EAI Endorsed Transactions
on Energy Web
Research Article EAI.EU

Comparison between LightGBM and other ML algorithms in PV fault classification

Paulo Monteiro[1,*], José Lino[1], Rui Esteves Araújo[2] and Louelson Costa[3]

[1]Faculty of Engineering of the University of Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
[2]INESC TEC and Faculty of Engineering of the University of Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
[3]INESC TEC, Campus FEUP, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

Abstract

In this paper, the performance analysis of Machine Learning (ML) algorithms for fault analysis in photovoltaic (PV) plants, is given for different algorithms. To make the comparison more relevant, this study is made based on a real dataset. The goal was to use electric and environmental data from a PV system to provide a framework for analysing, comparing, and discussing five ML algorithms, such as: Multilayer Perceptron (MLP), Decision Tree (DT), K-Nearest Neighbors (KNN), Support Vector Machine (SVM) and Light Gradient Boosting Machine (LightGBM). The research findings suggest that an algorithm from the Gradient Boosting family called LightGBM can offer comparable or better performance in fault diagnosis for PV system.

Keywords: Photovoltaic faults, Fault diagnostics, Fault classification, Data-driven, Machine Learning

Received on 11 November 2023, accepted on 07 January 2024, published on 16 January 2024

doi: 10.4108/ew.4865

1. Introduction

Detecting and identifying faults in PV systems plays a crucial role in the efficient and reliable operation of these systems [1]. Unidentified faults and inadequate maintenance issues can have several negative consequences, including environmental impacts, monetary losses, energy loss and reduced efficiency of the systems. In the environmental scope, the early detection of faults in PV systems can result in significant environmental impacts. Undetected failures, such as degradation of solar cells or failure to track the sun, can lead to a reduction in renewable energy production. This results in a greater reliance on conventional energy sources such as fossil fuels, which contributes to greenhouse gas emissions and global warming. Poor detection mechanisms of faults in PV systems can also have a significant financial impact. Unidentified faults can lead to a reduction in power generation and consequently a decrease in the revenue generated by PV systems. Lack of early detection and proper intervention can result in additional expenses and considerable financial losses. In a more technical aspect undetected failures or inaccurate predictions of failures in PV systems can lead to increase the mean time to repair (MTTR) of these systems. When individual components, such as PV modules or inverters fail, the overall energy output of the system is affected. In addition, undetected faults can spread and cause further damage to other components, further amplifying energy and system performance losses. In recent years, ML algorithms have become a valuable tool in the fault diagnosis of PV systems. With the ability to process and analyse large amounts of data, ML algorithms can identify potential faults and performance issues in PV systems, allowing for more effective and efficient fault analysis. The detection and subsequent classification of faults in solar PV systems is where these two environments meet. ML techniques can accurately identify patterns and correlations in the data that may not be immediately seem to human

[*]Corresponding author. Email: up201608557@edu.fe.up.pt

analysts. This enables the development of predictive maintenance strategies that can help prevent system failures before they occur, ultimately improving the reliability and performance of PV power systems [2, 3].

2. PV system and dataset

The dataset used in this paper was collected from [2]. It is a real PV system that has 4 types of faults implemented. In this way, the dataset used is from a real installation that has the functionality to correctly and accurately emulate the faults, as illustrated in Fig. 1. The system consists of 16 C6SU-330P PV modules (PV1,...,16), divided into two strings of 8 modules each. The output of each string is connected to a 5 kW grid-connected inverter. In addition, there is also an electrical panel that includes: Fuses and Transient Voltage Suppressors (Circuit Protection Block), String Circuit Breakers (DS1,2) and Main Circuit Breakers (DM1). In conjunction, there are also elements capable to inject faults into system: twelve sockets (T1,...12) can be connected to create different short- circuit conditions, or they can be used together with the auxiliary circuit breakers (D1,...,8) to insert arbitrary resistors (degradation resistors) in series with the strings to create degradation faults. The same breakers (D1,...,8) can also be used individually to generate open circuit faults. The shading fault was generated by physically and artificially blocking sunlight by using objects with high opacity [2].

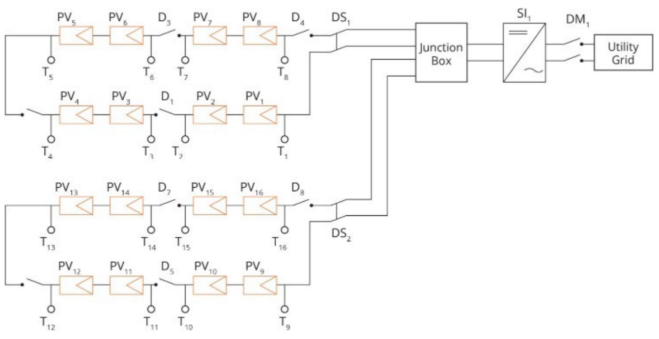


Fig. 1. Block diagram of the solar PV with emulate faults. Adapted from [2].

2.1. Data Description

The dataset obtained by the described PV system, has values that characterize the healthy and abnormal operation. Tables 1a and 1b summarize as the dataset was organized. The data contains five different types of classes: Degradation fault, Open-circuit (OC) fault, Short-circuit (SC) fault, Shadowing, and No fault. Each of these occurrences is built on a foundation of six distinct values (corresponding to the value of the input vector). These values are electric signals (current and voltage in the two strings of PV system) and environmental data (temperature and radiation). This dataset was saved in two Matlab files as follows: the electric dataset named dataset elec.mat, which contains the cur- rent and voltage values in the two strings. The environmental dataset named dataset amb.mat, which contained the environmental values (temperature and irradiance) [2].

Table 1. Variables obtained from the PV setup.

(a) Electrical variables from MATLAB file

dataset elec.mat	
Variable	**Description**
vdc1	Voltage - String 1
vdc2	Voltage – String 2
idc1	Current – String 1
idc2	Current – String 2

(b) Environmental variables from MATLAB file

dataset amb.mat	
Variable	**Description**
irr	Irradiance
pvt	PV module temperature
f nv	Fault Label

The electric and environmental values provided by the dataset are associated with a specific event (presence or absence of a fault). Again, the faults considered in the present study are: OC, SC, Degradation and shading. Labels were assigned to these occurrences as shown in table 2a. During the preprocessing of the data, the number of existing cases for each event was also analyzed, which is displayed in table 2b.

Table 2. Event Label and Data Proportion

(a) Fault Label

Fault Label	
Label	**Description**
0	Normal Operation
1	Short-Circuit
2	Degradation
3	Open Circuit
4	Shadowing

(b) Proportion of data.

Proportion of Data	
Class	**Nº of points collected**
SC	5999
OC	6024
Degradation	10371
Shadowing	184311
No Fault	309253

Something that can be perceived by analyzing the data available and it was that the dataset was imbalanced. In other words, there was a severe disproportionality of values between classes. The values present in each class are shown in the table 2b. In fact, the number of faulty values is smaller than the number of healthy values, but the number of failures in the various classes is large face to the amount of data representative of normal operation. This aspect represented a challenge for the goal of this paper, which was overcome by adapting the algorithms.

3. Applied algorithms

In this section we present the 5 types of ML algorithms used for fault classification. The algorithms were: MLP, DT, KNN, SVM and LightGBM (being this last one was used and adapted for the first time for works of this scope). These algorithms were adapted to the dataset used in this work, with recourse to data processing and treatment functions. The dataset was divided into two parts, one for training the algorithms and the other for testing them. As it is imbalanced, during the adaptation process several hypertunning cycles were performed in order to choose the best and most adequate parameters for each algorithm, so as to circumvent the unbalancing problem and obtain the best and most reliable results.

3.1. Python Language as a basic tool

All of the algorithms used in this work were written in Python. The Python code related to each algorithm is available in a public repository [4] and reveals implementation details that may be difficult to grasp based on pseudo code only. Readers are encouraged to download the code and run the algorithms and see [5] in order to find more details. The following open-source programs were leveraged in a Python script for data organization and manipulation:

- NumPy: NumPy serves as the principal container for data that is transmit- ted between algorithms in addition to giving Python quick array processing capabilities [6].
- SciPy: This library contains algorithms for a variety of problem types, including optimization, integration, interpolation, eigenvalue issues, algebraic equations, and differential equations. Moreover, it offers specific data structures, such as sparse matrices and k-dimensional trees [7].
- Pandas: Is a Python library that makes working with structured data made simple, easy, and intuitive with its built-in functions and rich data structures [6].

3.2. Multilayer perceptron

Artificial neural networks (ANNs) are computational models whose structure and operation are inspired by biological neural networks. They consist of inter-connected layers of "neurons" that process and transmit information. The neuron is the fundamental unit of an ANN, which receives input, processes it, and generates output. Each input value's significance is determined by weights applied to the input data. A neuron's output is determined by its activation function, which determines whether the neuron will produce an output. An input layer, one or more hidden layers, and an output layer are the organizational structure of neuronal layers. The input layer receives the input data and transmits it to the output layer, which generates the output. The hidden layers process the incoming data and forward it to the subsequent layer [8]. The ANN structure and parameters used are given in table 3.

Table 3. Chosen parameters for this MLP

Model Parameters	Chosen Parameters
Nº of hidden layers	3
Nº of neurons per hidden layer	13
Activation Function	relu
Solver for weight Optimization	adam
Learning Rate	adaptive
Maximum number of iterations	30

Note that ReLU, or rectified linear unit, is a popular activation function with the mathematical (equation 1), where x is the activation function's input. In this instance, the output of the ReLU function is the maximum of 0 and the its own input. It was used Adaptive Moment Estimation (ADAM) for weight solver optimizers because it performs well across a wide variety of tasks and requires minimal hypertuning. The remaining parameters were determined by a series of hypertuning sequences until the optimal set for this study case was identified.

$$f(\mathrm{x}) = \max\,(0, x). \tag{1}$$

3.3. Decision tree

A DT is a tree-like structure constructed by an algorithm to make a prediction or a determination. It divides a dataset into progressively smaller subsets while simultaneously developing an associated decision tree. The ultimate output is a tree composed of decision nodes and leaf nodes. A decision node has two or more branches, each of which represents a value for the tested attribute. A leaf node represents a numerical target decision. The decision node at the top of a tree that corresponds to the best predictor is known as the root node [9]. In order to adapt this algorithm to the data in present study the right parameters had to be chosen. The parameters used in this algorithm are presented in table 4.

Table 4. Chosen parameters for the DT.

Model Parameters	Chosen Parameters
criterion	Gini
splitter	Best
max_depth	4
max_leaf_nodes	7
class_weights	balanced

The DT parameters were chosen after several runs of the algorithm with ”trial-error” sessions and extensive hypertunning. The function that measures the quality of a split is a criterion. The ”gini” criterion is based on the ”giny” impurity measure, which is a measure of how often an item would be incorrectly labelled if it were randomly labelled based on the distribution of labels in the set. This is the best measure for this particular situation. The splitter is the parameter that determines the strategy for performing the split at each node. This parameter had the best value, which, as the name suggests, splits each node in the optimal manner. The parameter class weights have the power of associate weights to each class, it was chosen balanced to deal with the fact that the dataset used is unbalanced. The next parameter defines the maximum number of nodes in the tree and the second parameter defining the maximum number of leaves per node [10].

3.4 K-Nearest neighbors

KNN is a classification and regression algorithm for supervised learning. It works by locating the K data points in the training set that are closest to a new data point and making a prediction using those data points. In the case of classification, KNN works by locating the K closest training examples of the new data point and then voting on the majority class of K's nearest neighbours to determine the class of the new data point. For instance, if K is equal to three and two of the three nearest neighbors are class A and one is class B, then the new data point would be classified as class A. Besides this, KNN has a particularity that other classification algorithms do not have: it does not have a predefined training process, that is, it simply stores the data in groups and does not do it until it receives the test samples [11,13].
To adapt this model to the specific dataset, it was necessary to select the appropriate parameters, which are presented in the table 5.

Table 5. Chosen parameters for the KNN

Model Parameters	Chosen Parameters
n_neighbors	1000
algorithm	auto
weights	uniform

Extensive hypertuning was used to determine the best n neighbors's value. The second selected parameter specifies the algorithm for locating the k nearest points. Because this function can determine the optimal solution based on the inputted model data, this option was chosen. The final parameter specifies how distance-based weights are assigned to each class. In the classification, the chosen parameter gives greater weight to the closest points than to the most distant ones [11].

3.5. Support Vector Machine

In this study, classification was also carried out using the SVM algorithm, whose goal is to find the hyperplane in a high-dimensional space that maximally separates different classes and assigns the corresponding fault labels. To locate the hyperplane, the SVM algorithm identifies, as support vectors, the points in the training set that are closest to the hyperplane. It then maximizes the margin, which is the distance between the hyperplane and the support vectors. The resulting hyperplane is known as the maximum margin hyperplane. Increasing the margin increases the distance between the classes. This is advantageous because it decreases the possibility of misclassification of new data points. After obtaining the hyperplane, the SVM algorithm can predict new data by determining which side of the hyperplane the new data points fall on. Based on which side of the hyperplane the new data points fall on, the SVM algorithm will predict which class they belong to. In situations where the data cannot be separated linearly, the SVM algorithm can use a kernel trick to transform the data into a higher-dimensional space in which it can be separated linearly. In order to map the original data into a higher-dimensional space, a non-linear function known as a kernel function is applied to it. The SVM algorithm can then locate, within this higher-dimensional space, the hyperplane that maximally separates the various classes [12]. The most suitable parameters for the data used, can be found in table 6.

Table 6. Chosen parameters for SVM.

Model Parameters	Chosen Parameters
C	0.2
kernel	rbf
gamma	scale
class_weight	None
decision_function_shape	ovo

The first parameter is a regularization parameter that regulates the margin size (the separation between the nearest data points of various classes) in order to reduce misclassification error. Hypertuning led to the conclusion that 0.3 would be the ideal value for this work. The function used to transform the input data into a higher dimensional space, where it might be simpler to identify a linear boundary dividing the various classes, is called the kernel (second parameter). In our work, it was selected the radial base function (rbf), which computes the exponential of the negative Euclidean distance between the input vectors and permits highly nonlinear decision boundaries. This method was found to be the most effective after several cycles of hypertuning in conjunction with metric values. Gamma determines the shape of the decision boundary in the RBF kernel, so it is a parameter if “rbf” was selected as an option in the previous parameter. The model will produce a shape that is scaled to the type of data input since the scale option was selected. If RandomUnderSampling that distributed all classes equally was not made, the balanced option would be chosen, which is controlled by the class weights parameter. Since this was one of two options, it was decided not to alter the weight of the classes because it produced the best results. Last but not least, the parameter decision function shape is only defined when the data entered into the model has more than one class. The option “ovo”, which stands for one vs. one, was selected because it is the best option for training models with multiclass data, according to [12].

3.6. LightGBM

LightGBM is a gradient boosting framework designed specifically for classification tasks. It employs tree-based learning algorithms to build an ensemble of decision trees capable of accurately classifying input data. The training of a LightGBM classifier begins with the training of an initial tree on the entire dataset. The tree's predictions are then used to compute the residuals (the difference between the actual labels and the predicted labels) for the training set. The residuals are then used to train the second tree, whose predictions are used to update the residuals. This procedure is repeated for a predetermined number of times, or until a predetermined stopping criterion is met. The objective of each iteration is to minimize the loss function, which measures the deviation between the predicted and actual labels. Typically, this is accomplished in LightGBM by adjusting the weights of the training examples so that the next tree focuses more on the examples that were incorrectly predicted by the previous tree. Combining the predictions of all trained trees yields the final prediction. Each tree in the ensemble makes a prediction for a given input, and then these predictions are combined using a weighting scheme that assigns a greater weight to the trees that perform better. The most prevalent weighting scheme is the "weighted average" scheme, in which the weight of each tree is proportional to its accuracy on the training set. LightGBM's capacity to manage large-scale and high-dimensional data is one of its main advantages for classification applications. LightGBM employs histogram-based algorithms for continuous features, resulting in faster training and less memory consumption than conventional methods [14,15]. To make this model suitable for the given data, it was essential to make the right parameter selections, which are presented in table 7.

Table 7 Chosen parameters for the LightGBM.

Model Parameters	Chosen Parameters
boosting_type	goss
objective	multiclass
class_weight	cw
max_depth	4
num_leaves	7

The boosting type parameter is used to specify the boosting algorithm. In this case, goss (gradient-based one-side sampling) was selected, which divides the dataset into subsets as opposed to using the entire dataset (as in traditional Gradient Boosting algorithms) and demonstrated improved performance after hypertuning. In this case, the objective parameter was set to mulitclass, as classification of a multiclass dataset is the objective of this model. Class-specific weights are specified by the third parameter (class weight) [15]. For this parameter, it was determined the proportions between classes by calculating the weights for each class. The Scikit-Learn library’s LabelEncoder function was used to con- vert each fault label (categorical variable) into a numeric variable by assigning a unique integer value to each label or category in the input column. This allowed the labels to be passed to the computed class weight function of the same library, which can calculate weights to counteract imbalance in the dataset. The class weights used can be seen in table 8.

Table 8 Weights given to each class

Class	Weight
No Fault	0.28
SC	22.68
Degradation	13.12
OC	22.58
Shadowing	0.72

4 Results

This section presents and compares the results obtained by the algorithms de- scribed in section III. The comparative study is based on the criteria commonly used in the ML context. These are: precision, recall, f1- score and accuracy. So, we evaluate the performance of each algorithm in the classification of faults for the dataset used in this work. Then, we are discussing the obtained results from each algorithm.

4.1. Metrics

The performance of the algorithms will be determined by analyzing the fault classifications they produce. As mentioned previously, training values were incorporated into each model so that the algorithms could ”learn” to recognize patterns in the electrical and environmental values measured in the solar PV systems and assign them a fault label. Once the models have been trained, they will receive new test data and attempt to predict and classify the fault type corresponding to the measured values of the PV system. These will be compared to the previously assigned real values, thereby analysing its performance. As we are discussing algorithms with a classification function, the metrics will be associated with the truth of the classification, i.e. whether it is positive (instance classified as a member of the class that the classifier is attempting to identify) or negative (instance classified as not being a member of the class that the classifier is attempting to identify). It is from these considerations that the concept of True Positive (TP), True Negative (TN) and False Positive (FP) and False Negative (FN) is born [17]. The use of this 4 concepts changes to if the classification is binary or multiclass, as the data used in our work has more than one class, we will only explore these concepts for classifications with more than one class. The results will be analysed quantitatively through by commonly metrics used in context of ML, such as: Precision, Recall, F1-Score and Accuracy [18].

4.2. Discussion

Table 9 shows the overall performance of each algorithm using the parameters described previously and the metrics used. The conclusions drawn from these values are explained later in this section. At the end of the section, we also find a visual representation of the results to illustrate the differences clearer.

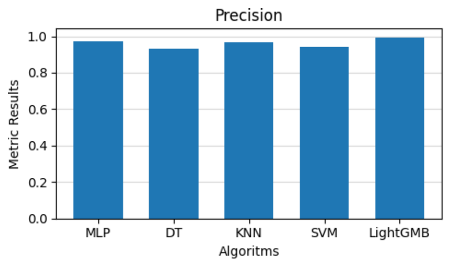
Algorithm	Metrics			
	Precision	Recall	F1-Score	Accuracy
MLP	0.97	0.97	0.98	0.96
DT	0.93	0.91	0.9	0.85
KNN	0.96	0.97	0.97	0.96
SVM	0.94	0.94	0.94	0.94
LightGBM	0.99	0.98	0.98	0.98

Table 9. Classification reports - Combined

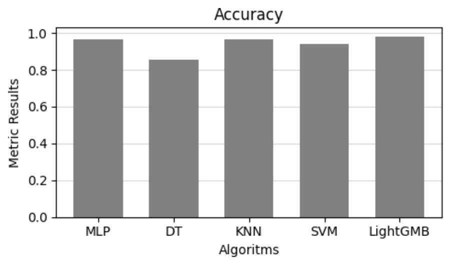
To sum up, in this paper it was used a multi-class imbalanced dataset of faults in PV systems to assess the efficacy of five ML algorithms (MLP, KNN, DT, SVM, and LightGBM). This classification exercise was designed to help determine what kind of PV system fault had occurred. The results showed that out of the five algorithms tested, LightGBM performed the best in terms of precision, accuracy, recall, and F1-score. The worst performing algorithm was DT. However, when the algorithms' readability and complexity are considered, the conclusions can be different. For instance, the decision-making process behind DTs is much simpler and straightforward to understand than that of other algorithms like MLP or SVM. However, LightGBM has a reputation for being less interpretable, which means that sometimes it can be difficult to fathom why the model has made a particular choice. When compared to MLP, SVM, and LightGBM, DTs are a straightforward model with few tuning options. In contrast, LightGBM is computationally intensive and may take more time to train than DTs because it has more parameters to adjust. The remaining algorithms also turned out to be good choices for this kind of work. MLP is a classic algorithm for work done in these environments, and as can be seen from the results, it performed well, made it easy to understand the data, and did not take up too much computing power. The KNN, unlike the MLP, has a high computational weight, even though it gets good metric results. Something that could only be solved by choosing a high k value in this work. Lastly, the SVM was the hardest to test because it took so much computing power, which could only be solved by resampling the data. This made it harder to understand this algorithm because the information it got was different from what it got before. The set of graphics presented in Fig. \ref {Comparison of Precision, Recall, F1-Score and Accuracy of all the algorithms} demonstrate and compare the overall metrics for each algorithm supporting the previously said in a more illustrative way.

Fig 2. Comparison between Precision, Accuracy, Recall and F1-Score of all the algorithms.

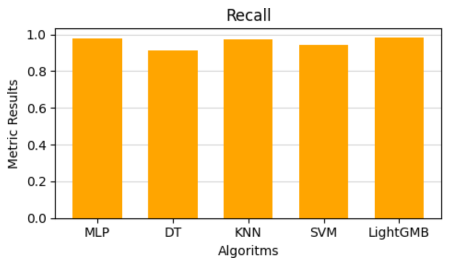
(a) Precision - All



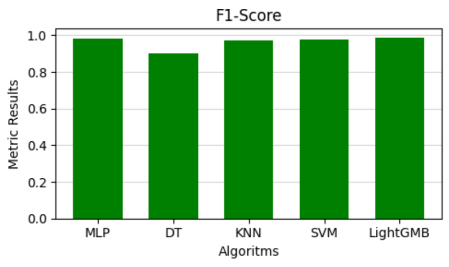
(b) Accuracy - All



(c) Recall - All



(d) F1-Score - All



From the studied algorithms, LightGBM was the one that performed the best. Besides the metrics obtained, its speed and efficiency in dealing with large and imbalanced datasets, its generalization capacity and it is data interpretability, due to the parameters an training procedure that constitutes LightGBM, made him significantly better then the other classic ML algorithms. So, from the results obtained in this work, it can be perceived that LightGBM constitutes a good choice for dealing with fault classification problem.

5 Conclusion

From this study it can be concluded that the LightGBM algorithms represent a promising avenue for recognizing correlations between the electric and environmental parameters of a solar PV system and, from there, classifying faults accordingly. After a systematic evaluation of studied algorithms, it can also be concluded that ML algorithms prove to be an excellent option for detecting and classifying faults in PV systems. However, it can be stated that there is no one-size-fits-all solution. Each algorithm possesses its own set of advantages and disadvantages.

As a final note, it is important to highlight that the behaviour of the algorithms and their results in the various metrics are in accordance with the data used, and that their behaviour, as well as the results obtained may vary. Indeed, these algorithms are highly susceptible to the data assigned to them.

References

[1] Sara Gallardo-Saavedra, Luis Hernandez-Callejo, Oscar Duque-Perez, Quantitative failure rates and modes analysis in photovoltaic plants. Energy, Volume 183, (2019), pp. 825-836, 0360-5442, https://doi.org/10.1016/j.energy.2019.06.185.

[2] Andre Eugenio Lazzaretti, Clayton Hilgemberg da Costa, Marcelo Paludetto Rodrigues, A monitoring system for online fault detection and classification in photovoltaic plants. Sensors (Switzerland), 20:1–30, 9 (2020), https://doi:10.3390/s20174688.

[3] Fouzi Harrou and Ying Sun and Bilal Taghezouit and Ahmed Saidi and Mohamed Elkarim Hamlati, Reliable fault detection and diagnosis of photovoltaic systems based on statistical monitoring approaches. Renewable Energy, Elsevier Ltd, (2018), https://doi:10.1016/j.renene.2017.09.048, 18790682.

[4] Paulo Monteiro, Github with Dissertation Scripts, github.com/paulo5930/Dissertation Scripts.

[5] Paulo Monteiro, Pattern Recognition Machine Learning Algorithms for Fault Classification of PV System, Master's Thesis, Faculty of Engineering of the University of Porto, 02-2023.

[6] Wes McKinney, Python for Data Analysis, 3rd edn. O REILLY, (2022), https://wesmckinney.com/book.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. Scipy 1.0: fundamental algorithms for scientific computing in python, Nature Methods, 17:261–272, (2020), https://doi.org/10.1038/s41592-019-0686-2.

[8] Walter H Delashmit and Michael T Manry, Recent developments in multilayer perceptron neural networks. In Proceedings of the Seventh Annual Memphis Area Engineering and Science Conference, MAESC (2005).

[9] Bahzad Charbuty and Adnan Abdulazeez, Classification based on decision tree algorithm for machine learning. Journal of Applied Science and Technology Trends, 2. 20-28, (2021), https://doi:10.38094/jastt20165

[10] Sickit Learn developers Homepage, DecisionTreeClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, last accessed 2023/10/19.

[11] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. KNN Model- Based Approach in Classification. Lecture Notes in Computer Science 2888, Pages 986 - 996, (2003).

[12] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing, Elsevier B.V., (2020), https://doi:10.1016/j.neucom.2019.10.118.

[13] Zhi-Hua Zhou: Machine Learning, Springer, Singapore (2021), https://link.springer.com/book/10.1007/978-981-15-1967-3.

[14] Essam Al Daoud: Comparison between xgboost, lightgbm and catboost using a home credit dataset. World Academy of Science, Engineering and Technology, Open Science Index 145, International Journal of Computer, and Information Engineering, 13(1), 6 - 10, (2019).

[15] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu, Lightgbm. A highly efficient gradient boosting decision tree, https: //github.com/Microsoft/LightGBM, last accessed 2023/10/19.

[16] LightGBM Homepage, lightgbm. LGBMClassifier, MicrosoftCorporation, https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html, last accessed 2023/10/19.

[17] Ankit Gupta, Evaluation Metrics For Multi-class Classification, Homepage Gaggle, https://www.kaggle.com/code/nkitgupta/evaluation-metrics-for-multi-class- classification, last accessed 2023/10/19.

[18] Teemu Kanstren, A Look at Precision, Recall, and F1-Score, Towards Data Science Homepage, https://towardsdatascience.com/a-look-at-precision-recall-and- f1-score-36b5fd0dd3ec, last accessed 2023/10/19.

[19] Sarang Narkhede, Understanding Confusion Matrix, Towards Data Science Homepage, https://towardsdatascience.com/understanding-confusion-matrix- a9ad42dcfd62, last accessed 2023/10/19.