# Operator Decision Aid Design via Multi-Dimensional Time-Series Event Prediction: A Hydrocracking Unit Application

by

**Ainaz Jamshidi**

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Engineering

KOÇ ÜNİVERSİTESİ

September 6, 2021

**Operator Decision Aid Design via Multi-Dimensional Time-Series Event Prediction: A Hydrocracking Unit Application**

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

**Ainaz Jamshidi**

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Dr. Barış Akgün (Advisor, Koç University)

_____

Assist. Prof. Dr. Mustafa Gökçe Baydoğan (Boğaziçi University)

_____

Assoc. Prof. Dr. Mehmet Gönen (Koç University)

Date: _____

*To my savior angel, the one whose arms are the suppressor of all the chaos out there...*

# ABSTRACT

**Operator Decision Aid Design via Multi-Dimensional Time-Series Event Prediction: A Hydrocracking Unit Application**
**Ainaz Jamshidi**
**Master of Science in Computer Engineering**
**September 6, 2021**

Refinery upsets lead to lower quality product, process shutdowns or equipment failure. Human operators monitor refinery processes and intervene if needed to prevent such upsets or reduce their effects. The equipment required for these processes are large and complex. There are usually many sensors to pay attention to, making the monitoring job difficult. In this work, we propose a data-driven methodology to develop operator decision aids that aim to decrease the monitoring burden. The decision aid raises *soft-alarms* to notify the operator that an undesirable event will happen within a certain time-frame. The event definition can be process specific (e.g. more than 5 degrees temperature rise within 15 minutes) or generic (e.g. the sensor measurement will be anomalous within 15 minutes). The idea is to predict these events using machine-learning models trained from historical data. The sensory information for a given equipment unit is treated as a multi-dimensional time-series. The events are predicted either by time-series regression and applying the event logic on the forecasts or by classification of whether there will be an event or not in a given time-frame. These events are usually rare which makes them hard to predict. This results in the trade-off between capturing all the events (ratio of correctly predicted events over total number of events) versus capturing the true events (correctly predicted events to all the positive predictions) or the trade-off between *recall/sensitivity* and *precision*. If the decision aid captures all the events but mispredicts many more, the trust of the operator will be eroded. Thus, we concentrate on getting good event prediction precision values while still keeping reasonable recalls. As such, we use the $F_{0.1}$ score to evaluate our results and secondarily look at whether we get at least 0.5 precision and 0.1 recall. We evaluate our approach with multiple machine learning models on the four beds of a real refinery hydrocracker unit. Our first evaluation is applying the event definitions directly. Our results show

that there is at least one method that achieve high $F_{0.1}$ scores for each bed, implying that building a decision aid is viable. However, there isn't a clear winner among the machine-learning methods and performance changes between different beds, which we argue is due to the data. Predicting events for beds, instead of sensors, did not yield significant gains. We additionally suggest an adaptive thresholding method to tune the precision-recall trade-off so that the operators have more flexibility. This is based on the running standard deviation of the prediction errors. Our evaluations show that the tuning is possible and it can be used to increase the $F_{0.1}$ performance significantly, albeit decreasing recall values. For the last evaluation, we directly predict the change of temperature values within the target time horizon in our regression methodology. We show that there are performance gains for a subset of our data utilizing this perspective of time series prediction.

# ÖZETÇE

**Çok Boyutlu Zaman Serisi Olay Tahminlemesi ile Operatör Karar Destek Sistemi Tasarımı: Bir Hidrokraking Ünitesi Uygulaması**

**Ainaz Jamshidi**

**Bilgisayar Mühendisliği, Yüksek Lisans**

**6 Eylül 2021**

İstenmeyen rafineri olayları daha düşük kaliteli ürüne, süreç kesintilerine veya ekipman arızalarına yol açar. Operatörler, rafineri süreçlerini izler ve gerektiğinde bu tür durumları önlemek veya etkilerini azaltmak için müdahale ederler. Rafineri süreçleri için gerekli donanımlar büyük ve karmaşıktır. Genellikle dikkat edilmesi gereken birçok sensör vardır ve bu da izleme işini zorlaştırır. Bu çalışmada, izleme yükünü azaltmayı amaçlayan operatör karar destek sistemleri geliştirmek için veriye dayalı bir metodoloji önermekteyiz. Karar desteği, belirli bir zaman çerçevesi içinde istenmeyen bir olayın meydana geleceğini operatöre bildirmek için *yumuşak alarmlar* verir. Olay tanımı sürece özel (ör. 15 dakika içinde 5 dereceden fazla sıcaklık artışı) veya genel (ör. 15 dakika içinde anormal sensör ölçümü) olabilir. Buradaki fikir, geçmiş verilerden eğitilmiş yapay öğrenme modelleri kullanarak bu olayları tahmin etmektir. Belirli bir ekipman birimi için sensör verileri, çok boyutlu bir zaman serisi olarak ele alınır. Olaylar, ya zaman serisi regresyonu ve tahminlere olay mantığının uygulanması ya da belirli bir zaman diliminde bir olayın olup olmayacağının sınıflandırılması yoluyla tahmin edilir. Bu olaylar genellikle nadirdir ve bu da onları tahmin etmeyi zorlaştırır. Bu, tüm olayları yakalamak (doğru tahmin edilen olayların toplam olay sayısına oranı) ile sadece doğru olayları yakalamak (doğru tahmin edilen olayların bütün gerçekleşen olaylara oranı) arasında, yani hatırlama/hassasiyet ve kesinlik arasında, ödünleşmeye neden olur. Karar destek sistemi tüm olayları yakalar, ancak daha fazlasını yanlış tahmin ederse, operatörün güveni azalacaktır. Bu nedenle, makul hatırlama değerlerini tutturarak kesinlik değerlerini arttırmaya odaklanıyoruz. Bu yüzden, sonuçlar $F_{0.1}$ puanını kullanarak ve ikincil olarak en az 0,5 kesinlik ve 0,1 hatırlama elde edilmesine bakılarak değerlendirilmektedir. Önerilen yaklaşım, gerçek bir rafinerinin dört yatağında birden fazla yapay öğrenme modeli ile değerlendirilmiştir. İlk değerlendirme olay

tanımlarını doğrudan uygulayarak tahminlemek üzerinedir. Sonuçlar, her yatak için yüksek $F_{0.1}$ puanları elde eden en az bir yöntem bulunduğunu ve böylece karar destek sistemi oluşturmanın uygulanabilir olduğunu göstermektedir. Ancak yapay öğrenme modelleri arasında belirgin bir kazanan yoktur ve farklı yataklar için farklı performanslar elde edilmektedir. Ayrı ayrı sensörler yerine yataklar için olayları tahmin etmek önemli bir kazanım sağlamamıştır. Ayrıca, operatörlerin daha fazla esnekliğe sahip olması adına hatırlama ve kesinlik ödünleşimini ayarlamak için uyarlanabilir bir eşikleme yöntemi önerilmektedir. Bu, tahmin hatalarının güncel standart sapmasına dayanmaktadır. Değerlendirmeler, ödünleşimin mümkün olduğunu ve hatırlama değerlerini azaltsa da $F_{0.1}$ performansını önemli ölçüde artırmak için kullanılabileceğini göstermektedir. Son değerlendirme için, doğrudan sıcaklık tahmini yerine sıcaklık farkı tahmini denenmiştir. Yöntemlerin alt-kümesine uygulanan bu fikir, performans arttırmıştır ve daha üzerine gidilebilir olduğunu göstermiştir.

# ACKNOWLEDGMENTS

First, I would like to express my profound gratitude to my advisor Prof. Barış Akgün who supported me through this research project with his enthusiasm, encouragement and his brilliant ideas. Also, I would like to sincerely thank the members of my jury committee as they generously gave their time to us, and their invaluable comments illuminated the future work plans for this study. Finally, I gratefully acknowledge TÜPRAŞ for their support.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| ML | Machine Learning |
| DL | Deep Learning |
| LSTM | Long Short Term Memory |
| VAR | Vector Auto Regression |
| LR | Linear Regression |
| SGD | Stochastic Gradient Descent |
| XGB | eXtreme Gradient Boosting |
| XGBR | eXtreme Gradient Boosting Regression |
| SVM | Support Vector Machine |
| GR | Ground Truth |
| TP | True Positive |
| FP | False Positive |
| TN | True Negative |
| FN | False Negative |
| ADF | Augmented Dicky-Fuller |
| KPSS | Kwiatkowski-Phillips- Schmidt-Shin |
| ACF | Auto correlation function |
| PACF | Partial auto correlation function |
| CV | Cross-Validation |

# Chapter 1

# INTRODUCTION

## *1.1 Motivation, Scope and Goal*

Refineries consist of many pieces equipment facilitating chemical processes that transform and refine crude oil into useful products. They are large scale and complex, require immense safety measures and their business side is extremely competitive. Any unplanned interruption, aka an operational upset, may lead to lower quality product, process shutdowns or equipment failures. These can potentially lead to monetary losses, injury and even death. To ensure operational and safety criteria are met, these processes are automatically controlled and monitored. Due to the stakes, human operators reside over them as well who intervene when needed.

Automatic monitoring raises alarms if a predefined event happens such as abrupt temperature rise in a unit. This is to make sure that the any issues are caught before they result in more significant problems. Operators handle the initial response to such alarms. However, refineries want to minimize the number of these alarms. Human operators sometimes realize that a unit might be heading to an alarm state and get involved beforehand. For example if a unit is potentially overheating, even within constraints, the operator may increases the coolant flow to prevent an alarm. This is a difficult task. Predicting a potential alarm condition requires experience. More importantly, operators are usually responsible for multiple units and each unit has many sensors and control signals to monitor. This puts a cognitive burden on even the most seasoned operators.

In this paper, we introduce a data-driven methodology for decision aid design to reduce the process monitoring burden of the operators. The idea is to predict potential future events and raise *soft-alarms* accordingly to notify the operator who

then decides on the next course of action. The events can be defined by Boolean conditions on the measurements. They should be process specific but more generic anomaly conditions may also be used. The predictions are done by machine learning methods learned from historical data. We treat the sensor measurements and potentially other signals coming from a process unit/sub-unit as multi-dimensional time-series. We then use machine learning models to either predict future values and apply the event logic or to directly classify whether there will be an event in the future or not. After the validation phase, the trained models are deployed. The time-series is processed online and when an event is predicted, the operator is notified[1]. We want to note that it is very difficult, if not impossible, to come up with hand-crafted rules for event prediction due to dimensionality and amount of available data.

These events are rare, on the order of one in one thousand, and it is difficult to predict them. As such, the decision aid will be prone to mistakes. There are two types of mistakes; (1) wrongly predicting an event when there was not going to be any and (2) failing to predict an event which later happened. Unfortunately, it is difficult to fix both mistakes at the same time, resulting in a trade-off. Catching all the events means making more predictions overall which ultimately result in wrong predictions (or false positives). Only making correct predictions mean making less predictions overall which ultimately leads to missing events (or false negatives). The exact trade-off should be left to the operator. As such, any decision aid should allow for tuning it. However, we argue that the first type of mistake is more severe since the operator may start to ignore the raised alarms due to many mis-predictions, and if the aid is making fewer but correct predictions, it will be better accepted and used. We evaluate our approach on four beds of a hybdrocracker unit of a refinery. We treat each bed separately. We train and test multiple machine learning models to predict events using historical data and defined events. Furthermore, we proposed an approach to fine-tune the aforementioned trade-off for the decision aid system performance.

[1]Due to operational constraints, we were only able to perform offline evaluations for this work.

## *1.2 Thesis Statement*

Designing operator process monitoring decision aids that raise soft-alarms for near-term rare events is feasible using a data driven methodology by learning both forecasting and classification models from historical data and applying adaptive thresholding. The feasibility is based on the application domain and implies a certain level of performance and the ability to achieve a desired trade-off between recall and precision.

## *1.3 Outline*

The rest of this thesis is organized as follows. In chapter 2, we review the relevant state of the work. This chapter is followed by introducing our main approaches to tackle the main goal of this study. Chapter 4, describes our case of study, data processing steps as well as data explanatory analysis details. Then we present our initial evaluation and discussion of our methodologies toward the event prediction goal in chapter 5. Also, chapter 6 is allocated to introducing and evaluating our new proposed approach to the end of fine-tuning the decision aid performance. We conclude the report by discussing the methods used and the conducted experimental results in chapter 7.

# Chapter 2

# RELATED WORKS

The literature on early warning systems and associated decision aids is sparse, especially for refinery applications. However, fault diagnosis and prognostics from a predictive maintenance perspective is studied more widely in many fields (refinery applications still comparatively lacking). Predictive maintenance methods aim to predict the failures or breakdowns in advance so that they can be avoided. We first provide a representative body of work from a decision aid perspective. We then present literature from the more general predictive maintenance domain. All the papers reviewed in this section are summarized in Table 2.1.

Garcia et al. (2010) present the Sensor Data Analysis for Equipment Monitoring (SDAEM) model to assist plant operators in predicting failures in advance [6]. They extract sequential patterns that lead to undesired events and the current window of time is compared to these to find any early sign of failures. SDAEM uses one dimensional time series data to predict coarser failures and which are not rare. Saybani et al. (2011) use classical linear time-series forecasting methods for predicting sensor measurements in advance in an oil refinery [13]. They propose this as a decision aid to the operators when a sensor becomes faulty; to use the output of their learned models instead of the sensor's. This study focuses on one dimensional time series and does not involve near-future event detection. Kuzin et al. (2016) employ different approaches to deal with early detection of sensor failures [11]. These include classification of sensor health and anomaly detection-based methods and are evaluated on accelerometer data. This approach is geared towards one dimensional time series and is about sensor health, not near-future rare events. Jiang et al. (2017) extract features within a window of data and use these features with classifiers to predict alarms [9]. This is similar to our classification based approach however it uses one dimensional time series and the class imbalance is not an issue.

Table 2.1: Summary of the selected related work, including the application, data-type, target output, machine learning approach (regression/forecast vs classification) and whether there is class-imbalance or not.

| Reference | Application | Data Type | Aim | Classification\Regression | Class Imbalance |
|---|---|---|---|---|---|
| [13] | Oil refinery | One-dimensional time series | Decision aid | Regression | No |
| [12] | Wood refinery | One-dimensional time series | Predictive maintenance based on failure detection | Regression | No |
| [2] | Oil refinery | Multi-dimensional Time series | Predictive maintenance | - | No |
| [11] | Accelerometers | One-dimensional time series | Sensor failure prediction / Predictive maintenance | Both | No |
| [5] | Gasoil plant | Multi-dimensional time series | Fault detection | Regression | No |
| [16] | Rolling bearing | Multi-dimensional time series | Fault diagnosis | Classification | No |
| [6] | Oil plant | One-dimensional time series | Operator assistance in failure prediction | - | No |
| [10] | Oil refinery | Multi-dimensional time series | Gross error classification, gross error detection | Classification | No |
| [7] | PC monitoring data | Multi-dimensional time series | Failure prediction | Classification | Yes (manually balanced) |
| [17] | Chemical process data | Multi-dimensional time series | Fault diagnosis | Classification | No |
| [9] | Refinery | One-dimensional time series | Early warning | Classification | No |
| [15] | Urban Sewer | One-dimensional time series | Sensor failure detection | Time series prediction | No |
| [14] | Oil refinery | Multi-dimensional time series | Prediction and diagnosis of machinery breakdowns | Classification | No |
| [4] | Pipeline leakage | Multi-dimensional time series | Fault diagnosis | Classification | No |
| [8] | Actuator of aircraft | Multi-dimensional time series | fault diagnosis | Classification | Yes |
| [1] | Rolling Bearing application (review paper) | Multi-dimensional time series | Fault diagnosis | Classification | No |
| This Thesis | Oil refinery | Multi-dimensional time series | Decision Aid / Event-detection | Both | Yes |

Failure prediction methods employ both classification and forecasting ideas. Some methods extract features from a time window and use off-the-shelf classifiers. Irrera et al. (2013) employ support vector machine (SVM) models to predict the failure events occurring in a Windows XP machine and investigate the effect of window size. They handle the class imbalance issue by using fault injection software [7] to eliminate the imbalance. Faysal et al. (2021) also utilize SVMs to tackle the leak diagnosis of a pipeline [4]. Steurtewagen et al.(2021) use XGB classifier to predict machinery machinery breakdowns in an oil refinery [14]. These feature extraction and classification approaches can handle multi-dimensional time-series data relatively easily and utilize familiar models but features need to be selected carefully and class imbalance must be handled. Moreover, some of them are not about predicting current failures, as opposed to near-term ones.

Some other methods are based on time-series forecasting and comparing the forecast to the measured value, for the same step to detect failures (anomaly detection perspective). Thiyagarajan et al. (2017) propose a forecasting solution for failure detection of a temperature sensor of an urban sewer using time-series models [15]. Ramos et al. (2014) present study for predicting equipment malfunctioning and failures [12]. They employ neural networks along with classical time-series models. The forecast of these models is used to detect the sensor failures. Filonov et al. (2016) adopt an approach based on Long Short Term Memory (LSTM) neural network to forecast gasoil plant measurements to detect faults [5]. Depending on the model, these approaches may be able to handle multi-variate data. Some of the models and the idea of comparing forecasts with measurements is similar to ours. However, these methods are geared towards detecting something that already happened whereas we are predicting near-term events.

There are also classification methods that directly take time-series as input. ZHAO et al. (2018) propose a fault diagnosis classifier, for a chemical process, based on Batch Normalized LSTM neural networks [17]. Yu et al. (2019) propose a method based on a stacked LSTM model to diagnose the failures of rolling bearings [16]. A novel fault diagnosis method for aircraft actuator based on an ensemble model is proposed by Jia et al.(2021) [8]. The proposed method is a deep network

constructed by multi-layer extreme learning machine based auto-encoder which was successful in dealing with imbalanced data. Khodabakhsh et al. (2018) study fault detection and prediction for an oil refinery application [10]. They combine multiple methods including machine learning and Kalman Filters. These methods are about detecting failures and not about event prediction. Antomarioni et al. (2018) propose an association rules mining for the purpose of developing a data-driven maintenance policy for an oil refinery application [2]. One of their main goals is to find which components are likely to break down within a given time interval. This is similar to our goals but only relevant for maintenance which require additional break-down data and is limited in its application.

# Chapter 3

# APPROACH

The objective of this study is to present a data-driven methodology for raising soft alarms to warn the operators about upcoming undesired events in a process. The idea is to use current data, event definitions, historical data and machine learning models towards this end. Even though the described methodology can be applied elsewhere, we concentrate on a refinery application.

The data can include sensor readings (e.g. temperature), computed values (e.g. highest measured temperature in the last 5 minutes), control variables (e.g. coolant flow set point) etc. These change with time and their temporal behavior is important in predicting the events. Thus, the data we deal with is multi-dimensional time series. In this paper, we mainly concentrate on temperature sensors of a hydrocracker unit of a refinery.

The events we deal with in this paper are about the near future so that the operator can take actions to alleviate them. Predicting something current or in the past is out of scope and has been widely studied. Detecting something far into the future is prone to errors and involve tremendous uncertainty. In this paper, we look at 15 minutes into the future using data with 1 minute resolution.

The events are defined as conditions dependent on the data. Our prototypical example is whether difference between the temperature measurements now and 15 minutes into the future is above 5 degrees or not. Wider time ranges, such as temperature rise within 5 to 15 minutes, absolute conditions, such as temperature rise above a certain value in 15 minutes (which we also utilize), generic anomaly conditions, such as sensor measurements going outside the three standard deviations in 5 minutes, more involved metrics, such as high frequency components getting larger or their combinations can all be used as events.

We adopt two machine-learning approaches to tackle the event prediction task;

(1) a regression based approach and (2) a classification based approach. In the regression, or more aptly forecasting, based approach, we predict the future data values using machine learning methods and apply the event logic on these forecasts and already measured data if required by the event. In the classification based approach, we directly predict whether there will be an event or not. An implicit assumption we make is that recent measurements is enough to predict the near future events. In describing our methodology, we adopt the following notation:

$t$ : Current time step

$d$ : Dimensionality of the data

$x_i(t)$ : The $i^{th}$ data dimension at time $t$ where $i = 1 \dots d$,
e.g., a specific sensor measurement

$x(t)$ : The $d$ dimensional vector of data at time $t$

$y(t)$ : Event status at time $t$
Encapsulates all the events
$x(t) = [x_1(t), \dots, x_d(t)]^T$

$X^w(t)$ : The window of data from $t - w + 1$ to $t$;
$X^w(t) = [x(t - w + 1), \dots, x(t - 1), x(t)]$,
represented as a $d \cdot w \times 1$ vector or a $d \times w$ matrix

$p$ : Time horizon to predict

$\Delta^p x(t)$ : Change between $t$ and $t + p$,
$\Delta^p x(t) = x(t) - x(t + p)$

$\hat{x}(t + p)$ : Predicted data at time $t + p$

$\hat{y}(t + p)$ : Predicted event status at time $t + p$

$\Delta^p \hat{x}(t)$ : Predicted change between $t$ and $t + p$

## *3.1 Regression/Forecasting Based Approach*

In the regression based approach, future data values are predicted from a window of recent measurements. A one-dimensional example of the input time window, target

time step and the data is given in Figure 3.1. The time series model we assume is:

$$x(t+p) = f(X^w(t)) + \epsilon \tag{3.1}$$

where $f(\cdot)$ is the potentially non-linear time series model and $\epsilon \sim \mathcal{N}(0, \Sigma)$ is a zero-mean, diagonal covariance Gaussian noise. The window size $w$ is data dependent (see Section 4.3.2) but the time horizon, $p$, comes from the event definition (and more than one horizon may be required by the event). We approximate $f(\cdot)$ with a function approximator using historical data. The generic form is

$$\hat{x}(t+p) = h(X^w(t), \theta) \tag{3.2}$$

where $h(\cdot)$ is a parametric function approximator (e.g. a time series model, neural network, etc.) and $\theta$ is its parameters to be calculated from historical data by minimizing the squared Euclidean norm between $x(t+p)$ and $\hat{x}(t+p)$ while watching out for overfitting.

Event logic is applied on the forecasts $\hat{x}(t+p)$, obtained using $h(\cdot)$, at each time step $t$ to predict events. For example, we check if $|x_i(t) - \hat{x}_i(t+p)| > \tau$ holds true to detect abnormal rise, quantified by $\tau$, in the $i^{th}$ data dimension in $p$ time-steps.

An alternative way is to directly predict the change of data values within the time horizon. In that case, the model we assume, along with the associated approximation, is similar:

$$\Delta^p x(t+p) = g(\Delta^p X^w(t)) + \epsilon \tag{3.3}$$

In this work, we utilize autoregressive linear regression (LR) per data dimension, Vector Autoregression (VAR), XGBoost Regression (XGBR, [3]) and LSTM based neural network models for the time-series forecasting task, i.e., as the $h(\cdot)$ function.

Due to the time-series nature of the data, the past values are used to predict the near-future. When the models are deployed, they get more data as time passes. This newly acquired data can be utilized to re-train the models. However, this needs to be fast since the models are running. As a result, we opt-for fine-tuning rather than fully re-training. LR and VAR models can be and LSTM models are trained with gradient descent algorithms which are easy to adapt for fine-tuning by only

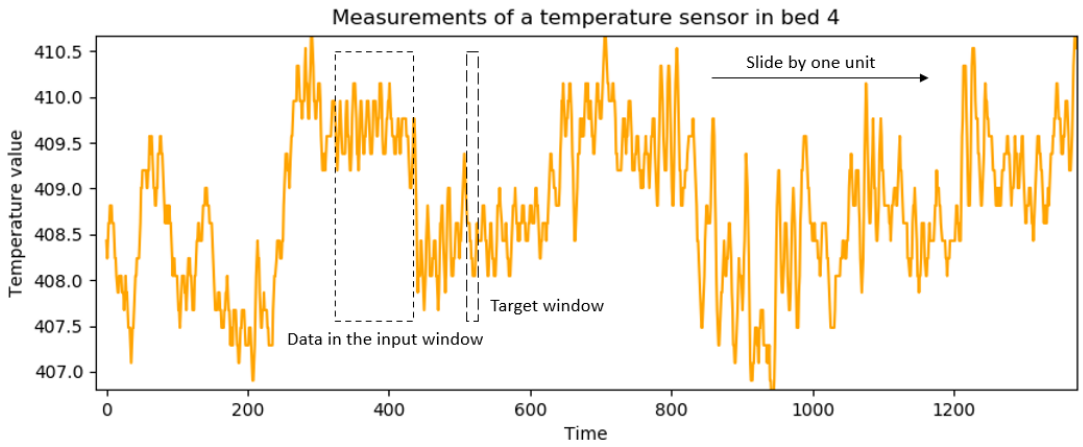



Figure 3.1: A depiction of the sliding window method to prepare input data for machine learning and deep learning models.

applying a small number of passes over the model with recently acquired data, at least on paper. We perform fine-tuning for LR and LSTM models. Due to practical issues, we perform full re-training on the VAR models. The details are given later.

## *3.2 Classification Based Approach*

In the classification based approach, the future events are directly predicted from a window of recent measurements. The generic machine learning form is:

$$\hat{y}(t+p) = h(X^w(t), \theta) \tag{3.4}$$

The predicted event status, $\hat{y}(t+p)$, can have multiple forms (for each data dimension); (1) binary for each data dimension where 0 means no event and 1 means any event, (2) categorical/integer where 0 means no event and other integers correspond to specific events, and (3) a one-hot vector where each event (including no event) has a dedicated dimension that is 1, all the others being 0. The first one makes it a binary classification and the others makes this a multi-class classification problem. In this work, we opt for binary classification[1]. Note that the regression-based approach does not need to make this distinction.

[1]A single model can be trained for each dimension or it can be cast as a multi-label classification problem, depending on the machine learning model

The learning targets, aka real events, $y(t+p)$, are obtained by applying the event logic on the historical data (e.g. $y(t+p) = |x_i(t) - x_i(t+p)| > \tau$, assuming $>$ returns 0 or 1). As mentioned in Section 1, the events are rare, and falsely predicting an event and failing to predict an event have different costs. This can be included in training by increasing the weight of the minority class (in our case the events). As such, the models are trained with historical data by minimizing the class-weighted cross-entropy between $y(t+p)$ and $\hat{t}(t+p)$ while watching out for overfitting.

In this work, we only present the results of XGB Classifier with class-weighting. Support vector machines (SVMs), LSTM-based and Convolutions neural networks performed very poorly[2]. We did not perform any re-training but it can be done if an appropriate classification method is used.

### *3.3 Programming Environment and Implementation Details*

Python programming language [3] is used to code and implement the machine learning and deep learning models and experiment the phases of our project. Sklearn and Keras libraries are used to implement machine learning and deep learning models, respectively. Sklearn is an open source library which features various machine learning algorithms. Keras is an open source library which provides a high level API to implement deep neural networks (DNN). Keras supports multiple back ends including TensorFlow and Theano. In this study, we used the Keras library which runs on top of TensorFlow. All the experiments of this study, are performed on the Koç university servers. The employed sever has Intel processor with Linux operating system. The machine have 40 CPUs (32-bit, 64-bit) and 4 GPUs. Our machine learning models are run on 64 bit CPUs. However, the deep learning models are implemented on an NVIDIA GeForce GTX 11 GB-GPU workstation with 11 GB of memory.

[2]Different architectures and hyper-parameters may have performed better but we did not want to spend time on this given a working alternative.

[3]python version 3.5

# Chapter 4

# REFINERY APPLICATION

As mentioned in Section 1, we apply our approach on four beds of a hydrocracker unit of a refinery. In this section, we provide details of this application.

## *4.1 Data Description*

The studied data is obtained from temperature sensor measurements of a hydrocraker unit of an oil refinery. The hydrocracker unit has four beds. Bed1 has 20, Bed2 has 22, Bed3 has 23, and Bed4 has 24 temperature sensors for a total of 89. All the sensor measurements span almost an eleven-month period from 01-03-2018 to 23-01-2019.

## *4.2 Data Pre-processing*

There maybe duplicate values, missing values and obvious outliers (e.g. a single mis-measurement) in the data. In addition, the data may not be sampled with the same frequency or even if they are, timestamps may not be synchronized. As a result, the data is pre-processed before analysis and learning.

For pre-processing, the duplicates are removed and outliers are clipped[1]. In addition, the data is re-sampled such that the interval between each value is one minute and time-stamps are synchronized. More frequently sampled observations were down sampled by averaging, and the less frequently sampled observations were up sampled by interpolation. Note that sampling takes care of missing values as well. If the missing observations are at the beginning/end of the time series, they were replaced by the first/last recorded observation. At the end of the pre-processing step, we have 473061 synchronized temperature values per sensor.

[1]There were only a handful 0 temperature readings which were clipped to 350°.

## *4.3 Exploratory Data Analysis*

### *4.3.1 Stationarity*

Informally, stationarity means that the properties of the time series do not change with the time of observation. It is important to know whether a time-series is stationary or not for pre-processing (e.g. some time series can be made stationary with manipulation), modelling (e.g. to include seasonal information or not) and training approach (e.g. non-stationary time-series require constant re-training).

We used Augmented Dicky-Fuller (ADF) statistical test for the existence of a unit root[2] and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) statistical test for stationarity. The ADF test results show that our data does not have a unit root, with high confidence, implying that our data is stationary or trend-stationary (rejecting the null hypothesis). KPSS test did not conclude that the data was non-stationary (failed to reject the null hypothesis). Combining these results, we conclude that the data is stationary. Note that this analysis was done on each data dimension.

### *4.3.2 Correlation and Lag Analysis*

The correlation between sensor measurements of the same time step are extremely high, all above 0.98, for sensors in a given bed. This is not surprising due to the fact that sensors are located next to each other. However, we are using past data to predict the future so we need to look at the correlation between these time steps which is called lag analysis.

For the lag analysis, we look at the Auto correlation function (ACF) and Partial auto correlation function (PACF) plots. The ACF is a measure of the correlation between the observation at the current time step and the observations at previous time steps while PACF directly looks at the relationship between an observation and its lag controlled for other time-steps in between (e.g. in PACF the correlation between $t$ and $t-2$ is calculated by removing the effect of $t-1$, whereas in ACF this is not the case).

[2]Stochastic processes with a unit root are non-stationary

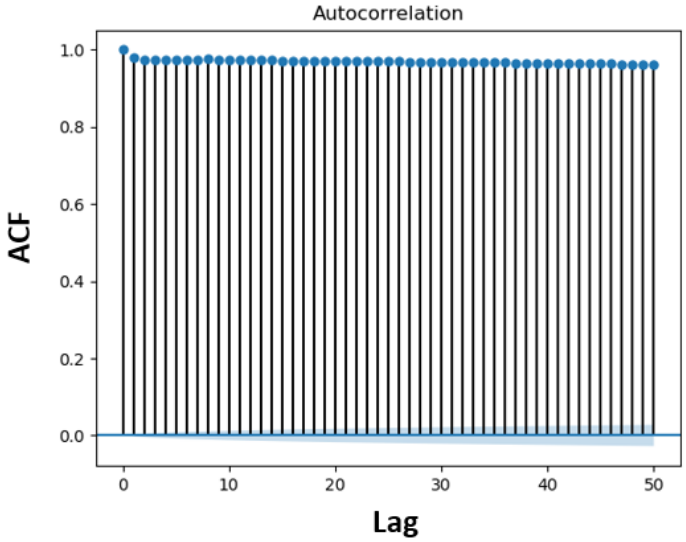
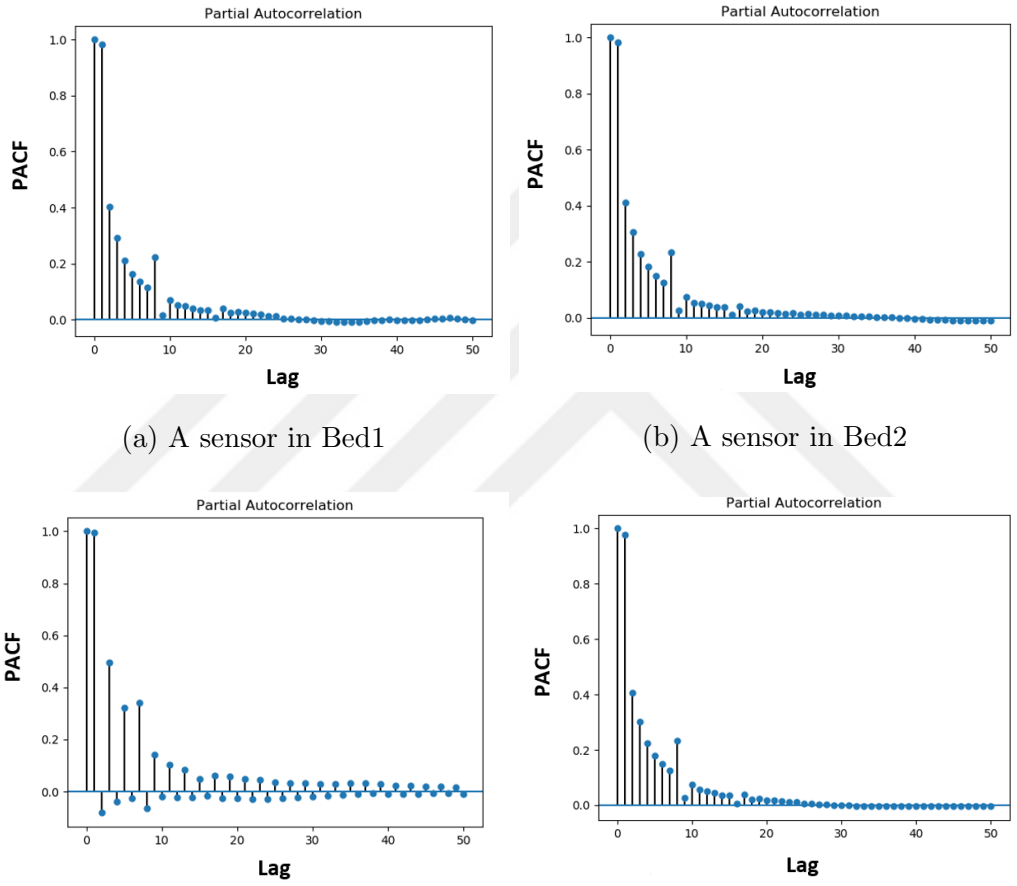



Figure 4.1: The auto-correlation plot related to measurements of a temperature sensor located in Bed4.

Our analysis of all the sensors yielded similar results as follows. The ACF values are significant for a large number of lags, as Figure 4.1 shows for a single temperature sensor of Bed4. However, the PACF decay down to zero after a certain time. Example figures from each bed are given in Figure 4.2. We found that the lag 25 is about the cut-off where the PACF values gets sufficiently close to zero for most of the sensors. As such, we are going to use this value as our look-back window size, $w$, for inputs to our models (i.e. the $w$ parameter in Equation 3.2).

This is a one-dimensional linear analysis and as such may miss certain temporal and spatial (between sensors) interactions. However, our empirical results suggest that a window size of 25 is adequate considering training duration and inference performance.




(a) A sensor in Bed1

(b) A sensor in Bed2

(c) A sensor in Bed3

(d) A sensor in Bed4 (the one in Fig. 4.1)

Figure 4.2: Example PACF plots of temperature sensors for each bed.

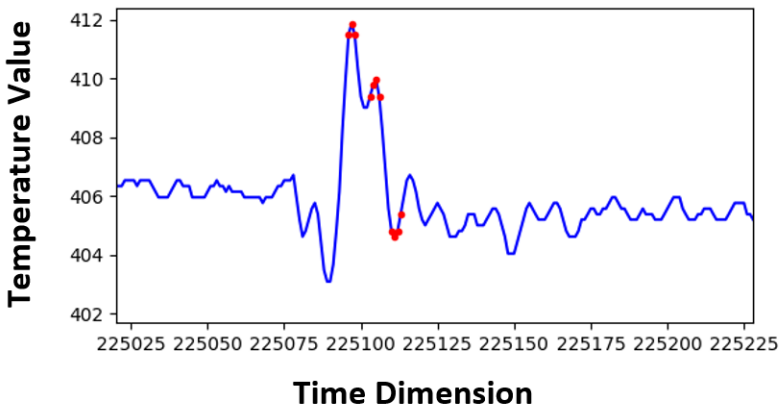



Figure 4.3: Visualizing example events

### *4.4 Event Definitions*

We utilize two events for the refinery application, where the superscript denotes the event number:

1. Temperature change of 5° in 15 minutes.

   Event condition is $y_i^1(t+15) = |x_i(t) - \hat{x}_i(t+15)| > 5$, where $i$ corresponds to a specific temperature sensor

2. Measurements over 440° in 15 minutes.

   Event condition is $y_i^2(t+15) = \hat{x}_i(t+15) > 440$, where $i$ corresponds to a specific temperature sensor

Example events of the first type for a single sensor are given in Figure 4.3.

These events are rare. Table 4.1 shows the average number of events per sensor for our data along with the ratio of events to all the data. Furthermore the events are not uniformly distributed in time. Figures 4.4-4.7 show the event timings for the each sensor and bed. These figures also show that the events for sensors mostly happen together for a bed, and to an extent between beds.

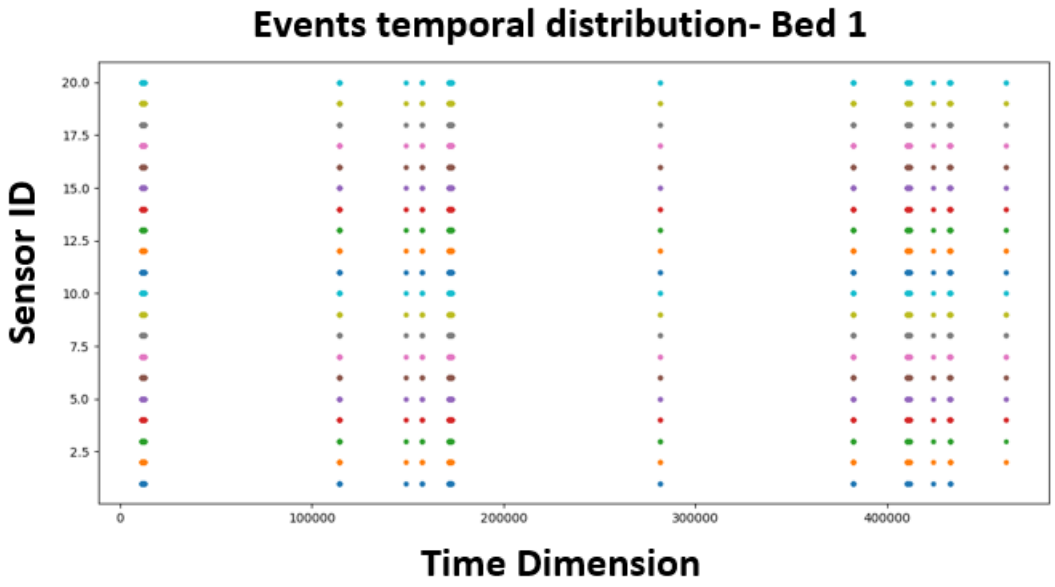



Figure 4.4: Bed 1 event distribution in time

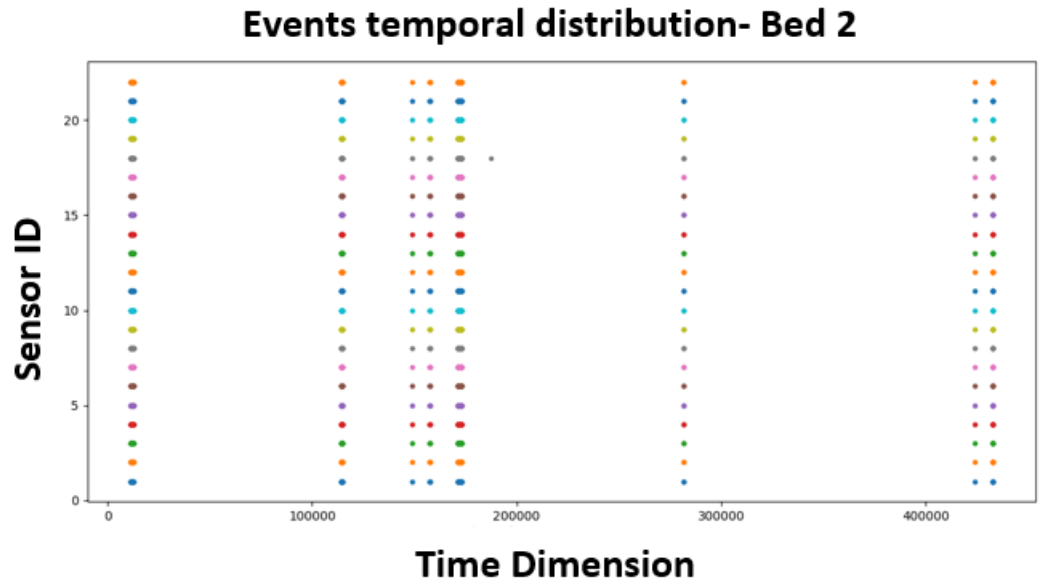



Figure 4.5: Bed 2 event distribution in time

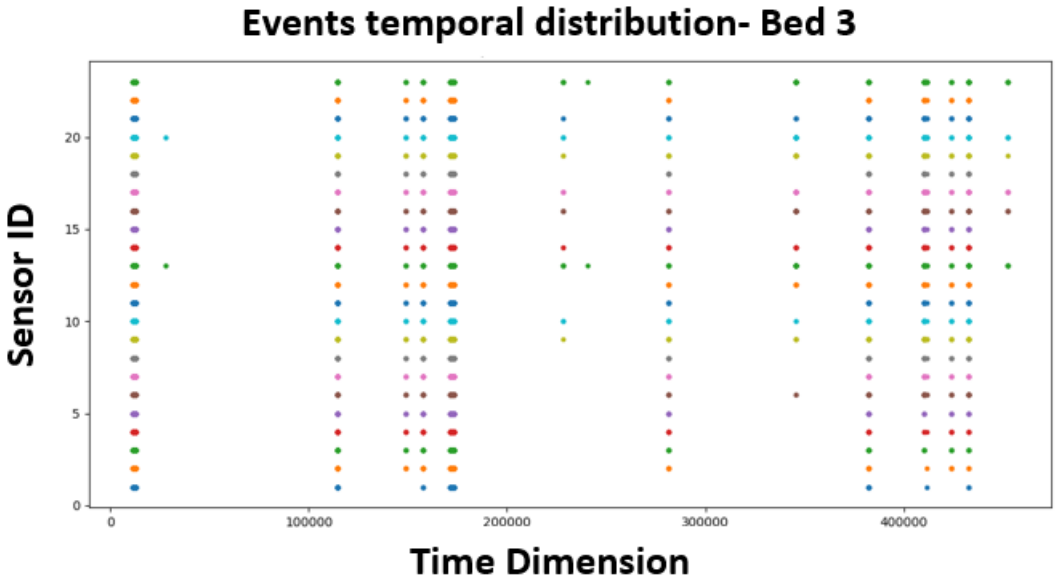



Figure 4.6: Bed 3 event distribution in time

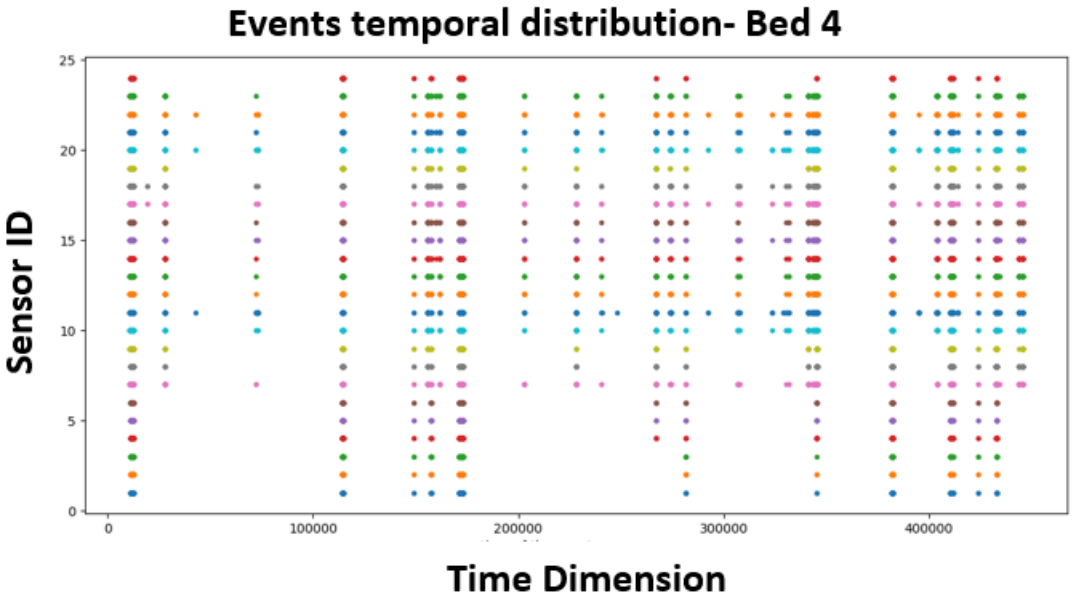



Figure 4.7: Bed 4 event distribution in time

Table 4.1: The number of data points, average number and ratio events of events per temperature sensor, and the total number of sensors for each bed.

| Bed | # of Data Points | Avg. # of Events | Avg. Ratio of Events | # of Sensors |
|---|---|---|---|---|
| 1 | 473061 | 921 | 0.19% | 20 |
| 2 | 473061 | 930 | 0.19% | 22 |
| 3 | 473061 | 1091 | 0.23% | 23 |
| 4 | 473061 | 1367 | 0.28% | 24 |

### *4.5 Performance Criteria and Trade-Off*

In our work, the goal of the decision aid is to predict events so that the operator can be warned. This way, the cognitive load and the workload of the operator can be reduced. However, there are two types of errors that the decision aid can make which may defeat the purpose; (1) falsely predicting that there will be an event (aka false positive) and (2) not predicting an event that happened (aka false negative). It is not possible to minimize both errors at the same time with a given model, especially in the context of rare events. To catch all the events, the system would need to make more positive predictions which leads to false positives. On the other hand the system would need to make less positive predictions to avoid this, leading to missing capturing some of the events.

We argue that the cost of these two errors are not the same and that wrong positive predictions are more costly. In this case, the operator would be interrupted more often and at some point would lose trust in the system, completely abandoning it. Missing some events are less severe since it does not affect the work-flow of the operator. However, certain amount of events need to be caught to make the decision aid worthwhile. Thus a trade-off needs to be made between making correct positive predictions while catching as many events as possible.

We use precision and recall values to analyze this trade-off. Precision is the ratio of correct positive predictions to all the predictions. A high precision implies that the system's positive predictions were mainly correct. Recall is the ratio of correct

positive predictions to all the positive events. A high recall implies that majority of the events are caught. Based on our cost discussion, we prefer higher precision values while keeping reasonable recall.

We pick our final performance criteria[3] as the $F_\beta$ score considering the severe imbalance (see Table 4.1) and the cost difference. $F_\beta$ score is a weighted combination of precision and recall. For example $F_1$ score gives equal importance to precision and recall and $\beta$ values less than 1 put more weight on the precision score. The relative importance, $\beta$, needs to come from the domain experts or users of the models. Emphasizing the importance of precision over recall, we pick $\beta = 0.1$.

Let $p$ correspond to precision and $r$ correspond to recall. Then, $F_\beta$ score can be calculated as follows :

$$F_\beta = (1 + \beta^2)\frac{p \cdot r}{(\beta^2 \cdot p) + r} \tag{4.1}$$

See Appendix A.1 for more details on the recall, precision and F-score definitions.

[3]We are not concerned about the regression/forecasting performance directly as all the models on average are very successful. However, due to the rare nature of the events, this is not sufficient for our purposes. Thus we concentrate on the event related criteria.

# Chapter 5

# VANILLA EVENT PREDICTION

In this section, we directly apply the event logic using several machine learning models. We start by giving the model information, our evaluation steps and finally results.

## *5.1 Model information*

We utilize the following six models:

- Linear Regression
- Linear Regression with Stochastic Gradient Descent (SGD) Re-Training (Fine Tuning)
- Fully Re-Trained Vector Auto Regression (VAR)
- Stacked LSTM based Neural Network with Re-Training (Fine Tuning)
- XGBoost Regression
- XGBoost Classification

There are hyper-parameters due to the model and the problem. The common hyper-parameter windows size is chosen as $w = 25$ as described in Section 4.3.2.

**Linear Regression Based Models:** We use the basic linear regression model with a intercept term and do not utilize any regularization. As a result, there are no hyper-parameters. For the fine-tuned version, we have several hyper-parameters; fine-tuning frequency is 25, training window size is 25, number of passes (aka epochs) is 1 and the learning rate is 0.00001. These were chosen empirically. There maybe

better parameters but we opted to spend effort elsewhere since these did not result in overfitting.

**Vector AutoRegression Model:** Given that $w = 25$, there are no hyper-parameters for this model other than the re-training related ones. Recall that we opted to fully re-train this models instead of fine-tuning. This decision was made due to practical and computational issues[1]. The only re-training hyper-parameter, re-training frequency, is selected as 25.

**Stacked LSTM:** Our Stacked-LSTM model has two LSTM layers with 32 and 16 number of hidden units respectively. Both layers have rectified linear activation functions. These are followed by a fully connected layer with output size same as the data dimensionality (i.e. the number of temperature sensors in the corresponding bed). We use the ADAM-optimizer to train and re-train this model. The re-training hyper-parameters are chosen empirically; fine-tuning frequency is 30, training window size is 100 and number of passes (aka epochs) is 1.

**XGBoost Regressor:** There are many hyper-parameters of this model most of which we leave in their default values. We only pick the number of estimators as 200 and the learning rate as 0.07. These numbers were chosen with a coarse grid search to limit overfitting.

**XGBoost Classifier:** Similar to the XGB Regressor,we only chose the number of estimators as 30 and the learning rate as 0.03. We chose these parameters with a grid search to limit overfitting. On top this, we employed class-weighting to help with the severe class-imbalance issues.

### *5.2 Evaluation Setup*

For time-series, we cannot randomly split the data but have to take their sequential nature into account (i.e. we do not want to learn from future data and test it on the past data) to test their generalization performance. Towards this end, we use the expanding window cross validation approach. The data is divided into five folds

[1]We are using an off-the-shelf VAR library and we were not able to accommodate a SGD extension that would run in a reasonable amount of time.

while keeping their order intact. The previous folds are used as training and the next future fold is used as test. Then the test fold is added to the training fold and the process is repeated. The first fold is never in testing and the last fold is never in training. We end up with 4 sets of evaluation.

We evaluate our events on two scenarios; (1) per temperature sensor event prediction and (2) per bed event prediction. Our hypothesis is that since the events for each sensor tend to occur together (See Figures 4.4-4.7), per bed results will be better than per sensor results which will help the decision aid with the caveat that we would lose the fine-grained per sensor information.

In the first scenario, the prediction performance scores are calculated by pooling all the predictions together instead of taking average of individual sensors to calculate the recall, precision and F-scores. This was preferred since each sensor has different number of events. In the second scenario, the ground truth and the prediction events are "or-ed" together (if a single sensor has an event, then the bed has an event) to calculate the performance metrics.

### *5.3 Results and Discussion*

The performance metrics for the per sensor scenario is reported in Table 5.1 along with the recall, precision and $F_{0.1}$ scores in Figures 5.1, 5.2 and 5.3, respectively, for easier visualization. Similarly, the performance metrics for the per bed scenario is reported in Table 5.2 along with the recall, precision and $F_{0.1}$ scores in Figures 5.4, 5.5 and 5.6. We make the following observations based on these results:

**No single best model for all beds in the per sensor evaluation:** There is no clear winner for all the beds in terms of average precision and $F_{0.1}$ scores among the models.

XGB Classifier performs the best for Bed 1 and Bed 2 in terms of precision and $F_{0.1}$ score. XGB Regressor is the best performing model for Bed 3 with linear regression not so far behind. Linear regression based models have an edge for Bed 4 with the basic version (not re-trained) doing better. Stacked LSTM model performs best for each bed in terms of recall at the expense of precision.

Table 5.1: Average per sensor event prediction performance for all beds. The number in parentheses are standard deviation.

| Model | Bed 1 | | | Bed 2 | | | Bed 3 | | | Bed 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **$F_{0.1}$ score** | **Precision** | **Recall** | **$F_{0.1}$ score** | **Precision** | **Recall** | **$F_{0.1}$ score** | **Precision** | **Recall** | **$F_{0.1}$ score** |
| **Stacked LSTM model** | 0.08 (±0.07) | 0.47 (±0.24) | 0.08 (±0.02) | 0.12 (±0.06) | 0.61 (±0.08) | 0.12 (±0.07) | 0.35 (±0.38) | 0.35 (±0.23) | 0.27 (±0.22) | 0.26 (±0.25) | 0.35 (±0.19) | 0.26 (±0.23) |
| **XGB regressor** | 0.08 (±0.11) | 0.29 (±0.2) | 0.08 (±0.04) | 0.04 (±0.04) | 0.29 (±0.17) | 0.04 (±0.02) | **0.56** (±0.14) | 0.23 (±0.16) | **0.54** (±0.14) | 0.29 (±0.24) | 0.20 (±0.08) | 0.28 (±0.22) |
| **SGD linear regression** | 0.28 (±0.16) | 0.23 (±0.2) | 0.27 (±0.06) | 0.29 (±0.23) | 0.27 (±0.18) | 0.29 (±0.08) | 0.21 (±0.14) | 0.15 (±0.16) | 0.21 (±0.09) | **0.56** (±0.29) | 0.07 (±0.06) | 0.45 (±0.17) |
| **Linear regression** | 0.30 (±0.27) | 0.40 (±0.26) | 0.30 (±0.09) | 0.30 (±0.25) | 0.43 (±0.11) | 0.30 (±0.09) | 0.48 (±0.29) | 0.33 (±0.26) | 0.48 (±0.24) | 0.48 (±0.21) | 0.28 (±0.19) | **0.47** (±0.17) |
| **VAR** | 0.24 (±0.43) | 0.38 (±0.16) | 0.24 (±0.07) | 0.24 (±0.2) | 0.35 (±0.13) | 0.23 (±0.07) | 0.25 (±0.13) | 0.26 (±0.21) | 0.25 (±0.1) | 0.37 (±0.17) | 0.29 (±0.12) | 0.37 (±0.15) |
| **XGB classifier** | **0.60** (±0.2) | 0.20 (±0.15) | **0.42** (±0.22) | **0.51** (±0.41) | 0.26 (±0.17) | **0.47** (±0.35) | 0.34 (±0.4) | 0.16 (±0.16) | 0.28 (±0.3) | 0.11 (±0.11) | 0.12 (±0.1) | 0.11 (±0.09) |

**Models are good enough for decision aid design:** Each bed has at least one model with over 0.5 precision (Bed 4 linear regression is close with 0.48) and the recall scores of the best models are larger than 0.2. We conclude that these models can be used to develop decision aids. However, note that a different model needs to be selected for a given bed.

**Per bed performance gains are not enough to drop the per sensor approach:** We expected that the per bed scenario to have a broader impact on the results. Even though there is performance gains apart from Bed 3, they are not enough to justify losing the per sensor granularity. On the high level, the observations are not too different than the per sensor case as well; we require different models for each bed and we can satisfy our decision aid conditions. As such, we continue with per sensor results.

**Lower complexity models perform better:** We expected the Stacked LSTM model to perform the best, given the deep learning performances on other problems along with the re-training idea. However, this was not the case. The simpler linear regression models performed better. Even the XGB Classifier was not too complex with 30 estimators. This suggests that the data does not require a high complexity model and that more complex models may have overfit the noise. Furthermore, re-training did not help but this was not a surprise since we found that our data was stationary (see Section 4.3.1). Note that the VAR model is also highly complex with many parameters (most among the methods) even if it is linear.

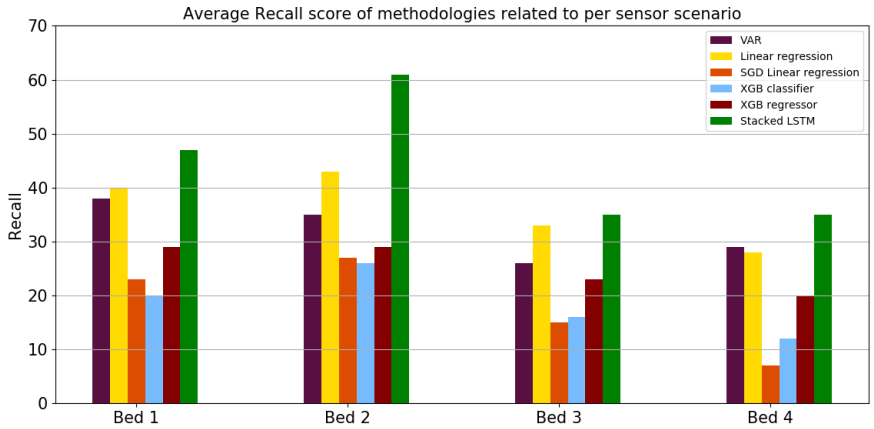



Figure 5.1: Average recall score of methodologies related to per sensor scenario

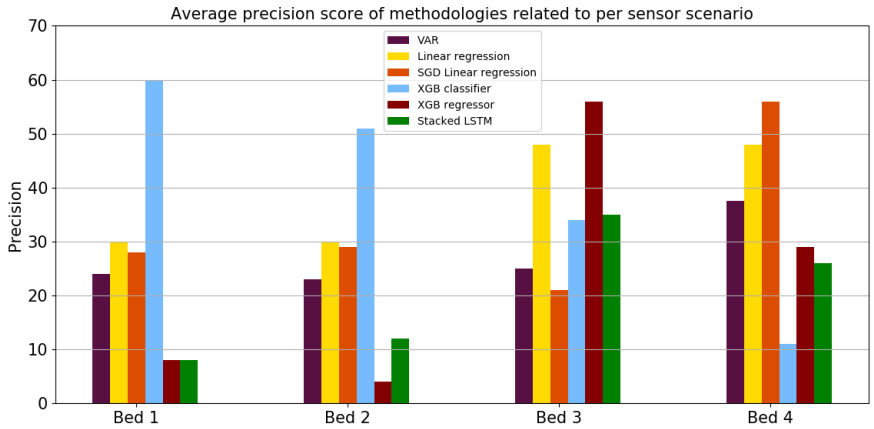



Figure 5.2: Average precision score of methodologies related to per sensor scenario

Table 5.2: Average per bed event prediction performance for all beds. The number in parentheses are standard deviation.

| Model | Bed 1 | | | Bed 2 | | | Bed 3 | | | Bed 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | $F_{0.1}$ score | Precision | Recall | $F_{0.1}$ score | Precision | Recall | $F_{0.1}$ score | Precision | Recall | $F_{0.1}$ score |
| **Stacked LSTM model** | 0.13 (±0.13) | 0.58 (±0.26) | 0.13 (±0.13) | 0.16 (±0.11) | 0.69 (±0.19) | 0.16 (±0.11) | 0.34 (±0.39) | 0.41 (±0.33) | 0.31 (±0.34) | 0.29 (±0.22) | 0.42 (±0.31) | 0.29 (±0.22) |
| **XGB regressor** | 0.04 (±0.05) | 0.36 (±0.21) | 0.03 (±0.05) | 0.02 (±0.02) | 0.41 (±0.19) | 0.02 (±0.02) | 0.30 (±0.1) | 0.31 (±0.22) | 0.29 (±0.1) | 0.22 (±0.19) | 0.35 (±0.15) | 0.22 (±0.19) |
| **SGD linear regression** | 0.35 (±0.2) | 0.34 (±0.23) | 0.33 (±0.21) | 0.39 (±0.23) | 0.41 (±0.16) | 0.40 (±0.23) | 0.34 (±0.27) | 0.20 (±0.2) | 0.34 (±0.27) | **0.66** (±0.26) | 0.14 (±0.16) | **0.58** (±0.2) |
| **Linear regression** | 0.33 (±0.29) | 0.41 (±0.27) | 0.33 (±0.29) | 0.27 (±0.17) | 0.48 (±0.21) | 0.27 (±0.17) | **0.53** (±0.33) | 0.35 (±0.28) | **0.53** (±0.33) | **0.59** (±0.22) | 0.32 (±0.22) | **0.59** (±0.22) |
| **VAR** | 0.28 (±0.22) | 0.43 (±0.14) | 0.28 (±0.21) | 0.32 (±0.22) | 0.50 (±0.2) | 0.32 (±0.22) | 0.38 (±0.35) | 0.33 (±0.28) | 0.38 (±0.35) | **0.50** (±0.18) | 0.37 (±0.18) | **0.50** (±0.18) |
| **XGB classifier** | **0.59** (±0.43) | 0.27 (±0.2) | **0.42** (±0.31) | **0.57**(±0.42) | 0.29 (±0.19) | **0.49** (±0.35) | 0.29 (±0.41) | 0.19 (±0.22) | 0.17 (±0.2) | 0.10 (±0.12) | 0.26 (±0.2) | 0.10 (±0.112) |

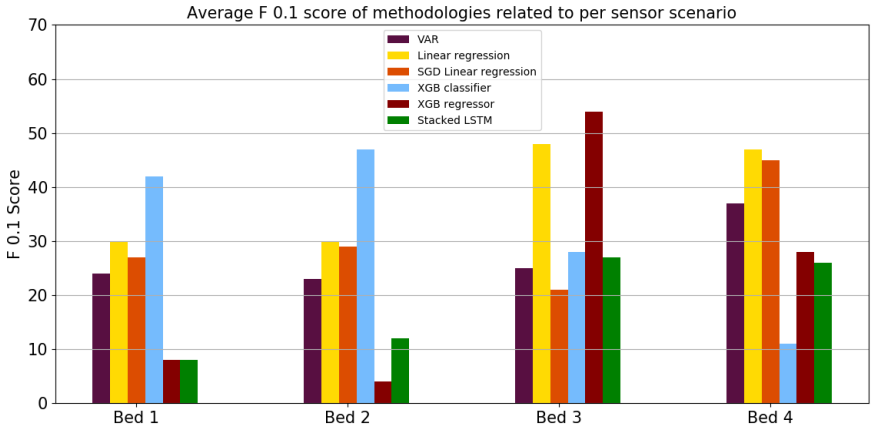



Figure 5.3: Average $F_{0.1}$ score of methodologies related to per sensor scenario

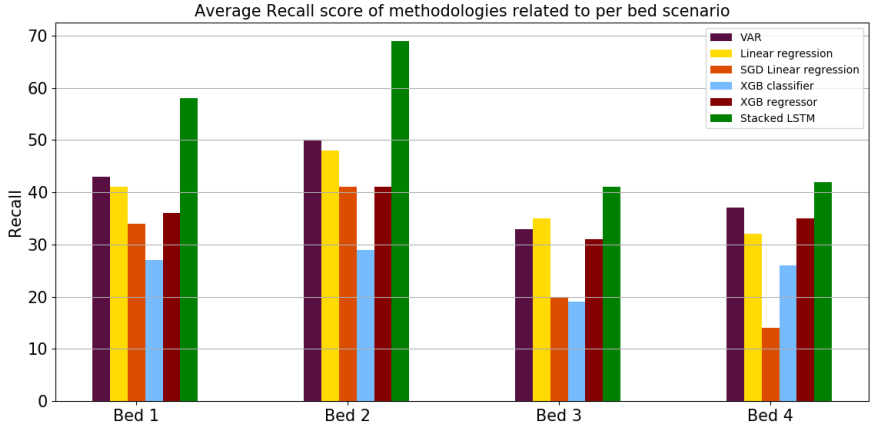



Figure 5.4: Average recall score of methodologies related to per bed scenario

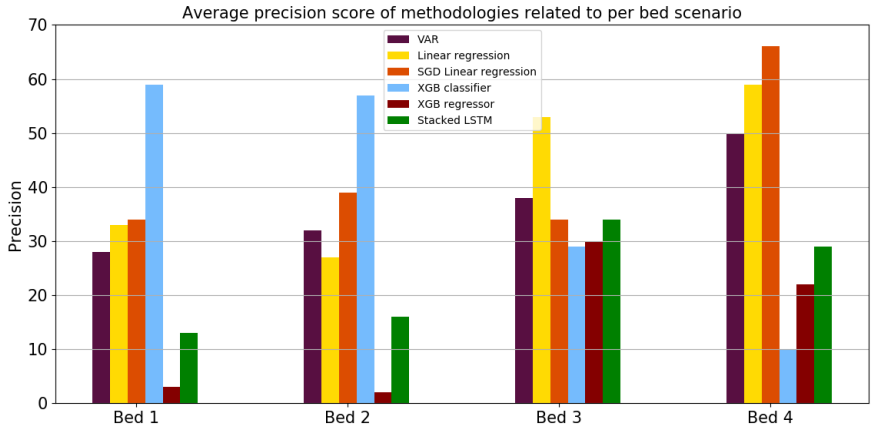



Figure 5.5: Average precision score of methodologies related to per bed scenario

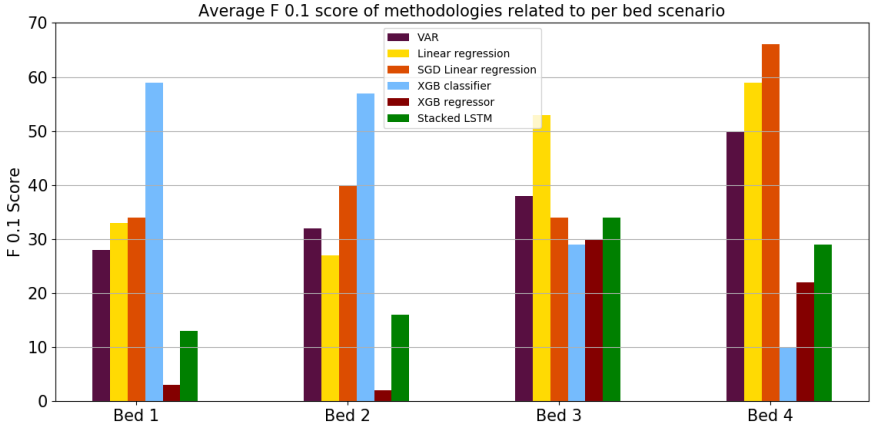



Figure 5.6: Average $F_{0.1}$ score of methodologies related to per bed scenario

### *5.4 Further Hyper-Parameter Analysis on the XGBoost Classifier*

Figures 5.3 and 5.6 show interesting differences between beds 1&2 and beds 3&4 for the tree ensemble models. Their performances are opposite of each other; the XGBoost Classifier performs well for the former and bad for the latter whereas it is vice-versa for the XGBoost Regressor. It should be noted that their hyper-parameters are quiet different as described in Section 5.1, thus the first hypothesis is that this discrepancy is because of this.

We perform an analysis by testing several number of tree estimators for the XGBoost classifier, and see if there is qualitative behavior change to make it similar to the behavior of the XGBoost Regressor. Our results are presented in Table 5.3. The table shows that the behaviors do not change drastically with number of estimators for beds 1, 2 and 3. In fact, the results are very similar for bed 1. Results for bed 2 show some decline but do not get close to that of the regressor's. For bed 3, the only change is for the 200 estimator case. For bed 1, overfitting is not an issue. For bed 2 it is slow to settle. For bed 3, it starts to become an issue for the 200 estimators case but this is the opposite behavior of the regressor. The story for bed4 is different as the regressor and the classifier have the same performances qualitatively. Only the bed4 results support the hypothesis.

With these results, we partially conclude that the discrepancy is not due to hyper-parameter values but due to data and the regression-classification difference (e.g. different loss functions). However, more time-consuming evaluations ared needed to be sure.

Table 5.3: Hyper parameter investigation on the XGBoost classifier.

| **Data** | **Number of estimators** | **Precision** | **Recall** | $F_{0.1}$ |
|---|---|---|---|---|
| **Bed1** | 10 | 0.56 (±0.42) | 0.26 (±0.20) | 0.39 (±0.29) |
| **Bed1** | 20 | 0.56 (±0.42) | 0.24 (±0.18) | 0.39 (±0.28) |
| **Bed1** | 30 (our study) | **0.60** (±0.43) | 0.20 (±0.15) | **0.42** (±0.22) |
| **Bed1** | 40 | 0.55 (±0.41) | 0.23 (±0.19) | 0.38 (±0.27) |
| **Bed1** | 50 | 0.55 (±0.41) | 0.24 (±0.20) | 0.38 (±0.28) |
| **Bed1** | 200 | 0.55 (±0.41) | 0.24 (±0.20) | 0.38 (±0.28) |
| **Bed1** | 200 (XGB regressor) | 0.08(±0.11) | 0.29 (±0.20) | 0.08 (±0.04) |
| **Bed2** | 10 | 0.49 (±0.36) | 0.27 (±0.17) | 0.45 (±0.33) |
| **Bed2** | 20 | 0.50 (±0.38) | 0.35 (±0.09) | **0.49** (±0.37) |
| **Bed2** | 30 (our study) | **0.51** (±0.41) | 0.26 (±0.17) | 0.47 (±0.35) |
| **Bed2** | 40 | 0.47 (±0.36) | 0.33 (±0.11) | 0.46 (±0.35) |
| **Bed2** | 50 | 0.45 (±0.40) | 0.31 (±0.09) | 0.48 (±0.32) |
| **Bed2** | 200 | 0.44 (±0.33) | 0.30 (±0.12) | 0.47 (±0.32) |
| **Bed2** | 200 (XGB regressor) | 0.04 (±0.04) | 0.29 (±0.17) | 0.04 (±0.02) |
| **Bed3** | 10 | 0.31 (±0.40) | 0.16 (±0.16) | 0.25 (±0.30) |
| **Bed3** | 20 | 0.35 (±0.40) | 0.15 (±0.15) | 0.29 (±0.31) |
| **Bed3** | 30 (our study) | 0.34 (±0.40) | 0.16 (±0.16) | 0.28 (±0.30) |
| **Bed3** | 40 | 0.33 (±0.40) | 0.17 (±0.16) | 0.27 (±0.30) |
| **Bed3** | 50 | **0.36** (±0.38) | 0.17 (±0.17) | **0.30** (±0.28) |
| **Bed3** | 200 | 0.30 (±0.36) | 0.19 (±0.17) | 0.26 (±0.38) |
| **Bed3** | 200 (XGB regressor) | 0.56 (±0.14) | 0.23 (±0.16) | 0.54 (±0.14) |
| **Bed4** | 10 | 0.09 (±0.10) | 0.11 (±0.08) | 0.09 (±0.10) |
| **Bed4** | 20 | 0.11 (±0.11) | 0.11 (±0.09) | 0.11 (±0.11) |
| **Bed4** | 30 (our study) | 0.11 (±0.11) | 0.12 (±0.10) | 0.11 (±0.11) |
| **Bed4** | 40 | 0.28 (±0.24) | 0.13 (±0.09) | 0.26 (±0.21) |
| **Bed4** | 50 | 0.27 (±0.24) | 0.15 (±0.07) | 0.27 (±0.23) |
| **Bed4** | 200 | **0.29** (±0.05) | 0.23(±0.09) | **0.29** (±0.05) |
| **Bed4** | 200 (XGB regressor) | 0.29 (±0.24) | 0.20 (±0.08) | 0.28 (±0.22) |

# Chapter 6

# DYNAMIC THRESHOLDING

The event definitions are True/False statements based on the measured data. These already include a threshold (e.g. the events in this work) or can be converted into a threshold-based statement (e.g. for a condition involving a categorical variable, we can use model probabilities and thresholds). The results presented in Section 5.3 show that we can design decision aids using the default thresholds. However, this does not let us tune the precision-recall trade-off. It is the case that other applications or individual operators may desired different decision aid conditions (i.e. recall-precision trade-off points). Towards this end, we present an approach to perform dynamic thresholding which includes a hyper-parameter to tune the precision-recall trade-off. We are specifically going to cover our first event (5 degree change in 15 minutes), however, other types of events involving measurements can be handled similarly[1].

The dynamic thresholding approach is based on the model assumed in Equations 3.1 and 3.2 and the running standard deviation of prediction errors. We use the difference between the current measurement and the predicted value as our prediction error:

$$e(t) = x(t) - \hat{x}(t+p) = x(t) - h(X^w(t), \theta) \tag{6.1}$$

Let $\sigma^c(t)$ denote the running standard deviation of the prediction errors between $t - c + 1$ and $t$. We use the following threshold to make a decision:

$$\tau_{dyn}(t) = \tau + K \cdot \sigma^c(t) \tag{6.2}$$

where $K$ is a tunable parameter to handle the precision-recall trade-off. Note that $K$ can be both negative (expected to increase recall) or positive (expected to increase

[1]Dynmic thresholding for probabilities would require another approach which we are not covering in this work.

precision). For this work, we chose the window size as $c = 25$. In addition to allowing for fine tuning, we also expect this proposed method to be more robust to temporally local measurement noise due to the involvement of the standard deviation. Note that this approach resembles simple hypothesis testing.

### *6.1 Evaluation with Temperature Forecasting*

Our preliminary analysis on thresholds showed that the re-trained linear regression model is the most amenable model to dynamic thresholding; it showed smooth trade-off behavior for all the beds (e.g. as opposed to sudden jumps or no improvement). Thus, we continue with that model in this analysis.

Since we did not find a significant difference in per sensor versus per bed performance, we continue with per sensor results. However, we are no longer using time-series cross-validation to get our results but instead use a more continuous approach. We use the first 20% of the data to train an initial model and then continue forecasting and re-training until the end. This is more fitting since the events are not distributed uniformly in time and we are already using a model with re-training. For analyzing this approach, we vary the hyper-parameter $K$ from 0 to 2 in 0.1 increments. We are not using any negative values as we concentrate on precision.

The performance scores w.r.t the parameter $K$ are given in Figures 6.1-6.4 for each bed, including multiple F-scores. As we increase $K$, precision increases and recall decreases which was expected. We can also see that $F_{0.1}$, $F_{0.25}$ and to an extent $F_{0.5}$ scores get better, at least initially. This implies that dynamic thresholding, specifically the hyper-parameter $K$, can be used to optimize for the desired version of the F-score. The parameter value and the performance for the best $F_{0.1}$ scores are given in Table 6.1. The recall scores are all lower than the desired point of 0.1 but our purpose in this analysis is to show that the trade-off is possible. Note that $K$ selection can be modified to include such conditions.

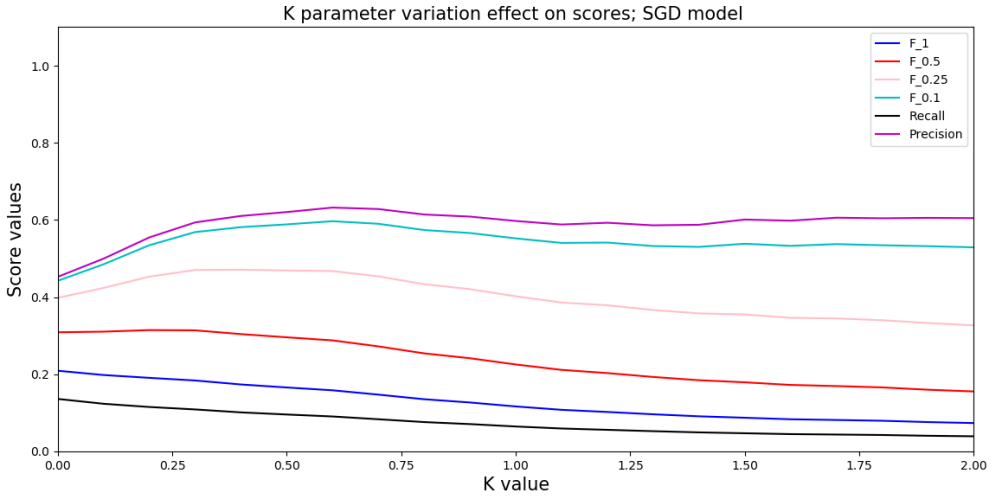



Figure 6.1: Dynamic thresholding for bed 1

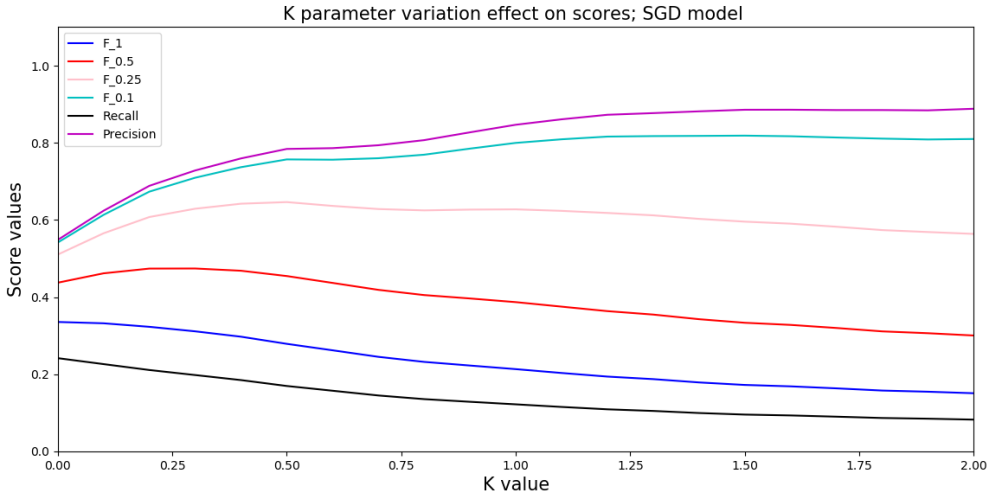



Figure 6.2: Dynamic thresholding for bed 2

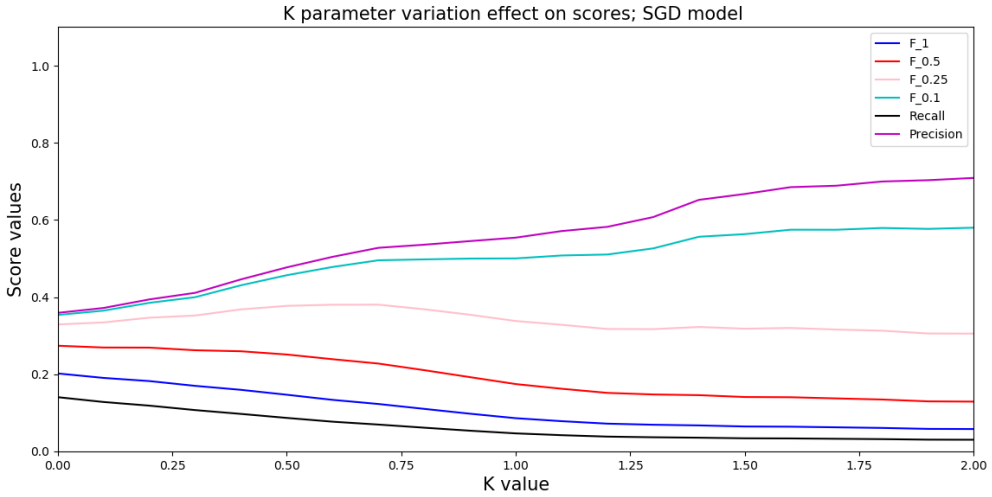



Figure 6.3: Dynamic thresholding for bed 3

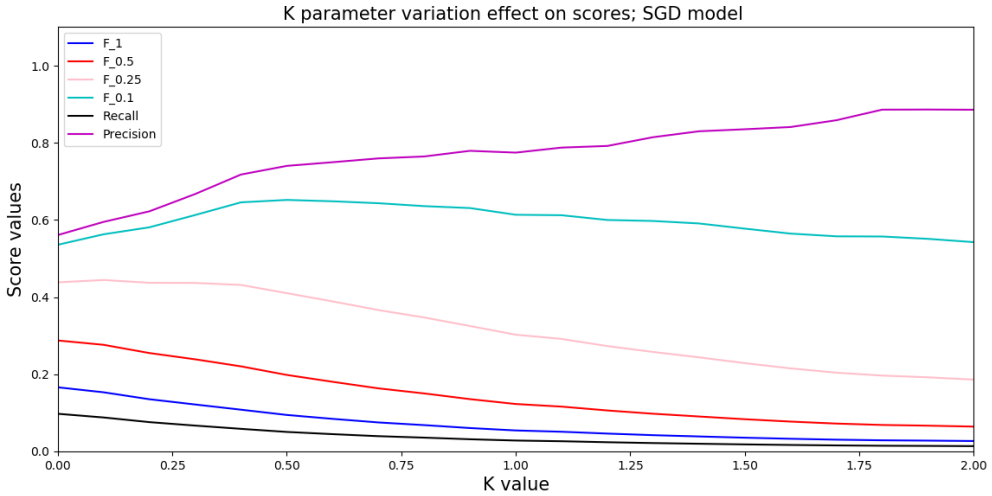



Figure 6.4: Dynamic thresholding for bed 4

Table 6.1: SGD linear regression model performance scores using dynamic thresholding approach

| **Data** | **K** | **Precision** | **Recall** | $\mathbf{F}_{0.1}$ |
|---|---|---|---|---|
| **Bed 1** | 0.6 | 0.63 | 0.09 | 0.60 |
| **Bed 2** | 1.5 | 0.89 | 0.09 | 0.82 |
| **Bed 3** | 2 | 0.71 | 0.03 | 0.58 |
| **Bed 4** | 0.5 | 0.74 | 0.05 | 0.65 |

## *6.2 Evaluation with Temperature Difference Forecasting*

In Section 3.1, we discuss that predicting the change/difference from the current value within the time horizon is also an option for time series forecasting, instead of direct value prediction. In this section, we train our models with difference inputs and predict the value change, as described by 3.3. Due to the same arguments given in the previous section, we only use the re-trained linear regression approach for with dynamic thresholding. We also just use the best $K$ values of the direct prediction approach from Section 6.1. This evaluation is limited but our aim is to look at the feasibility of change prediction instead of fully analyzing it.

The results are given in Table 6.2. These results suggest that the difference forecasting lead to $F_{0.1}$ score improvement for bed 1 and bed 3 data. In bed 1 case, both the recall and the precision improved. However, this approach did not yield improvement for the other beds. In overall, we conclude that utilizing difference time series forecasting method is promising but does not show obvious improvements. More investigation is needed.

Table 6.2: RE-trained (SGD) Linear Regression model performance scores with the dynamic thresholding approach.

| | **Direct Forecasting-Dynamic Thresholding** | | | **Difference Forecasting-Dynamic Thrsholding** | | |
|---|---|---|---|---|---|---|
| **Data** | **Precision** | **Recall** | $\mathbf{F}_{0.1}$ | **Precision** | **Recall** | $\mathbf{F}_{0.1}$ |
| **Bed1** | 0.63 | 0.09 | 0.60 | 0.77 | 0.11 | **0.73** |
| **Bed2** | 0.89 | 0.09 | 0.82 | 0.66 | 0.05 | 0.59 |
| **Bed3** | 0.71 | 0.03 | 0.58 | 0.79 | 0.02 | **0.60** |
| **Bed4** | 0.74 | 0.05 | 0.65 | 0.65 | 0.15 | 0.63 |

# Chapter 7

# CONCLUSION AND DISCUSSION

In this thesis, we have presented a data driven methodology to develop a decision aid system that can reduce the monitoring burden of the operators. There are only a few existing work that deals with developing a similar early warning system. However, these have certain limitations. Most of them use one dimensional time series data, whereas we have multiple dimension. Similarly, most of them do not deal with rare events (i.e. class-imbalance). Lastly, most of them applied an anomaly detection approach, instead of a more general event based approach (note that anomaly's can be defined as separate events). We argue that our approach is more comprehensive for a real application scenario.

We used both regression-based and classification-based machine learning methods towards event prediction. We showed that our direct event approach can be employed for a decision aid design. However, we did not find a single machine learning model that performed the best for all the cases. The most stark example was the performance of the XGBoost models between beds 1&2 and bed 3&4. Our hyper-parameter analysis suggests that this may be data related. This implies that multiple models should be taken into consideration while choosing one for the decision aid. Lastly, there was no significant gains by predicting events for the beds instead of individual sensors.

Since fine-tuning can be worthwhile as it can be more helpful to operators, we proposed an online thresholding method based on prediction errors. We showed that dynamic thresholding can achieve a desired trade-off behavior with a re-trained linear auto-regression model for our application. The trade-off can be achieved by setting a hyper-parameter to maximize a $F_\beta$ score (in our case $F_{0.1}$) and additionally by setting hard recall and precision limits if the results allow. We showed that dynamic thresholding can significantly increase a desired performance metric. We

also did a small analysis on using difference predictions. The results were in general similar to direct predictions but they warrant further study.

Our main goal in this work was to show that a data-driven methodology made it feasible to design operator decision aids. However, there are some missing points and potential future directions. We did not evaluate any re-training ideas for the tree ensemble models and only performed a small evaluation for change prediction. We also did not include the sensors of other beds for detecting the event of a current bed. These evaluations may reveal further useful points for decision aid design. We also did not specify a method to chose the trade-off hyper-parameter online. Future work should expand on these ideas by doing more evaluations and developing a full decision aid system that does not require additional intervention once setup or until the desired trade-off conditions change.

# Appendix 1

# APPENDICES

## *A.1 Performance Criteria*

In our problem, "1" corresponds to event and "0" corresponds to no event. The models' predictions are commonly compared with the ground truth (GT) as following:

- If both the GT class and the predicted class are 1, we call this a "True Positive" or $TP$ for short.
- If both of them are 0, we call this a "True Negative" ($TN$).
- If the GT class is 1 but the predicted class is 0 we call this a "False Negative" ($FN$).
- If the GT class is 0 but the predicted class is 1 we call this a "False Positive" ($FP$).

A way to visualize these is to use a confusion matrix as below:

| | | Predicted Class | |
|---|---|---|---|
| | | 1 | 0 |
| Ground Truth | 1 | $TP$ = number or ratio of $tp$'s | $FN$ = number of ratio or $fn$'s |
| | 0 | $FP$ = number or ratio of $fp$'s | $TN$ = number of ratio or $tn$'s |

**Recall:** The ratio of correctly predicted events to the total number of events:

$$recall = \frac{TP}{TP + FN} \tag{A.1}$$

**Precision:** The ratio of correctly predicted events to total number of positively predicted events

$$precision = \frac{TP}{TP + FP} \tag{A.2}$$

$F_\beta$ **Score:** The F-beta score is the weighted harmonic mean of precision and recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 precision + recall}$$

## *A.2 Cross validation detailed results*

### *A.2.1 Per sensor scenario*

Table A.1: Event prediction performance of our methodologies on bed 1 data pertain to the first scenario of the soft alarm, per temperature sensor, in each fold of the cross-validation method.

| Model | Fold 1 | | | Fold 2 | | | Fold 3 | | | Fold 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** |
| **Stacked LSTM** | 0.75 | 0.19 | 0.19 | 0.50 | 0.03 | 0.03 | no event | no event | no event | 0.16 | 0.03 | 0.03 |
| **XGB regressor** | 0.36 | 0.24 | 0.24 | 0.49 | 0 | 0 | no event | no event | no event | 0.02 | 0 | 0 |
| **SGD linear regression** | 0.17 | 0.50 | 0.51 | 0.50 | 0.18 | 0.18 | no event | no event | no event | 0.01 | 0.13 | 0.15 |
| **LR** | 0.66 | 0.68 | 0.68 | 0.5 | 0.09 | 0.09 | no event | no event | no event | 0.04 | 0.13 | 0.13 |
| **VAR** | 0.48 | 0.52 | 0.52 | 0.5 | 0.06 | 0.06 | no event | no event | no event | 0.15 | 0.13 | 0.13 |
| **XGB classifier** | 0.21 | 0.77 | 0.79 | 0.38 | 0 | 0 | no event | no event | no event | 0.01 | 0.50 | 1 |

Table A.2: Event prediction performance of our methodologies on bed 2 data pertain to the first scenario of the soft alarm, per temperature sensor, in each fold of the cross-validation method.

| Model | Fold 1 | | | Fold 2 | | | Fold 3 | | | Fold 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** |
| **Stacked LSTM** | 0.70 | 0.13 | 0.13 | 0.50 | 0.04 | 0.04 | no event | no event | no event | 0.62 | 0.20 | 0.20 |
| **XGB regressor** | 0.27 | 0.09 | 0.09 | 0.50 | 0.03 | 0.03 | no event | no event | no event | 0.09 | 0 | 0 |
| **SGD linear regression** | 0.25 | 0.60 | 0.61 | 0.50 | 0.15 | 0.15 | no event | no event | no event | 0.06 | 0.11 | 0.11 |
| **LR** | 0.51 | 0.64 | 0.64 | 0.50 | 0.08 | 0.08 | no event | no event | no event | 0.28 | 0.17 | 0.17 |
| **VAR** | 0.38 | 0.52 | 0.52 | 0.50 | 0.07 | 0.07 | no event | no event | no event | 0.18 | 0.11 | 0.11 |
| **XGB classifier** | 0.25 | 0.53 | 0.54 | 0.47 | 0 | 0 | no event | no event | no event | 0.06 | 0.87 | 1 |

Table A.3: Event prediction performance of our methodologies on bed 3 data pertain to the first scenario of the soft alarm, per temperature sensor, in each fold of the cross-validation method.

| Model | Fold 1 | | | Fold 2 | | | Fold 3 | | | Fold 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** |
| **Stacked LSTM** | 0.66 | 0.12 | 0.12 | 0.40 | 0.11 | 0.11 | 0.02 | 0.67 | 1 | 0.32 | 0.16 | 0.16 |
| **XGB regressor** | 0.39 | 0.45 | 0.45 | 0.40 | 0.80 | 0.81 | 0.05 | 0.46 | 0.50 | 0.10 | 0.46 | 0.48 |
| **SGD linear regression** | 0.16 | 0.39 | 0.40 | 0.40 | 0.23 | 0.23 | 0 | 0 | 0 | 0.04 | 0.20 | 0.21 |
| **LR** | 0.69 | 0.75 | 0.75 | 0.43 | 0.54 | 0.54 | 0 | 0 | 0 | 0.21 | 0.63 | 0.64 |
| **VAR** | 0.52 | 0.70 | 0.70 | 0.40 | 0.11 | 0.11 | 0 | 0 | 0 | 0.13 | 0.21 | 0.21 |
| **XGB classifier** | 0.23 | 0.32 | 0.32 | 0.40 | 0.03 | 0.03 | 0 | 0 | 0 | 0.03 | 0.76 | 1 |

Table A.4: Event prediction performance of our methodologies on bed 4 data pertain to the first scenario of the soft alarm, per temperature sensor, in each fold of the cross-validation method.

| Model | Fold 1 | | | Fold 2 | | | Fold 3 | | | Fold 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** | **Recall** | $F_{0.1}$ | **Presicion** |
| **Stacked LSTM** | 0.63 | 0.10 | 0.10 | 0.11 | 0.04 | 0.04 | 0.30 | 0.68 | 0.69 | 0.37 | 0.23 | 0.26 |
| **XGB regressor** | 0.33 | 0.19 | 0.19 | 0.18 | 0.03 | 0.03 | 0.11 | 0.64 | 0.67 | 0.18 | 0.26 | 0.26 |
| **SGD linear regression** | 0.17 | 0.58 | 0.59 | 0.07 | 0.19 | 0.19 | 0.02 | 0.67 | 1 | 0.02 | 0.37 | 0.45 |
| **LR** | 0.57 | 0.68 | 0.68 | 0.14 | 0.17 | 0.17 | 0.33 | 0.64 | 0.65 | 0.28 | 0.40 | 0.48 |
| **VAR** | 0.48 | 0.48 | 0.48 | 0.15 | 0.12 | 0.12 | 0.29 | 0.55 | 0.56 | 0.23 | 0.34 | 0.34 |
| **XGB classifier** | 0.28 | 0.24 | 0.24 | 0.11 | 0 | 0 | 0 | 0 | 0 | 0.10 | 0.21 | 0.22 |

### *A.2.2 Per bed scenario*

Table A.5: Event prediction performance of our methodologies on bed 1 data pertain to the second scenario of the soft alarm, per bed, in each fold of the cross-validation method.

| Model | Fold 1 | | | Fold 2 | | | Fold 3 | | | Fold4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** |
| **Stacked LSTM** | 0.94 | 0.31 | 0.31 | 0.50 | 0.01 | 0.01 | no event | no event | no event | 0.31 | 0.06 | 0.06 |
| **XGB regressor** | 0.51 | 0.17 | 0.10 | 0.5 | 0 | 0 | no event | no event | no event | 0.06 | 0 | 0 |
| **SGD linear regression** | 0.50 | 0.62 | 0.62 | 0.50 | 0.14 | 0.14 | no event | no event | no event | 0.02 | 0.24 | 0.27 |
| **LR** | 0.69 | 0.74 | 0.74 | 0.5 | 0.06 | 0.06 | no event | no event | no event | 0.04 | 0.18 | 0.19 |
| **VAR** | 0.55 | 0.57 | 0.57 | 0.5 | 0.05 | 0.05 | no event | no event | no event | 0.23 | 0.23 | 0.23 |
| **XGB classifier** | 0.31 | 0.75 | 0.76 | 0.5 | 0 | 0 | no event | no event | no event | 0.01 | 0.50 | 1 |

Table A.6: Event prediction performance of our methodologies on bed 2 data pertain to the second scenario of the soft alarm, per bed, in each fold of the cross-validation method.

| Model | Fold 1 | | | Fold 2 | | | Fold 3 | | | Fold4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** |
| **Stacked LSTM** | 0.95 | 0.42 | 0.27 | 0.50 | 0.02 | 0.01 | no event | no event | no event | 0.61 | 0.30 | 0.20 |
| **XGB regressor** | 0.59 | 0.05 | 0.05 | 0.50 | 0 | 0 | no event | no event | no event | 0.15 | 0 | 0 |
| **SGD linear regression** | 0.54 | 0.69 | 0.69 | 0.50 | 0.13 | 0.13 | no event | no event | no event | 0.18 | 0.35 | 0.35 |
| **LR** | 0.72 | 0.50 | 0.50 | 0.50 | 0.13 | 0.08 | no event | no event | no event | 0.21 | 0.23 | 0.23 |
| **VAR** | 0.70 | 0.61 | 0.61 | 0.50 | 0.07 | 0.07 | no event | no event | no event | 0.30 | 0.28 | 0.28 |
| **XGB classifier** | 0.33 | 0.71 | 0.72 | 0.50 | 0 | 0 | no event | no event | no event | 0.03 | 0.76 | 1 |

Table A.7: Event prediction performance of our methodologies on bed 3 data pertain to the second scenario of the soft alarm, per bed, in each fold of the cross-validation method.

| **Model** | **Fold 1** | | | **Fold 2** | | | **Fold 3** | | | **Fold4** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** |
| **Stacked LSTM** | 0.94 | 0.13 | 0.13 | 0.21 | 0.02 | 0.02 | 0.07 | 0.88 | 1 | 0.41 | 0.22 | 0.22 |
| **XGB regressor** | 0.70 | 0.18 | 0.18 | 0.21 | 0.21 | 0.21 | 0.17 | 0.37 | 0.38 | 0.17 | 0.41 | 0.42 |
| **SGD linear regression** | 0.52 | 0.72 | 0.72 | 0.21 | 0.19 | 0.19 | 0 | 0 | 0 | 0.07 | 0.44 | 0.47 |
| **LR** | 0.79 | 0.85 | 0.85 | 0.36 | 0.50 | 0.50 | 0 | 0 | 0 | 0.26 | 0.75 | 0.77 |
| **VAR** | 0.76 | 0.87 | 0.87 | 0.21 | 0.11 | 0.11 | 0 | 0 | 0 | 0.35 | 0.55 | 0.55 |
| **XGB classifier** | 0.55 | 0.17 | 0.17 | 0.21 | 0.01 | 0.01 | 0 | 0 | 0 | 0.01 | 0.50 | 1 |

Table A.8: Event prediction performance of our methodologies on bed 4 data pertain to the second scenario of the soft alarm, per bed, in each fold of the cross-validation method.

| **Model** | **Fold 1** | | | **Fold 2** | | | **Fold 3** | | | **Fold4** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** | **Recall** | $F_{0.1}$ | **Precision** |
| **Stacked LSTM** | 0.91 | 0.22 | 0.22 | 0.07 | 0.02 | 0.02 | 0.26 | 0.63 | 0.64 | 0.46 | 0.29 | 0.29 |
| **XGB regressor** | 0.60 | 0.06 | 0.06 | 0.26 | 0.01 | 0.01 | 0.20 | 0.42 | 0.43 | 0.33 | 0.39 | 0.39 |
| **SGD linear regression** | 0.42 | 0.67 | 0.67 | 0.06 | 0.25 | 0.26 | 0.04 | 0.80 | 1 | 0.03 | 0.58 | 0.71 |
| **LR** | 0.67 | 0.82 | 0.82 | 0.14 | 0.23 | 0.23 | 0.32 | 0.70 | 0.71 | 0.14 | 0.60 | 0.62 |
| **VAR** | 0.67 | 0.68 | 0.68 | 0.19 | 0.23 | 0.23 | 0.34 | 0.64 | 0.65 | 0.28 | 0.45 | 0.45 |
| **XGB classifier** | 0.55 | 0.10 | 0.10 | 0.28 | 0 | 0 | 0 | 0 | 0 | 0.23 | 0.29 | 0.29 |

# BIBLIOGRAPHY

[1] Abdelelah Almounajjed, Ashwin Kumar Sahoo, Mani Kant Kumar, and Muhannad Drak Alsebai. Investigation techniques for rolling bearing fault diagnosis using machine learning algorithms. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 1290–1294. IEEE, 2021.

[2] Sara Antomarioni, Maurizio Bevilacqua, Domenico Potena, and Claudia Diamantini. Defining a data-driven maintenance policy: an application to an oil refinery plant. *International Journal of Quality & Reliability Management*, 2019.

[3] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[4] A Faysal, MSNA Adhreena, E Vorathin, ZM Hafizi, and WK Ngui. Leak diagnosis of pipeline based on empirical mode decomposition and support vector machine. In *IOP Conference Series: Materials Science and Engineering*, volume 1078, page 012023. IOP Publishing, 2021.

[5] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model. *arXiv preprint arXiv:1612.06676*, 2016.

[6] Ana Cristina B Garcia, Cristiana Bentes, Rafael Heitor C de Melo, Bianca Zadrozny, and Thadeu JP Penna. Sensor data analysis for equipment monitoring. *Knowledge and information systems*, 28(2):333–364, 2011.

[7] Ivano Irrera, Carlos Pereira, and Marco Vieira. The time dimension in predicting failures: A case study. In *2013 Sixth Latin-American Symposium on Dependable Computing*, pages 86–91. IEEE, 2013.

[8] Zhen Jia, Zhenbao Liu, and Yongyi Cai. A novel fault diagnosis method for aircraft actuator based on ensemble model. *Measurement*, 176:109235, 2021.

[9] Dazhi Jiang, Jian Gong, and Akhil Garg. Design of early warning model based on time series data for production safety. *Measurement*, 101:62–71, 2017.

[10] Athar Khodabakhsh, Ismail Ari, Mustafa Bakir, and Ali Ozer Ercan. Multivariate sensor data analysis for oil refineries and multi-mode identification of system behavior in real-time. *IEEE Access*, 6:64389–64405, 2018.

[11] Tomás Kuzin and Tomás Borovicka. Early failure detection for predictive maintenance of sensor parts. In *ITAT*, pages 123–130. Citeseer, 2016.

[12] Patrícia Ramos, José Manuel Soares Oliveira, and Paula Silva. Predictive maintenance of production equipment based on neural network autoregression and arima. In *21st International EurOMA Conference-Operations Management in an Innovation Economy*, 2014.

[13] Mahmoud Reza Saybani, Teh Ying Wah, Adel Lahsasna, Amineh Amini, and Saeed Reza Aghabozorgi. Data mining techniques for predicting values of a faulty sensor at a refinery. In *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, pages 792–796. IEEE.

[14] Bram Steurtewagen and Dirk Van den Poel. Adding interpretability to predictive maintenance by machine learning on sensor data. *Computers & Chemical Engineering*, page 107381, 2021.

[15] Karthick Thiyagarajan, Sarath Kodagoda, and Linh Van Nguyen. Predictive analytics for detecting sensor failure using autoregressive integrated moving

average model. In *2017 12th IEEE conference on industrial electronics and applications (ICIEA)*, pages 1926–1931. IEEE, 2017.

[16] Lu Yu, Jianling Qu, Feng Gao, and Yanping Tian. A novel hierarchical algorithm for bearing fault diagnosis based on stacked lstm. *Shock and Vibration*, 2019, 2019.

[17] Haitao Zhao, Shaoyuan Sun, and Bo Jin. Sequential fault diagnosis based on lstm neural network. *IEEE Access*, 6:12929–12939, 2018.