ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

MACHINE LEARNING FOR ANOMALY DETECTION IN CONTINUOUS SIGNALS

A. A. Saoulis*, K.R.L. Baker, R. A. Burridge, S. Lilley, M. Romanovschi
ISIS Neutron and Muon Source, Didcot, UK

Abstract

High availability at accelerators such as the ISIS Neutron and Muon Source is a key operational goal, requiring rapid detection and response to anomalies within the accelerator's subsystems. While monitoring systems are in place for this purpose, they often require human expertise and intervention to operate effectively or are limited to predefined classes of anomaly. Machine learning (ML) has emerged as a valuable tool for automated anomaly detection in time series signal data. An ML pipeline suitable for anomaly detection in continuous signals is described, from labelling data for supervised ML algorithms to model selection and evaluation. These techniques are applied to detecting periods of temperature instability in the liquid methane moderator on ISIS Target Station 1. We demonstrate how this ML pipeline can be used to improve the speed and accuracy of detection of these anomalies.

INTRODUCTION

The ISIS Neutron and Muon source, located at the Rutherford Appleton Laboratory site in Oxfordshire, UK, creates neutron and muon beams used to perform a range of high quality scientific experiments. The facility has developed a great deal over its 35 years of operation [1], both increasing the complexity of the facility and the production of machine data. Currently, anomaly detection and response are generally handled manually by operators, and require large amounts of domain knowledge and expertise. Outlined in this paper is a pipeline to take unlabelled, continuous time series data and train a model that can detect anomalies on live data during operations, improving the response-time and effectiveness of reacting to these anomalies.

ISIS TS1 Methane Moderator

The facility accelerates protons up to 800 MeV, which are used to generate neutrons through a spallation process at ISIS Target Station 1 (TS1) [1], the first of the two target stations at ISIS. These neutrons are moderated at TS1 through several different moderators, one of which is a liquid methane moderator, in order to perform neutron scattering experiments. For high quality experiment data, very stable temperature in the methane moderator is required.

Whilst methane has many properties that give it excellent performance as a moderator [2], it is well known that it is susceptible to radiation damage [3]. The irradiation of the methane produces long chain polymers and releases hydrogen [3,4], the former causing the moderator to fail and require replacement roughly every six months. The build-up of free hydrogen within the moderator system causes loss of flow and leads to unpredictable pressure variability and spiking. This causes the temperature in the methane moderator to become unstable, which increases the variance of neutron energy leaving the moderator; it is therefore of key operational importance that these losses of flow are dealt with quickly. One method through which the operations team have dealt with this issue is through daily, scheduled "recoveries" of the moderator that consist of flushing through one third of the liquid methane in the system into a dump tank. This has improved its stability, but occasional periods of flow, and thus pressure and temperature variability, still occur.

The operations team currently have systems in place for detecting these instabilities, such as monitoring differential pressure in the system for any reduction in pressure. If an anomaly is detected, the operations team inspect the recent behaviour of the moderator and decide whether to run an unscheduled recovery in order to return the system to normal operation. The current systems often fail to flag up ongoing anomalies, leading to long periods of temperature variability that can cause the data recorded in downstream instruments to be unusable.

This paper will investigate the automated detection of these periods of temperature variability, with the goal of aiding the operations team to track down and fix issues faster. The paper will make use of supervised Machine Learning (ML) algorithms, which require labelled data (i.e. each data instance has an associated class label, such as whether it is "normal operation" or an "anomaly") to train a model. Finally, a brief description of the process of deployment of such trained models to live operation will be given.

DATA PIPELINE

Here, a pipeline is given that takes raw time series data without labels and produces a dataset that is suitable for training ML models. In the case of the ISIS TS1 methane moderator, there was neither a logbook nor a convenient signal that could be used to automatically label temperature anomalies in the historic data. One key contribution of this paper is to define a general procedure for automatically labelling periods of anomalous behaviour in historic time series data so that models can be trained to detect these periods during live operation. Note that while this paper focuses on a single signal (i.e. a univariate time series), these methods are generalisable to multivariate time series.

* alex.saoulis@stfc.ac.uk

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

Raw Data

The ISIS accelerator control systems have recently undergone several upgrades [1, 5] that have allowed for the long term storage of time series data from a huge number of sources across the facility. This has provided a wealth of raw time series data of signals across the accelerator side of the facility stretching back to the end of 2018. As is outlined in [5], this data is stored in a database called InfluxDB [6], which specialises in time series storage. This allows for fast and simple access to this time series data, greatly easing the use of this data for data analysis and ML purposes.

The raw time series data is collected at intervals of approximately 2 – 3 s, provided the value of a signal has changed. This leads to irregular and unaligned raw data that must be preprocessed before use in ML applications. The temperature data used in this paper is taken from signal *TC68M*, a thermocouple located close to the liquid methane.

Data Preprocessing

The first step of the data pipeline is to preprocess the data into a form that is practically useful and convenient for use in ML [7]. The preprocessing work was done in Python, primarily using the Python library pandas [8]. The data must be filtered such that only data covering operational periods of the accelerator is included in the training and test data. This is to ensure the training and test data matches the target domain data in production; in other words, the ML model should be trained and evaluated on the same type of data that it will see when it is detecting anomalies during live operation of the accelerator. This means that out-of-cycle periods, periods of bad/missing data (for example caused by a hardware malfunction), and periods outside normal operational bounds (such as temperature bounds beyond which it can be inferred that the moderator was not in use) must all be filtered from the data.

ML models require an input vector, also known as a feature vector, which must be of fixed length. For an irregularly sampled time series, simply using the raw data points as feature vectors would lead to temporally mismatched inputs to the model; a feature vector of length 100 could span from e.g. 3 minutes - 10 minutes, depending on how frequently the signal was changing during that period. Instead, feature vectors should be regularised such that each element of the feature vector corresponds to a fixed length of time. The raw temperature data is downsampled into bins of fixed time by taking a mean over that period. If there are no elements in a downsampled period, it is inferred that the temperature did not change from its previous value and the last known value is forward filled into the new bin.

The data must be scaled since the stability of many ML models require input data to be within reasonable bounds [7]. In the case of the TS1 methane moderator temperature, the mean of each temperature window was subtracted from the raw data. This ensured that the scaled values in the feature vectors always lay in the range of $[-10, 15]$, as well as helping to account for non-stationarity in the time series due to long-term drift of the moderator temperature. This choice was made for simplicity, and the conventional choice of z-scaling the data or normalising between $[0, 1]$ would also do.

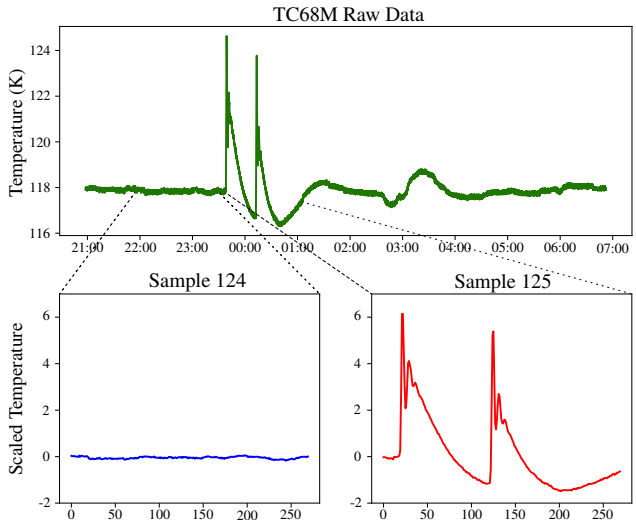

Figure 1: A short excerpt from the raw temperature time series, as well as two sample feature vectors generated from the data. Sample 124 is an example of normal operation, and sample 125 is an example of an anomaly.

Feature Vector Selection

In order to settle on a suitable feature vector, there are a few factors that must be taken into account. The window length must be long enough such that it can completely cover an anomaly event; this requires data exploration to look at the timescale over which anomalies occur in a system. A second important consideration is the bin lengths: these should be short enough to preserve any fine-grained structure in the time series that distinguishes normal operation from anomalies. Finally, in order to train models that can be evaluated constantly in real time, the feature vectors should be randomly sampled from the historic time series data such that a model can be applied at any moment in time

For the TS1 methane moderator, anomalies in the temperature tended to last between 1 – 2 hours, though certain indicators of anomalies such as rapid oscillatory periods occurred over much shorter timespans. As a result, 90 minute long feature vectors with bin lengths of 20 s were chosen, creating feature vectors of dimension (270, 1). The feature vectors were then generated by going through the downsampled time series data chronologically and extracting windows of length 270, skipping a small, random number of bins in between feature vectors to ensure that they had been randomly sampled. Roughly 4300 wholly distinct (i.e. no overlap) feature vectors were generated from the 2 years of historic temperature data, obeying the data preparation rules laid out above. Two such feature vectors, one from each class, are given in Fig. 1.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

LABELLING DATA

As mentioned earlier, in order to implement supervised ML algorithms, a class label corresponding to each feature vector is required. These are initially used to train a model to correctly identify the class of an input feature vector, and afterwards to evaluate the model on a test data set. Without any signals or logbooks that could be used to identify instances of anomalies in the historic data, labels must be generated directly from the feature vectors. This would typically be extremely time consuming and require a great deal of tedious manual work and expert knowledge. Here a generic method for partially automated labelling of time series feature vectors is presented.

Clustering through t-SNE

t-distributed Stochastic Neighbourhood Embedding (t-SNE) [9] is a dimensionality reduction technique, differing from the well known Principal Component Analysis (PCA) [10] in that it is a non-parametric, non-linear technique for embedding high dimensional feature vectors into low dimensional space according to their similarities. The main advantage of this technique over other dimensionality reduction methods is that it tends to produce well defined clusters of similar data points, which can be used to generate labels from feature vectors. Since t-SNE is tailored for visualising high dimensional data, it proves very well-suited to the data labelling process that will be introduced in this section.

There are also difficulties associated with using t-SNE; suitable hyperparameters for t-SNE must be chosen in order to generate meaningful clusters of data, which can become a tedious tuning process. Additionally, since the algorithm is stochastic, it may take several runs with the same hyperparameters to generate well defined clusters. One other disadvantage of t-SNE is that it is a non-parametric dimensionality reduction technique, meaning that, in contrast to e.g. PCA, the algorithm doesn't produce a mathematical formula that can map any high dimensional feature vector to its low dimensional embedding. This means that new data points cannot be embedded into the low dimensional space after the algorithm has been run.

In this paper, t-SNE will be used to generate clusters of similar looking feature vectors, which in turn will be used to generate labels for a large percentage of the data with minimal manual labelling. This relies on one key assumption, which is that a similarity score between feature vectors can be defined which is able to meaningfully distinguish between anomalous feature vectors and normal operation. The original t-SNE algorithm used the Euclidean distance between two feature vectors to calculate similarity, but for time series that can be very misaligned (due to the fact that for continuous time signals, there are no preset start and end points), a Euclidean distance will fail as a similarity metric. Instead, a distance metric better suited to misaligned time series is required.

Dynamic Time Warping

Dynamic Time Warping (DTW) [11] is an algorithm well suited to calculating the similarity between two misaligned time series. The algorithm is designed to work for time series that start and end at the same point, but whose main features vary in speed and so may be offset from each other along the time axis. There are a number of adaptations of DTW that can be used to tailor the algorithm for a specific use-case and set of constraints [12]. Even so, DTW works well even when there is no guarantee that two time series have the same start and end point; empirically, in preparation for this paper, standard DTW performed just as well as adaptations from [12] that accounted for these effects. As such, the standard DTW algorithm was used to calculate the similarity between feature vectors.

Note that in order to compare multivariate time series, the DTW algorithm can be applied to each individual signal (or a select few) across separate multivariate feature vectors, and a weighted sum of the individual DTW similarities can be used as the total similarity metric between two feature vectors.

Labelling the Temperature Windows

In order to apply t-SNE, the parallelisable function `metrics.pairwise_distances` from the Python library scikit-learn [13] was used to generate a similarity matrix explicitly between every pair of feature vectors, using DTW as the "metric". This is a computationally expensive step that has computational complexity $\mathcal{O}(n^2m^2)$, where n are the number of feature vectors in the data set, and m is the length of each feature vector. This step took around 30 minutes to run for the temperature samples on six cores on a desktop class machine. Note that this method may quickly become infeasible for very large datasets (e.g. when the number of samples $> 10^5 - 10^6$), or large feature vectors, without access to high performance computing clusters. Once the similarity matrix is computed, t-SNE can be run repeatedly for hyperparameter tuning without incurring the cost of calculating the similarity matrix each time. t-SNE embeddings of the temperature samples were created through a hyperparameter grid search until a suitable embedding was generated that had two distinct clusters of points. The final hyperparameters used for the scikit-learn function `manifold.TSNE` are given in Table 1.

Table 1: t-SNE Hyperparameters

Parameter name	Value
# Embedding Dims	2
Perplexity	30
Learning rate	25
# Iterations	5000
Method	"barnes_hut"
Angle	0.5

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

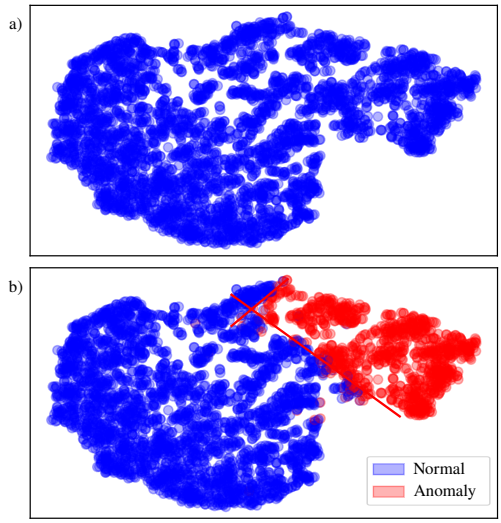

Figure 2: t-SNE plots showing the labelling process once two clusters have been generated. a) The raw 2D embedding generated by the t-SNE algorithm. b) The final labels assigned to the data after all labelling steps. The interactive plotting tool was used to get a first-pass estimate of the class boundaries, shown by the red lines.

In this case, as can be seen in Fig. 2, two large-scale, distinct clusters are easy to discern. An interactive plotting tool that allowed a user to select a point in the 2D embedding and view the corresponding time series was developed to inspect the data. Since the task was to classify normal operation versus anomaly, and not to distinguish between different types of anomaly, it was simple to manually define cutoff planes in the embedded space to make a first order approximation of the feature vector labels. If there are more than two classes, or the clusters are not simple to define manually, it could be preferable to use an unsupervised clustering algorithm such as k-means to calculate class boundaries. Then, the same plotting tool was used to manually change the label of any data points that were mislabelled on the first pass. This step can be time-consuming, especially in cases with large data sets and difficult-to-define class boundaries, and may need contributions from a domain expert to correctly classify edge-case time series.

One final step to ensure that this partially automated labelling process has generated accurate labels is to use a simple model or heuristic to classify the feature vectors, and compare the model classifications with the labels to check for bad labels. In this case, a rolling standard deviation of the time series was calculated, and feature vectors that exceeded a threshold were labelled as anomalies. This discovered a number of mislabelled data points, and was thus used to decrease the noisiness of the labels. Nonetheless, it should be borne in mind that no such labelling process will be perfect, and there will always be a degree of noisiness in the data set labels.

NEURAL NETWORK MODELS

Once a labelled dataset has been generated, a model can be trained using supervised ML. Neural networks have shown great promise for time series classification tasks [14], with recent papers and architectures delivering state-of-the-art performance on benchmark time series datasets [15]. This paper will introduce, train and evaluate three different neural network (NN) based models, as well as explore the differences between them. All of the models were created, trained and evaluated in Python using Keras [16] as the frontend for the popular ML library Tensorflow [17]. Neural networks have an added of bonus of generalising naturally to multivariate time series since they are data driven models.

Feedforward Neural Network

The simplest type of neural network is a feedforward neural network, also known as a multi-layer perceptron. The input to the neural network is the time series feature vector. Each subsequent "layer" of the neural network can be thought of as a matrix multiplication by matrix of size $m \times n$, where m is the size of the input vector and n is the number of "nodes" in a layer, followed by an elementwise non-linear transformation acting on the result of the matrix multiplication, known as an "activation function". The input is "fed forward" through the network of several layers, each of which can have different activation functions and numbers of nodes, until a final "output" layer that transforms the penultimate vector into a vector of length two, with each element corresponding to the probability that the feature vector belongs to either the "normal operation" class or "anomaly" class.

There are two main problems with using feedforward NNs for time series classification of the sort that has been described in this paper. Firstly, feedforward NNs learn the relationships between fixed elements in the input vector; since the position of features in the time series can occur at any point in the input, and their positions with respect to one another will be highly variable, a feedforward NN is not well suited to learning specific relationship between features. That said, for simple problems, it should still have some capacity to learn general features that it can use to make correct classifications. Secondly, feedforward NNs are generally thought of as black-box models, which means that it is very difficult to understand why a NN has made a particular classification. This can be problematic for models used in a control system to make decisions, since an operator may want to have some understanding of why a model detects an anomaly before making any changes.

Attention-based LSTM

The Long Short-Term Memory (LSTM) network has emerged as a valuable tool for time series classification and prediction problems in ML [15, 18]. It is an improvement of the Recurrent Neural Network (RNN) architecture, which processes an input vector in temporal order, preserving the causal structure of a time series that is generally lost in

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

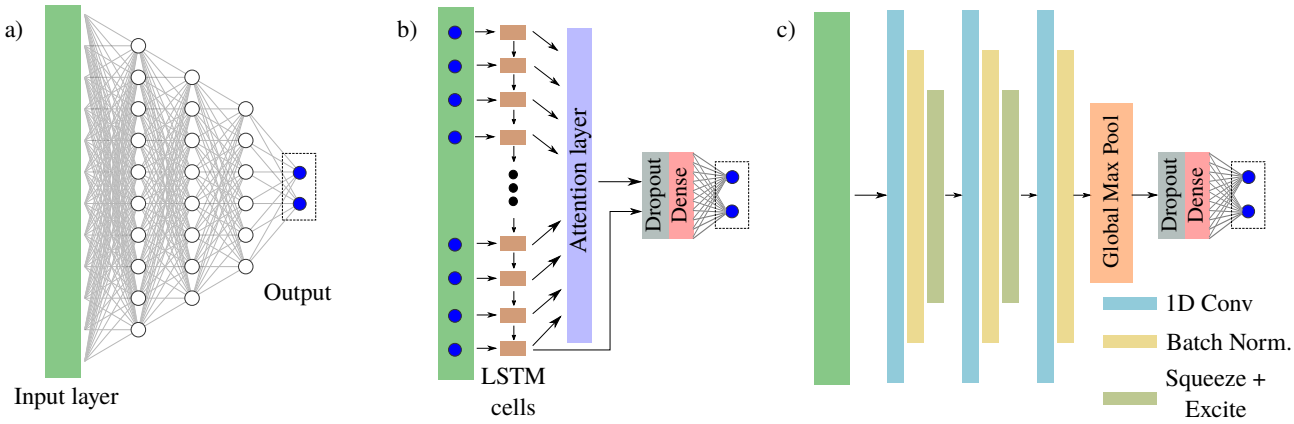

Figure 3: The three types of neural network used in this paper. ReLU activation function everywhere unless specified. Input layer always has dimension 270, output always has dimension 2 with softmax activation. a) The simple feedforward network with 4 hidden layers of size 64, 32, 16, 8 respectively. Dropout of 0.3 between each layer. b) The ALSTM architecture. Each LSTM cell had 16 nodes. The attention layer output and final LSTM output (both tanh activation functions) are combined before a dropout layer and a densely connected output layer. c) The CNN architecture with 3 1D convolutional layers: number of filters (32, 64, and 32) and filter size (8, 5, and 3) respectively. See [15] for a detailed explanation of architectures b) and c).

simpler NNs like feedforward networks. The LSTM introduces features that allow for information to be preserved throughout the network, giving it the ability to learn long-term temporal dependencies in the feature vectors.

This architecture is then expanded to include an attention mechanism, creating the Attention-based LSTM (ALSTM) [15]. In short, attention in neural networks allow the network to learn how to identify the most important areas in an input feature vector to make a classification, further expanding the ability of the LSTM to learn long term dependencies. Attention also has the bonus of providing some explainability to the neural network; after evaluating the neural network on an input feature vector, the attention activations can be examined to see which portions of a time series were most important in making the classification.

1D Convolutional Neural Network

The convolutional neural network (CNN) rose to prominence in the field of deep learning as a standout performer for image classification [19]. While it is most commonly used for tasks like computer vision, 1D CNNs have also been shown to perform well in time series classification tasks [14, 15]. CNNs work by learning filters that are run across an input feature vector to identify small scale edges and features, and then combining activations of these filters to build higher level features. The parallel between how CNNs work and how a human goes about time series classification are striking; much like a CNN, a human searches for shapes that match their understanding of what an anomaly might look like, and bases their classification on what types of shape appear in the time series.

Just as with the ALSTM, there has been a large amount of research that focuses on making CNNs explainable. These techniques include saliency maps [20], which use network activations to generate maps of regions of importance over the input, and occlusion methods [21] that mask regions of the input to determine which parts of the input were most used to make the classification decision.

TRAINING AND RESULTS

Each of these types of models were explored through network architecture tweaks and hyperparameter tuning. The final architectures and hyperparameters are given in Fig. 3. Since there was a data imbalance with a ratio of roughly $3:1$ between normal operation and anomaly in the feature vectors, anomalies were given a 3× class weight in the training process to ensure that the models would not learn a bias toward the more frequently occurring class. All of the models were trained using the default Keras Adam optimiser, and all models finished with a softmax layer to output a vector of probabilities of the input belonging to each class.

Of the 4300 time series samples, 65% of samples were used for training, 15% of samples were used for validation and the remaining 20% were used for a holdout test set on which each model was evaluated just once after the hyperparameter tuning was complete. The test set accuracies are the best indicator of out-of-sample performance, and are given in Table 2.

Table 2: Model Performance on the Test Set

Network Type	Accuracy (%)
Feedforward NN	97.5
ALSTM	96.0
CNN	98.3

The misclassifications on the test set were plotted and examined in each case. Since the ground truth labels were noisy due to the inherent problems in the data labelling

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

process, many of the "misclassifications" were actually the model outperforming the labelling process. In the case of the CNN, the majority of misclassifications were just bad labels, which suggested that the true CNN accuracy could be even greater than the accuracy given in Table 2.

APPLICATION

Once these models were trained and evaluated, the feed-forward NN and ALSTM (the CNN was omitted due to time limitations), as well as a simple heuristic that used the rolling standard deviation of the temperature time series to calculate the probability of anomaly, were deployed into production. To do this, a software stack using Docker [22] containers that sat on top of the existing ISIS accelerator controls MQTT messaging [23] system was developed. In the existing ISIS accelerator control system, live changes are sent to an MQTT broker, which are then consumed by several downstream applications. These messages were processed in the same way the training data was generated in the data processing pipeline to generate rolling feature vectors every 20 seconds. The feature vectors were then fed into the neural networks each time a new bin was generated (again, every 20 seconds). The outputs of the networks were then sent over the same MQTT network, after which they were collected by the metrics aggregator Telegraf [24]. The outputs, alongside the raw time series data, were stored in an InfluxDB [6] database, and displayed in real time in a Grafana [25] dashboard. The software stack is shown in Fig. 4.

Once the software stack and models had been tested for robustness and stability for a few weeks on the live system, a feature was added to generate control system alarms that would appear in the Main Control Room once an anomaly was detected. In order minimise the rate of false positives, an alarm was only generated if all models agreed with high probability that there was an anomaly for over 20 minutes. This alarm generation system was nominally successful, though since this work was done so near to the start of the ISIS long shutdown there was not enough time to generate data to report here.

FUTURE WORK

A natural extension of the work in this paper would be to extend the anomaly detection problem to one of anomaly prediction. This has clear practical importance to operators, since it would provide warning that could be used to minimise the impact of an anomaly, or potentially even prevent it entirely.

Some preliminary work was carried out using results given in this paper: the labels generated during the t-SNE labelling process were used to get approximate timestamps of the beginnings of the anomalies. Then, a number of extra signals recommended by domain experts (such as pressures, beam current, etc.) were collected and added to a feature vector directly preceding the anomaly, giving a multivariate time series acting as the input to the model. A model combining a CNN and ALSTM (see [15]) was trained to identify whether a time series was a precursor to an anomaly (or not), but it performed very poorly even after a long period of hyperparameter tuning. Some steps that will be explored in future work include feature engineering (likely after including an even larger set of time series signals), different ML models such as Gradient Boosted Trees, and addressing the large class imbalance through data augmentation.

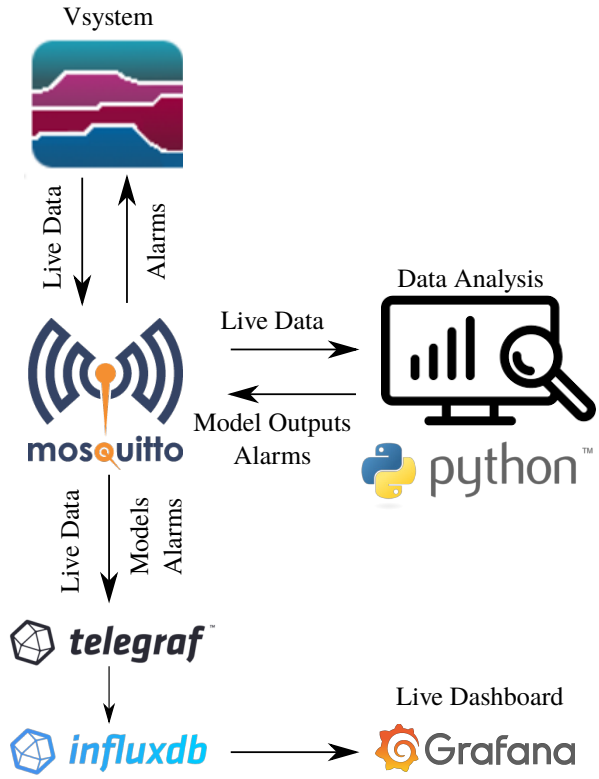

Figure 4: The software stack used to generate alarms and create live dashboards. The neural networks are evaluated in the data analysis layer, and their outputs are fed back into the MQTT messaging system.

CONCLUSION

A raw time series without labels was converted into a labelled data set using the clustering technique t-SNE. This dataset was then used to train several NN models which were first evaluated, and then deployed into live operations to generate alarms for the operations team any time an anomaly was detected. This pipeline provides a process through which anomaly detection can be applied to generic time series data streams, and proved a promising avenue for improving the response time to anomalies and thus availability of beam at large scale facilities.

REFERENCES

[1] Thomason, J.W.G., "The ISIS spallation neutron and muon source—The first thirty-three years" *Nuclear Instruments and Methods in Physics Research Section A: Accelerators,*

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBL01

Spectrometers, Detectors and Associated Equipment, vol. 917, pp. 61–67, 2019.

[2] Kirichek, O., Lawson, C.R., Draper, G.L., Jenkins, D.M., Haynes, D.J. and Lilley, S., "Solid methane moderators: Thermodynamics and chemistry", *Journal of Neutron Research*, (Preprint), pp. 1–6, 2020.

[3] Evans, D., "Irradiation effects in liquid methane used as a neutron moderator", *Cryogenics*, vol. 35, no. 11, pp. 763–766, 1995.

[4] Dean, R., Harrison, P., Burridge, R., Jenkins, D. and Probert, M., "Process filtration of liquid methane radiation products using centrifugal separation". in *Journal of Physics: Conference Series*, vol. 1021, no. 1, p. 012074, IOP Publishing, May, 2018.

[5] Finch, I.D. and Howells, G., "Controls Data Archiving at the ISIS Neutron and Muon Source for in-depth analysis and ML applications", presented at the 18th Int Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21), Shanghai, China, 2021, paper WEPV049, this conference.

[6] InfluxDB: time series database, `https://www.influxdata.com/`.

[7] Kotsiantis, S.B., Kanellopoulos, D. and Pintelas, P.E., "Data preprocessing for supervised leaning", *International journal of computer science*, vol. 1, no. 2, pp. 111–117, 2006.

[8] pandas: data analysis tool in Python, `https://pandas.pydata.org/`.

[9] Van der Maaten, L. and Hinton, G., "Visualizing data using t-SNE", *Journal of machine learning research*, vol. 9, no. 11, 2008.

[10] Jolliffe, I.T. and Cadima, J., "Principal component analysis: a review and recent developments", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[11] Berndt, D.J. and Clifford, J., July. "Using dynamic time warping to find patterns in time series", in *KDD workshop*, vol. 10, no. 16, pp. 359–370, 1994.

[12] Giorgino, T., "Computing and visualizing dynamic time warping alignments in R: the dtw package", *Journal of statistical Software*, vol. 31, no. 7, pp. 1–24, 2009.

[13] Scikit-learn: machine learning in Python, `https://scikit-learn.org/stable/`.

[14] Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L. and Muller, P.A., "Deep learning for time series classification: a review", *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[15] Karim, F., Majumdar, S., Darabi, H. and Harford, S., "Multivariate LSTM-FCNs for time series classification", *Neural Networks*, vol. 116, pp. 237–245, 2019.

[16] Keras: deep learning API for Tensorflow, `https://keras.io/`.

[17] Tensorflow: an end-to-end open source ML platform, `https://www.tensorflow.org/`.

[18] Siami-Namini, S., Tavakoli, N. and Namin, A.S., "A comparison of ARIMA and LSTM in forecasting time series", inn *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, pp. 1394–1401, Dec. 2018.

[19] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[20] Simonyan, K., Vedaldi, A. and Zisserman, A., "Deep inside convolutional networks: Visualising image classification models and saliency maps", *arXiv preprint*, 2013. `arXiv:1312.6034`

[21] Fong, R.C. and Vedaldi, A., "Interpretable explanations of black boxes by meaningful perturbation", in *Proceedings of the IEEE international conference on computer vision*, pp. 3429–3437, 2017.

[22] Docker, `https://www.docker.com/`.

[23] MQTT: message broker, `https://mosquitto.org/`.

[24] Telegraf: metrics aggregator, `https://www.influxdata.com/time-series-platform/telegraf/`.

[25] Grafana: dashboarding tool, `https://grafana.com/`.