Engineering Applications of Artificial Intelligence 122 (2023) 106079

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Photon/electron classification in liquid argon detectors by means of Soft Computing

Javier León [a,*], Juan José Escobar [b], Marina Bravo [c], Bruno Zamorano [c], Alberto Guillén [a]

[a] *Department of Computer Engineering, Automatics and Robotics, CITIC, University of Granada, Granada, Spain*
[b] *Department of Software Engineering, CITIC, University of Granada, Granada, Spain*
[c] *Department of Theoretical Physics and Cosmology, University of Granada, Granada, Spain*

A R T I C L E   I N F O

Keywords:
Particle Physics
Photon
Electron
Machine learning
LAr time-projection chamber

A B S T R A C T

In the field of Particle Physics, the behaviors of elementary particles differ among themselves on subtle details that need to be identified to further our understanding of the universe. Machine learning is being increasingly applied in order to solve this task by extracting and extrapolating patterns from detector data. This paper tackles the classification of simulated traces from a liquid argon container into photon- or electron-induced events. Several viable dataset representations are proposed and evaluated on nine supervised learning algorithms to find promising combinations. After that, a hyperparameter optimization step is applied on some of the classifiers to try to maximize their accuracy. Random Forest and XGBoost achieve the best results with roughly 88% test-set accuracy, which shows the potential of machine learning to solve a significant research question in a subfield that is expected to keep growing in the coming years.

1. Introduction

Neutrinos are one of the hot topics in Physics nowadays due to the number and significance of the questions still unanswered about them, to the point that research on their properties earned T. Kajita and A. McDonald the 2015 Nobel Prize in Physics. The current neutrino program in the United States includes the analysis of neutrino oscillations at both short (SBN, Short-Baseline Neutrino program) (Machado et al., 2019) and long baseline (DUNE, Deep-Underground Neutrino Experiment) (Abi et al., 2020). Despite their different baseline and design, both programs share a common detection technology based on Liquid Argon Time-Projection Chambers (LArTPCs) (Rubbia, 1977).

The working principle of LArTPCs consists in the application of a uniform electric field to a large volume of ultra-pure liquid argon. When charged particles traverse the argon, they produce free electrons that are drifted by the electric field and then collected by a network of wires. In addition to this ionization, the passage of charged particles also produces an excitation of the atoms. The de-excitation of liquid argon generates ultraviolet light that travels much faster than charge through the argon. This light can be used to precisely determine the starting time of neutrino interactions. Thanks to the precise measurement of time provided by the light detection system, and combining it with the 2D footprint that the ionization produces in the wires, physical events can be reconstructed in three dimensions.

The aim of this paper is to propose a first approach in trying to classify the particle that has generated the interactions inside the LArTPC. In particular, we will focus on the problem of distinguishing the signal left by electrons and photons. When either of these particles interacts with the detector, it produces an electromagnetic shower. However, depending on what the primary particle is, there are some distinguishing features: if the primary particle is an electron, after traveling some distance in the argon it will emit a high-energy photon that will subsequently generate an electron–positron pair.[1] These particles can repeat the process for some generations, until the available energy for secondary particles is below a threshold (called critical energy) and the cascading process stops. If, instead, the process is started by a photon, it will not produce any signal in the detector, and the generated electron–positron pair will be the first visible signal coming from the shower. Therefore, the very beginning of the electromagnetic shower will consist in a single track-like deposition consistent with that of an electron (whenever the primary particle is an electron), or a double signal produced by the electron–positron pair (if the primary is a photon). This technique has been used in traditional analyses (Antonello et al., 2013) to distinguish both primaries, with an efficiency of about 90% and a photon contamination of a few percent for the composition of the neutrino beam.

* Corresponding author.
E-mail address: jleon@ugr.es (J. León).

[1] The positron is the antiparticle of the electron. It has an electric charge of $+1e$ and the same mass as an electron. When a positron collides with an electron they annihilate generating photons.

https://doi.org/10.1016/j.engappai.2023.106079
Received 30 June 2022; Received in revised form 4 February 2023; Accepted 28 February 2023
Available online xxxx

In this work, we will address the problem of distinguishing the signal produced by electrons and photons using machine learning. The main contributions of this paper are:

- We provide successful classification results for photon/electron from raw data considering different energy values. To do so we define several viable representations of the original data in a format that is appropriate for machine learning models. This task is already a serious challenge considering that the number of recorded bins greatly depends on the energy of the particle. Nevertheless, to be applicable in real scenarios, models should be accurate for a wide range of energy values, which leads to a large variety of potential representations.
- The comparison of the resulting datasets with nine types of classifiers, including traditional ones, bagging and boosting approaches, and convolutional neural networks.
- The optimization of the hyperparameters of the most promising classifiers using a Tabu Search procedure to avoid the setting of magic numbers in the definition of the models.
- An energy–time assessment that highlights the importance of efficient computing and provides a baseline for future related comparisons. To our knowledge, reporting this type of information is uncommon in the literature, but it adds nuance to the discussion, especially in terms of environmental and cost-saving concerns.

The rest of paper is organized as follows: Section 1.1 analyzes previous work in the field of applied machine learning and soft computing to Particle Physics, Section 2 will describe the data and formulate the problem. Section 3 depicts the methodology followed, focusing on how to represent the information generated from simulations, the feature engineering process and a brief description of the classification paradigms applied. Section 4 presents the results and discussion and, finally, Section 5 draws conclusions.

1.1. State-of-the-art

The fields of machine learning and Physics have become progressively intertwined in the last years, with one inspiring or directly producing advances in the other (Carleo et al., 2019). From this topic we are particularly interested in the application of machine learning techniques to Particle Physics problems (Radovic et al., 2018). There are two main areas where these techniques may be applied: generation or refinement of simulated data (Paganini et al., 2018; Erdmann et al., 2018), event reconstruction (Huennefeld, 2017), and classification. The latter can be broken down into many tasks: photon–hadron (Carrillo-Perez et al., 2021; Assunção et al., 2019), or photon–electron (the problem tackled in this paper), among others.

The classification task can be approached from many different angles depending on the characteristics of the data and the models employed. In Guillén et al. (2019), the signal of the entire event is summarized through integration, while in Carrillo-Perez et al. (2021) the signal is processed by the classifier as a time series. Class imbalance can also occur when the focus is on detecting signals from background processes, as particle activity is much less common; XGBoost was used in Cornell et al. (2022) in combination with F-score and other metrics to overcome the drawbacks of evaluating classification performance via accuracy in this context.

With regard to the efficiency in the use of computational resources, it is important to consider not only the ability of a machine learning model to produce accurate predictions, but also the associated costs. For instance, in Guillén et al. (2020) a comparison of several machine learning paradigms is made in terms of memory requirements, showing substantial variations among them. In addition to memory, running time and also energy consumption are often overlooked in the literature. While the search for high accuracy rates is a demanding task in itself, some effort is due in this aspect, since in the real world not all computing systems have enough resources to spare.

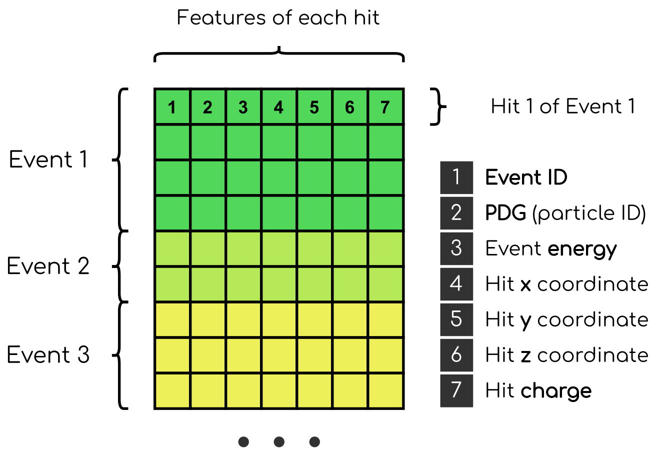

Fig. 1. Format of the original dataset.

2. Problem and data description

The data used throughout this paper was obtained using the specialized Physics simulation software GEANT4 (Agostinelli et al., 2003; Allison et al., 2006, 2016), which simulates the passage of particles through matter and is the standard tool in Nuclear and Particle Physics. In particular, we have simulated an argon volume of 5 m in length, 4 m in height and 4 m in width, consistent with the dimensions of the active volume of the SBN near detector (SBND, Short-Baseline Near Detector) (Machado et al., 2019). In order to emulate the actual reconstruction carried out by LArTPC detectors, the ionization simulated by GEANT4 is smeared using a resolution of 3 mm (consistent with the spacing between wires in SBND). In addition, when the smeared spatial energy deposition is above a fixed threshold, we define it as a region of interest called *hit*, resembling the usual jargon used in LArTPC reconstruction.

In summary, each instance of a particle (namely, either an electron or a photon) entering the LArTPC is called an *event* and it consists of multiple hits, which are defined by their time and space coordinates and the deposited energy. The original dataset structure is made up of one row per hit, where each hit has the following 7 features:

- Event ID: the identifier of the event which contains the hit. It is common to all hits of the same event.
- PDG code: an integer that unequivocally identifies the primary particle, as standardized by the Particle Data Group (Zyla et al., 2020). In this case, it represents either an electron (PDG code = 11) or a photon (PDG code = 22).
- Event energy: energy of the event that contains the hit (expressed in MeV). Thus, the value is shared by all the hits in the same event. This feature cannot be used in the problem-solving stage, as it would not be known for real-world measurements outside the simulations.
- x: horizontal coordinate of the hit in the detector with range: $[-200, 200]$ cm.
- y: vertical coordinate of the hit in the detector with range: $[-200, 200]$ cm.
- z: depth coordinate of the hit in the detector with range: $[0, 500]$ cm.
- Hit charge: the density of energy deposited in the spatial surroundings of the hit, measured in arbitrary units.

The data arrangement described above can be visualized in Fig. 1. As can be seen, hits from the same event are stored consecutively; their arrangement within the event is chronological.

Tables 1 and 2 show brief statistical descriptions of the original data for the variables that will be used later.

Table 1
Statistical summary of variables for the Electron class.

Electron	Min.	Max.	Avg.	Median	SD
Energy	0.05	0.30	0.20	0.21	0.07
Charge	1.26	32,900	794.14	538.32	893.69
x	−115.76	199.13	102.10	102.48	19.16
y	−188.78	197.64	0.09	0.01	19.93
z	46.93	458.73	249.87	249.93	24.05

Table 2
Statistical summary of variables for the Photon class.

Photon	Min.	Max.	Avg.	Median	SD
Energy	0.05	0.30	0.20	0.22	0.07
Charge	1.31	34,032	845.75	578.99	957.82
x	−127.09	199.11	101.61	102.14	27.67
y	−193.63	192.79	0.06	0.18	29.18
z	29.83	499.63	250.78	250.53	35.75

The original dataset consists of 19,900 events: 10,000 correspond to photon particles and 9900 to electron particles. No missing values are present, and therefore no removal or imputation is necessary.

From a research standpoint, the problem to be solved here is the separation of events into two classes: electron events and photon events. As their behaviors inside the detector are extremely similar to the human eye, machine learning is a highly promising approach to extract the subtle but useful patterns that differentiate the two classes.

The separation into classes can be understood both as a classification and a regression problem. In the present paper we have chosen the former, as it is the most straightforward given the data.

3. Methodology

This section is devoted to four main aspects of the experimentation pipeline: the preprocessing of the original dataset, the machine learning paradigms initially considered for the classification task, the further hyperparameter optimization performed on the most promising paradigms, and the metrics used to evaluate the results throughout the manuscript.

3.1. Data preprocessing and problem encoding

3.1.1. Preprocessing

The data in its original form presents two main issues that need to be addressed before its use in machine learning:

- The rows contain individual hits instead of events. Since each event is conceptually one instance of the final dataset, an aggregation of hits into events is necessary.
- There are two files that contain hit information: one for the Electron class, and one for the Photon class. In both files the event IDs start at 0, so merging as-is would cause clashes.

These issues can be solved as follows: firstly, the hits are aggregated into events independently in each of the two files; secondly, the two resulting partial datasets are merged by (i) offsetting the event IDs of one of them by the size of the other and concatenating them; (ii) adding the corresponding class labels to each row of the final dataset by looking at the PDG (particle ID) feature: class 0 for photons and class 1 for electrons.

The first step (aggregation of hits into events) requires filtering the information of every hit row. As the event ID, event energy, and PDG are properties of the whole event, only the coordinates and the charge are added to each event row on a per-hit basis. The event energy is appended at the end of every row.

The operations described above leave the dataset in a standard format of `instances` × `features`. In addition to them, two more steps are performed to prepare the data for use in machine learning: the creation of the training, validation, and testing splits; and data normalization. The training split contains 70% of the instances, whereas the validation and testing ones contain 10% and 20%, respectively. The coefficients for a z-score normalization are calculated from the training instances and applied to all three splits (i.e., training, validation, and testing).

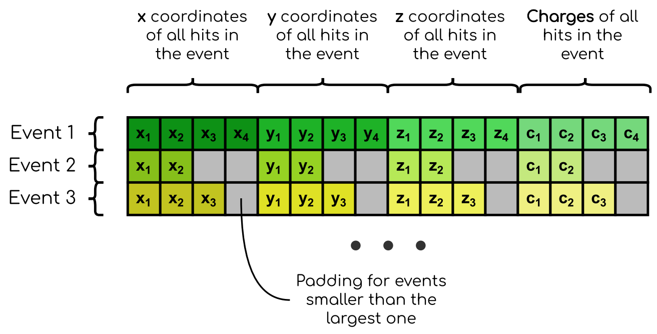
x coordinates of all hits in the event
y coordinates of all hits in the event
z coordinates of all hits in the event
Charges of all hits in the event
Event 1
Event 2
Event 3
x1 x2 x3 x4 y1 y2 y3 y4 z1 z2 z3 z4 c1 c2 c3 c4
x1 x2 y1 y2 z1 z2 c1 c2
x1 x2 x3 y1 y2 y3 z1 z2 z3 c1 c2 c3
Padding for events smaller than the largest one

Fig. 2. Dataset A: concatenation of hit features grouped by event.

3.1.2. Problem encoding

The contents of the dataset obtained at the end of Section 3.1.1 can be arranged or extended in many ways. In this section we discuss some of them.

Before that, let us have a brief recap of the information that is kept for each hit: its x, y, and z coordinates, as well as its charge. As stated before, the energy, apart from being an attribute of the events, cannot be used to solve the problem since a real-world detector would only yield a reconstructed estimate of this observable.

The first and simplest of the problem encodings shown here is to keep that format mostly unchanged. As can be observed in Fig. 2, the four features of a hit are grouped by type instead of by the hit they correspond to.

One detail to consider is the fact that events vary in length, i.e., number of hits. In this encoding, the solution is to add padding at the end of events smaller than the largest one. More precisely, the padding is added after every block of values of the same feature. We will call this generic encoding *Dataset A*.

Another relevant aspect of the dataset is the explicit ordering of the hits inside each event. This is not a trivial issue: apart from the ordering being a source of information in itself, hits that are given lower priority may be left out of the dataset if a hit cap is enforced.

By default, the hits are chronologically sorted; let us call this *Dataset* A_{chrono}. In order to explore the importance of explicit ordering for the hits of an event, two alternate versions called *Dataset* A_{charge} and *Dataset* A_{random} have also been created. Both have the same dimensionality but the hits of each event have been rearranged using different order criteria.

The hits inside each event of Dataset $\mathrm{A}_{\mathrm{random}}$ have been randomly sorted. As there is no discernible pattern behind it, this serves as a baseline against which the rest of ordering criteria are tested.

Dataset $\mathrm{A}_{\mathrm{charge}}$ (Fig. 3) consists of events whose hits have been sorted by charge in descending order. There are admittedly three other immediate ways to sort the hits in an event using the current attributes: by x, y, or z coordinates. Nonetheless, it is not clear from the semantics of such criterion how it would improve the dataset: it should not matter where in the detector a given particle happens to land.

One of the experiments in this paper is devoted to the assessment of the differences between these three dataset variants.

That said, the instances of Dataset A have some drawbacks. The main one is the way the width of the dataset is determined: at this point, it is dictated by the size of the largest event in the dataset. For

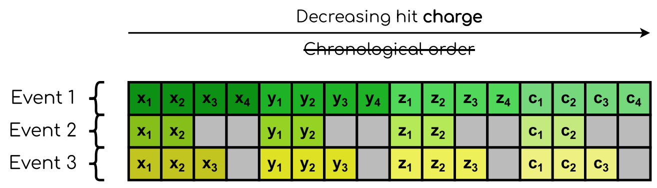

Fig. 3. Dataset A_{charge}: concatenation of hit features grouped by event and sorted by decreasing hit charge.

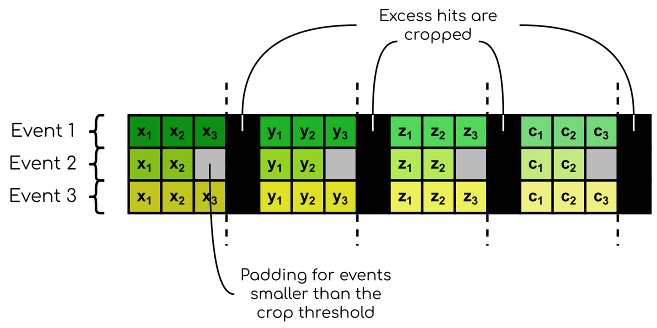

Fig. 4. Dataset B: concatenation of hit features grouped by event and cropped at a certain threshold.

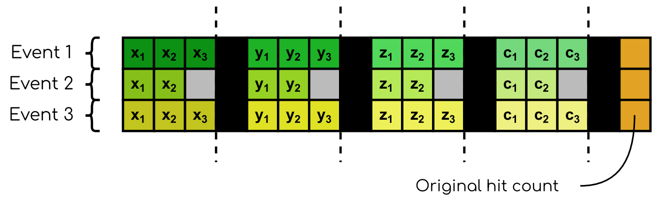

Fig. 5. Dataset C: concatenation of hit features grouped by event, cropped at a certain threshold, and with the original number of hits as the last feature.

events smaller than that, padding solves the issue; however, for new, larger events there is no choice but to crop them at this arbitrary length. In addition, depending on the size of the largest event, concerns about computational costs can arise. Therefore, finding an alternate way to set a per-event hit cap is highly desirable. The particular method employed here will be discussed in detail in Section 4.

Regardless of how the limit is set, let *Dataset B* be a modified version of Dataset A where a pre-determined hit cap has been set that does not necessarily rely on the size of the largest event. Dataset B is depicted in Fig. 4.

A potential side effect of limiting the amount of hits per event is the loss of relevant information. Any variant of Dataset A is susceptible to it when transformed into Dataset B since, a priori, it is unknown whether important features are being left out in favor of uninformative ones. The enforcement of a per-event hit limit rests upon a suitable ordering of the hits. This is why exploring alternatives in this regard is beneficial.

Finally, since adding back the hits is not an option, some kind of feature engineering could prove helpful to mitigate information loss. One of the most straightforward examples is *Dataset C* (Fig. 5), where the original hit count has been included as a feature at the end of each row. This transformation can be applied to any instance of Dataset B, but the precise ones for this paper are decided empirically in Section 4.

These five datasets (Datasets A_{chrono}, A_{charge}, A_{random}, B, and C) will be used throughout the experimentation both to extract insights about the data and to solve the classification problem.

3.2. Classification paradigms

Initially, up to nine different classification algorithms (paradigms) will be tested on the datasets, of which the most promising according to the experiment results will undergo a hyperparameter optimization phase: the linear Support Vector Machine (SVM) (Cortes and Vapnik, 1995); Logistic Regression (LR) (Cramer, 2002); Multi-Layer Perceptron (MLP) (LeCun et al., 2015); AdaBoost (Freund et al., 1996; Freund and Schapire, 1997); Gradient Boosting (Friedman, 2001, 2002); XG-Boost (Chen and Guestrin, 2016); Bagging (Breiman, 1996); Random Forest (Breiman, 2001); and CNN (Guo et al., 2016).

3.3. Hyperparameter optimization

Depending on the outcomes of previous stages of the experimentation, some of the techniques described above will undergo hyperparameter optimization. The hyperparameters of a learning algorithm are those parameters that need to be set before training begins, i.e., that are not learned but instead have influence on the learning process.

Hyperparameter optimization is far from trivial, since at its core it is a combinatorial optimization problem. A given classifier can have many such hyperparameters with discrete or continuous value ranges, which often results in intractably many potential solutions. As brute-force exact approaches are out of the picture, a number of major optimization paradigms have been proposed over the last decades: Evolutionary Algorithms (Eiben et al., 2003), Particle Swarm Optimization (Poli et al., 2007), Simulated Annealing (Kirkpatrick et al., 1983), Ant Colony Optimization (Dorigo et al., 2006), or Tabu Search (Glover and Laguna, 1998), among others. These algorithms and their subsequent variants have achieved state-of-the-art results in many applications including hyperparameter optimization. We have chosen the last one, Tabu Search, for the task.

Tabu Search is a widely-used metaheuristic for many applications, be it as a standalone method or in combination with other ones (e.g., with Genetic Algorithms): a search for the term in Scopus returns over two thousand matches in titles, abstracts, and keywords for the last five years, which proves the relevance of this technique in the literature to this day. Although other alternatives like Evolutionary Algorithms boast great popularity in recent times, no method is necessarily the best for all problems (Shalev-Shwartz and Ben-David, 2014), and empirical evidence is needed to inform the conclusions.

Tabu Search sequentially explores the solution space by moving from one point in space (solution) to another according to a guiding criterion. This criterion is applied on a neighborhood of closely-related solutions to select the most appropriate from among them. To encourage novel exploration paths, recently visited solutions are penalized with an intensity chosen by the user. Recent solutions are stored in a *tabu list* either in full or split into parts depending on the desired granularity.

Notice that Tabu Search appears remarkably similar to a local search. However, it was designed with the ability to escape local optima in mind, and it achieves that purpose through several mechanisms: (i) already explored regions are explicitly penalized via the tabu list; (ii) when deciding the next move, a solution can be chosen even if it is worse than the current one; and (iii) the search can be restarted from another point in the solution space.

Going into more detail, its main elements are the following (depicted in Fig. 6):

- Initialization: creates a valid solution that will serve as the starting point of the search procedure. As the procedure usually enforces limits for the values of the attributes, ‘valid’ means correctly formatted and within the range of allowed values.
- Reinitialization: creates a valid solution that will serve as a new starting point of the search procedure. The main goal of reinitialization is to escape from local optima.
- Neighborhood generation: creates a pool of variations (the neighborhood) of a given solution from which the next solution will be chosen. Note that, although the notion of neighbor may imply spatial closeness, this relationship can follow any pattern the designer deems appropriate for the problem.

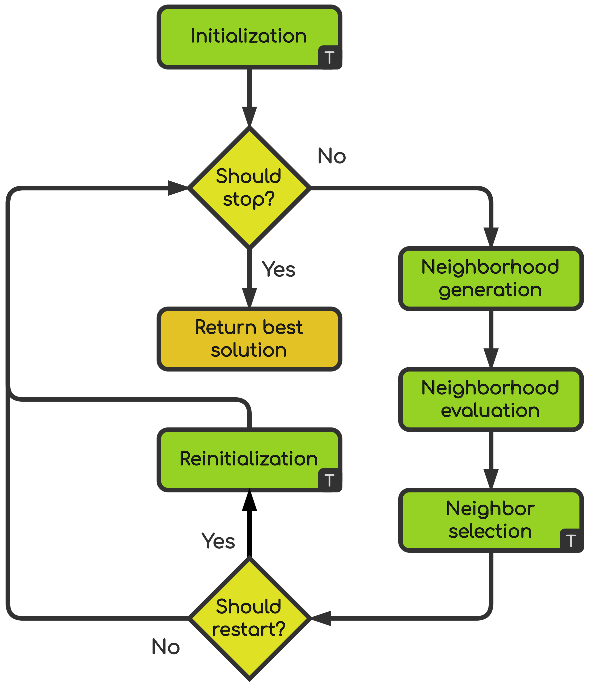

Fig. 6. Overview of the tabu search algorithm. A small T in the lower right corner of an element indicates that its output may be included in the tabu list.

- Neighborhood evaluation: evaluates the neighborhood using a predefined fitness metric.
- Neighbor selection: once the neighborhood has been evaluated, chooses the next solution based primarily on fitness. Several variations exist depending on the penalty attached to a solution included in the tabu list.
- Stopping criteria: check whether the search procedure should finish.
- Reinitialization criteria: check whether the current search path should be abandoned and a new one started.

Regarding the computational complexity of the algorithm, it is difficult to calculate for the general case because some of its elements can belong to a wide range of complexity classes depending on the problem. Most notably, the fitness function is usually a bottleneck in metaheuristics applied to machine learning due to the cost of model training. This is the case here, as Tabu Search is used to optimize hyperparameters and the evaluation is carried out after training the corresponding model.

Assuming limits m and n in the number of iterations and the number of neighbors generated at each iteration, respectively, there would be $m \times n$ evaluations of the fitness function, giving $O(m \times n \times O_{fitness})$. Since the fitness function is the element most directly affected by the size of the problem (i.e., the size of the dataset), the choice of classifier is of utmost importance in this respect.

The configuration of the Tabu Search in the context of hyperparameter tuning will be described in Section 4.1.2.

3.4. Performance metrics

Several metrics are employed in this paper to assess the strengths and weaknesses of machine learning models. While some of them are straightforward, a brief description is provided below for each one. Remember that the positive class or 1 corresponds to electrons, and the negative class or 0 corresponds to photons.

- Accuracy: the rate of correctly classified entries without focus on any particular class.
- Recall: $\frac{TP}{TP+FN}$. It is the rate of positives (electrons) discovered with respect to the total amount of positives in the sample.
- Precision or Positive Predictive Value (PPV): $\frac{TP}{TP+FP}$. It tells the purity in the classification of positives, which penalizes negatives (photons) wrongly labeled as positives (electrons).
- Specificity: $\frac{TN}{TN+FP}$. The equivalent of recall for the negative class (photons).
- Negative Predictive Value (NPV): $\frac{TN}{TN+FN}$. The equivalent of precision for the negative class (photons).
- Execution time: the running time of a given experiment, not including data preprocessing/loading nor subsequent operations on the results.
- Energy consumption: the energy consumption of a given experiment, again not including data preprocessing/loading nor subsequent operations on the results.

4. Experiments and discussion

4.1. Experimental setup

4.1.1. Code and computing resources

All the code is written in *Python 3.6.8* and is available on GitHub.[2] Some state-of-the-art machine learning and data science libraries have been used as well: *Scikit-Learn 0.24.2* (Pedregosa et al., 2011), *NumPy 1.19.5* (Harris et al., 2020), *TensorFlow 2.6.2* (Abadi et al., 2015), and *Keras 2.6.0* (Chollet et al., 2015). The paired permutation test (Pitman, 1937) is part of Mlxtend (Raschka, 2018). Bayesian tests are implemented in the R library `rNPBST` (Carrasco et al., 2017).

The experiments have been run in homogeneous cluster nodes with the following specifications:

- 2x Intel Xeon Silver 4214 @ 2.2 GHz: 24 threads/12 cores and a Thermal Design Power (TDP) of 85W.
- 64 GB of DDR4 RAM.
- NVIDIA Quadro RTX 6000: 1.77 GHz, 4,608 CUDA cores, 24 GB of GDDR6 RAM, and a TDP of 295W.

Except for the experiments with Tabu Search, every machine learning model is run with the default hyperparameters specified by the library that implements it, as they are assumed to be a fair starting point. However, Keras does not have defaults for all hyperparameters of CNN, so a single convolutional layer with 300 filters and a filter size of 3 was chosen as a reasonable default.

4.1.2. Hyperparameter optimization

As the final step of the experimentation, two classifiers (Random Forest and XGBoost) are selected from the nine candidates for a final hyperparameter tuning procedure. For Random Forest, the hyperparameters are the following (Anon, 2022a):

- `n_estimators`: number of trees in the forest.
- `max_features`: size of the subset of features considered when splitting a tree node.
- `min_samples_leaf`: minimum amount of samples in each branch of a node to consider it a leaf node.
- `min_samples_split`: minimum amount of training samples needed to split a node.

For XGBoost the hyperparameters are (Anon, 2022b):

- `n_estimators`: amount of gradient-boosted trees.
- `learning_rate`: shrinkage factor for feature weights used to prevent overfitting.
- `subsample`: ratio of subsampling for the training instances.
- `max_depth`: maximum tree depth.

[2] https://github.com/aguillenATC/photon-electr-clasif-LArTPC

Table 3
Classification accuracy (validation, 10 runs) ± standard deviation of each model in Datasets A_{chrono}, A_{random}, and A_{charge}.

Dataset variant	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
Dataset A_{chrono}	0.7887	0.8592 ± 0.0036	0.77 ± 0.0151	0.8418	0.5002 ± 0.0083	0.503	0.5068 ± 0.0102	0.8897 ± 0.0016	0.8817
Dataset A_{random}	0.7634	0.8487 ± 0.006	0.7468 ± 0.045	0.8509	0.5044 ± 0.0092	0.5061	0.5066 ± 0.007	0.8603 ± 0.0021	0.8771
Dataset A_{charge}	0.7639	0.8235 ± 0.0052	0.7539 ± 0.0537	0.8392	0.5008 ± 0.0044	0.5076	0.509 ± 0.0058	0.8552 ± 0.0026	0.8827

The optimization is done through 200 Tabu Search iterations, where a solution consists of each hyperparameter (attribute) mapped to a value. Initialization and reinitialization are both random. The reinitialization criterion is triggered when fitness has stagnated after a certain number of iterations. The neighbors of a given solution are found by adding and subtracting from the value of a numerical attribute, or by swapping for another possibility when an attribute is categorical. The next solution is chosen from non-tabu neighbors, except if the best tabu neighbor has higher fitness than the best solution found so far. Tabu inclusion is on a per-attribute (versus per-solution) basis, and each attribute has its own tabu tenure. The fitness of a solution is computed as the classification performance on the validation set.

4.2. Results

The experiments carried out seek to answer a number of research questions while helping to incrementally build a solution to the classification problem. The first two experiments focus on the assessment of some characteristics of the original data, as well as the evaluation of different problem encodings; the next two deal with the selection and optimization of a machine learning algorithm that maximizes classification accuracy on the chosen problem encoding; the penultimate one introduces an alternate optimization stage based on the context of the application; the last one explores the computational cost of finding such a machine learning model in terms of time spent and energy consumed.

4.2.1. Analysis of hit order

The original dataset contained hits sorted chronologically in the context of each event. The reason behind this arrangement is solely because of the output format of the simulations. Therefore, a good question is to what extent this particular ordering is relevant for the classification task. This experiment is aimed at examining how **Datasets A_{chrono}** and **A_{charge}** compare against **Dataset A_{random}**, which acts as a control group. The learning algorithms described previously have been trained and evaluated 10 times on each dataset in order to obtain an estimate of the gaps in classification accuracy. Table 3 contains the results.

A trend in favor of Dataset A_{chrono} can be readily spotted: AdaBoost, Bagging, CNN, MLP, Random Forest and XGBoost score higher when trained and tested with it instead of Dataset A_{random}. On the other hand, the opposite occurs for Gradient Boosting, Linear SVM and Logistic Regression; however, note that in this case the advantage is much smaller and that three of those classifiers are just above a coin toss in accuracy. The performances on Datasets A_{charge} and A_{random} are very similar, and it could even be said that Dataset A_{charge} looks inferior. Admittedly, though, the performance of Dataset A_{random} may be partly due to mere chance in this particular random arrangement of the hits (only one hit randomization is carried out).

Hypothesis testing can help shed more light on this issue, although it should not be the only or even the main criterion. Let the null hypothesis be that the results for the two compared datasets have been sampled from the same distribution. Given the small sample size, as well as the non-normality of the data distribution, a paired permutation test is more appropriate. For each classifier, its accuracy in one dataset is matched (paired) with its accuracy in the other, as they are dependent samples. Two tests have been carried out to compare Dataset A_{chrono} and Dataset A_{charge} to Dataset A_{random}.

The first test outputs a p-value of 0.078. For a 95% significance level, this means that the equivalence of Datasets A_{chrono} and A_{random} cannot be ruled out. A reasonable take in light of the results and the p-value would be that the chronological order appears to have an inconclusive advantage over a random one. Perhaps the classifiers are still able to derive useful bits of information implicit in the chronology no matter how the hits are arranged in the working dataset; or, perhaps, chronological order is relevant but only secondary.

The output of the second test is a p-value of 0.5703, which again means that we cannot reject the hypothesis that Datasets A_{charge} and A_{random} are equivalent. It is worth noting, though, that the gap between Datasets A_{charge} and A_{random} is much smaller than the gap between Dataset A_{chrono} and A_{random}.

Dataset A_{chrono} appears to be the obvious choice for two main reasons. Firstly, assuming the equivalence of the three datasets, it is the only one that does not require further preprocessing in order to obtain the desired hit order: the original data is already structured in that way. Secondly, if we instead acknowledge the limitations of the test for such a small sample size, the accuracy results suggest that Dataset A_{chrono} has an advantage. Therefore, it will be the starting point of the next experiment.

4.2.2. Determining a suitable number of hits

As discussed in Section 3.1.2, the choice of how many hits are included in the dataset is not trivial: every additional hit brings more information to the table (i.e., more features for the dataset), but more does not always equal better: the utility of the new information and the added computational complexity have to be taken into account.

We have evaluated the accuracy of the classifiers in several variants of **Dataset B**. Remember that each variant consists in a cropped version of Dataset A_{chrono} according to a different number of hits per event. Fig. 7 shows the results. As can be observed, there are two main groups that follow opposite trends. The first one is composed of bagging and boosting algorithms and CNN, whose performance goes up as the available hits increase. Conversely, the second group appears to suffer from heavy overfitting; this is especially noticeable for MLP, which starts off with a similar degree of accuracy as the first group but quickly loses out. This phenomenon is common, as not all hits in an event are necessarily informative for the classification problem and could even introduce noise. The exact results are shown in Table 4. Notice there that the standard deviations are small, meaning that the numbers are stable and telling of what to expect in the average run.

Further elaborating on the performance of the algorithms, there are some interesting details worth mentioning if we take a look at Tables A.10, A.11, A.12, and A.13 (included in the Appendix). SVM is consistently bad in all metrics on average, but especially at detecting all instances of the negative class (photons); however, it is somewhat better at detecting all instances of the positive class (electrons). Logistic Regression has a promising start for recall that contrasts with the low specificity values, although in general the performance is still underwhelming. MLP is more balanced in the sense that all metrics start relatively high and follow a similar descending trend with occasional jumps. What is really peculiar, though, about SVM and MLP is the large standard deviation in recall and specificity, which suggests that they alternate between mislabeling photons or electrons depending on initialization.

Regarding the bagging and boosting models, there is a clear trend where recall and NPV are initially low but they quickly pick up as more hits are added to the dataset. In other words, with fewer hits these models fail more at spotting all electrons and at preventing false

Table 4
Classification accuracy (validation, 10 runs) ± standard deviation of each model for different hit caps. The best result for each column is highlighted in bold.

# hits	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
10	0.7508	0.7399 ± 0.0069	0.7215 ± 0.0051	0.7649	**0.5668 ± 0.0049**	0.5622	**0.734 ± 0.0095**	0.7663 ± 0.0036	0.7604
20	0.7508	0.7547 ± 0.0042	0.7259 ± 0.0118	0.7765	0.551 ± 0.0132	0.5662	0.713 ± 0.0116	0.7825 ± 0.0047	0.7674
30	0.7599	0.7662 ± 0.0057	0.7274 ± 0.0082	0.7807 ± 0.0002	0.5532 ± 0.0097	**0.5839**	0.6875 ± 0.0099	0.7951 ± 0.0027	0.777
40	0.7664	0.79 ± 0.0066	0.7171 ± 0.0239	0.7907	0.5445 ± 0.0149	0.5758	0.661 ± 0.0081	0.8191 ± 0.003	0.8028
50	0.771	0.802 ± 0.0058	0.7347 ± 0.0116	0.8049	0.5328 ± 0.0144	0.5551	0.6467 ± 0.0139	0.8356 ± 0.0035	0.815
60	0.7715	0.8135 ± 0.0056	0.7326 ± 0.0247	0.8145	0.5482 ± 0.0153	0.541	0.6186 ± 0.0154	0.8454 ± 0.0031	0.8291
70	0.7892	0.825 ± 0.0051	0.7415 ± 0.02	0.8251	0.537 ± 0.0152	0.5162	0.6007 ± 0.0151	0.8528 ± 0.0029	0.8392
80	0.7811	0.8353 ± 0.0031	0.7573 ± 0.0109	0.8263 ± 0.0002	0.5395 ± 0.0109	0.5212	0.5874 ± 0.0107	0.8671 ± 0.0031	0.8584
90	0.7811	0.8447 ± 0.0046	0.7598 ± 0.0126	0.8316	0.5247 ± 0.0122	0.5121	0.5631 ± 0.0099	0.8729 ± 0.003	0.8635
100	0.7907	0.8465 ± 0.0042	0.7535 ± 0.019	0.8377 ± 0.0002	0.5237 ± 0.0105	0.5086	0.5553 ± 0.0165	0.8757 ± 0.0025	0.868
110	0.7897	0.8545 ± 0.0038	0.7569 ± 0.0233	0.8354 ± 0.0002	0.5178 ± 0.0117	0.5025	0.5494 ± 0.0141	0.8853 ± 0.0021	0.8756
120	**0.7922**	0.8593 ± 0.0034	0.7456 ± 0.0212	0.8407	0.5176 ± 0.0143	0.499	0.5325 ± 0.0158	0.8881 ± 0.0019	0.867
130	0.7826	0.8591 ± 0.0046	0.7658 ± 0.0116	0.8418	0.5109 ± 0.0157	0.498	0.5292 ± 0.0084	0.8895 ± 0.0037	0.8817
140	0.7867	0.8599 ± 0.0039	0.7623 ± 0.0158	0.8392	0.5108 ± 0.0104	0.5061	0.523 ± 0.0145	0.8904 ± 0.0013	**0.8862**
150	0.7887	0.8603 ± 0.0041	0.7605 ± 0.0139	**0.8458**	0.5108 ± 0.0105	0.5066	0.5236 ± 0.0153	0.8897 ± 0.0024	0.8766
175	0.7887	0.8606 ± 0.0039	0.7644 ± 0.0127	0.8433	0.5161 ± 0.0113	0.501	0.5059 ± 0.0139	**0.8907 ± 0.0032**	**0.8862**
200	0.7887	0.8606 ± 0.004	**0.7691 ± 0.011**	0.8433	0.5108 ± 0.0079	0.504	0.5072 ± 0.0111	0.8896 ± 0.0011	0.8751
250	0.7887	**0.8614 ± 0.0032**	0.7578 ± 0.0134	0.8418	0.5018 ± 0.005	0.5035	0.5037 ± 0.0059	0.8885 ± 0.0035	0.8817

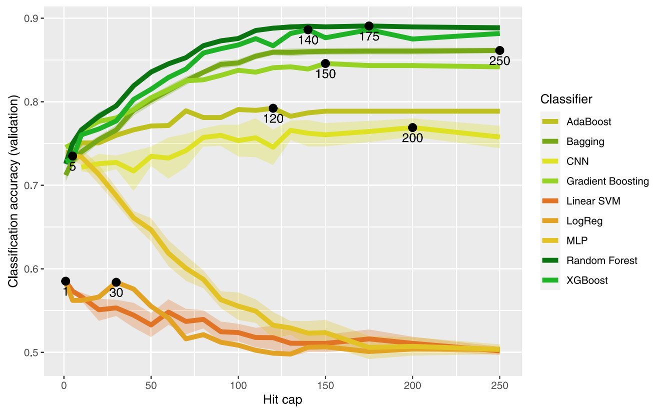

Fig. 7. Average classifier accuracy in the validation set for each hit cap tried. Shaded areas represent standard deviation.

negatives, as they are labeling more particles as photons. All models of this type exhibit the same behavior, but the most remarkable example of this change in performance is Random Forest, which goes from 0.72 to 0.91 in recall and from 0.74 to 0.91 in NPV.

Finally, CNN lies in a middle ground of not being particularly good or bad at any metric; as more information (hits) becomes available, its performance increases, which is normal in this model because it has its own built-in feature selection mechanisms.

Note that there is no single best method to choose the hit cap. Although we have focused on classifier performance, training cost is also a sensible metric, especially since some algorithms scale better than others. Moreover, choosing a single cap would simplify the workflow but at the expense of a suboptimal outcome, as evidenced by the results.

After finding an appropriate cutoff point for the size of the events, a related issue arises: is there a way to mitigate the information loss that inevitably occurs with the hit cap? The answer, if affirmative, probably involves adding back compressed information to the dataset rows. This is because we do not want to fully negate the reduction in computational cost and overfitting. In this paper, we explore a straightforward alternative (**Dataset C**): an additional feature that represents the original amount of hits of each event. This covers the possibility of event size being relevant on its own while still keeping dataset dimensionality in check.

The average accuracy of the nine classifiers in the corresponding validation split of Datasets B and C can be seen in Table 5. The numbers suggest that there are no apparent advantages to adding the hit count as a feature. Rather, it even shows notably poorer performance for some classifiers (CNN, Linear SVM, Logistic Regression). A paired permutation test produces a p-value of 0.2734. The evidence is not strong enough to reject this kind of feature engineering in a general context, but the results do not encourage its use either. In this particular context, and assuming the equivalence of both alternatives, Dataset B is preferable as less processing is required.

4.2.3. Which learners are more promising in this context?

From the experiments performed up to this point it is already possible to observe significant differences among the classifiers. Although an exhaustive exploration of the potential of every one of them would be the ideal course of action to maximize accuracy, it is not feasible in most real-world circumstances. With the partial results obtained up to this point in mind, we have chosen Random Forest and XGBoost for the next step: hyperparameter optimization. Not only are these the two highest-performing algorithms in the group in terms of accuracy, but they can be parallelized by design, which will allow for a larger experimental scale. Moreover, they represent two different approaches to the bias–variance tradeoff: bagging and boosting, respectively.

Nevertheless, it is only fair to acknowledge some important nuances in this decision: XGBoost and Random Forest have been chosen for their classification accuracy, but other contexts may call for alternative criteria. For example, let us consider training times, which are depicted in Fig. 8 for increasing hit caps. There are four algorithms that score above 80% in accuracy, but when we consider the ratio of both metrics it is easy to see that Gradient Boosting would be the worst choice. On the opposite side, the low-performing algorithms, although comparatively cheap, do not come up to standard in accuracy.

Given the similar accuracy of Random Forest and XGBoost and the availability of a high-performance computing cluster, we can afford to optimize them both. With severe enough computational constraints, Random Forest would be the most sensible choice in light of the information we have.

There is one last detail to acknowledge before closing this Section: the training time for CNN presents an anomaly in the lower end of dataset sizes (hit caps). Since it does not make much sense for that to happen in view of the other values of the series, it is likely to be caused by the Tensorflow library.

4.2.4. Tabu search-based hyperparameter tuning for the selected learners

The two previous experiments tackled the choice of the best dataset from a set of proposals with different structures and features. Thus far, the classifiers have only used default parameters, which are not necessarily the most adequate for this application.

As stated in Section 3.3, the choice of a suitable set of hyperparameter values entails the resolution of a combinatorial optimization

Table 5
Classification accuracy (validation, 10 runs) ± standard deviation of each model in Datasets B and C with their best hit cap.

Dataset variant	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
Dataset B	0.7922	0.8614 ± 0.003	0.7691 ± 0.011	0.8458	0.5668 ± 0.005	0.5839	0.7340 ± 0.010	0.8907 ± 0.003	0.8862
Dataset C	0.7846	0.8600 ± 0.003	0.7365 ± 0.041	0.8423	0.5039 ± 0.01	0.5546	0.7534 ± 0.01	0.8885 ± 0.002	0.8883

Table 6
Average classification accuracy (test, 50 runs) ± standard deviation, and maximum and minimum accuracy of Tabu Search-optimized Random Forest and XGBoost with their best hit cap. Baseline with default parameters added for comparison.

Classifier	Avg. accuracy (baseline)	Min. & max. accuracy (baseline)	Avg. accuracy (opt.)	Min. & max. accuracy (opt.)
Random Forest	0.8786	(0.8741, 0.8847)	0.8819 ± 0.001	(0.8794, 0.8837)
XGBoost	0.8746	(0.8746, 0.8746)	0.8823 ± 0.003	(0.8705, 0.8882)

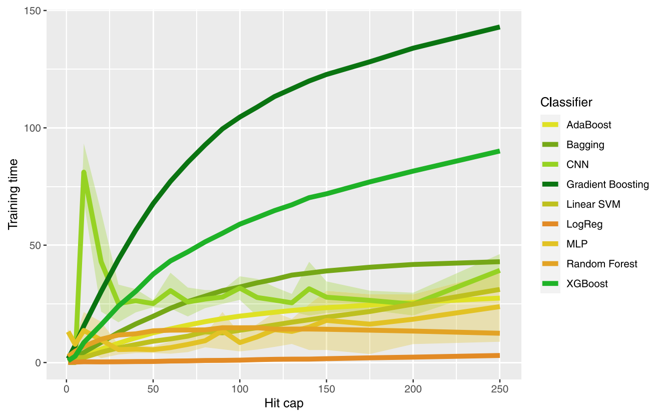

Fig. 8. Average classifier training time (10 runs) for each hit cap tried. Shaded areas represent standard deviation.

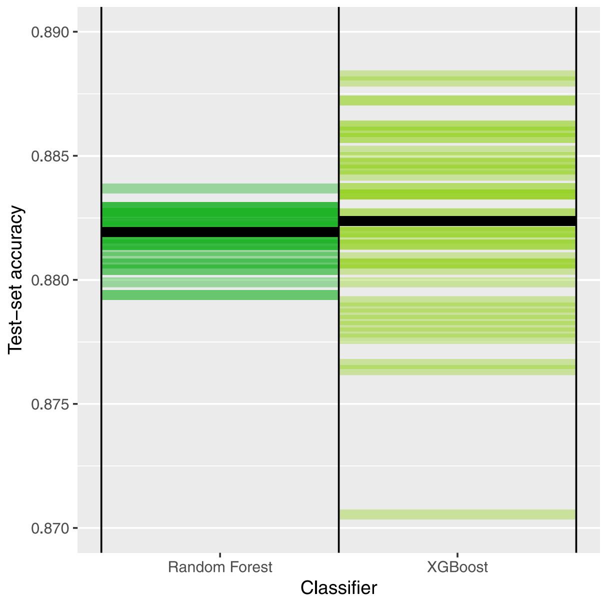

Fig. 9. Test-set accuracy distribution of the 50 optimization trials for each classifier.

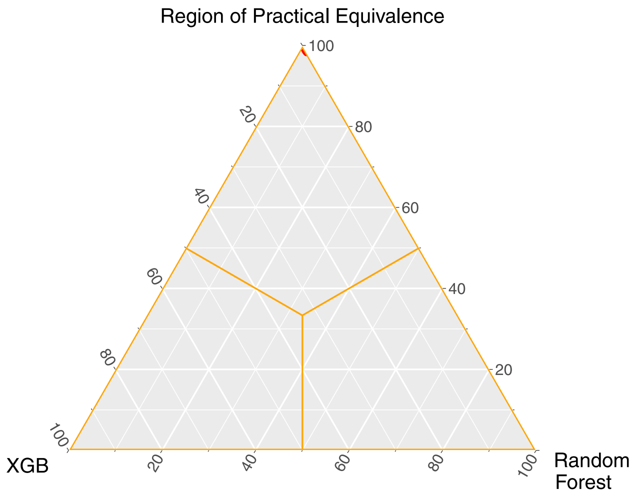

Fig. 10. Heatmap of the Bayesian Signed-Rank Test that compares Random Forest to XGBoost. Proximity of the point cloud to a vertex indicates a greater probability of prevalence for that option. In this case, the test strongly suggests practical equivalence.

problem. Thus, both manual and brute-force search are out of the question, although for different reasons: manual search is slow, arbitrary, and hardly reproducible; brute-force search is computationally expensive for any serious attempt at optimizing a Random Forest or XGBoost model.

The goal of this experiment is to find a hyperparameter combination through Tabu Search that is able to improve classification accuracy.

The test-set results of 50 runs of the Tabu Search for each classifier can be found in Table 6, alongside those of 50 runs of the baseline unoptimized models. Two things stand out: the first is that both classifiers achieve a nearly identical performance on average; the second one is that the hyperparameter optimization procedure is not able to improve the results by any large margin.

The first one can be studied in more detail by observing Fig. 9. The averages are quite similar, but XGBoost is more unstable in comparison: it achieves the highest and lowest accuracy of the two. However, it is also worth saying that the vertical axis exaggerates the differences for the sake of the analysis by focusing on a narrow range of values.

A Bayesian Signed-Rank Test (heatmap in Fig. 10) with a region of practical equivalence of (−0.01, 0.01) tells us that the differences between Random Forest and XGBoost are negligible: it assigns a probability of 99.93% to their practical equivalence. This is readily noticed when looking at the heatmap, as the point cloud is at the top vertex of the triangle.

For completeness, the Random Forest model with the highest test-set accuracy had the following hyperparameter values (default values shown in brackets):

- `n_estimators`: 920 (100).
- `max_features`: square root (square root).
- `min_samples_leaf`: 2 (1).
- `min_samples_split`: 8 (2).

For XGBoost, the values for the best model are:

- `n_estimators`: 838 (100).
- `learning_rate`: 0.0169 (0.3).

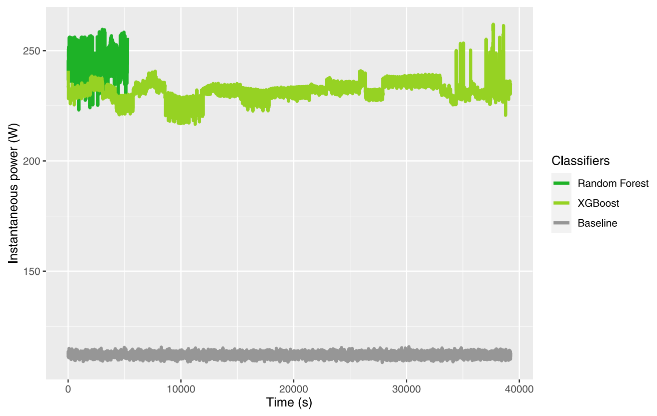

Fig. 11. Instantaneous power over time for one run of hyperparameter optimization for Random Forest and XGBoost.

- `subsample`: 97.92% (100%).
- `max_depth`: 10 (6).

As acknowledged earlier, the hyperparameter optimization procedure does not achieve a very noticeable accuracy improvement in relation to the baseline, especially in the case of Random Forest. There, the overall performance sees a small increase, but the maximum accuracy was achieved by one of the unoptimized models (this is possible because different seeds were used, as what matters is the population as a whole). For XGBoost, the procedure appears to be more effective, but it is still a modest gain of 0.8 percentage points of accuracy. In general, what the optimization procedure does consistently produce is more robust outcomes (i.e., with smaller standard deviation).

The cause of such a small improvement could be a combination of several potential factors. Firstly, that better models do exist and the Tabu Search has not been able to find them with the current configuration. Secondly, that these two models achieve near-maximum performance out of the box (i.e., with default hyperparameters) and there is not much margin for improvement with this dataset representation. Thirdly, that this dataset representation has its limit around 88%–89% of accuracy and we have already reached it. The first is an issue of finding a more effective optimization procedure, while the second and third concern the use of other classifiers or dataset representations.

Regardless of the cause of the small performance gain, few alternatives are left as the second and third possible origins are restricted by problem formulation and the first one could involve an unreasonable amount of computation by carrying out a brute force approach.

To conclude this section, Table 7 contains additional test-set metrics about the optimized models. As can be observed, Random Forest wins in NPV and recall, which means that it is slightly better at selecting a very pure sample of photons, and at not missing true electrons, respectively. On the contrary, XGBoost wins in precision and specificity: it is better at simultaneously selecting electrons without a photon contamination and at not missing true photons. In other words, each algorithm is better at isolating instances of one class and detecting a higher number of instances of the opposite class.

This mix of strengths in different classes could seem strange, but it has some logic. The pairs precision–recall and NPV-specificity tend to generate conflicts of interest: predicting all instances of a class and predicting only true instances of that class is usually a trade-off. Thus, if for example one algorithm is good at finding all the electrons, it may introduce some false positives along the way; the opposite is also true.

Depending on the application, the emphasis can be placed on some metrics over others. The next experiment elaborates on this topic in the context of our problem.

Table 7
Average classification performance for different metrics (test, 50 runs) ± standard deviation of Tabu Search-optimized Random Forest and XGBoost with their best hit cap.

Metric	RF	XGBoost
NPV	0.8990 ± 0.001	0.8884 ± 0.005
Precision	0.8661 ± 0.001	0.8765 ± 0.004
Recall	0.9029 ± 0.001	0.8896 ± 0.005
Specificity	0.8610 ± 0.001	0.8752 ± 0.006

Table 8
Average classification performance for different metrics (test, 50 runs) ± standard deviation of Tabu Search-optimized Random Forest and XGBoost with their best hit cap. Here, the fitness function is precision ×recall.

Metric	RF	XGBoost
Accuracy	0.8817 ± 0.001	0.8823 ± 0.004
NPV	0.8987 ± 0.002	0.8883 ± 0.004
Precision	0.8659 ± 0.002	0.8765 ± 0.004
Recall	0.9025 ± 0.002	0.8894 ± 0.005
Specificity	0.8610 ± 0.002	0.8752 ± 0.005

4.2.5. *Alternate hyperparameter tuning in the context of particle physics*

Bearing in mind the physical motivation of the problem at hand, an alternative to optimizing a classifier by maximizing the accuracy is optimizing the statistical significance of a given signal (e.g., electrons) over an undesired selected background (photons). Under the assumption of a Poisson-distributed number of entries in each of the samples, this is mathematically equivalent to maximizing the product of recall × precision. This choice can also be justified intuitively: in the example, recall represents the efficiency of the classifier in selecting electrons, whereas precision measures the purity of the final selection. Ideally, one would aspire to maximize both metrics simultaneously; however, as stated in the previous section, increasing one generally comes at the prize of reducing the other, and a compromise can be found through the maximization of their product.

In line with this reasoning, we have applied the Tabu Search procedure again to Random Forest and XGBoost with exactly one change: accuracy has been swapped for the product of recall and precision as the fitness function.

Table 8 contains the test-set metrics for this experiment. When compared to the results found in Tables 6 and 7, it becomes apparent that the differences are minimal; the same conclusions from Section 4.2.4 can be extracted from these numbers. The hypothesis of the limit of the dataset stated there, though, becomes a bit more likely due to the now multiple occurrences of values in a narrow range.

The previous result does not necessarily imply that maximizing the product of precision and recall has no benefits. Given that our dataset comes from simulations, aspects like class balance are under control. However, in the real-world counterpart of this type of application, some particles may be much costlier to generate or much rarer than others, which would create large class imbalances. In these situations, if the goal was to detect the minority class, metrics that prioritize it would be sensible choices. If an additional goal was to also not overestimate the findings for the target class, the precision × recall product would work in that direction.

4.2.6. *The cost of hyperparameter tuning*

As stated earlier in this work, a balance between classification accuracy and computational cost has to be decided on; this balance will depend on the circumstances of the application. In this section, we want to provide a brief energy–time analysis of the most time-consuming part of the experimentation.

Table 9 contains the average energy and time consumption of the hyperparameter optimization step of both Random Forest and XGBoost. Vast differences can be seen in both metrics, which indicates that XGBoost is clearly more resource-intensive in these terms.

Table 9
Average time and energy consumption (5 runs) ± standard deviation, and maximum and minimum values for the hyperparameter optimization experiment using Tabu Search.

Classifier	Avg. time (s)	Min. & max. time (s)	Avg. energy ($W \cdot H$)	Min. & max. energy ($W \cdot H$)
Random Forest	5335 ± 400	(4649, 5695)	374 ± 40	(314, 423)
XGBoost	43977 ± 5000	(38394, 48168)	2768 ± 300	(2377, 3069)

Table A.10
Classification NPV (validation, 10 runs) ± standard deviation of each model for different hit caps. The best result for each column is highlighted in bold.

# hits	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
10	0.7392	0.7092 ± 0.0056	0.7291 ± 0.0099	0.7612	**0.591 ± 0.0056**	0.5906	**0.7456 ± 0.0069**	0.7459 ± 0.0049	0.7481
20	0.7415	0.722 ± 0.005	0.7378 ± 0.0145	0.7708	0.5782 ± 0.0176	0.5776	0.7369 ± 0.0107	0.7603 ± 0.0045	0.7524
30	0.7459	0.7338 ± 0.0056	0.7322 ± 0.0214	0.777 ± 0.0003	0.5754 ± 0.0085	**0.5863**	0.6988 ± 0.0106	0.7734 ± 0.0036	0.7652
40	0.7472	0.7557 ± 0.0074	0.718 ± 0.0327	0.7829	0.5612 ± 0.0118	0.5774	0.6937 ± 0.0324	0.8011 ± 0.0035	0.7953
50	0.7531	0.7706 ± 0.006	0.7443 ± 0.033	0.802	0.5574 ± 0.0265	0.5532	0.652 ± 0.022	0.8251 ± 0.0045	0.8083
60	0.7592	0.7866 ± 0.0073	0.7407 ± 0.0566	0.8124	0.556 ± 0.0178	0.5399	0.6657 ± 0.0594	0.8444 ± 0.0055	0.8228
70	0.7845	0.801 ± 0.005	0.7525 ± 0.0386	0.8299	0.5526 ± 0.0179	0.5157	0.6236 ± 0.059	0.8612 ± 0.0035	0.8478
80	0.7722	0.8126 ± 0.0033	0.7753 ± 0.0363	0.8313 ± 0.0004	0.549 ± 0.0267	0.5204	0.6099 ± 0.0657	0.8811 ± 0.0052	0.8645
90	0.7744	0.8236 ± 0.0052	0.778 ± 0.0267	0.8433	0.5334 ± 0.0122	0.5116	0.6148 ± 0.0723	0.8864 ± 0.0036	0.8622
100	**0.7869**	0.8259 ± 0.0042	0.7657 ± 0.0285	0.8473	0.529 ± 0.0193	0.5086	0.5937 ± 0.0542	0.8908 ± 0.0022	0.8731
110	0.7842	0.834 ± 0.0046	0.7732 ± 0.0523	0.8426 ± 0.0004	0.5281 ± 0.0196	0.5029	0.6012 ± 0.0543	0.9012 ± 0.0032	0.8819
120	0.783	0.8422 ± 0.0046	0.7631 ± 0.0548	0.8497	0.5201 ± 0.0166	0.4995	0.5765 ± 0.0696	0.9062 ± 0.0027	0.8744
130	0.775	0.842 ± 0.0055	0.7977 ± 0.035	0.8545	0.5211 ± 0.027	0.4986	0.5644 ± 0.0699	0.9086 ± 0.0051	0.8881
140	0.7768	0.8442 ± 0.0047	0.7815 ± 0.0425	0.8544	0.5126 ± 0.0168	0.5059	0.5427 ± 0.047	0.9116 ± 0.002	**0.8947**
150	0.7793	0.8437 ± 0.0055	0.758 ± 0.047	**0.8571**	0.5107 ± 0.0139	0.5066	0.5257 ± 0.0251	0.9111 ± 0.0041	0.883
175	0.7793	0.8448 ± 0.0057	0.7668 ± 0.039	0.8564	0.5211 ± 0.0156	0.5015	0.5072 ± 0.013	**0.9134 ± 0.0035**	**0.8947**
200	0.7793	0.8442 ± 0.0048	**0.7778 ± 0.0352**	0.8564	0.5275 ± 0.021	0.5045	0.5206 ± 0.0329	0.9119 ± 0.0027	0.8826
250	0.7793	**0.8455 ± 0.0033**	0.782 ± 0.0517	0.853	0.5474 ± 0.0814	0.5041	0.4985 ± 0.0219	0.9122 ± 0.0056	0.8881

Table A.11
Classification precision (validation, 10 runs) ± standard deviation of each model for different hit caps. The best result for each column is highlighted in bold.

# hits	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
10	0.7636	0.7814 ± 0.0114	0.7156 ± 0.0148	0.7688	**0.5528 ± 0.0047**	0.5471	**0.7246 ± 0.0204**	0.7905 ± 0.0022	0.774
20	0.7608	0.799 ± 0.0062	0.7176 ± 0.0233	0.7826	0.5379 ± 0.011	0.5576	0.6953 ± 0.023	0.809 ± 0.0068	0.7845
30	0.7755	0.8092 ± 0.0064	0.7258 ± 0.0199	0.7844	0.542 ± 0.0112	**0.5816**	0.6796 ± 0.0214	0.8207 ± 0.0042	0.79
40	0.789	0.8353 ± 0.0065	0.7299 ± 0.0523	0.799	0.5375 ± 0.0183	0.5743	0.6456 ± 0.0309	0.8396 ± 0.0035	0.8108
50	0.7917	0.8419 ± 0.0074	0.7329 ± 0.0307	0.8078	0.5238 ± 0.0116	0.5572	0.6523 ± 0.0387	0.8468 ± 0.0034	0.8219
60	0.7851	0.8463 ± 0.0053	0.741 ± 0.0323	0.8165	0.5476 ± 0.0225	0.5421	0.6126 ± 0.0496	0.8465 ± 0.0032	0.8357
70	0.794	0.8533 ± 0.0069	0.7444 ± 0.0467	0.8204	0.5342 ± 0.0203	0.5167	0.6257 ± 0.0671	0.8447 ± 0.0029	0.831
80	0.7906	0.8617 ± 0.0061	0.7485 ± 0.0293	0.8214	0.5489 ± 0.0287	0.5222	0.6308 ± 0.0746	0.8542 ± 0.0027	0.8526
90	0.7882	0.8688 ± 0.0055	0.7476 ± 0.0255	0.8208	0.5293 ± 0.0218	0.5127	0.5838 ± 0.078	0.8603 ± 0.0039	0.8648
100	0.7947	0.87 ± 0.0052	0.7448 ± 0.0247	0.8285 ± 0.0002	0.5242 ± 0.014	0.5086	0.5765 ± 0.0634	0.8618 ± 0.0033	0.8631
110	0.7955	0.8779 ± 0.0046	0.753 ± 0.0269	0.8284	0.5157 ± 0.0099	0.5021	0.5612 ± 0.0552	0.8707 ± 0.0019	0.8695
120	**0.8021**	0.8784 ± 0.0037	0.7501 ± 0.0528	0.8322	0.5304 ± 0.0344	0.4984	0.5442 ± 0.039	0.8716 ± 0.003	0.86
130	0.7906	0.8782 ± 0.0056	0.7441 ± 0.0222	0.8299	0.5119 ± 0.0247	0.4973	0.5926 ± 0.0777	**0.8721 ± 0.0033**	0.8755
140	0.7973	0.8771 ± 0.0047	0.7556 ± 0.0392	0.8252	0.5245 ± 0.0245	0.5063	0.5464 ± 0.0712	0.8712 ± 0.0023	0.8781
150	0.7987	0.8788 ± 0.0049	**0.7743 ± 0.0296**	**0.8351**	0.5214 ± 0.03	0.5066	0.5723 ± 0.0699	0.8705 ± 0.0025	0.8705
175	0.7987	0.8782 ± 0.0058	0.7691 ± 0.025	0.8311	0.5194 ± 0.0134	0.5005	0.5465 ± 0.0549	0.8703 ± 0.0039	**0.8781**
200	0.7987	0.8787 ± 0.0062	0.7674 ± 0.0278	0.8311	0.5083 ± 0.0089	0.5036	0.512 ± 0.016	0.8695 ± 0.0023	0.8679
250	0.7987	**0.8789 ± 0.0059**	0.7509 ± 0.0427	0.8312	0.5198 ± 0.0426	0.503	0.5585 ± 0.1016	0.8673 ± 0.0029	0.8755

If we take a look at the behavior over time (Fig. 11), a perhaps surprising detail can be spotted: although XGBoost optimization takes much longer to complete, it is Random Forest optimization that averages higher instantaneous power consumption. This fact has the implication that Random Forest (in this particular case) is more efficient at taking advantage of available computing resources.

Nonetheless, from an accuracy-per-watt perspective, it is clear that Random Forest outperforms XGBoost in the application at hand: both achieve similar test-set accuracy rates but Random Forest takes roughly 12% of the time XGBoost needs.

5. Conclusions and future work

The analysis of subatomic particles through data obtained from detectors and its subsequent processing by machine learning techniques has been gaining traction in the last few years. In this work we have focused on the classification of simulated showers produced by either a photon or an electron inside a liquid argon container.

The creation of viable dataset representations is no easy task, as there are particular issues and trade-offs to manage: the variable size of dataset entries may require a cutoff to standardize the dimensionality; some representations may benefit some classifiers more than others; or the processing of the original data should be scalable to much larger datasets, which might discourage overly complex operations or inefficient data structures. In this paper, we have proposed a core dataset representation scheme whose variants achieve varying levels of success when applied to a classification task via machine learning. All in all, with a test-set accuracy of roughly 88% for the best models, we consider this a promising starting point for further research.

An immediate area of improvement would be the hyperparameter optimization step. Although Tabu Search, as configured in this work, has produced slight benefits, more experiments are required to explore the potential impact of this step. This might include other configurations of the Tabu Search procedure or completely different approaches such as evolutionary algorithms or Bayesian optimization.

Another area of interest is the generalization capabilities of the procedures proposed in this paper. This field of application is still

Table A.12
Classification recall (validation, 10 runs) ± standard deviation of each model for different hit caps. The best result for each column is highlighted in bold.

# hits	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
10	0.7257	0.6656 ± 0.0086	0.7361 ± 0.0236	0.7571	0.6968 ± 0.0154	**0.7166**	0.7564 ± 0.0187	0.724 ± 0.0072	0.7348
20	0.7308	0.68 ± 0.0089	0.7482 ± 0.0323	0.7652	0.7229 ± 0.0288	0.6366	**0.7613 ± 0.0275**	0.7391 ± 0.0063	0.7368
30	0.7308	0.6961 ± 0.0077	0.7334 ± 0.0421	0.7735 ± 0.0004	0.6955 ± 0.0527	0.5951	0.7131 ± 0.033	0.7547 ± 0.0056	0.754
40	0.7267	0.722 ± 0.0105	0.7077 ± 0.0833	0.7763	0.6764 ± 0.0868	0.583	0.7303 ± 0.08	0.7886 ± 0.0045	0.7895
50	0.7348	0.7432 ± 0.0085	0.7455 ± 0.0648	0.7996	0.7386 ± 0.0734	0.5324	0.6502 ± 0.0841	0.819 ± 0.0055	0.8036
60	0.747	0.7658 ± 0.0103	0.7263 ± 0.1142	0.8107	0.5994 ± 0.1119	0.5213	0.7065 ± 0.152	0.8434 ± 0.007	0.8188
70	0.7804	0.7846 ± 0.0063	0.7504 ± 0.0724	0.832	0.6553 ± 0.1547	0.4858	0.6021 ± 0.1944	0.8641 ± 0.0037	0.8512
80	0.7642	0.7985 ± 0.0049	0.7815 ± 0.0653	0.8334 ± 0.0005	0.55 ± 0.197	0.4879	0.5467 ± 0.239	0.8851 ± 0.0057	0.8664
90	0.7682	0.8116 ± 0.0064	0.788 ± 0.0439	0.8482	0.6047 ± 0.2176	0.4696	0.6576 ± 0.2505	0.8901 ± 0.004	0.8613
100	**0.7834**	0.8145 ± 0.0049	0.7731 ± 0.0417	0.8512	0.5479 ± 0.1414	0.4798	0.6294 ± 0.2621	0.8947 ± 0.0022	0.8745
110	0.7794	0.8233 ± 0.0057	0.7715 ± 0.0961	0.8455 ± 0.0005	0.5982 ± 0.1909	0.4798	0.6857 ± 0.2661	0.9049 ± 0.0034	0.8836
120	0.7753	0.8338 ± 0.0057	0.7572 ± 0.1082	0.8532	0.4972 ± 0.2211	0.4696	0.6238 ± 0.2759	0.9101 ± 0.003	0.8765
130	0.7682	0.8336 ± 0.0069	**0.8134 ± 0.0514**	0.8593	0.5987 ± 0.2243	0.4666	0.4664 ± 0.3588	0.9127 ± 0.0052	0.8897
140	0.7682	0.8366 ± 0.0056	0.7852 ± 0.0713	0.8603	0.4261 ± 0.2446	0.4504	0.5418 ± 0.2661	0.916 ± 0.0022	**0.8968**
150	0.7713	0.8356 ± 0.0069	0.7421 ± 0.0831	**0.8613**	0.4288 ± 0.2353	0.4686	0.4482 ± 0.2953	0.9155 ± 0.0043	0.8846
175	0.7713	**0.8371 ± 0.0075**	0.76 ± 0.0666	**0.8613**	0.4517 ± 0.2629	0.4858	0.3417 ± 0.3255	**0.918 ± 0.0035**	**0.8968**
200	0.7713	0.8363 ± 0.0061	0.7772 ± 0.0572	**0.8613**	0.6192 ± 0.2881	0.5	0.6257 ± 0.2888	0.9165 ± 0.003	0.8846
250	0.7713	0.838 ± 0.0044	0.785 ± 0.0863	0.8573	0.4599 ± 0.4331	0.5081	0.3925 ± 0.3638	0.917 ± 0.0057	0.8897

Table A.13
Classification specificity (validation, 10 runs) ± standard deviation of each model for different hit caps. The best result for each column is highlighted in bold.

# hits	AdaBoost	Bagging	CNN	Gradient Boosting	Linear SVM	LogReg	MLP	RF	XGB
10	0.7758	0.814 ± 0.0127	0.707 ± 0.0303	0.7727	0.4372 ± 0.0203	0.4081	**0.7117 ± 0.0352**	0.8085 ± 0.0018	0.7859
20	0.7707	0.8292 ± 0.0072	0.7037 ± 0.0487	0.7879	0.3795 ± 0.0412	0.496	0.6647 ± 0.0469	0.8258 ± 0.0077	0.798
30	0.7889	0.8362 ± 0.0059	0.7214 ± 0.0418	0.7879	0.4112 ± 0.0678	0.5727	0.662 ± 0.0489	0.8355 ± 0.0052	0.8
40	0.8061	0.858 ± 0.0059	0.7266 ± 0.0984	0.8051	0.4128 ± 0.1124	0.5687	0.5918 ± 0.0944	0.8496 ± 0.0039	0.8162
50	0.8071	0.8607 ± 0.0075	0.7239 ± 0.0637	0.8101	0.3275 ± 0.0866	**0.5778**	0.6431 ± 0.1072	0.8521 ± 0.0036	0.8263
60	0.796	0.8612 ± 0.0056	0.7389 ± 0.0811	0.8182	0.4971 ± 0.1324	0.5606	0.5308 ± 0.1695	0.8474 ± 0.0042	0.8394
70	0.798	0.8654 ± 0.0072	0.7325 ± 0.0994	0.8182	0.419 ± 0.1742	0.5465	0.5993 ± 0.2206	0.8415 ± 0.0031	0.8273
80	0.798	0.872 ± 0.0069	0.7332 ± 0.0601	0.8192	0.529 ± 0.2092	0.5545	0.628 ± 0.2523	0.8492 ± 0.0032	0.8505
90	0.7939	0.8777 ± 0.0057	0.7317 ± 0.051	0.8152	0.4449 ± 0.2382	0.5545	0.4688 ± 0.2656	0.8558 ± 0.0047	0.8657
100	0.798	0.8785 ± 0.0053	0.734 ± 0.0389	0.8241 ± 0.0003	0.4996 ± 0.1414	0.5374	0.4813 ± 0.2851	0.8568 ± 0.0039	0.8616
110	0.8	**0.8857 ± 0.0048**	0.7424 ± 0.063	0.8253	0.4376 ± 0.1871	0.5253	0.4133 ± 0.281	0.8659 ± 0.0022	0.8677
120	**0.8091**	0.8847 ± 0.004	0.734 ± 0.1054	0.8283	0.5381 ± 0.2239	0.5283	0.4413 ± 0.2957	0.8662 ± 0.0037	0.8576
130	0.797	0.8845 ± 0.0059	0.7183 ± 0.0485	0.8242	0.4233 ± 0.2137	0.5293	0.5919 ± 0.3658	**0.8665 ± 0.0036**	0.8737
140	0.8051	0.883 ± 0.005	0.7394 ± 0.0783	0.8182	**0.5953 ± 0.2482**	0.5616	0.5042 ± 0.272	0.8648 ± 0.0029	**0.8758**
150	0.8061	0.8849 ± 0.0054	**0.7789 ± 0.0603**	**0.8303**	0.5925 ± 0.2343	0.5444	0.5989 ± 0.3047	0.864 ± 0.003	0.8687
175	0.8061	0.884 ± 0.0067	0.7688 ± 0.0499	0.8253	0.5804 ± 0.2535	0.5162	0.6698 ± 0.3345	0.8634 ± 0.0046	**0.8758**
200	0.8061	0.8847 ± 0.0068	0.761 ± 0.0567	0.8253	0.4025 ± 0.2824	0.5081	0.3889 ± 0.3007	0.8627 ± 0.0031	0.8657
250	0.8061	0.8847 ± 0.0066	0.7307 ± 0.0867	0.8263	0.5436 ± 0.4353	0.499	0.6146 ± 0.3653	0.86 ± 0.0032	0.8737

in its early development, with crucial facilities still being built and machine learning approaches in the beginning stages of research. A valuable addition would be to examine the behavior of proposals like the one presented here in more datasets of similar characteristics as they become available.

Finally, interpretability is a major concern in the current landscape of machine learning. Many state-of-the-art techniques resemble black boxes in that we cannot understand the rationale behind their choices; neural networks are a prominent example. In order to extract valuable insight about the application, explainable AI might be a path worth the effort.

CRediT authorship contribution statement

Javier León: Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Visualization. **Juan José Escobar:** Conceptualization, Methodology, Writing – review & editing, Supervision. **Marina Bravo:** Validation, Resources, Data curation. **Bruno Zamorano:** Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing, Funding acquisition. **Alberto Guillén:** Conceptualization, Methodology, Software, Formal analysis, Resources, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Alberto Guillen reports financial support was provided by Spanish Ministry of Science, Innovation, and Universities.

Data availability

The manuscript contains a link to a GitHub repository where the results and code will be made available upon acceptance.

Acknowledgments

This research has been funded by the Spanish Ministry of Science, Innovation, and Universities under grants PGC2018-098813-B-C31 and PID2019-104676GB-C32, and the Project HPC-EUROPA3 (INFRAIA-2016-1-730897), FEDER-Andalucia Operative Programme 2010–2014 (A-FQM-472-UGR20), with the support of the EC Research Innovation Action under the H2020 Programme; in particular, authors gratefully acknowledge the support of Adrian Jackson and the computer resources and technical support provided by EPCC. Authors would like to thank as well Antonio Bueno for his valuable comments and discussions. Funding for open access charge: Universidad de Granada / CBUA.

Appendix. Classification metrics for different hit caps

See Tables A.10–A.13.

References

Abadi, M., et al., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL https://www.tensorflow.org/guide.

Abi, B., et al., 2020. Deep underground neutrino experiment (DUNE), far detector technical design report, volume I introduction to DUNE. JINST 15 (08), T08008. http://dx.doi.org/10.1088/1748-0221/15/08/T08008, arXiv:2002.02967.

Agostinelli, S., et al., 2003. Geant4—A simulation toolkit. Nucl. Instrum. Methods Phys. Res. A 506 (3), 250–303. http://dx.doi.org/10.1016/S0168-9002(03)01368-8.

Allison, J., et al., 2006. Geant4 developments and applications. IEEE Trans. Nucl. Sci. 53 (1), 270–278. http://dx.doi.org/10.1109/TNS.2006.869826.

Allison, J., et al., 2016. Recent developments in Geant4. Nucl. Instrum. Methods Phys. Res. A 835, 186–225. http://dx.doi.org/10.1016/j.nima.2016.06.125.

Anon, 2022a. Documentation of the scikit-learn random forest implementation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. (Accessed: 22 April 2022).

Anon, 2022b. Official xgboost documentation for the scikit-learn API. https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.sklearn. (Accessed: 22 April 2022).

Antonello, M., et al., 2013. Experimental search for the "LSND anomaly" with the ICARUS detector in the CNGS neutrino beam. Eur. Phys. J. C 73 (3), 2345. http://dx.doi.org/10.1140/epjc/s10052-013-2345-6, arXiv:1209.0122.

Assunção, F., Correia, J., Conceição, R., Pimenta, M.J.M., Tomé, B., Lourenço, N., Machado, P., 2019. Automatic design of artificial neural networks for gamma-ray detection. IEEE Access 7, 110531–110540. http://dx.doi.org/10.1109/ACCESS.2019.2933947.

Breiman, L., 1996. Bagging predictors. Mach. Learn. 24 (2), 123–140. http://dx.doi.org/10.1007/BF00058655.

Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32. http://dx.doi.org/10.1023/A:1010933404324.

Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., Zdeborová, L., 2019. Machine learning and the physical sciences. Rev. Modern Phys. 91 (4), 045002. http://dx.doi.org/10.1103/RevModPhys.91.045002.

Carrasco, J., García, S., del Mar Rueda, M., Herrera, F., 2017. Rnpbst: An R package covering non-parametric and bayesian statistical tests. In: International Conference on Hybrid Artificial Intelligence Systems. Springer, pp. 281–292. http://dx.doi.org/10.1007/978-3-319-59650-1_24.

Carrillo-Perez, F., Herrera, L., Carceller, J., Guillén, A., 2021. Deep learning to classify ultra-high-energy cosmic rays by means of PMT signals. Neural Comput. Appl. 33 (15), 9153–9169. http://dx.doi.org/10.1007/s00521-020-05679-9.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794. http://dx.doi.org/10.1145/2939672.2939785.

Chollet, F., et al., 2015. Keras. URL https://keras.io.

Cornell, A.S., Doorsamy, W., Fuks, B., Harmsen, G., Mason, L., 2022. Boosted decision trees in the era of new physics: A smuon analysis case study. J. High Energy Phys. 2022 (4), 1–36. http://dx.doi.org/10.1007/JHEP04(2022)015.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297. http://dx.doi.org/10.1007/BF00994018.

Cramer, J.S., 2002. The origins of logistic regression. http://dx.doi.org/10.2139/ssrn.360300, SSRN.

Dorigo, M., Birattari, M., Stutzle, T., 2006. Ant colony optimization. IEEE Comput. Intell. Mag. 1 (4), 28–39. http://dx.doi.org/10.1109/MCI.2006.329691.

Eiben, A.E., Smith, J.E., et al., 2003. Introduction to Evolutionary Computing, Vol. 53. Springer, http://dx.doi.org/10.1007/978-3-662-44874-8.

Erdmann, M., Geiger, L., Glombitza, J., Schmidt, D., 2018. Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks. Comput. Software Big Sci. 2 (1), 1–9. http://dx.doi.org/10.1007/s41781-018-0008-x.

Freund, Y., Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. System Sci. 55 (1), 119–139. http://dx.doi.org/10.1006/jcss.1997.1504.

Freund, Y., Schapire, R.E., et al., 1996. Experiments with a new boosting algorithm. In: Icml, Vol. 96. Citeseer, pp. 148–156. http://dx.doi.org/10.5555/3091696.3091715.

Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. Ann. Statist. 1189–1232. http://dx.doi.org/10.1214/aos/1013203451.

Friedman, J.H., 2002. Stochastic gradient boosting. Comput. Statist. Data Anal. 38 (4), 367–378. http://dx.doi.org/10.1016/S0167-9473(01)00065-2.

Glover, F., Laguna, M., 1998. Tabu search. In: Handbook of Combinatorial Optimization. Springer, pp. 2093–2229. http://dx.doi.org/10.1007/978-1-4615-6089-0.

Guillén, A., Bueno, A., Carceller, J., Martínez-Velázquez, J., Rubio, G., Peixoto, C.T., Sanchez-Lucas, P., 2019. Deep learning techniques applied to the physics of extensive air showers. Astropart. Phys. 111, 12–22. http://dx.doi.org/10.1016/j.astropartphys.2019.03.001.

Guillén, A., Martínez, J., Carceller, J.M., Herrera, L.J., 2020. A comparative analysis of machine learning techniques for muon count in UHECR extensive air-showers. Entropy 22 (11), 1216. http://dx.doi.org/10.3390/e22111216.

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., Lew, M.S., 2016. Deep learning for visual understanding: A review. Neurocomputing 187, 27–48.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585 (7825), 357–362. http://dx.doi.org/10.1038/s41586-020-2649-2.

Huennefeld, M., 2017. Deep learning in physics exemplified by the reconstruction of muon-neutrino events in icecube. Verh. Dtsch. Phys. Ges..

Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680. http://dx.doi.org/10.1126/science.220.4598.671.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444. http://dx.doi.org/10.1038/nature14539.

Machado, P.A., Palamara, O., Schmitz, D.W., 2019. The short-baseline neutrino program at Fermilab. Annu. Rev. Nucl. Part. Sci. 69 (1), 363–387. http://dx.doi.org/10.1146/annurev-nucl-101917-020949.

Paganini, M., de Oliveira, L., Nachman, B., 2018. CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. Phys. Rev. D 97 (1), 014021. http://dx.doi.org/10.1103/PhysRevD.97.014021.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Pitman, E.J., 1937. Significance tests which may be applied to samples from any populations. Suppl. J. R. Stat. Soc. 4 (1), 119–130. http://dx.doi.org/10.2307/2984124.

Poli, R., Kennedy, J., Blackwell, T., 2007. Particle swarm optimization. Swarm Intell. 1 (1), 33–57. http://dx.doi.org/10.1007/s11721-007-0002-0.

Radovic, A., Williams, M., Rousseau, D., Kagan, M., Bonacorsi, D., Himmel, A., Aurisano, A., Terao, K., Wongjirad, T., 2018. Machine learning at the energy and intensity frontiers of particle physics. Nature 560 (7716), 41–48. http://dx.doi.org/10.1038/s41586-018-0361-2.

Raschka, S., 2018. MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. J. Open Source Softw. 3 (24), http://dx.doi.org/10.21105/joss.00638, URL http://joss.theoj.org/papers/10.21105/joss.00638.

Rubbia, C., 1977. The Liquid-Argon Time Projection Chamber: A New Concept for Neutrino Detectors. CERN-EP-INT 77–08.

Shalev-Shwartz, S., Ben-David, S., 2014. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, pp. 60–65.

Zyla, P.A., et al., 2020. Review of Particle Physics. PTEP 2020 (8), 083C01. http://dx.doi.org/10.1093/ptep/ptaa104.