# Pre-processing for Anomaly Detection on Linear Accelerator

## September 2020

AUTHOR(S):
***Martin Molan***
Jozef Stefan International postgraduate school

SUPERVISOR(S):
***Dr. Yann Donon***
***Dr. Alberto Di Meglio***

CERN openlab

# PROJECT SPECIFICATION

The goal of the CERN openlab summer project was to:

- parallelize existing pre-processing code for anomaly detection on linear accelerator
- deploy the parallelized pre-processing solution to CERN's Hadoop cluster

# ABSTRACT

This report describes the implementation of the data pre-processing for a novel anomaly detection technique. Proposed anomaly detection technique is based on using stochastic matrices as input for convolutional neural networks. Pre-processing step transforms raw data into 3d tensors combining stochastic matrices for a given event. Proposed solution for pre-processing is split into data reading part, which is parallelized with Dask and data processing part, which is parallelized with Spark. Data reading part runs on CERN's Swan notebook and data processing part runs on a Hadoop server (specifically general purpose server Analytics).

# TABLE OF CONTENTS

# 1. Introduction

Goal of the project is to construct an anomaly detection approach for linear accelerators (LINACs). One of CERN's use of those LINACs is as boosters for the Large Hadron Collider (LHC) complex. Beside high energy physics, linear accelerator are also used in other disciplines such as medicine, for radiation therapy.

The operation of complex structures such as linear accelerators (LINACs) is monitored by services that generate huge amounts of data, which's meaningfulness is often hard to discern. A lot of raw data is available but its sheer size poses a daunting challenge for system administrators who aim to analyse the data for predictive and maintenance tasks. The use of machine learning (ML) is introduced to assist in the implementation and realization of such demanding tasks. Automated models can greatly benefit system operators and facility owners (Beneventi, et al., 2017).

The aim of introducing ML approaches for systems reliability is to extend the capability of reporting services and potentially automate some system maintenance tasks or give them additional insights into the operation of the system. Specifically, ML applications for systems reliability focus on two fields:

(i) Anomaly detection
*Identification of system errors/anomalies*

(ii) Predictive maintenance
*Prediction of system failures and possible prevention of downtime*

# 2. Related work

The main focus of this work is to construct pre-processing for anomaly detection models. These insights can however also be applied (in the future) to the task of predictive maintenance.

There are two basic approaches to solve an anomaly detection problem: supervised ML methods and semi-supervised ML methods (Beneventi, et al., 2017), (Borghesi, et al., 2019) (Borghesi, et al., 2019), (Borghesi, et al., 2019), (Kotsiantis, et al., 2007). The nature of the problem of anomaly detection on linear accelerator suggests a use of semi-supervised ML methods, as there are not labelled input data. A standard method for anomaly detection that achieves state of the art results in some applications is the use of autoencoders (Kwon, et al., 2017), (Lan, et al., 2010). Other methods focus on the combinations of informative features and machine learning approaches that work well with limited datasets (Molan, et al.), (Molan, 2019).

## a. Autoencoders

State-of-the-art anomaly detection models take advantage of large quantities of granular data by training deep learning autoencoders on that data. Autoencoders are a type of a neural network that is trained on the task of replication of the original sequence. Autoencoders usually contain a hidden layer that is lower-dimensional than the input and output layers. This hidden layer and possible regularization (dropout) layers prevent the autoencoder from replicating the input data perfectly. The difference (loss) between input and output sequence in the training faze is used to drive the training of the network and, in the prediction phase, to identify deviations from the baseline. The methods proposed in based on autoencoders, achieve very high anomaly recognition accuracy on the case of anomaly recognition on high performance computing systems (Borghesi, et al., 2019) (Borghesi, et al., 2019) (Borghesi, et al., 2019).

## b. Hidden Markov models

Hidden Markov models (HMM) are based on Markov chains – finite state machines governed by stochastic processes. Stochastic process is estimated with a stochastic matrix where the element at position *ij* describes the probability of state *i* transitioning into state *j.* The general use of modelling the system dynamics with a Markov model would be to:

- Learn the normal (baseline) system dynamic (on all data) and represent them with a Markov model
- Estimate anomalies by comparing the sequence probability with probabilities for baseline sequences

Raw sensor data is not inputted into hidden Markov models; some pre-processing steps are necessary to produce more reliable results (Arpaia, et al., 2019). The most important of these pre-processing steps is discretization (conversion of continuous variables to discrete ones). Discretization can be done with clustering or binning.

Another approach to anomaly detection using hidden Markov models is to treat stochastic matrices as objects that will be inputted into other machine learning models (specifically CNN networks) (Y. Donon, 2019). The main idea of this approach is to have each sequence generate its own stochastic matrix; these matrices are then inputted into a convolutional neural network. Pre-processing step for this approach is implemented in this report.

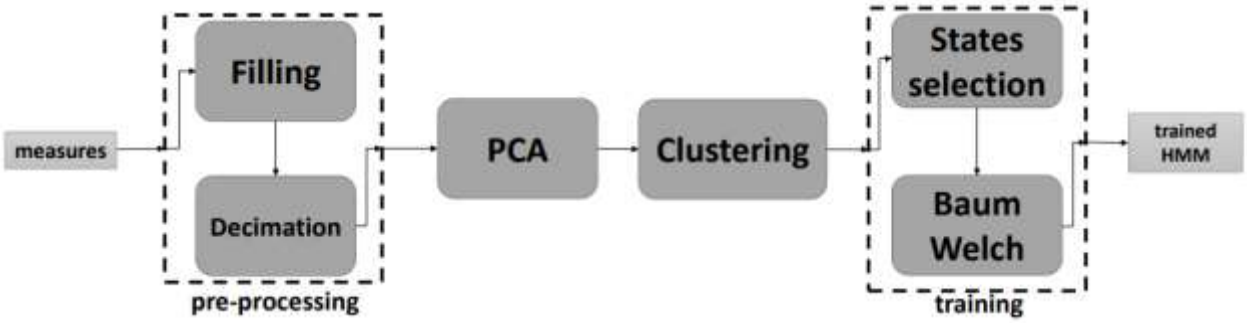


*Figure 1 Pre-processing for a HMM adopted from (Arpaia, et al., 2019)*

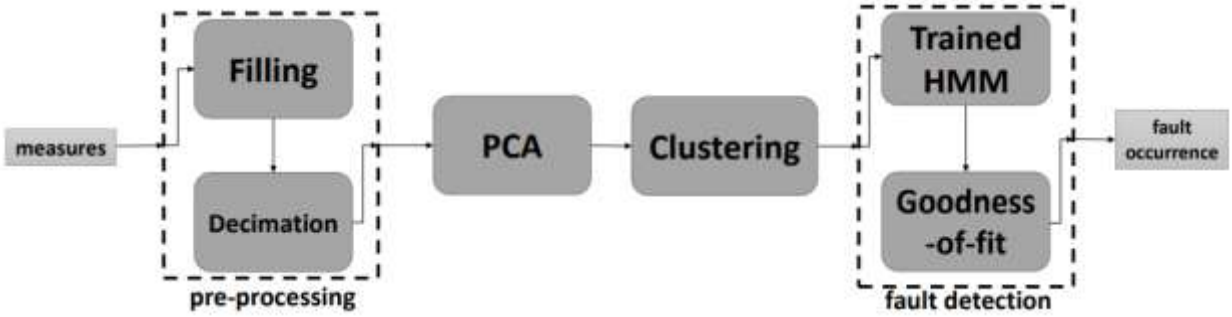


*Figure 2 HMM used for fault detection adopted from (Arpaia, et al., 2019)*

## 3. Software implementation

The main goal of software implementation of the pre-processing step is to create a parallelized solution that would work on CERN Hadoop clusters. Specifically the included solution is written in combination of PySpark and Dask parallelization frameworks and it runs on Analytics (general purpose Hadoop) cluster. Software implementation contains two parallelized parts that run on two different cloud servers:

Data reading, filtering and job submission. Job submission and management part runs in CERN's Swan notebook. Inside Swan notebook, reading from the database is done in parallel with Dask. Dask parallelization framework was used as it works nicely with Python environments where there is no need for creation of a cluster. Each Dask worker reads allotted data from the database, performs rudimentary filtering (not computationally complex) and sends the raw data to PySpark (it submits a job to PySpark cluster).

Discretization, filtering, stochastic matrix calculation. Majority of computationally complex data processing is handled by PySpark. PySpark job calculates stochastic matrix for each sequence (map job) and then combines matrices for several functions into a tensor (reduce function).

The general goal of the pre-processing step is to extract sequences of events form a database and represent them as a stochastic matrix (2d tensor). These 2d tensors are grouped, according to their ids into 3d tensors that can be used as inputs for deep learning.

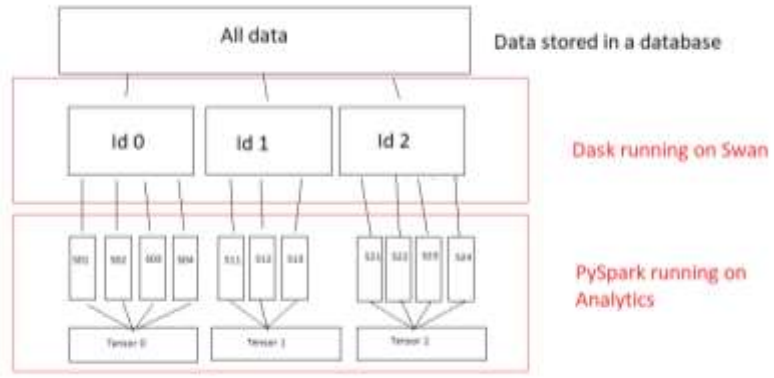


*Figure 3 Parallelization scheme*

### a. Data reading

Data, used in the experiment is hosted on the SmartLINAC web platform (hosted at CERN). In order to speed up the overall computation and to simplify the implementation (no need to include permanent storage for raw data) the decision has been made not to store raw data. Raw data is stored only in system memory and sent to the Spark cluster.

Data is stored and organized by different Id. Each specific Id describes a group of events (a group of sequences) from a specific source (e.g. hardware sensor on a linear accelerator). All events under specific Id either belong to the baseline operation or they represent an anomaly. Data under a specific Id is split in sequences of fixed length (usually several thousand data points represent a single sequence). Data from each Id is thus split into several sequences; each sequence is then sent to Spark cluster to be pre-processed in parallel. Result of each sequence transformation is a 2d tensor; 2d tensors from all sequences from a single Id are then transformed into a 3d tensor.

Each specific Id forms a separate Dask job (Dask worker). Each worker reads data form a database and sends that data to a Spark cluster in a form of a Spark job.

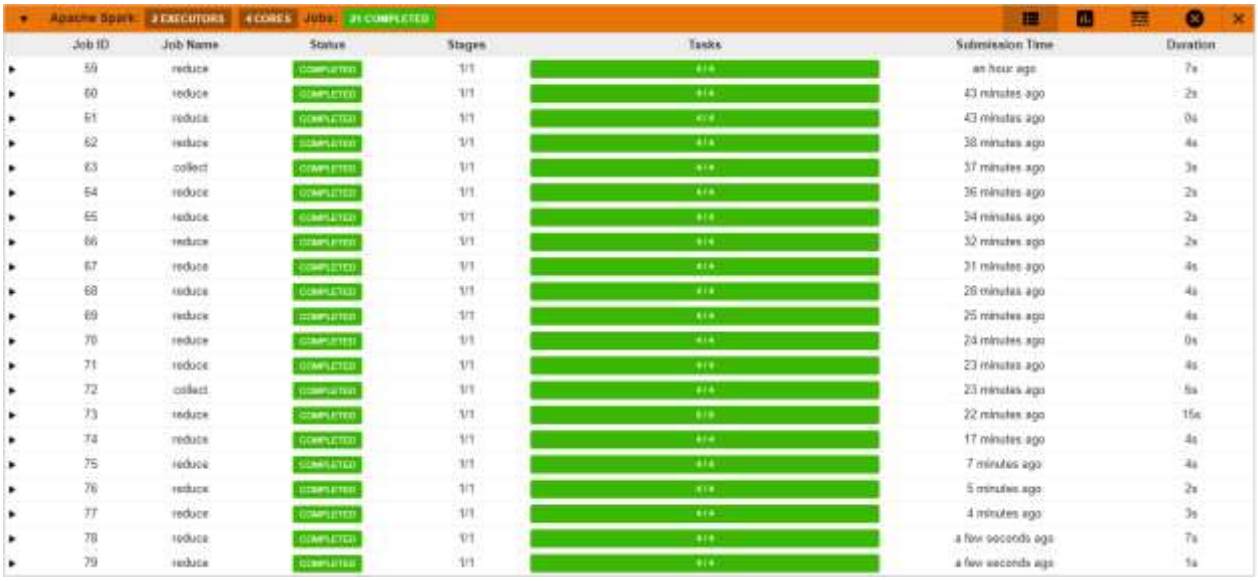


*Figure 4 Summary of submitted Spark jobs*

## b. Data processing

Each spark job consists only of the data from a single Id; the result of the job will be a 3d tensor representing that Id. Input for a spark job is raw data that is transformed into a *delayed data frame.* Delayed data frame is a list of objects that will be processed in parallel. Processing consists of two steps:

- Map. Transform each element in a delayed data frame. Map job performs most of the computationally expensive tasks.
- Reduce. Reduce functions collects the results of all map functions and returns a single object – in our case a 3d tensor.

```
def submit_spark_job(context, local_bind_address = 3306, idSource = 27, increment = 5000, test = False):
    try:
        results_l = get_raw_data_list(local_bind_address, idSource, increment, test)
    except Exception as e:
        print(e)
        return []
    rdd = context.parallelize(results_l)
    rdd_calc = rdd.map(statisticalAnalysis_raw)
    fr = rdd_calc.reduce(combine_tensor)
    return fr
```

*Figure 5 Main Spark function*

### i. Map

Map step *maps* the main pre-processing function to each partition of the data. The pre-processing function consists of:

- Applying Gaussian filter. Here Scikit-learn implementation is used (Géron, 2017).

- Discretizing the data with k-binning. The data is clustered into k clusters; the labels of the clusters are then used as a discrete variables that replace the original continuous values. Scikit-learn implementation of k-binning is used (Géron, 2017). Other binning strategies can also be used such as binning with fixed thresholds.
- Estimation of stochastic matrix. Discrete valued sequence is treated as a Markov chain. From there, stochastic matrix can be estimated.

```
def StochasticMatrix(arrayInitial, classes):

    if len(arrayInitial) == 0:
        return np.zeros([classes,classes])

    arrayLength = len(arrayInitial)-1

    matrice = np.zeros((classes, classes))

    for i in range(arrayLength):
        matrice[arrayInitial[i], arrayInitial[i+1]] += 1


    #Change the matrice with probabilities by line (normalization)
    for i in range(classes):

        sumline = 0;
        for j in range(classes):
            sumline = sumline + matrice[i, j]


        for j in range(classes):
            matrice[i,j] = matrice[i, j]/sumline

    return matrice
```

*Figure 6 Stochastic matrix estimation*

### ii. Reduce

Reduce function combines stochastic matrices form all sequences for a single Id and combines them into a 3d tensor (that can be used in CNNs). Reduce function has to be:

- Commutative: *reduce(a,b) = reduce(b,a)*
- Associative: *reduce(reduce(a,b), c) = reduce(a,reduce(b,c))*. Associativity also means that we have to be able to combine 2d matrix and 3d tensor. If we have as input a 2d matrix we first transform it into a 3d tensor (that has one dimension 1).

```
def combine_tensor(a,b):
    if len(a.shape) != 3:
        a = np.dstack(a)
    if len(b.shape) != 3:
        b = np.dstack(b)
    return np.concatenate([a,b])
```

*Figure 7 Reduce function. It has to be commutative and associative.*

## 4. Conclusions

As mentioned in the introduction, the motivation for the presented work is to construct an anomaly detection model for linear accelerator LINAC 4.

So far the most important bottleneck is the speed of the database especially in parallel reading. Reading step for dataset of 26 Ids with max 100 sequences of 5000 timestamps for Id, takes approximately 1.6 hours (on 4 core virtual machine hosted on Swan). Pre-processing step (map reduce in Spark) takes only seconds in comparison (as evident form figure 4).

Another problem is the limitation of capabilities of Swan virtual machines. The most performant configuration available is 4 cores and 16GB of memory. In order to address the slow reading – especially in use cases with more IDs – it would make sense to employ a machine with more cores for parallelization of reading part.

Lastly the solution to host reading parallelization in Jupyter notebook is not ideal. Computationally expensive job can fail even to problems with the hosting of the notebook. For a production version of the approach it would make sense to migrate the reading part to computational cluster that supports general purpose parallelization such as computer clusters with Slurm scheduler.

Presented pre-processing methodology serves as a basis for implementation of anomaly detection techniques that are either based on hidden Markov models or the stochastic matrix itself. The main bottleneck of the presented approach – the reading of the raw data form the database – can be alleviated by using more performant machine for data reading. The presented methodology runs inside Python virtual environment and is as such capable of running on different hardware that was used in this project (for example on supercomputer cluster). The map reduce step computed in this work can relatively easily be adopted to be used with other parallelization frameworks such as Dask and deployed on more general clusters or high performance computing systems.

## 5. References

**Arpaia Pasquale [et al.]** Fault Detection on Fluid Machinery using [Journal]. - 2019.

**Beneventi F., Bartolini A. and et al.** Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools [Conference] // Proceedings of the Conference on Design, Automation & Test in Europe. - 2017.

**Borghesi A., Bartolini A. and et al.** A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems [Journal] // Engineering Applications of Artificial Intelligence. - [s.l.] : Elsevier, 2019. - Vol. 85. - pp. 634–644.

**Borghesi A., Bartolini A. and et al.** Anomaly detection using autoencoders in HPC systems [Conference] // Proceedings of the AAAI Conference on Artificial Intelligence. - 2019.

**Borghesi A., Libri A. and et al.** Online anomaly detection in hpc systems [Conference] // 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). - 2019. - pp. 229–233.

**Borghesi Andrea [et al.]** Online Anomaly Detection in HPC Systems [Journal]. - 2 2019.

**Géron Aurélien** Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems [Book]. - [s.l.] : O'Reilly Media, 2017. - ISBN: 1491962291.

**Kotsiantis Sotiris B., Zaharakis I. and Pintelas P.** Supervised machine learning: A review of classification techniques // Supervised machine learning: A review of classification techniques. - 2007.

**Kwon Donghwoon [et al.]** A survey of deep learning-based network anomaly detection [Journal] // Cluster Computing. - [s.l.] : Springer, 2017.

**Lan Zhiling, Zheng Ziming and Li Yawei** Toward automated anomaly identification in large-scale systems [Journal] // IEEE Transactions on Parallel and Distributed Systems. - [s.l.] : IEEE, 2010. - Vol. 21. - pp. 174–187.

**Molan Martin [et al.]** An Explainable Model for Fault Detection in HPC Systems (in preparation) [Journal].

**Molan Martin** Explaining the faults of HPC systems [Journal]. - [s.l.] : PRACE, 2019. - Vol. Summer of HPC final report.

**Y. Donon A. Kupriyanov, D. Kirsh, A. Di Meglio, R. Paringer, P. Serafimovich, S. Syomic** ANOMALY DETECTION AND BREAKDOWN PREDICTION IN RF POWER SOURCE OUTPUT: A REVIEW OF APPROACHES [Journal]. - Budva, Becici, Montenegro : Proceedings of the 27th International Symposium Nuclear Electronics and Computing, 2019.