AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Master Thesis

Development of Tools for Data Quality Control for ALICE Experiment at CERN, Using Machine Learning Methods.

Rozwój Narzędzi do Monitorowania Jakości Danych, dla Eksperymentu ALICE w CERN, z Użyciem Metod Uczenia Maszynowego.

Author:	*Bartłomiej Cerek*
Field of Study:	*Computer Science*
Thesis Supervisor:	*dr hab. Adrian Horzyk, prof. AGH*

Kraków, 2020

I would like to express my gratitude to my thesis supervisor for all the guidance, as well as to Professor Jacek Otwinowski and Sebastian Bysiak for the valuable advice.

Table of Content

1. Introduction

In recent years, **Machine Learning (ML)** and **Deep Learning (DL)** are frequently listed among the most prominent technologies. *"We are now solving problems with machine learning and artificial intelligence that were… in the realm of science fiction for the last several decades. And natural language understanding, machine vision problems, it really is an amazing renaissance."* [1]. This is a quote from Jeff Bezos, CEO of Amazon, one of the biggest companies in the world, that succeeded among others, because of the great usage of the data. It is not the only case. More and more industries use data science to create business leverage and to outrun the competition.

A natural question to put forward is, "Can these technologies also be used to help researchers?". The development of new machine learning tools and models is a primarily academic field, and up until now, its findings were mostly applied in business. Nowadays, also scientists are more willing to use techniques of **Artificial Intelligence (AI)** in their work. Some experiments in medicine, physics, or chemistry produce a huge number of data, and automated algorithms can be efficiently applied for finding patterns and dependencies that are difficult to be spotted by humans. Research facilities like **CERN** employ ML and DL to push forward the limitations of our knowledge [2]. Organized in 2014, The Higgs Machine Learning Challenge brought many AI enthusiasts together to work on solving High Energy Physics problems and encouraged the further application of ML in science [3]. This enthusiasm is generally growing as intelligent algorithms, tools and technologies are getting more available. Thanks to easy use and universality of the application, the toolkits and libraries like Scikit-learn for Python are getting a vast number of users across many specializations [4].

There are, however, problems and limitations in the use of machine learning that should be mentioned. Nowadays, many scientists discuss the so-called 'reproducibility crisis', a growing problem in metascience, which is the inability to replicate or reproduce scientific studies [5]. Since ML algorithms have been designed to find patterns in given data, even when there are none, they can fixate on noise and cannot asses their own uncertainty. As most AI models work as black boxes, they take specific inputs and produce outputs, and no explanation of their decisions is given. This can be not only deceptive, but sometimes even counterproductive, especially when the ultimate goal stays with the understanding of the underlying processes. Lack of the decision argumentation reduces trust in ML methods and can even make them unusable in the field like medicine, where the confidence in decisions is crucial.

These issues challenge machine learning and deep learning techniques also used for scientific research. The growing movement of ‘explainable artificial intelligence’ proposes new algorithms, which not only make a decision, but also provide argumentation. Tools for explaining choices of existing black-box models are investigated and developed [6][7].

The goal of this project is to propose and compare different machine learning and deep learning solutions for quality control tasks in CERN’s ALICE experiment while addressing the aforementioned issues. The scope of the project includes:

- analysis of sample dataset from ALICE detector,
- data mining,
- development of different classification models, and
- proposing pipelines for their usage in unsupervised and supervised cases.

Evaluation of the obtained solutions is also provided. An important aspect is the comparison of methods based not only on their accuracy but also on the possibility of practical implementation in the environment of the ALICE experiment.

The thesis consists of 8 chapters, bibliography, and appendix. The first part is dedicated to the introduction of quality control tasks in CERN’s ALICE and description of software tools used in the project. Following this, the in-depth analysis of the sample dataset is presented together with the conclusions. Chapters 5. and 6. contain propositions and implementation descriptions of classic machine learning as well as deep learning models that can be applied as a solution to the presented problem. Advantages, disadvantages and possible workflow pipelines of each method are presented and discussed. The following chapter touches the topic of AI expandability, presenting the most suitable applications. The overall results, possible improvements, and final results are outlined in chapter 8, concluding the thesis.

2. Quality Control in CERN's ALICE

The following chapter describes the quality control problem in the CERN's ALICE experiment. The facility purposes and the detector are summarized before presenting the task requirements and evaluation guidelines.

2.1. Introduction

Conseil Européen pour la Recherche Nucléaire (CERN) is a research organization, focused on particle physics. It was established in 1954 and is located on the Swiss/French border, near Geneva. CERN is well known for its particle accelerators used for high-energy physics research and for advancing engineering and computer science [9]. W and Z bosons (1983), as well as Higgs boson (2012), were discovered there [10]. Moreover, CERN is the birthplace of the World Wide Web (www).

Large Hadron Collider (LHC) is the largest particle collider in the world. It was designed and built at CERN and started operation in 2008. LHC has 27 km circumference and is currently able to reach the energy of 6.5 TeV per beam. It consists of 1 232 superconducting, dipole magnets that curve the beam to keep it on the circular path. In addition, the quadrupole magnets help to focus the beam. Collider was constructed to obtain an answer to some fundamental, open questions in physics, most notably regarding the Standard Model and mass of elementary particles [11].

CERN's experiments are located on LHC's intersection points. The biggest of them, i.e. ATLAS and CMS, are general-purpose particle detectors. Thanks to them, Higgs boson was discovered. Others are dedicated to specialized research. For the purpose of this thesis, **A Large Ion Collider Experiment (ALICE)** will be presented in more detail. Its goal is to study the collisions of heavy ions (like plomb) to improve understanding of **quantum chromodynamics (QCD)**, the strong-interaction part of the Standard Model. Beyond that, ALICE can take proton beams data at the highest LHC energies to collect references for the heavy-ion program. ALICE weighs 10 000 tones and is located underground, close to the village of Saint Genis-Pouilly in France [12] [13].

Operations of LHC are divided into active periods and ‘Long Shutdowns’. Experiments and measurements are carried out during the active periods, while Long Shutdowns (LS) are intended for maintenance and introduction of new technologies. For the period between 2019 and 2021, CERN is in Long Shutdown 2. During this time a great upgrade of experiments is being performed. After 2021, the ALICE detector is planned to produce a huge amount of data (more than 3 TB/s). To cope with that, new systems for processing and quality control have been introduced [14].

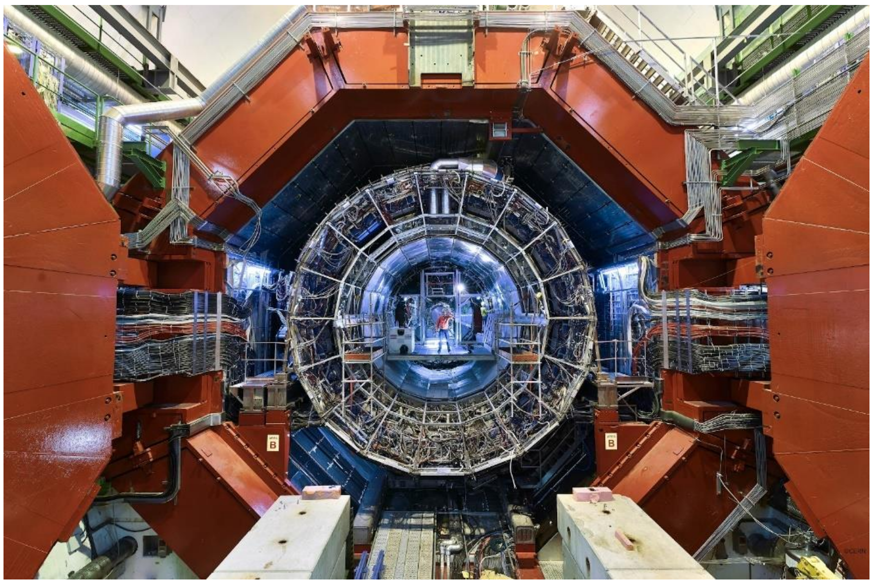


Image 1. Opened ALICE detector [15].

The goal of data quality control in LHC’s detectors is to select the correct measurements required for storage and analysis. All anomalies and acquired outliers should be discarded. In this thesis, the focus is put on data from the Time Projection Chamber (TPC). TPC is the main tracking detector in the ALICE experiment, which also serves particle identification. Measurements have been divided for ‘runs’, which include data from the single fill of the LHC, usually lasting up to 12 hours. Moreover, runs are divided for ‘chunks’, short 8 - 15 minutes periods of data acquisition. Finally, one sample of data to be analyzed is a chunk from a given run, that contains raw and derivated parameters.

Raw, low-level features are related to reconstructed particles’ tracks like a mean number of measured points in the particles' trajectories. Some others are related to global particles’ tracks, like the distance between the vertex and the extrapolated track. There is also a group that gathers information from all reconstructed tracks and checks the efficiency and quality of extrapolation of the tracks to the neighboring detectors.

2.2. Anomaly Detection

Data cannot be easily analyzed by humans due to its multidimensionality. Since the parameters of the experiment can change, it is also challenging to set fixed limits on them. The current approach to automated outliers detection is based on gathering data from a certain time frame and calculation tools and standard deviation for all parameters. Then, for all features, a warning or outlier flag was set depending on how many sigmas sample has deviated from the mean. Three sigmas corresponded to warning and five to an outlier. In the end, flags for all parameters have been combined. Consequently, if for at least one parameter the warning was set, the sample got a warning label, and similarly, if at least one parameter had the outlier flag then the sample was labeled as an outlier.

This solution has many drawbacks, e.g. the obvious one is oversensitivity to singular deviated features. When data is highly multidimensional, as in this case, the singular skewed feature does not have to indicate the sample anomaly. A comparison of just simple statistical parameters does not grasp their underlying relations. On the other side, the approach has some advantages, it is fully automated, quick and does not require any intervention from personnel. Moreover, it can point out the most problematic features, e.g. the strongest deviations.

The quality assurance problem seems like a great application case for machine learning, as it requires finding patterns in a large amount of multidimensional data. There are, however, a few possible ways to apply ML to this task. First, would be supervised learning. In this case, the model would learn to assign labels based on some training set that was already marked. It could, for example, learn to imitate the classic approach, which does not bring much value. The situation would be different if an expert user would label some of the data points. Then, the model could learn to imitate the operator. This still sets some limitations, i.e. if one model is used for all types of runs (not regarding varying base parameters of measurement itself), it can generalize too strongly. If many models are used, the user would have to mark each dataset for training separately, and that could be very time-consuming.

The perfect approach would be a completely unsupervised solution, a model that can learn to separate data by itself. This model will not only save time, that have to be invested in creating training datasets, but also possibly learn new ways to execute the task and outperform former automated outlier selection, as well as a manual one.

2.3. Anomaly Explanation

Going further, if a machine learning model could learn how to detect outliers in a new, more robust way, it would be greatly informative explaining its choices to humans. Consequently, it would be easier to build and raise users’ trust. A common problem with ML models stays with the fact that they work like black boxes. The user is specifying inputs and expects certain outputs, but inner working, and the algorithm remain unknown. Models are often intended to work this way, but nowadays more and more scientists aspire towards having **Explainable Artificial Intelligence (EAI)**. It aims to create solutions that not only can predict answers but also support them with arguments, for example, by pointing out features of the tested sample that mostly influenced prediction.

If this approach is applied for quality control task in CERN’s ALICE, researchers could not only use machine learning as an automated solution for outliers detection that they could truly trust, but also possibly obtain new information regarding data, which was not considered before.

3. Software Tools

The development of tools for data quality control for the ALICE experiment is a software project focused on data analysis. Adhering to best practices of data science research, CERN applies primarily open-source and currently maintained tools. This chapter provides brief descriptions of the most important of them that are used in the project.

2.1. Python

Python is an interpreted, general-purpose programming language that was created by Guido van Rossum and released in 1991. It is multi-paradigm and highly extensible [16] [17]. Thanks to its flexibility, readability, and user-friendliness it is one of the most widespread programming languages in the world. Python is a language of choice of data scientists and machine learning engineers [18]. It is delivered with an extensive standard library, which contains the most important functions and structures for basic software development. The great advantage of Python is the fact that it is an open-source project maintained by multiple contributors.

Python can be extended by downloading and installing external packages developed by the community. For instance, pip is often used as a package management tool, however, Conda is a popular choice of data science-related developers. It is dedicated and also open-source package manager, which tracks dependencies of installed libraries and allows for creating separate virtual environments to handle conflicts [19].

Most of the code and documentation for the project were developed in Jupyter Notebooks, web-based interactive computational environments [20]. They allow for transparent and well-annotated software prototyping and data analysis. Each chapter of the thesis corresponds with a set of notebooks, guiding through all steps of the reasoning process.

Image 2. Python logo [21].

2.2. Data Manipulation and Computation Libraries

The popularity of Python among the data science community is mostly determined by the availability of powerful, open-source libraries that handle data. As they are structured in Pythonic (following guidelines of PEP-8) style, they are easy to use and intuitive [22]. Most of them are written in C (using Python C API) for low-level memory management and high performance. Crucial packages used in this project are:

- **Numpy,** a basic package for scientific computing. It allows for robust and seamless operations on big matrixes of numeric data and contains functions for linear algebra. Numpy performs much faster calculations than similar implementations written in pure Python [23].
- **Pandas,** a library for general data manipulation, allowing to group chunks of multi-type vectors as data frames, data management and execution SQL-like queries. Pandas are a great choice for exploratory analysis as they contain many functions for summarizing and visualizing basic parameters of data. They can be used to conveniently load and save data frames from and to files [24].
- **Matplotlib,** a toolkit used for visualization purposes. It contains tools for the easy creation of elegant graphs, plots and schemas. All graphical elements generated in this project utilize the Matplotlib module, pyplot [25].

Image 3. NumPy, Pandas, and Matplotlib logos [26].

2.3. Scikit-learn

Scikit-learn, often also referred to as sklearn, is an open-source machine learning library. It is built on top of NumPy, SciPy, and Matplotlib, so that it can be seamlessly integrated with most data science projects developed in Python. As it is distributed on a BSD license, it can be freely used commercially [27].

Sklearn contains a variety of working out-of-box models for regression, classification, clustering, and other standard ML tasks. Linear Regression, SVM, Gaussian Mixture or Naive Bayes are just to name a few popular examples. The user is also equipped with preprocessing and utility functions that cover most of the repeatable parts of projects. An important aspect of the library is standardized work pipelines. The majority of classes in Sklearn implement similar methods and properties. Having understood the workflow of one model, the user can perform the same steps with the majority of other models and expect similar behaviors. This intuitiveness, along with an approachable learning curve and great functionality, make Scikit-learn the first choice for many machine learning projects.

The library comes with detailed documentation and many examples that can be starting points for development. It also provides comparisons of different models and lists of criteria for the selection of the right tool. Most of classic machine learning models used in the project were trained and evaluated using sklearn functions.

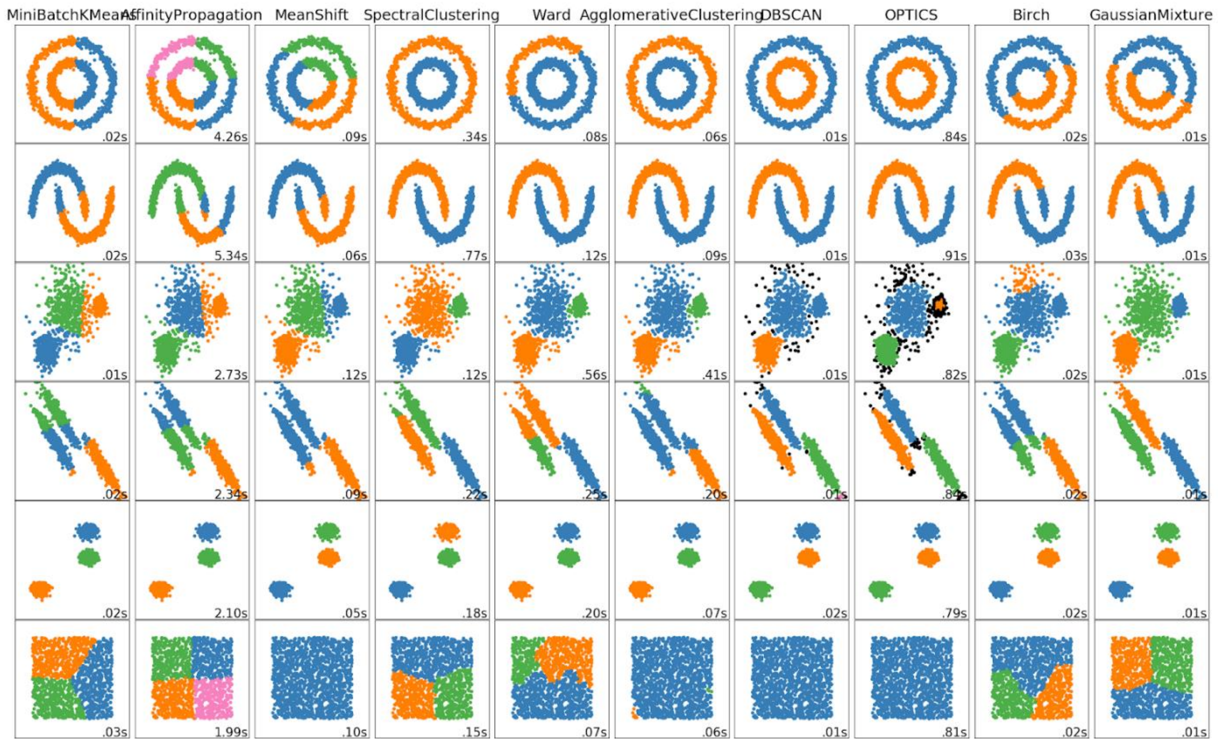


Image 4. Comparison of the clustering algorithms from scikit-learn documentation [27.]

2.4. PyTorch

PyTorch is an open-source Python library based on Torch written in Lua language. It is mainly developed by Facebook AI Research Lab for deep learning tasks. Many of the most commonly used DL models, being part of libraries like Uber’s Pyro or HuggingFace’s Transformers, are created in PyTorch [28].

Library provides functions for advanced tensor computations, including Graphics Processing Units (GPU) acceleration with Compute Unified Device Architecture (CUDA). It also implements crucial elements required for the creation and training of artificial neural networks like optimizers, loss functions, and datasets handlers. A big advantage of PyTorch stays with the automatic differentiation package (autograd). It tracks operations performed on tensors, and independently calculates required gradients, making backpropagation steps very easy to implement.

In general, PyTorch is more flexible than its major competitor Keras (being part of the TensorFlow package as of version 2.0), but it requires a more low-level approach. The bigger code overhead is necessary for simple neural networks, however, just as Keras, the package comes with ready to use standard layers like dense, convolutional or recurrent. To make easier use of PyTorch, packages have been developed to provide wrappers with sklearn-like interfaces. One of the most notable examples is Skorch [29].

PyTorch

Image 5. PyTorch logo [28].

4. Data Analysis

It is a good practice, to perform in-depth data analysis and preprocessing before applying any machine learning models. This chapter elaborates on the dataset used in the project and describes its properties. It also guides through the process of obtaining a lightweight version of the dataset.

4.1. Exploratory Data Analysis

Typically, the first step to undertake when dealing with the new dataset is **Exploratory Data Analysis (EDA).** It is an approach to data on the purpose of which is to sum its main characteristics, using basic statistical tools and visualization [30]. It was popularized by John Tukey, recognized American mathematician and creator of Fast Fourier Transform (FFT). EDA allows the user to quickly get a general orientation in the dataset and formulate the first hypothesis. Even though, if not very precise, it can be a good starting point. The basic parameters of the sample dataset from the ALICE detector, were collected in *Table 1*. Descriptive parameters of all features were calculated afterward.

Table 1. Basic parameters of data from the ALICE detector.

Samples	Incomplete Samples	All Cols.	Metadata Cols.	No Variation Cols.	Labels	Features
3429	0	257	18	6	2	231

A simple conclusion that can be drawn from shapes of the obtained data frame, is that we are dealing with strongly multidimensional data while having not that many samples. Around **3500**, is enough to train most of the simple machine learning binary classifiers, however it can be insufficient for more complex models and deep learning.

After dropping metadata and columns with no variation (all the same values), **231** features which can be used for classification, were left. There are also two binary labels: **alias_global_Warning** and **alias_global_Outlier**. They were both placed by a former automatic outlier detection algorithm, and can be used as a baseline for accuracy tests of created models. They can also be used to some degree for labels simulation placed by an expert. In this kind of anomaly detection problem, it is preferred to have high **recall** (probability of detection) to surely detect all outliers even at the expense of a few false positives. Afterwards, specific thresholds for use cases can be established. Therefore, in all further chapters, more sensitive, **alias_global_Warning** will be used as a default label of the former method. Only **4,9%** of all samples were marked with this label.

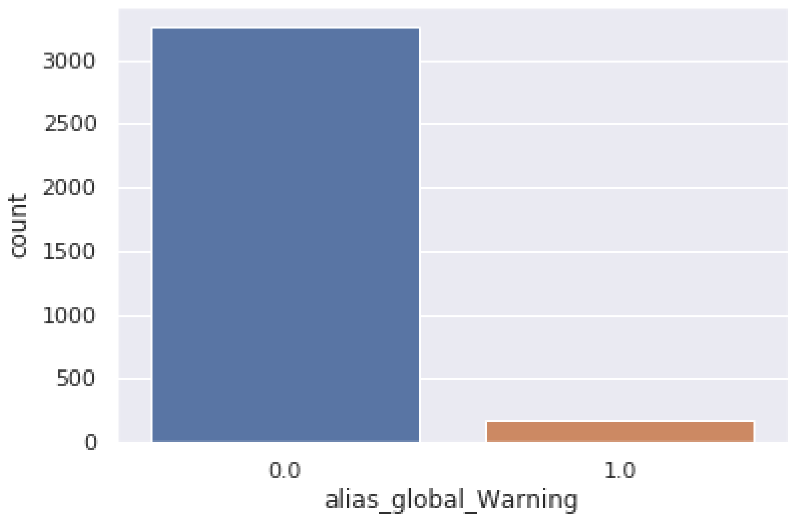


Image 6. Bar plot comparing the number of inliers to the number of outliers in the sample dataset.

As it was mentioned in chapter **2**. **Quality Control in CERN's ALICE,** some features derivate from others, taking us to the assumption of high correlations between them. Moreover, as the correlation of each feature to the label was examined, it turned out that many of them (70+) show close to no correlation.

To obtain a visual representation of this issue a matrix of correlations was generated for all relevant features and labels, as it is presented in **Appendix A**. Since is not easy to go through the matrix of 233 x 233 elements, all the irrelevant entries below diagonal and elements with absolute correlations below the arbitral value of **0.75** were left blanc. Even after filtering, there were **163** high correlated pairs left. Taking into consideration 231 features, the number is high, although it is not that informative. All these features could, e.g. correlate to only one other feature. The generated matrix shows us the quite coherent spread of pairs. That indicates that, in general, there is a big **redundancy** in the dataset. This can cause problems while working with machine learning models and waste computational resources.

The two most important points of attention, which can result from this exploratory analysis are as follows:

- dataset is relatively small, but highly multidimensional and imbalanced, and
- there is a big assumption of **data redundancy**.

Taken into consideration all the aforementioned inputs, a conclusion can be made that **dimensionality reduction** is a natural next step in the preprocessing phase.

4.2. Features Selection

Dimensionality reduction is a process that aims to limit the number of considered random variables and obtain a set of principal variables [31]. It is usually applied when the amount of data to be analyzed is overwhelming or parts of it is redundant. Additional, useless information not only makes it difficult to operate on data but also slows down training and using machine learning models. It can also create confusion for algorithms and reduce their accuracy [32].

Conventional approaches to dimensionality reduction are either removal of some variables or transforming data to contain the crucial information, from original data, in a smaller set of variables. In recent years, the last method was widely spread thanks to very efficient techniques like Principal Component Analysis (PCA) or Autoencoders [33] [34]. The problem with those is that the obtained set of parameters is difficult for analysis by humans. Information like [height, weight, clothing size] can be quite successfully compressed by PCA to a two-element vector [V1, V2]. This new data can be used for classification tasks, but the new features are generally meaningless if not further processed.

One of the main goals of the project is to explain why a certain sample was classified as outlier or inliner, while pointing out features that mostly influence the model's decision. Because of that, dimensionality reduction based on the creation of a new set of features is not applicable, even though it could bring better accuracy.

Feature selection is a simpler technique that aims to select the best parameters for further classification applying some criteria, e.g., correlation with the label. Usually, good criteria can be how strongly the accuracy of the selected model decreases after removing the feature from the training set. **Recursive Feature Elimination (RFE)** is a method that applies this approach recursively and allows for ranking all features from most useful to the most redundant one [35]. Scikit learn contains (working out of the box) implementation of RFE that is compatible with most of the standard models. Selecting a model for use is, however, not that trivial. Since many different classifiers are used for this project, performing multiple refittings would take a lot of time and resources. Moreover, some of them will be trained in a not supervised manner, so it can be challenging to specify the accuracy metric that RFE could use. It would be more convenient to specify one model that universally will select the best parameters, even with some error margin.

As the main goal is to identify if the new sample is outlier or inlier, simple binary classifiers could be a right choice for that. **Logistic regression** was selected for the task with 3-fold cross-validation, as the most appropriate one to predict **alias_global_Warning** label.

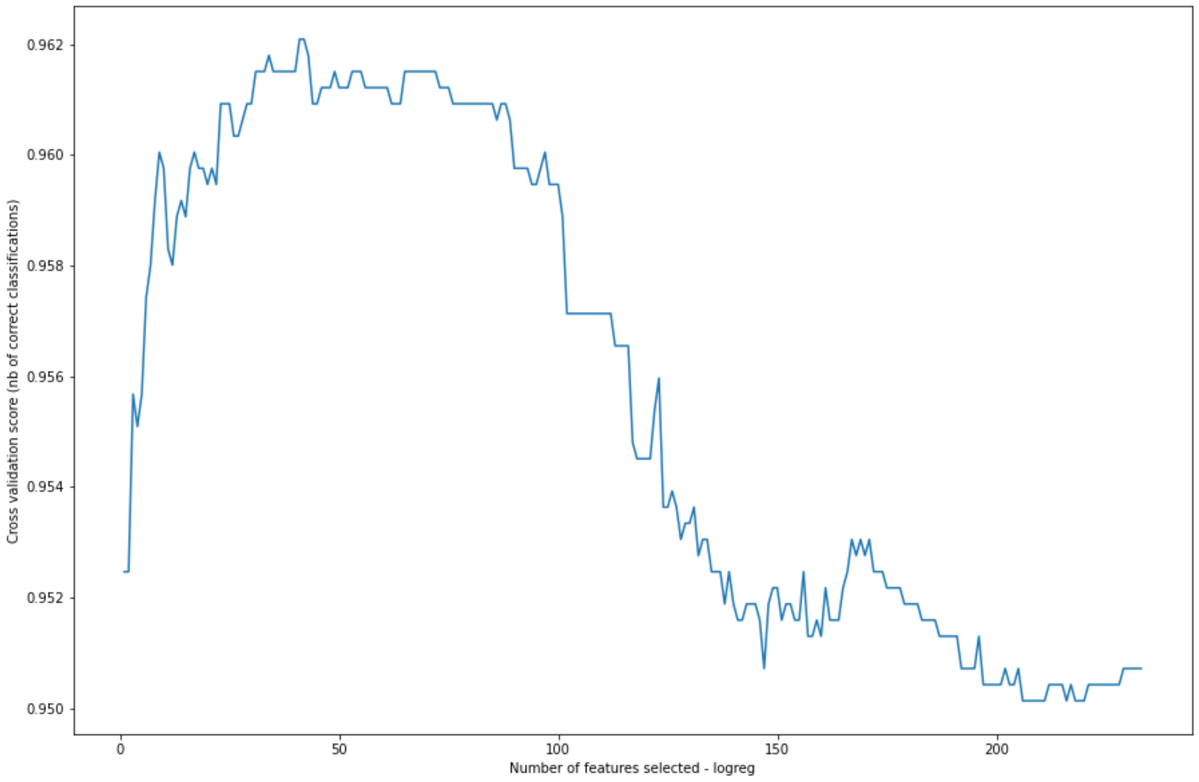


Image 7. Cross-Validation Score for different numbers of used features.

After performing RFE with the classifier of choice, features were ranked and saved in order in the file for further tasks. The number of parameters to use can be selected by finding the point where the accuracy is the highest one or arbitrarily, by balancing accuracy and resources. *Image 7.* shows how the cross-validation score is changing when more features are added. Initially, it is rising quickly as crucial data is added to the model. Best accuracies are achieved around 40 and 90 used features, and then the score is dropping as more redundant parameters are being introduced.

It is worth to notice two things. The range, in which values are changing, is very small. The lowest recorded score is above 0.95 while the highest is below 0.97. This means that logistic regression had no problems learning how to imitate a former automated detection algorithm, and that was expected since it was using only statistical parameters. An interesting point for consideration is that a very small number of features was enough to almost fully reproduce its effect.

Recursive Feature Elimination allowed to determine which variables are important from the classification point of view. However, the remaining features should not, be blindly rejected. The goal is to surpass the classic algorithm for outliers detection, not to imitate it. When new models are introduced, their performance may be improved by not using some of the redundant features, and this has to be validated.

Physicists working on ALICE data made their own list of **sensitive features**. These 47 features are crucial from the point of view of scientists interpreting raw data, who can usually figure out that the sample is an outlier. A comparison, between features, selected by RFE and sensitive features was performed. It turned out that among the top **100** parameters (out of 231) ranked by REF by means of logistic regression, there were 19 sensitive features. It does not seem to be many, however, since data contains derivated features, which are sometimes non-linear transformation of others, they can be actually more informative for the models and allow for easier classifications.

In conclusion, when pure machine learning classification is performed, parameters selected by the RFE method are the most appropriate ones. The situation is different when explainability is taken into account. It might be more beneficial to append sensitive features to RFE-selected ones and check how algorithm evaluates their impactfulness.

5. Classic Machine Learning

The following chapter is focused on applying machine learning methods (not involving deep learning) to solve the problem of data quality control in CERN’s ALICE experiment. This approach nowadays often referred to as classic machine learning. It makes use of tools like regressions and decision trees. Supervised, unsupervised, and novelty detection techniques will be described and compared in this chapter as well [37].

5.1. Supervised Outlier Detection

Chapter **4.2. Feature Selection**, touches the usage of machine learning models for assessment of features usefulness. It was **linear regression**, one of the simplest binary classifiers. These models utilize labeled training set to learn prediction on which one of two categories a new sample will belong. In the quality assurance problem, this technique can be used to mimic the classic automatic algorithm, by using for training labels generated by it. More interestingly it can also lean to make decisions like the expert user.

To check the accuracy of the abovementioned solution, three popular classifiers were trained and compared. Due to the absence of the expert data, labels assigned by the automatic solution were used for the proof of concept. Classifiers were run on two datasets. Full, containing all 233 features, and lightweight, containing only 133. The lightweight dataset is a union between 100 most important features selected by RFE, and sensitive features selected by ALICE physicists. 80% of data was used for training and 20% for testing. Stratification was applied to ensure a similar distribution of outliers in both groups. As the dataset is highly imbalanced, it contains much more inliers than outliers. Apart from accuracy, **balanced accuracy** was used as a metric. The following classifiers were used for the tasks:

- **Logistic Regression** - the most common approach to model is the probability of binary events such as win/lose, pass/fail. It is based on estimating the parameters of the logistic function. It is possible to apply regularization methods to this function, i.e. l1 and l2, in order to improve its ability to generalize [38].
- **Random Forest -** ensemble method, which uses bagging of multiple decision trees created from partial sets of features with bootstrapping. It is one of the most robust and well generalizing methods employing trees [39].

- **Support Vector Machine** - an algorithm that places hyperplane in between two sets of samples from different classes. Kernels can be used for spotting separation where the linear split is not possible [40].

Table 2. Comparison of the accuracy of different supervised classifiers.

Classifier	Full Dataset		Lightweight Dataset	
	Accuracy	Balanced Acc.	Accuracy	Balanced Acc.
Logistic Regression	90.96%	90.83%	98.83%	87.80%
Random Forest	99.42%	95.38%	98.83%	94.76%
Support Vector Machine	98.32%	94.24%	99.13%	92.35%

Based on the above results the following conclusions can be drawn:

- similar to the one from subchapter **4.2. Features Selection**, i.e. simple machine learning algorithms are able to reach balanced accuracy by around 90% without any fine-tuning, therefore, imitating standard algorithm is relatively easy to apply;
- the newly created, lightweight dataset, even though it is almost half smaller, allows for obtaining quite similar results.

5.2. Unsupervised Outlier Detection

Unsupervised outlier detection is far more practical as it does not require a labeled training set, although using it introduces new challenges. The most important one is the evaluation of the result. Groups obtained by unsupervised ML algorithms can be compared with labels from the previous automatic approach, however, since it is not ground truth and new techniques are expected to **outperform** the former approach, this is not the most suitable method.

Quality control data is highly multidimensional, so even for an expert user, it is not easy to decide whether the sample is an outlier. It would be convenient to reduce the dimensionality of the data for visualization purposes in order to get some intuition about the localization of anomalies. For this task, **t-distributed Stochastic Neighbor Embedding (t-SNE)** can be applied. It allows for nonlinear dimensionality reduction by creating probability distribution over samples in multidimensional space, then a similar distribution in lower-dimensional space and reducing **Kullback–Leibler divergence** between them. Those distributions are created to give a high probability of pick to pairs of similar objects and low to dissimilar objects. t-SNE iteratively decreases differences between them and as a result, similar samples are close to each other in a low dimensional space, therefore, a visual inspection becomes possible. Unfortunately, just as in the case of PCA, obtained features cannot be easily interpreted. Another drawback of t-SNE is relatively slow execution. However, it doesn't seem to be an issue in case of rarely performed visualization [41].

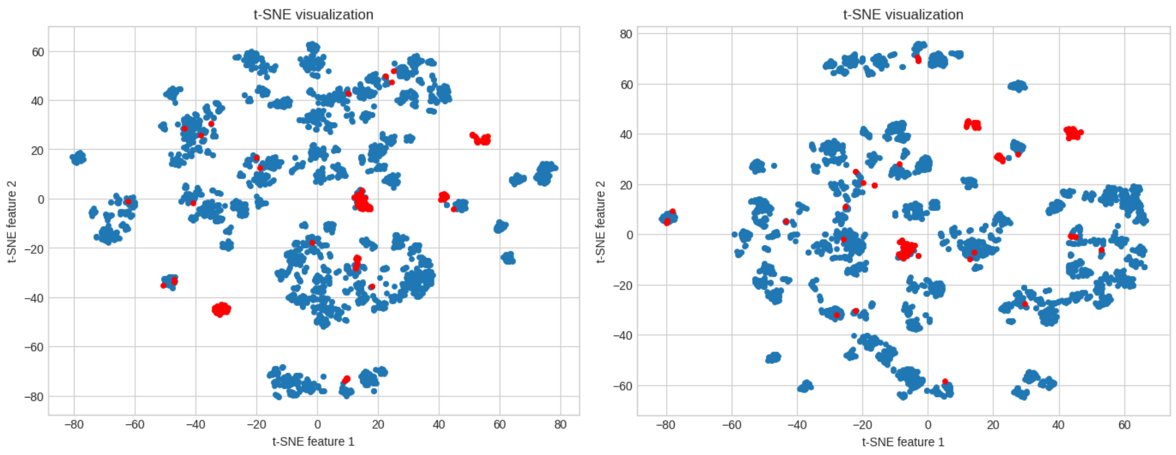


Image 8. t-SNE visualization of normal (left) and lightweight (right) datasets with samples marked as outliers by the former automated method marked in red.

Having the ability to visualize anomaly detection is the first step of the result evaluation. t-SNE method for both versions of the dataset created many sparse clusters with few outlying regions rather than one big blob and surrounding outliers. It means that data is not coherent and probably the correct samples vary strongly. The analyzed set contains measurements from around 200 runs of LHC. Samples from each run can be marked with a different color. Those that have a longer time frame between them will have a bigger difference in color.

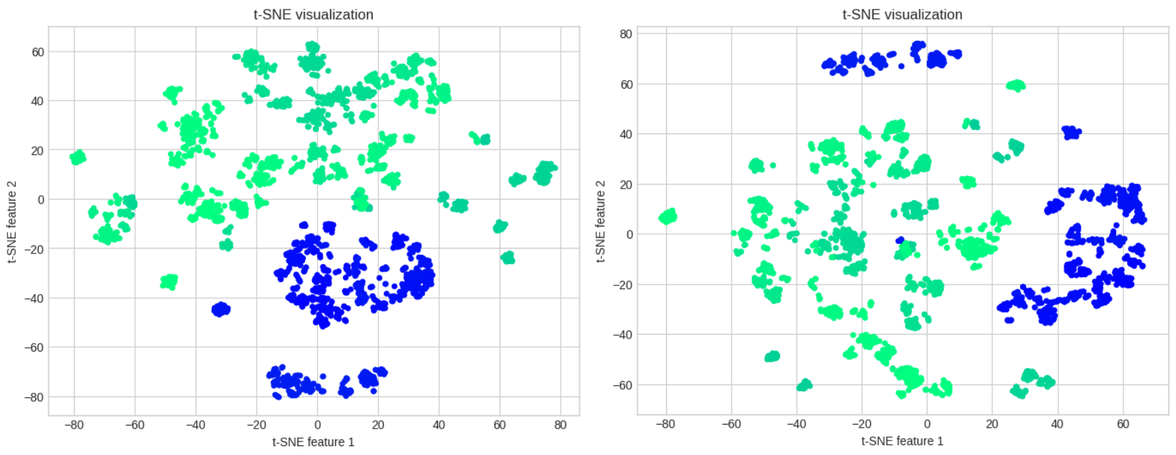


Image 9. t-SNE visualization of normal (left) and lightweight (right) datasets with different runs marked with different colors.

This way, it is clearly visible that there are considerable differences between runs and the longer timeframe between them, the more visible they are. This is another proof that finding anomalies for quality control is not a trivial task. It is worth noticing that the orientation of samples does not play an important role as t-SNE only tries to preserve similarity by distance, while the exact location is not critical.

General methods for initial evaluation, i.e. t-SNE visualization and comparison with the former algorithm, are established so unsupervised training can be performed. As there are no impactful differences between normal and lightweight datasets since now only results from lightweight will be presented in the thesis, while full can be found in corresponding Jupyter Notebooks.

One of the most popular classic machine learning algorithms for cauterization is the **k-means** algorithm. It is a simple technique to group data by minimizing the sum of squares within clusters as a criterion [42]. After setting the number of required clusters to 2 (inliers and outliers) the algorithm was run on data, and results were marked on t-SNE visualization. See *Image 10.*

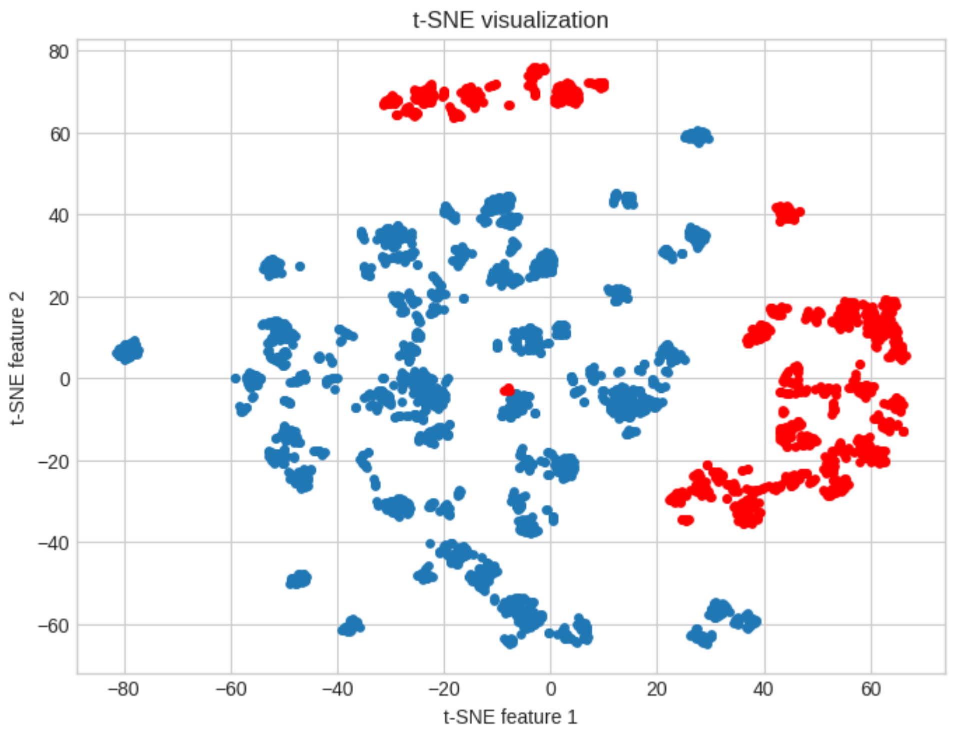


Image 10. t-SNE visualization of lightweight datasets with k-means clusters marked.

The visualization shows that k-means did not bring expected results. Trying to create clusters with similar variations, the algorithm failed to select outlying samples. Valuable information that can be retrieved from the comparison of the results presented on *Image 9* and *Image 10*, is that k-means separated the two biggest groups of runs. The big difference between them has to be taken into account when applying further methods.

More sophisticated algorithms for clustering are **Density-based spatial clustering of applications with noise (DBSCAN)** and **Ordering points to identify the clustering structure (OPTICS)**. They are based on the exploration of low and high-density regions in data and do not require the determination of the expected number of clusters. Moreover, they are able to point out outliers [43][44][45]. With this approach not only unsupervised detection is obtained, but also information about differences between samples groups (e.g. differences between runs in case of this dataset) can be found and presented to the user. DBSCAN is generally less sensitive to irregularities in data; therefore, it was used for the task, and the results were marked on the same t-SNE visualization as previous methods. The behavior of the algorithm can be modified by two key parameters;

- **eps,** which is the maximum distance that two samples can have between each other to be considered as in the neighborhood of the other, and
- a **minimum number of samples** that have to be in the neighborhood of point to consider it as the core point.

Changing those will adjust the DBSCAN for specific goals. In the case of ALICE quality control, relatively strong generalization is required, as detecting multiple correct clusters does not bring any new information. From the other side, sensitivity is still important to detect anomalies. For the selection process two arbitrary requirements were set:

- Number of correct detected clusters should be below **10**
- Percentage of found anomalies should be **between 2 - 5%**

Using the **grid search** method parameters eps=0.7, min_samples=30 were selected.

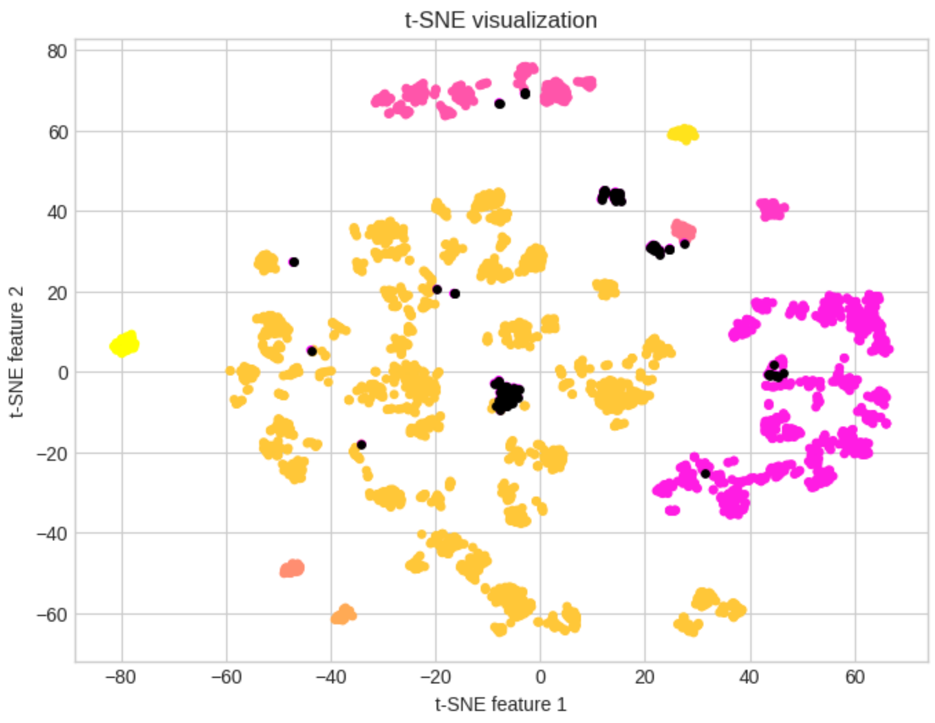


Image 11. t-SNE visualization of lightweight datasets with DBSCAN clusters marked, outliers are black.

DBSCAN parameterized accordingly found 8 clusters from which the 3 biggest one corresponds to the biggest groups of runs, as seen on *Image 9*. It also located 134 (3,9%) of anomalies from which 104 were also flagged by the former automatic algorithm. This seems like a very promising result, DBSCAN turned out to be less restricting than comparing deviation from the mean, leaving out samples that had only a few outlying features.

5.3. Dedicated Anomaly and Novelty Detection Algorithms

Problems of novelty and outlier detection were often tackled as supervised binary classification problems. Usual unavailability of training sets and their lack of balance were forcing researches to move to unsupervised methods. Simple techniques dedicated to anomalies, like computing robust covariance of samples, proved to be useful only for coherent datasets. At the beginning of the 21st century, new, more specialized methods were developed. One of the most popular ones is the **Isolation Forest (iForest)**.

Isolation Forest utilizes the fact that outliers are easier to separate from the core of the dataset vs. normal samples. Just like Random Forest, it is an ensemble method using trees, in this case, called iTrees. Multiple iTrees are being created by randomly splitting the dataset in a manner to isolate each sample. Fewer separations are required to isolate the sample, the higher is its probability of being an anomaly. In the end, all created trees vote, and in this way, final decisions are being made. This straightforward algorithm is highly universal. It can be applied to most datasets without any preprocessing, and it requires only to determine the number of trees to be an ensemble. It is very fast, and on top of that has a low memory requirement since it has linear time complexity [46].

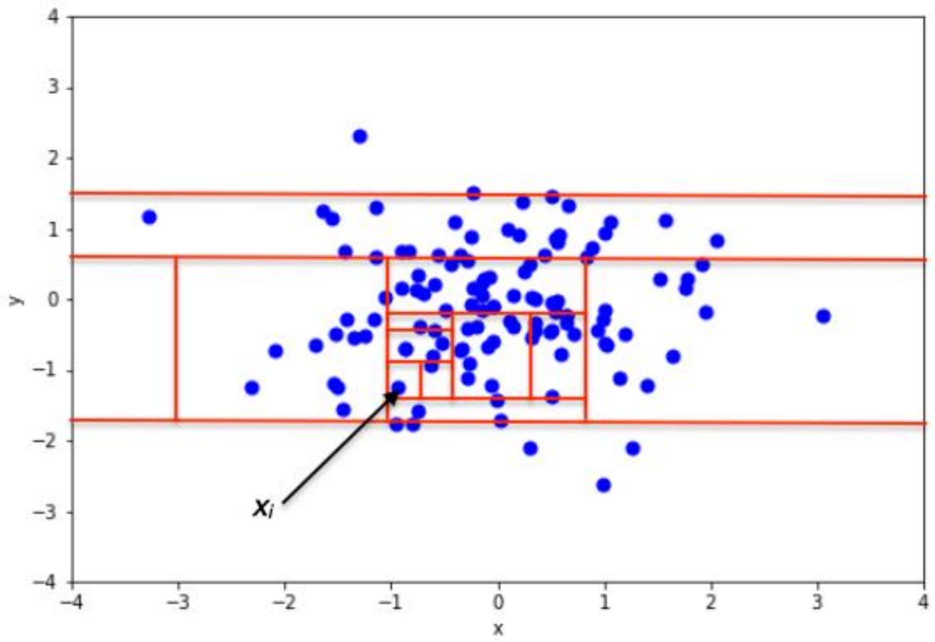


Image 12. Example of random separations made by Isolation Forest to find outlier [47].

All these characteristics make Isolation Forest a perfect choice for quality assurance tasks. The classifier, with a default number of iTrees (100), was fitting the lightweight dataset. Detected outliers are marked on t-SNE visualization.

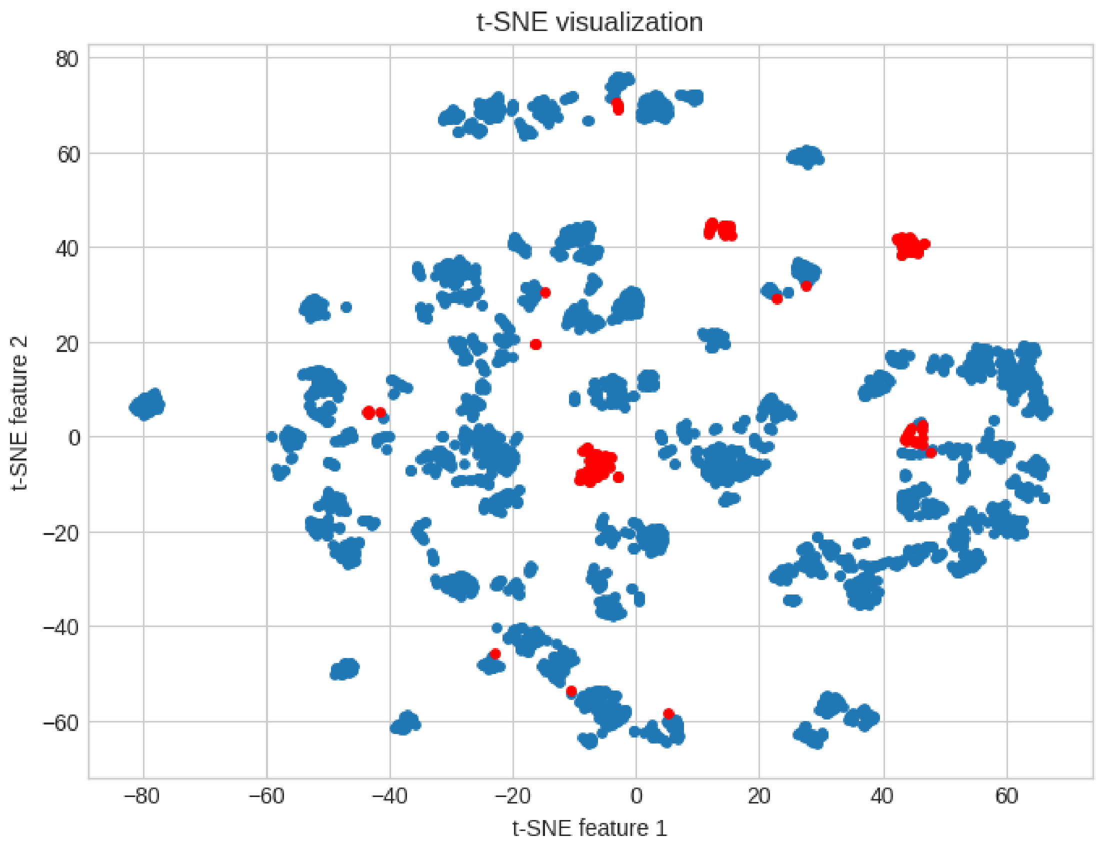


Image 13. t-SNE visualization of lightweight datasets with Isolation Forest outliers marked.

Isolation Forest detected **176** anomalies, which indicates a slightly higher number than the one identified by the former algorithm. Out of those, **131** are matching, and again, it shows a very good alignment. From visual comparison on *Image 8*, it is clear that the same big concentration zones of anomalies are marked by both methods. There are, however, differences, i.e. Isolation Forest marked more samples around contracted zones and omitted many singular chunks labeled as outliers by a simple method. Most probably, they are false positives.

In conclusion, it seems that DBSCAN and Isolation Forest are the most suitable classic machine learning algorithms for the ALICE quality control task. The later method provides greater sensitivity for outliers and fast execution, while DBSCAN can bring additional information about differences in correct data. A full comparison between methods and juxtaposition with different approaches is presented in chapter **8. Summary and Conclusions.**

6. Deep Autoencoders

The following chapter is focused on leveraging deep learning, by using deep autoencoders to create better machine learning models for data quality control in CERN's ALICE experiment. The chapter also describes training and evaluation of autoencoders for outlier detection, and provides a comparison of two common AEs architectures.

6.1. Introduction

Deep learning is a subclass of machine learning that utilizes **Artificial Neural Networks (ANNs).** When first introduced around 1960, it gathered considerable attention from the scientific community despite its mediocre effectiveness. Neural networks gave way to more robust and accurate classic machine learning methods. Things started to change during the last two decades. New architectures, training methods, and optimizers were introduced. The processing power of GPUs rose dramatically, and open-source projects and tools made deep learning more accessible. These days, almost all state-of-the-art machine learning solutions are based on ANNs. On certain tasks, models outperform expert human users, and as the amount of available training data and speed of computing hardware are growing, further developments in the field are expected [48].

Just like in the case of classic machine learning algorithms, neural networks can be trained for supervised, unsupervised, and semisupervised tasks. In general, deep learning is expected to yield better results for the price of more complex fine-tuning and longer training. As was outlined in the previous chapter, the approach to training binary classifiers on ALICE quality control data for outlier detection is relatively easy and not useful. Because of that, in this chapter more focus will be put on the revision of unsupervised deep learning methods, i.e., **Autoencoders (AEs)**. In another CERN's experiment, CMS, autoencoders were used successfully for a similar anomaly detection task [49].

Autoencoder is a neural network that learns in an unsupervised manner to encode data in latent space (bottleneck), which usually has lower dimensionality than the input. It is built of two parts, i.e. the encoder that compresses the data and the decoder that tries to recreate input from compressed representation. In most cases the network is trained to minimize the difference between input and output [50].

6.2. Simple Autoencoder

A standard autoencoder is built using only fully connected layers, where the one in the middle has the smallest number of neurons, while encoder and decoder are mirrored in both, i.e. numbers of hidden layers and neurons. This kind of network can be trained to very well compress and decompress data of a certain type, e.g. pictures of handwritten digits. One of the most popular datasets for benchmarking machine learning models, MNIST, contains exactly such digits [51]. During the learning process, AE is exposed to multiple samples of digits, which are first compressed to latent space, and then decompressed to output., Metric, like **Mean Square Error (MSE),** can be used as a loss function. It will compare input and output and evaluate how well the neural network managed to reconstruct the picture; in other words, it will calculate **Reconstruction Error.**

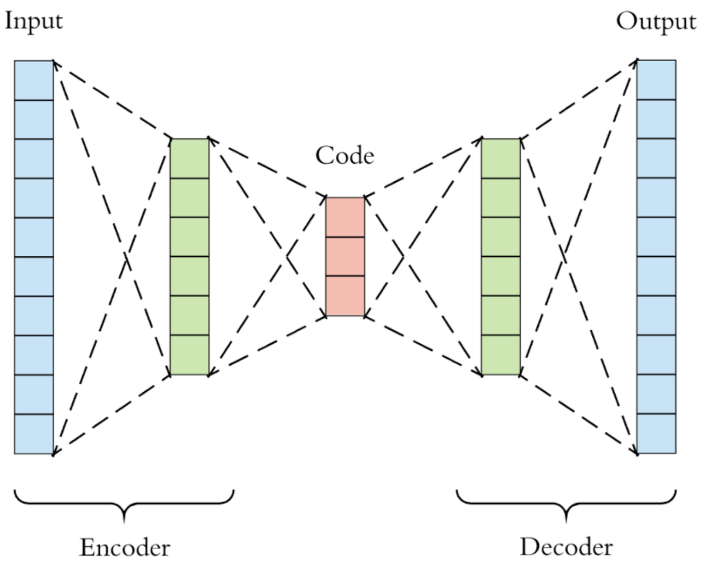


Image 14. Visual representation of autoencoder [52].

Autoencoder learns only the most common properties of handwritten digits. All noise, and sample-specific artifacts are lost in latent space. The smaller the bottleneck is, the stronger is the generalization, and the worse is the reconstruction. This gives trained autoencoder interesting property. The new image, obtained by forwarding digit picture through the network, will be very similar to the input provided that it is normal, where normal means that many samples, with very similar characteristics, are present in the training dataset. If the input picture contains anomalies, autoencoders will not be able to recreate them. Consequently, the output will be a generalized version of the input. By using the MSE metric it can be figured out how difficult it is for a model to reconstruct the input.

This behavior allows for using autoencoders as an unsupervised anomaly and novelty detectors. When AE is trained, there is an assumption that a number of outliers in data are small and their characteristics are not going to be learned. Then, new data is passed through the network, the output is compared against the input, and MSE or another metric are used to calculate reconstruction error, and provide information on how strongly the sample is skewed. Autoencoders can be used not only with simple data like MNIST. In recent years architectures, which utilize convolutions and recurrence, were adjusted with AEs, thus allowing work with complex images and sequential data [53].

Difficulties in using autoencoders are similar to most common deep learning and unsupervised learning problems. The architecture of the network and hyperparameters have to be tuned to the use case and it is not easy to evaluate obtained results. The training dataset should be bigger than the one used with most classic machine learning algorithms in order not to cause overfitting. Moreover, the threshold of reconstruction error which indicates if a given sample qualifies as an anomaly has to be set arbitrarily if no ground truth is provided.

A simple autoencoder built of fully connected layers was employed for the ALICE's quality control task. *Table 3.* presents crucial parameters of the model and training process with comments. To avoid overfitting the PyTorch model was trained on a lightweight dataset with 3-fold cross-validation. Due to the relatively small amount of data, it converged quickly. Changes in training and validation loss can be seen in *Image 15.*

Table 3. Parameters of the Simple Autoencoder model and training process.

Parameter	**Value**	**Comment**
Number of layers	5	2 layers for the encoder, 2 for the decoder and 1 latent layer
Number of neurons per layer as a fraction of input size	[0,5-0,3-0,1 -0,3-0,5]	Since datasets with different numbers of features can be used, sizes of layers are expressed as fractions of numbers of input features. e.g. If the dataset has 100 features: [50-30-10-30-50]
Dropout	0.3	The fraction of random neurons that are being dropped in the training process.
Epochs	50	The number of training cycles.
Batch size	128	The number of samples per minibatch.
Learning Rate	0.00007	The learning rate is very low to avoid getting stuck in a local minimum.
Optimizer	Adam [54]	A popular optimization algorithm for DL.
Criterion	MSE	Mean Square Error.

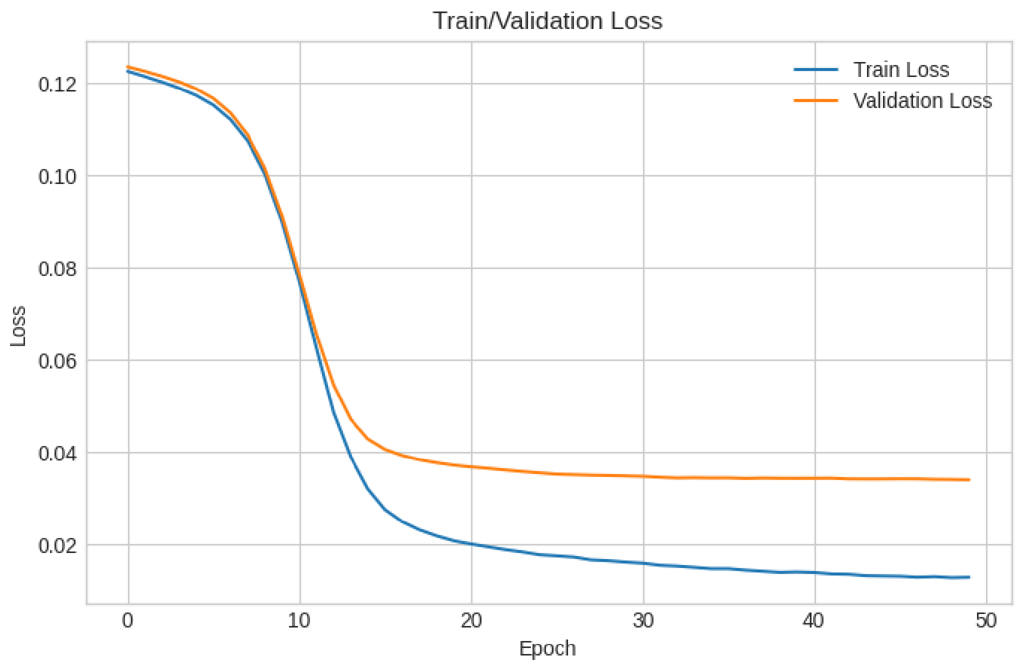


Image 15. Changes in training and validation loss during training.

The trained model was used to calculate the reconstruction errors of all samples in the dataset. A convenient way to visualize their distribution is a boxplot graph, presented on *Image 16.*

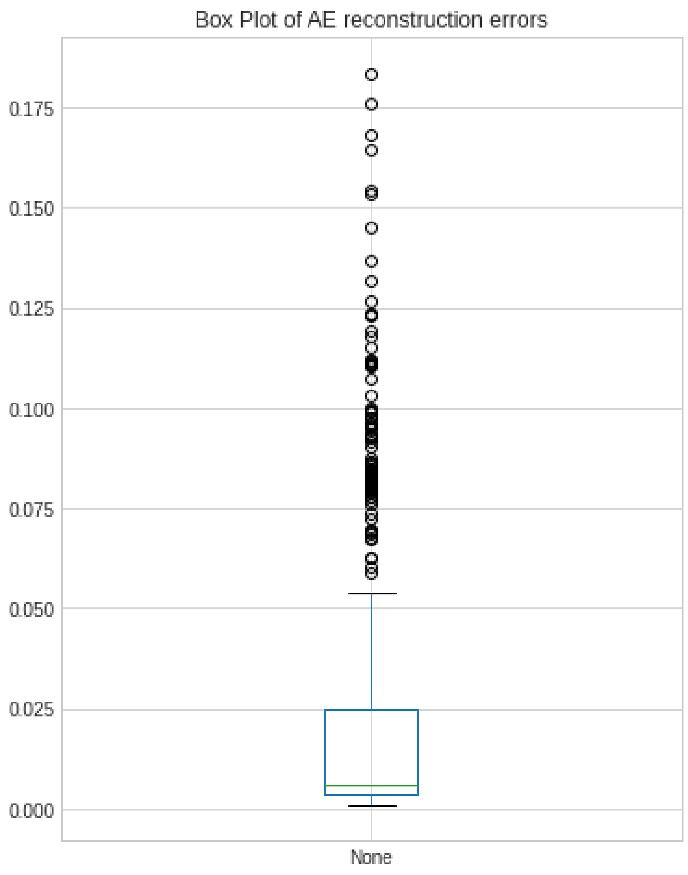


Image 16. Boxplot of simple AE reconstruction errors of samples in the lightweight dataset.

Most of the data (around 75%) yielded very small reconstruction errors, i.e. below **0.025**. These are normal samples, that were recreated well, even though the latent layer forced 10 times compression (0.1 fractions of neurons in bottleneck). All samples above the upper mustache of the box are definitive outliers, with strong errors. The problem is deciding if samples in space between these two zones are anomalies. Placing the threshold can be adjusted depending on the preferred property of classification, high recall, or high precision. In the case of the data assurance task, it is known that around **5%** of samples were marked as outliers by the former algorithm, which could be oversensitive to singular skewed features. Because of that, it can be assumed that the real number of outliers is slightly smaller, and the arbitrary threshold was set to **0.039**, so that **4%** of samples are labeled as outliers. This way **106** anomalies pointed out by the former algorithm were also chosen by autoencoder. Outliers marked by both methods can be seen and compared on *Image 17.*

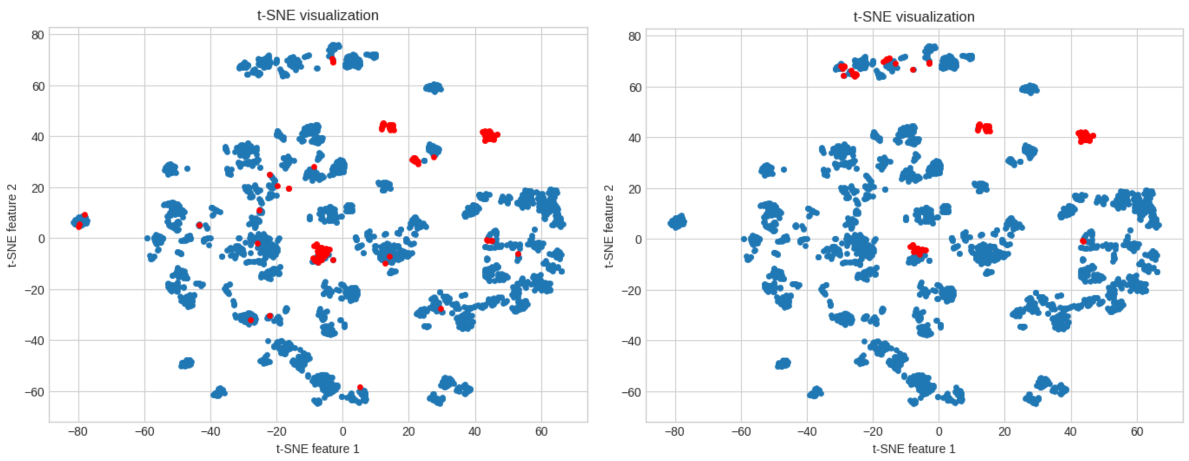


Image 17. t-SNE visualizations of lightweight datasets with outliers selected by the former algorithm (on the left) and simple autoencoder (on the right).

If ground truth, like an expert labeling, is available, better functions for evaluation of the autoencoder are **Receiver Operating Characteristic (ROC) curve** or the **Precision-Recall curve**. ROC curve is obtained by plotting True Positive Rate against False Positive Rate at different classification thresholds. It allows for the selection of the best possible cut-off point. Similar logic is applied to the second curve, just with recall and precision functions. In both cases, **Areas Under the curve (AUCs)** can evaluate how well classifiers performed in general. A good practice for both types of curves is to plot with them the line of the naive classifier, applying random labels. It helps to get quick intuition if correct classification does not drop at certain point to unacceptable level [55]. Having only labels selected by the former algorithms; these curves can be used to check how strong the new model agrees with its predecessor.

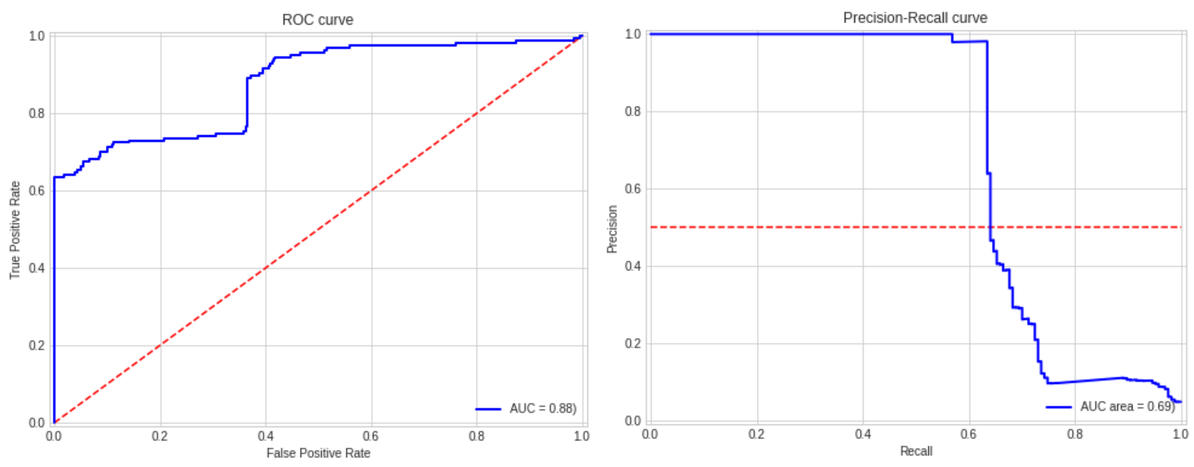


Image 18. ROC and Precision-Recall curves using scores of simple autoencoder and labels assigned by the former algorithm as ground truth.

It was proven that simple autoencoder could be trained to perform anomaly detection. By analyzing t-SNE visualization and level of agreement with the former algorithm, it can be concluded that effects are satisfying. As automatic methods of fine-tuning deep learning models are getting popular, selecting hyperparameters should not be a significant obstacle. The only remaining problem is selecting the reconstruction error threshold. A **semi-supervised** technique is the best approach for that. The model trained in an unsupervised manner could be evaluated on a smaller sample of expert-labeled test data. Based on ROC and Precision-Recall curves the best cut-off point will be selected. This way, the proposed solution can be applied in practice without setting any arbitrary values.

6.3. Variational Autoencoder

Although well designed and trained on a large amount of simple data autoencoder performs very well, there are other architectures, which in certain cases can outperform it. A notable example is **Variational Autoencoder (VAE).** The model covered in the previous subchapter was of **discriminative** type. It was trying to calculate the conditional probability of Y, given X, which can be expressed as P(Y|X). VAE, on the other hand, is a **generative** model that finds joint probability distribution of X and Y, P(X, Y). The most common use cases for these types of networks are generating new, real-like samples. They can work like Generative Adversarial Networks (GANs), but they are simpler to train and use. Unfortunately, typically, they are also less effective. Beyond that, VAEs in different configurations can be used for unsupervised anomaly detection, and in this case, they can surpass standard autoencoders [56].

To obtain the generative nature of the model, both architecture and loss function need to be adapted. The overall, hourglass-like, shape of the layers remains unchanged, but now the encoder returns two vectors. One of them represents **means** and other **standard deviations**, so from a practical point of view, it returns a set of distributions. Decoder samples from those distributions and tries to recreate input. The random sampling exposes the decoder to a greater variety of smooth samples from latent space. From that moment, the reconstruction is not deterministic.

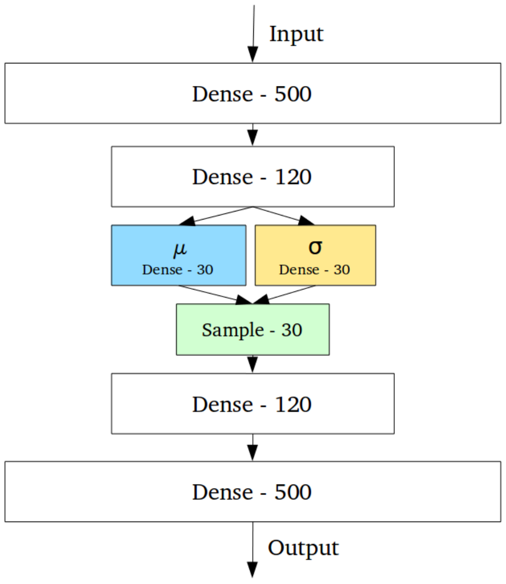


Image 19. Example architecture of Variational Autoencoder [57].

If left only with these modifications, an autoencoder will learn to return narrow and pointy distributions, so that sampled values do not vary much, and in fact, behave just as standard AE. To prevent that from happening, a new term is introduced to the cost function. Besides the reconstruction error, the loss also consists of the **Kullback-Leibler (KL) divergence** term. KL divergence between two distributions determines how different they are. For two identical distributions, it is 0. In VAE, distributions obtained from the encoder are compared using KL divergence to standard, normal distribution ($\mu = 0$, $\sigma = 1$) [58].

This approach to loss functions ensures that AE during training recreates input as good as possible, but also punishes it for returning unnormal distributions. This keeps output varied and ensures the generative nature of the autoencoder. Random sampling introduces a problem with backpropagation since the gradient cannot proceed through the random node. However, it can be solved by using a so-called ‘reparameterization trick’ [59].

Defined as described, variational autoencoder can be used to detect anomalies in a similar manner to a standard autoencoder, just after taking the average of few output samples for a given input. PyTorch implementation of the model was trained on the same data as the previous autoencoder. Crucial parameters with comments of the model and training process are summarized in *Table 4*.

Table 4. Parameters of the VAE model and training process.

Parameter	**Value**	**Comment**
Number of layers	5	2 layers for the encoder, 2 for the decoder and 1 latent layer
Number of neurons per layer as a fraction of input size	[0,5-0,3-0,1 -0,3-0,5]	Since datasets with different numbers of features can be used, sizes of layers are expressed as fractions of numbers of input features. e.g. If the dataset has 100 features: [50-30-10-30-50]
Dropout	0.3	The fraction of random neurons that are dropped in the training process.
Epochs	50	The number of training cycles.
Batch size	128	The number of samples per minibatch.
Learning Rate	0.0001	The learning rate is low to avoid getting stuck in a local minimum.
Optimizer	Adam [54]	A popular optimization algorithm for DL.
Criterion	MSE + KL Divergence	The loss function consists of two terms, MSE ensuring similarity of outputs to input and KL Divergence ensuring normality of encoded distributions.

For the purpose of evaluations consistency, after training of VAE the same steps were performed as in the case with the standard autoencoder. *Image 20*, shows distributions of reconstruction errors.

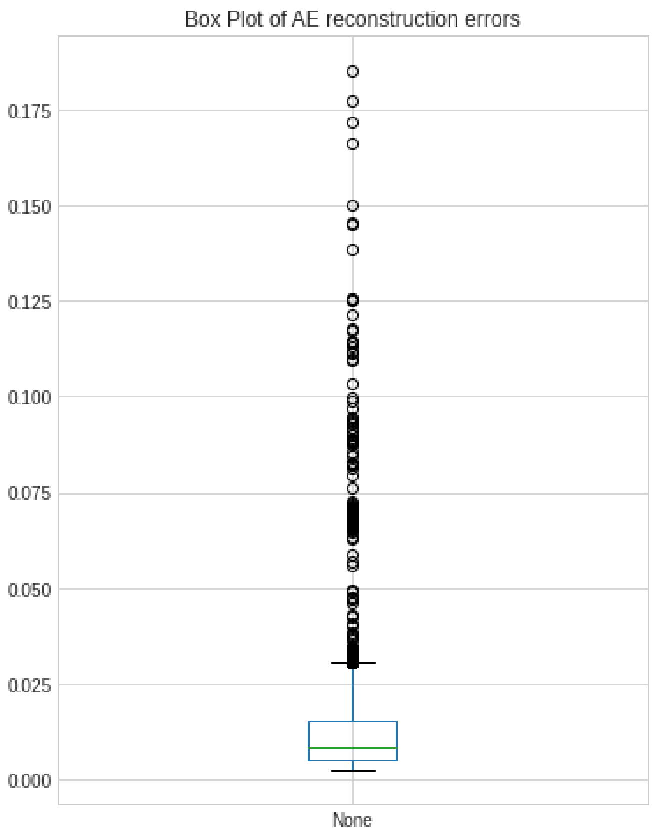


Image 20. Boxplot of VAE reconstruction errors of samples in the lightweight dataset.

The range of calculated errors is almost the same as in the case of the simple autoencoder. However, it can be noticed that the concentration of normal data is stronger, and the threshold cutting off **4%** of most outlying values was set to **0.047**, which is a higher number than it was distributed previously. There is not enough data to fully confirm this hypothesis, but obtained errors' distribution may indicate better separation abilities of VAE. Inliers are concentrated around one value while anomalies are strongly skewed in the reconstruction process.

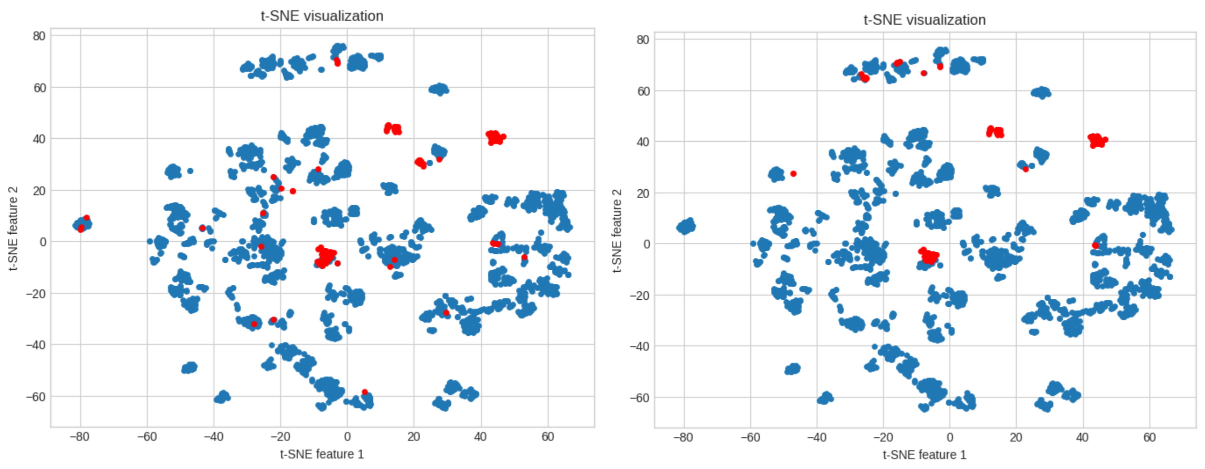


Image 21. t-SNE visualizations of lightweight datasets with outliers selected by the former algorithm (on the left) and VAE (on the right).

Visual inspection of t-SNE embeddings leads to the conclusion that almost the same groups of anomalies are detected by simple AE and VAE with only a few minor differences. **116** chunks were marked as outliers in agreement by both VAE and former algorithms, which is slightly more than it was in the previous method.

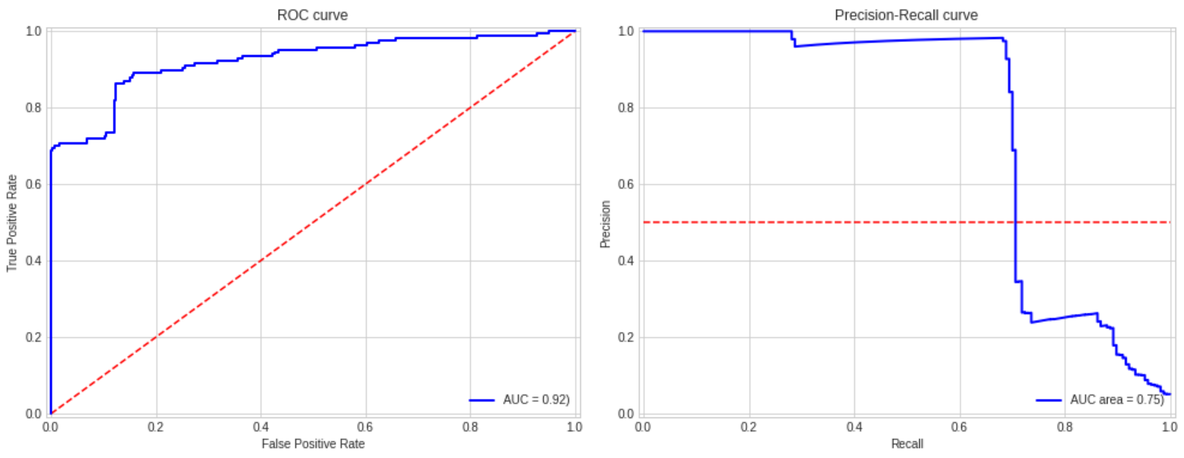


Image 22. ROC and Precision-Recall curves using scores of VAE and labels assigned by the former algorithm as ground truth.

Further similarities can be found comparing *Image 18* and *Image 22*. ROC and Precision-Recall curves are alike on both pictures, but areas under curves are bigger in the case of VAE. That, and better agreement in chunks selection, point out that the second architecture of the autoencoder obtains results closer to the former algorithm. It does not necessarily mean that VAE is better in anomaly detection. It just created a decision boundary, which is closer to the old approach.

Both architectures of autoencoders managed to perform quality control on ALICE's data and find samples that realistically can be anomalies. Usage of deep learning can give the edge of capturing completely new, underlying relations in data and outperform even expert users. The choice between simple AE and VAE is not easy as there is no proper ground truth that can be used for benchmarking. There is a probability that keeping new results more similar to the former approach, as it’s done by VAE, is a better solution. It’s obvious that simple AE found a new, better way for the detection of outliers. The hint that could be found in the performed result analysis is that the variational approach seems to produce easier for dividing distributions of reconstruction errors.

Deep learning reveals its full potential when applied to a great amount of data, so the next step in researching autoencoders for quality assurance tasks should be training and evaluation on a bigger dataset. As results already obtained from both architectures are quite similar, it might be the case that after training on more samples, two networks will behave identically. On the contrary, results may also diverge, and make an easier choice. For any outcome, the final decision on which architecture to be chosen should be made in consultation with data experts.

7. Model Explainability

The following chapter explores possibilities to explain decisions made by classic machine learning and deep learning models introduced in 2 previous entries. Usage of the LIME technique [6] and native properties of autoencoders are described and compared with the former, statistics-based approach.

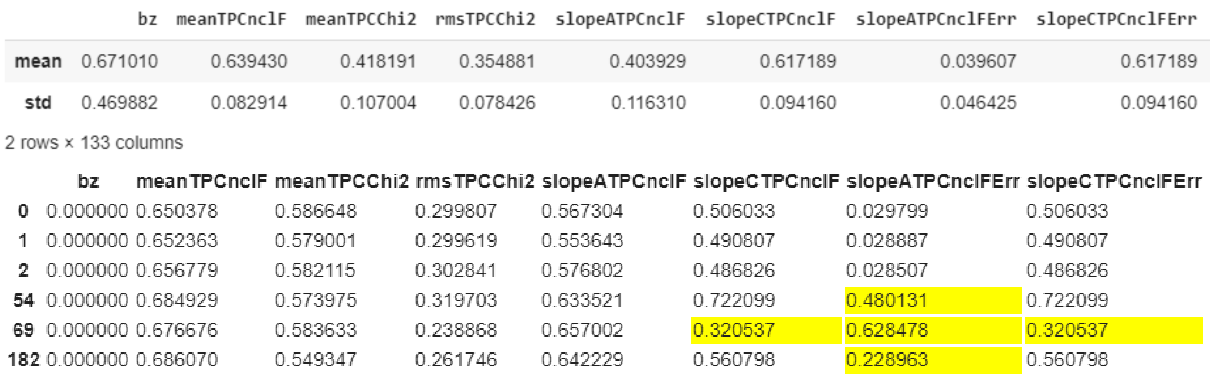
7.1. Introduction

Assessing trust in machine learning models is a growing concern [60]. Vast adoption of automated solutions needs to go in pair with building users’ confidence in them. It is especially true in cases where the decision made by the algorithm has a great impact. Examples can be self-driving cars and computer-made medical diagnostic decisions. Ensuring the explainability can also allow for a more precise and conscious evaluation of the models. Analysis of usual performance metrics like accuracy and ROC curves sometimes cannot give enough insight and can be even misleading. A great example of this type of situation can be found in the LIME research paper [6]. Image classifier was trained to differentiate Husky dogs and wolfs, resulting in a good evaluation accuracy. However, all training pictures of wolves were taken in the forest, in a winter setting. Precise analysis has shown that the model learned not to distinguish animals but to detect snow in the picture. The tool used to make this finding is presented in this chapter.

What can be recognized as a proper model explanation? In the simplest and most useful case, pointing out which features of the analyzed sample influenced model decision. For image classification, it would be regions (groups of pixels) of the picture that allowed for the distinction. For standard tabular data regressor, input elements that were most important for classification. For outlier detection tasks, it could be information on which features cause sample abnormality.

The explainability of anomaly detection models, which would be used in the ALICE experiment has two crucial benefits. First of all, just as in the abovementioned examples, it will build users’ trust in the solution. They can be sure that rejected data is truly abnormal by inspecting the values of features pointed out by the model. Secondly, they can also gain new information about problems in the experiment. Abnormality of obtained samples, caused by the skewness of a certain group of parameters, allows for reasoning about problems with LHC beam or detector settings.

The former approach to the quality control task was based on calculating the deviation from the mean values of all features. This explains that the solution can point out features, which are most skewed, and this brings some insight into the experiment. The problem here is that the simplistic approach did not take into consideration any relations between parameters.



	bz	meanTPCnclF	meanTPCChi2	rmsTPCChi2	slopeATPCnclF	slopeCTPCnclF	slopeATPCnclFErr	slopeCTPCnclFErr
mean	0.671010	0.639430	0.418191	0.354881	0.403929	0.617189	0.039607	0.617189
std	0.469882	0.082914	0.107004	0.078426	0.116310	0.094160	0.046425	0.094160

2 rows × 133 columns

	bz	meanTPCnclF	meanTPCChi2	rmsTPCChi2	slopeATPCnclF	slopeCTPCnclF	slopeATPCnclFErr	slopeCTPCnclFErr
0	0.000000	0.650378	0.586648	0.299807	0.567304	0.506033	0.029799	0.506033
1	0.000000	0.652363	0.579001	0.299619	0.553643	0.490807	0.028887	0.490807
2	0.000000	0.656779	0.582115	0.302841	0.576802	0.486826	0.028507	0.486826
54	0.000000	0.684929	0.573975	0.319703	0.633521	0.722099	0.480131	0.722099
69	0.000000	0.676676	0.583633	0.238868	0.657002	0.320537	0.628478	0.320537
182	0.000000	0.686070	0.549347	0.261746	0.642229	0.560798	0.228963	0.560798

Image 23. Mean and STD values for the first 8 features in the lightweight dataset (above) and values of those features for the first 3 inliers and 3 outliers (below). Values being 3 sigmas away from the feature mean were highlighted.

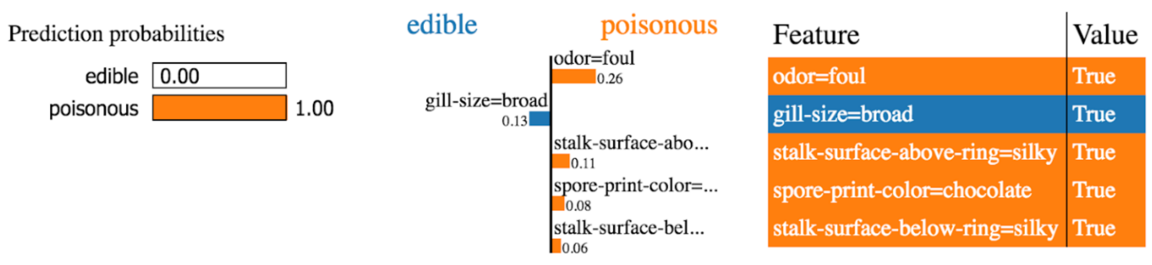
The methodology of the former approach is presented on *Image 23*. One may be tempted to judge based only on the first 8 features (out of 133 in the lightweight dataset) that sample **69** (5th on the picture) is the strongest anomaly. All presented chunks that were labeled as outlying have skewed **slopeATPCncIFErr** parameter, while only sample 69 has also strongly deviated **slopeCTPCnclF** and **slopeCTPCnclFErr**. Making a decision based only on the number of features that have values 3 sigmas away from the mean is an oversimplification of the problem.

7.2. LIME with Isolation Forest

Explainability techniques can be divided into model **specific** and model **agnostic**. The first group is strongly correlated with the model itself, and often the explanation is made based on some inner mechanics of the algorithm. Being often very robust, these techniques cannot be widely used and in many cases are impossible to apply. Model agnostic methods aim to create universal solutions, explaining all kind of black-box classifiers without ‘looking inside’. This ensures their quick integration with existing models and keeps great separation of responsibilities.

One of the most popular agnostic model explainability method is **Local Interpretable Model-Agnostic Explanation (LIME)** [6]. It is a local technique, which means that argumentations behind classifier decisions are delivered on a sample-to-sample basis, and the whole dataset is not required. Inputs of the algorithm have already trained model returning class probabilities and analyzed samples. LIME will slightly modify the input and forward it through the provided classifier. This process is performed multiple times, while changes in the output are monitored and analyzed. Intuitively, this recreates the human behavior of finding the black box algorithm by looking for dependencies between input and output values.

As a result, the user is provided with the list of most important features with the estimation of their impactfulness, as well as information about to which decision they contribute. This technique was used to analyze Husky vs. wolf classifier performance. LIME was able to point out super pixels, which mostly were influencing the decision, and those were pixels depicting snow. A simple example of the explanation of binary classifier of fruit edibility based on binary features can be seen on *Image 24.*



Feature	Value
odor=foul	True
gill-size=broad	True
stalk-surface-above-ring=silky	True
spore-print-color=chocolate	True
stalk-surface-below-ring=silky	True

Image 24. Example LIME explanation of binary classifier.

LIME is very powerful, although not a perfect tool. In its basic implementation, it approximates local behavior using only liner models. In some cases, drastic non-linearities in the close surrounding of the sample can prevent the correct explanation. Moreover, as all methods that use features to generate explanations, LIME requires a human-understandable set of parameters. Outputs of PCA, t-SNE, or any set of embedded values used as inputs to the classifier will not provide any valuable information. One more, especially important from the perspective of this thesis, the problem is that LIME requires a black-box model that returns class probabilities. This is a standard behavior for most of the supervised classifiers and neural networks but it is not a case for most of the unsupervised techniques. As it was already determined in chapter **5. Classic Machine Learning**, methods that do not require labeled training sets are far more useful in the quality control task.

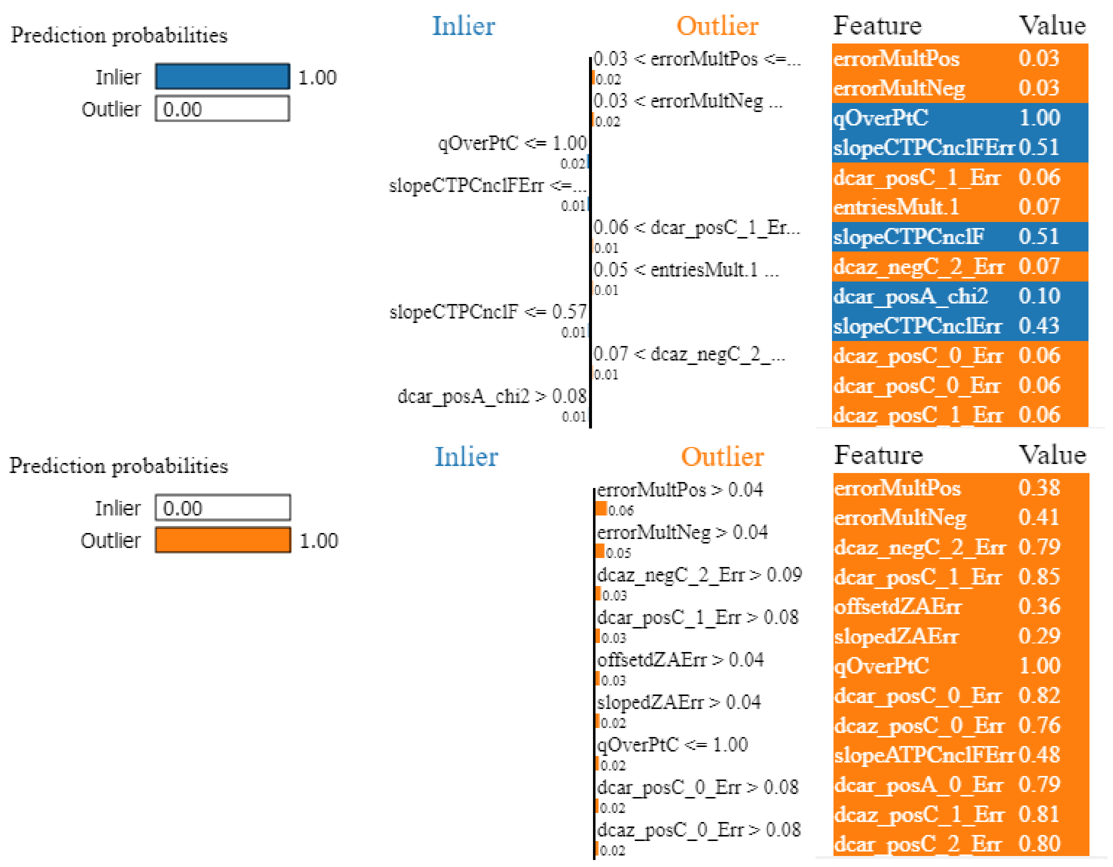
All the above-mentioned problems need to be addressed in order to employ LIME for outlier detection in ALICE. When it comes to local non-linearities it can be assumed that they are not present in a high number of parameters in the dataset. Even a lightweight dataset contains over 100 features. There is a very low probability that most of them would be highly nonlinear in very narrow intervals. The lightweight dataset was created exactly to reduce the amount of data while keeping meaningful features without the creation of embeddings. Therefore, the second issue is not applicable.

The third problem requires a workaround. Isolation Forest and DBSCAN turned out to be the most efficient classic machine learning solutions for the project task. Both of these techniques do not return class probabilities. DBSCAN can only define the number of neighboring points of the sample, which is not usable with LIME. Isolation Forest calculates anomaly scores, which can be returned; however, they correspond with the depth of the leaf containing observation in the iTree. For the whole dataset, scores can be normalized between 0 and 1 and used as a probability. It unfortunately does not have any theoretical foundation, disagrees with locality ideas and has proven to not yield good results. To successfully use LIME with Isolation Forest an intermediate step needs to be performed. Instead of explaining the decisions of iForest, another, a supervised classifier can be trained on the dataset and labels predicted by the unsupervised method, and after that it can be used with LIME. One can assume that the second binary classifier will be able to reproduce anomaly detection abilities of Isolation Forest while providing class probabilities and allowing for meaningful explanations.

The perfect intermediate classifier will train fast with high accuracy. It should be resistant to the imbalanced dataset and return natively class probabilities. The dense neural network can be applied for this task, but it will require a balancing dataset and can take longer time to train. If classic machine learning can provide high enough accuracy it will be a better choice. Based on the comparison from chapter **5. Classic Machine Learning** Random Forest is the most suitable choice.

Random Forest classifier was trained on the lightweight dataset and labels assigned were by Isolation Forest. It was evaluated on 20% of stratified data, balanced accuracy of **100%** in 7 out of 10 training sessions on randomly split data. These results confirm the suitability of the selected algorithm.

The intermediate model was used with the LIME algorithm to create predictions and explanations. For each decision, the most influential features were selected and their contributions to the decision were calculated. *Image 25,* presents example explanations for chunks **0** and **54 (**first inliner and an outlier in the dataset as marked by the former algorithm).



Feature	Value
errorMultPos	0.03
errorMultNeg	0.03
qOverPtC	1.00
slopeCTPCnclFErr	0.51
dcar_posC_1_Err	0.06
entriesMult.1	0.07
slopeCTPCnclF	0.51
dcaz_negC_2_Err	0.07
dcar_posA_chi2	0.10
slopeCTPCnclErr	0.43
dcaz_posC_0_Err	0.06
dcar_posC_0_Err	0.06
dcaz_posC_1_Err	0.06

Feature	Value
errorMultPos	0.38
errorMultNeg	0.41
dcaz_negC_2_Err	0.79
dcar_posC_1_Err	0.85
offsetdZAErr	0.36
slopedZAErr	0.29
qOverPtC	1.00
dcar_posC_0_Err	0.82
dcaz_posC_0_Err	0.76
slopeATPCnclFErr	0.48
dcar_posA_0_Err	0.79
dcaz_posC_1_Err	0.81
dcar_posC_2_Err	0.80

Image 25. LIME explanations for Random Forest classifier trained on the lightweight dataset and labels assigned by the Isolation Forest. Sample 0, inliner (above) and sample 54, outlier (below).

Explanations provided for sample 0, which is inliner, are not easy to interpret. Although Random Forest is 100% sure about the normality of the sample, features that are used to support this decision are vague. Mostly LIME detected those ones that stand in opposition to this decision, and it signaled that their values are too low to impact the made a choice.

The quality of arguments is way higher in the case of chunk 54, an outlier. Most of the selected features are of error type, like reconstruction errors. LIME points out that the sample is an anomaly because of the high values. It can also be confirmed that most of the selected parameters have also values that are deviated by more than three sigmas from means, since they were selected by the former algorithm. One may ask if LIME brought any additional value. Since many parameters in the dataset are correlated and derived one from another, a large amount of them can strongly deviate from means, while still marking only one problem. The explanation algorithm manages to find more subtle relations. For example, in the case of sample 54, although most of **dcar error** parameters are heavily skewed, LIME pointed out that smaller deviations in parameters of type **mult error** and **offsetdZA error** stronger influence decision that this chunk is an outlier. Moreover, parameters that differ their means the most, i.e., **dcarCP0 and dcarCP1** (both by more than 13 sigmas), were not selected by LIME.

The example explored in the previous paragraph contains an interesting contradiction. Parameter **qOverPtC** in the case of both samples had a value of **1.0** and was marked as an important feature. However, in the case of chunk 0, it was used as an argument for sample normality while in the case of chunk 54 it was used to prove sample animality. Closer analysis of this ambivalent parameter in the whole dataset shows that it has very low variation. It almost always takes values close to 1.0. This and local nature of LIME probably caused this unusual behavior. Since value is almost not changing, the Random Forest classifier could be completely insensitive to its changes, which are causing LIME confusion. The solution to this issue would be ignoring this parameter in the explanation process, or perhaps applying even stronger features selection in the stage of creation of the lightweight dataset.

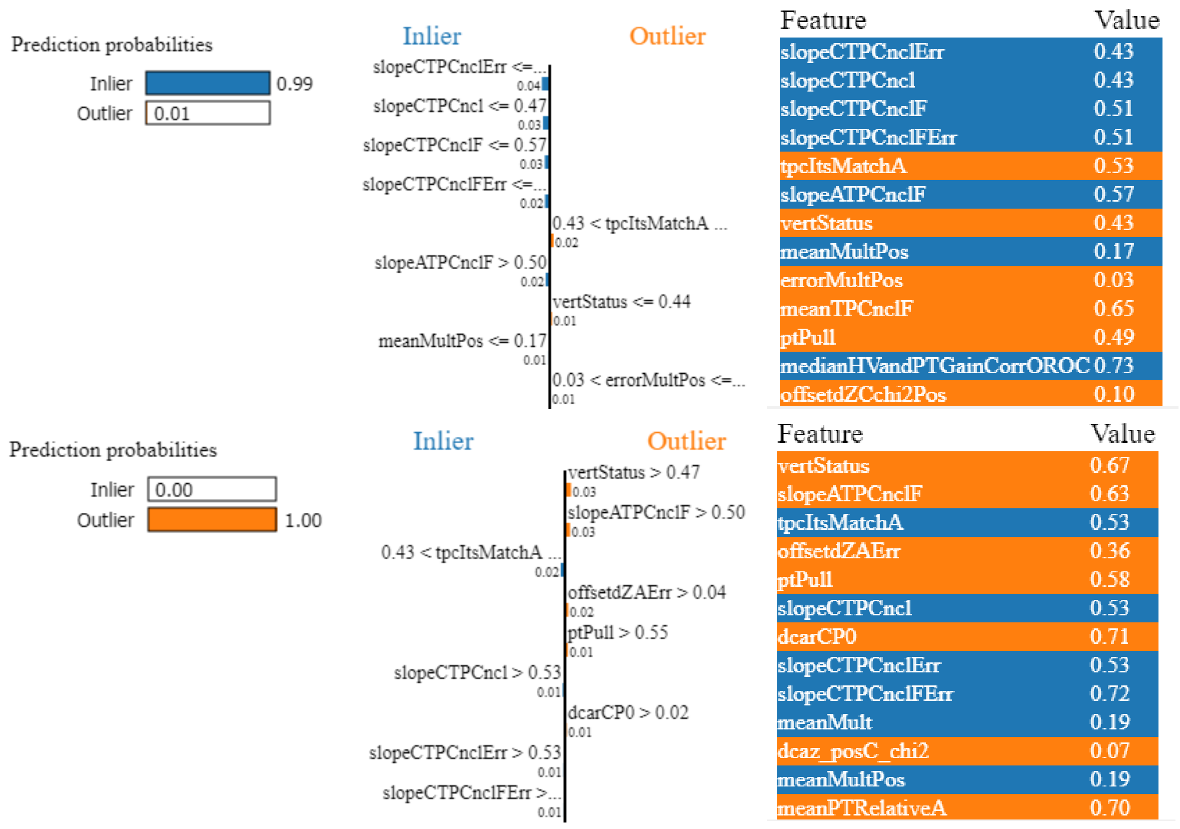
The conclusion is that the combination of Isolation Forest, Random Forest, and LIME allows for efficient, easy to implement and well-explainable anomaly detection on ALICE data. The described pipeline could be a full, classic machine learning solution to data quality tasks.

7.3. LIME with Autoencoder

Similar explanation logic can be applied to autoencoders. Adding an intermediate step allows for using any unsupervised classifier, which assigns labels to samples. To compare results with previously obtained argumentations, the trained model, described in subchapter 6.2 **Simple Autoencoder,** was used in the same work pipeline as Isolation Forest.

Once again 4% of chunks with the highest reconstruction errors were marked as anomalies. These labels were used to train Random Forest and evaluate them. In contrast to the iForest case, accuracy never reached 100%. Average accuracy from 10 training sessions on the random split, stratified dataset in 20 to 80 proportion was **99,7%,** while balanced accuracy is **96.3%**. It was more difficult for a binary classifier to reproduce AE labeling, but the obtained score is still very high. Mistakes were made only of few border case samples; therefore, the solution is still suitable.

Trained Random Forest Classifier was used with the LIME algorithm to generate explanations for the same set of samples used in the previous subchapter. Results for chunk 0 and 54 are presented on *Image 26.*



Feature	Value
slopeCTPCnclErr	0.43
slopeCTPCncl	0.43
slopeCTPCnclF	0.51
slopeCTPCnclFErr	0.51
tpcItsMatchA	0.53
slopeATPCnclF	0.57
vertStatus	0.43
meanMultPos	0.17
errorMultPos	0.03
meanTPCnclF	0.65
ptPull	0.49
medianHVandPTGainCorrOROC	0.73
offsetdZCchi2Pos	0.10

Feature	Value
vertStatus	0.67
slopeATPCnclF	0.63
tpcItsMatchA	0.53
offsetdZAErr	0.36
ptPull	0.58
slopeCTPCncl	0.53
dcarCP0	0.71
slopeCTPCnclErr	0.53
slopeCTPCnclFErr	0.72
meanMult	0.19
dcaz_posC_chi2	0.07
meanMultPos	0.19
meanPTRelativeA	0.70

Image 26. LIME explanations for Random Forest classifier trained on the lightweight dataset and labels assigned by the Autoencoder. Sample 0, inliner (above) and sample 54, outlier (below).

Explanations generated by the new workflows are more complex than the previous one. Again, both inliner and outlier cases have over 99% of probability to correct classes which shows strong confidence of classifier. Arguments used for sample **0** are clearer than those generated with Isolation Forest. The normality of the sample is supported strongly by the low values of the **slopeCTPC** group. Previously, also a few parameters from this group were selected, but these are mostly of **Error** type. Explanations generated by two methods generally agree, but Autoencoder seems to choose more coherent groups of features.

Chunk **54**, the outlier, was supported by a wider variety of parameters than previously. LIME found more features that could suggest normality of the sample and decided that **Error** type features were not as impactful as in the case of iForest. This can be interpreted as bigger sensitivity for subtle changes.

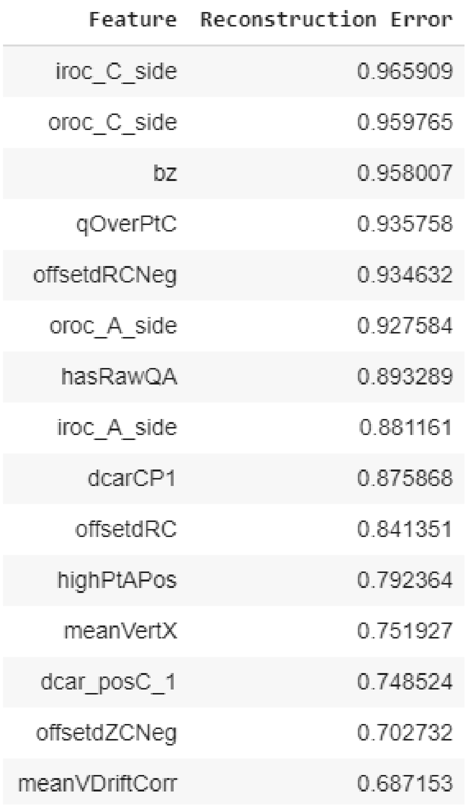
Sets of explanations differ between Isolation Forest and Autoencoder, however, both are convincing. In general, the first method tends to react stronger to big features deviations, just like a former, statistics-based algorithm. It stills manages to capture more subtle changes but argumentations for outlier cases are dominated by parameters, which are far from their means. Autoencoder based approach seems to capture more underlying relations, however, without expert data knowledge, it is not easy to decide if some, unsure, features used in explanation are not random.

7.3. Native Autoencoders' properties

In the case of autoencoders there is one more, simpler, possibility of anomaly explanation. AEs utilize reconstruction error to detect outliers, and this property can be leveraged to point out features of interest. Mean Square Error between whole input and output, was used to evaluate and score samples, and the same metric can be applied feature-wise. This way parameters can be ordered from worse to best reconstructed.

The aforementioned method fits well with the current autoencoders workflow. After selecting outlying chunks by calculating MSE of all features, a list of parameters with highest errors can be presented to user. It can be reasoned that parameters, that were not reconstructed well by network, are not normal and mostly influenced labeling of a sample. The model becomes not only classification, but also native explanation tool. It does not require using additional frameworks, like LIME. At the same time, it eliminates intermediate steps, like Random Forest in previous examples. As MSE is calculated for sample scoring, not many extra computing steps need to be performed. There is almost no overhead.

Reconstruction error of features does not bring much value in explaining the normality of samples, so most impactful parameters were selected only for chunk **54**, outlier used in previous examples and can be found on *Image 27.*



Feature	Reconstruction Error
iroc_C_side	0.965909
oroc_C_side	0.959765
bz	0.958007
qOverPtC	0.935758
offsetdRCNeg	0.934632
oroc_A_side	0.927584
hasRawQA	0.893289
iroc_A_side	0.881161
dcarCP1	0.875868
offsetdRC	0.841351
highPtAPos	0.792364
meanVertX	0.751927
dcar_posC_1	0.748524
offsetdZCNeg	0.702732
meanVDriftCorr	0.687153

Image 27. 15 features with highest reconstruction errors of example outlier (54).

It can be seen that no variables directly overlap with those selected by LIME used in combination with the autoencoder, but groups of selected features are similar. If greater subset, e.g. 35 features is considered, the overlap is stronger. Not many obvious (most skewed by statistical parameters) variables were chosen, which can be signed, that explanation grasps underlying data relations.

Using native autoencoders' properties for decision explanation seems like a promising alternative to model agnostic frameworks, but has one important drawback. As neural network is trained to only recreate input, it is not ensured in any way that the worst reconstructed features are the ones mostly influencing the decision. In fact, some features, that exhibit strong binarity and are difficult to recreate, can always be assigned one value by the network (for example an average of two binary inputs). This will cause constant, high reconstruction error to feature without any relation to the input sample. This anomaly could explain why **bz**, general feature, was assigned top third MSE, while being almost completely omitted by all LIME explanations.

Recent works explore the possibility of applying frameworks like **SHapley Additive exPlanations (SHAP)** directly on autoencoders [61]. SHAP is a game-theory based method, that proved to be efficient in explaining the decisions of many machine learning models. Moreover, it is said to be competition to LIME. Using it with deep unsupervised models promises more meaningful and complete explanations, then raw reconstruction errors.

Because of its simplicity and resource efficiency native autoencoders' properties can be used in combination with LME and Random Forest or as first, initial research step. Even if the method cannot find exact problematic parameters, it can point user to groups of interest.

8. Summary and Conclusions

The goal of the thesis was to design, implement and compare machine learning-based solutions for quality control task in CERN's ALICE experiment. Supervised, unsupervised, and semisupervised methods were considered. Code for the project, was developed using Python and state-of-art, open-source data science libraries.

In the initial three chapters reader was introduced to the problem of the project and used software tools. Short descriptions of CERN, ALICE, and LHC were provided to better explain the goals and motivation behind the thesis. For toolkits and programming language, beyond their descriptions, arguments for choosing them were outlined.

The first tackled practical problem was in-depth data analysis and feature selection, that lead to the creation of **the lightweight dataset**. Even though, it contains almost two times fewer parameters than the original, it allows for obtaining almost the same results in all described in thesis machine learning tasks. Reduction of size, also brought a major boost in performance, cutting training times of all models. When a greater number of samples are used in the real application, this improvement allows for saving computing resources.

Chapter **5. Classic Machine Learning** compared algorithms not using deep learning. Limited applications of supervised methods were outlined, followed by far more practical unsupervised and semisupervised techniques. Two algorithms stood out from the rest, namely DBSCAN and Isolation Forest. Both allow for robust outlier detection without a labeled training dataset and minimum hyperparameters tuning. In this chapter, **t-SNE visualization** and accuracy based on labels generated by the classic algorithm were proposed as preliminary evaluation methods.

Next entry regarded **Deep Autoencoders** as an unsupervised anomaly detection technique. After a brief description of this kind of neural networks, simple and variational architectures were presented. Models implemented in PyTorch framework were trained and tested on a sample dataset. Evaluation metrics were extended by using ROC and Precision-Recall curves. As deep learning methods often manage to capture difficult to spot, underlying relations in data, use of autoencoders has the potential to outperform expert users on the task.

The last chapter described model explainability, achieved primarily by using the **LIME** framework. Information on which features influenced classifier decision is crucial to building users' trust in machine learning models, as well as to extending knowledge of the experiment itself. Intermediate step allowing for using LIME with unsupervised models was discussed and tested. Example decision explanations were obtained for both, strongly inlying and outlying samples and analyzed in terms of usability. The most important parameters scored by LIME were compared to the most skewed ones as indicated by statistical parameters. The possibility of using feature-wise reconstruction error of autoencoders was also presented.

The most promising methods were collected for comparison in *Table 5*. Strong advantages of given solutions were marked in green while drawbacks in red. The table provides a general overview of researched techniques, so parameters were simplified e.g. accuracy is qualified as **high**, **low** etc. and **training time** is qualified either as **long** or **short**.

Table 5. General comparison of anomaly detection methods.

Parameter	Random Forest	DBSCAN	Isolation Forest	Autoencoder
Accuracy on former method labels	Very high	High	High	High
Can work unsupervised	No	Yes	Yes	Yes
Generalization abilities on not labeled data	-	Low	Medium	High
Amount of hyperparams. and tuning difficulty	Low	High	Low	Very high
Training Time	Short	Short	Short	Long
Can find subgroups in data	No	Yes	No	No
Can outperform expert user?	No	Possibly	Possibly	Yes
Can be used with LIME?	Yes	Requires intermediate step	Requires intermediate step	Requires intermediate step but can also leverage native proper.

Performed simulations and evaluations allow for concluding those goals of the project were realized. Proposed workflows could be applied in CERN's ALICE environment for robust and explainable anomaly detection.

For further improvements additional topics in the subject could be researched. Better results might be obtained with Variational Autoencoder by using **Reconstruction Probability** rather than a reconstruction error [56]. SHAP method used directly with autoencoders could provide more meaningful explanations [61]. Other alternatives for LIME are contextual outlier interpretation methods [62].

Bibliography

Disclaimer:
All cited web resources are referenced in their state and availability as of 01.05.2020.

[1] *A.I. is in a golden age and solving problems that were once in the realm of sci-fi;* CNBC; https://www.cnbc.com/2017/05/08/amazon-jeff-bezos-artificial-intelligence-ai-golden-age.html

[2] J. Collins, K. Howe, B. Nachman: *Anomaly Detection for Resonant New Physics with Machine Learning*; Physical Review Letters 121, 241803; 12.12.2018.

[3] C. Adam-Bourdarios et al.: *The Higgs Machine Learning Challenge*; J. Phys.: Conf. Ser. 664 072015; 2015.

[4] *Scikit-learn documentation*; https://scikit-learn.org/stable/

[5] *Replication Crisis;* Wikipedia; https://en.wikipedia.org/wiki/Replication_crisis

[7] W. Samek, T. Wiegand, and K.R. Müller: *Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models;* arXiv 1708.08296; 28.08.2017.

[8] M. T. Ribeiro, S. Singh, C. Guestrin: *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*; arXiv 1602.04938; 16.02.2016.

[9] *CERN's Home Website;* https://home.cern/

[10] M. Spira et al.: *Higgs Boson Production At THE LHC*; arXiv 9504378; 24.04.1995.

[11] *CERN's LHC website;* https://home.cern/science/accelerators/large-hadron-collider

[12] *CERN's ALICE website;* https://home.cern/science/experiments/alice

[13] The ALICE Collaboration: *The ALICE experiment at the CERN LHC*; Journal of Instrumentation Volume 3; 14.08.2008.

[14] B. Haller1, P. Lesiak and J. Otwinowski3 for the ALICE Collaboration: *Design of the data quality control system for the ALICE O2;* https://indico.cern.ch/event/505613/contributions/2227515/attachments/1342855/2026384/Poster-v4-120.pdf

[15] Image: https://www.indissoluble.com/wp-content/uploads/2018/04/ALICE1-1.jpg

[16] *Pyhton website;* https://www.python.org/about/

[17] *Python;* Wikipedia; https://en.wikipedia.org/wiki/Python_(programming_language)

[18] *TIOBE Index website;* https://www.tiobe.com/tiobe-index/

[19] *Conda;* Wikipedia; https://en.wikipedia.org/wiki/Conda_(package_manager)

[20] *Jupyter website;* https://jupyter.org/

[21] Image: https://upload.wikimedia.org/wikipedia/commons/thumb/f/f8/Python_logo_and_wordmark.svg/1280px-Python_logo_and_wordmark.svg.png

[22] *PEP8 documentation;* https://www.python.org/dev/peps/pep-0008/

[23] *Numpy website;* https://numpy.org/

[24] *Pandas website;* https://pandas.pydata.org/

[25] *Matplotlib website*; https://matplotlib.org/

[26] *Scipy website;* https://www.scipy.org/

[27] *Scikit-learn website;* https://scikit-learn.org/stable/

[28] *PyTorch website;* https://pytorch.org/

[29] *Skorch website;* https://skorch.readthedocs.io/en/stable/

[30] *Exploratory Analysis;* Wikipedia; https://en.wikipedia.org/wiki/Exploratory_data_analysis

[31] *Dimensionality Reduction;* Wikipedia; https://en.wikipedia.org/wiki/Dimensionality_reduction

[32] L. van der Maaten et al.: *Dimensionality Reduction: A Comparative Review*; TiCC TR 2009–005; 26.11.2009.

[33] S. Wold et al.: *Principal Component Analysis*; Chemometrics and Intelligent Laboratory Systems, 2 37-52; 1987.

[34] W. Wang et al.: *Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction;* CVPR2014 Workshop*;* 490 – 497; 2014

[35] K. Yan, D. Zhang: *Feature Selection and Analysis on Correlated Gas Sensor Data with Recursive Feature Elimination.*; Sensors and Actuators B: Chemical 212. 10.1016; 02.2015;

[36] *Scikit-learn, RFE documentation;* https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

[37] *Deep learning vs. classical machine learning;* Towards Data Science; https://towardsdatascience.com/deep-learning-vs-classical-machine-learning-9a42c6d48aa

[38] *Scikit-learn, Logistic Regression documentation;* https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[39] *Scikit-learn, Logistic Regression documentation*; https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[40] *Scikit-learn, SVC documentation;* https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[41] L. Maaten, G. Hinton: *'Visualizing Data using t-SNE'*; Journal of Machine Learning Research 9 2579-2605; 11.2008.

[42] *Scikit-learn, KMeans documentation;* https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[43] E. Schubert et al.: *'Why and How You Should (Still) Use DBSCAN'*; ACM Transactions on Database Systems no.19; 07.2017.

[44] M. Ankerst et al.: '*OPTICS: ordering points to identify the clustering structure'; ACM SIGMOD Record; 06.1999.*

[45] *Scikit-learn, DBSCAN documentation;* https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[46] F.T. Liu et al.: *'Isolation Forest'; Eighth IEEE International Conference on Data Mining*, pp. 413-422, 12.2008.

[47] Image: https://en.wikipedia.org/wiki/File:Isolating_a_Non-Anomalous_Point.png

[48] *Deep Learning;* Wikipedia; https://en.wikipedia.org/wiki/Deep_learning

[49] A.A. Pol et al.: *Anomaly detection using Deep Autoencoders for the assessment of the quality of the data acquired by the CMS experiment*; EPJWeb of Conferences 214 06008; 2019.

[50] J. Schmidhuber: *Deep Learning in Neural Networks: An Overview;* chapters 5,7; arXiv 1404.7828; 30.04.2014.

[51] L. Deng: *The MNIST Database of Handwritten Digit Images for Machine Learning Research*; IEEE Signal Processing Magazine Volume 29 Issue: 6; 11.2012.

[52] Image: https://miro.medium.com/max/3524/1*oUbsOnYKX5DEpMOK3pH_lg.png

[53] K. G. Dizaji et al.: *Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization*; arXiv 1704.06327; 20.04.2017.

[54] D. P. Kingma, J. Ba: *Adam: A Method for Stochastic Optimization*; arXiv 1412.6980; 22.12.2014.

[55] J. Brownlee: *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*; https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

[56] J. An, S. Cho: *Variational Autoencoder based Anomaly Detection using Reconstruction Probability*'; 2015-2 Special Lecture on IE; 27.12.2015.

[57] Image: https://miro.medium.com/max/1314/1*CiVcrrPmpcB1YGMkTF7hzA.png

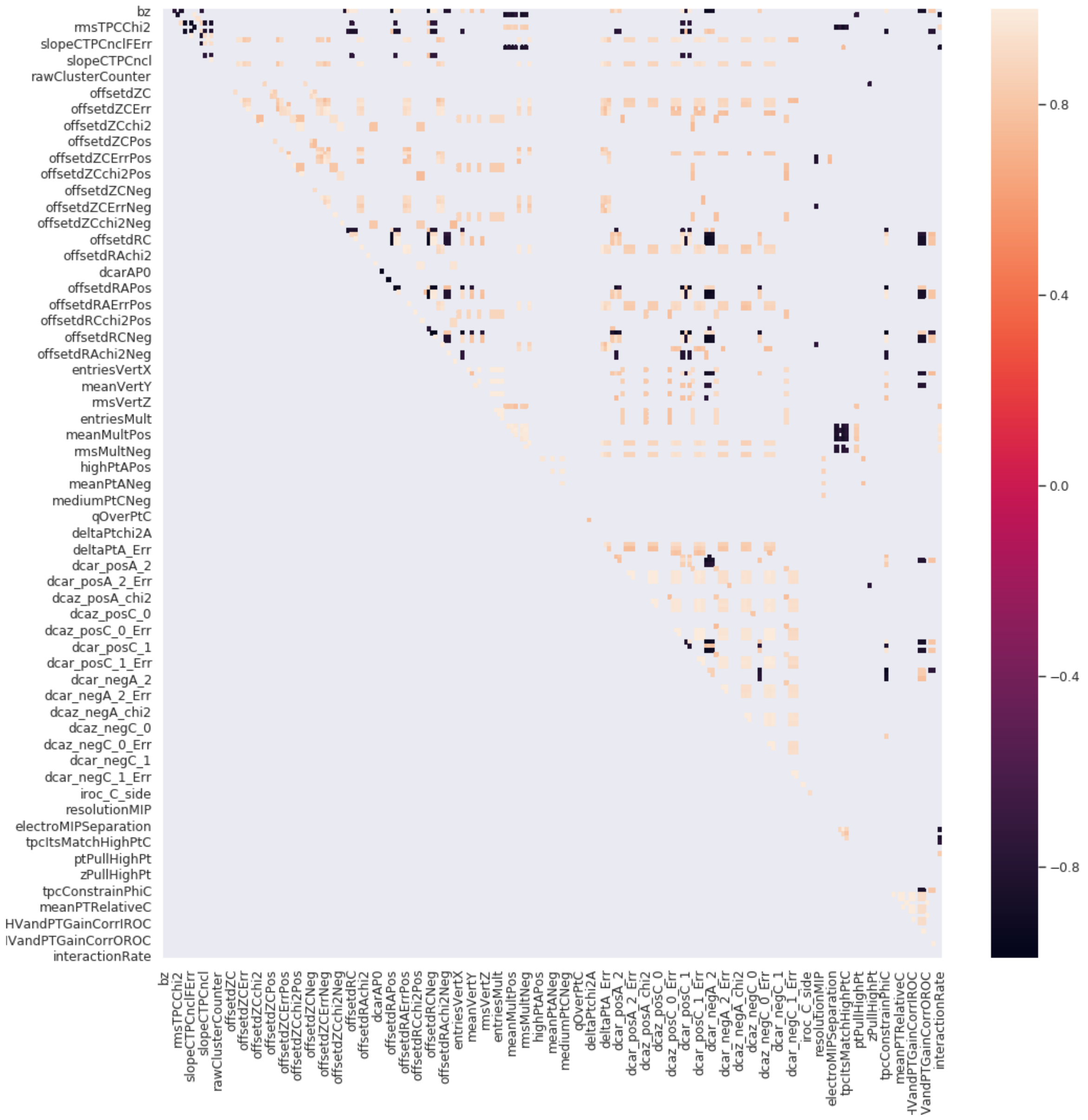
[58] *Kullback Leibler Divergence;* Wikipedia; https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

[59] G. Gundersen: *The Reparameterization Trick*;
https://gregorygundersen.com/blog/2018/04/29/reparameterization/

[60] M. Yin et al.: *Understanding the Effect of Accuracy on Trust in Machine Learning Models*; Conference on Human Factors in Computing Systems Proceedings CHI 2019; 09.05.2019.

[61] N. Liu et al.: *Contextual Outlier Interpretation*; arXiv 1711.10589; 04.05.2018.

Appendix A


bz
rmsTPCChi2
slopeCTPCnclFErr
slopeCTPCncl
rawClusterCounter
offsetdZC
offsetdZCErr
offsetdZCchi2
offsetdZCPos
offsetdZCErrPos
offsetdZCchi2Pos
offsetdZCNeg
offsetdZCErrNeg
offsetdZCchi2Neg
offsetdRC
offsetdRAchi2
dcarAP0
offsetdRAPos
offsetdRAErrPos
offsetdRCchi2Pos
offsetdRCNeg
offsetdRAchi2Neg
entriesVertX
meanVertY
rmsVertZ
entriesMult
meanMultPos
rmsMultNeg
highPtAPos
meanPtANeg
mediumPtCNeg
qOverPtC
deltaPtchi2A
deltaPtA_Err
dcar_posA_2
dcar_posA_2_Err
dcaz_posA_chi2
dcaz_posC_0
dcaz_posC_0_Err
dcar_posC_1
dcar_posC_1_Err
dcar_negA_2
dcar_negA_2_Err
dcaz_negA_chi2
dcaz_negC_0
dcaz_negC_0_Err
dcar_negC_1
dcar_negC_1_Err
iroc_C_side
resolutionMIP
electroMIPSeparation
tpcItsMatchHighPtC
ptPullHighPt
zPullHighPt
tpcConstrainPhiC
meanPTRelativeC
HVandPTGainCorrIROC
IVandPTGainCorrOROC
interactionRate
0.8
0.4
0.0
−0.4
−0.8