\_ss20/recurrent-neural-networks-and-their-applications-in-nlp.html](https://compstat-lmu.github.io/seminar_nlp_ss20/recurrent-neural-networks-and-their-applications-in-nlp.html) (accessed Jun. 17, 2021).
- [86] A. Graves, “Generating Sequences With Recurrent Neural Networks,” Aug. 2013, Accessed: Jun. 18, 2021. [Online]. Available: <http://arxiv.org/abs/1308.0850>.
- [87] “MIT Deep Learning 6.S191.” <http://introtodeeplearning.com/> (accessed Jun. 18, 2021).
- [88] “CS 230 - Recurrent Neural Networks Cheatsheet.” <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks> (accessed Jun. 18, 2021).
- [89] “Recurrent Neural Network (RNN) Tutorial for Beginners.” <https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn> (accessed Jun. 18, 2021).
- [90] “LSTM | Introduction to LSTM | Long Short Term Memor.” <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/> (accessed Jun. 18, 2021).
- [91] I. E. Livieris, E. Pintelas, and P. Pintelas, “A CNN–LSTM model for gold price time-series forecasting,” *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17351–17360, Dec. 2020, doi: 10.1007/s00521-020-04867-x.
- [92] L. Deng and J. C. Platt, “Ensemble deep learning for speech recognition,” 2014.
- [93] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, “Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7657 LNCS, pp. 216–223, doi: 10.1007/978-3-642-35395-6\_30.
- [94] “SVM | What is SVM | Support Vector Machine | SVM in Python.” <https://www.analyticsvidhya.com/blog/2021/04/insight-into-svm-support-vector-machine-along-with-code/> (accessed Jun. 19, 2021).
- [95] “SVM | Support Vector Machine Algorithm in Machine Learning.” <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/> (accessed Jun. 19, 2021).
- [96] V. Jakkula, “Tutorial on Support Vector Machine (SVM),” *Sch. EECS, Washingt. State*

## References

- Univ., pp. 1–13, 2011, [Online]. Available: <http://www.ccs.neu.edu/course/cs5100f11/resources/jakkula.pdf>.
- [97] “Decision Tree Algorithm, Explained - KDnuggets.” <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html> (accessed Jun. 20, 2021).
- [98] “Random Forest Simple Explanation. Understanding the Random Forest with an... | by Will Koehrsen | Medium.” <https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d> (accessed Jun. 20, 2021).
- [99] M. Reza, S. Miri, and R. Javidan, “A Hybrid Data Mining Approach for Intrusion Detection on Imbalanced NSL-KDD Dataset,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 6, pp. 1–33, 2016, doi: 10.14569/ijacsa.2016.070603.
- [100] “KNN Classification using Scikit-learn - DataCamp.” <https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn> (accessed Jun. 27, 2021).
- [101] “Dynamic Time Warping k-Nearest Neighbors Classifier (KNNClassifier) — sequentia 0.12.0 documentation.” <https://sequentia.readthedocs.io/en/latest/sections/classifiers/knn.html> (accessed Jun. 27, 2021).
- [102] “How to use Learning Curves to Diagnose Machine Learning Model Performance.” <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/> (accessed Jun. 25, 2021).
- [103] “Towards Data Science.” <https://towardsdatascience.com/> (accessed Jun. 29, 2021).
- [104] “Analytics Vidhya - Learn Machine learning, artificial intelligence, business analytics, data science, big data, data visualizations tools and techniques. | Analytics Vidhya.” <https://www.analyticsvidhya.com/> (accessed Jun. 29, 2021).
- [105] V. Kapp, M. C. May, G. Lanza, and T. Wuest, “Pattern recognition in multivariate time series: Towards an automated event detection method for smart manufacturing systems,” *J. Manuf. Mater. Process.*, vol. 4, no. 3, 2020, doi: 10.3390/JMMP4030088.

## Appendix

**N.B. Implementation in python is inspired from YouTube Channel “Codebasics” and website “<https://machinelearningmastery.com/>”.**

### Ensemble Learning (CNN, LSTM, CNN-LSTM):

```

.....

import numpy as np

import pandas as pd

from numpy import array

labelled_data_file_name = 'Hopper_labelled_data.csv'

dataset = pd.read_csv(labelled_data_file_name, skiprows=0)

dataset[' Delivery Air Dewpoint (F)']= pd.to_numeric(dataset[' Delivery Air Dewpoint (F)'],
errors = 'coerce')

dataset['Regen Temp Wheel Inlet (F)']= pd.to_numeric(dataset['Regen Temp Wheel Inlet
(F)'], errors = 'coerce')

dataset['Hopper 1 Hopper Outlet Temp (F)']= pd.to_numeric(dataset['Hopper 1 Hopper Outlet
Temp (F)'], errors = 'coerce')

dataset['Hopper 1 Drying Monitor 2 Temp (F)']= pd.to_numeric(dataset['Hopper 1 Drying
Monitor 2 Temp (F)'], errors = 'coerce')

dataset['Hopper 1 Drying Monitor 4 Temp (F)']= pd.to_numeric(dataset['Hopper 1 Drying
Monitor 4 Temp (F)'], errors = 'coerce')

dataset['Hopper 1 Drying Monitor 6 Temp (Top) (F)']= pd.to_numeric(dataset['Hopper 1
Drying Monitor 6 Temp (Top) (F)'], errors = 'coerce')

dataset['labels']= pd.to_numeric(dataset['labels'], errors = 'coerce')

dataset.columns = ['DAD', 'RTAS', 'RTWI', 'RTWO', 'H1DAT', 'H1HOT', 'H1DM1T',
'H1DM2T', 'H1DM3T', 'H1DM4T', 'H1DM5T', 'H1DM6T', 'labels']

dataset['DAD'].fillna(method='pad', inplace=True)

dataset['RTWI'].fillna(method='pad', inplace=True)

dataset['H1HOT'].fillna(method='pad', inplace=True)

dataset['H1DM2T'].fillna(method='pad', inplace=True)

dataset['H1DM4T'].fillna(method='pad', inplace=True)

dataset['H1DM6T'].fillna(method='pad', inplace=True)

columns_scaling = ['DAD', 'RTAS', 'RTWI', 'RTWO', 'H1DAT', 'H1HOT', 'H1DM1T',
'H1DM2T', 'H1DM3T', 'H1DM4T', 'H1DM5T', 'H1DM6T']

from sklearn.preprocessing import MinMaxScaler

```



```

scaler = MinMaxScaler()
dataset[columns_scaling] = scaler.fit_transform(dataset[columns_scaling])
rows, columns = dataset.shape
count_test = int((rows/60)*0.2)*60
count_train = rows - count_test
dataset_train = dataset[:count_train]
dataset_test = dataset[count_train:]
dataset_train = np.array(dataset_train)
dataset_test = np.array(dataset_test)
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = 60*i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[60*i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
# choose a number of time steps
n_steps = 60
# convert into input/output
trainX_pre, trainy_pre = split_sequences(dataset_train, n_steps)
trainy_pre = trainy_pre.reshape(trainy_pre.shape[0],1)
trainX_pre = np.asarray(trainX_pre).astype(np.float32)
testX, testy = split_sequences(dataset_test, n_steps)

```

```

testy = testy.reshape(testy.shape[0],1)
testX = np.asarray(testX).astype(np.float32)
print(trainX_pre.shape, trainy_pre.shape, testX.shape, testy.shape)
dim1 =trainX_pre.shape[0]
dim2 =trainX_pre.shape[1]
dim3 =trainX_pre.shape[2]
trainX_pre = trainX_pre.reshape(dim1, dim2*dim3)
trainX_pre = pd.DataFrame(trainX_pre)
trainy_pre = pd.DataFrame(trainy_pre)
trainy_pre.columns = ['labels']
dataframe = pd.concat([trainX_pre, trainy_pre], axis =1)
count_train0, count_train1 = dataframe.labels.value_counts()
dataframe_train0 = dataframe[dataframe['labels']==0]
dataframe_train1 = dataframe[dataframe['labels']==1]
dataframe_train0.shape, dataframe_train1.shape
def train_set(df_majority, df_minority, start, end):
    df_train = pd.concat([df_majority[start:end], df_minority], axis =0)
    df_train = df_train.sample(frac=1)
    trainX = df_train.drop(['labels'], axis =1)
    trainy = df_train['labels']
    trainX = np.array(trainX)
    trainy = np.array(trainy)
    new_dim = trainX.shape[0]
    trainX = trainX.reshape(new_dim, dim2, dim3)
    trainy = trainy.reshape(trainy.shape[0],1)
    return trainX, trainy

from numpy import mean
from numpy import std
from tensorflow import keras

```

```

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import TimeDistributed
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sn

def CNN(trainX, trainy, testX, testy, loss):
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
    model = Sequential()
    model.add(Conv1D(filters=16, kernel_size=5, activation='relu',
input_shape=(n_timesteps,n_features)))
    model.add(Conv1D(filters=16, kernel_size=5, activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(200, activation='relu'))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss=loss, optimizer='adam', metrics=['accuracy'])
    history = model.fit(trainX, trainy, validation_split = 0.1, epochs=100, batch_size=64)
    print(history.history.keys())
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])

```

```

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
print(model.evaluate(testX, testy, batch_size=64))
yp = model.predict(testX)
y_pred = []
for element in yp:
    if element>0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
print(classification_report(testy, y_pred))
cm = tf.math.confusion_matrix(labels = testy, predictions =y_pred)
plt.figure(figsize =(10,6))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
return y_pred

trainX11, trainy11 = train_set(dataframe_train0, dataframe_train1.sample(548), 0, 548)

```

```

print(trainX11.shape, trainy11.shape, testX.shape, testy.shape)
trainX21, trainy21 = train_set(dataframe_train0, dataframe_train1, 0, 791)
print(trainX21.shape, trainy21.shape, testX.shape, testy.shape)
trainX31, trainy31 = train_set(dataframe_train0, dataframe_train1, 0, 914)
print(trainX31.shape, trainy31.shape, testX.shape, testy.shape)
y_pred11_CNN = CNN(trainX11, trainy11, testX, testy, 'binary_crossentropy')
y_pred21_CNN = CNN(trainX21, trainy21, testX, testy, 'binary_crossentropy')
y_pred31_CNN = CNN(trainX31, trainy31, testX, testy, 'binary_crossentropy')
trainX12, trainy12 = train_set(dataframe_train0, dataframe_train1.sample(548), 548, 1096)
print(trainX12.shape, trainy12.shape, testX.shape, testy.shape)
trainX22, trainy22 = train_set(dataframe_train0, dataframe_train1, 791, 1582)
print(trainX22.shape, trainy22.shape, testX.shape, testy.shape)
trainX32, trainy32 = train_set(dataframe_train0, dataframe_train1, 914, 1828)
print(trainX32.shape, trainy32.shape, testX.shape, testy.shape)
y_pred12_CNN = CNN(trainX12, trainy12, testX, testy, 'binary_crossentropy')
y_pred22_CNN = CNN(trainX22, trainy22, testX, testy, 'binary_crossentropy')
y_pred32_CNN = CNN(trainX32, trainy32, testX, testy, 'binary_crossentropy')
trainX13, trainy13 = train_set(dataframe_train0, dataframe_train1.sample(548), 1096, 1644)
print(trainX13.shape, trainy13.shape, testX.shape, testy.shape)
trainX23, trainy23 = train_set(dataframe_train0, dataframe_train1, 1582, 2742)
print(trainX23.shape, trainy23.shape, testX.shape, testy.shape)
trainX33, trainy33 = train_set(dataframe_train0, dataframe_train1, 1828, 2742)
print(trainX33.shape, trainy33.shape, testX.shape, testy.shape)
y_pred13_CNN = CNN(trainX13, trainy13, testX, testy, 'binary_crossentropy')
y_pred23_CNN = CNN(trainX23, trainy23, testX, testy, 'binary_crossentropy')
y_pred33_CNN = CNN(trainX33, trainy33, testX, testy, 'binary_crossentropy')

trainX14, trainy14 = train_set(dataframe_train0, dataframe_train1.sample(549), 1644, 2193)
print(trainX14.shape, trainy14.shape, testX.shape, testy.shape)
y_pred14_CNN = CNN(trainX14, trainy14, testX, testy, 'binary_crossentropy')

```

```

trainX15, trainy15 = train_set(dataframe_train0, dataframe_train1.sample(549), 2193, 2742)
print(trainX15.shape, trainy15.shape, testX.shape, testy.shape)
y_pred15_CNN = CNN(trainX15, trainy15, testX, testy, 'binary_crossentropy')
def final_result23(y_pred1, y_pred2, y_pred3):
    y_pred_final = y_pred1.copy()
    for i in range(len(y_pred1)):
        n_ones = y_pred1[i]+y_pred2[i]+y_pred3[i]
        if n_ones>1:
            y_pred_final[i]=1
        else:
            y_pred_final[i]=0
    return y_pred_final
y_pred_final2_CNN = final_result23(y_pred21_CNN, y_pred22_CNN, y_pred23_CNN)
y_pred_final3_CNN = final_result23(y_pred31_CNN, y_pred32_CNN, y_pred33_CNN)
print(classification_report(testy, y_pred_final2_CNN), classification_report(testy,
y_pred_final3_CNN))
cm2_CNN = tf.math.confusion_matrix(labels = testy, predictions =y_pred_final2_CNN)
plt.figure(figsize =(10,6))
sn.heatmap(cm2_CNN, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')

cm3_CNN = tf.math.confusion_matrix(labels = testy, predictions =y_pred_final3_CNN)
plt.figure(figsize =(10,6))
sn.heatmap(cm3_CNN, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
def final_result1(y_pred1, y_pred2, y_pred3, y_pred4, y_pred5):
    y_pred_final = y_pred1.copy()
    for i in range(len(y_pred1)):
        n_ones = y_pred1[i]+y_pred2[i]+y_pred3[i]+y_pred4[i]+y_pred5[i]

```

```

if n_ones>2:
    y_pred_final[i]=1
else:
    y_pred_final[i]=0
return y_pred_final

y_pred_final1_CNN = final_result1(y_pred11_CNN, y_pred12_CNN, y_pred13_CNN,
y_pred14_CNN, y_pred15_CNN)

print(classification_report(testy, y_pred_final1_CNN))

cm1_CNN = tf.math.confusion_matrix(labels = testy, predictions =y_pred_final1_CNN)

plt.figure(figsize =(10,6))

sn.heatmap(cm1_CNN, annot=True, fmt='d')

plt.xlabel('Predicted')

plt.ylabel('Truth')

LSTM model cell:

n_timesteps, n_features, n_outputs = trainX21.shape[1], trainX21.shape[2], trainy21.shape[1]

model = Sequential()

model.add(LSTM(100, input_shape=(n_timesteps,n_features)))

model.add(Dropout(0.5))

model.add(Dense(200, activation='relu'))

model.add(Dense(n_outputs, activation='sigmoid'))

model.summary()

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(trainX21, trainy21, validation_split = 0.1, epochs=10, batch_size=64)

print(history.history.keys())

# summarize history for accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

```

```

plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
print(model.evaluate(testX, testy, batch_size=64))
yp = model.predict(testX)
y_pred21 =[]
for element in yp:
    if element>0.5:
        y_pred21.append(1)
    else:
        y_pred21.append(0)
print(classification_report(testy, y_pred21))
cm = tf.math.confusion_matrix(labels = testy, predictions =y_pred21)
plt.figure(figsize =(10,6))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
CNN-LSTM cell:
n_timesteps, n_features, n_outputs = trainX11.shape[1], trainX11.shape[2], trainy11.shape[1]
n_steps, n_length = 4, 15
trainX11 = trainX11.reshape((trainX11.shape[0], n_steps, n_length, n_features))
testX = testX.reshape((testX.shape[0], n_steps, n_length, n_features))
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=16, kernel_size=5, activation='relu'),
input_shape=(None,n_length,n_features)))

```



```

model.add(TimeDistributed(Dropout(0.5)))
model.add(TimeDistributed(Conv1D(filters=16, kernel_size=5, activation='relu')))
model.add(TimeDistributed(Dropout(0.5)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(100))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu'))
model.add(Dense(n_outputs, activation='sigmoid'))
model.summary()
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(trainX11, trainy11, validation_split = 0.1, epochs=10, batch_size=64)
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
print(model.evaluate(testX, testy, batch_size=64))

```

```

yp = model.predict(testX)
y_pred11 = []
for element in yp:
    if element>0.5:
        y_pred11.append(1)
    else:
        y_pred11.append(0)
print(classification_report(testy, y_pred11))
cm = tf.math.confusion_matrix(labels = testy, predictions =y_pred11)
plt.figure(figsize =(10,6))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')

```

### **SMOTE (CNN, LSTM, CNN-LSTM, ML algorithms) cells:**

```

dim1 =trainX_pre.shape[0]
dim2 =trainX_pre.shape[1]
dim3 =trainX_pre.shape[2]
dim1, dim2, dim3
trainX_pre = trainX_pre.reshape(dim1, dim2*dim3)
trainX_pre.shape
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy = 'minority')
trainX_sm, trainy_sm = smote.fit_resample(trainX_pre, trainy_pre)
trainX_sm = pd.DataFrame(trainX_sm)
trainy_sm = pd.DataFrame(trainy_sm)
trainy_sm.columns = ['labels']
dataframe = pd.concat([trainX_sm, trainy_sm], axis =1)
dataframe = dataframe.sample(frac=1)
dataframe.labels.value_counts()
trainX = dataframe.drop(['labels'], axis =1)

```

```

trainy = dataframe['labels']
trainX = np.array(trainX)
trainy = np.array(trainy)
new_dim = trainX.shape[0]
trainX = trainX.reshape(new_dim, dim2, dim3)
trainy = trainy.reshape(trainy.shape[0],1)
trainX.shape, trainy.shape
print(trainX.shape, trainy.shape, testX.shape, testy.shape)

```

### **ML algorithms application cells:**

```

from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sn
def define_models(models=dict()):
    models['knn'] = KNeighborsClassifier(n_neighbors=7)
    models['cart'] = DecisionTreeClassifier()
    models['svm'] = SVC(kernel = 'poly')
    models['bayes'] = GaussianNB()
    models['bag'] = BaggingClassifier(n_estimators=50)
    models['rf'] = RandomForestClassifier(n_estimators=50)
    models['et'] = ExtraTreesClassifier(n_estimators=100)

```

```

models['gbm'] = GradientBoostingClassifier(n_estimators=100)
print('Defined %d models' % len(models))

return models

def evaluate_model(trainX, trainy, testX, testy, model)
    model.fit(trainX, trainy)
    yhat = model.predict(testX)
    cm = tf.math.confusion_matrix(labels = testy, predictions =yhat)
    plt.figure(figsize =(10,6))
    sn.heatmap(cm, annot=True, fmt='d')
    plt.xlabel('Predicted')
    plt.ylabel('Truth')
    accuracy = accuracy_score(testy, yhat)
    print(classification_report(testy, yhat))
    return accuracy * 100.0

def evaluate_models(trainX, trainy, testX, testy, models):
    results = dict()
    for name, model in models.items():
        # evaluate the model
        results[name] = evaluate_model(trainX, trainy, testX, testy, model)
        # show process
        print('>%s: %.3f' % (name, results[name]))
    return results

def summarize_results(results, maximize=True):
    # create a list of (name, mean(scores)) tuples
    mean_scores = [(k,v) for k,v in results.items()]
    # sort tuples by mean score
    mean_scores = sorted(mean_scores, key=lambda x: x[1])
    # reverse for descending order (e.g. for accuracy)
    if maximize:
        mean_scores = list(reversed(mean_scores))

```

```
print()
for name, score in mean_scores:
    print('Name=%s, Score=%.3f' % (name, score))
models = define_models()
results = evaluate_models(trainX, trainy, testX, testy, models)
summarize_results(results)
```