School of Computer Sciences

CDS 590 — Consultancy Project & Practicum

Final Report

IMPROVED DETECTION SCHEME IN CARBON DIOXIDE FLUX ANOMALIES USING A LINEAR ONE-CLASS SVM WITH DEEP LEARNING

CHEE SAI WAI

P-COM0145/19

Supervisor: Dr. Mohd Nadhir Bin Ab Wahab

Lecturer(s): Dr. Nasuha Lee Abdullah

DECLARATION

"I declare that the following is my own work and does not contain any ***unacknowledged*** work from any other sources. This project was undertaken to fulfill the requirements of the Consultancy Project & Practicum for the Master of Science (Data Science and Analytics) program at Universiti Sains Malaysia".

Signature : ……

Name : Chee Sai Wai

Date : 02/07/2021

ACKNOWLEDGEMENT

First of all, I would like to thanks to my supervisor, Dr. Mohd Nadhir Bin Ab Wahab for his precise guidance, encouragement and support throughout this practicum. With his supervision, this project most likely is able to succeed till this stage.

I would also like to thanks to my industry mentor, Dr. Yusri Yusup which give an opportunity for me to work under his team. I am pleasure can meet excellence team members here and working together contribute to the project.

Not to mention our course lecturers Dr. Zarul Fitri Zaaba and Dr. Nasuha Lee Abdullah for the continuous support and valuable suggestion throughout the whole course since CDS506.

Last but not least, I would like to express my gratefulness to my parents and friends as well for their continuous support and always motivate me during this practicum time.

ABSTRACT

Eddy covariance is a simple evaluation of first-order statistics used for trend in automated exploration of exchanges of gas on the uniform Earth cover. An eddy covariance station has been set up at the Centre for Marine and Coastal Studies of Universiti Sains Malaysia to solve the most basic situation to collect EC data between atmosphere and ocean. With the eddy measurements among sites and time scales, an improved monitoring scheme is required to detect any hidden patterns of anomalies in the high dimensional EC flux dataset. Common unlabeled flux dataset is always many large-scale data caused by hard-designed features as well as unreliable sensor connectivity. This help produces reliable and interpretability in large dataset which makes extra care in handling detection accuracy and overfitting. This also involve performance of existing binary labels on combined single forecast stimulation record to correct extreme situations. This work focused on the analysis of new anomaly threshold and choice of deep model within simpler deep learning architecture using single layer autoencoder, stacked autoencoder or stacked autoencoder with dropout. All the algorithms are compared using validation loss and accuracy metrics, AUC-ROC metrics and threshold robustness. Given fine-tuning parameters trained on deep models, a better model can be achieved with the highest AUC-ROC value (0.82), meanwhile a better threshold was MAD'z-score with the perfect recall and therefore a better convergence was smoother loss curve. Overall, research work is completed until data science activities within sense of commitment otherwise demo is discussed and presented. In this practicum, I was able to understand client problem and manage to propose not a best yet effective solution to tackle the problem. In the mean process, I benefit a lot from impact on which the team worked to the individual or organization. Since this project is associated with deep anomaly detection, I also aware needs or opportunities of problem might include, for example, use of new technology or introduction of a new process. Whatever the need must now turned into an image of how the project will look on completion.

ABSTRAK

Eddy covariance adalah penilaian ringkas statistik pesanan pertama yang digunakan untuk trend penerokaan automatik pertukaran gas di penutup Bumi yang seragam. Sebuah stesen kovarians eddy telah didirikan di Pusat Pengajian Laut dan Pantai di Universiti Sains Malaysia untuk menyelesaikan situasi paling asas untuk mengumpulkan data EC antara atmosfera dan lautan. Dengan pengukuran eddy di antara laman web dan skala waktu, skema pemantauan yang lebih baik diperlukan untuk mengesan sebarang corak anomali tersembunyi dalam set data fluks EC dimensi tinggi. Set data fluks biasa yang tidak berlabel selalu banyak data berskala raya yang disebabkan oleh ciri-ciri yang direka keras dan juga sambungan sensor yang tidak boleh dipercayai. Ini membantu menghasilkan kebolehpercayaan dan penafsiran dalam set data raya yang memberikan perhatian tambahan dalam menangani ketepatan pengesanan dan pemasangan berlebihan. Ini juga melibatkan prestasi label binari yang ada pada rekod rangsangan ramalan tunggal gabungan untuk membetulkan keadaan yang melampau. Karya ini memfokuskan pada analisis ambang anomali baru dan pilihan model mendalam dalam seni bina pembelajaran mendalam yang lebih sederhana menggunakan autoencoder lapisan tunggal, autoencoder bertumpuk atau autoencoder bertumpuk dengan putus sekolah. Semua algoritma dibandingkan menggunakan metrik kehilangan dan ketepatan pengesahan, metrik AUC-ROC dan ketahanan ambang. Memandangkan parameter penyesuaian yang dilatih pada model dalam, model yang lebih baik dapat dicapai dengan nilai AUC-ROC tertinggi (0.82), sementara itu ambang yang lebih baik adalah skor MAD dengan penarikan yang sempurna dan oleh itu penumpuan yang lebih baik adalah keluk kerugian yang lebih lancar . Secara keseluruhan, kerja penyelidikan diselesaikan sehingga aktiviti sains data dalam arti komitmen sebaliknya demo dibincangkan dan dibentangkan. Dalam praktikum ini, saya dapat memahami masalah pelanggan dan berjaya mencadangkan bukan penyelesaian terbaik namun berkesan untuk mengatasi masalah tersebut. Dalam proses min, saya mendapat banyak manfaat daripada kesan pasukan bekerja kepada individu atau organisasi. Oleh kerana projek ini dikaitkan dengan pengesanan anomali yang mendalam, saya juga menyedari keperluan atau peluang masalah mungkin termasuk, misalnya, penggunaan teknologi baru atau pengenalan proses baru. Apa pun keperluannya sekarang mesti berubah menjadi gambaran bagaimana projek itu akan siap.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

AE - Autoencoder

CAE - Contractive autoencoder

DAD - Deep Anomaly Detection

DAE - Denoising autoencoder

DBN - Deep Belief Network

EC - Eddy Covariance

OCSVM - One-class Support Vector Machine

SAE - Sparse autoencoder

SVM - Support Vector Machine

CHAPTER 1

INTRODUCTION

1.1 Background

School of Industrial Technology (PPTI) is one destination of Environmental Technology faculty at Universiti Sains Malaysia (USM), as yet government held. Compliments of public research activities in PPTI is planning engagement on atmosphere interaction research. Anyone of the research team are coming in participating technology program at the Centre for Marine and Coastal Studies (CEMACS) of USM. Dr. Yusri Yusup is responsible for a group of collaborators, dealing as a mentor in the strategic support of atmosphere-sea flux and global carbon-moisture cycle, and coordinating Eddy Covariance (EC) approach. CEMACS is paying attention to surrounding meteorological parameters, where EC stations are delivering information for instance carbon dioxide flux, latent and sensible heat fluxes, relative humidity, atmospheric temperature along with the explainable derived variables. Web databases at PPTI is bringing storage supplies during every 30 min interval measurements for every day since 2015. The dataset is source of data tables to study the mechanism uptake of measurements between ocean and atmosphere. All uptake of measurements are competing against selected parameters of climate models. Climate models, both data simulation and report generation are expected, which is by the time with the many applications in the integration of the global carbon cycle model available online.

1.2 Domain Background

Eddy covariance (EC), a simple evaluation of first-order statistics (Foken, Aubinet, & Leuning, 2012), or an alternative to the used “eddy correlation” in 1951, is a trend in automated exploration of exchanges of gas on the uniform Earth cover. These weather-based aids have seen massive speed improvements in many areas, including hydrology, ocean science and agriculture science. Today, distribution of EC data within academic community compete in near future as EC method makes sense of the world. Its goal solves the most basic situation to collect EC data, despite some success in between, such as flux measurement networks, FLUXNET (Jung et al.,

2011) become the best eddy covariance tower sites in the global scale or such as the application of Ameriflux (Xiao et al., 2008) for estimation of net carbon system exchange.

There are two primary components of an EC general station: 1) an ultrasonic anemometer that measures wind velocity and direction, and 2) Infrared Gas Analyzer (IRGA) measures the airborne gas concentration (Burba & Anderson, 2010). For this EC approach, there are neither standardized procedures nor consistent terms and conditions for implementation. This method is difficult from a computational perspective and depends on the design of experiments (DOE) and needs. Common examples are choice of measurement spot, products of data acquiring, procedure of data handling, or conservation of EC station.

In the past decade, air-sea flux interactions have received enormous attention. Many of the complete works centered on fashion and flow of carbon dioxide flux (McGillis et al., 2004; McKinley, Follows, & Marshall, 2004); in particular study of the mechanisms of the flux variability, link with other weather-related attributes, and estimates of means, variances and covariances expressed in a climate model. Fluxes are constant with height, justifying the use of molecular mass to measure the turbulent movement of the atmospheric surface layers (Burba & Anderson, 2010). Land, water and coastal waters effects behind the simple terms can be used for energy. Land type energy is a soil transport task; water type energy is a sea transport task; coastal waters balance energy over two source regions through flows.

The data gathered helps researchers conduct analyses, such as developing models to predict the sustained trend of CO_2, it is critical to also analyse the root of greenhouse gases closer to the EC stations. In addition, the data will enable decision makers to write laws to address urgent issues and better quality of life for people. For this reason, many EC sites have been established in their own states and countries for ongoing environmental monitoring.

1.3 Problem Statement

Given that the process of creating standard weather measures is minor, the bias will reflect the uncertainty of climate variability. For spikes, simple deletion reveals “outliers” or “abnormalities” when anything causes data to be undone. According to Aubinet et al. (2012), besides instrument error and problem with the sensor of EC station (Vitale et al., 2020), this

issue refers as representativeness of the data, footprint of the measurements and assumption about eddy measurements among sites and time scales. A basic level of data quality, prescriptive procedures and objectives of spikes elimination have been derived through quality control for all predictor variables.

At any time, quality assurance (QA) and quality control (QC) are tests on the fulfillment to high quality of flux data. As such, real time of minimal data loss by QC is connected with management of a measuring program by QA. All measurement site are different level of controls. QA of eddy measurements therefore has checks of maintained towers, choice of testing development and data processing pipelines. Since eddy measurements are underestimating half hourly flux estimates, QC has important standardized procedures to have rejected values taken into effective correction of EC flux dataset.

This automatic detection open issues and challenges faced while adopting background of unsupervised machine learning for simple and complex features. Hence, these anomalies need more investigation by extracting useful knowledge from deep learning. This ensure the prediction of any hidden patterns of anomalies in the EC flux dataset. Nevertheless, the flow of EC flux data is much more diverse, reducing necessary data dimension and developing visualization outside of model architecture settings.

1.4 Objectives of Project

The objectives of this study are listed as follows:

1. To build outliers detector of carbon dioxide flux by unsupervised deep autoencoder and OCSVM algorithms, and compare them to an efficient method of learning parameters.

2. To obtain the comparative evaluation metric to the availability based on IQR labeling for detection on future anomalous carbon dioxide flux.

1.5 Benefit of Project

The benefits of this study are listed as follows:

1. Applying a tested IQR labels for normal data relied on quality of deep learning methods. Common unlabeled flux dataset is always many large-scale data, hence the existing solution will capture complex structures in finding outliers.

2. Promoting time savers by improving analysis of new anomaly threshold and choice of deep model within simpler deep learning architecture for flux data users, being applied in data presentation, thus help the department handling detection accuracy, overfitting and produce a reliable dataset.

CHAPTER 2

RELATED WORKS

2.1 Introduction

Anomaly detection has been the major focus of several kinds of research and scientific papers over the years and according to Pang et al. (2020). Therefore, in the existing literature, several techniques and approaches have been proposed to detect anomalies as well as to improve the performance of existing anomaly detection techniques. In this section, the most relevant techniques in the state-of-the-art will be discussed.

2.2 Deep Anomaly Detection (DAD) in Application Domains

Deep anomaly detection (DAD) is a common problem across several domains. Current methods are being improved on demand, using data patterns as part of several market applications. Better drivers in architectures and found functionality would be the equivalent of current DAD approaches. In the literature on deviation identification, two types of examples are commonly used: domain-based and reconstruction-based. Shallow DAD approaches go beyond interpretable models by carefully choosing and fine-tuning architecture, regardless of hot topics in areas of concern such as network intrusion, patient health checking, fault identification, and fraud detection. Similarly, interpretability of other methods is as much a feature of learning capacity as it is of unsupervised capabilities in adapting change to detection accuracy of a particular domain.

One of the current studies focused on the application of the DAD technique in network security domain. Intrusion detection, according to Vieira and Ribeiro (n.d.), will greatly enhance the plan for analysing and detecting malicious code. The key concept is to detect cyber-attacks by finding suspicious machine activity, and shallow learning models (like SVMs) can be a useful tool in this regard. Erfani et al. (2016) used numerous benchmark datasets to test Deep Belief Networks (DBNs) with a linear one-class SVM. Authors applied self-taught techniques can be used also with a sparse autoencoder, so as to learn features of an unlabeled network intrusion data set

(Javaid, Niyaz, Sun, & Alam, 2016). Over the NSL-KDD dataset, the UNSW-NB15 dataset has a similar architecture for outlier detection (Albahar & Binsawad, 2020).

Examples where image and sequential dataset in medical domain have a significant effect on the ability and accuracy of learning algorithms, as demonstrated by authors for interpretability in large datasets (Chalapathy & Chawla, 2019). Authors applied DBNs for medical imaging such as EEG (electroencephalography) records, to detect anomalies such as the malfunction of the brain (Wulsin, Blanco, Mani, & Litt, 2010). Likewise, authors proposed interpretable models for patient health diagnosis in medical application in the industrial context (ElShawi, Sherif, Al-Mallah, & Sakr, 2020). Some other works focused on combined single medical record such as patient health record, by checking regions with high residual errors ranging from ECG reports (time series data), blood test reports (key value pairs), and x-rays (images) (Erhan et al., 2021).

One of the current studies have also been used in climate science experiments in the context of industrial sensor networks (Reichstein et al., 2019). Diaz and Hollmen (2002) also used an autoencoder, where the residual mean-square error is used to quantify novelty. Using autoencoder in similar unlabeled observations, detailed various noise sources and high-dimensional space recorded in temperature and humidity sensors (Luo & Nagarajan, 2018). In a study of unsupervised DAD architectures as Reichstein et al. (2019), for examples applying spatial features, events, and forecast simulations to correct extreme situations. Furthermore, Oliveira et al. (2021) proposed good reconstructed patterns from common tabular data over fault detection datasets, which has rarely been explored in the literature. Similarly, in the domain of energy consumption, authors detailed features in an unlabeled data set with sufficient training consumption observation (Himeur, Ghanem, Alsalemi, Bensaali, & Amira, 2021).

When dealing with high-dimensional data, by exploring deviations in data, unsupervised DAD approaches are flexible and more efficient in finance domain for fraud detection (Chalapathy & Chawla, 2019; Oliveira et al., 2021). Streaming data and the imbalance property of banking datasets are revised deep anomaly reviews of detecting credit card frauds (Chalapathy & Chawla, 2019). Using deviation analysis of reconstructed input features has been applied in vibration data for fraud detection in synchronous mechanical units (Diaz & Hollmen, 2002; Pimentel, Clifton,

Clifton, & Tarassenko, 2014). Thus, it is more suitable for autoencoder-based DAD applications in large-scale systems as a fraudulent source of funds in a transaction.

2.3 Autoencoder-based Architectures for Unsupervised Deep Method

To that end, a variety of deep learning methods can be used in different types inside an autoencoder (AE) in an unsupervised environment. Recent works have emerged when features are manually extracted, taking advantage of an overview for various forms of autoencoders. As a result, the effectiveness of a particular algorithm is preferred. Most frameworks, such as simple autoencoders, can be adapted by reconstructing the natural behaviour when designing and employing them. This includes highlighting problems when using it to distinguish uncommon and highly variable occurrences, or when scaling output for enhanced detection schemes in CO_2 flux of industrial measurement sites.

An AE is made up of two networks: an encoding network and a decoding network. The retained information is needed to reduce the overall reconstruction error. This architecture allows AEs to be used for efficient data compression or dimension reduction (Hinton & Salakhutdinov, 2006; Jiang, Gao, Hong, & Cai, 2014). As work demonstrated by Oliviera et al. (2021), an AE can be compressing the input through lower dimension layers (encoder), and then decompressing (reconstructing) the original input to its original size (decoder). As the paper (Himeur et al., 2021) noted, the data instances such as anomalies that deviate from the majority of the data are poorly reconstructed. The data reconstruction error can therefore be directly used as anomaly score. The basic formulation of this approach is given as follows.

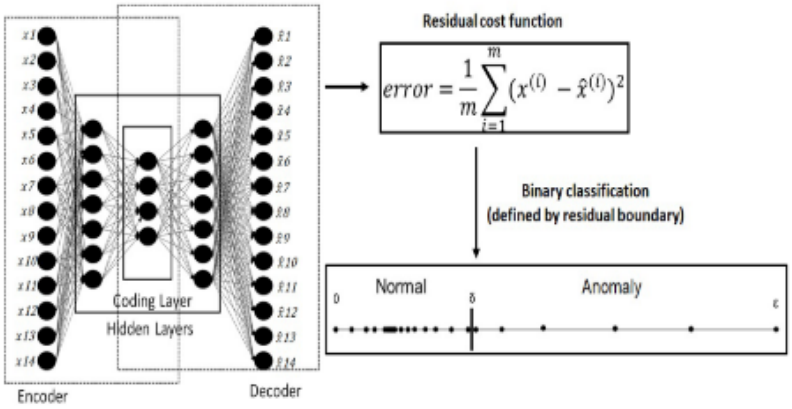


Figure 2.1: Traditional approach for standard autoencoder applied to anomaly detecction. (Oliveira et al., 2021)

The reconstruction loss is the loss function for a standard autoencoder. In the case of continuous data, the MSE loss function is a popular option because it calculates the difference between the reconstructed and real input. Since autoencoders share similar characteristics under the AE framework, Pang et al. (2020) introduced regularized autoencoders for the problem of deep anomaly detection, namely sparse autoencoder (SAE), denoising autoencoder (DAE), contractive autoencoder (CAE), and variational autoencoder (VAE). These autoencoder selections are often based on the quality of the data or are combined with limited preprocessing in anomaly detection tasks.

In the paper (Glorot, Bordes, & Bengio, 2011), the ReLU activation function was used to implement a method of generating an SAE representation by achieving true zero in the code layer. Glorot et al. (2011) found many reasons why an SAE representation could be appealing, including knowledge disentangling and efficient variable-size representation, to name a few. Makhzani and Frey (2013) combined the activation units of the secret layer of SAE are used in the training of an SAE from a desired sparsity.

DAE attempts to recreate data from predefined corrupted data instances rather than original data (Vincent et al., 2010). Autoencoders for denoising are equipped to map a noisy input to an output. The noise in the input layer can be conducted to learn and separate the most stable features from noise. CAE can be taken an account by learning small variations in the instances of their neighbours (Rifai, Vincent, Muller, Glorot, & Bengio, 2011). In Rifai et al. (2011), Frobenius norm of the encoder's Jacobian matrix is used as a penalty expression. Authors used DAE with CAE and squared error loss and then parameterization to achieve the objectives (Alain & Bengio, 2014).

A VAE is a stochastic generative model with calibrated probabilities and can be trained with gradient-based methods (Kingma & Welling, 2013; Rezende, Mohamed, & Wierstra, 2014). For this reason, autoencoders (AEs), denoising autoencoders (DAEs), and sparse autoencoders (SAEs) are referred as deterministic model without probabilistic foundation. Authors described the reconstruction likelihood as the anomaly score in VAE, while the reconstruction error as only source in others (An & Cho, 2015). The same technique in (Doersch, 2016) introduced regularisation to the representation space using encoding data instances for the latent space.

2.4 Advantages, Drawbacks and Challenges by DAD and Conventional Method as Anomaly Detector

Recent research has focused on combining one-class neural networks (Chalapathy, Menon, & Chawla, 2018; Nguyen & Vien, 2018). One of the main advantages of one-class classification-based anomalies was well studied, leading to a solid basis approaches in the reconstruction-based group. Moreover, one argument in favour feature extractor was as a positive effect on the 'curse of dimensionality'. Further consideration found instead the original input space, using one-class hyperplane from the neural network-enabled low-dimensional representation space (Chalapathy & Chawla, 2019; Pang et al., 2020). However, as Pang et al. (2020) discovered, the identification efficiency is reliant on one-class classification-based anomaly controls. Furthermore, Chalapathy and Chawla (2019) discovered that when individual layers are objective to optimise detection efficiency, models perform better.

Recent studies dedicated to combine one-class SVM with AEs (Andrews, Morton, & Griffin, 2016; Yahaya, Langensiepen, & Lotfi, 2018). One argument in favour reconstruction-based group were more focused on the concept of unsupervised deep anomaly detection. The study of discovered AEs is simple and adaptable to various types of data. Also, the various types of strong AE variants that can be used to execute anomaly. Adding to this, a cost-effective technique for distinguishing between regular and anomalous of annotated data for training (Chalapathy & Chawla, 2019; Pang et al., 2020). Pang et al. (2020) however found a drawback, since infrequent existence of anomalies in the training data can skew the trained feature representations. This was also found the effect of dimension reduction on the objective role of data reconstruction than in anomaly detection.

Both methods described above can be generalised to pros and cons on one-class SVM. Conventional one-class SVM is to learn a hyperplane that maximize a margin between training data instances and the origin. In domain-based category, Pimentel et al. (2014) related to the novelty boundary using only those nearly data and found to not influence distribution in the training set. Pimentel et al. (2014) however found a drawback, as the complexity associated with the computation of the kernel functions. One argument against the studies aimed at resolving the problem of selecting the right kernel function. Obviously those that mainly agreed a potential

danger, for example reporting one-class kernel Fisher discriminant classifier (Roth, 2006) and exploring subspaces of the data, then training a separate model for each subspace (Evangelista, Embrechts, & Szymanski, 2006). It is difficult to choose values for the parameters that govern the size of the boundary field, as it was in the previous analysis.

While Pang et al. (2020) found end-to-end optimization of the whole anomaly detection pipeline in deep methods, learning of representations for anomaly detection also has been addressed. Little support was found, especially established approaches lack these two skills, which are critical for addressing the challenges. Except for the data type from experiments, some classified normal data or other labelled anomaly data might well be accessed. Among these results, three studies proved reducing large-scale labeled data as in fully supervised settings (Aggarwal, 2017; Zimek, Schubert, & Kriegel, 2012). Pang et al. (2020) found this subsequently results in more informed models and thus better recall rate. For unsupervised methods of the recent surveys (Breunig, Kriegel, Ng, & Sander, 2000; Liu, Ting, & Zhou, 2012), still often incur high false positives on real-world datasets (Campos et al., 2016; Pang, Shen, & van den Hengel, 2019).

For the anomaly explanation challenge, deep methods are often black-box models. This findings was similar to that in Pang et al. (2020) to unify anomaly detection and explanation into single frameworks. Thus, deep model explanation (Du, Liu, & Hu, 2019) and actionable knowledge discovery (Cao, Zhang, Philip, & Zhao, 2010) can be seen as a possible explanation for anomalies discovered by particular models. Cao (2015) extended the difficulties for studying complex systems and interactions from data types, including high-dimensional data, image data, and video data. This capability is critical for a variety of problems, including anomaly detection in high-dimensional data. (Zimek et al., 2012), low anomaly detection recall rate (Campos et al., 2016), data-efficient learning of normality/abnormality (Aggarwal, 2017), and detection of complex anomalies. In the previous studies (Goodfellow, Bengio, Courville, & Bengio, 2016), principled systems train heterogeneous data sets, as well as data-efficient normality/abnormality learning and dynamic anomaly detection. While shallow methods exist for dealing with complex data, they are typically much poorer and less adaptable than deep methods. Moreover, Pang et al. (2020) considered many effective and easy-to-use network architectures.

2.5 Data Science and Analytics Tools in Deep Learning Framework

TensorFlow is a deep learning library from Google (Abadi et al., 2016) that can be used to directly build neural networks, as well as that is used for models using the Keras library (Chollet, 2015). Keras utilizes TensorFlow code to run, and then using Keras objects and functions for the most part. Keras is built because it is often more to easily create models and get them training faster. Pandas is a package that is useful for data I/O, cleaning, manipulation, and many other things, applying it to read in our data (McKinney & others, 2010). NumPy is an efficient numerical computation library that first store data into arrays (Van Der Walt, Colbert, & Varoquaux, 2011). Pandas and NumPy are then part of the larger SciPy ecosystem, open source Python libraries for scientific computing (Jones, Oliphant, Peterson, & others, 2001). There are many existing analytic tools which are good for implementing data science techniques. Table 2.1 summarizes the pros and cons offered by these analytic tools.

Table 2.1: Comparison between analytical tools based on highlighted features

Features	**Python**	**R**	**RapidMiner**	**Weka**
Software License	Free	Free	Free	Free
Type	Programming Language	Programming Language	GUI	GUI
Ease of Learning	Moderate	Difficult	Easy	Easy
Example of some Data Preprocessing Package/Library/Operator	*pandas, numpy, matplotlib, Tensorflow, Keras*	*dplyr, tidyverse*	*Filter Examples, Filter Attributes*	all functions under *Preprocess* tag
Example of some Outlier Detection Algorithm Package/Library/Operator	PyOD	OutlierDetection	k-NN, Detect Outlier (LOF)	all functions under *Classifiers* tag
IDE	Jupyter Notebook, Pycharm, Visual Studio	RStudio	-	-

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This section describes data science project lifecycle, including the proposed approach to achieve the objectives with the description and justification of the used techniques. First, in an ideal environment, every successful project must go through a data science lifecycle starting with data collection, model development, model evaluation, and results presentation (Donoho, 2017). Second, adopting this life cycle could determine the project's success and having useful outcomes (Zumel, Mount, & Porzak, 2014). Finally, the proposed approach consists of four main stages, as illustrated in Figure 3.1 below with the proposed mains tasks in each stage.

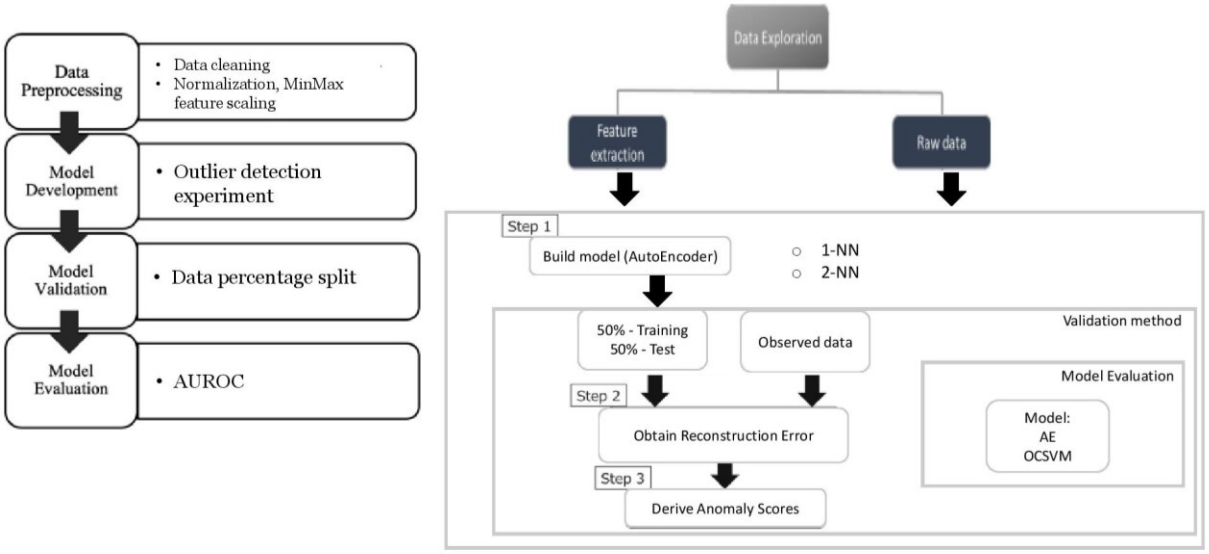


Figure 3.1: Methodology Frameworks for Data Science Activities.

3.1.1 Activity Plan and Gantt Chart

Data Science Activities Gantt Chart

Gantt Chart Template © 2006-2018 by Vertex42.com

Chee Sai Wai

Project Start Date: 4/8/2021 (Thursday) | Display Week: 2

WBS	TASK	START	END	DAYS	% DONE	WORK DAYS
1	**Data Exploration and Preparation**		-			-
1.1	Import libraries about read and manipulate data	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.2	Import libraries about visualisation	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.3	Import libraries about ML	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.4	Connect data to git repository and products	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.5	Data gathering	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.6	Explore context about data generation process	Thu 08/4/21	Wed 14/4/21	7	100%	5
2	**Data Representation and Transformation**		-			-
2.1	Undersample CO2 flux dataset	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.2	Isolate features from labels	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.3	Handle missing and inconsistent data	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.4	Visualize clusters in high-dimensional data	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.5	Handle instances on CO2 flux dataset with redundant columns	Thu 4/29/21	Wed 5/05/21	7	100%	5
2.6	Rename features into the lower case	Thu 4/29/21	Wed 5/05/21	7	100%	5
2.7	Generate statistical based class labels used for AUROC metrics	Thu 4/29/21	Wed 5/05/21	7	100%	5
3	**Computing with Data**		-			-
3.1	Rescale features by MinMax	Thu 4/29/21	Mon 5/10/21	12	100%	8
3.2	Rescale features by Standard	Thu 4/29/21	Mon 5/10/21	12	100%	8
3.3	Write both normal and labeled CO2 flux dataset by random splitting, used for training and testing purpose	Thu 4/29/21	Mon 5/10/21	12	100%	8
3.4	Store both normal and labeled CO2 flux dataset into readable format	Thu 4/29/21	Mon 5/10/21	12	100%	8
4	**Data Modeling**		-			-
4.1	Feed the input data to train autoencoders	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.2	Blend autoencoder architectures and hyperparameters (e.g. number of layer)	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.3	Feed regularizer in the encoding layer avoiding overfitting (e.g. l2 regularizer)	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.4	Plot autoencoder model	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.5	Fit the autoencoder model to reconstruct input	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.6	Learn AutoEncoder training history (e.g training and test curves)	Wed 5/26/21	Wed 6/09/21	15	100%	11

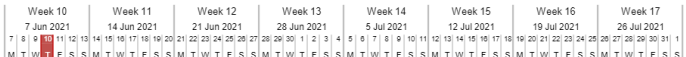
Data Science Activities Gantt Chart

Gantt Chart Template © 2006-2018 by Vertex42.com

Chee Sai Wai

Project Start Date: 4/8/2021 (Thursday) | Display Week: 10

WBS	TASK	START	END	DAYS	% DONE	WORK DAYS
1	**Data Exploration and Preparation**		-			-
1.1	Import libraries about read and manipulate data	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.2	Import libraries about visualisation	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.3	Import libraries about ML	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.4	Connect data to git repository and products	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.5	Data gathering	Thu 08/4/21	Wed 14/4/21	7	100%	5
1.6	Explore context about data generation process	Thu 08/4/21	Wed 14/4/21	7	100%	5
2	**Data Representation and Transformation**		-			-
2.1	Undersample CO2 flux dataset	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.2	Isolate features from labels	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.3	Handle missing and inconsistent data	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.4	Visualize clusters in high-dimensional data	Fri 4/16/21	Tue 4/27/21	12	100%	8
2.5	Handle instances on CO2 flux dataset with redundant columns	Thu 4/29/21	Wed 5/05/21	7	100%	5
2.6	Rename features into the lower case	Thu 4/29/21	Wed 5/05/21	7	100%	5
2.7	Generate statistical based class labels used for AUROC metrics	Thu 4/29/21	Wed 5/05/21	7	100%	5
3	**Computing with Data**		-			-
3.1	Rescale features by MinMax	Thu 4/29/21	Mon 5/10/21	12	100%	8
3.2	Rescale features by Standard	Thu 4/29/21	Mon 5/10/21	12	100%	8
3.3	Write both normal and labeled CO2 flux dataset by random splitting, used for training and testing purpose	Thu 4/29/21	Mon 5/10/21	12	100%	8
3.4	Store both normal and labeled CO2 flux dataset into readable format	Thu 4/29/21	Mon 5/10/21	12	100%	8
4	**Data Modeling**		-			-
4.1	Feed the input data to train autoencoders	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.2	Blend autoencoder architectures and hyperparameters (e.g. number of layer)	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.3	Feed regularizer in the encoding layer avoiding overfitting (e.g. l2 regularizer)	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.4	Plot autoencoder model	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.5	Fit the autoencoder model to reconstruct input	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.6	Learn AutoEncoder training history (e.g training and test curves)	Wed 5/26/21	Wed 6/09/21	15	100%	11
4.7	Augment percentage of data to be used for validation and ability to generalize	Fri 6/11/21	Thu 6/17/21	7	100%	5
4.8	Learn calculation about mean square error and reconstruction threshold	Fri 6/11/21	Thu 6/17/21	7	100%	5
4.9	Locate number of outliers sample of the CO2 flux dataset	Fri 6/11/21	Thu 6/17/21	7	100%	5
4.10	Build the OCSVM and fit the training data by labeled CO2 flux dataset	Fri 6/11/21	Thu 6/17/21	7	100%	5
4.11	Build the autoencoders and fit the training data by labeled CO2 flux dataset	Fri 6/11/21	Thu 6/17/21	7	100%	5
4.12	Analyze and compare unsupervised AE & OCSVM models with AUROC metrics	Fri 6/11/21	Thu 6/17/21	7	100%	5
5	**Data Visualization and Presentation**		-			-
5.1	Generate log files well to represent average trials on experiments	Mon 6/21/21	Sun 6/27/21	7	100%	5
5.2	Locate log files onto Jupyter extension by gaining context in Tensorboard	Mon 6/21/21	Sun 6/27/21	7	100%	5



3.2 Data Exploration and Preparation

All the measurements collected by sensors from EC station at CEMACS since 2015 until now can be accessed from this link: https://atmosfera.usm.my/api.html. The sample dataset was used in this research since fluxes measured were from the tropical coastal ocean surface (Yusup, Alkarkhi, Kayode, & Alqaraghuli, 2018), where data duration was obtained from 2019-01-01 until 2019-12-31 (1 years' data). In addition, for each instance in the data as illustrated in Table 3.1 below, the number of columns depends on the microclimate variables in each series. To complement the eddy covariance system, "Biomet" system of slow-response sensors were used for the scope of this study. The daily averages of the microclimate variables were stored by factory-calibrated instruments in a specific units: 1) atmospheric temperature (TA) used is indicating accuracy of ± 0.7 °C; 2) relative humidity (RH) used is indicating accuracy of ± 1.7 % for examples. In total there are 14 features and 17,511 observations and extracted is in the format of JavaScript Object Notation (json). After each instance in the dataset, there is an invalid value presented, which is the '-9999'. Regarding this, domain expert reported that this happens when the network interruption and therefore the sensors missed any measurements as well as many missing values. Furthermore, some data points have unique values and there are also between 0 and 2, as those were mainly reported with '0' as good quality score, while '1' as intermediate quality scores. Thus, it was used when these high values and immediately rejected due to this flagged codes as '2', therefore, any instance has 3868 counts was considered as anomaly. As reported by domain experts, the labels are in outlier context by taking the interquartile range values on normal data instances.

Table 3.1: Dataset Attributes Description

Features	Description	Type
date	Time since Carbon dioxide flux and the Monsoons on coastal ocean surface	hierarchical
wind_speed	Wind speed measure between 1 to 12 Months	numerical
wind_dir	Wind direction measure between 1 to 12 Months	numerical
ta_1_1_1	Daily averaged atmospheric temperature	numerical
ts_1_1_1	Daily averaged sea water temperature	numerical
rh_1_1_1	Relative humidity (RH)	numerical
rn_1_1_1	Several measures of variation in net radiation	numerical
rg_1_1_1	Several measures of variation in global radiation	numerical

p_rain_1_1_1	Precipitation events in Monsoon	numerical
ppfd_1_1_1	Photosynthetic active radiation (PAR)	numerical
le	Latent heat	numerical
h	Sensible heat	numerical
co2_flux	Flux uptakes on above coastal ocean surface	numerical
qc_co2_flux	Quality control test on CO2 flux footprint; flagged as low-quality as code "2"	categorical

3.3 Data Representation and Transformation

We removed all the instances that contains "-9999" value since the proportion of it is not large. In addition, since we focus on univariate outlier detection, particularly CO_2 flux data, so we picked the *date* and *co2_flux* attributes only involve in our dataset for further analysis. Dataset is filtered based on 2 attributes, namely *wind_dir* and *qc_co2_flux*. For *wind_dir*, we included those carbon dioxide (CO_2) flux greater than 315 degree and less than 45 degree. This is because the flux captured by sensor outside the 2 mentioned ranges will most probably have affected by wind direction and thus resulting from land, instead of ocean flux. For *qc_co2_flux* which defined quality control of the CO_2 flux, we removed all those "2" value which denoted "bad" quality data as requested by client. In addition to the redundant items from attributes, *p_rain_1_1_1*, *date* and *qc_co2_flux* were dropped.

In real word applications, data is rarely follows normal and Gaussian distribution, and thus traditional way 3 sigma outlier detection cannot be applied. According to Brownlee (2018), a good statistic for summarizing non-normal data could be the interquartile range (IQR). The IQR statistic can be interpret as how spread the values are in a set of observations, and can be calculated as:

$$IQR = Q3 - Q1 \tag{3.1}$$

where Q3 and Q1 denoted third quartile and first quartile in a dataset, respectively. From there, data point can be treat as outlier if it falls beyond Q1 – k*IQR, or above Q3 + k*IQR, where k = {1.5, 3, 6} is a constant factor. In our computation, we used k = 3 which is a widely used value in statistical context.

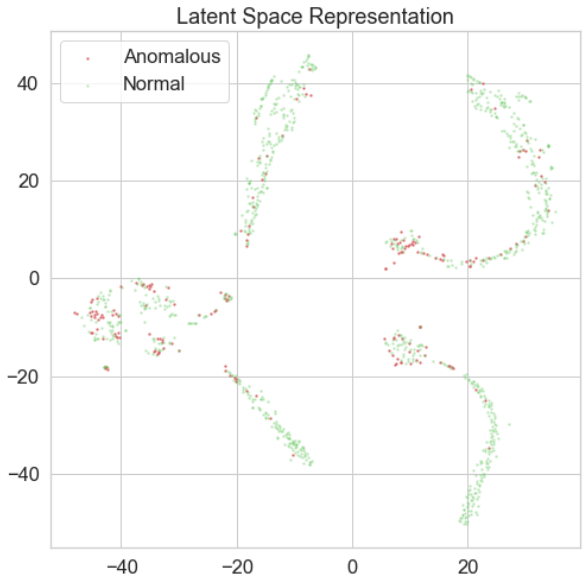


Figure 3.2: Visualizing clusters in high dimensional flux data

t-SNE is a dimensionality reduction technique used for visualisations of complex datasets, according to Linderman and Steinerberger (2017). It maps clusters in high-dimensional data to a two or three dimensional plane so we can get an idea of how easy it will be to discriminate between classes. Undersampling the normal in the flux data denoted small subsamples with the goal of simplifying the identification of clusters. The normal dataset has been undersampled from 1,859 instances to 935 instances. Observation from Figure 3.2, these models tend to reproduce anomalies from normal observations with high performance. Thus, these models discriminate samples by calculating a residual error between normal and later these instances will be dropped since they were considered anomalies.

Once identified all the outliers in the flux data, we added an attribute class called *label*, where "0" denoted inlier, while "1" denoted outlier. Preprocessing tasks resulted in a ready and clean dataset while it affected the number of the instances since some instances were dropped, Normal data sets was described as dimension measurement with 1859 training examples, whereas mix of that for normal and anomalous samples were directly applicable in 2046 training samples.

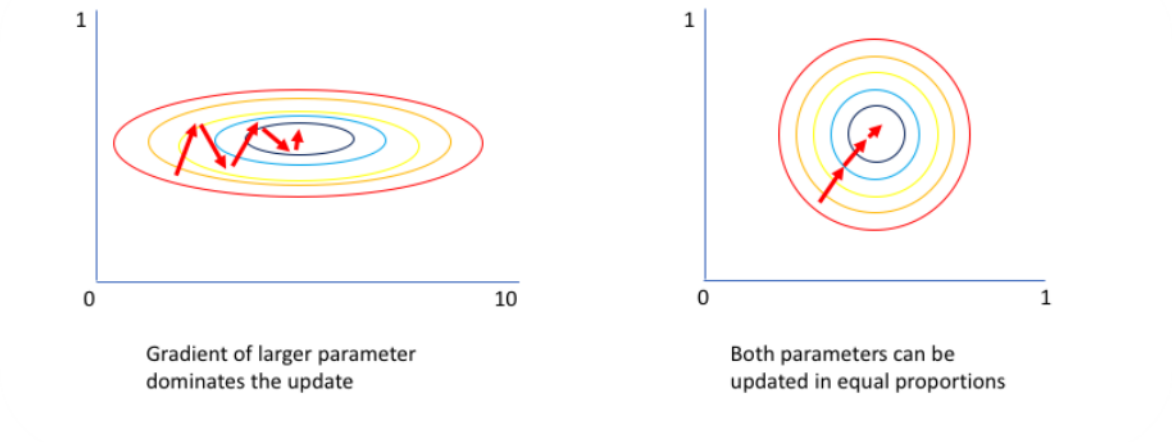


Figure 3.3: Normalizing and centering data for feature space

Other preprocessing techniques such as standard scaling or MinMax scaling since it will be used in dimensional space. One of the main advantages were monitored by centering and normalising the data for numerical input within [0, 1]. This is because different data types were computed as more weight can be converged well during training gradient descent in equal proportions. In this way, we can test and evaluate whether our model can generalize well to new, unseen data points. Finally, the training set of two parts was further applied— a standard labeled training set and another set of labeled examples. A validation set was constructed by taking examples from both the parts. Half of it were taken from the standard set (930 per class) and half from the extra set (1032 per class), a total of 1962 samples. Labels for the test set for both dataset with the help of the *scikit-learn* with the domain expert were confirmed and this was complimented with Github repository. The content downloaded in CSV format files from reproducing these results can be obtained from https://github.com/minajs9918/cds502/tree/DL4CDS590.

3.4 Deep Anomaly Detection Experiments

A deep learning model were built based on the prepared data using multiple anomaly detection algorithms and were compared their performance to pick the best one. The selected algorithms are AE and OCSVM. We ran each of these selected algorithms multiple times with many parameters' tunings and then used regularization, learning rate and number of layers method on the best models. The implementation is done in python 3.6 with KERAS which uses an end-to-end machine learning platform called TENSORFLOW at backend.

As in the review of related works (Alain & Bengio, 2014; Albahar & Binsawad, 2020; Jordan, 2017), regularization are used widely to overcome overfitting. Specifically, the authors in (Albahar & Binsawad, 2020) reported that this type of learning is suitable to solve such a problem. For each selected algorithm during the training phase, entire features were taken from the training set, L2 weight regularization is used to further reduce inefficient learning and each layer used dropout set to a value of 0.5. The encoder and decoder are composed of four layers with an encoding dimension of 6 units. The number of units is halved at each subsequent layer in the encoder, with the inverse being true for the decoder. A total of 11 input and output units are used to learn data representation as well as a validation curve plotted in Figure 3.4. A hyperbolic tangent activation was adopted as an activation function for output and a ReLU was for latent layer. Each activation was also used in equations 3.2 and 3.3. The loss function used is a mean square error (MSE) for 100 epochs of stochastic gradient descent (SGD).

$$ReLU = max(x, 0) \tag{3.2}$$

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.3}$$

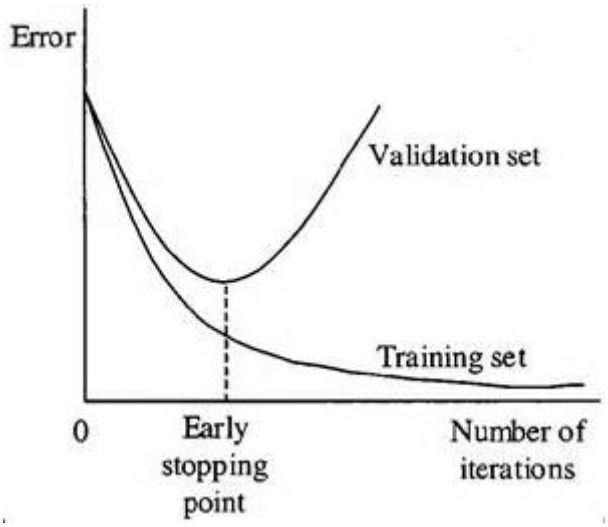


Figure 3.4: Validation curves for early stopping conditions

An acceptable number of neurons to use in the hidden layers between the size of the input layer and the size of the output layer were included. First problem, as stated by the authors in (Heaton,

2017) is when the training data is sufficient. Second, Heaton (2017) used in the problem when the neural network has so much information processing capacity. Since the final goal is to ensure picking good hyperparameters, therefore, a good learning rate good overall improvements in the objective function. For this reason, we used rate of 0.01 or 0.001 as stated by the authors (Le, 2015), which means a better choice when the learning rate is chosen poorly. Moreover, a choice regarding batch size depends on the computation power available. For example, 10 training elements were selected to the next training size element and also updated training set size of 100 elements. Once ten times has been used, the iteration is considered complete.

Although most of the authors agreed on regularization method, early stopping and dropout are used widely to overcome overfitting. The idea behind this is to have even after the number of layers on each hidden layer has been minimized. If the method performs badly on the training set then, Kazak (2018) proved early stopping for optimising computational time. Furthermore, the dropout is as stated in (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014), increasing the generalization performance of models. We have set a name for our model to be saved as an HDF5 file (Koziol, Robinson, & others, 2018). This is an efficient format to store our model in for IO purposes. Our early stopper will make sure that we stop training if our validation loss does not improve with a patience of ten epochs. The model checkpointer will save our best model as we train. The model will be fit with the centered and standardized data. Again, we will use a random 10% of the training data as a validation set for each epoch. After running this code, the output will look very similar to the output from the previous section. Finally, we load the best model that we got during training, and see how well it performs on the training and test sets. At the training stages, we considered contamination parameter of c=0.09, and then fitting OCSVM model. We imported all of the log files by use of the % character in "%load_ext tensorboard.notebook". Thus, we got graphical insights with Tensorboard while the autoencoder models still have experiments with other hyperparameters.

It is important to tune each algorithm by varying its key parameter value so that we can get an optimal and reliable result by averaging them. Table 3.2 summarized all the parameters tuning for the algorithms during implementation using Python.

Table 3.2: Summary of key parameters

Hyperparameter	Value
Number of layers	Single layer (11-6-11) / Stacked (11-8-6-8-11)
Batch size	10
Learning rate	0.01 / 0.001
Dropout	ReLU (second and fourth layers) / None (other layers)

Table 3.3: Network Architecture Details

Architecture	Input layer	Hidden layer	Learning Rate (LR)
AE1	11	6	0.01
AE2	11	6	0.001
AE3	11	8,6	0.01
AE4	11	8,6	0.001
AE5	11	Dropout(p=0.5),8, Dropout(p=0.5),6	0.01
AE6	11	Dropout(p=0.5),8, Dropout(p=0.5),6	0.001

3.5 Model Evaluation

The first metric is reconstruction loss plot of higher reconstruction test data points in dimensions. This metric enables evaluating the model by diagramming a reconstruction error of each outlier in the subspace against data point index. To complement this, we need to define a threshold for each implementation and iteratively apply the set threshold from the first rank until last rank. Also, the results are the product of averaged metrics obtained in all trials. For the evaluation purpose, Accuracy (ACC), Precision (P), Recall (R) and F1-measure (F1) metrics are applied. These metrics are calculated as follows:

• **Recall (R)**: quantifies the number of positive class predictions made out of all positive examples in the dataset. We want a high R value:

$$Recall = \frac{TP}{TP+FN} \tag{3.4}$$

• **Precision (P)**: indicates the proportion of correct predictions of anomalies divided by the total of predicted anomalies in the testing process. The higher P then the lower false alarm is:

$$Precision = \frac{TP}{TP + FP} \tag{3.5}$$

• **Accuracy (ACC)**: indicates the proportion of correct classifications of the total records in the testing set:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.6}$$

• **F-score (F1)**: Provides a single score that balances both the concerns of precision and recall in one number:

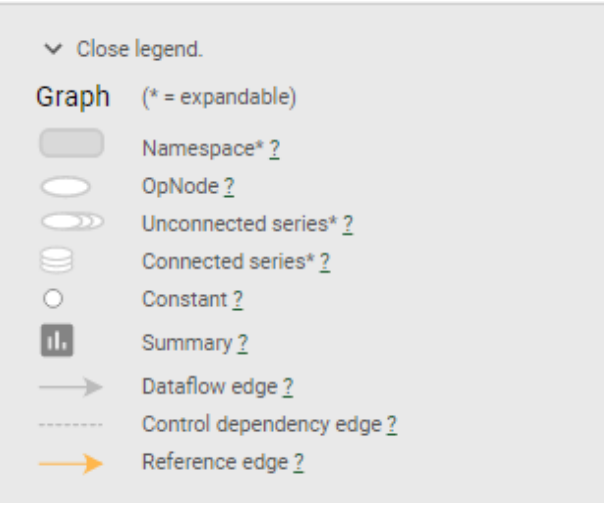
$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} \tag{3.7}$$

After training the AE, we calculated the MSE of the test data and estimated the contributing dimensions if the MSE exceeded the threshold. The threshold of the MSE was set to $\mu_{mse}+3\sigma_{mse}$, where μ_{mse} and σ_{mse} were the mean value and standard deviation of the MSEs with training data, respectively. As discussed in (Ikeda, Ishibashi, & Nakano, n.d.), we proposed two unsupervised thresholds for data reconstruction of anomaly detector. One estimates contributing reconstruction error to anomaly scores detected using AEs to localize the anomalies in flux data. While the other is modified Z-score, proposed method has used Median Absolute Deviation (MAD) by taking the score of 3.5 as our cut-off value. This means that every point with a score above 3.5 will be considered an outlier. As discussed in (Iglewicz & Hoaglin, 1993), we evaluated the two unsupervised thresholds including real measured data and validated accuracy of our AEs for detecting anomalies. Therefore, the higher the value of ROC, the better the algorithm since it is able to assign correct score most of the time. The second metric is ROC (Receiver Operating Characteristics) scores using the open source PyOD toolkit. Once anomaly detector on a training set has been fitted, decision function can generate raw outlier scores for unseen data. ROC were evaluated using *evaluate_print* in the library for quick performance evaluation.

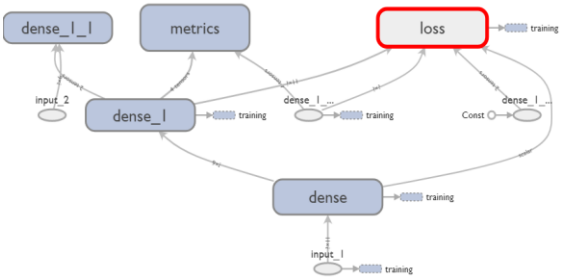
3.6 Data Presentation Tools for Accessing the Autoencoders

For this project, the version used is the latest stable one available 2.0.0-alpha, and all the scripts and models have been built with the Google Colab. Further details of the implementation will be specified when necessary in the following. Inside the version has not been given locally, some library dependencies made it not possible to sync once upgrading Tensorflow. Therefore, we proposed notebook experience such as Google Colab or cloud environment.

Another experience success is the TensorBoard visualisation tool, which allows users to visualise computation graphs and summary statistics for a better understanding of the model developed and the training process. Some internal features allowed for the creation of a model tracing profile, allowing the user to track the device placement of each node, node dependencies, and calculation time. A whole graph with interconnected nodes represented each of the operations declared in the code. There are also a variety of colouring choices, so nodes can be coloured based on their structure or the device on which they are put, as in Figure 3.5 show.



(a) Graph and color device legends



(b) Example of encoding and decoding dense layer device placement

Figure 3.5: Tensorboard screenshots for computation graphs

TensorBoard provides real-time monitoring of the running model, as well as a dashboard for tracking the weights in the various layers during training. Dashboards such as scalars, graphs, histograms, and others can now be viewed as in Figure 3.6 shows. By issuing the same command, the same TensorBoard backend is used. A new instance of TensorBoard would be launched if a different logs directory was selected. TensorBoard reads tensors and metadata from the logs of tensorflow projects. The path to the log directory is specified with log_dir below:

```
%tensorboard --logdir logs
```

```
from tensorboard import notebook
notebook.list() # View open TensorBoard instances
```

```
# Control TensorBoard display. If no port is provided,
# the most recently launched TensorBoard is used
notebook.display(port=6006, height=1000)
```

For this project, we will be using /logs/fit/. To load the data into Tensorboard, we need to save a training checkpoint to that directory, along with metadata that allows for visualization of a specific layer of interest in the model. The management of ports is automated. Start training a new model and see TensorBoard update automatically every 30 seconds in these instances.

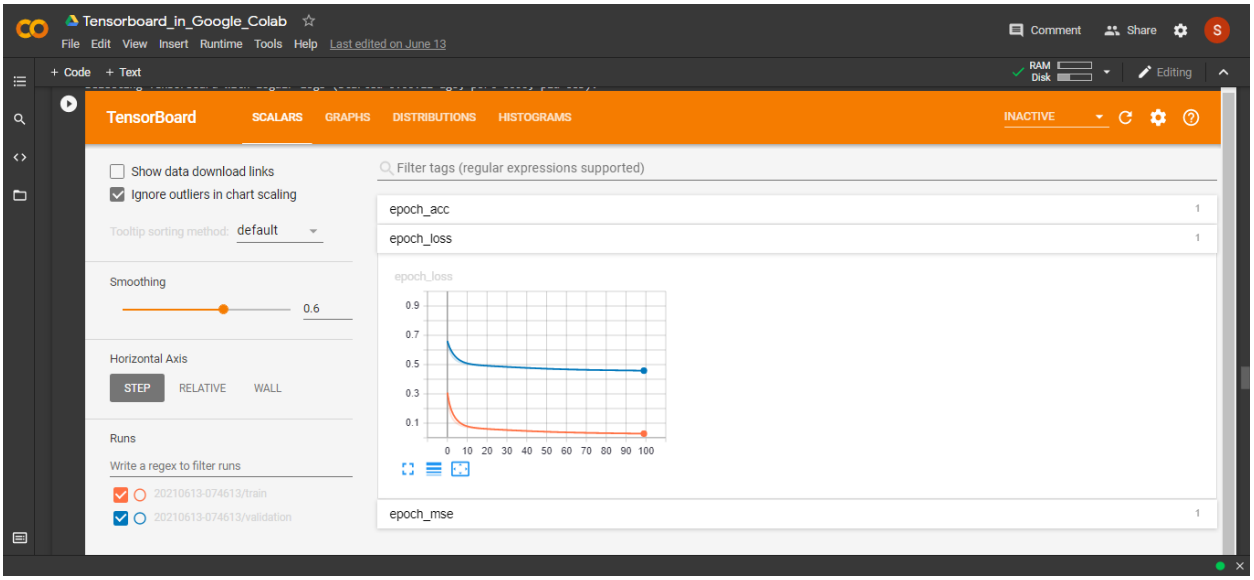


Figure 3.6: Tensorboard screenshots for various layers during training

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In most of our experiments we use the single layer autoencoder and stacked autoencoder. In the following we present the analysis as the performance metric. Among these are reconstruction plot, evaluation of ROC/overfitting, threshold robustness to anomaly records in our experiments.

4.2 Interpretability of reconstruction error based approach

As a measure that combines reconstruction error and threshold methods, outlier scores are considered where 0 implies that no outliers are detected and 1 indicates that all, and only, the outliers are detected. This study indicated the 930 instances closer to the index value of data point, and from that starting index we chose 100 number of instances as the majority true labels (good data/spikes) of individual data points in subspace. This behavior allows us to detect anomalies by setting a threshold for the reconstruction error. If a data sample has a reconstruction error higher than the preset threshold then the sample is classified as an anomalous CO_2 flux. Otherwise, it is classified as normal CO_2 flux.

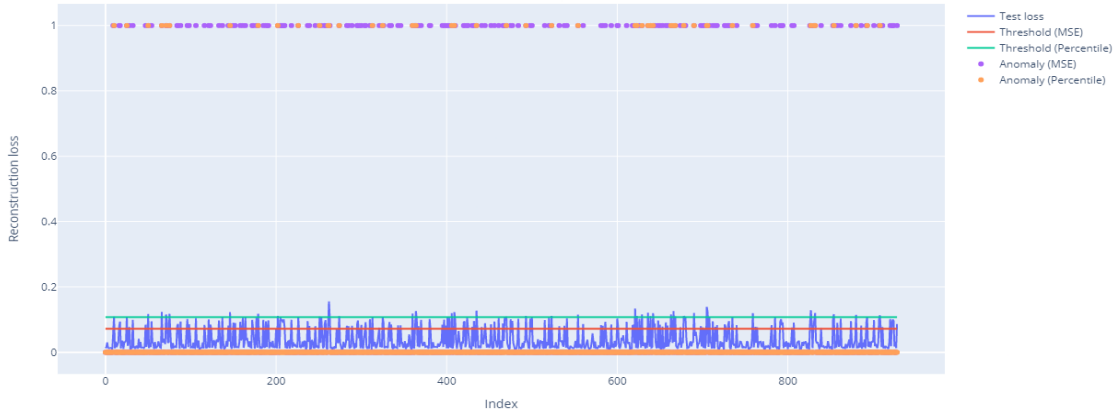


Figure 3.7 (a): Testing reconstruction errors of the testing data points along with MSE and percentile threshold

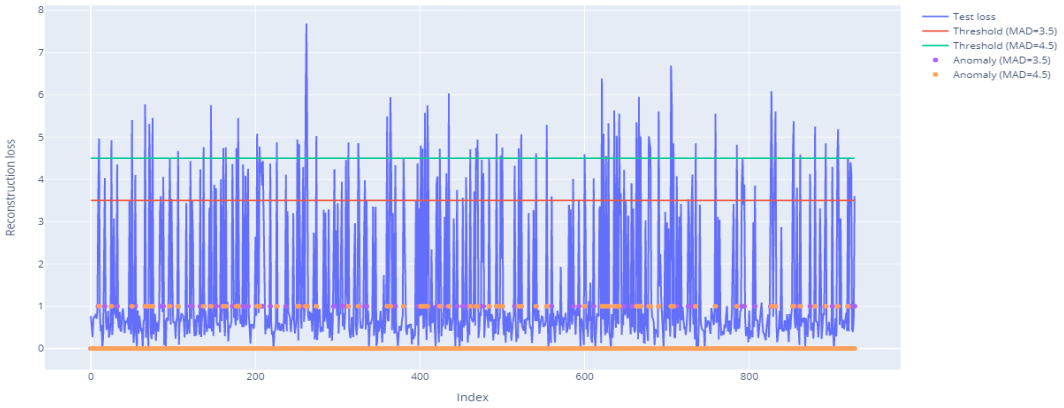


Figure 3.7 (b): Testing reconstruction errors of the testing data points along with different MAD's z-score threshold

In the first scenario, illustrative examples of the data reconstruction procedure are depicted in Figure 3.7 (a) and 3.7 (b) and implemented using Plotly provided in the Python 3.6 package. For any given threshold value, the ability of the deep learning models was assessed by comparing the number of reconstruction spikes while, at the same time, keeping decisions of the types false negative (FN) and false positive (FP) at the lowest levels possible.

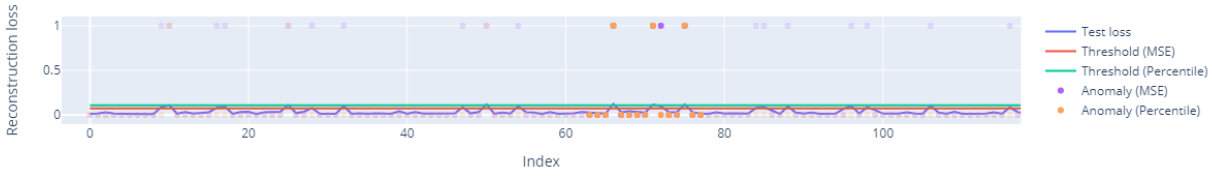


Figure 3.8 (a): The wrongly labeled data points outlying in subspace [65,66,67] and [74,75,76]

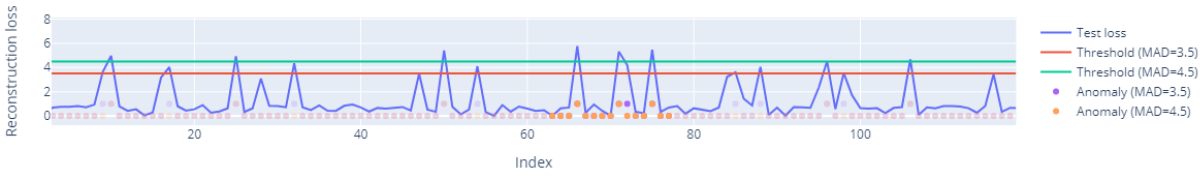


Figure 3.8 (b): The wrongly labeled data points outlying in subspace [70,71,72,73]

In the second scenario instead, illustrative examples of the 100 data points in dimensions are depicted in Figure 3.8 (a) and 3.8 (b). The AE 1 model as with a threshold of 0.0657 then the error offset is labeled as MSE. As with the threshold of 0.0993 then given the error offset is labeled as percentile. As with the MAD's z-score, the wanted low reconstruction error of individual data points obtained a threshold of 3.5, indicating that error offset was labeled as MAD=3.5. Thus, other evaluation metrics with high threshold values of 4.5, indicating that error offset was labeled as MAD=4.5.

As we can see, the point 66 and point 75 implied a higher reconstruction error in dimensions [65,66,67] and [74,75,76] respectively, suggesting correctly that it is outlying in this subspace. Similarly, point 71, which is an outlier in subspace [70,71] and [72,73], affecting most of the variables measured as there is high reconstruction error in dimensions [70,71,72,73]. Point 72 also have a relative high reconstruction error in subspace if we set a MAD's z-score. This is a false positive, because point 72 does not show a clear outlier behavior in this subspace. As with the AE1 model, MAD's z-score of higher threshold values was clear winner when it comes to minimize the overall autoencoder reconstruction error.

4.3 Evaluation of threshold robustness

To aid in comparison, we summarized the results of different threshold values and number of anomaly records in terms of ACC, P, R, and F1. As we can see, with a higher threshold, we get a higher detection ACC. We noticed that single layer autoencoder with learning rate of 0.01 can outperform than those of greater learning rate. Even with a recall increase, this model tends to perform better as 66% identified by the percentile method, while MAD's z-score achieved perfect recall. As before, the appropriateness of the selected model resulted more unseen data points and therefore the importance of generalization.

With stacked autoencoder trained on 930 normal samples, increasing layers and smaller learning rate achieved 47 detected anomalies in percentile-based threshold and 178 detected anomalies in MSE-based threshold. While preserving the recall of 100% with MAD's z-score, however, the other metrics drop with high threshold values. The reason for this trend is that percentage of CO_2 flux data having number of normal records falsely identified will increase then also 19.16%

detected anomalies with both MSE and a lower threshold of MAD's z-score. Problems can also be caused as detecting all anomalies for an increase in falsely identified normal data points.

Table 3.4: Evaluation of anomaly detection performance for both MSE and percentile threshold using autoencoder architectures AE 1 - AE 6

Model	**MSE Anomalies (#)**	**MSE Anomalies (%)**	**Percentile Anomalies (#)**	**Percentile Anomalies (%)**
AE1 (LR=0.01) {11,6}	190	0.2045	*47*	*0.0506*
AE2 (LR=0.001) {11,6}	200	0.2151	*47*	*0.0505*
AE3 (LR=0.01) {11,8,6}	221	0.2376	*47*	*0.0505*
AE4 (LR=0.001) {11,8,6}	178	0.1916	*47*	*0.0506*
AE5 (LR=0.01) + Dropout {11,8,6}	213	0.2290	*47*	*0.0505*
AE6 (LR=0.001) + Dropout {11,8,6}	193	0.2075	*47*	*0.0505*

Among a set of MSE specifications, we observed 47 anomalies with percentile based threshold not being the best neither the worst model. Given the 190 anomalies with MSE was 5.06% in AE 1 model, whereas the 200 anomalies with MSE was 5.05% in AE 2 model. This change can be expected because data points with a 5% greater than the error offset and scored better than 95% observations with MSE-based threshold. Although excellent in terms of precisions, percentile method suffers in terms of recall.

Table 3.5: Evaluation of anomaly detection performance for different MAD's z-score threshold using autoencoder architectures AE 1 - AE 6

Model	MAD=3.5 Anomalies (#)	MAD=3.5 Anomalies (%)	MAD=4.5 Anomalies (#)	MAD=4.5 Anomalies (%)
AE1 (LR=0.01) {11,6}	75	0.0807	*38*	*0.0409*
AE2 (LR=0.001) {11,6}	102	0.1097	*23*	*0.0247*
AE3 (LR=0.01) {11,8,6}	217	0.2333	*136*	*0.1462*
AE4 (LR=0.001) {11,8,6}	178	0.1916	*146*	*0.1572*
AE5 (LR=0.01) + Dropout {11,8,6}	212	0.2280	*147*	*0.1581*
AE6 (LR=0.001) + Dropout {11,8,6}	157	0.1688	*94*	*0.1011*

Compared with MAD's z-score, the single layer autoencoder with smaller learning rate obtained poor performance then the error offset are wrongly labeled as normal samples. There is our case when with learning rate of 0.001, resulting 102 detected anomalies in lower threshold values and resulting 23 detected anomalies in higher threshold values. Also, precision was not that high since AE 1 model can differ much from that of the AE 2 model. Therefore, AE 1 model achieved the best robust threshold given CO2 flux dataset due to more importance should be given to false negative. This could be possible since allowing actual case a long-lasting impact on subsequent percentage of measurement instruments having anomaly records that were correctly identified.

Table 3.6: Recall metrics comparison against autoencoder architectures AE 1 - AE 6

Model	Threshold	ACC (%)	P (%)	R (%)	F1 (%)
AE1 (LR=0.01) {11,6}	0.0657	0.80	**0.82**	**0.29**	0.44
	0.0993	0.94	**0.79**	**0.66**	0.72
	3.5000	0.91	**0.05**	**1.00**	0.10

	4.5000	0.91	**0.03**	**1.00**	0.06
AE2 (LR=0.001) {11,6}	0.1023	0.79	**0.73**	**0.27**	0.39
	0.1409	0.91	**0.64**	**0.55**	0.59
	3.5000	0.91	**0.11**	**1.00**	0.20
	4.5000	0.91	**0.08**	**1.00**	0.15
AE3 (LR=0.01) {11,8,6}	0.1044	0.80	**0.81**	**0.29**	0.43
	0.1429	0.92	**0.73**	**0.58**	0.65
	3.5000	0.91	**0.09**	**1.00**	0.17
	4.5000	0.91	**0.07**	**1.00**	0.13
AE4 (LR=0.001) {11,8,6}	0.0779	0.79	**0.79**	**0.27**	0.41
	0.1116	0.91	**0.70**	**0.54**	0.61
	3.5000	0.91	**0.11**	**1.00**	0.21
	4.5000	0.91	**0.09**	**1.00**	0.17
AE5 (LR=0.01) + Dropout {11,8,6}	0.1113	0.79	**0.79**	**0.27**	0.41
	0.1602	0.93	**0.70**	**0.62**	0.66
	3.5000	0.91	**0.08**	**1.00**	0.15
	4.5000	0.91	**0.06**	**1.00**	0.12
AE6 (LR=0.001) + Dropout {11,8,6}	0.1043	0.77	**0.76**	**0.26**	0.38
	0.1496	0.93	**0.61**	**0.65**	0.63

	3.5000	0.91	**0.09**	**1.00**	0.17
	4.5000	0.91	**0.08**	**1.00**	0.15

In addition to the previous outlined metrics, autoencoder with dropout achieved most of equal recall metrics, even though we also increase the false alarm rate. There is notably 94 detected anomalies with a higher threshold of MAD's z-score, whereas indicated the presence of a stacked autoencoder with dropout, reducing learning rate achieved 147 detected anomalies. There is our case when with learning rate of 0.01, resulting precision was low, in which some normal data points that were labeled as outliers.

Overall, MAD's z-scores reported relatively high recall, which is a wanted property in some anomaly detection tasks. Therefore, the remaining models achieved worst performance given MSE-based threshold as having some perfect precision rather than a higher detection recall. This is not the case when many other allowed for an increase in the false positives for a small increase of true positives.

4.4 Evaluation of AUC-ROC metric

Table 3.7: Comparative evaluation of the AUC-ROC metric against unsupervised anomaly detection models

Model	**AUC-ROC**			
Trial (#)	**1**	**2**	**3**	**Average**
AE1 (LR=0.01) {11,6}	0.8198	0.8379	0.8133	0.82
AE2 (LR=0.001) {11,6}	0.7806	0.7776	0.7848	0.78
AE3 (LR=0.01) {11,8,6}	0.7978	0.8044	0.8068	0.80

AE4 (LR=0.001) {11,8,6}	0.7817	0.7824	0.7772	0.78
AE5 (LR=0.01) + Dropout {11,8,6}	0.7792	0.7803	0.7770	0.78
AE6 (LR=0.001) + Dropout {11,8,6}	0.7775	0.7772	0.7777	0.78
OCSVM (nu=0.01)	0.7028	0.7028	0.7028	0.70

For an overall evaluation of the anomaly detection performance, we reported the best performing results based on parameterizations. For each technique, we described in the highest area under the ROC curve (AUC-ROC) using 1,032 instances of the training features for validation. We followed the procedure similar to single layer autoencoder, where we found the best performance with only 0.82 on average using the AE 1 model and then learning rate which had to be set to higher values. Overall, OCSVM achieved a fairly poor performance with only 0.70 on average. This might due to several reasons and one of them perhaps is the incorrect chosen outlier fraction for the implementation. Another reason is that this algorithm more likely not desirable in big data and high-dimensional anomaly detection tasks.

On average, AE 2, AE 4, AE 5 and AE 6 showed a medium to poor performance. We tried many network architectures and found that shallow models gave improvements in AUC-ROC over all of them. However, the improvement was not as significant as that for the AE 3 model with only 0.80 on average. Dropout with p = 0.5 was used in all the encoding layers. Even when no dropout with a small learning rate of 0.001 was used, we obtained AUC-ROC with only 0.78 on average. Each layer was trained on this data set to get the best results. It was important to use a higher learning rate of about 0.01. Thus, adding dropout did not give any improvements. Although this could imply longer computational times due to iterations involved, large p was more likely not to produce enough dropout to prevent overfitting.

4.5 Evaluation of MSE accuracy and loss outcome

To this end, we compared the alternative visualization of average of training accuracy, average of validation accuracy, average of training loss and average of validation loss. The initial training epochs resulted in a swift decrease in the reconstruction error for most of the algorithms. For each technique, we evaluated against the testing datasets more likely over/under fitting.

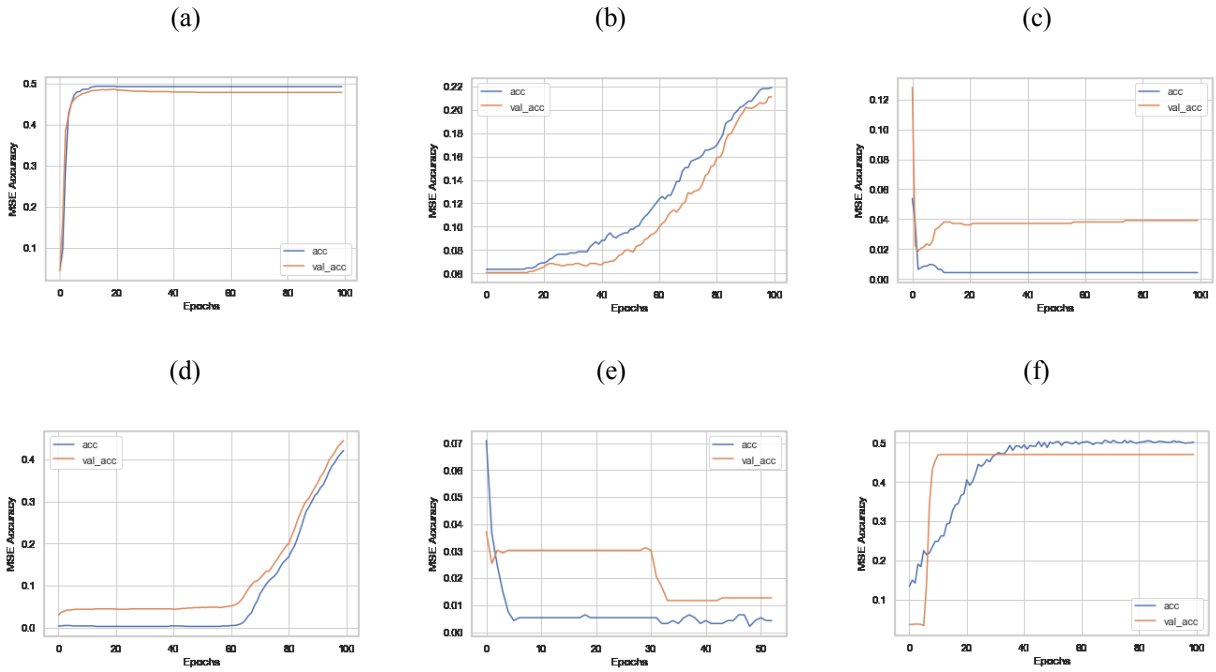


Figure 3.9: MSE accuracy curves for (a-b) Single layer autoencoders, (c-d) Stacked autoencoders and (e-f) Stacked autoencoders with dropout

The test set accuracy for the AE 1 model was 0.48 on the first 20 training epochs which is the highest validation accuracy achieved among all of our proposed models. While the accuracy in AE 2 model was higher 0.21, the loss in AE 1 was 0.16 compared to 0.56 in AE 2 model which is much less than AE 1 model. Given examples of CO_2 flux dataset, also our proposed stacked autoencoder with small learning rate achieved 0.42, which is the second highest validation accuracy achieved among the developed models.

AE 3 model does not generalize to new cases, suggesting we need to reduce the number of hidden layers. Furthermore, the graph on Figure 3.9 (c) is an evidence of overfitting – there is a

divergence between accuracy curves of train and test data after 10 epochs. Finally, regarding validation accuracy of stacked autoencoder with dropout, AE 5 model achieved an accuracy of 0.016, which is considered low compared to the stacked autoencoder was used without dropout.

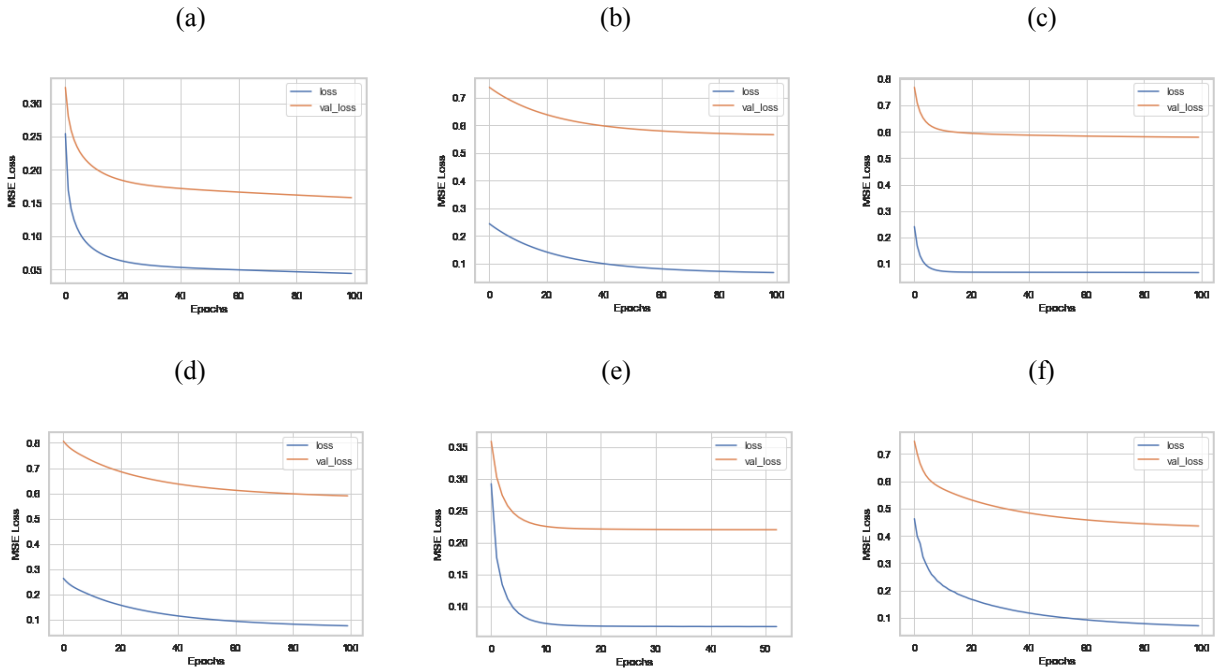


Figure 3.10: MSE loss curves for (a-b) Single layer autoencoders, (c-d) Stacked autoencoders and (e-f) Stacked autoencoders with dropout

After about 20 training epochs, the loss curve for AE 1 model is much smoother and sloping downward gradually. The loss curve on the graphs from Figure 3.10 (a) and 3.10 (f) indicates that the model copes with data in a more efficient way gradually reducing the loss. Once again, an augmented number of training epochs requires a thorough testing, although sometimes produces good results. The loss curve on the graphs from Figure 3.10 (c) and 3.10 (e) remains almost flat meaning that weight adjustment does not lead to a loss reduction. Moreover, a further confirmation showed a large learning rate is suitable for lower layers, the latter generated small changes in the parameters in higher layers. Meanwhile, a large learning rate not suitable for higher layers because great fluctuations in the objective function. There is no general agreement as limiting the layers to a short duration.

The limitation of this research was due to the limited hardware capability where increasing the number of hidden layers were previously specified for each model. Once fixed, hyperparameters could be investigated more to find the optimal deep learning model for this type of dataset. The problem with parameter updates is that the stacked autoencoders can be greatly affected by noise, and one alternative is to replace them with robust dropout rate. We reported test error in AE 6 model was 0.42 which is much less than 0.6 in AE 4 model. The introduction of the dropout regularizer equal to 50% in each layer of the stacked autoencoder illustrated on Figure 5.4 (d) reveals a smoother loss curve. Therefore, using a dropout network is similar to decision function to collect more evidence from other neurons.

4.6 Compare Results from related Literature

Most of the published work have attempted various unlabeled data come from the same distribution. In the work of (Schreyer, Sattarov, Borth, & Aug, n.d.), the best obtainable AUC-ROC was deepest architecture consisting of 17 hidden layers and ReLU activations. Comparing both the analysis for datasets, the AE 1 model otherwise accomplished highest AUC-ROC (0.82) in our first analysis. For deep autoencoder results, similar considerations were found, 136 less detected false positive anomalies; while for shallow autoencoder results, 38 less detected false positive anomalies. Therefore, increasing number of hidden layers results in faster error convergence and decreases the number of detected anomalies.

Consequently, more features have been required to feed models for better results in (Kazak, 2018), for dealing with Parkinson's telemonitoring data. The stacked AE yielded slightly better outcome compared to the architecture with the least hidden layers. That is, mean squared error provided better results by 106.51 (Single layer AE) than 70.14 (Stacked AE) – by 28.16% and 34.15%, respectively. However, single layer AE shown the best result in terms of metrics in our experiment. At the same time, the increase in epochs requires additional assessment for possible overfitting.

4.7 Challenges and Solutions

One of the challenges we faced perhaps is the unsupervised deep learning particularly in outlier detection context. This topic will become challenging especially domain experts deal with

unlabeled dataset. The research work in this area is still growing and automatic feature learning is developed for the outlier detection in various domains as there is a weak control environment around certain domain. One of the concerns is the fine-tuning for each algorithm. Unsupervised deep learning is different with machine learning context as there is choice of good hyperparameter between training and testing phase. We directly apply the loss function, optimizer, learning rate, and number of hidden layers in question by considering their data types of the instances. This implied that ad hoc tasks will give the optimal result and this more to expensive grid search. Another concern is that the chosen of performance metric to evaluate the algorithms as there are few metric available in previous student's final report. Hence, carefully study the past metrics and choose the candidates models in such a way to aid in comparison.

The second challenge is visualizing Tensorboard using saved model into Jupyter environment. This is because we run into higher installation package such as Tensorflow 2.0. Concerning the presentation for final report, the possible solution is to do simple briefing for what comes at the browser. This can be done as our role provided clients the sample source codes with zipped log files on the next trial. Therefore, it is tough choice if we let go the promise and one could put the alternative to this end.

4.8 How Practicum Experience Help Apply What We Learned in Class

Practicum aims at cultivating authentic data consultant as part of first act although our second act as student. In academic class, we only learned the basics which is enough to build new theories from professors in our chosen subject only. When come to practicum, it is the time for us to apply soft skills knowledge, such as value added communication; while for client to solve their problem, such as technical design with data science techniques, hard skills are considered.

For example, in this research we applied the revised version of deep anomaly detection in environmental science domain, which is quite new to us. Firstly, lab session was not much from that during academic lesson. Next, this domain knowledge is somehow not familiar as we did our specific coursework degree as before. The most interesting data science techniques can apply in environmental science domain and bring solution in the context. Once explored, practicum seems a good sign for student to carry out hands on and contribute what we gained in class to real world applications.

4.9 Observation Made during Practicum related to Professional and Operational Issues

From the ethnical standpoint, professional and operational issue we considered during practicum associated to quality control CO_2 flux labelling. Given the context of anomaly detection problems, client should indicate whether is having binary class or not in the analysis part. Depending on useful information by domain experts, the literature used for EC data can cause misleading criteria. For example, we might choose raw observation to relations with CO_2 flux. Then, user has no information about possible anomalies unless a further confirmation using box plot method. This incident easily occurs, it is therefore, easier to obtain data that reflects how the EC tower operates in normal state using similar datasets. Concerning real world scenario, data consultant tends to handle data lifecycle in a conservative manner to mitigate risks. The choice of data modelling procedure happens to exactly match Gantt diagram if one of its attributes was not reconstructed correctly.

CHAPTER 5

CONCLUSION & LESSON LEARNED

5.1 Introduction

This chapter summarized all results, findings, and lesson learned in this practicum project. In addition, some recommendations for future research are enumerated in Section 5.5. Finally, our opinion and thought regarding practicum experience integrated in DSA program is presented in Section 5.6.

5.2 Conclusion from Practicum Experience

In summary, based on the analyzed results we can conclude that AE 1 is the best outlier detector, meanwhile MAD's z-score is the most robust threshold. The analysis part stays the same as the decision threshold using AE 1 to monitor the incoming flux data. Here, we able to achieved the listed parameters, but not one measure as opposed to PyOD prerequisite. Data modelling procedure is mainly the deep part of the model which main models consider Keras library of research work. Generally, from my understanding and knowledge the training stage can be further split into validation when the call-back is the number of epoch complete training and loss optimization where convergence is considered. We employ PyOD toolkit including model around certain parameters like dropout rate or activation function in presenting the AUC-ROC scores to the end users. Regarding Keras library is like how to connect our model to Tensorflow backend, how we want to run our model to monitor the visualisation and all these things need to be consider in the data presentation. As previously stated, this bring the challenge to our thought as there is no least or most hidden layers. Therefore, it is difficult to achieve exhaustive search over specific parameter values within this limited practicum time. Yet, any practicum experience holds considerable importance in motivating the work from there.

5.3 Lesson Learned from Experience

There are few lessons learned from this practicum yet important insights: 1) technical interest, 2) domain relevance, and 3) project management. Technical interest is more about the technological impact on which the team worked to the individual or organization. Since this project is associated with deep anomaly detection, hence we need to assign method directly from data through a process of discovery, address a set of business challenges and needs, among others. After about projects delegated to other fields and sciences, the outcomes of this review from domain experts can be used to deep learning model since we choose the non-linear dimensionality reduction yet powerful outlier detector. Though exploring literature review can be done, we are able to know autoencoder used for detecting the anomaly flux data. This framework integrated successfully if we developed from an industry perspective. In addition, we also aware there is few existing statistical outlier method such as standard box plot.

Concerning Covid-19 pandemic, we can get into the virtual meetings by adopting data science technique into discussion, which can also benefit similar implementation in this research. Throughout a review of existing literature on analytics and data science skills with domain expert, we then proceed to present a discussion on EC data. Among these are attributed to capture and measure the reading, which sensor is used to capture particular parameter.

Despite a coursework in this practicum progress, we need sponsors but unfortunately we do not. Time management is important to allocate our time in particular task and that is why Gantt chart is playing the important role here. Client sometimes operate on the basis “I am not sure what I want, but I recognize it when I see it.” Care must be taken not to “gold-plate” their needs or assume we know best. At the beginning, we inserted the important milestone and make sure my client to understand where future goals are next. This effort with other expertise can cause confusion, meanwhile I restated my expertise so we can focus decent collaboration. Hence, needs or opportunities of problem might include, for example, use of new technology or introduction of a new process. Whatever the need, it must now turned into an image of how the project will look on completion.

5.4 Data Science Pipeline and Theories used in Practicum

The first stage is the data exploration and preparation. In this practicum, the sensor collected CO_2 flux footprint of various microclimate variables is stored in a dedicated database and the raw data at process-based level can be retrieved directly from a webpage.

The second stage is the data representation and transformation. Data cleaning involve how we remove and replace invalid records on the raw data itself, including how we treat the missing values and null values in our dataset.

Next stage is the computing with data, in which we conducted exploratory data analysis like labelling normal and abnormal flux data and check its scaling in the normal splitting for both training and test sets.

Modelling data is the core stage in overall process. In this stage, we can use deep learning models to learn more features about normal or anomalous data points in our dataset. Due to the nature of data we worked on, unsupervised shallow and deep autoencoders are more preferable in this research.

Lastly, interpreting the findings based on the results or visualisations. We used reconstruction plot, evaluation of ROC/overfitting, threshold robustness to anomaly records to evaluate anomaly detection performance. At the end, we manage to determine the good quality outlier detector for flux dataset.

5.5 Recommendations for Future Research

In this research, reconstruction based learning for anomaly detection consists of single layer autoencoders, stacked autoencoders while domain based learning, for instance, OC-SVM are considered. It is worthwhile by considering other way of research design in future research, including:

a) Other than neural network trained on dataset, there are variant types of autoencoders, for example variational autoencoder based anomaly detection using reconstruction probability and denoising autoencoder with dropout based anomaly detection, as recommended by An and Cho

(2015) and Mohamed (2019). Besides reconstruction based method, one-class neural network (OC-NN) is also recommended outlier detector similar to autoencoders for compressing the input data into a lower-dimensional space. Moreover, there could be combination of dimensionality reduction into a framework in the first step. The second step feeds some features into OCSVM in classification context, separating the normal observations from the anomalous ones. Among these are possible classification techniques to involve popular decision trees and random forest.

b) AUC-ROC metric is used to evaluate performance of algorithm in this research. Also, threshold robustness can be seen as a way of adding threshold values to increase false alarm rate. However, there are other hyperparameter tuning at the corners with multiple splitting of initial dataset, which led to improvement with evaluation metrics. Grid search will be chosen for further experiments, as recommended by Schmidt (2020). K-fold cross-validation allows the training dataset into multiple folds so that the results are less biased and not overfitted. These alternatives have not widely used in recent work yet due to the effect of network architecture on the overfitting issues. The provided Monte Carlo experiment for the work by Wang and Manning (2013) contained deterministic versions of dropout. These models further explored the idea of marginalizing dropout to speed-up training.

5.6 Practicum Experience integrated in the DSA Program

Based on my viewpoint, this practicum stays the same in future DSA program. This is because what we learned would be our survival kits as well as no contextual cues from real world application. Let's take an example, in academic class like assignment or practical class, the dataset is often ideally cleaned. This is not the case when big data as it normally makes our hands dirty and often in either semi structured or unstructured format. We learned in real life however we should gain more relationship among business stakeholders instead becoming a good data consultant. Hence, it is part of learning journey through this practicum course. To this end, I feel it is pretty dangerous to stay in only a career; getting to right people is far more important when you now make a living as Covid-19 lockdown.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … others. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv Preprint ArXiv:1603.04467*.

Aggarwal, C. C. (2017). An introduction to outlier analysis. In *Outlier analysis* (pp. 1–34). Springer.

Alain, G., & Bengio, Y. (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, *15*(1), 3563–3593.

Albahar, M. A., & Binsawad, M. (2020). Deep Autoencoders and Feedforward Networks Based on a New Regularization for Anomaly Detection. *Security and Communication Networks*, *2020*(Ml). https://doi.org/10.1155/2020/7086367

An, J., & Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, *2*(1), 1–18.

Andrews, J. T. A., Morton, E. J., & Griffin, L. D. (2016). Detecting anomalous data using auto-encoders. *International Journal of Machine Learning and Computing*, *6*(1), 21.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 93–104.

Brownlee, J. (2018, April). *How to Remove Outliers for Machine Learning*.

Burba, G., & Anderson, D. (2010). *A brief practical guide to eddy covariance flux measurements: principles and workflow examples for scientific and industrial applications*. Li-Cor Biosciences.

Campos, G. O., Zimek, A., Sander, J., Campello, R. J. G. B., Micenková, B., Schubert, E., … Houle, M. E. (2016). On the evaluation of unsupervised outlier detection: measures,

datasets, and an empirical study. *Data Mining and Knowledge Discovery*, *30*(4), 891–927.

Cao, L. (2015). Coupling learning of complex interactions. *Information Processing & Management*, *51*(2), 167–186.

Cao, L., Zhang, C., Philip, S. Y., & Zhao, Y. (2010). Knowledge Actionability. In *Domain Driven Data Mining* (pp. 75–91). Springer.

Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. *ArXiv*, 1–50.

Chalapathy, R., Menon, A. K., & Chawla, S. (2018). Anomaly detection using one-class neural networks. *ArXiv Preprint ArXiv:1802.06360*.

Chollet. (2015). *Deep Learning for humans*. Retrieved from https://github.com/keras-team/keras

Diaz, I., & Hollmen, J. (2002). Residual generation and visualization for understanding novel process conditions. *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, *3*, 2070–2075.

Doersch, C. (2016). Tutorial on variational autoencoders. *ArXiv Preprint ArXiv:1606.05908*.

Donoho, D. (2017). 50 years of data science. *Journal of Computational and Graphical Statistics*, *26*(4), 745–766.

Du, M., Liu, N., & Hu, X. (2019). Techniques for interpretable machine learning. *Communications of the ACM*, *63*(1), 68–77.

ElShawi, R., Sherif, Y., Al-Mallah, M., & Sakr, S. (2020). Interpretability in healthcare: A comparative study of local machine learning interpretability techniques. *Computational Intelligence*.

Erfani, S. M., Rajasegarar, S., Karunasekera, S., & Leckie, C. (2016). High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognition*, *58*, 121–134.

Erhan, L., Ndubuaku, M., Di Mauro, M., Song, W., Chen, M., Fortino, G., … Liotta, A. (2021). Smart anomaly detection in sensor systems: A multi-perspective review. *Information Fusion*, *67*, 64–79. https://doi.org/10.1016/j.inffus.2020.10.001

Evangelista, P. F., Embrechts, M. J., & Szymanski, B. K. (2006). Taming the curse of dimensionality in kernels and novelty detection. In *Applied soft computing technologies: The challenge of complexity* (pp. 425–438). Springer.

Foken, T., Aubinet, M., & Leuning, R. (2012). The eddy covariance method. In *Eddy covariance* (pp. 1–19). Springer.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 315–323.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Heaton, J. (2017, June). *The Number of Hidden Layers | Heaton Research*. Retrieved from https://www.heatonresearch.com/2017/06/01/hidden-layers.html

Himeur, Y., Ghanem, K., Alsalemi, A., Bensaali, F., & Amira, A. (2021). Artificial intelligence based anomaly detection of energy consumption in buildings: A review, current trends and new perspectives. *Applied Energy*, *287*(April), 1–41. https://doi.org/10.1016/j.apenergy.2021.116601

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Iglewicz, B., & Hoaglin, D. (1993). *Volume 16: how to detect and handle outliers, The ASQC basic references in quality control: statistical techniques, Edward F. Mykytka*. Ph. D., Editor.

Ikeda, Y., Ishibashi, K., & Nakano, Y. (n.d.). *Anomaly Detection and Interpretation using Multimodal Autoencoder and Sparse*. 1–20.

Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016). A deep learning approach for network intrusion detection system. *Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS)*, 21–26.

Jiang, X., Gao, J., Hong, X., & Cai, Z. (2014). Gaussian processes autoencoder for dimensionality reduction. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 62–73.

Jones, E., Oliphant, T., Peterson, P., & others. (2001). *SciPy: Open source scientific tools for Python*.

Jordan, J. (2017, July). *Deep neural networks: preventing overfitting.* Retrieved from https://www.jeremyjordan.me/deep-neural-networks-preventing-overfitting/

Jung, M., Reichstein, M., Margolis, H. A., Cescatti, A., Richardson, A. D., Arain, M. A., … others. (2011). Global patterns of land-atmosphere fluxes of carbon dioxide, latent heat, and sensible heat derived from eddy covariance, satellite, and meteorological observations. *Journal of Geophysical Research: Biogeosciences*, *116*(G3).

Kazak, V. (2018). *Unsupervised feature extraction with autoencoder: for the representation of parkinson' s disease patients*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *ArXiv Preprint ArXiv:1312.6114*.

Koziol, Q., Robinson, D., & others. (2018). *HDF5*.

Le, Q. V. (2015). *A Tutorial on Deep Learning Part 1 : Nonlinear Classifiers and The Backpropagation Algorithm*. 1–18.

Linderman, G. C., & Steinerberger, S. (2017). Clustering with T-SNE, provably. *ArXiv*, *1*, 1–15. https://doi.org/10.1137/18m1216134

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *6*(1), 1–39.

Luo, T., & Nagarajan, S. G. (2018). Distributed anomaly detection using autoencoder neural networks in wsn for iot. *2018 Ieee International Conference on Communications (Icc)*, 1–6.

Makhzani, A., & Frey, B. (2013). K-sparse autoencoders. *ArXiv Preprint ArXiv:1312.5663*.

McGillis, W. R., Edson, J. B., Zappa, C. J., Ware, J. D., McKenna, S. P., Terray, E. A., … others. (2004). Air-sea CO2 exchange in the equatorial Pacific. *Journal of Geophysical Research: Oceans*, *109*(C8).

McKinley, G. A., Follows, M. J., & Marshall, J. (2004). Mechanisms of air-sea CO2 flux variability in the equatorial Pacific and the North Atlantic. *Global Biogeochemical Cycles*, *18*(2).

McKinney, W., & others. (2010). Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, *445*, 51–56.

Mohamed, S. (2019). *Denoising Autoencoder with Dropout based Network Anomaly Detection*. (c), 98–103.

Nguyen, M.-N., & Vien, N. A. (2018). Scalable and interpretable one-class SVMs with deep learning and random fourier features. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 157–172.

Oliveira, D. F. N., Vismari, L. F., Nascimento, A. M., Almeida, J. R. de, Cugnasca, P. S., Camargo, J. B., … Neves, M. (2021). *A new interpretable unsupervised anomaly detection method based on residual explanation*. (4600043577). Retrieved from http://arxiv.org/abs/2103.07953

Pang, G., Shen, C., Cao, L., & Hengel, A. van den. (2020). Deep learning for anomaly detection: A review. *ArXiv Preprint ArXiv:2007.02500*.

Pang, G., Shen, C., & van den Hengel, A. (2019). Deep anomaly detection with deviation networks. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 353–362.

Pimentel, M. A. F., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014). A review of novelty detection. *Signal Processing*, *99*, 215–249. https://doi.org/10.1016/j.sigpro.2013.12.026

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., & Prabhat. (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, *566*(7743), 195–204. https://doi.org/10.1038/s41586-019-0912-1

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning*, 1278–1286.

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. *Icml*.

Roth, V. (2006). Kernel fisher discriminants for outlier detection. *Neural Computation*, *18*(4), 942–960.

Schmidt, M. (2020). *Autoencoder-Based Representation Learning to Predict Anomalies in Computer Networks*.

Schreyer, M., Sattarov, T., Borth, D., & Aug, L. G. (n.d.). *Detection of Anomalies in Large-Scale Accounting Data using Deep Autoencoder Networks*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22–30.

Vieira, A., & Ribeiro, B. (n.d.). *Introduction to Deep Learning Business Applications for Developers*.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a

local denoising criterion. *Journal of Machine Learning Research*, *11*(12).

Vitale, D., Fratini, G., Bilancia, M., Nicolini, G., Sabbatini, S., & Papale, D. (2020). A robust data cleaning procedure for eddy covariance flux measurements. *Biogeosciences*, *17*(6), 1367–1391.

Wang, S., & Manning, C. (2013). Fast dropout training. *International Conference on Machine Learning*, 118–126.

Wulsin, D., Blanco, J., Mani, R., & Litt, B. (2010). Semi-supervised anomaly detection for EEG waveforms using deep belief nets. *2010 Ninth International Conference on Machine Learning and Applications*, 436–441.

Xiao, J., Zhuang, Q., Baldocchi, D. D., Law, B. E., Richardson, A. D., Chen, J., … others. (2008). Estimation of net ecosystem carbon exchange for the conterminous United States by combining MODIS and AmeriFlux data. *Agricultural and Forest Meteorology*, *148*(11), 1827–1847.

Yahaya, S. W., Langensiepen, C., & Lotfi, A. (2018). Anomaly detection in activities of daily living using one-class support vector machine. *UK Workshop on Computational Intelligence*, 362–371.

Yusup, Y., Alkarkhi, A. F. M., Kayode, J. S., & Alqaraghuli, W. A. A. (2018). Statistical modeling the effects of microclimate variables on carbon dioxide flux at the tropical coastal ocean in the southern South China Sea. *Dynamics of Atmospheres and Oceans*, *84*(August), 10–21. https://doi.org/10.1016/j.dynatmoce.2018.08.002

Zimek, A., Schubert, E., & Kriegel, H.-P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *5*(5), 363–387.

Zumel, N., Mount, J., & Porzak, J. (2014). *Practical data science with R*. Manning Shelter Island, NY.

APPENDIX A

Log Book

Appendix A.1: Log Book – Meeting with Mentor

Index No. (CS-PIS records)	Date/ Time	Channel	Hour(s)
3	01-Apr-2021, 15:00	Webex & Mobile	1
7	08-Apr-2021, 15:00	Mobile	1
12	15-Apr-2021, 17:00	Mobile	1
17	22-Apr-2021, 17:00	Mobile	1
44	02-Jun-2021, 14:00	Mobile	1
52	14-Jun-2021, 14:00	Mobile	1
54	16-Jun-2021, 14:00	Mobile	1

Cumulative Contact Hours: 7 hours

Please login using industry ID or visit https://cspis.usm.my/

Appendix A.2: Log Book – Meeting with Supervisor

Index No. (CS-PIS records)	Date/ Time	Channel	Hour(s)
2	31-Mar-2021, 10:00	Mobile	0.5
29	11-May-2021, 10:00	Teams	1
53	15-Jun-2021, 11:30	Teams	1

Cumulative Contact Hours: 2.5 hours

Please login using staff ID or visit https://cspis.usm.my/

Dr. Mohd Nadhir Bin Ab Wahab
Senior Lecturer
School of Computer Sciences
Universiti Sains Malaysia
11800 USM, Penang, Malaysia

APPENDIX B

Python Source Codes

Table of Contents

Prepared by P-COM0145/19

If you are interested in an introduction to eddy covariance, head over to PPTI USM' web database.

Data Exploration and Preparation

Import Libraries & set Random Seeds

In [ ]:

```
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import datetime, os

from keras.models import Sequential
from keras.models import Model, load_model
from keras.layers import Input, Dense, Dropout
from keras.losses import mean_squared_error
from keras.callbacks import Callback, ModelCheckpoint, TensorBoard
from keras.optimizers import SGD
from keras import regularizers
import seaborn as sns
from sklearn.model_selection import train_test_split
from pyod.models.auto_encoder import AutoEncoder
from pyod.models.ocsvm import OCSVM
from pyod.utils.data import evaluate_print

from sklearn.preprocessing import Normalizer, MinMaxScaler
from sklearn.pipeline import Pipeline

# visualisations
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid', context='notebook')
%matplotlib notebook
```

In [ ]:

```
data = pd.read_json(r'Desktop/data2019.json')
```

In [ ]:

```
from IPython.display import display
display(data.head(5))
display(data.describe())
display(data.shape)
```

In [ ]:

```
#check for any  nullvalues
print("Any nulls in the dataset ",data.isnull().values.any() )
```

```
print('-------')
print("No. of unique labels ", len(data['qc_co2_flux'].unique()))
print("Label values ",data.qc_co2_flux.unique())
#0 is for normal flux sensor data
#1 is for anomalous flux sensor data
print('-------')
print("Break down of the Normal and Abnormal Flux Sensor Data")
print(pd.value_counts(data['qc_co2_flux'], sort = True) )
```

Data Representation and Transformation

- Handle missing and inconsistent data
- Isolate features from labels
- Handle instances on CO2 flux dataset with redundant columns
- Rename features into the lower case
- Generate statistical based class labels used for AUROC metrics
- Undersample CO2 flux dataset
- Visualize clusters in high-dmensional data

In [ ]:

```
# Filter all rows for which
# wind direction greater than 315 and less than 45
data = data[(data.wind_dir >= 315) | (data.wind_dir <= 45)]
# drop bad quality data from the original data set
data['qc_co2_flux'] = data.qc_co2_flux.replace(1, 0)
data['qc_co2_flux'] = data.qc_co2_flux.replace(2, 1)
data.insert(0, "qc_co2_flux", data.pop("qc_co2_flux"))
data.insert(0, "date", data.pop("date"))
# Replace all rows for which
# NaN with zero
data = data.fillna(0)
# column values with '-9999' value
data = data[(data != -9999).all(axis=1)]
data = data[(data.qc_co2_flux != 1)]
# Data labelling - Statistical method
q25, q75 = np.percentile(data['co2_flux'], 25), np.percentile(data['co2_flux'], 75)
iqr = q75 - q25
cut_off = iqr * 3
lower, upper = q25 - cut_off, q75 + cut_off
data['label'] = [1 if x < lower or x > upper else 0 for x in data['co2_flux']]
# convert the columns to lower case
data.columns = map(str.lower, data.columns)
# drop redundant columns from the original data set
data = data.drop(['p_rain_1_1_1', 'date', 'qc_co2_flux'], axis=1)
display(data.head(5))
display(data.shape)
# splitting by class for normal/anomalous
anomalous = data[data.label == 1]
normal = data[data.label == 0]
display(normal.head(5))
display(normal.shape)
```

Computing with Data

- Rescale features by MinMax
- Rescale features by Standard
- Write both normal and labeled CO2 flux dataset by random splitting, used for training and testing purpose
- Store both normal and labeled CO2 flux dataset into readable format

Train/Validate/Test split

In [ ]:

```
# train // validate - no labels since they're all clean anyway
from sklearn.model_selection import train_test_split
# last column is the target
# 0 = anomaly, 1 = normal
TARGET = "label"

features = data.drop(TARGET, axis=1)
target = data[TARGET]

x_train, x_test, y_train, y_test = train_test_split(
    features, target, test_size=0.5, stratify=target
)

# use case is novelty detection so use only the normal data
# for training
train_index = y_train[y_train == 0].index
train_data = x_train.loc[train_index]

# configure our pipeline
pipeline = Pipeline([('normalizer', Normalizer()),
                     ('scaler', MinMaxScaler())])
# get normalization parameters by fitting to the training data
pipeline.fit(train_data)
x_train_scaled = pipeline.transform(train_data.copy())
x_test_scaled = pipeline.transform(x_test.copy())
# summary
print(f"""Shape of the datasets:
    training (rows, cols) = {x_train_scaled.shape}
    validate (rows, cols) = {x_test_scaled.shape}""")
```

Data Modeling // Wk 9

- Feed the input data to train autoencoders
- Blend autoencoder architectures and hyperparameters (e.g. number of layer)
- Feed regularizer in the encoding layer avoiding overfitting (e.g. l2 regularizer)
- Fit the autoencoder model to reconstruct input
- Learn AutoEncoder training history (e.g training and test curves)

In [ ]:

```
# data dimensions // hyperparameters
nb_epoch = 100
batch_size = 10
input_dim = x_train_scaled.shape[1]
encoding_dim = 8
learning_rate = 1e-2

# deconstruct / encode =====Only AE=====
input = Input(shape=(input_dim, ))
encoded_output1 = Dense(encoding_dim, activation='relu',
activity_regularizer=regularizers.l2(learning_rate))(input)

# reconstruction / decode =====Only AE=====
decoded_output1 = Dense(input_dim, activation='tanh')(encoded_output1)
autoencoder = Model(inputs=input, outputs=decoded_output1)

# deconstruct / encode =====Stacked AE=====
# input = Input(shape=(input_dim, ))
# encoded_output1 = Dense(encoding_dim, activation='relu',
activity_regularizer=regularizers.l2(learning_rate))(input)
# encoded_output2 = Dropout(0.5)(encoded_output1)
# encoded_output3 = Dense(6, activation='relu',
activity_regularizer=regularizers.l2(learning_rate))(encoded_output2)

# reconstruction / decode =====Stacked AE=====
# decoded_output1 = Dropout(0.5)(encoded_output3)
# decoded_output2 = Dense(encoding_dim, activation='relu')(decoded_output1)
# decoded_output3 = Dense(input_dim, activation='tanh')(decoded_output2)
# autoencoder = Model(inputs=input, outputs=decoded_output3)

encoder = Model(inputs=input, outputs=encoded_output1)
```

```
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(inputs=encoded_input, outputs=decoder_layer(encoded_input))

opt = SGD(lr=learning_rate, clipnorm=1.)
autoencoder.compile(optimizer=opt,
                    loss='mean_squared_error',
                    metrics=['mse', 'acc'])

# print an overview of our model
autoencoder.summary()
```

In [ ]:

```
# current date and time
log_dir = os.path.join('logs','fit', datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))

# new folder for a new run
log_subdir = f'{log_dir}_batch{batch_size}_layers{len(autoencoder.layers)}_lr{learning_rate}'

# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min')

checkpointer = ModelCheckpoint(
    filepath="model.h5",
    monitor='val_loss',
    mode='min',
    verbose=0,
    save_best_only=True)

tensorboard = TensorBoard(log_dir=log_subdir,
                          batch_size=batch_size,
                          histogram_freq=0)

# fit the autoencoder model to reconstruct input
history = autoencoder.fit(x_train_scaled, x_train_scaled,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(x_test_scaled, x_test_scaled),
                    validation_split=0.1,
                    verbose=1,
                    callbacks=[tensorboard, checkpointer, early_stop]).history
```

In [ ]:

```
fig, ax = plt.subplots(1, 1)
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.legend(['loss', 'val_loss'])
fig, ax = plt.subplots(1, 1)
plt.plot(history['acc'])
plt.plot(history['val_acc'])
plt.xlabel('Epochs')
plt.ylabel('MSE Accuracy')
plt.legend(['acc', 'val_acc'])
plt.show()
```

Data Modeling // Wk 10

- Learn calculation about mean square error and reconstruction threshold
- Locate number of outliers sample of the CO2 flux dataset

- Build the OCSVM and fit the training data by labeled CO2 flux dataset
- Build the autoencoders and fit the training data by labeled CO2 flux dataset
- Analyze and compare unsupervised AE, Deep AE and OCSVM models with AUROC metrics

In [ ]:

```
# calculating the mean squared error reconstruction loss per row in the numpy array

reconstructions = autoencoder.predict(x_train_scaled)
reconstruction_errors = np.mean(np.power(x_train_scaled - reconstructions, 2), axis=1)

def find_threshold(autoencoder, x_train_scaled):
  reconstructions = autoencoder.predict(x_train_scaled)
  # provides losses of individual instances
  reconstruction_errors = np.mean(np.power(x_train_scaled - reconstructions, 2), axis=1)

    # threshold for anomaly scores
  threshold = np.mean(reconstruction_errors) \
      + np.std(reconstruction_errors)
  return threshold

def find_threshold_method_two(autoencoder, x_train_scaled):
  # another method to find threshold  (Method Percentile)
  reconstructions = autoencoder.predict(x_train_scaled)
  # provides losses of individual instances
  reconstruction_errors = np.mean(np.power(x_train_scaled - reconstructions, 2), axis=1)

  threshold_2 = np.percentile(reconstruction_errors, 95)
  return threshold_2


# another method to find threshold (Method Modified Z-score)

def mad_score(points):
    m = np.median(points)
    ad = np.abs(points - m)
    mad = np.median(ad)

    return 0.6745 * ad / mad

threshold_3 = 3.5
threshold_4 = 4.5
z_scores = mad_score(reconstruction_errors)

def get_predictions(model, x_test_scaled, threshold):
  predictions = autoencoder.predict(x_test_scaled)
  # provides losses of individual instances
  errors = np.mean(np.power(x_test_scaled - predictions, 2), axis=1)
  # 0 = normal, 1 = anomaly
  anomaly_mask = pd.Series(errors) > threshold
  preds = anomaly_mask.map(lambda x: 1.0 if x == True else 0.0)
  return preds
```

In [ ]:

```
threshold = find_threshold(autoencoder, x_train_scaled)
print(f"Threshold method one: {threshold}")

threshold_2 = find_threshold_method_two(autoencoder, x_train_scaled)
print(f"Threshold method two: {threshold_2}")
```

In [ ]:

```
outliers_1 = reconstruction_errors > threshold

print("\nNumber of anomaly samples | Method-1: ", np.sum(outliers_1))
print("\nIndices of anomaly samples: ", np.where(outliers_1))
print(f"Detected {np.sum(outliers_1):,} outliers in a total of {np.size(reconstruction_errors):,} C
O2 flux [{np.sum(outliers_1)/np.size(reconstruction_errors):.2%}].")

outliers_2 = reconstruction_errors > threshold_2

print("\nNumber of anomaly samples | Method-2: ", np.sum(outliers_2))
print("\nIndices of anomaly samples: ", np.where(outliers_2))
```

```
print(f"Detected {np.sum(outliers_2):,} outliers in a total of {np.size(reconstruction_errors):,} C
O2 flux [{np.sum(outliers_2)/np.size(reconstruction_errors):.2%}].")

outliers_3 = z_scores > threshold_3

print("\nNumber of anomaly samples | Method-3: ", np.sum(outliers_3))
print("\nIndices of anomaly samples: ", np.where(outliers_3))
print(f"Detected {np.sum(outliers_3):,} outliers in a total of {np.size(z_scores):,} CO2 flux [{np.
sum(outliers_3)/np.size(z_scores):.2%}].")

outliers_4 = z_scores > threshold_4

print("\nNumber of anomaly samples | Method-3: ", np.sum(outliers_4))
print("\nIndices of anomaly samples: ", np.where(outliers_4))
print(f"Detected {np.sum(outliers_4):,} outliers in a total of {np.size(z_scores):,} CO2 flux [{np.
sum(outliers_4)/np.size(z_scores):.2%}].")
```

In [ ]:

```
# ACCURACY METRICS FOR DIFFERENT THRESHOLDS

from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
preds = get_predictions(autoencoder, x_test_scaled, threshold)
print(f"==== MSE ====\n accuracy: {str(accuracy_score(preds, y_test))[:4]} F1: {str(f1_score(preds,
y_test))[:4]} Precision: {str(precision_score(preds, y_test))[:4]} Recall: {str(recall_score(preds,
y_test))[:4]} ")

from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
preds = get_predictions(autoencoder, x_test_scaled, threshold_2)
print(f"==== PERCENTILE ====\naccuracy: {str(accuracy_score(preds, y_test))[:4]} F1: {str(f1_score(
preds, y_test))[:4]} Precision: {str(precision_score(preds, y_test))[:4]} Recall: {str(recall_score
(preds, y_test))[:4]} ")

from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
preds = get_predictions(autoencoder, x_test_scaled, threshold_3)
print(f"==== MAD_3.5 ====\naccuracy: {str(accuracy_score(preds, y_test))[:4]} F1: {str(f1_score(pre
ds, y_test))[:4]} Precision: {str(precision_score(preds, y_test))[:4]} Recall: {str(recall_score(pr
eds, y_test))[:4]} ")

from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
preds = get_predictions(autoencoder, x_test_scaled, threshold_4)
print(f"==== MAD_4.5 ====\naccuracy: {str(accuracy_score(preds, y_test))[:4]} F1: {str(f1_score(pre
ds, y_test))[:4]} Precision: {str(precision_score(preds, y_test))[:4]} Recall: {str(recall_score(pr
eds, y_test))[:4]} ")
```

In [ ]:

```
# Train single layer AE detector
clf_ae_only = AutoEncoder(hidden_neurons=[11,6,11], hidden_activation='relu', output_activation='ta
nh',
                    optimizer=SGD(0.01), epochs=20, batch_size=batch_size,
                    dropout_rate=0, l2_regularizer=0.001, validation_size=0.5, verbose=0)
clf_ae_only.fit(x_test_scaled)

# Get the prediction labels of the training data
y_train_pred_ae_only = clf_ae_only.labels_

# Outlier scores
y_train_scores_ae_only = clf_ae_only.decision_scores_

# Evaluate on the training data
evaluate_print('Only-AE', y_test, y_train_scores_ae_only)
```

In [ ]:

```
# Predict the anomaly scores
y_test_scores = clf_ae_only.decision_function(x_test_scaled)
y_test_scores = pd.Series(y_test_scores)

# Plot the histogram
import matplotlib.pyplot as plt
plt.hist(y_test_scores, bins=50, alpha=.6, color="green")
plt.title("Histogram for Model [Only-AE, lr=0.001] Anomaly Scores")
plt.show()
```

In [ ]:

```
# Train Stacked AE detector
clf_sae = AutoEncoder(hidden_neurons=[11,8,6,8,11], hidden_activation='relu', output_activation='tanh',
                      optimizer=SGD(0.01), epochs=20, batch_size=batch_size,
                      dropout_rate=0, l2_regularizer=0.001, validation_size=0.5, verbose=0)
clf_sae.fit(x_test_scaled)

# Get the prediction labels of the training data
y_train_pred_sae = clf_sae.labels_

# Outlier scores
y_train_scores_sae = clf_sae.decision_scores_

# Evaluate on the training data
evaluate_print('Stacked-AE', y_test, y_train_scores_sae)
```

In [ ]:

```
# Predict the anomaly scores
y_test_scores = clf_sae.decision_function(x_test_scaled)
y_test_scores = pd.Series(y_test_scores)

# Plot the histogram
import matplotlib.pyplot as plt
plt.hist(y_test_scores, bins=50, alpha=.6, color="green")
plt.title("Histogram for Model [Stacked-AE, lr=0.001] Anomaly Scores")
plt.show()
```

In [ ]:

```
# Train OC-SVM detector
clf_OCSVM = OCSVM(nu=0.01, contamination=0.09, kernel='rbf')
clf_OCSVM.fit(x_test_scaled)

# Get the prediction labels of the training data
y_train_pred_OCSVM = clf_OCSVM.labels_

# Outlier scores
y_train_scores_OCSVM = clf_OCSVM.decision_scores_

# Evaluate on the training data
evaluate_print('OC-SVM', y_test, y_train_scores_OCSVM)
```

In [ ]:

```
# Predict the anomaly scores
y_test_scores = clf_OCSVM.decision_function(x_test_scaled)
y_test_scores = pd.Series(y_test_scores)

# Plot the histogram
import matplotlib.pyplot as plt
plt.hist(y_test_scores, bins=50, alpha=.6, color="green")
plt.title("Histogram for Model OCSVM Anomaly Scores")
plt.show()
```

In [ ]:

```
test_score_df = pd.DataFrame(x_train_scaled)
test_score_df['mseloss'] = reconstruction_errors
test_score_df['threshold'] = threshold
test_score_df['threshold_2'] = threshold_2
test_score_df['anomaly'] = test_score_df['mseloss'] > test_score_df['threshold']
test_score_df['anomaly_2'] = test_score_df['mseloss'] > test_score_df['threshold_2']
anomalies = test_score_df.loc[test_score_df['anomaly'] == True]
anomalies.shape
```

In [ ]:

```
anomalies.head()
```

In [ ]:

```
test_score_df["anomaly"] = anomalies["anomaly"]
test_score_df["anomaly"] = test_score_df["anomaly"].fillna(0)
test_score_df["anomaly"] = test_score_df["anomaly"].astype(int)
test_score_df["anomaly_2"] = test_score_df["anomaly_2"].fillna(0)
test_score_df["anomaly_2"] = test_score_df["anomaly_2"].astype(int)
```

In [ ]:

```
test_score_df['Index'] = test_score_df.index

import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['mseloss'], name='Test loss'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['threshold'], name='Threshold
(MSE)'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['threshold_2'], name='Threshold
(Percentile)'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['anomaly'], mode='markers',
name='Anomaly (MSE)'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['anomaly_2'], mode='markers',
name='Anomaly (Percentile)'))
fig.update_layout(showlegend=True, xaxis_title="Index", yaxis_title="Reconstruction loss")
fig.show()

# To generate the HTML file for the plotly visualization, use:
import plotly.io as pio
pio.write_html(fig, file='Desktop/MSE_plot.html', auto_open=False)
```

In [ ]:

```
test_score_df = pd.DataFrame(x_train_scaled)
test_score_df['madloss'] = z_scores
test_score_df['threshold'] = threshold_3
test_score_df['threshold_2'] = threshold_4
test_score_df['anomaly'] = test_score_df['madloss'] > test_score_df['threshold']
test_score_df['anomaly_2'] = test_score_df['madloss'] > test_score_df['threshold_2']
anomalies = test_score_df.loc[test_score_df['anomaly'] == True]
anomalies.shape
```

In [ ]:

```
test_score_df["anomaly"] = anomalies["anomaly"]
test_score_df["anomaly"] = test_score_df["anomaly"].fillna(0)
test_score_df["anomaly"] = test_score_df["anomaly"].astype(int)
test_score_df["anomaly_2"] = test_score_df["anomaly_2"].fillna(0)
test_score_df["anomaly_2"] = test_score_df["anomaly_2"].astype(int)
```

In [ ]:

```
test_score_df['Index'] = test_score_df.index

import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['madloss'], name='Test loss'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['threshold'], name='Threshold
(MAD=3.5)'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['threshold_2'], name='Threshold
(MAD=4.5)'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['anomaly'], mode='markers',
name='Anomaly (MAD=3.5)'))
fig.add_trace(go.Scatter(x=test_score_df['Index'], y=test_score_df['anomaly_2'], mode='markers',
name='Anomaly (MAD=4.5)'))
fig.update_layout(showlegend=True, xaxis_title="Index", yaxis_title="Reconstruction loss")
fig.show()

# To generate the HTML file for the plotly visualization, use:
import plotly.io as pio
pio.write_html(fig, file='Desktop/MAD_plot.html', auto_open=False)
```