UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Development of Boosted Decision Trees for energy reconstruction of Inverse Beta Decay events in JUNO

Relatore

Prof. Alberto Garfagnini

Correlatore

Dr. Yury Malyshkin

Laureando

Alessandro Compagnucci

Anno Accademico 2017/2018

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

BDT Boosted Decision Trees

IBD Invese Beta Decay

NMO Neutrino Mass Ordering

PMT Photomultiplier Tube

JUNO Jiangmen Underground Neutrino Observatory

FV Fiducial Volume

DN Dark Noise

1 INTRODUCTION

One of the open problems in particle physics is to determine the mass ordering of the three fundamental neutrinos [3, 12]. Several experiments in the years opened the possibility to do that by measuring flavor oscillation with a large reactor neutrino experiment. JUNO [2, 6], currently under construction in South of China, is a detector built for this main reason, is planned to be ready by 2020.

In the detector neutrinos are detected by the Inverse Beta Decay (IBD) and photons are collected by thousands of Photomultiplier Tubes (PMTs) [2, 6]. The energy resolution expected for JUNO, in order to fullfill his main objective, is expected to be the highest ever achieved in reactor neutrino experiments. the optimization of the problem is non trivial and several kind of approaches need to be evaluated, even in this phase of the experiment, with simulated data.

Supervised learning algorithms [10] provide several ways to tackle the energy reconstruction challenge in an automated manner. Usually very little preprocessing of the data needed to feed the model is required in order to get accurate results whereas optimizing the problem in the traditional way could be very challenging (i.e. finding the right set of hyperparameters in a multi-dimensional space).

Boosted Decision Trees (BDT) [5] are a very promising algorithm in supervised learning, they group together several weaker predictors (single regression trees) into stronger ones in a *greedy* manner and can be trained very effectively thanks to a smart approach in optimizating the objective function taking also into account the regularization term.

In this work will began explaining the motivations behind the experiment and talk about the theoretical model used by the libraries.A simple analysis on Monte Carlo data is then performed with the Catboost [14] BDT library along with the Python scientific suite, several models are built with different sets of features from the events and performance at different energies is evaluated. After that we tackle other problems such as building a classifier to reject outliers, seeing how many training data are really needed and the effect of real phenomena such as dark noise on the model. Possibilities of vertex reconstruction are also explored at the end. Then, finally, conclusions are drawn.

Part I

BACKGROUND

2

THE JUNO EXPERIMENT

The Jiangmen Underground Neutrino Observatory (JUNO) [2, 6] is an international cooperation experiment which is designed to study neutrino physics. Its main purpose is to determine the mass hierarchy, which is a fundamental property of neutrinos and still an open question. The focus of this section is to develop some background on the scientific motivation behind the search for better reconstruction methods and the JUNO experiment in general.

2.1 NEUTRINO PHYSICS, AND MOTIVATION

We know that neutrinos exist in three flavor eigenstates [12]

$$|\nu_e\rangle\,, \quad |\nu_\mu\rangle\,, \quad |\nu_\tau\rangle$$

and three mass eigenstates of masses m_1, m_2, m_3:

$$|\nu_1\rangle\,, \quad |\nu_2\rangle\,, \quad |\nu_3\rangle\,.$$

They are related by the Maki-Nakagawa-Sakata-Pontecorvo (MNSP) matrix

$$\begin{pmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{pmatrix} = \begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix}$$

where the U matrix that can be expressed as follows:

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix} P_\nu$$

where $c_{ij} = \cos(\theta_{ij})$ and $s_{ij} = \sin(\theta_{ij})$ are defined, and $P_\nu = Diag\{e^{i\rho}, e^{i\sigma}, 1\}$ is the Majorana phase matrix.

Flavor oscillation can be expressed with the transition probabilities

$$P_{\alpha\to\beta} = \left|\,\langle\nu_\beta|\nu_\alpha(t)\rangle\right|^2 = \left|\sum_i U^*_{\alpha i} U_{\beta i} e^{-im_i^2 L/2E}\right|^2$$

For example the survival probability of $\bar{\nu}_e$ can be expressed as

$$\begin{aligned} P(\bar{\nu}_e \to \bar{\nu}_e) = 1 &- \sin^2 2\theta_{12} \cos^4 \theta_{13} \sin^2 \frac{\Delta m^2_{21} L}{4E} \\ &- \sin^2 2\theta_{12} \cos^4 \theta_{13} \sin^2 \frac{\Delta m^2_{31} L}{4E} \\ &- \sin^2 2\theta_{13} \cos^4 \theta_{12} \sin^2 \frac{\Delta m^2_{32} L}{4E} \end{aligned}$$

where oscillations are driven by the mixing factors θ_{12}, θ_{13} and the mass works as *frequencies*: $\Delta m^2_{21} = m^2_2 - m^2_1$, $\Delta m^2_{31} \approx \Delta m^2_{32} = m^2_3 - m^2_2$.

Measuring neutrino oscillation is crucial to shed light on the Neutrino Mass Ordering (NMO) problem (Fig. 2.1), i.e. we know that $m_2 > m_1$ and $\Delta m^2_{31} \gg \Delta m^2_{21}$ from other evidences, but we want to know if $m_3 > m_{1,2}$ or $m_{1,2} > m_3$.

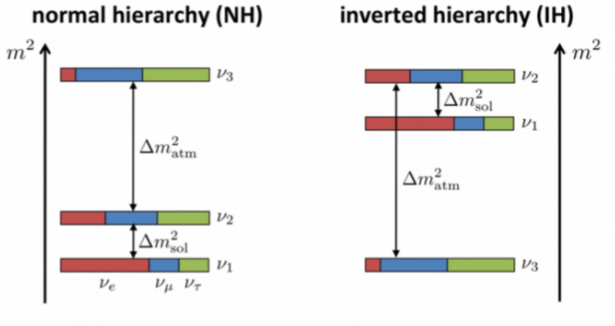


Figure 2.1: The Neutrino Mass Ordering (NMO) problem

The discovery of a non null θ_{13} mixing factor by Daya Bay [1] opened the possibility to measure MH by a large reactor neutrino experiment, so Jiangmen Underground Neutrino Observatory (JUNO) was proposed in 2008. Later approved in 2013, it is under design and construction, with the primary goal to determine NMO. JUNO will also measure other mixing parameters with world-leading precision, and its superior detector properties also provide a great opportunity to study neutrinos from other sources such as Supernova, the Earth's interior and the Sun, sterile neutrinos, dark matter and other exotic searches.

2.2 JUNO STRUCTURE AND REQUIREMENTS

The JUNO experiment will be constructed in Jiangmen, China located 700 meters underground and about 53 km away from two nuclear power plant. At its core the main detector is a 35.4 m a spherical ball of acrylic filled with 20 000 tons of liquid scintillator immersed in pure water. The primary component installed around the detector is the Photomultiplier Tube (PMT). About 18 000 20-inch PMTs are installed in the pool around the detector along with smaller ones ($\sim$ 36 000) to

increase effective coverage of the detector and cross-calibrate the large ones. The structure is illustrated in Fig. 2.2.

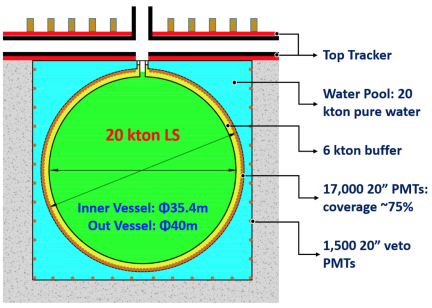


Figure 2.2: JUNO detector structure

Antineutrinos are produced by 2 nuclear power plants (10 total cores) located at the same distance from the detector in order to maximize the total yield, the energy peaks at around 4 MeV, with most of them at less of 10 Mev. The main channel for detecting antineutrinos is the Invese Beta Decay (IBD) $\overline{\nu_e} + p \rightarrow e^+ + n$, this is the reaction with the largest cross section in a few MeV range and with far the largest power to reject backgrounds. The positron carries almost all the energy and forms the *prompt* signal, the positron travels only a few centimeters, which can be ignored compared to the size of the detector. Therefore, the positron track in the liquid scintillator can be regarded as point-like light source at the IBD vertex position.

In order to address its main purpose JUNO is expected to have over 6 years of total run time ($\sim 10^5$ collected events). The energy resolution expected can be parametrized as $\frac{\sigma}{E} = \sqrt{\frac{A^2}{E} + B^2 + \frac{C^2}{E^2}}$, where A, the stochastic term, is determined by the photo electrons yield, B is a constant term and C is a noise term, mainly given by PMT dark noise. Taken into account all of these effects the projected energy resolution of the JUNO detector is $3.0\%/\sqrt{E(\text{MeV})}$.

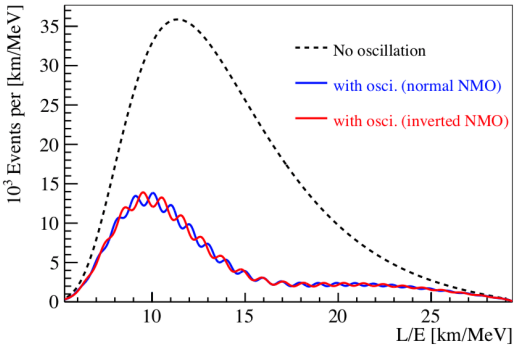


Figure 2.3: Expected reactor $\bar{\nu}_e$ spectrum, for no oscillation and for different mass orders. Taken from [6]

The expected reactor $\overline{\nu_e}$ spectrum at JUNO for different NMOs is shown in Fig. 2.3. An excellent energy resolution is needed in order to discriminate between the two hypothesis. The reconstruction problem therefore needs to be addressed also with different techniques such as neural networks and deep learning in order to take full advantage of the experimental resolution of the detector.

3

MACHINE LEARNING AND BOOSTED DECISION TREES

Boosted Decision Trees (BDT) [5] are used for supervised learning problems, where we use the training data (with multiple features) x_i, for example the data for the event from the detector, to predict a target variable y_i, in our case the energy E_{rec} of the event. We will review some basic concept in supervised learning and then talk about boosted trees and the Catboost library [14].

3.1 SUPERVISED LEARNING INTRODUCTION

In supervised learning the mathematical structure by which the predictions y_i are made from the inputs x_i is usually called the *model*. A common example is a linear model, where the prediction is given as $y_i^\star = \sum_j \theta_j x_{ij}$, a linear combination of weighted input values.

The *parameters* are the unknown part that we need to learn from data. In linear regression problems, the parameters are the coefficients θ_j.

The task of *training* the model amounts to finding the best parameters θ_j that fit better the training data x_i and labels y_i. In order to train the model, we also need to define the *objective function* to measure how well the model fits the training data and to provide a testing dataset to see if it also generalize well.

A salient characteristic of objective functions is that they consist of two parts: training loss and regularization term,

$$\text{Obj}(\theta) = L(\theta) + \Omega(\theta).$$

The training loss measures how *predictive* our model is with respect to the training data. For example we will use for L the *root mean squared error* (RMSE) function

$$L(\theta) = \sqrt{\sum_i (y_i - y_i^\star)^2}.$$

The *regularization term* controls the complexity of the model and helps preventing overfitting (we will go into details later). One of the principles of machine learning is that a model should be as simple as possible in order to performe well with unseen input data in a tradeoff known as the *bias-variance tradeoff*.

3.1.1 *Decision tree ensemble*

The model that we will take into consideration is built on *decision trees ensemble*. A decision (regression) tree is shown in Fig. 3.1, input data are split by feature and at the end of each leaf a real score is assigned.

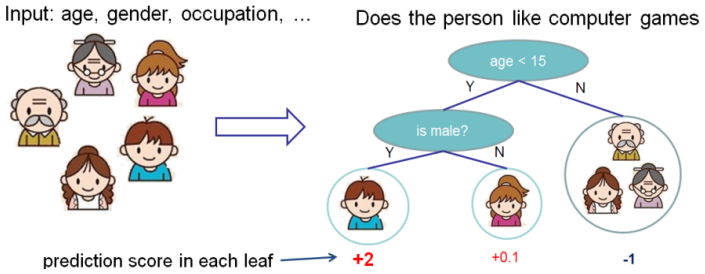


Figure 3.1: A simple tree model. Taken from [5].

A decision tree ensemble takes the predictions from several trees to make usually stronger predictions (Fig. 3.2).

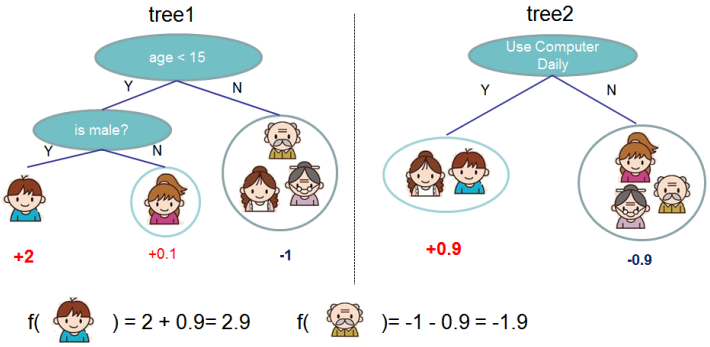


Figure 3.2: A tree ensemble example. Taken from [5].

The predictions from the two trees in the example are added together. Predicted values can be written as

$$y_i^\star = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

where K is the number of trees and f a function in the functional space $\mathcal{F}$ of all the possible trees, and the objective function to optimize in training is

$$\text{Obj}(\theta) = \sum_{i=1}^{n} L(y_i, y_i^\star) + \sum_{k=1}^{K} \Omega(f_k).$$

In order to optimize the objective function it is unfeasible to try to optimize every tree at once, instead an additive strategy is used in which at every step a new tree is bult and added to the model. So that predicted values $y_i^{\star(t)}$ at every step t became

$$
\begin{aligned}
y_i^{\star(0)} &= 0 \\
y_i^{\star(1)} &= f_1(x_i) = y_i^{\star(0)} + f_1(x_i) \\
&\dots \\
y_i^{\star(t)} &= \sum_{k=1}^{t} f_k(x_i) = y_i^{\star(t-1)} + f_t(x_i)
\end{aligned}
$$

At each step a the new tree added is the one that optimizes the objective function

$$\text{Obj}^{(t)}(\theta) = \sqrt{\sum_{i=1}^{n}(y_i - (y_i^{\star(t-i)} + f_t(x_i))^2} + \sum_{i=1}^{t}\Omega(f_i)$$

where we added the RMSE loss function defined before. Usually we take a taylor expansion of our loss function

$$\text{Obj}^{(t)}(\theta) = \sum_{i=1}^{n}[l(y_i, y_i^{\star(t-i)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

with g_i and h_i defined as

$$
\begin{aligned}
g_i &= \partial_{y_i^{\star(t-1)}} l(y_i, y_i^{\star(t-i)}) \\
h_i &= \partial^2_{y_i^{\star(t-1)}} l(y_i, y_i^{\star(t-i)}),
\end{aligned}
$$

the derivatives of the loss function with respect to predicted values calculated on the previous iteration. So after removing all of the constant terms the function to minimize at every step for the new tree become

$$\sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t).$$

The regularization term $\Omega(f)$ is a function that take care of the complexity of the tree in our case. Before, we redefine $f(x)$ as

$$f_t(x) = w_{q(x)}, \quad w \in \mathbb{R}^D, \quad q : \mathbb{R}^d \to \{1, 2, \dots, T\}$$

where w is the vector of scores on leaves, q a function assigning each data point to each leaf and T the number of leaves. The complexity is then definited as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

3.1.2 *Structure score*

After re-formulating the tree model, we can write the objective value with the t-th tree as:

$$\begin{aligned}
\text{Obj}^{(t)} &\approx \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \\
&= \sum_{j=1}^{T}[(G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T
\end{aligned}$$

where $I_j = \{i|q(x_i) = j\}$ is the set of indices of data point assigned to the j-th leaf. The form of the last expression is quadratic in w_j and the best score w_j possible for each tree structure $q(x)$ and the best objective reduction possible are, respectively:

$$\begin{aligned}
w_j^* &= -\frac{G_j}{H_j + \lambda} \\
\text{obj}^* &= -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T
\end{aligned}$$

The last equation measures *how good* a tree structure is. Basically for every instance the statistics g_i and h_i are pushed to the leaves they belong to and added together. The latest score, which takes into account also model complexity is then assigned.

3.1.3 *Tree structure learning*

Ideally, we should enumerate all the possible tree structures and pick the one who scores better. In pratice this approch is intractable, one level of the tree is optimized at a time instead. A leaf is split into two, usually performing a scan from left to right on the sorted data assigned to that leaf, and *gain* for the loss function is calculated as

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L^2 + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

which is the difference between the score of the new right/left leaves and the old leaf. The best split is the one who maximise gain. Note also that if the gain is negative because of γ it is better not to add that leaf (same as *pruning* in other tree implementation).

3.2 THE CATBOOST LIBRARY

CatBoost [14] is the name of an implementation of boosted decision trees algorithm by Yandex. It delivers state-of-art performance compared to other solutions and its main focuses are in a different application of BDTs in order to fight the *prediction shift* found in other solutions for certain kind of distributions and in the support for categorical features with a new faster approch [9]. Thanks to the optimization of the library and to the approach in general there's also no need to run the software on expensive hardware or for a long time, a session of training with the dataset taken into consideration (5/6 features, 1 million entries) takes only about 2/3 minutes on a standard laptop, this is a great improvement compared to other machine learning approaches such as neural networks.

The Catboost python library will be used in this work along with other standard scientific libraries (*numpy*[1], *matplotlib*[2], *scikit-Learn*[3], etc ...), the main focus will be on the use of the class *CatboostRegressor* for our reconstruction purpose, although we will build also a classifier based on the class *CatboostClassifier*. The model comes with a set of tunable parameters to change the tree structure and the training speed, the set of the most notable ones are:

- `iterations` : The maximum number of trees that can be built (default value is 1000), the final number could be less if training is interrupted i.e. for overfitting
- `learning_rate` : Used to reduce the gradient step in the process (default value automatically set based on the other parameters and dataset properties).
- `loss_function` : the metric used in training[4], Catboost also support custom metrics.
- `depth` : The maximum detph of the trees, maximum supported number is 16 (default value is 6)
- `l2_leaf_reg` : The regularization coefficent for the quadratic term, i.e. λ in the former introduction to BDTs (default value is 3).

1 `http://www.numpy.org`
2 `https://matplotlib.org`
3 `https://sklearn.org`
4 For a list of provided metrics see `https://tech.yandex.com/catboost/doc/dg/concepts/loss-functions-docpage`

For overfitting detection, a training dataset will be splitted and a small portion will be used for testing. Loss is calculated at every iteration also on the testing dataset and if no improvement is done after some step (i.e. 50) training is ended with the final model shrinked to the one with the better score on testing data. To do this we use the parameters `od_type = "Iter"`, `od_wait = 50` in the model. This also saves some time if no improvement could be made.

Part II

ANALYSIS

4

BOOSTED DECISION TREES FOR ENERGY RECONSTRUCTION

BDTs are typically trained to address *classification* and *regression* problems. For example in our case we are trying to estimate $E_{\text{rec}} = f(p_1, p_2, \dots)$ as a function of input parameters p_i, is therefore the estimation of a continuous variable, a regression problem.

The goal of our model will be to reconstruct f given:

- The most informative set of input parameters.
- A good set of hyperparameters to tune the model of the CatBoost library for the best performance.
- A training dataset with both input parameters and corresponding output true values, which should also be large enough.

Monte Carlo samples are very flexible and give the opportunity to produce a lot of training data therefore they are usually preferred initially, whereas real calibration data, produced by positioning radioactive sources at fixed locations give limited statistics and should really be used afterwards. The output from the simulation libraries, thousands of waveforms from the PMTs, should be processed to produce a training dataset with only a small set of feature to feed the model. An existing dataset has been provided [7] to address this problem, containing only features comprehensive of an event as long as the labels associated with it (true energy E_{true}, and position coordinates).

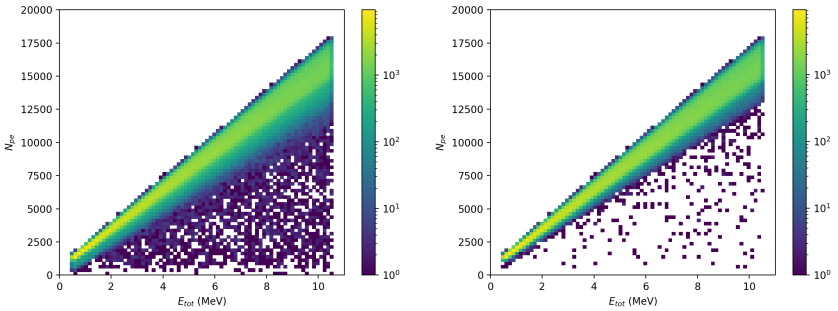
4.1 DATA STRUCTURE AND FEATURES

The training dataset consists of one milion Monte Carlo events stored in a ROOT [4] `tree` with the following parameters:

ENERGY (`E0`) True total energy of the positron ($E_0 = E_{\text{kin}} + m_e c^2$), with flat energy distribution in the range $[0.511 : 10.511]$ MeV. The events are simulated with given kinetic energy between 0 and 10 MeV (continuous).

POSITION (`x`, `y`, `z`, `r`) True vertex of the event: x, y, z and radial component.

TOTAL NUMBER OF PHOTO-ELECTRON (`totalPE_lpmt`,`totalPE_spmt`) Total number of photo-electrons collected by large and small PMTs.



(a) *without FV cut* (b) *with FV cut*

Figure 4.1: Total energy vs. total number of photo-electrons collected by larger PMTs, in (b) a lot of outliners are removed simply removing events with $r > 17.2$ m

It provides a good estimate of the total energy of the event (Fig. 4.1), although some spatial dependence prevents good results without other information.

HIT TIME (`ht_1`, `ht_mean`, `ht_rms`) Hit time features: first hit time, mean and RMS of the total hit time distributions. In order to get comparable informations between the events we need to feed to the model `ht_mean-ht_1`.

CENTER OF HITS (`coh_x`, `coh_y`, `coh_z`) We can define the quantity

$$CoH = \frac{1}{R_{\text{CD}} N_{\text{PMT}}} \sum_{i}^{N_{\text{PMT}}} P_i n_i^{\text{p.e.}}$$

where R_{CD} is the radius of the central detector, P_i the position of the i-th PMT, $n_i^{\text{p.e.}}$ the number of photo-electrons collected. We can use as features the three coordinates or the radial one. It provides a good estimate of the event position (Fig. 4.2). The dependence is roughly linear from the center to about 15.5 meters then a knee is clearly visible. This effect is caused mainly by total reflection of photons near the edge between the acrylic sphere, filled with scintillator, and the surrounding water in which the PMTs are located.

4.2 ANALYSIS

After splitting the whole dataset in 900k events for training and 100k events for testing, we use a CatboostRegressor model with the following set of initial parameters:

- `learning_rate = 0.1`
- `iterations = 1000`

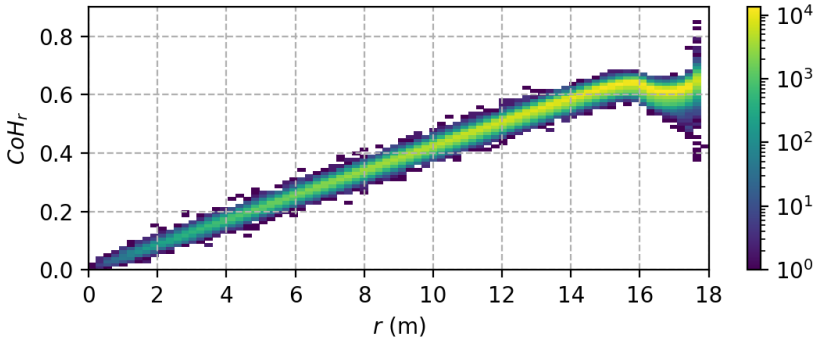
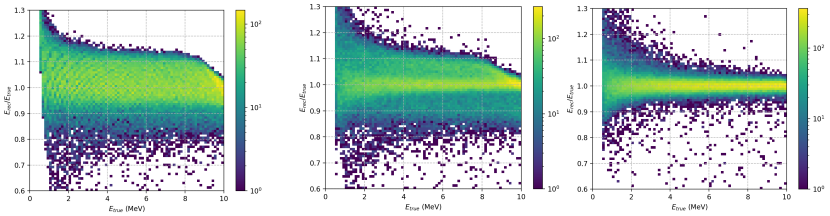


Figure 4.2: Radial component of the center of Hits vs. radius of the event

- `depth = 10`
- `loss_function = 'RMSE'`
- `L2_leaf_reg = 14`

The early stopping feature of the Catboost library also prevents overfitting by halting the training and shrinking the model if no improvement for the loss function is made in the testing datasets after 50 iteration. As input we take at first only the total number of photo-electrons collected by PMTs and then we add the three component of the center of hits and time informations (the difference between first hit time and mean of the hits time distribution). Training time on a traditional laptop is only 2-3 minutes for each run, which seems reasonable.



(a) *only the number of photo-electrons:* RMSE = 0.4558

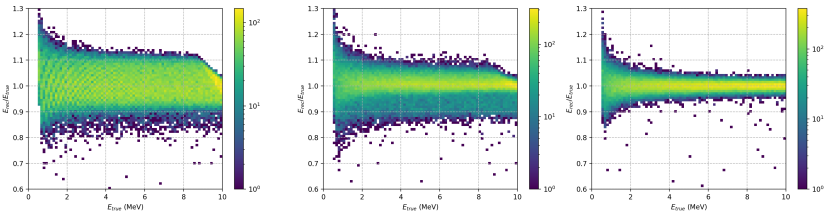
(b) *adding C.o.H. coordinates:* RMSE = 0.4030

(c) *adding time information:* RMSE = 0.2378

Figure 4.3: Model comparison using 900k events for training and 100k for testing, the value for the loss function used (RMSE) is also shown

The results are shown in Fig. 4.3, the presence of a certain number of outliners is evident. Further inspection reveals that in fact all of them are from events with higher radius ($r > 17$ m), near the detector edge and even inside the PMTs, it is convenient to get rid of that events introducing the *fiducial volume* (FV) cut, rejecting events with $r > 17.2$ m; this leads to about 8 % loose of statistics. The cut is performed *a*

priori knowing the real radius of the events, we will see later how to build a model to discriminate this events using only the features. With the remaining 917 561 events we train again the model (Fig. 4.4), the result are now much better with 100 000 events left for testing. A lot of outliners are removed and the model seems more reliable



(a) *only the number of photo-electrons:* RMSE = 0.3228

(b) *adding C.o.H. coordinates:* RMSE = 0.2313

(c) *adding time information:* RMSE = 0.1147

Figure 4.4: Model comparison after the introduction of the FV cut, 100k for testing

4.3 PERFORMANCE OF THE MODEL

To evaluate the performance (i.e. see if the target $3\%/\sqrt{E}\ \sigma$ is achievable with the trained model) we use 10 datasets of 2000 events each simulated with fixed kinetic energy of the positron (in the range $[0 : 10]$ MeV), the FV cut is performed and prediction are made only with the viable events. We fit the prediction for each dataset with a gaussian distribution and then plot E_{vis} vs. σ/E_{vis} (Fig. 4.5) where

$$E_{\text{vis}} = E_{\text{kin}} + E_{\gamma} = (E_0 - m_e c^2) + 1.022\ \text{MeV} = E_0 + 0.511\ \text{MeV}.$$

It is clear that adding all the information leads to results comparable with the most recent traditional recostruction algorithm, in this case for example data are compared to the function

$$\frac{\sigma}{E_{\text{vis}}} = \sqrt{\left(\frac{2.821}{\sqrt{E_{\text{vis}}}}\right)^2 + 0.5947^2 + \left(\frac{0.0}{E_{\text{vis}}}\right)^2},$$

in which the coefficients are taken from recent internal presentations on energy reconstruction with conventional methods, for more details on the parametrization see [2]. Note that the feature at 1.022 MeV for the dataset at 0 MeV of kinetic energy are caused by the nature of training data which never gets lower than $E_{\text{vis}} = 1.022$ and so the predictions gets overestimate.

To look more closely at the charateristics of the prediction, we produce bias distributions $((E_{\text{rec}} - E_{\text{true}})/E_{\text{true}})$ which are shown in Fig. 4.6.

Some cosideration:

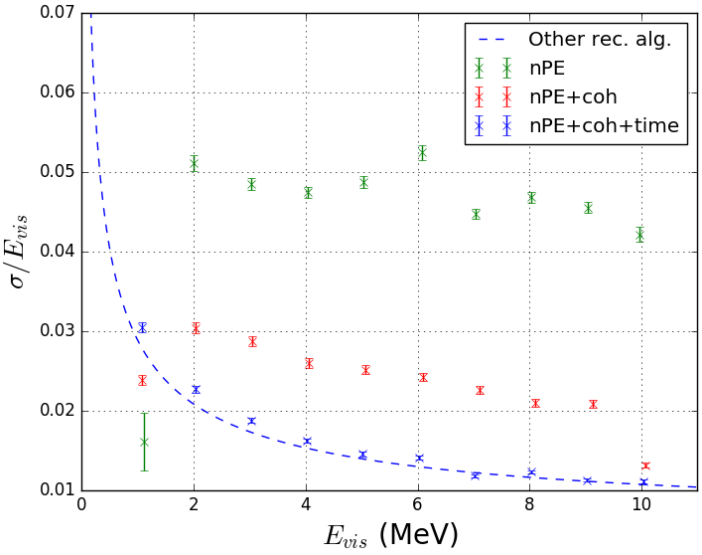
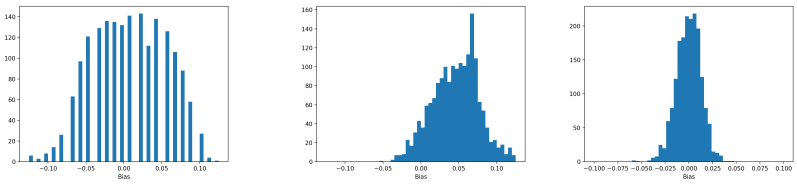


Figure 4.5: Performance of the models compared to traditional method (dashed line). Adding together $N_{\text{p.e.}}$, *Coh* components and time informations the results are comparable with non-ML techniques.



(a) *total number photo-electrons, 6 MeV* (b) *time model, 0 MeV* (c) *time model, 6 MeV*

Figure 4.6: Some bias distributions

- With only the total number of photo-electrons the results of the predictions are discrete, due to the nature of decision trees (a). The effect on models with more features seems negligeble.
- Also at 0 MeV prediction have significant bias due to the spectrum of the training data (b).
- No bias is evident in the central region at 6-8 MeV (c).

To clearly show that there is no significant bias we plot the visible energy vs. the related bias distribution center (picture 4.7). The bias is always less than 1%, well below the sigma on the measuraments, positive bias for the model with CoH without time could be explained from the asimmetry of the graph 4.4 (b).

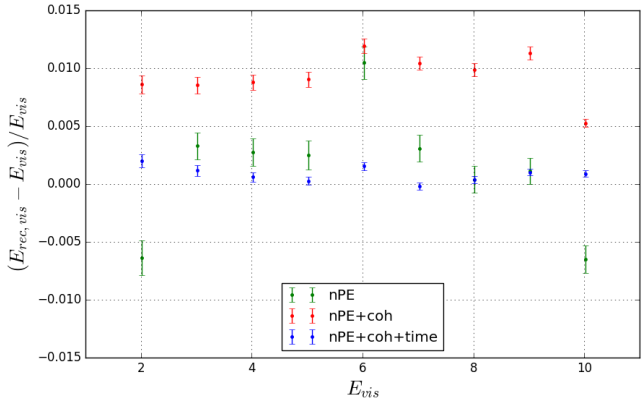


Figure 4.7: Bias vs E_{vis} plot, the first dataset at $E_{\text{vis}} = 1.022$ is not shown because is very biased (see Fig. 4.6 (b))

4.3.1 *Tuning the model: GridSearchCV*

There are a significant number of parameters in the model which can be adjusted in order to achieve a better performance. It is important also to *cross-validate* the model to see if it generalizes on testing data as it should and prevents overfitting. The function `GridSearchCV`[1] for the Scikit-learn [8] Python package addresses this problem, giving the opportunity to train the model with a pool of parameter and pick the ones that generalize better with testing data. The dataset is splitted in n sets and a model with parameters from the pool is trained n times using $n - 1$ sets for training and 1 for testing, a score is given to each model as the average of the results and the best model is choosen between the pool of parameters. For the test we work with a smaller dataset of 100k events, the number of iteration is fixed at 500 as well as the loss function `RMSE` and we search between the parameters:

- `depth`: 4, 7, 10
- `learning_rate`: 0.01, 0.03, 0.1
- `l2_leaf_reg`: 1,4,9

From the run the best model seems to be the one with depth $=$ 10, learning rate $=$ 0.03 and leaf regularization $=$ 4. Looking at the output results from the run we can also point out that:

- The leaf regularization parameter doesn't seem to matter that much for our model (at least in our case, and in the range taken into consideration).
- The depth should be high enough (i.e. > 7) but not to high (i.e. < 12) to avoid a model to be too complex.

1 `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

- the learning rate should be choosen in combination with number of iterations, so that the loss value for the testing dataset settle at the end of the training to avoid overfitting.

Results of the parameters optimization do not differ that much from our initial guesses or even from the default parameters, this shows that the library work really well as it is or with minimal tweaking.

4.4 BUILDING A CLASSIFIER FOR FV CUT

A model to classify events in the *fiducial volume* from the experimental data is needed in order to use the model with real data. It is also, in theory, a good opportunity to test BDTs which are in general more suited for classification problem rather than for regression [11].

One can build a classifier with the CatboostClassifier[2] class of the Python library, it handles automatically the conversion of the output of the trees in probability (i.e. by logistic trasformation) and the tunable parameters are more or less the same as for the regressor (with exception for the loss function).

For the model we use the following features:

- Total number of photo-electrons collected by larger PMTs
- Center of hits coordinates as well as the radial component.
- Hit time distribution mean.

The dataset is splitted in 990 000 events for training and 10 000 events for testing. We train a CatboostRegressor model with 1000 iterations, the depth set to 6 and learning rate set to 0.03, the loss function is the classical binary logistic function [3] (`LogLoss`) for classification problems. The final accuracy on the testing data is 98.77 % and the performance on the fixed energies datasets are shown on table 4.2.

Accuracy without Coh_r	98.40 %
Accuracy with Coh_r	98.77 %

Table 4.1: An example of *kernel trick*: adding the radial component of CoH improve the model because of the linear dipendence with the radius (accuracy calculated on 10 000 test events).

It is also important to note that all of the inaccurate predictions are for events on the edge of Fiducial Volume (FV) cut distributed within $\sigma \approx 10$ cm, and no significant outliner is present (Fig. 4.8).

2 `https://tech.yandex.com/catboost/doc/dg/concepts/python-reference_catboostclassifier-docpage/`

3 $\ell = -\sum_i c_i \log(p_i) + (1 - c_i) \log(1 - p_i)$ where $i = 1, \ldots, N$ are elements of the test dataset, c_i the class label (0,1) of the element for a binary classification problem, p_i the predicted probabilities.

Energy (MeV)	Accuracy (%)
0	96.60
1	98.70
2	98.95
3	98.70
4	98.90
5	98.65
6	98.90
7	99.25
8	98.85
9	99.00
10	99.30

Table 4.2: Accuracy performance on fixed energy datasets.

In conclusion the classifier should be very reliable and results are virtually the same as with the *a priori* fiducial cut (although new results of the reconstruction with only events classified by the model are omitted, results are not spoiled by a few misclassified events at the edge of the cut).

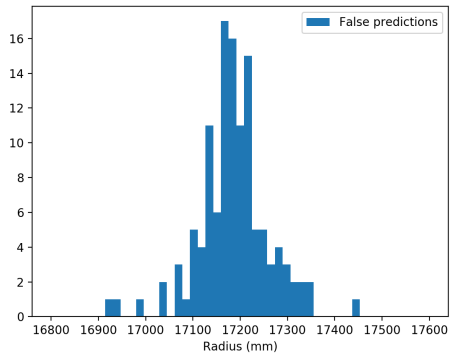


Figure 4.8: False prediction of the classfier for 10 000 testing events, no significant outlier is present.

4.5 PERFORMANCE WITH SMALLER DATASETS

A common problem in machine learning application is to determine how many events are enough to train the model to expected performance. This will be especially important later on when the experiment will be calibrated with real radiation sources placed in specific places

into the detector, the question is: How large the statistic of the training events should be for the model to be reliable?

When other common implementations such as neural networks require a lot of data, decision trees should in theory be capable to achieve expected performance with smaller datasets. The knowledge of the density of events needed to achieve expected performance is also important in the prospect of real data acquisition and energy calibration of the detector.

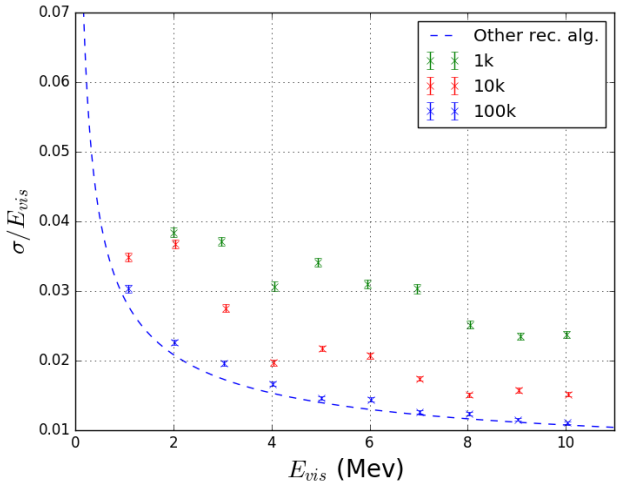


Figure 4.9: Performance with smaller datasets compared to a traditional recostruction algorithm

We train our model with subsets of 1000, 10 000, 100 000 events of the larger datasets. The models are essentially the same as in the former training except for a few tweaks. After training are then tested also on the same datasets at fixed energy to see the performance achievable every situation.

As shown in Fig. 4.9 performance comparable with the best model in the first analysis (Fig 4.5) are achieved with the 100k dataset (≈ 4 events$/m^3$), actually very low statistic. The performance for the other two datasets are arguable but, considered the low amount of data that was used for training, the results seems promising.

4.6 PERFORMANCE WITH DARK NOISE

Dark noise is a real effect that should be taken into consideration when dealing with real data. It is generated by PMTs unwanted firing because of thermoemission from the first dynode and random fluctuations (See [13]), so it essentially spoils the hit time distribution (Fig. 4.10) and the other spatial informations.

This effect is not taken into account in the simulation and should be added later on manually. A new dataset [7] contains 1 million events with added dark noise along with 10 datasets of 2000 events

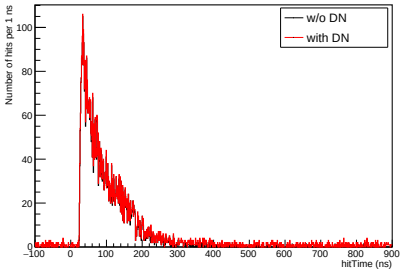
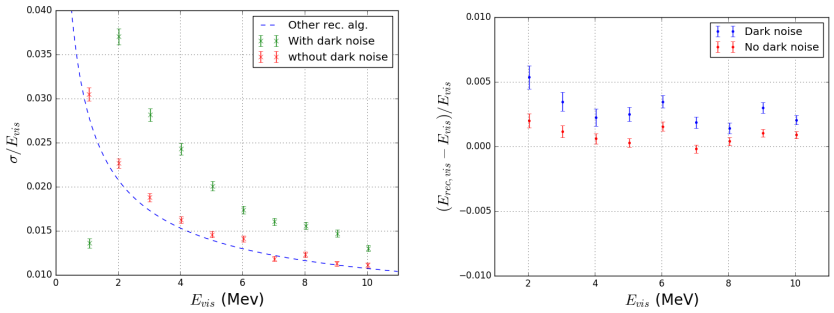


Figure 4.10: An example of an hit time distribution with added dark noise

at fixed energies. The results of the analysis are shown in Fig.4.11: the effect of the DN on the recostruction is considerable especially at low energies and it could be a serious problem even for traditional reconstruction algorithm, positive bias is also introduced and should be further investigated. A solution to mitigate the problem could be adding more informations from the raw output from the PMTs.



(a) *Dark noise performance* (b) *Dark noise bias*

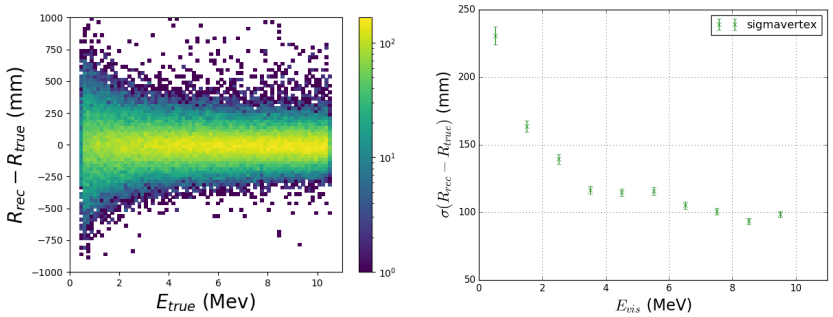
Figure 4.11: Results of the dark noise analysis: the DN spoils the predictions significantly and also adds some bias

5

BDTS FOR VERTEX RECONSTRUCTION: PRELIMINARY RESULTS

The precision measurement of the vertex is also crucial in the JUNO experiment. The Catboost library is very flexible and so the model could be easily adapted to predict the radial component of the vertex for example. The same set of features is used, the only change made is the replacement of the labels of the event to reconstruct, R the true radial component of the event vertex.

Training takes a few more iterations for the loss to settle (≈ 3000). After the training, histogram of the distribution of $R_{\text{rec}} - R_{\text{true}}$ for the new model are plotted along with σ at different energies for the fixed energies datasets.



(a) *Model for radius reconstruction* (b) *Performance of the model*

Figure 5.1: Results for BDTs implementation for vertex reconstruction (radial component)

Results seems promising and show that $\sigma_R \approx 10$ cm for higher energies is already achieved. Further improvement could be made adding more features to the input of the model, for example better precision should be achievable with informations from the individual PMTs, by now only integral informations was used for simplicity. More work should focus on choosing an appropriate loss function for the training (`RMSE` or `MSE`). Full vertex components recostruction is also mandatory for a complete analysis.

6 CONCLUSION

The problem of energy reconstruction in the JUNO experiment was addressed by the means of machine learning. The chosen algorithm, Boosted Decision Trees (BDT), is definitely one of the promising implementations of this paradigm.

Several applications of BDT with the Catboost library [14] were made. Training a model is really easy and not resource intensive as other methods such as neural network. Some suggestion for tuning the hyperparameters was also given in the analysis section.

The plots of the performances at different energies (Fig 4.5) show that, after adding the needed informations from the detectors, BDT works at least on par with the newest traditional (non machine learning) algorithm. There seems to be some bias left in the predictions that needs to be investigated trying to use different loss functions, although the bias is always way less than the experimental uncertainty obtainable with the reconstruction. Target performance of 3.0% σ at 1 MeV seems definitely achievable by now.

A classifier is then built to remove problematic events at the edge of the detector, reducing statistics only by 8 %. Using BDT for this task seems really suited because all of the misclassified events falls really close to the fiducial cut (Fig. 4.8).

The analysis on the minimal quantity of data needed shows that in order to get the expected performance at least a dataset of 100 000 events is needed to be provided (corresponding to ≈ 4 events$/m^3$). smaller datasets give rather questionable results but the densities of datas taken into consideration are quite low. Performance on the dark noise should be compared with other method but the model seems to work fine, although some positive bias is introduced and needs further investigation (see Fig. 4.11).

Finally, considering vertex reconstruction, it has been shown that BDTs give reasonable preliminary results, but further improvements could definitely be done.

BIBLIOGRAPHY

[1] D. Adey et al. "Measurement of electron antineutrino oscillation with 1958 days of operation at Daya Bay." In: (2018). arXiv: 1809.02261.

[2] Fengpeng An et al. "Neutrino Physics with JUNO." In: *J. Phys.* G43.3 (2016), p. 030401. arXiv: 1507.05613.

[3] Alessandro Bettini. *Introduction to Elementary Particle Physics*. 2nd ed. Cambridge University Press, 2014.

[4] Rene Brun and Fons Rademakers. "ROOT — An object oriented data analysis framework." In: *Nucl. Instr. and Meth.* 389.1 (1997), pp. 81 –86. ISSN: 0168-9002.

[5] Tianqi Chen. *Introduction to Boosted Trees*. https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf. [Online; accessed 16-November-2018]. 2014.

[6] Zelimir Djurcic et al. "JUNO Conceptual Design Report." In: (2015). arXiv: 1508.07166.

[7] Yury Malyshkin. Private communication.

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] L. Prokhorenkova, G. Gusev, A. Vorobev, A. Veronika Dorogush, and A. Gulin. "CatBoost: unbiased boosting with categorical features." In: *ArXiv e-prints* (June 2017). arXiv: 1706.09516.

[10] Sebastian Raschka. *Python Machine Learning*. Birmingham, UK: Packt Publishing, 2015. ISBN: 1783555130.

[11] Byron P. Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor. "Boosted decision trees as an alternative to artificial neural networks for particle identification." In: *Nucl. Instr. and Meth.* 543.2 (2005), pp. 577 –584. ISSN: 0168-9002.

[12] M. Tanabashi et al. "Review of Particle Physics." In: *Phys. Rev. D* 98 (3 2018), p. 030001.

[13] A. G. Wright. *The photomultiplier handbook*. Oxford University Press, 2017.

[14] Yandex. *Catboost*. https://tech.yandex.com/catboost/. 2014-2018.