Machine Learning Applied to High Energy Physics

Author: Vanessa Costa Ledesma
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.[*]

Advisor: Carla Marín Benito

(Dated: June 16, 2022)

Abstract: Machine learning algorithms have gained traction in a variety of fields throughout the last decade. This final degree project focuses on a bank problem and on a high-energy physics problem: searching for a rare Λ_b^0 decay. Two different machine learning methods are used: Neural Networks and Boosted Trees, implemented in three different Phython libraries: TensorFlow and Keras, PyTorch and XGBoost. Using the AUC-ROC curve, the models between the three libraries are compared, and finally, models try to predict whether the Λ_b^0 decay happens for a given data. Results for the bank problem shows nearly the same performance for TensorFlow and PyTorch, while XGBoost seems significantly better. For the high-energy problem XGBoost seems better, followed by TensorFlow and last PyTorch. However, predictions made on new data shows similar performance for XGBoost and PyTorch.

Key Words: Classification, Neural Networks, Boosted Trees, AUC-ROC.

I. INTRODUCTION

Machine learning is growing in importance in many sectors due to the enormous volume of data that researchers and enterprises have to deal with. It allows a user with a computer to estimate the tendency of a particular topic with a percentage of reliability. It is focused on building programmes that can learn from data and make decisions or identify patterns using automated optimization methods.

At CERN (Conseil Européen pour la Recherche Nucléaire), researchers need to handle enormous volumes of data produced by the detectors and analyse them to study the process of their interest, hence machine learning algorithms are applied. Some research groups are trying to implement a functional and reliable model in Python, a programming language.

The goal of this project is to study two different machine learning algorithms: Neural Networks and Boosted Trees, on two different problems (bank and pyshics), and their implementation on three different Python libraries: Tensorflow and Keras, PyTorch, and XGBoost. The bank problem is about whether a person subscribes to a term deposit. The high energy physics' problem is about whether the rare Λ_b^0 to proton, kaon, positron and electron decay is produced, in LHCb data. In the final section, it is discussed which model is more reliable for each dataset and which has the easiest implementation.

II. SUPERVISED MACHINE LEARNING

Before discussing what machine learning is and what it entails, it is necessary to distinguish between Machine Learning (ML) and Artificial Intelligence (AI). AI is the science of constructing a machine that can mimic human behavior, such as doing specified tasks or thinking. ML refers to the algorithms that underpin AI, which are concerned with how the computer learns from data and adapts to new; as a result, ML is a specific subset of AI.

Depending on the ingested data, there are three major types of ML algorithms: Supervised Learning (SL), where data is labeled by the input and output; Unsupervised Learning (UL), where data is unlabeled and the task is to find a pattern in the data; and Reinforcement Learning, where the machine (or model) learns by *trial and error*.

This project focuses on SL, namely Neural Networks and Decision Trees, two widely used algorithms. Which is the best is up for argument, with certain references favouring one over the other, and the answer — I would argue — is contingent on the dealed problem.

All algorithms on SL start from the same principles: a mathematical function that establishes a relationship between the output ($\boldsymbol{y_i}$, the predicted element) and the input ($\boldsymbol{x_i}$, the feeded data); the task to be achieved: Classification (discrete answer) or Regression (continous value); the objective function: composed by the loss function (or accuracy), which is a measure of how well the model predicts the training data; and the regularization term which controls the model's complexity. The following steps are defining the model, training it, and testing it by making predictions and evaluating the accuracy.

As an illustration of what the previous paragraph means: a linear model such as $\hat{y_i} = \sum_j \theta_j x_{ij}$, with θ_i denoting the parameters the model adjusts as a result of its training. The objective function can be defined as $Obj(\theta) = L(\theta) + \Omega(\theta)$ where L represents the loss function, and Ω the regularization term. An example of a loss function is the *mean squared error*: $L(\theta) = \sum_i (y_i - \hat{y_i})^2$, the smaller loss the better model. One common regularization term is the *Ridge Regression*: $\lambda \sum_i \theta_i^2$, where λ is the parameter that determines how much the complexity of the model should be penalised [1].

[*] vcostale7@alumnes.ub.edu

The way the model is trained for the classification problem is what makes the difference between the two algorithms applied in this project: Neural Networks and Decision Trees.

A. Neural Networks

Neural Networks (NN) are a biological neural network-inspired algorithm, used for SL and UL problems, made up of a network of neurons-like processing units, called nodes, which are interconnected by layers. The input layers gather input patterns, followed by the hidden layers, and finally the output layers (see Figure 1). The connections between nodes are known as weights, whose values are optimised during the training process and through the hidden layers. Larger weights means that a variable is more relevant to the ouput among others. This output is fed into a function called "activation," which determines whether the data is passed to the next layer, and if it is, the node is called "activated." The purpose of training is to narrow the gap between the predictions and true values by adjusting the weight parameters (initially determined randomly) following a function called "optimiser" and finding the loss funtion's minimum. Weights are repeatedly updated until they converge. [2].

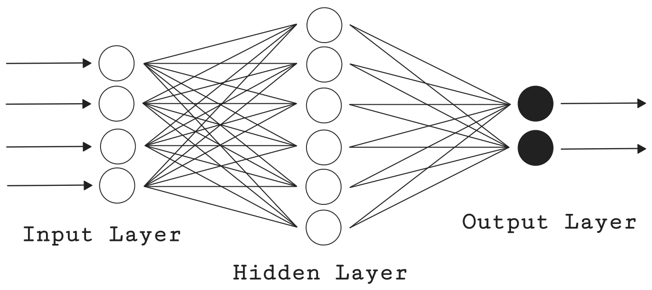

Figure 1: Illustration of a simple neural network, with only one hidden layer. Image by author.

B. Decision Trees

Decision Trees are a SL algorithm, used for classification and regression problems, that assigns a target value for each data sample using a binary tree graph (each node has two children), where the tree leaves show the target values. The sample is propagated via nodes until it reaches the leaf, with each node deciding which descendent node it should move to. A decision is made based on the characteristics of the selected sample, and usually just one criteria is examined (one feature is used in the node to make a decision). The process of discovering the best rules at each internal tree node according to the chosen metric is known as decision tree learning [3]. But this is just the basis of many algorithms like Random Forests, Bagging, and Boosted Decision Trees. This report focuses on the last one, Boosted Decision Trees.

Boosted Decision Trees gathers many Decision Trees (see Figure 2). This algorithm applies the *Boosting* to the Decision Trees classifier, which follows the idea of making a "strong learning algorithm" from a "weak learning algorithm" [4]. In this context, a "weak learning algorithm" is just a Decision Tree. Iteratively, a tree is created considering the error of previous trees, by giving a weight to their accuracy. Missclassified data is given a higher weight, so the following trees will focus on improving it, without forgetting the goal – to reduce the loss function. There are many boosting algorithms, which differ in the way the "weak learners" are added, here, XGBoost (eXtreme Gradient Boosting) is used.

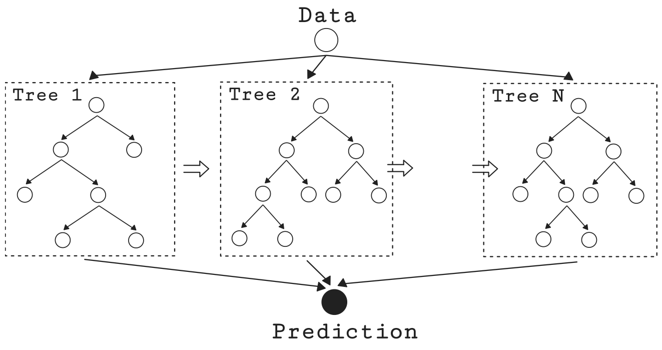

Figure 2: Illustration of a boosted tree. Note that trees may be different. Image by author.

C. Classification Problem and AUC-ROC curve

A classification problem is a discrete output that the model must provide. However, their calculations are not discrete: the model generates the predictions as a probability for each required-predicted class, based on how certain it is that the response is correct. Therefore, there must be a point at which the model determines whether it is a class or not: this is known as a threshold.

The simplest approach to determining how well a model performs is using the AUC-ROC curve (ROC: Receiver Characteristic Operator, AUC: Area Under the Curve). The ROC curve shows the performance of a classification model at all thresholds, hence it displays the True Positive Rate (TPR) in front of the False Positive Rate (FPR), which are defined as:

$$TPR = \frac{TP}{TP+FN} \quad ; \quad FPR = \frac{FP}{FP+TN}$$

where the acronyms read as True Positive (TP), False Negative (FN), False Postive (FP) and True Negative (TN).

At different classification thresholds, the curve displays TPR in front of FPR, from 0 to 1. The higher the area of a ROC curve, the better the model's performance, consequently the AUC is also calculated when making ROC plots.

III. METHODOLOGY

The aim of this project is to examine two datasets and three machine-learning libraries in Python: TensorFlow, Pytorch, and XGBoost.

TensorFlow's and Pytorch's models were extracted from each library's documentation. As these libraries are Neural Networks algorithms, the models used are as similar as the libraries permit: both have only two layers, defined by the ReLu activation function (defined as max(0,x)), the same loss function, Cross Entropy [5] and the same optimiser, Adam [6].

XGBoost's model is a pre-build classifier which presents quite good performance, the `xgb.classifier`, the default parameters are used. The main parameters are: type of booster (can be linear functions), the maximium level of the tree (which increases complexity), maximum leaves, numbers of parallel trees, among many others [1].

AUC-ROC curve is easy to implement on Python using the Scikit-Learn library, since it has a function for this purpose, the `roc_auc_score` [5], which only requires the true values and predictions' probability as input.

A. Bank Afiliation Dataset

The Bank Afiliation datased was extrated from [7]. It contains relevant features to determine whether a person will subscribe to a term deposit, hence it is made with some non-number columns like "job", "marital", "education", "housing", among others. The last one is the target "y", which means whether a person is subscribed (1) or not (0). Notice algorithms described need numeral inputs, therefore some information need to be converted to numbers. In order to train the model, this file is splitted in two parts: one that is used to train, and one that is used to make predictions (the latter is the AUC-ROC curve). The model employs all of the training variables except "y," which it must learn to predict.

B. LHCb Λ_b^0 Decay Dataset

This dataset contains relevant information of the Λ_b^0 (quark content udb) decay to proton (p), kaon (K^-), positron (e^+), and electron (e^-). This decay is produced by the collision of two high-energy p. Actually, it contains two files: the computation approach using Monte Carlo (MC) simulations of the signal process, and the measured signal at the LHCb experiment, which contains real data (real decays of interest and background). This two files can be read and written with the Python's library Uproot [8], using Numpy. The goal is to train the model to predict whether the decay of interest is produced. To reach this, the focus is on the following features:

P Momentum: particles' momentum.

$\mathbf{P_T}$ Transverse momentum: particles' momentum perpendicular to the beam direction.

$\mathbf{IP\chi^2}$ Significance of the smallest distance from the particles' direction to the pp collision.

$\mathbf{FD\chi^2}$ Significance of the flying distance: particles' distance traveled before decaying from the pp collision.

$\boldsymbol{\eta}$ Pseudorapidity: particles' relative angle to the beam axis.

$\boldsymbol{\beta}$ Normalised momentum asimetry: defined as follows:

$$\beta = \frac{p(e^+e^-) - p(p) - p(K)}{p(e^+e^-) + p(p) + p(K)}$$

$\mathbf{DTF\chi NDOF^2}$ Decay tree fitter: information about how well the Λ_b^0 is reconstructed from the remaining particles of the collision.

DIRA: Direction angle: angle between the direction of the Λ_b^0 reconstructed from the children particles' quadrimomentum and the direction formed by the origin and decay vertex of the Λ_b^0. Ideally 0.

The data in MC is the signal used to teach the models that the decay happens. As the mass of the Λ_b^0 is known ($\sim$5.6 GeV), the events above this mass in the real data, can not be part of the decay of Λ_b^0, hence this data is used to train the model that decay does not happen. Again, this information is slipped in two parts: one for train, and the other to calculate the AUC-ROC curve for the models' evaluation.

In addition, as the goal here is to predict whether the decay happens, the data around the mass of the Λ_b^0 is used to make the predictions. Therefore the model will be useful to predict the Λ_b^0 happens on real data (known as generalise), and how it performs.

IV. RESULTS

In this section, the results of the two studied datasets are presented. For the Bank Afiliation dataset only the models between the three machine-learning libraries in Python are compared using the AUC-ROC curve, without applying them on new data. For the LHCb Λ_b^0 Decay Dataset, the models are compared using again the AUC-ROC curve, and finally, whether the Λ_b^0 decay occurs on new data that has never been seen before. Given the past examination and discussion of this dataset [9], one may expect, in an ideal situation, the models utilised in this report to perform similarly. This dataset was big enought for the model used in the reference, and it does have Λ_b^0 decays, therefore examine the discussed models' performance.

A. Bank Afiliation Dataset

Models were trained on 80% of the data and evaluated on the remaining 20%. Figure 3 shows the AUC-ROC curve for the three different libraries.

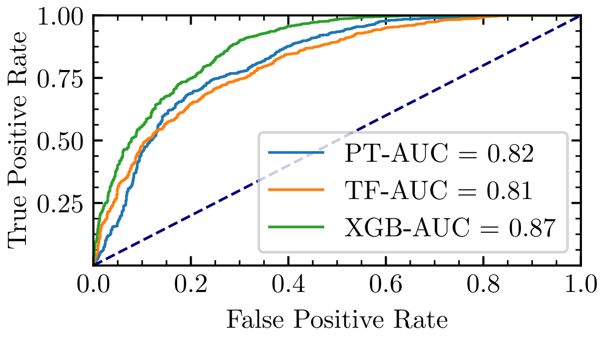

Figure 3: AUC-ROC curve plot for the bank problem. Each line corresponds to one of the three different models applied, see legend.

The AUC score demonstrates that the XGB model may have a better performance, followed by Pytorch and TensorFlow.

It is interesting to point out that the PyTorch's and TensorFlow's ROC curves intersect at a point: at low (< 0.1) FPR TensorFlow is better than Pytorch, but at high (> 0.5) TPR Pytorch is better. This is relevant for the working point of concern, for example, one model predicts less but is more confident in its forecasts, while the other predicts almost all cases.

B. LHCb Λ_b^0 Decay Dataset

Models were trained on 80% of the data and evaluated on the remaining 20%. Figure 4 shows the AUC-ROC curve for the three different libraries.

The AUC score demonstrates that the XGB model has a better performance, followed by TensorFlow and the last one, Pytorch.

Models' efficiency is defined as the quotient of the Real Positive classfied as positive between all Real Positive, and the same definition for the Negative, which gives a number from 0 to 1, and is used as a threshold. In the appendix table I, the figures used to calculate this values can be found for each library. Here, the interest lies in having good background removal efficiency rather than making good predictions for decay.

To test the models, the data bellow 5.8 GeV of mass is used to make predictions. Figure 5 shows the distribution of the invariant mass between 4.2 GeV and 6 GeV contained on the dataset, before any selection applied (background), while Figure 6 shows the distribution of the invariant mass for the events predicted to be signal made by each library at the corresponding threshold. Between this two plots, the magnitude of the peaks have to be highlighted: recall this decay is a very rare decay.

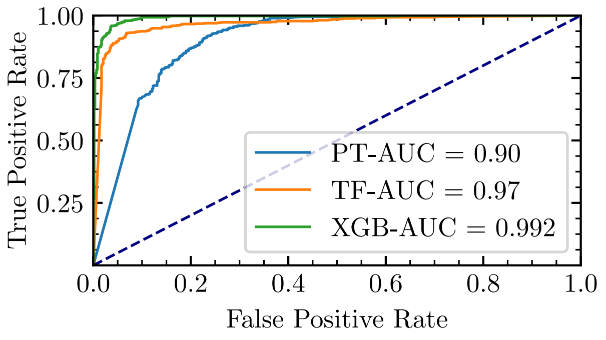

Figure 4: AUC-ROC curve plot for the high-energy physics problem. Each line corresponds to one of the three different models applied, see legend.

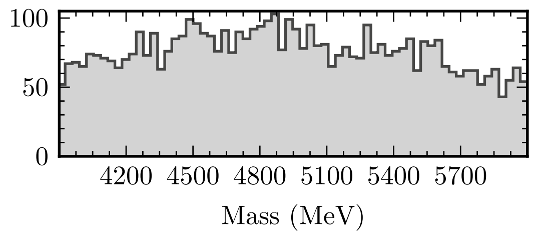

Figure 5: Background distribution between 4.2 GeV and 6 GeV containted on the datased.

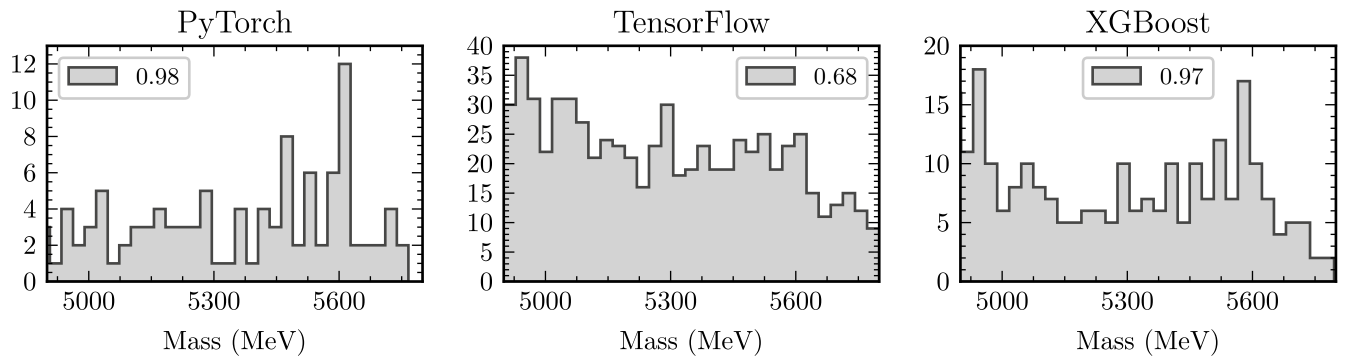

Figure 6: Distribution of the predicted mass, legend indicates the threshold used. More details see appendix.

V. CONCLUSIONS

Regarding the bank problem, XGBoost is better than the Neural Networks algorithms, albeit it is not a surprise since this dataset was acquired from a web page whose example was an XGBoost model. On the other hand, both neural-inspired algorithms have practically the same AUC, despite the fact that TensorFlow seems significantly better at low FPR and PyTorch for high TPR. Given the problem at hand, whether a person subscribes to a term deposit or not, a model that predicts almost all cases is preferable, albeit not a 100% confident model, therefore here, the PyTorch model is significantly better.

Focusing on the Λ_b^0, boosted trees overcome again neural algorithms, however in AUC terms, XGBoost and TensorFlow are not that different. In terms of efficiency, PyTorch outperforms TensorFlow and XGBoost. Observing Figure 6, it can be perceived that TensorFlow is not able to remove as much background as XGBoost and PyTorch. However between the last two, PyTorch seems a bit better since its peak on 5.6 GeV is significantly more notable. Comparing to Figure 5, all three models considerably remove background. Furthermore, contrasting with the previous report [9], it can be appreciated that this data contained roughly 20 decays, and in this project is compatible with this.

Comparing the two problems, bank and physics, using only the AUC-ROC curve —since the bank problem have not been test in new data— it seems models perform better learning physics connections rather than human behavior. It is feasible that more sophisticated models or complementary information are needed to forecast human activity. Maybe from a point of view of a machine, humans' interaction and nature is more caothic than parcicles' interaction.

It is not surprising that for the models used here, XGBoost performs significantly better since the model is prebuilt and optimised, making it extremely simple to use. Nevertheless, the neural models are more flexible at an user level, allowing to add as many type-layers as needed. This flexibility is what makes NN more difficult to implement and optimise, as it requires prior and in-depth knowledge of the topic, but it is also what allows the user to create a better model and adapt it to their problem and data.

The NN models used here have been acquired from their corresponding examples in the documentation, which, for sure, have been optimised for the examples shown there. They are probably not the best for the two examined datasets, but specially for the Λ_b^0 decay, the results have been satisfactory.

Speed calcultion time has not been discussed and studied in dept here, but in a nutshell, both NN models lasts roughly two minutes to compile the Λ_b^0 datased, and XGBoost does not last a minute. A Windows interface with an Intel Core i3 has been used in this project.

A more in-depth study and particular to each library is needed to undoubtedly conclude whether XGBoost overcome Neural Networks.

ACKNOWLEDGMENTS

I would like to earnestly acknowledge the sincere efforts and time given by my advisor, Carla Marín. Without her valuable guidance and weekly feedback, I would not have been able to learn as much as I did on this project. I am also sorry for all the email I sent you, I have been really stressed. I would like to mention the support system and consideration of my parents and sister, who have always been there in my life, and my boyfriend, who has always believed in my abilities and has been really patient with me during my most anxious moments, not a few. Last but not least, my friends Alberto, Arnau, Laura, Marta, the two Marcs, and again my boyfriend David, who have always been here during this degree. It's sad to think this is the beginning of the end of our lunches in this faculty, which are at the order of the days of our friendship.

Without them, I could never had completed this project, and this degree.

Thanks a lot. ♡

[1] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016.

[2] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence*, vol. 40, no. 1, pp. 185–234, 1989.

[3] J. F. T. Hastie, R. Tibshirani, *The Elements of Statistical Learning, Data Mining, Inference and Prediction.* Springer, second edition ed.

[4] C. Rudin, *15.097 Lecture 10: Boosting.* MIT OpenCourseWare OCW, [Web Archive], United States, 2012.

[5] e. a. Lars Buitinck, Gilles Louppe, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[7] B. Wasike, "Machine learning with xgboost and scikit-learn." `https://www.section.io/engineering-education/machine-learning-with-xgboost-and-scikit-learn/`, accessed 2022-03-30.

[8] e. a. Jim Pivarski, Pratyush Das, "scikit-hep/uproot: 3.12.0," July 2020.

[9] C. M. Benito, "Lepton flavour universality tests in electroweak penguin decays at lhcb," 2021.

VI. APPENDIX

This appendix gathers some relevant plots to calculate the effiency for the Λ_b^0 models. Recall effiency is calculated from the quocient of the number of the well-predicted class between the number of the actual class. Plots in this appendix are made as the AUC-ROC curve, with the 20% of the split dataset, which contains 589 signal-data, and 588 background-data. The histograms show the distribution of the predictions' probabilites classified as signal (decay happens), and as background (decay does not happen). Higher probabilities mean models are more confident whether the decay happens, and the opposite for lower. The ideal model is a classification distribution with two peaks: one at 1, and one at 0, meaning its confident about its two classifications. Additionally, the confusion matrix shows again this distribution but in a very compact plot includes the magnitudes of the classified and miss classifed predictions. Notice a ideal model should have the off-diagonal part darker.

Efficiency	Signal	Background
PyTorch	0.98	0.54
TensorFlow	0.68	0.99
XGBoost	0.97	0.95

Table I: Efficiency for all three libraries at each classification work. The values obtained in this table for Signal are the used for threshold in the Figure 6.

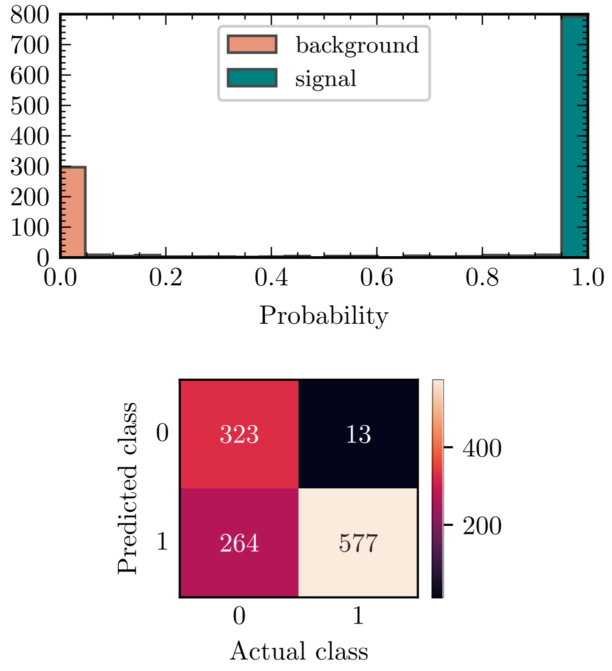

Figure 7: PyTorch's classifications distribution on top, and confussion matrix on bottom, at threshold 0.5.

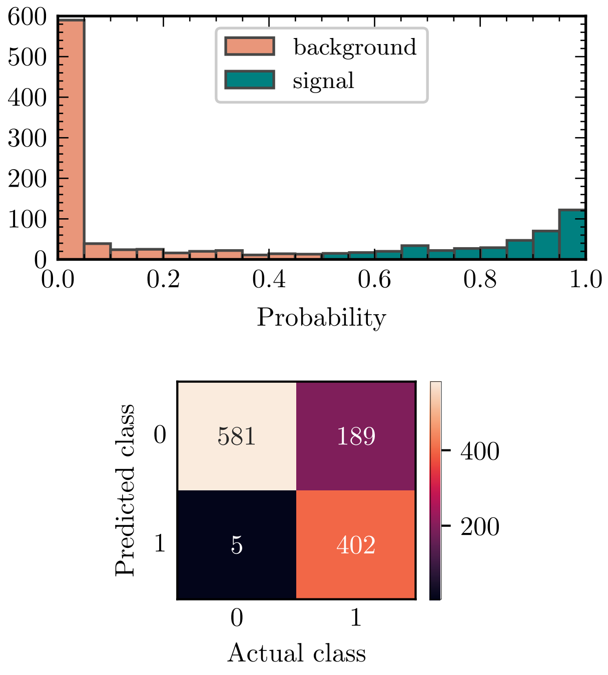

Figure 8: TensorFlow's classifications distribution on top, and confussion matrix on bottom, at threshold 0.5.

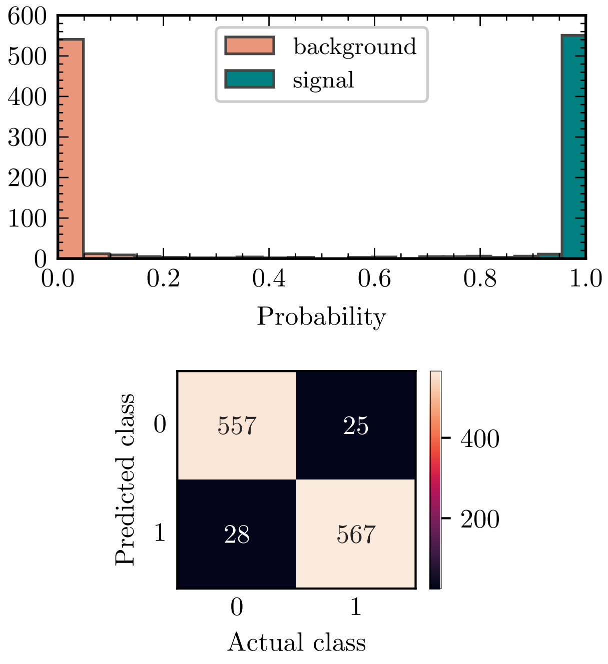

Figure 9: XGBoost's classifications distribution on top, and confussion matrix on bottom, at threshold 0.5.