Journal of Physics G: Nuclear and Particle Physics

GUIDE • OPEN ACCESS

Trees and forests in nuclear physics

To cite this article: M Carnini and A Pastore 2020 *J. Phys. G: Nucl. Part. Phys.* **47** 082001

Credit: ESO/M. Kommesser
AMERICAN ASTRONOMICAL SOCIETY
IOP | ebooks™
Your first choice for astronomy, astrophysics, solar physics, and planetary science ebooks.
Start exploring the collection–download the first chapter of every title for free.

OPEN ACCESS
IOP Publishing
Journal of Physics G: Nuclear and Particle Physics

J. Phys. G: Nucl. Part. Phys. **47** (2020) 082001 (22pp)
https://doi.org/10.1088/1361-6471/ab92e3

Trees and forests in nuclear physics

M Carnini[1] **and A Pastore**[2,3]

[1] Features Analytics, Rue de Charleroi 2, 1400 Nivelles, Belgium
[2] Department of Physics, University of York, Heslington, York, Y010 5DD, United Kingdom

E-mail: marco.carnini@features-analytics.com and alessandro.pastore@york.ac.uk

Received 24 February 2020, revised 7 May 2020
Accepted for publication 13 May 2020
Published 8 July 2020

Abstract

We present a simple introduction to the decision tree algorithm using some examples from nuclear physics. We show how to improve the accuracy of the classical liquid drop nuclear mass model by performing feature engineering with a decision tree. Finally, we apply the method to the Duflo–Zuker model showing that, despite their simplicity, decision trees are capable of improving the description of nuclear masses using a limited number of free parameters.

Keywords: statistical methods, machine learning, nuclear mass models, binding energy, decision tree

S Supplementary material for this article is available online

(Some figures may appear in colour only in the online journal)

1. Introduction

In recent years, there has been an enormous growth of new statistical tools for data science [1, 2]. Although these methods are extremely powerful to understand complex data and detect novel patterns, they are still rarely adopted by the nuclear physics community. Only a few groups are currently pioneering the applications of these methods to the field. These topics have been recently discussed in a series of workshops on information and statistics in nuclear experiment and theory (ISNET). Recent developments in this field are documented in the associated focus issue published in *Journal of Physics G* [3]. The aim of this guide is to illustrate an algorithm used widely in data analysis. Similarly to our previous guide on bootstrap techniques

[3] Author to whom any correspondence should be addressed.

0954-3899/20/082001+22$33.00

[4], we present the decision tree starting from very basic models, then finally apply it to more realistic problems, like improving models for nuclear mass predictions.

Decision trees are already implemented within major experimental collaborations, such as MiniBooNE, to improve the performances of particle detectors [5, 6], but they are not yet widely used in low energy nuclear physics, where they could help to analyse both experimental data [7] and theoretical models.

Following the notation and terminology of Leo Breiman's paper *Statistical Modelling: the Two Cultures* [8], we want to investigate a process f that transforms some input X into an output Y. That is to say, f is a function:

$$f : X \rightarrow Y, \tag{1}$$

where the input X can be quite general, from images to a table of data, while the output Y can be a discrete or continuous set. In the first case we speak of a *classification* problem, in the latter of a *regression* problem.

Rather than focussing on investigating the fine details of the process f with many restrictive assumptions (an approach that is named *data model culture* in [8]), we consider f as a *black box* mapping X to Y and we try to approximate it. That is, we give up trying to investigate all the fine details of f and we focus on finding a *representation* (or approximation) $\tilde{f}$ for f. $\tilde{f}$ is called a model and it is a function with the same domain X of the process f and codomain $\hat{Y}$:

$$\tilde{f} : X \rightarrow \hat{Y}. \tag{2}$$

The process $\tilde{f}$ depends on variables (usually named *features*), parameters (coefficients that can be learned with the algorithm) and *hyper-parameters*, that are set before training the model (and thus are not learned). We will present an extended discussion on how to select the features of the model to improve performances in section 2.3. Another goal for the feature selection process is to reach a representation as parsimonious as possible.

Since 'all models are wrong, but some are useful' [9], it is necessary to introduce a definition of what a good model looks like in order to pick the best one out of a set of possible candidates. Or in other terms, we need to assess how faithfully $\tilde{f}$ represents the process f. Mathematically, the goal for training a model $\tilde{f}$ is to minimise a particular *scoring function*, sometimes improperly called 'a metric'. Without loss of generality, we are considering only the minimisation problem: changing the sign of a scoring function to be maximised reduces the problem to a minimisation task.

For example, a natural choice for the scoring function is the mean squared error (MSE) or variance, that is the difference between the predicted value ($\hat{Y}$) and the observed, experimental data (Y) [10]:

$$\mathrm{MSE}(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2. \tag{3}$$

It is worth noting that this is not the only option and within the machine learning literature we encounter other scoring functions as the logarithmic mean squared error (MSLE):

$$\mathrm{MSLE}(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N} (\log_e(1 + Y_i) - \log_e(1 + \hat{Y}_i))^2, \tag{4}$$

or the median absolute error [10]:

$$\text{MedAE}(Y, \hat{Y}) = \text{median}(|Y_1 - \hat{Y}_1|, \ \ldots, \ |Y_N - \hat{Y}_N|). \tag{5}$$

Different scoring functions correspond to different modelling choices and the importance we assign to specific sub-sets of the database. The use of MedAE would be more appropriate to obtain a model that is robust to outliers: a few poorly described experimental points will not alter significantly the performances. In the current work, we have chosen the mean standard error (or equivalently its square root, RMS) which is the default in most libraries. Given the high accuracy of measurements in nuclear physics, especially for masses as discussed here, we do not need to worry about possible outliers in our data sets and MSE therefore represents a reasonable choice.

Another important aspect in building a model is the decision on the tradeoff required between model performances and *explainability*. That is, the choice between (possibly) better performances with the chosen scoring and easier explanations of the model in plain language. Among the regressors usually considered to be explainable are linear regression and decision trees. However, some recent research allows explaining (although approximately) even the results from algorithms deemed black-box, such as neural networks, or such as gradient boosting in explainable models like linear regression [11] and simple decision trees [12].

In this guide, we chose to illustrate decision trees because they retain explainability, they do not rely on the assumption of linearity nor on the linear independence of the *features* and they are not significantly affected by monotonic transformations (no input data scaling is required, nor monotonic transformations like taking the logarithm or the square of one variable). Also, decision trees are the key elements in building other regressors like Random Forests [13] or Xgboost [14] that usually perform better with regard to scoring.

Last but not least, an important aspect of modelling is the balance between the complexity of the chosen model and the generality of the results. As an analogy, it is useful to consider the problem of approximating N experimental distinct points using a polynomial. A complex polynomial of degree N will be able to describe perfectly the data. Whenever some new data are added, the perfect description will (in general) no longer be true. A correct assessment of the performance of a regressor should be performed on unseen data, i.e. data that were not used during the training.

A common practice to estimate the performance on unseen data is the k-fold cross validation, with $k \in \mathbb{N}$. In essence, the data are permuted and then separated in sets of size k with each subset (fold) roughly of the same cardinality. The model is then trained on $k - 1$ subsets and validated on the subset not used while training. As an extreme case, when $k = N - 1$, all data but one are used for training and the performances are assessed on only one datum. This scheme is called 'leave-one out validation' [15].

In the following sections, we will illustrate the behaviour of decision trees using some nuclear mass models. The article is organised as follows: in section 2, we provide an introduction to what a decision tree is, using very simple examples. In section 3, we introduce the nuclear models to which we will apply the decision trees. The results of our work are presented in section 5 and we illustrate our conclusions in section 5. In the supplementary material (available online at [stacks.iop.org/JPhysG/47/082001/mmedia]), we provide the Python script used to perform the calculations. The script has been structured in the same way as the current guide to facilitate its usage[4].

[4] We provide an HTML version of the material at the web address https://mlnp-code.github.io/mlnp/

2. Decision tree

With a decision tree, the function $f: X \rightarrow Y$ is approximated with a step function with n steps as

$$\tilde{f} = \sum_{i=1}^{n} \alpha_i \mathbb{I}(\Omega_i), \tag{6}$$

with $\Omega_i \subseteq X$, $X \subset \mathbb{R}^d$ where d is the number of features, $\mathbb{I}(\Omega_i)$ is the indicator function:

$$\mathbb{I}(\Omega_i) = \begin{cases} 1 & x \in \Omega_i \\ 0 & x \notin \Omega_i \end{cases} \tag{7}$$

and Ω_i are half-planes in $\mathbb{R}$.

Any measurable function can be approximated in terms of step functions [16], thus the approximation is justified as long as the function f is expected to be measurable. That is, using enough step functions we can approximate any measurable function.

Each step function required to build the model $\tilde{f}$ (the tree) is called a *leaf*, thus the number of leaves of the model corresponds to the number of step functions employed.In order to determine the optimal values for the α_i and Ω_i of equation (6), one should provide a splitting criterion; for example, being an extreme value (maximum or minimum) for a given function $\mathcal{L}$. Here, we decide to minimise the $\mathbb{L}^2$ norm of the difference between f and $\tilde{f}$, that is:

$$\mathcal{L} = \|f - \tilde{f}\|_2. \tag{8}$$

This function should be chosen to approximate the scoring function selected; then determining the extremes of $\mathcal{L}$ guarantees that we have optimised the desired scoring function. Since in this guide we chose as a scoring function the MSE, a natural choice for the splitting criterion is the $\mathbb{L}^2$ norm of equation (8); we will use it through all the following examples.

We are going to focus on the CART algorithm as presented in [1, 17]. Calculating all the possible splits for all the features to get the optimal $\tilde{f}$ as in equation (6) is computationally unfeasible for large data sets. For this reason, a greedy approximation is used for training the decision tree: at every iteration of the algorithm, a local optimal split is selected. This is a heuristic approach and there is no guarantee of converging to the global optimum.

At the first step of the algorithm, all the possible splits for all the features are explored and the best split (that minimises $\mathcal{L}$) is selected. Then, all the data are split between the two leaves (a leaf for each half-plane). Then, for every leaf, the procedure is iterated until a stopping criterion is reached. There are many different stopping criteria: a leaf can not be further split if it contains only one observation or if all the features are constant. However, the training is usually stopped as a modelling choice to avoid poor performance on unseen data (*overfitting*): once a given number of leaves or a maximum depth (that is, a maximum number of splits between the root and any leaf) are reached, the algorithm stops. Alternatively, leaves are not split if they contain fewer than a specified number of observations. In this process, some of the features may have never been used; in this case, they are irrelevant to the model and their absence from the input will make no difference.

In the next subsections, we provide some examples of how a decision tree operates, by showing artificially simple examples of trees with few features and very few leaves that can be easily understood. The more realistic cases will be illustrated in section 3.

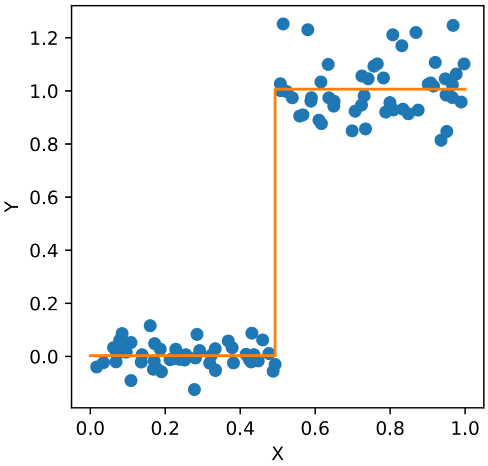

Figure 1. The full dots represent the *original* data set, while the line represents the approximating function $\tilde{f}$. See text for details.

2.1. A single variable example

As a first example, we illustrate the first iteration of a decision tree, splitting the data (a single feature) into only two leaves. We build an example training set X_{tr} defined as the union of two sets X_1 and X_2. X_1 contains random points uniformly distributed in $[0, 0.5]$ and analogously X_2 with points in $(0.5, 1]$. Notice that $X_1 \cap X_2 = \emptyset$. In this case, there is only one feature, so $d = 1$. In this example as well as in the following one, we will directly code the necessary steps to obtain the desired solution; the more advanced reader can skip these two cases to section 2.3 where an existing library is used.

For the images through f of X_1 (X_2), here named Y_1 (Y_2), we use a Gaussian distribution with mean 0 (1) and a variance of 0.05 (0.1). The training set is illustrated in figure 1. All figures presented in this article have been realised with the Python [18] library matplotlib [19].

By construction, the data set can be fully described using a decision tree with only two leaves, that is:

$$\tilde{f} = \alpha_0 \mathbb{I}(\Omega_1) + \alpha_1 \mathbb{I}(\Omega_2). \tag{9}$$

By visually inspecting the data shown in figure 1, we notice that the current data belong to two groups and a single split along the x axis will be enough to describe them. In more advanced examples, the number of leaves will be selected algorithmically. To train a decision tree means to determine the function given in equation (9), in such a way that

$$\mathcal{L} = \|f - \alpha_0 \mathbb{I}(\Omega_1) - \alpha_1 \mathbb{I}(\Omega_2)\|_2, \tag{10}$$

is minimal. $\mathcal{L}$ is equivalent to equation (3) apart from a global scaling factor. The sets Ω_1 and Ω_2 are defined as

$$\Omega_1 = \{x | x \leqslant x_*\},$$
$$\Omega_2 = \{x | x > x_*\}.$$

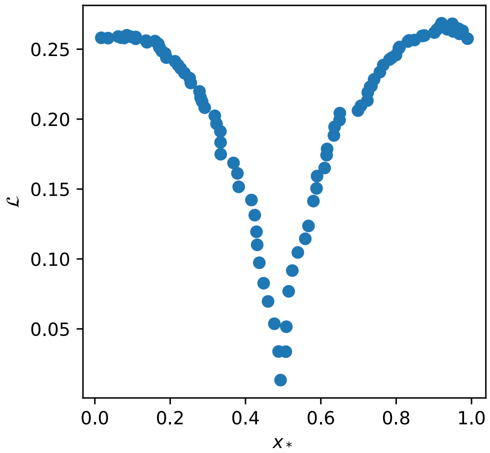

Figure 2. Evolution of the $\mathcal{L}$ norm defined in equation (10) as a function of the splitting point x^*. See text for details.

It is easy to prove that the constant which best approximates (in terms of $\mathbb{L}^2$ norm) a set of values is the average of the values, $\bar{x}$.

The optimal value for x_* (0.5) is obvious when the generating process for the data is known, but how do we determine it when f is not known? The answer is reported in figure 2, where we plot $\mathcal{L}$ as a function of x^*. The optimal value x^* is the one that minimises $\mathcal{L}$.

For this particular case we obtained $x_* = 0.493$, $\alpha_0 = 0.003$ and $\alpha_1 = 1.007$. More details for reproducing the results are provided in the supplementary material. Thus the decision tree reads:

$$\tilde{f} = \begin{cases} 0.003 & \text{if } x \leqslant 0.493 \\ 1.007 & \text{if } x > 0.493. \end{cases} \tag{11}$$

Following these simple steps, we have built a model $\tilde{f}$ that is able to provide a reasonable description of the main structure of the data, i.e. we recognise that the data are separated in two groups. This is represented by the solid line in figure 1. We say that this tree has two *leaves* since we have separated the data into two subgroups. Notice that the MSE is not exactly equal to zero since there was some noise in the generated data.

2.2. A two variables example

We now consider a slightly more complex problem with two variables x_1, x_2. The aim of this example is to familiarise with the concept of multiple splits to treat a complex problem via simple operations. As in the previous case, we will apply the basic steps to explicitly build a decision tree. We generate a new set of data points as:

$$X = \{(x_1, x_2) | x_1 \sim \mathcal{U}(0,1), x_2 \sim \mathcal{U}(0,1)\}, \tag{12}$$

with response:

$$Y = \begin{cases} 10.0 & \{(x_1, x_2) \in X | x_1 \leqslant 0.5\}, \\ 1.0 & \{(x_1, x_2) \in X | x_1 > 0.5, x_2 \leqslant 0.5\}, \\ 0.0 & \{(x_1, x_2) \in X | x_1 > 0.5, x_2 > 0.5\}. \end{cases} \tag{13}$$

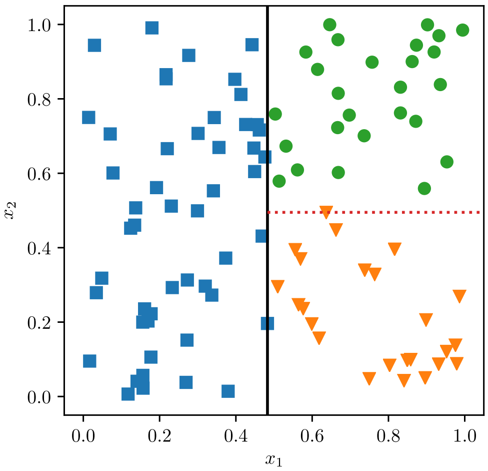

Figure 3. Graphical representation of the data set X defined by equation (12). The squares correspond to $Y = 10$, the dots $Y = 1$ and the triangles to $Y = 0$. The solid line corresponds to the first optimal split, the dotted line to the second split performed by the decision tree. See text for details.

Graphically, we represent this data set in figure 3. The data are clearly clustered (by construction) in three regions of the x_1, x_2 plane. The aim of the current example is to illustrate how to perform successive splits to correctly identify these regions.

We apply a two-step procedure: firstly, we separate the data along the x_1 direction. Following the procedure highlighted in the previous example, we create a model in the x_1 direction of the form

$$\tilde{f} = \alpha_0 \mathbb{I}(\Omega_1) + \alpha_1 \mathbb{I}(\Omega_2).$$

We now perform a systematic calculation of the $\mathcal{L}$ norm looking for the value x^* that leads to its minimal value. We refer to the supplementary material for details. We find $x_* = 0.482$ as the value for dividing the plane. By observing the data, we see that there is no gain by adding further splits in this direction. We will come back to this aspect in the following sections. We can further refine the model by adding an additional separation along the x_2 direction. The procedure follows the same steps as before and we find that $x_* = 0.495$. The result is reported in figure 3.

An important quantity for analysing the model and for assessing the importance of its input variables is an estimate of the feature importance. This is particularly relevant for the ensemble methods that rely on decision trees: while with a single decision tree the role of the features is obvious once the tree is visually represented (see for example figure 4), it is unpractical to represent all of the decision trees in a random forest (there may be thousands). Also, a feature may participate multiple times in different trees, so a definition of importance should take this into account.

Starting from figure 4, we want to assess the importance of the features to the building of the decision tree. We recall here that by *features* we mean the variable of the model. In this particular example, we have considered x_1, x_2 as a natural choice, but one could also consider other combinations: $x_1, x_2, x_1 + x_2, x_1/x_2, \ldots$. We refer the reader to section 2.4 for a more detailed discussion.

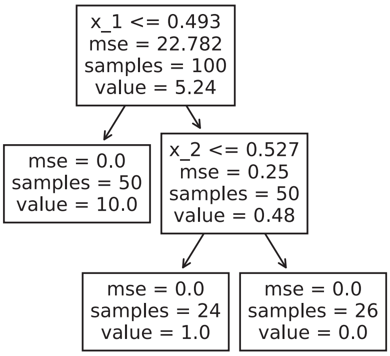

Figure 4. Decision tree with two variables obtained using the `scikit-learn` package [10] for the data set reported in figure 3.

Following reference [10], we calculate the feature importance in the following way: for each split s in the tree, we calculate the weighted impurity decrease as

$$\frac{N_{\mathrm{s}}}{N}\left(I-\frac{N_{\mathrm{s,R}}}{N_{\mathrm{s}}}I_{\mathrm{R}}-\frac{N_{\mathrm{s,L}}}{N_{\mathrm{s}}}I_{\mathrm{L}}\right). \tag{14}$$

N is the total number of observations, N_{s} is the number of observations at the current node, $N_{\mathrm{s,L}}$ is the number of samples in the left leaf and $N_{\mathrm{s,R}}$ is the number of samples in the right leaf. I represents the impurity (in our case, MSE), with the subscripts having the same meaning as before.

By inspecting figure 4, we observe that for the first split there are in the current node (the root of the tree) as many observations as the total, that is $N = N_{\mathrm{s}} = 100$. The initial impurity (MSE) is 22.782, $N_{\mathrm{s,R}} = N_{\mathrm{s,L}} = 50$, right impurity is 0, left impurity 0.25. Thus we obtain

$$\frac{100}{100}\times\left(22.782-\frac{50}{100}\times 0-\frac{50}{100}\times 0.25\right)\simeq 22.657.$$

For the second split, we get:

$$\frac{50}{100}\times\left(0.25-\frac{24}{100}\times 0-\frac{26}{100}\times 0.0\right)\simeq 0.125.$$

Normalising the total weighted variation to 1, we obtain that the column x_1 has importance equal to 99.5% for the model, while x_2 has an importance of 0.5%. If the variables entered in different splits, the relative importance would be summed.

Estimating feature importance is fundamental for improving the quality of the model: by discarding irrelevant features, i.e. features that are not reducing the impurity in the tree, more parsimonious models can be trained. This is especially useful for models involving hundreds (or thousands) of features. For example, anticipating the results of the following section, we see that in figure 10, a simpler model could be obtained using only 6 features instead of 9. Whenever features are generated for improving the model, a critical assessment of their relevance should be performed.

After these examples that were easily implemented with a few lines of code, in the next sections (and for realistic problems, in terms of the number of features and the number of possible leaves) we are going to rely on existing libraries.

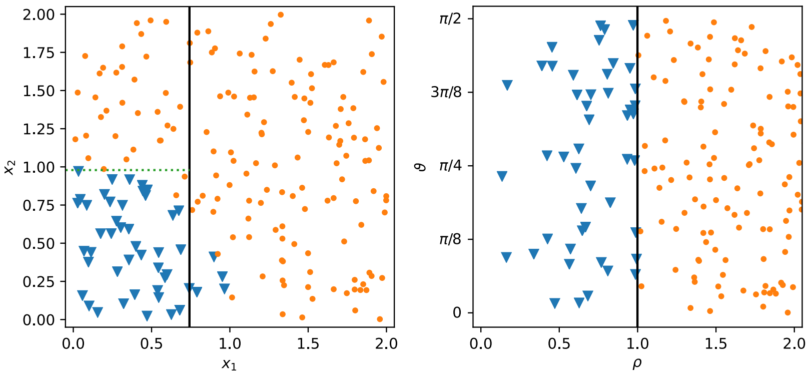

Figure 5. Graphical representation of the data set (X, Y): on the left panel the Cartesian coordinates are used while on the right panel the data are represented in polar form. The triangles correspond to $Y = 1$ and the dots to $Y = 0$. The solid lines represent the cuts done to reproduce the data.

2.3. A two variables example (revisited)

In this section, we make use of the function `DecisionTreeRegressor` from the Python package `scikit-learn` to determine the structure of the tree, using all default hyper-parameters apart from the number of leaves. For the sake of simplicity, we still consider x_1, x_2 as features of the model and we also impose the number of leaves to be three. In more advanced examples, we will let it be a free parameter.

In this case, contrary to the previous example, we do not need to decide if the split along the x_1 direction happens before the one along the x_2 or vice-versa: all possible splits on the available data for all features are explored with the algorithm.

The algorithm used to perform such a split can be represented as in figure 4 using the `scikit-learn` package [10], but an analogous result could have been obtained using R [20], and the libraries `rpart` [21] for model training and `rpart.plot` [20] for visualisation.

The visualization of a `scikit-learn` tree consist of a series of boxes counting basic information: the value of the variable at which the separation takes place, but also the mean value of the data (named *value*) and the impurity (MSE in our example). It also provides information concerning the amount of data grouped in each leaf. In this case, the tree has a total of three leaves. As the MSE is zero (thus, minimal) on each leaf, adding extra splits to the model would not lead to any real gain in the description of the data, but it would only increase the model complexity.

2.4. The importance of feature engineering

In the previous examples, we have approximated the data using simple step functions; although this choice is mathematically justified, the problem is that the approximation may lead to a single observation per leaf, with the result that the generalisation on unseen data may be unsatisfactory.

To overcome the problem and possibly to make the models easier to explain, it is important to explore the data and apply *convenient* transformations on the input variable for the model that may highlight some patterns. We consider as an example the case of a two variable data-set

Table 1. Evolution of MSE with the number of leaves of the tree for the data set shown in figure 5.

MSE	leaves
0.278	2
0.073	3
0.164	4
0.042	5
0.222	6
0	7

obtained as follow:

$$X = \{(x_1, x_2)|x_1 \sim \mathcal{U}(0,1), x_2 \sim \mathcal{U}(0,1)\}, \tag{15}$$

where the response is chosen as

$$Y = \begin{cases} 1.0 & \left\{(x_1, x_2) \in X|x_1^2 + x_2^2 \leqslant 1\right\} \\ 0.0 & \left\{(x_1, x_2) \in X|x_1^2 + x_2^2 > 1\right\}. \end{cases} \tag{16}$$

In the left panel of figure 5, we illustrate the data points using Cartesian coordinates. By using `DecisionTreeRegressor`, we build a series of decision trees as a function of the number of leaves. In table 1, we report the MSE of the tree for various number of leaves.

From the table, we see that the lowest MSE is given by a seven-leaves tree. This tree is quite complex, with leaves containing only 2 or 4 observations. For illustrative purposes let us consider the case with 3 leaves, which corresponds to $MSE = 0.073$. The splits are performed along the x_1 direction at $x_* = 0.742$ and along the x_2 direction at $x_* = 0.979$. The result is reported using solid and dashed lines in the left panel of figure 5.

By inspecting the data, we realise that by applying a unitary transformation from Cartesian to polar coordinates:

$$x_1 = \rho \ \sin \ \theta, \tag{17}$$

$$x_2 = \rho \ \cos \ \theta, \tag{18}$$

we can highlight a very specific structure in the data. The result of such a transformation is illustrated in the right panel of figure 5. Using these new variables in the model obviously helps the performances: using a single split, i.e. a tree with two leaves, we obtain a total MSE of zero using fewer leaves and with features that are easier to understand. Only one feature is relevant (ρ), so while the possible splits on the other features are explored with the algorithm, they are irrelevant and do not play any role in the model. Notice that by construction, there is only radial dependence and no dependency on the phase. Thus we obtain a more parsimonious model by applying features engineering.

2.5. Random forests

We conclude this section on decision trees by introducing the concept of a random forest (RF). following [1, 13, 22], we define an RF as an ensemble of decision trees. The first step for training an RF is to use bagging, also named bootstrap aggregating [23, 24]. Given a training set X_{tr}, a uniform sampling with replacement is performed, obtaining two data sets: one containing the sampled data (with repetitions), X_1 and one containing the data that were never sampled, named out-of-the-box (OBB), X_2. X_2 contains roughly one-third of the initial observations.

In fact, given a number N of observation, assuming no repetitions in the data, the probability for a datum of *not* being extracted at the ith draw is simply $p_i = 1 - \frac{1}{N}$. Thus the probability of not being extracted after N draws, having replacement and assuming all the draws to be independent, is:

$$p = \prod_{i=1}^{M} p_i = \left(1 - \frac{1}{N}\right)^N \xrightarrow[N \to \infty]{} \frac{1}{\mathrm{e}} \approx \frac{1}{3}. \tag{19}$$

The second step of the RF algorithms consists of selecting a random subset of the features of X_1. The number of features used is an adjustable hyper-parameter of the algorithm; with [10] for example, the user provides the fraction in (0, 1) out of the total number of features that will be used for each tree. A decision tree is then built on X_1 using only the selected features and the performances are estimated on X_2. In [10] the procedure is fully automatic and many trees can be trained in parallel, but the estimation on X_2 is not performed by default (but the option can be activated). Repeating the bagging and the tree training for a number T of trees (another adjustable hyper-parameter) and averaging the response $\hat{Y}$ over the ensemble of predictions, a random forest is obtained.

The bagging and the random selection of features are tools to inject noise in the training data. The noise can be reduced by averaging on the response of each tree and this empirically improves the performance of the regressor [13, 23]. For the theoretical reasons why the performances are better after injecting some noise in the system, the reference is [13].

Intuitively, the trees in the forest should not all provide the same response, otherwise averaging on all the trees would be of no benefit. Consider for example the dataset X of equation (12): if all the trees are trained on the same data with the same input features, then they will all provide the same output and the random forest will be equivalent to a single decision tree.

A reason that made random forests very popular is that they can be trained on data sets with more features than observations without prior feature selection, a characteristic that made then a relevant tool, for example, for gene expression problems in bioinformatics. For more details, see [25].

3. Nuclear mass models

Before applying the decision tree to a more realistic case, we now introduce the nuclear models we are going to study. According to the most recent nuclear mass table [26], more than 2000 nuclei have been observed experimentally. To interpret such a quantity of data, several mass models have been developed over the years [27] with various levels of accuracy. For this guide, we selected two simple ones: the Bethe–Weizsäcker mass formula [28] based on the liquid drop (LD) approximation and the Duflo–Zuker [29] mass model. The reason of this choice is twofold: the models contain quite different physical knowledge about the data, for example, the lack of shell effects in LD case, but they are relatively simple and not CPU intensive, thus giving us the opportunity to focus more on the statistical aspects of the current guide.

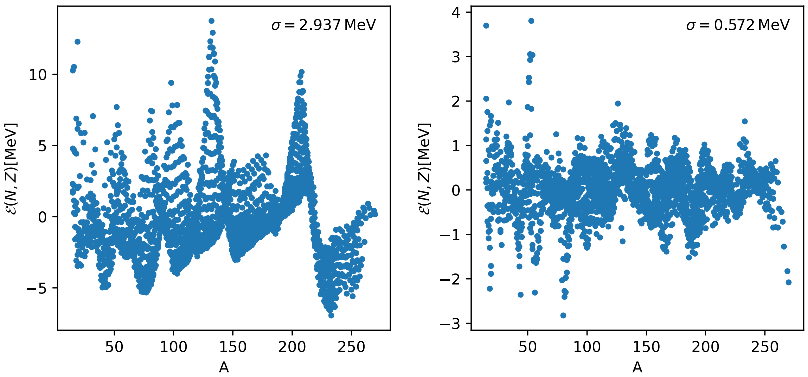

Figure 6. Residuals (expressed in MeV) obtained using the liquid drop model (left panel) and the DZ10 model (right panel).

3.1. Liquid drop

Within the LD model, the binding energy (B) of a nucleus is calculated as a sum of five terms as a function of the total number of neutrons (N) and protons (Z) as

$$B_{\rm th}^{\rm LD}(N,Z) = a_{\rm v}A - a_{\rm s}A^{2/3} - a_{\rm c}\frac{Z(Z-1)}{A^{1/3}} - a_{\rm a}\frac{(N-Z)^2}{A} - \delta\frac{\mathrm{mod}(Z,2)+\mathrm{mod}(N,2)-1}{A^{1/2}}, \tag{20}$$

where $A = N + Z$. The set of optimal parameters have been tuned in reference [4]. These parameters are named volume ($a_{\rm v}$), surface ($a_{\rm s}$), Coulomb ($a_{\rm c}$), asymmetry ($a_{\rm a}$) and pairing (δ) and they refer to specific physical properties of the underlying nuclear system [28].

3.2. Duflo–Zuker

The Duflo–Zuker [29] is a macroscopic mass model based on a generalised LD plus the shell-model monopole Hamiltonian and it is used to obtain the binding energies of nuclei along the whole nuclear chart with quite a remarkable accuracy. The nuclear binding energy for a given nucleus is written as a sum of ten terms as

$$B_{\rm th}^{\rm DZ10} = a_1V_{\rm C} + a_2(M+S) - a_3\frac{M}{\rho} - a_4V_T + a_5V_{TS} + a_6s_3 - a_7\frac{s_3}{\rho} + a_8s_4 + a_9d_4 + a_{10}V_{\rm P}. \tag{21}$$

We defined $2T = |N-Z|$ and $\rho = A^{1/3}\left[1 - \frac{1}{4}\left(\frac{T}{A}\right)^2\right]^2$. The ten different contributions can be grouped into two categories: in the first one we find terms similar to the LD model as Coulomb ($V_{\rm C}$), symmetry energy (V_T, V_{TS}) and pairing $V_{\rm P}$. The other parameters originate from the averaging of shell-model Hamiltonian. See [30] for more details. The model described in equation (21) is usually referred as DZ10 and its parameters have been recently tuned in [31]. Within the literature, it is also possible to find other versions with extra parameters [32], but we will not consider them here for the sake of simplicity.

In figure 6, we illustrate the behaviour of the residuals $\mathcal{E}(N,Z)$ obtained with the two mass models i.e. the difference between the empirical data and the models predictions. We assume

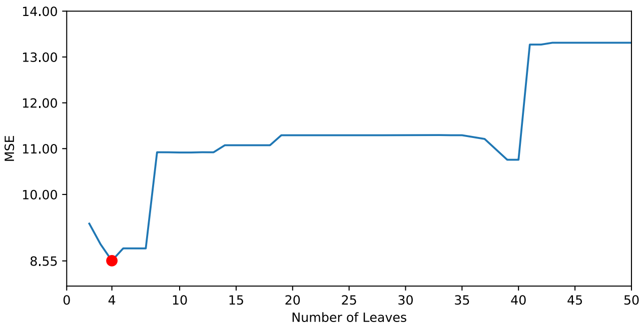

Figure 7. Evolution of the MSE as a function of the number of leaves. The dot corresponds to the absolute minimum. See text for details.

that nuclear data [26] have negligible experimental error compared to the model and we discard all data having an uncertainty larger than 100 keV. This is a reasonable assumption to be made since the typical discrepancy between models and data is usually one or two orders of magnitude larger than the experimental errors [27]. See discussion in [4] for more details.

In each panel of figure 6, we also provide the root mean square (RMS) deviation σ. We thus see that DZ10 is roughly one order of magnitude more accurate in reproducing data than the simple LD. The detailed analysis of these residuals has been already performed in [4, 31] showing that they do not have the form of a simple white noise, but they contain a degree of correlation.

4. Results

We apply the decision tree to the case of nuclear data. Using the same notation adopted in the previous examples, the input X is now a matrix with 3 columns N, Z, A while the response $\hat{Y}$ is the residual.

As specified before, the goal is to minimise the RMS on unseen data or, in other words, learning without overfitting. While it appears obvious that a tree with only one leaf, which means replacing all the values of Y with the average $\overline{Y}$, or with as many leaves as there are observations are not very useful, determining the optimal value for the number of leaves is not straightforward. The approach is empirical: experimenting with a reasonable set of values for the number of leaves and pick the best results according to the cross-validation.

With only one adjustable hyper-parameter like the maximum number of leaves, exploring the parameter space is straightforward: all the possible values are tested, with a cost of M cross-validated models, where M is the number of possible values for the number of leaves. In our example, exploring trees with a maximum number of leaves between 2 and 500 implies cross-validating for $M = 499$ models.

On the other hand, with regressors with many adjustable parameters, as for example Xgboost [14], exploring the hyper-parameter space is more challenging. For example, with 10 hyper-parameters, exploring M values for each of them means exploring a grid with 10^M points. In this case, it is better to use dedicated libraries [33].

As a first application, we train a simple decision tree over the residuals of the LD model as shown in the left panel of figure 6. In figure 7, we illustrate the evolution of the MSE as a

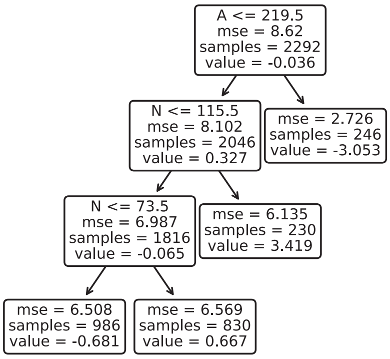

Figure 8. Decision tree for LD model using only three features N, Z, A. See text for details.

function of the number of leaves. For sake of clarity we truncated the figure to 50 leaves, the full plot can be found in the supplementary material.

From figure 7, we notice that the optimal number of leaves is four. The structure of the tree is reported in figure 8. By inspecting the splits of the data, we notice that the main feature of the data is associated with the neutron number N. The tree splits the nuclei in super-heavy ($A > 219$) and non-super-heavy. Then it further splits into very neutron-rich and not. Finally, the tree separates out the remaining nuclei into two groups: light and heavy.

Having the optimal tree, we now translate it into a simple code. Here we use Fortran, but any other computer language can be used with no difficulty.

```
! Example of decision tree for the LD model with 2 features

!...
!input: N,Z (neutron/protons) / integers
!output: BEtree (correction to binding energy) /real
!...
  A=N+Z
  if(A < 219.5)then
     if(N < 115.5)then
        if(N < 73.5) then
           BEtree=-0.681
        else
           BEtree=0.667
        endif
     else
        BEtree=3.419
     endif
  else
     BEtree=-3.053
  endif
...
```

Using the previous code, we now calculate the nuclear binding energies as

$$\mathcal{B}_{\rm th} = B_{\rm th}^{\rm LD} + B_{\rm tree}, \tag{22}$$

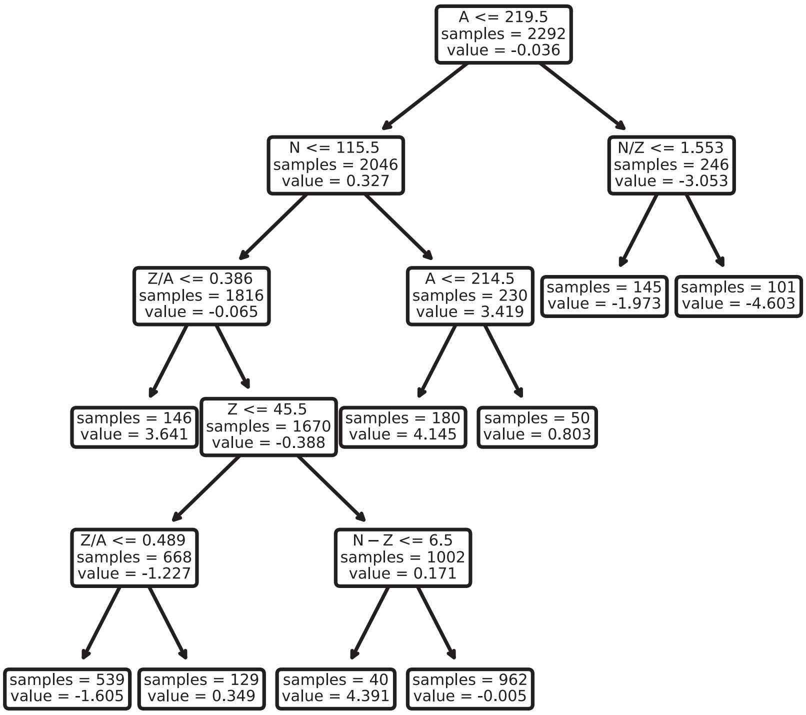

Figure 9. Improved decision tree for LD model using feature engineering. For the sake of clarity and readability, the impurity (MSE) was omitted.

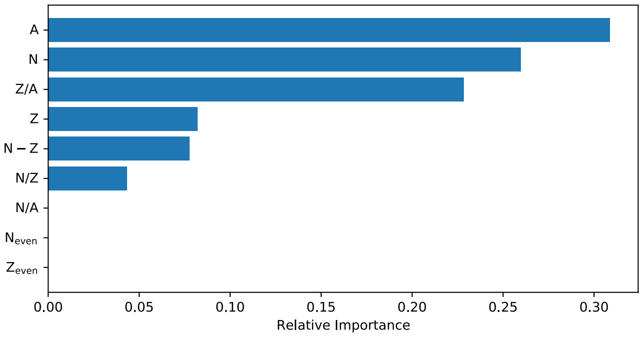

Figure 10. Relative importance of the features (reduction in impurity normalised) in the liquid drop model. The features N/A, N_{even} and Z_{even} (equal to 1 if N or Z is even and 0 otherwise) were not used in the model and as a consequence they have zero importance.

where B_{tree} represents the binding energy calculated with the decision tree. By comparing the predicted masses obtained with equation (22) with the experimental ones, we obtain an RMS of $\sigma_{\text{C,tr}} = 2.467$ MeV. This is what is called *training error*, which is the RMS between the response and the predictions of the model trained on all data. A more conservative estimation

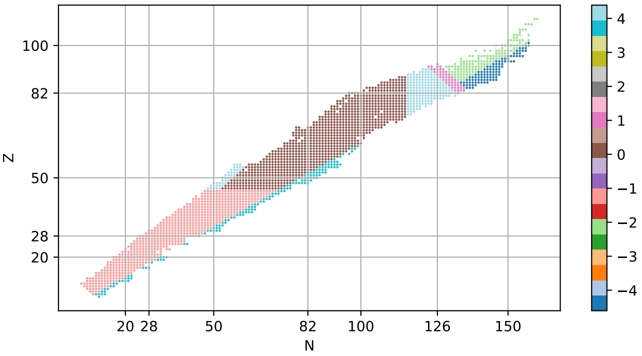

Figure 11. Graphical representation of the splits done by the decision tree illustrated in figure 9 on the Segré chart of nuclei. The various zones correspond to the energy corrections expressed in MeV derived from the decision tree to the LD model as a function of N, Z.

that should be preferred is the validation error on unseen data, i.e the RMS estimated on data that were not used during the training. In this case, $\sigma_{\mathrm{C,val}} = 2.925$ MeV.

It is possible to further improve on this result, by using feature engineering as discussed previously. To this respect, we provide some additional information to the tree: A, $N-Z$, N/Z, Z/A, $Z/N\ldots$. The full list of features can be seen from figure 10. By inspecting equation (20), we observe that these features are already used to build the LD model and as such we help the decision tree to identify new patterns in the data. It is worth noting here that other features may be used instead, but a monotonic transformation of existing features (like $A^{1/3}$ if we are using A) will provide little to no performance improvement. See for example [34] for an empirical discussion of the topic. Identifying patterns into the data is of great help since it may lead (in complex cases) to better solutions.

In figure 9, we report the structure of this new tree. The optimal number of leaves is 9. By implementing this tree into a simple numerical code, as done previously and applying it to the LD residuals we obtain a slight improvement. The total RMS over the entire nuclear chart now falls to $\sigma_{\mathrm{C,tr}} = 2.069$ MeV (on unseen data, $\sigma_{\mathrm{C,val}} = 2.881$ MeV).

Although the decision tree performs less well (in terms of RMS) than a more complex neural network [35], we can still use it to identify possible trends in the data set. By inspecting figure 10, we observe that not all the 9 features have been used to build the code. In figure 10, we illustrate the relative importance of the features of the LD model, calculated using equation (14). We see that the proton fraction Z/A is more important than the individual number of neutrons and protons. It is interesting to note that the decision tree is not affected by even/odd nuclei. This may imply that either the simple pairing term in equation (20) is enough to grasp the odd–even staggering, or the granularity required to the tree is too high, leading to a number of leaves comparable with the number of data points, or other features can surrogate the odd–even features. We also observe that in this tree the total number of nucleons A and the proton fraction Z/A are as important or more than the number of neutrons N. This clearly explains why the performances of this new tree have improved compared to the one given in figure 8. A detailed understanding of the trend in the data would require a more in-depth analysis and so we leave it for future investigations.

In figure 11, we present a graphical illustration of the energy corrections found by the decision tree for the different nuclei along the Segré chart. This figure is an alternative way to represent the various leaves shown in figure 9. We observe that we have 6 major splits

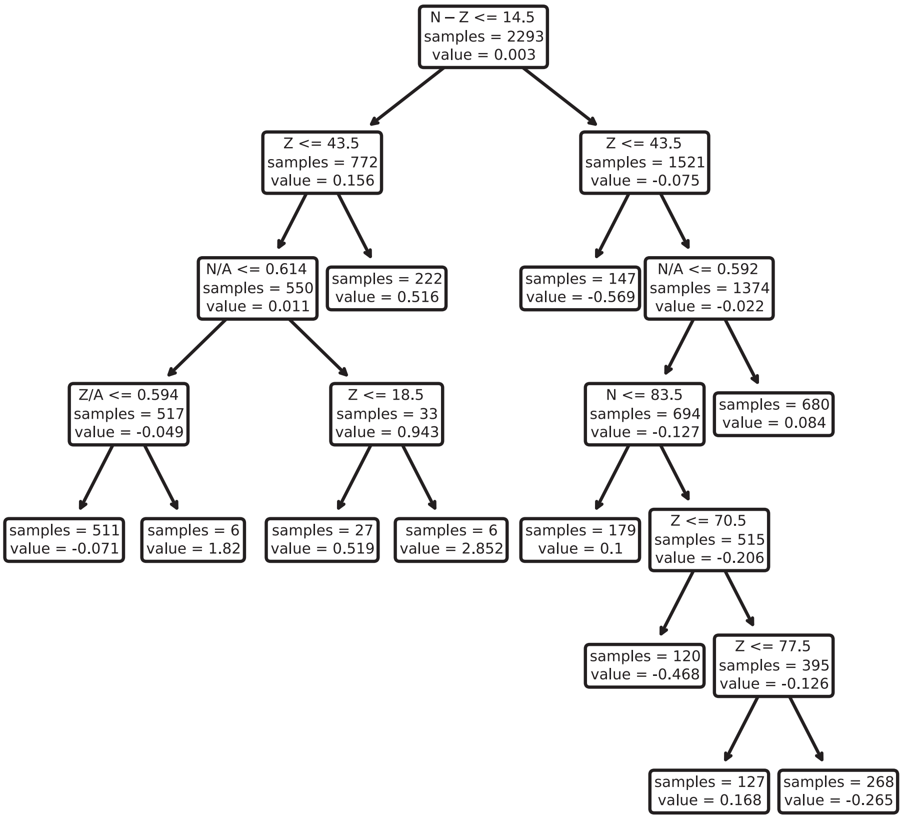

Figure 12. Decision tree for DZ10 model given in equation (21). For the sake of clarity and readability, the impurity (MSE) was omitted.

along the valley of stability where we find light, medium-heavy and heavy nuclei. The latter are then still separated into 4 smaller groups. The other cuts occur along the region of proton-rich and neutron-rich, thus the edges of the chart. From this general overview, we may conclude that the residuals of the LD model are quite homogeneous (only two separations) along the valley of stability up to medium-heavy nuclei. Outside this range, the number of splits increase since the tree identifies a larger variation in the data. This may imply some missing physics in the model (choice of features) for these particular regions of the chart.

Having seen how the decision tree works for a schematic model as LD, we now apply it to the more sophisticated DZ10. We adopt the same features as shown in figure 10 to obtain the best performances. Since the structure of the residuals is different there is no *a priori* reason to use such features, but for the sake of simplicity of the current guide, we keep them the same.

For the DZ10 model, the optimal tree has now 11 leaves and it is illustrated in figure 12. As discussed previously, the tree can be easily translated into a small numerical code using a simple structure.

In figure 13, we illustrate the importance of the features used to build such a tree. It is interesting to observe that the most important feature is the charge dependence and the isovector dependence of the model ($N - Z$). By comparing with figure 10, we observe that the relative

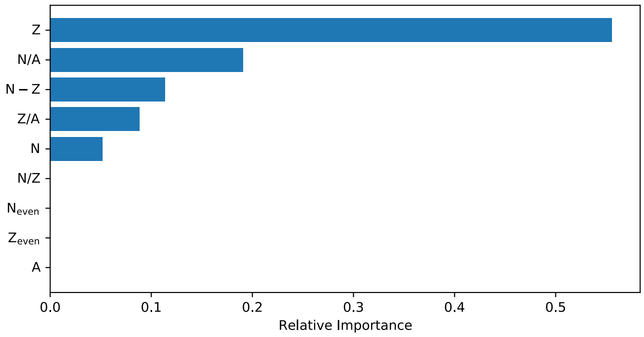

Figure 13. Relative importance of the features (reduction in impurity normalised) in the DZ10 model. As before, the features N/A, N_{even} and Z_{even} (defined as in the caption of figure 10) were not used in the model, has zero importance in the model and can thus be discarded.

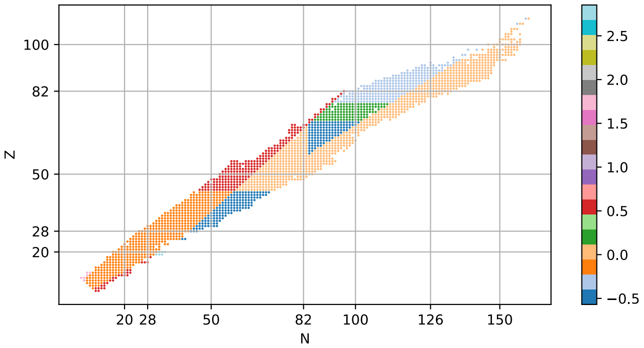

Figure 14. Graphical representation of the splits done by the decision tree illustrated in figure 13 on the Segré chart of nuclei. Corrections to the DZ10 model as a function of N, Z.

importance of the features strongly depends on the model. In particular, four features of nine turned out not to be relevant during the optimisation of the tree. We could further simplify the tree by reducing the features used or exploring new ones. This investigation goes beyond the scope of the present guide since we are only interested in illustrating how the algorithm works.

We implement such a tree within a simple Fortran code. See appendix A for details. With such a code, we calculate the new binding energies as

$$\mathcal{B}_{\mathrm{th}} = B_{\mathrm{th}}^{\mathrm{DZ10}} + B_{\mathrm{tree}}. \tag{23}$$

The global RMS drops to $\sigma_{\mathrm{C,tr}} = 0.471$ MeV ($\sigma_{\mathrm{C,val}} = 0.569$ MeV). The improvement on the binding energies is not as good as the one obtained [31] using a more complex neural network, but the model we produced is far simpler. It is worth noticing that the final model given in equation (23) is fully specified by 43 parameters (the 10 original parameters from the DZ10 models, the 7 pairs describing the variable and the value to split on, the 9 values of the response

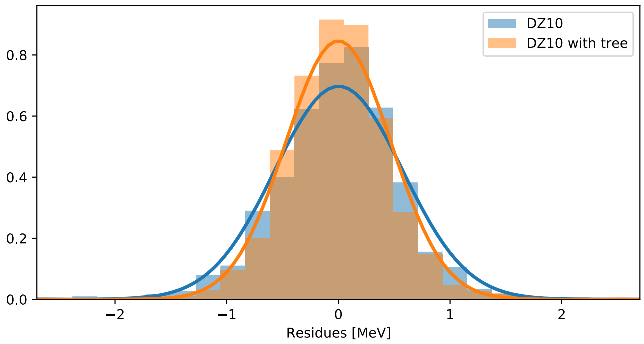

Figure 15. Normalised histograms comparison between residues with and without the tree correction.

Table 2. Here, σ_M is the original model RMS, $\sigma_{\mathrm{C,tr}}$ is the RMS once the corrections are added $\sigma_{\mathrm{C,val}}$ the RMS on unseen data (with corrections).

Model	σ_M	$\sigma_{\mathrm{C,tr}}$	$\sigma_{\mathrm{C,val}}$	Improvement
Liquid drop	2.936	2.467	2.925	16.0%
Liquid drop with features	2.936	2.070	2.881	29.5%
DZ10 with features	0.572	0.471	0.569	17.6%

on each leaf). This number is comparable with most nuclear mass models [36–39], which perform similarly to equation (23).

As done previously for the LD model, we represent in figure 14 the splits of the tree given in figure 12. We see that the most important cuts take place along Z. This was also the most important feature of the model as shown in figure 13. Interestingly, using the decision tree we have identified a large area in the residuals corresponding to the medium-heavy neutron-rich nuclei for which the correction is very small. On the contrary, the same mass range, but on the proton-rich side, requires a much more significant energy correction. This may be a symptom of poor treatment of the isovector channel in the model.

In figure 15, we represent the comparison between the original residuals obtained with DZ10 model and the improved one using the decision tree. The histogram has been normalised. We see that the new residuals are now more clustered around the mean value, although we see that there are still some heavy tails that we have not been able to eliminate. We have checked the normality of the residual using the standard Kolmogorov Smirnov test [40] and we can say that the residuals are not normally distributed with a 95% confidence, thus showing there is still some signal left in the data that we have not been able to grasp.

We conclude this section by summarising the impact of decision trees on the residuals of the various mass models and different features used in the calculations. The results are reported in table 2. We observe that using feature engineering, we have been able to reduce the RMS of the LD model by ≈30%. Adopting the same features for the DZ10 model, we have improved the global RMS by ≈ 18%.

It is worth noting that the numbers given in table 2 are strictly dependent on the features we used to build the trees. Different choices would lead to different numbers.

5. Conclusion

In this guide, we have illustrated a well-known decision tree algorithm by providing very simple and intuitive examples. We have also shown the importance of analysing the data to improve the performances of the method.

We have applied the decision tree to the case of two well known nuclear mass models: liquid drop and Duflo–Zuker. In both cases, using a small number of leaves (9 and 11 respectively), we have been able to improve the global RMS of the models by 29.5% and 17.6%, respectively. More consistent improvements have been obtained in the literature using neural networks [31, 41, 42], but using a larger set of adjustable parameters.

We have also illustrated how to represent graphically the decision tree to better highlight the regions of the splits: this allows us to identify possible patterns in the data-set and eventually use them to improve the original model. By analysing the importance of the features, it is then possible to identify possible missing structure in the model.

Finally, we have also illustrated how to translate the decision tree into a simple numerical code that could be easily added to existing ones to calculate nuclear masses.

Acknowledgements

We thank M Shelley for helping us with this guide. We also thank the two anonymous referees for their precious comments. This work has been supported by STFC Grant No. ST/P003885/1.

Appendix A. Decision tree: pseudo-code

For completeness, we provide here a possible translation of the decision tree into a Fortran code.

```
!
! Example of decision tree for DZ10 model
!

!input: N,Z (neutron/protons) / integers
!output: BEtree (correction to binding energy) /real

...
if(N-Z< 14.5)then
   if(Z < 43.5)then
      if(N/A <= 0.614)then
         if(Z/A <= 0.594)then
            BEtree=-0.071
         else
            BEtree=1.82
         endif
      else
         if(Z < 18.5)then
            BEtree=0.519
         else
            BEtree=2.852
         endif
      endif
      else
         BEtree=0.516
      endif
   else
      if(Z < 43.5)then
         BEtree=-0.569
      else
         if(N/A <= 0.592)then
            if(N < 83.5)then
               BEtree=0.1
            else
               if(Z < 70.5)then
                  BEtree=-0.468
               else
                  if(Z < 77.5)then
                     BEtree=0.168
                  else
                     BEtree=-0.265
                  endif
               endif
            endif
         else
            BEtree=0.084
         endif
      endif
   endif
```

Notice that a decision tree is formed by a simple sequence of conditional statements and the example given here can be easily ported to any other used computer language.

ORCID iDs

A Pastore https://orcid.org/0000-0003-3354-6432

References

[1] Friedman J, Hastie T and Tibshirani R 2009 *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (New York: Springer)

[2] Mehta P, Bukov M, Wang C-H, Day A G, Richardson C, Fisher C K and Schwab D J 2019 *Phys. Rep.* **810** 1–124

[3] Ireland D G and Nazarewicz W 2015 *J. Phys. G: Nucl. Part. Phys.* **42** 030301

[4] Pastore A 2019 *J. Phys. G: Nucl. Part. Phys.* **46** 052001

[5] Roe B P, Yang H-J, Zhu J, Liu Y, Stancu I and McGregor G 2005 *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip.* **543** 577

[6] Yang H-J, Roe B P and Zhu J 2005 *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip.* **555** 370

[7] Bailey S 2017 New analytical techniques for the investigation of alpha clustering in nuclei *PhD Thesis* University of Birmingham

[8] Breiman L 2001 *Stat. Sci.* **16** 199

[9] Box G E 1976 *J. Am. Stat. Assoc.* **71** 791

[10] Pedregosa F *et al* 2011 *J. Mach. Learn. Res.* **12** 2825–30

[11] Ribeiro M T, Singh S and Guestrin C 2016 *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (San Francisco, CA, USA 13–17 August 2016) pp 1135–44

[12] Boz O 2002 *Proc. of the 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining KDD '02* (New York: Association for Computing Machinery) pp 456–61
[13] Breiman L 2001 *Mach. Learn.* **45** 5
[14] Chen T and Guestrin C 2016 *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining KDD 16* (New York: Association for Computing Machinery) pp 785–94
[15] Lachenbruch P A and Mickey M R 1968 *Technometrics* **10** 1
[16] Maderna C and Soardi P M 1985 *Lezioni di Analisi Matematica* (Milan: CLUED)
[17] Breiman L, Friedman J H, Olshen R A and Stone C J 1983 *Classification and Regression Trees* (London: Chapman and Hall)
[18] Van Rossum G and Drake F L Jr 1995 *Python Tutorial* (Amsterdam: Centrum voor Wiskunde en Informatica Amsterdam)
[19] Hunter J D 2007 *Comput. Sci. Eng.* **9** 90
[20] Milborrow S 2019 rpart.plot: plot 'rpart' models: an enhanced version of 'plot.rpart' *r package version 3.0.8* https://CRAN.R-project.org/package=rpart.plot
[21] Therneau T and Atkinson B 2019 rpart: recursive partitioning and regression trees *r package version 4.1-15* https://CRAN.R-project.org/package=rpart
[22] Louppe G 2014 arXiv:1407.7502
[23] Efron B 1979 *Ann. Stat.* **7** 1
[24] Breiman L 1996 *Mach. Learn.* **24** 123
[25] Okun O and Priisalu H 2007 Random forest for gene expression based cancer classification: overlooked issues *Proc. of the 3rd Iberian Conf. on Pattern Recognition and Image Analysis, Part II* pp 483–90
[26] Wang M, Audi G, Kondev F, Huang W, Naimi S and Xu X 2017 *Chin. Phys.* C **41** 030003
[27] Sobiczewski A and Litvinov Y A 2014 *Phys. Rev.* C **89** 024311
[28] Krane K S *et al* 1987 *Introductory Nuclear Physics* (New York: Wiley)
[29] Duflo J and Zuker A 1995 *Phys. Rev.* C **52** R23
[30] Mendoza-Temis J, Hirsch J G and Zuker A P 2010 *Nucl. Phys.* A **843** 14
[31] Pastore A, Neill D, Powell H, Medler K and Barton C 2020 *Phys. Rev.* C **101** 035804
[32] Qi C 2015 *J. Phys. G: Nucl. Part. Phys.* **42** 045104
[33] Komer B, Bergstra J and Eliasmith C 2019 *Automated Machine Learning: Methods, Systems, Challenges* eds F Hutter *et al* (Berlin: Springer International) pp 97–111
[34] Heaton J 2017 arXiv:1701.07852
[35] Utama R, Piekarewicz J and Prosper H 2016 *Phys. Rev.* C **93** 014311
[36] Liran S and Zeldes N 1976 *At. Data Nucl. Data Tables* **17** 431
[37] Möller P, Myers W, Swiatecki W and Treiner J 1988 *At. Data Nucl. Data Tables* **39** 225
[38] Goriely S, Chamel N and Pearson J 2009 *Phys. Rev. Lett.* **102** 152503
[39] Wang N and Liu M 2011 *Phys. Rev.* C **84** 051303
[40] Barlow R J 1993 *Statistics: A Guide to the Use of Statistical Methods in the Physical Sciences* vol 29 (New York: Wiley)
[41] Utama R and Piekarewicz J 2017 *Phys. Rev.* C **96** 044308
[42] Neufcourt L *et al* 2018 *Phys. Rev.* C **98** 034318