Density Estimation Trees as fast non-parametric modelling tools

Lucio Anderlini

Istituto Nazionale di Fisica Nucleare, Sezione di Firenze – via Sansone 1, Sesto Fiorentino, 50019 Italy

E-mail: Lucio.Anderlini@cern.ch

Abstract. Density Estimation Trees (DETs) are decision trees trained on a multivariate dataset to estimate its probability density function. While not competitive with kernel techniques in terms of accuracy, they are incredibly fast, embarrassingly parallel and relatively small when stored to disk. These properties make DETs appealing in the resource-expensive horizon of the LHC data analysis. Possible applications may include selection optimization, fast simulation and fast detector calibration. In this contribution I describe the algorithm, made available to the HEP community in a RooFit implementation. A set of applications under discussion within the LHCb Collaboration are also briefly illustrated.

1. Introduction

The fast increase of computing resources needed to analyse the data collected in modern hadron-collider experiments, and the higher cost of processing units with respect to storage, pushes High-Energy Physics (HEP) experiments to explore new techniques and technologies to move as much as possible of the data analysis at the time of the data acquisition (*online*) in order to select candidates to be stored on disk, with maximal, reasonably achievable, background rejection. Besides, the complexity of the data analyses and the important inputs that the HEP community is receiving from the fast-growing community of *Data Scientists* motivate research and studies of multivariate algorithms able to operate in the distributed computing environments, being today key elements in data processing and analysis.

The statistical inference of the probability density function underlying a given dataset is extremely common in *High Energy Physics*. Fitting the dataset with a function is an example of *parametric* density estimation. When possible, defining a parametric form of the the underlying distribution and choose the values of those parameters maximising the likelihood is usually the right approach. In multivariate problems with a large number of variables and important correlation, however, fitting may become unpractical, and *non-parametric density estimation* becomes a valid, largely employed, solution.

In HEP, the most common non-parametric density estimation is probably *kernel density estimation* [1], based on the sum of normalized kernel functions centered on each data-entry.

In this write-up, I discuss *Density Estimation Trees*, algorithm based on *multivariate, binary tree* structure oriented to *non-parametric density estimation*. Density Estimation Trees, are less accurate than kernel density estimation, but they are incredibly faster and robust. Integrating Density Estimation Trees is also trivial and fast, making iterative search algorithms convenient.

Finally, storing a Density Estimation Trees and propagate it through the computing nodes of a distributed system is relatively cheap, which make them reasonable solutions for compressing the statistical information of a given dataset.

An implementation of the algorithm in ROOT/RooFit is available through CERN GitLab[1].

2. The algorithm

The idea of iteratively splitting a data sample, and estimating the probability density function of each portion of the parameter-space from the number of data entries it includes and its hyper-volume is not new. However, the technique had little room for applications in analyses of datasets up to a few thousands of entries described by small sets of correlated variables.

Recently, *kd*-trees [2], have been used to split large samples in sub-sets of roughly the same amount of data entries. The idea underlying *kd*-trees is the iterative split of the data-sample using as threshold the median of a given projection. While powerful to solve a vast range of problems, including notably nearest-neighbour searches, the lack of appropriateness of this technique to estimate probability densities should be evident thinking to samples with a non-negligible number of multiple data-entries (in at least one variable), as it is common with boolean or integer inputs.

Density Estimation Trees define the threshold to split the node ℓ, into the two sub nodes ℓ_R and ℓ_L, by minimizing the *Gini* index, $G(\ell) = R(\ell) - R(\ell_R) - R(\ell_L)$, where $R(\ell)$ represents the *replacement error* and is defined by [3]

$$R(\ell) \equiv -\frac{N_\ell^2}{N_{\rm tot}^2 V_\ell} \tag{1}$$

where V_ℓ is the hyper-volume of the portion of the data-space associated to the node ℓ, and N_ℓ the number of data entries it includes; $N_{\rm tot}$ is the number of data entries in the whole dataset.

Figure 1 shows an example of how the training is performed.

2.1. Overtraining

As in the case of Classification algorithms, overtraining is the misinterpretation of statistical fluctuations of the dataset as relevant features to be reproduced by the model.

An example of overtraining of Density Estimation Trees is presented in Figure 2. In the presented dataset, the alignment of data-entries in one of the input variables is interpreted as narrow spikes. To compensate spikes, in terms of absolute normalization, the density is underestimated in all of the other points of the parameter space.

Overtraining can be fought using either techniques developed specifically for decision trees or for density estimation algorithms. Overtraining in decision trees is controlled through an iterative approach consisting in *pruning* and *cross-validation*: finding and removing those branches of the tree increasing the complexity of the tree without enhancing the statistical agreement with a set of test samples (for example generated with the *Leave-One-Out* technique). Cross-validation is very expensive in terms of computing power, scales poorly with the number of data entries, and often fails to identify problems of over-training arising close to the root of the decision tree.

Overtraining in density estimation is fought, instead, by defining *a priori* an expected resolution width, neglecting fluctuation under that resolution while building the statistical model. For example, kernel density estimation algorithms require as an input a parameter, named *bandwidth*, which is related to the width of the kernel function. Abundant literature exists on algorithms to optimise the bandwidth for a certain dataset, or to use different bandwidths for different regions of the data space (*adaptive* kernel density estimation). Most of these techniques

[1] gitlab.cern.ch/landerli/density-estimation-trees

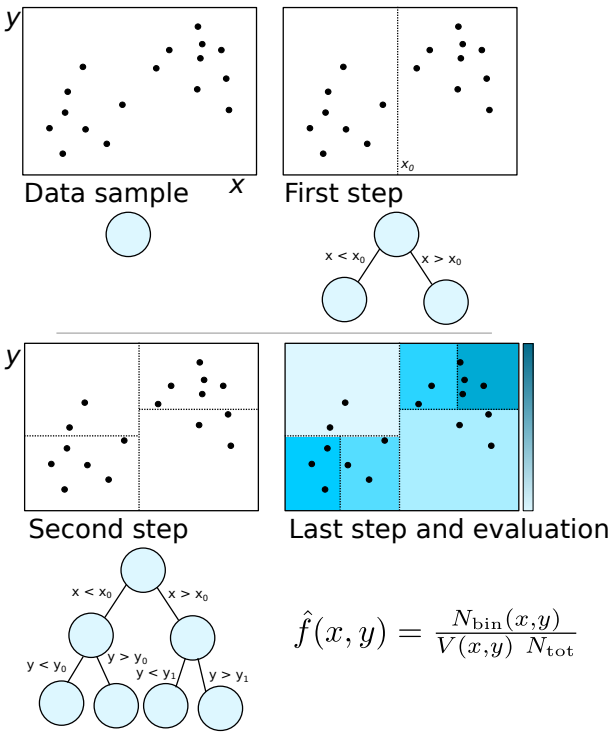

Figure 1. Schematic representation of the training and the evaluation procedures of a Density Estimation Tree.

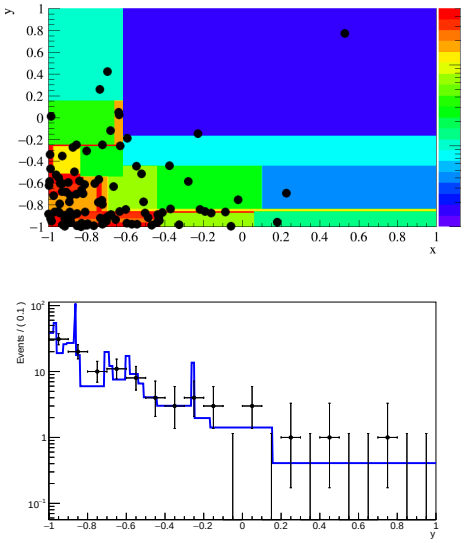

Figure 2. Top, an example of overtraining of Density Estimation Trees. The random alignment of data-entries with respect to one of the variables describing the problem is misinterpreted as a spike in the density estimation, as evident in the projection onto the vertical axis shown on the bottom.

are implemented as a preliminary step of the density estimation algorithm, often exploiting univariate variance estimators or clustering techniques. Growing a Density Estimation Tree with a minimal leaf width is fast, doesn't require post-processing and it is found to result in better-quality estimations with respect to cross-validation procedures. The same techniques and algorithms used to compute the optimal bandwidth parameter for kernel density estimation algorithm can be used to define *a priori* the optimal minimal leaf width of the Density Estimation Tree.

2.2. Integration

As mentioned in the introduction, fast integration of the statistical model built using Density Estimation Trees in one of the strengths of the algorithm. Integration usually respond to two different needs: *normalization* and *slicing*.

Integrals to compute the overall normalization of the density estimation, or the contribution in a large fraction of the data-space, should target the majority of the leaf nodes, and would gain little exploiting the tree structure of the density estimator. A sum over the contributions of each leaf node is considered the best strategy in this case

Instead, integrals over a narrow subset of the data-space should profit of the tree structure of the density estimation to exclude from the integration domain as many leaves as possible, as early as possible. In other words, in case of no intersection between the integral domain and the portion of data-space associated to a branch, it is useless to check every single leaf, associated to that branch for possible contributions to the integrals. Exploiting the tree structure when performing integrals of slices of the data-space can reduce the computing time drastically in

large density estimation trees.

Slicing is used often when projecting the density estimation to a histogram: each bin of the histogram is associated to a slice of the data-space and can be filled upon a slice integral as described above.

2.3. Operations with Density Estimation Trees

Density Estimation Trees can be used, for example, to model data sample including different components. It is therefore useful to combine different Decision Trees, with different weights, into a unique Density Estimation Tree. To achieve combination of Decision Trees, one needs at least the capability of performing scalar operations (such as multiplying by a constant) and binary operations (such as summing two Density Estimation Trees). The implementation of the former is pretty trivial, since it is sufficient to apply the scalar operation to value contained in each node composing the Density Estimation Tree, therefore here I focus on the latter.

Combining two Density Estimation Trees is not trivial because the boundaries are obviously different. The algorithm to combine two Density Estimation Trees consists in the iterative spit of the terminal nodes of the first tree, following the boundaries of the terminal nodes of the second tree. Once the combination is done, the first tree is compatible with the second and the binary operation (*e.g.* the multiplication) can be performed node per node. The resulting tree may have several additional layers with respect to the originating trees, therefore a final step removing division between negligibly different nodes is advisable.

3. Discussion of possible applications

Density Estimation Trees are useful to approach problems defined by many variables and for which huge statistical samples are available. To give a context to the following examples of applications, I consider the calibration samples for the Particle IDentification (PID) algorithms at the LHCb experiment.

PID calibration samples are sets of decay candidates reconstructed and selected relying on kinematic variables only to distinguish between different types of long-lived particles (electrons, muons, pions, kaons, and protons).

The PID strategy of the LHCb detector consists in the combined response of several detectors: two ring Cherenkov detectors, an electromagnetic calorimeter, a hadronic calorimeter and a muon system [4]. The response of the single detectors are combined into likelihoods used at analysis level to define the tightness of the PID requirements.

Calibration samples count millions of background-subtracted candidates, each candidate is defined by a set of kinematic variables (for example momentum and pseudorapidity) and a set of PID likelihoods, one per particle type. The correlation between all variables is important and not (always) linear.

3.1. Efficiency tables

The first application considered is the construction of tables defining the probability that a candidate having a given set of kinematic variables, have likelihoods satisfying some criteria.

Building two Density Estimation Trees, with the kinematic variables defining the data-space, one with the full data sample (tree t_{all}), and one with the portion of data sample passing the PID criteria (tree t_{pass}) allows to compute the efficiency for each combination of the kinematic variables by evaluating the Density Estimation Tree obtained taking the ratio $t_{\text{pass}}/t_{\text{all}}$.

For frequently-changing criteria a dynamic determination of the efficiency can be envisaged. For simplicity, consider the generic univariate PID criterion $y > 0$. In this case a single Density Estimation Tree $d(x_1, x_2, y)$ defined by the kinematic variables (x_1, x_2), plus one PID variable y, has to be trained on the calibration sample. The dynamic representation of the efficiency for

a candidate having kinematic variables $(\hat{x}_1, \hat{x}_2)$ is the ratio

$$\epsilon(x_1, x_2; y > 0) = \int_{y>0} d(\hat{x}_1, \hat{x}_2, y)\mathrm{d}y \Big/ \int_{\text{any } y} d(\hat{x}_1, \hat{x}_2, y)\mathrm{d}y. \quad (2)$$

Thanks to the fast slice-integration algorithm, the computation of this ratio can be included in an iterative optimization procedure aiming at an optimization of the threshold on y.

3.2. Sampling as fast simulation technique

Another important application is related to fast simulation of HEP events. Full simulation, including interaction of the particles with the matter constituting the detectors, is becoming so expensive to be expected exceeding the experiments' budgets in the next few years. Parametric simulation is seen as a viable solution, as proved by the great interest raised by the DELPHES project [5]. However, parametrizing a simulation presents the same pitfalls as parametrizing a density estimation: when correlation among different variables becomes relevant, the mathematical form of the parametrization increases in complexity up to the point in which it is unmanageable.

Density Estimation Trees are an interesting candidate for non-parametric fast simulation. Let $d(x_1, ..., x_N, y_1, ..., y_N)$ be a Density Estimation Tree trained on a set of candidates characterized from the variables $(x_1, ..., x_N)$, for which full simulation is performed for variables $(y_1, ..., y_N)$.

Given a new set of variables $(\hat{x}_1, ..., \hat{x}_N)$, it is possible to exploit the tree structure of d to identify quickly which leaves have non-null intersection with $(\hat{x}_1, ..., \hat{x}_N)$. Choosing randomly one of those leaves, and generating a set of random numbers $(\hat{y}_1, ..., \hat{y}_N)$ within that leaf, is a fast procedure to obtain a set of output variables following the same distributions as described by the full simulation, including correlations with input and output variables.

4. Summary and outlook

I discussed Density Estimation Tree algorithms as fast modelling tools for high statistics problems characterized by a large number of correlated variables and for which an approximated model is acceptable. The fast training and integration capabilities make these algorithms of interest for the high-demanding future of the High-Energy Physics experiments. The examples discussed, which benefited from an active discussion within the Particle Identification Group of the LHCb collaboration, explore cases where the statistical features of huge samples have to be assessed faster than how standard estimators would require. In future, Density Estimation Trees could be used to train Regression Multivariate Algorithms, such as Neural Networks, in order to further speed up the query time (but loosing the fast-integration properties).

Acknowledgements

I thank Alberto Cassese, Anton Poluektov, and Marco Cattaneo for the encouragements in developing this work and for the useful discussions we had.

References

[1] K. S. Cranmer, Comput. Phys. Commun. **136** (2001) 198 doi:10.1016/S0010-4655(00)00243-5 [hep-ex/0011057].

[2] J. L. Bentley, Communications of the ACM **18** (1975) 9 doi:10.1016/S0010-4655(00)00243-5

[3] P. Ram and A. G. Gray, Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 627-635

[4] A. A. Alves *et al.*, The LHCb detector at the LHC, JINST 3 (2008)

[5] J. de Favereau *et al.* [DELPHES 3 Collaboration], JHEP **1402** (2014) 057 doi:10.1007/JHEP02(2014)057 [arXiv:1307.6346 [hep-ex]].