Understanding binary-single encounters in Nuclear Star Clusters with Machine Learning

Flavio Calzolaio,[1] Fabrizio Muratore,[1] Linda Nardo[1] ⋆
[1]*Department of Physics and Astronomy, University of Padua, Vicolo dell'Osservatorio, 3, 35122, Padua*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

In this work, we present a machine learning method to classify binary-single encounters between black holes in nuclear star clusters. The interaction can result in four different outcomes: fly-by, two types of exchanges, and ionization. The data set consists of 100000 simulations of the encounters, split into 60% training and 40% test set. To simplify the classification task, we normalised the data and merged the exchanges into a single class. We compared two different algorithms: fully connected neural network and random forest. To account for the unbalance between the different outcomes, we chose to apply weighting factors to the models, in order to get comparable accuracies for each class. We finally analyzed the feature importance obtained from the two algorithms in order to understand which physical features were considered major players. The two models resulted in consistent values of the overall accuracy (∼ 73%). We argue that this value is near the maximum achievable since the problem is intrinsically chaotic and as such can not be perfectly classified by any algorithm.

Key words: methods: machine learning – three body problem – stars: black holes – nuclear star clusters

1 INTRODUCTION

Chaos is present in most stellar dynamical systems and it manifests itself through the exponential growth of small perturbations. Their exponential divergence drives time irreversibility and increases the entropy in the system (Boekholt et al. 2021). The Newtonian three-body problem is one of the standard examples for illustrating chaos. This line of research dates back to Poincaré, but more recent demonstrations have been made using computers. This problem exhibits exponential sensitivity to small changes in the initial condition. Over the lifetime of the interaction, small perturbations grow by about 9 orders of magnitude (Boekholt et al. 2020). Unfortunately, there is no analytic way to solve a three-body problem and the only reliable method to understand the evolution of this type of configuration is the numerical integration of the Newtonian equation. The main issue of this approach is the computational cost of evaluating the integration. Indeed, these calculations take a lot of computing time from servers. Due to the chaotic nature of the problem and the absence of an analytic method, understanding the configuration at a specific time is impossible without integration. A powerful way to solve the three-body problem comes from machine learning techniques (e.g. Breen et al. 2020), which are able to build algorithms that learn and improve their performances based on the analysed data set.

This work is focused on three-body encounters between black holes (BHs), in which two of them are initially bound in a binary system. These interactions can lead to different outcomes: a fly-by, meaning that the single object passes close to the binary but leaves the configuration unperturbed, an exchange between one of the two components of the binary and the intruder, or the ionization of the system, which is the destruction of the binary after the encounter, resulting in all bodies becoming single objects. The goal of this project is to use supervised learning methods for classification tasks in order to generate models able to predict the outcomes of the interactions between three BHs in nuclear star clusters.

The information we used came from 100000 simulations of three bodies interacting for 10^5 years, described in Dall'Amico et al. 2023. The data set is composed of 17 columns representing the initial conditions, while the others describe the final state of the system: outcome of the interaction (fly-by, exchange between m_3 and m_1, exchange between m_3 and m_2 and ionization), merging of the binary and formation of a second generation binary black hole. For the aim of this work we used only the information regarding the outcome of the encounter. The 17 features are: the masses m_1, m_2 and m_3 of BHs in solar masses; the semi-major axis of the original binary a in parsec, the eccentricity e, the phase of the binary at the beginning of the simulation f, the eccentric anomaly and the coalescence time in years of the binary, the impact parameter b in parsec and the maximum of the distribution from which it was sampled, the angles θ, ϕ, ψ in spherical coordinates and the velocity v of the single BH with respect to the center of mass of the binary.

The paper is structured as follows: in Section 2 we describe the classification algorithms, with a focus on each specific implementation in subsections 2.1 and 2.2; in Section 3 we summarize our work, describe the results and discuss the models.

⋆ flavio.calzolaio@studenti.unipd.it, fabrizio.muratore@studenti.unipd.it, linda.nardo@studenti.unipd.it

2 METHODS

In order to classify the data set we implemented a Fully Connected Neural Network (FCNN) and a Balanced Random Forest (BRF). In this section, we describe both methods.

The basic idea of a Neural Network (NN) is to process a data set through a series of interconnected nodes, and then produce an output. These nodes, also known as neurons, are arranged in layers. The input layer takes in the initial data, the hidden layers build the model based on the inputs, and the output layer produces the final result. During training, the NN adjusts the weights of the connections between the neurons in order to improve its own ability to accurately predict the output. This is done by comparing the predicted outputs with the outcomes of the simulations. A fully connected neural network is a type of artificial NN where all neurons in one layer are connected to all neurons in the next one. The output of each neuron in the hidden layers is a linear weighted sum of its inputs, transformed by the activation function into a non-linear value to be fed to the next hidden layer.

A Random Forest (RF) is a learning method that combines the predictions of multiple decision trees to make a more accurate final prediction. A decision tree is an algorithm that takes decisions by recursively splitting the data into subsets, based on threshold values of its features until a leaf node corresponding to a predicted output value is reached. The impurity of a splitting node is usually determined by the Gini function, which measures the frequency with which a random attribute gets misclassified. A RF combines the predictions of many decision trees. Each of them is trained on random subsets of both training set and features through the *bagging* technique. The randomness helps to reduce the risk of overfitting and improves the accuracy of the final prediction.

Due to their ability to learn from large high-dimensional data sets with complex and nonlinear relationships between the features and outputs, FCNN and RF are valid tools to model a problem such as ours.

In the pre-analysis, we dealt with the optimization of the data set. We merged the exchange classes' labels to deal with only three outcomes and simplify the task. Furthermore, we selected a subset of relevant features to minimize the complexity of the model and reduce the computational time. We therefore worked with m_1, m_2, m_3, a, b, θ, ϕ, ψ, f, ν, eccentric anomaly and time of coalescence. After that, we applied a random permutation to our data and split it into training and test set. Once the training set was defined, we applied the `scikit-learn` function `preprocessing.StandardScaler()` to normalize the data set.

2.1 Fully Connected Neural Network (FCNN)

The main python package used for building the neural network is `Keras`, with additional commands coming from the `scikit-learn` library.

Our first step was to vectorize the three possible outcomes, which in the database are integers 0, 1 or 2. This method, called One-Hot Encoding, transforms each integer outcome in a 1x3 vector: it has value 1 in the position corresponding to the integer, and zeros elsewhere. This is necessary as the design of our NN can provide only outputs between 0 and 1 and, as such, the outcome 2 can not be classified. This choice also necessarily fixes the output layer. It is obtained with a `softmax` activation function that returns a probability distribution between 0 and 1 for each outcome.

The entire layout of the NN has been chosen comparing different parameters with `GridSearchCV` and `RandomizedSearchCV` from `scikit-learn`: the former has been used to choose the main features of the neural network, such as the number of layers and neurons, and the latter to scan more parameters, such as batch size, activation functions in the layers, number of epochs, learning rate and optimizer. We then worked on the results of these searches, manually optimizing the model to get the best accuracy.

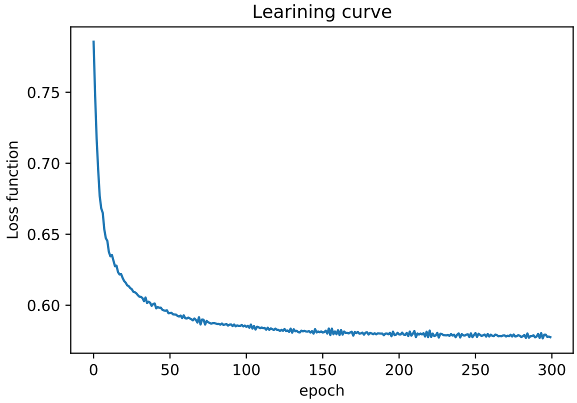

Figure 1. Loss function over epochs of the FCNN. The flattening of the trend shows the stabilizing effect of the regularization over the loss function and the learning rate.

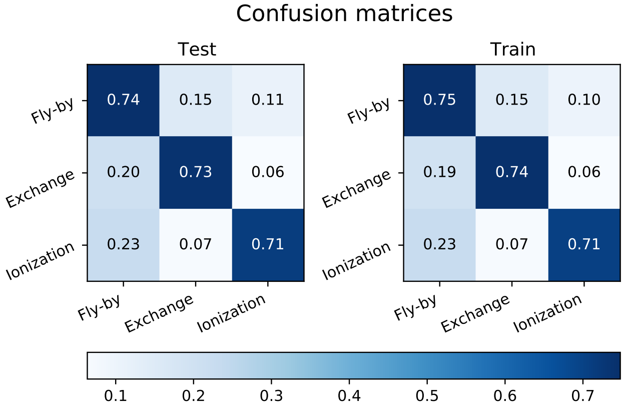

Figure 2. Confusion matrix for test and training set derived from the FCNN model. It has one column for each predicted label and one row for each true label and it shows for each class in the corresponding row how many samples belonging to that class get each possible output label. Notice that the diagonal contains the correctly classified samples, while the other cells correspond to errors.

The resulting structure of the neural network has an input layer of 13 neurons (as the number of features chosen in the data set), a single hidden layer of 40 neurons with a `ReLu` activation function, and an output layer of 3 neurons (as the number of outcomes). The best optimization algorithm for this work is `Adam`, a stochastic gradient descent method to update network weights iteratively. It is based on adaptive estimation of first-order and second-order moments. Furthermore, we decided to apply a regularized loss function with a L1L2 regularizer, which allows applying both L1 and L2 penalties on layer parameters or layer activity during optimization. These penalties are summed to the loss function in order to both minimize the empirical risk and prevent overfitting. This provides

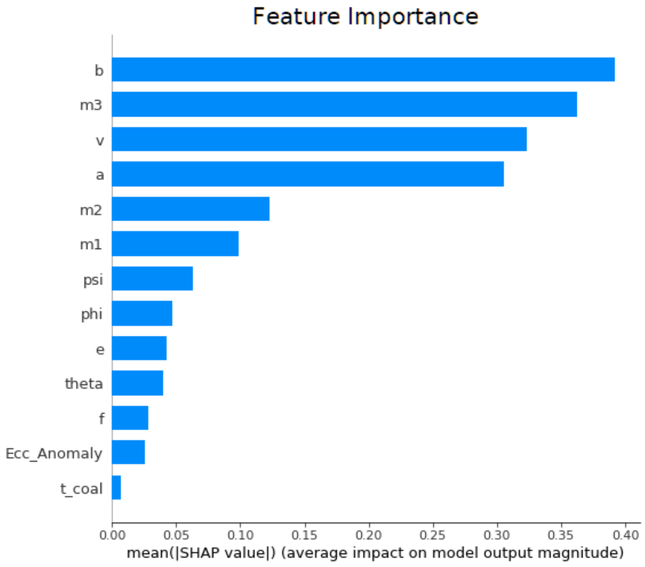

Figure 3. Pseudo feature importance plot which shows the SHAP value of each feature.

stability to the model, which can be clearly observed in the regularity of the loss function over the epochs (Figure 1).

To improve the model, we had also to take into account the unbalance between the three outcomes in the database. With this purpose, we decided to manually insert a bias, adding weights to the outcomes. We started from the basic "balanced" function provided from `scikit-learn` that chooses weights in order to artificially balance the data. This resulted in an uneven classification that had very high accuracy for the minority class (~ 80%) but low accuracy for the majority class. To avoid this problem, we choose to fix the weights manually by observing a posteriori the confusion matrix, to obtain a more homogeneous classification. The matrix compares the actual target values with those predicted by the model, giving us an overview of how well our model is performing and what kind of errors it is making. The final weights, along with the other parameters used, are listed in Table 2.1. To compile the model we used the categorical cross-entropy loss function and the categorical accuracy, which returns a training score of 74.3%, a validation score of 73.6% and a test score of 73.5%.

In order to understand how the model classifies the data, we used the `DeepExplainer` tool from the `SHAP` (SHapley Additive exPlanations) package of Python. This method is based on the concept of Shapley values from cooperative game theory and is one of the most used to explain how a ML model works. SHAP values assign each feature an importance value for a particular prediction in order to understand how the features of a data set are related to the outputs. The SHAP value of a feature is calculated by comparing the model's prediction for a given input with the prediction that would be made if that feature was not present. This technique can be particularly useful in cases where the model is complex and difficult to interpret, such as in the case of neural networks. The results of this analysis are shown in Figure 3. We can observe that b is the most influential parameter (~ 0.40), followed by the mass of the third body m_3 (~ 0.37). Other features such as velocity v (~ 0.32) and the semi-major axis of the binary a (~ 0.30) are also major players in determining the model, while the others have lower influence.

Parameters	**Values**	
Architecture	Input-layer: 13 Dense-layer: 40 Output-layer: 3	Structure of the NN
Activation function	Dense-layer: ReLu Output-Layer: Softmax	Mathematical function applied to the output of each neuron
Optimizer	Adam	Algorithm to update the weights
Learning rate factor	0.0005	Modulation of the learning rate of the optimizer
Validation splitting	0.20	Fraction of training set used as validation set
Epochs	300	Number of times the entire data set is processed during training
Batch size	100	Number of training examples used in backpropagation
Class weights	{0:2.7, 1:3, 2:3.9}	Weights associated with the classes

Table 1. Hyperparameters used in the FCNN.

2.2 Balanced Random Forest (BRF)

As an alternative method, we decided to use a modified Random Forest (RF), more able to learn how to classify the underrepresented classes. The function used is `BalancedRandomForestClassifier` from the Python library `imbalanced-learn` and it takes as input the usual parameters of the RF plus a set of weights associated with the classes, in order to bias the calculations of the accuracy in favour of the minority class. Furthermore, it allows choosing the sampling strategy, which in this case was set to "not majority", meaning that the algorithm resamples all classes apart from the majority in order to obtain a similar number of samples in each class.

With the `GridSearchCV` command, we performed an exhaustive search over specified parameter values for the BRF estimator. The best model is found with the combination of the parameters reported in Table 2.2, which returns a training accuracy of 76.5%, a validation accuracy of 72.2% and a test accuracy of 71.6%.

It is possible to visualise how the model performs in classifying each class by building a confusion matrix both for the test and the training sets. The results can be observed in Figure 4. Like for the FCNN, the values of the weights in Table 2.2 were chosen by analysing the confusion matrix with different sets of weights: other possibilities, such as the "balanced" function provided by `scikit-learn`, kept the same overall accuracy but then failed to effectively classify the singular classes. The final set of weights is the one that keeps homogeneous results in the confusion matrices for each class without worsening the total accuracy.

Finally, we checked the influence of each feature in the classification process. The result is shown in Figure 5. We can observe that m_3 is the most influential parameter (~ 24%), followed by the semi-major axis of the binary a (~ 23%). Other features such as velocity v (~ 9.8%) and impact parameter b (~ 8.8%) almost reach 10%, while the others have lower influence.

Parameters	Values	
Number of estimators	100	Number of trees in the forest
Max features	'sqrt'	Number of features to consider when looking for the best split
Bootstrap	True	Whether to use bootstrap samples
Sampling strategy	'not_majority'	Sampling information to sample the data set
Class weights	{0:0.75, 1:2, 2:2}	Weights associated with the classes
Criterion	'gini'	Function to measure the quality of a split
Max leaf nodes	450	Maximum number of leaves in a tree
Cross validation	3	Number of folds to perform cross validation

Table 2. Hyperparameters used in the BRF.

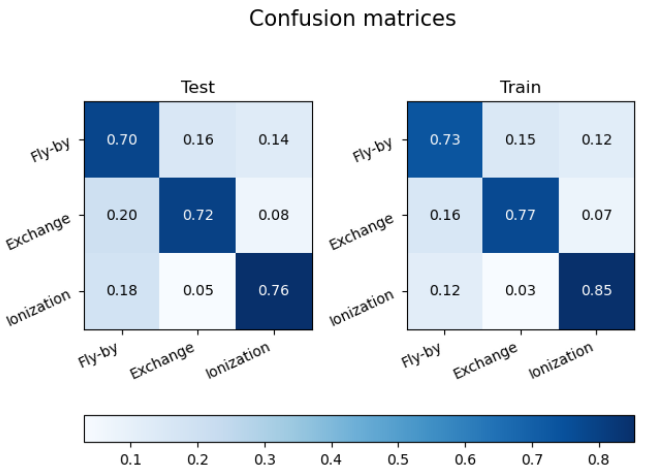

Figure 4. Confusion matrix for test and training set derived from the BRF model. The structure is analogous to 2.

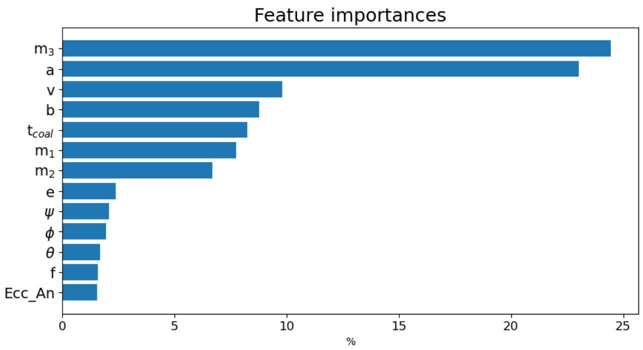

Figure 5. Feature importance in the BRF classifier.

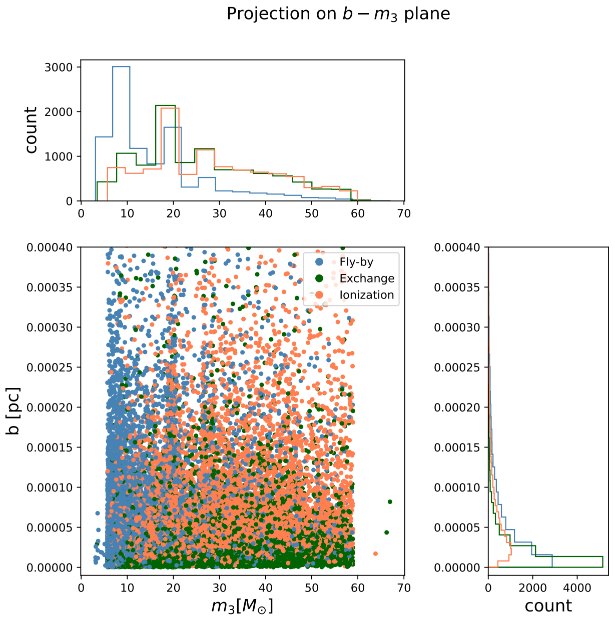

Figure 6. Projection of data points on $b - m_3$ plane. The two histograms show the distribution of points for the two features. The distribution of b in the exchange class is strongly peaked for lower values and only slightly spread, for the fly-by it is peaked for lower values as well but more spread, and for the ionization, it is slightly peaked while being widely spread across all values. About the intruder mass, the distribution shows a striking difference between fly-bys and the other outcomes: we see a very strong peak at lower masses ($5 - 10M_\odot$) for fly-bys, whereas exchanges and ionizations tend to peak at intermediate masses ($\sim 20M_\odot$).

3 RESULTS AND CONCLUSIONS

In our work, we analyzed the binary-single encounters using two different machine learning algorithms: a balanced random forest and a fully connected neural network. We split the 100000 data set into a training set of 40000 and a test set of 60000 data points. We then reduced the number of classes from 4 to 3 by merging the two different exchange encounters into a single "exchange" class to simplify the problem. We optimized the two algorithms to account for the unbalance of the set of data coming from the simulation by using weighting factors to counterbalance the intrinsic bias of the data set. We obtained ~ 71.6% accuracy with the BRF and ~ 73.5% with the FCNN on the test data set. Plotting the confusion matrices, we can see that the algorithm is capable of classifying with similar accuracy all the classes, so we can state that we were able to mitigate the issue of unbalanced data.

Despite this, our classifiers provide good but not optimal results for the problem. We can say that obtaining higher accuracies is hardly possible: this is due to the intrinsic randomness of the problem. The single-binary encounter is not entirely predictable with the present knowledge, there is indeed an intrinsic degree of chaos in the physical situation, which is reflected in the uncertainty of the classification. Furthermore, we have to consider that this problem is not analytically solvable, and as such, we cannot predict with 100% accuracy the result of such an interaction.

From the feature importance of the two models, we observe that

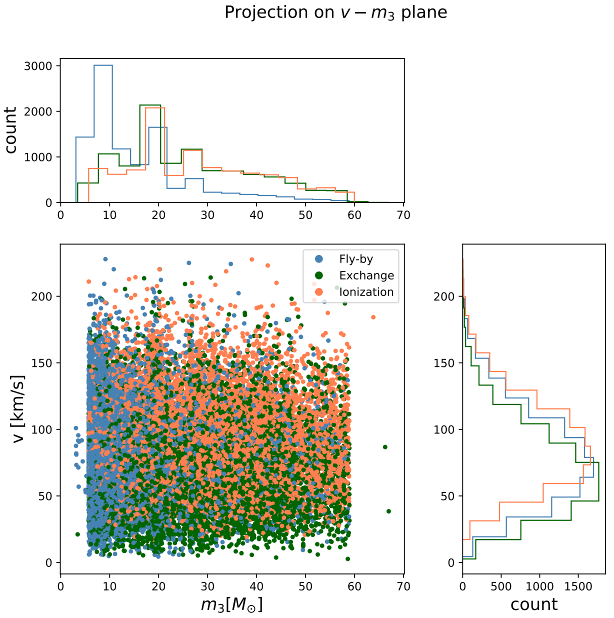

Figure 7. Projection of data points on $v - m_3$ plane. The two histograms show the distribution of points for the two features. The distributions of v are quite similar, they differ for a slight shift in the peak position. The exchange class peaks at lower velocities ($\sim 60km/s$), fly-by in the intermediate range, and ionization at higher values ($\sim 80 - 90km/s$).

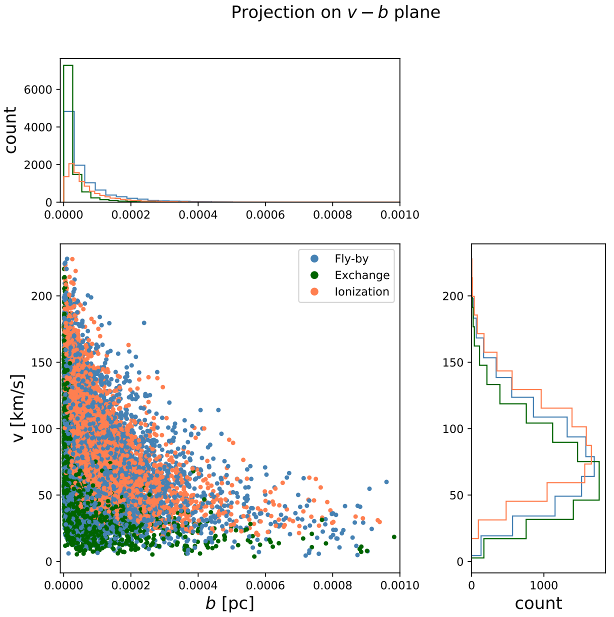

Figure 8. Projection of data points on $v - b$ plane. The two histograms show the distribution of points for the two features. A weak anti-correlation between b and v can be observed here.

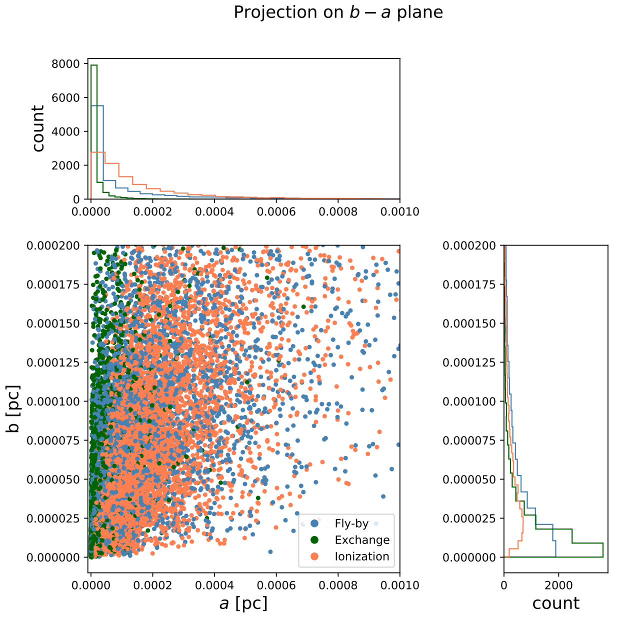

Figure 9. Projection of data points on $b - a$ plane. The two histograms show the distribution of points for the two features.

four main parameters emerge as major players in determining the outcome of the interaction. These parameters are a, the semi-major axis of the binary, b, the impact parameter, m_3, the mass of the third body, and its velocity v. From the plots of 30000 data points set as examples, we analyzed the correlation between these features, which are of little significance and do not simplify the classification task, and their distributions, which instead provide insight into the classifier reasoning. From the figures 6, 7, 8, and 9 we see that the distributions of these parameters differ between each other in a often significant way for the three outcomes. This phenomenon explains, at least partially, why the classifiers have chosen these as most influential quantities. With this analysis, we can state that the intruder plays an important role in the physical scenario.

Additional work in the future can be done by optimizing and improving the models. This can increase the accuracy, but we suppose that the maximum obtainable value is not going to be much higher than ours. Still, constraining the maximum accuracy is of high interest because it represents an indirect method to estimate the intrinsic chaotic randomness of the problem (Gilpin 2021). Furthermore, an improved model that could classify correctly all four outcomes instead of merging two of them would provide a more complete view.

DATA AVAILABILITY

The input file and the scripts are available at the following link: https://drive.google.com/drive/folders/1RhYBKBbvL-OL_9m8aOSMNZxkP-GyzSyU?hl=it

REFERENCES

Boekholt T. C. N., Portegies Zwart S. F., Valtonen M., 2020, MNRAS, 493, 3932

Boekholt T. C. N., Moerman A., Portegies Zwart S. F., 2021, Phys. Rev. D, 104, 083020

Breen P. G., Foley C. N., Boekholt T., Portegies Zwart S., 2020, MNRAS, 494, 2465

Dall'Amico M., Mapelli M., Torniamenti S., Arca Sedda M., 2023, arXiv e-prints, p. arXiv:2303.07421

Gilpin W., 2021, in Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2).

This paper has been typeset from a TEX/LATEX file prepared by the author.