Politechnika Warszawska

WYDZIAŁ FIZYKI

Praca dyplomowa inżynierska

na kierunku Fizyka Techniczna
w specjalności Fizyka Komputerowa

Implementation of machine learning algorithms for particle identification in the CBM experiment

Julian Nowak
Numer albumu: 298198

promotor:
dr hab. inż. Hanna Zbroszczyk, prof. uczelni

WARSZAWA 2022

Streszczenie

Tytuł pracy: Implementacja algorytmów uczenia maszynowego do identyfikacji cząstek w eksperymencie CBM

W ostatnich dekadach obserwuje się rosnącą popularność algorytmów uczenia maszynowego (ML) w licznych dziedzinach biznesu i nauki. Wiele różnych bibliotek ML zostało stworzonych, pozwalając specjalistom spoza dziedziny informatyki na implementację algorytmów uczenia maszynowego, pozwalają tym samym na np. szybszą analizę dużych, skomplikowanych zbiorów danych, utrzymując przy tym wysoką dokładność. Z tego powodu ML jest też coraz częściej wybierany przez naukowców zajmujących się fizyką wysokich energii. Niniejsza praca opisuje możliwość zastosowania algorytmów uczenia maszynowego do identyfikacji cząstek w eksperymencie CBM.

Modele uczenia maszynowego do rekonstrukcji Kaonów o krótkim czasie życia (z ang. K-short), jak i identyfikacji trzech grup cząstek (jako odpowiednik metody TOF) zostały zaprezentowane w tej pracy, używając symulowanych danych z modeli Monte Carlo, przepuszczonych przez program GEANT4, w którym symulowany jest układ eksperymentu CBM. Przygotowanie, trening, jak i ewaluacja modelu została zaprezentowana, podczas gdy możliwość wdrożenia modeli ML do oprogramowania eksperymentu została poddana dyskusji w podsumowaniu.

słowa kluczowe:
uczenie maszynowe, identyfikacja cząstek, zderzenia ciężkich jonów, CBM, FAIR, GSI

(podpis opiekuna naukowego) (podpis dyplomanta)

Abstract

Title of the thesis: Implementation of machine learning algorithms for particle identification in the CBM experiment

In the last decades, the increasing popularity of machine learning (ML) algorithms has been observed in many different branches of business and science. Multiple ML libraries have been developed, allowing specialists from other fields than computer science easy implementation of these algorithms, which enables e.g., a fast analyse of big, complicated data sets, while maintaining high accuracy. For this reason, ML is gaining popularity in the high-energy physics community as well. In this thesis, the possibility of applying ML algorithms for the identification of particles in the CBM experiment is discussed.

Machine learning models for reconstruction of short-lived Kaons (K-short) and identification of three groups of particles (as a counterpart of the TOF method) have been prepared in this work, using data from Monte Carlo models passed through simulated CBM experiment setup in GEANT4. Model preparation, training, and evaluation have been presented, while the possibility of deploying the ML models in the experiment software has been discussed in the summary.

Keywords:
machine learning, particle identification, heavy-ion collisions, CBM, FAIR, GSI

Oświadczenie o samodzielności wykonania pracy

Politechnika Warszawska

Julian Nowak
298198
Fizyka Techniczna

Oświadczenie

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Warszawa, dnia (data) (czytelny podpis dyplomanta)

Oświadczenie o udzieleniu Uczelni licencji do pracy

Politechnika Warszawska

Julian Nowak
298198
Fizyka Techniczna

Oświadczenie studenta w przedmiocie udzielenia licencji Politechnice Warszawskiej

Oświadczam, że jako autor / ~~współautor~~* pracy dyplomowej pt.

Implementation of machine learning algorithms for particle identification in the CBM experiment

udzielam / ~~nie udzielam~~* Politechnice Warszawskiej nieodpłatnej licencji na niewyłączne, nieograniczone w czasie, umieszczenie pracy dyplomowej w elektronicznych bazach danych oraz udostępnianie pracy dyplomowej w zamkniętym systemie bibliotecznym Politechniki Warszawskiej osobom zainteresowanym. Licencja na udostępnienie pracy dyplomowej nie obejmuje wyrażenia zgody na wykorzystywanie pracy dyplomowej na żadnym innym polu eksploatacji, w szczególności kopiowania pracy dyplomowej w całości lub w części, utrwalania w innej formie czy zwielokrotniania.

Warszawa, dnia (data) (czytelny podpis dyplomanta)

* - niepotrzebne skreślić

Contents

Introduction

Machine learning (ML) is one of the fastest-growing technologies in the last decades. Its applications can be found in different fields of research, and its popularity is growing in high-energy physics (HEP) as well, as HEP experiments produce big amounts of data. The application of ML in this field has been recently appearing in an increasing number of publications, such as ”Generative Models for Fast Calorimeter Simulation: the LHCb case”, in which ML allowed physicists to produce a sufficient amount of simulated data needed by the next HL-LHC experiments using limited computing resources (2019) [1]. It was also used for Λ Hyperon Reconstruction in the CBM experiment (2021) [2] which was a significant inspiration for this thesis.

There are two main goals of this work: using ML for the reconstruction of the short-lived particles (K-short in this example) and identification of three groups of particles (a counterpart of the TOF method), in the planned CBM experiment in FAIR, Darmstadt.

In the first two chapters, the physical motivation, and the CBM experiment are described. In the next chapter, the machine learning concepts are presented. The next chapter presents the traditional methods of the reconstruction and identification of particles. The following chapter describes reconstruction of K-short particles using ML, while the next one presents TOF identification using ML. The last chapter contains a discussion and summary, as well as a description of possible further developments.

Physical motivation

2.1 Standard Model

The Standard Model, which was formulated in the 1970s, describes the elementary particles and the interactions between them. It combines the theory of the elementary particles, quantum mechanics, quantum chromodynamics (strong interactions), and electroweak forces (unified description of the weak and electromagnetic forces). While most of its assumptions were confirmed in the 1980s, the most recent one - the confirmation of the existence of the *Higgs boson*, dates from 2013. [3]

However, the Standard Model is not complete, as it does not describe e.g., the gravitational force, or the new findings such as the existence of the dark matter and the muon's magnetism [4]. While the explanation of these phenomena could prove the theory wrong, it describes well the majority of interactions between matter.

According to the Standard model, there are two main classes of elementary particles: **fermions**, which constitute matter, and **bosons**, which carry interactions. They are presented on Figure 2.1 and described below.

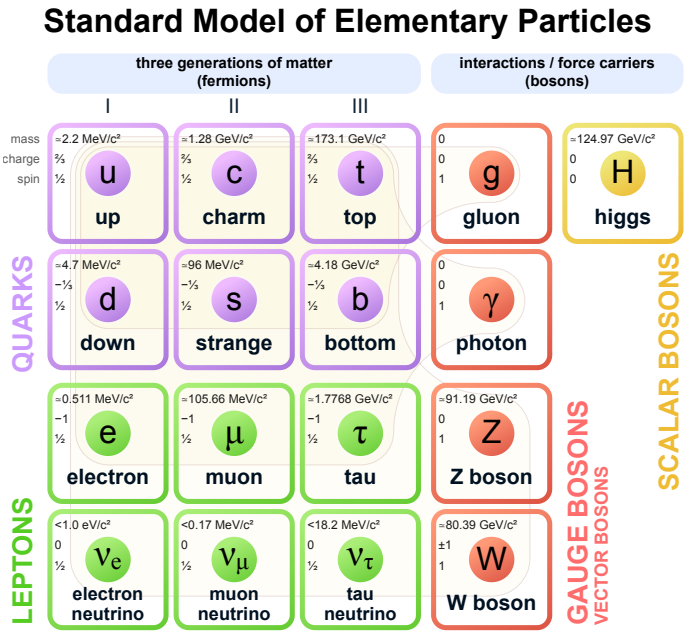


Figure 2.1: Standard Model of Elementary Particles [5]

Fermions follow the Fermi-Dirac statistics (hence the name), they have half odd integer spin and thus follow the **Pauli exclusion principle**. They can be divided into two groups: **leptons** and **quarks**. Leptons, unlike the quarks, have an integer charge number; they do not have the **color charge**, so they do not engage in strong interactions. There are three generations of fermions:

I. quarks: *up (u)* and *down (d)*; leptons: *electron* (e^-) and *electron neutrino* (ν_e)

II. quarks: *charm (c) and* strange (s) ; leptons: *muon* (μ^-) and *muon neutrino* (ν_μ)

III. quarks: *top (t)* and *bottom (b)*; leptons: *tau* (τ^-) and *tau neutrino* (ν_τ)

Bosons follow (*nomen omen*) Bose-Einstein statistics, they have an integer spin. There are five elementary bosons carrying:

- strong forces: *gluons* (g)
- weak forces: bosons $W^\pm$ and Z^0
- electromagnetic forces: *photons* (γ)
- mass: *Higgs boson* (H)

2.2 Quantum chromodynamics

Quantum chromodynamics, a part of the Standard Model theory, describes strong interactions between quarks and gluons. It explains i.a. why quarks are confined in the hadrons[3]. According to this theory:

- there are three color charges: *Red*, *Green*, and *Blue* which are exchanged between the quarks via bosons - *gluons*
- gluons interact with both quarks and other gluons
- the strong interaction has a pulling character, the *color potential* can be described by the formula:

$$V(r) = -\frac{4}{3}\frac{\alpha_s}{r} + kr \tag{2.1}$$

where α_s and k - coupling constants, r - distance. According to the formula 2.1, the magnitude of the force grows with distance, unless the latter is bigger than a certain threshold (as shown on Figure 2.2) - then, a new particle-antiparticle couple is created. This effect, called **quark confinement**, explains why the quarks cannot be separated and form **hadronic matter**. However, if the distance r is very small, and the energy of the system is big enough, there exists another state of matter called **quark-gluon plasma (QGP)** for which the quarks reach asymptotic freedom (asymptotic, as the strong interactions still do not allow the existence of *free* quarks).

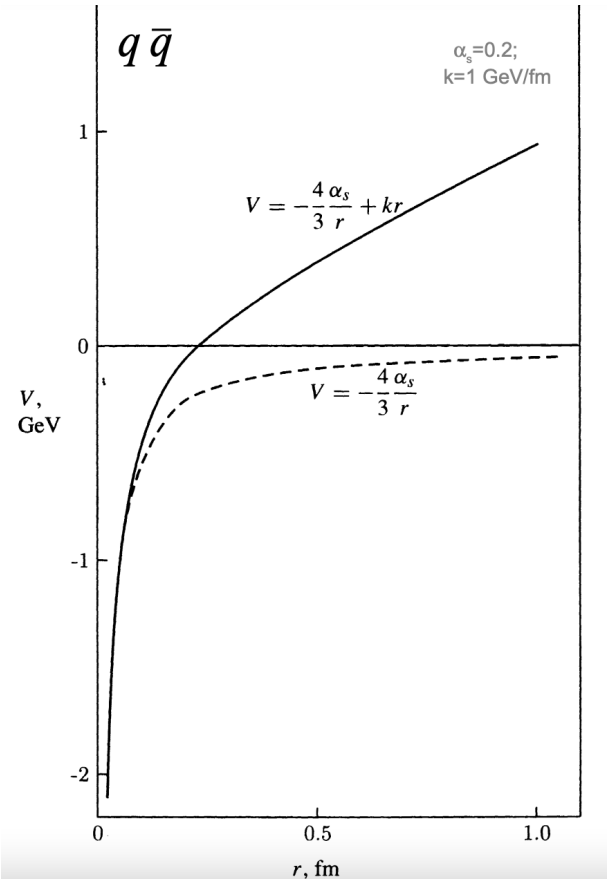


Figure 2.2: Dependence of the color charge potential and the distance between the quarks and gluons. At long distances, the binding energy is too high and a new particle-antiparticle pair is created [6]

As the matter can exist in both states: *hadronic matter* and *QGP*, the phase transition between the two is certain. The phase diagram can be presented on a 2D graph (with net baryon density on the *x*-axis and temperature on the *y*-axis) (Figure 2.3). The QCD matter diagram, especially the phase transition, is investigated by multiple high-energy physics experiments.

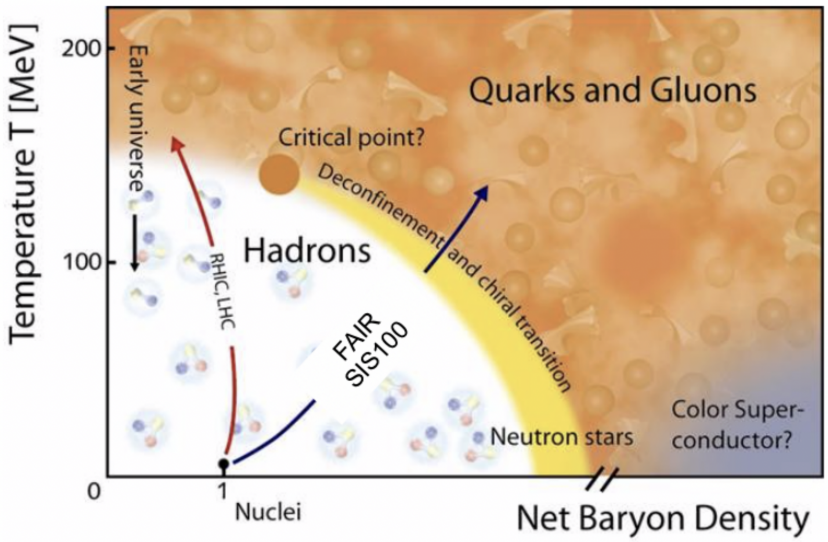


Figure 2.3: Phase diagram of the QCD matter [7]

2.3 Heavy-ion collisions

Heavy-ion collisions at relativistic energies allow achieving both high temperature and high net baryon density, consequently creating *QGP* for a short time. In the *fixed-target* experiments, a heavy-ion is aimed at e.g., a foil; in the *collider* experiments, two heavy-ions are accelerated and then collided, allowing to achieve higher energies.

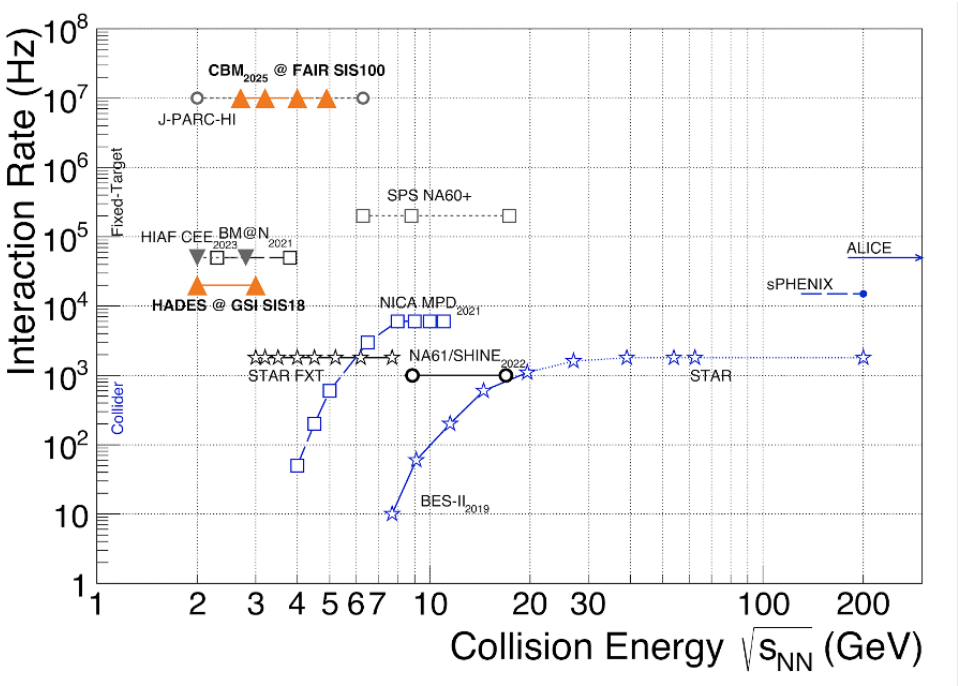


Figure 2.4: The "map" of the heavy-ion collision experiments [9]

Besides the type of setup, the high energy physics experiments also differ from one to another by i.e. achieved temperature, interaction rate, atomic number of the *bullet* (collided) ion, etc. The most important, current experiments are shown on the Figure 2.4. As we see on both Figure 2.3, and Figure 2.4, while the experiments at the Large Hadron Collider (LHC) in CERN aim for bigger collision energies and hence temperatures[10], the experiments performed (or planned) on SIS accelerators in FAIR aim for bigger interaction rates and consequently bigger net baryon density[11].

2.3.1 Reconstruction of short-lived strange particles

In the heavy-ion experiments, the strange quarks are only produced during collisions. Thus, they provide information about the evolution of nuclear matter. It is expected[12] that in the **mixed-phase** (when both nucleon and quark degrees of freedom are present), which can be produced at FAIR energies, the yield of strange particles could be comparable with particles composed of *u* and *d* quarks (as shown on Figure 2.5). To verify this, the reconstruction of strange particles, also short-lived ones like Λ^0, Ξ^-, K^0_S which cannot be detected directly, is important.

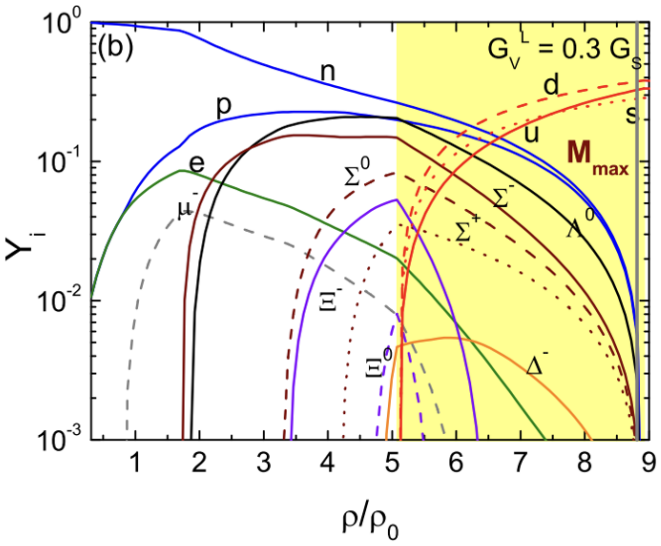


Figure 2.5: Particle population vs density (mixed-phase highlighted in yellow) [12]

FAIR and CBM

3.1 FAIR

FAIR (*Facility for Antiproton and Ion Research*) [13] is an international accelerator facility for the research with antiprotons and ions which is being developed and built in cooperation with international partners, i.a. Poland. FAIR is being built at the GSI Helmholtzzentrum für Schwerionenforschung in Darmstadt, Germany. The existing GSI accelerators will become part of the future FAIR facility and serve as the first acceleration stage [13]. The new accelerator, SIS 100, is foreseen to become operational in 2025 [7]. The map of the existing facilities of GSI, and FAIR (which is still under construction) is shown on Figure 3.1. Several experiments will be held at FAIR: APPA, HADES, NUSTAR, PANDA, and CBM.

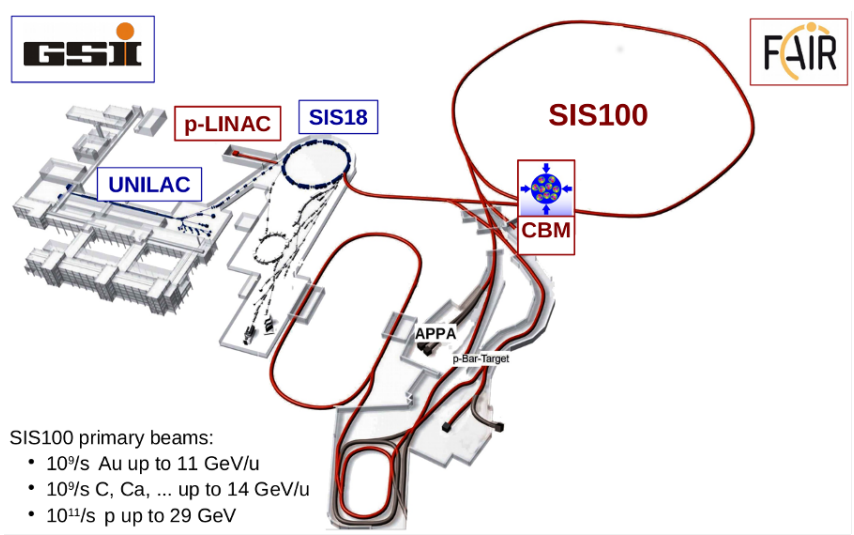


Figure 3.1: Map of FAIR[13]

3.2 CBM experiment

The CBM experiment will be held at the Facility for Antiproton and Ion Research. Its goal is the exploration of the QCD phase diagram in the region of high net baryon densities and moderate temperatures. It will allow i.a. studying the equation-of-state of nuclear matter at neutron star core densities. The measurements will be performed at reaction rates up to 10 MHz. It requires a highly efficient particle identification and reconstruction framework. [14]

3.2.1 CBM setup

In order to identify the particles coming from the collisions, the setup of 8 detectors is under development in FAIR (Figure 3.2).

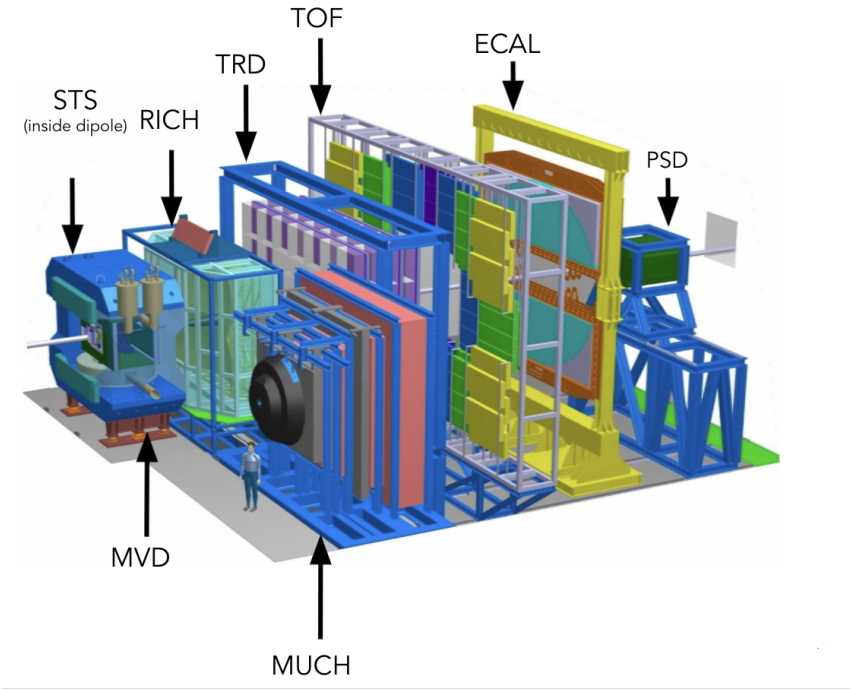


Figure 3.2: CBM setup [7]

The CBM detectors setup will consist of [7]:

- STS - Silicon Tracking System
- MVD - Micro Vertex Detector
- RICH - Ring Imaging Cherenkov Detector, replaceable with:
- MUCH - Muon Chamber System
- TRD - Transition Radiation Detector
- TOF - Time of Flight Detector
- ECAL - Electromagnetic Calorimeter
- PSD - Projectile Spectator Detector.

3.2.2 Data Simulation

As the main CBM accelerator, the SIS100, will not start functioning earlier than in 2025, the Monte Carlo models are handy in simulating the possible results and planning the actual setup of the experiment [7].

The majority of the simulations performed by the CBM Collaboration are performed using two Monte Carlo-based simulation packages: **URQMD** (Ultra relativistic Quantum Molecular Dynamics) [15], and **DCM-QGSM-SMM** (Dubna Cascade Model and Statistical Multifragmentation Model) [16]. Both models treat the production of new particles via the

formation and fragmentation of specific colored objects, strings. The differences between the models arise on different stages of a string formation and fragmentation. [16] A heavy-ion collision is visualised on Figure 3.3.

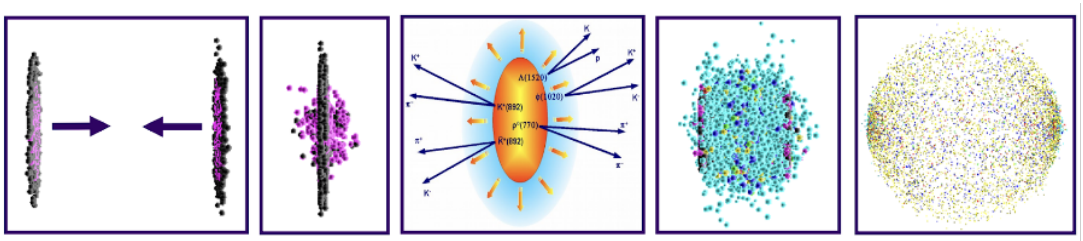


Figure 3.3: Visualisation of heavy-ion collisions in the URQMD model[17]

The data from the Monte Carlo models is later passed through **GEANT4** - a toolkit for simulating the passage of particles through matter [18]. The CBM setup is simulated in it, allowing to recreate the behaviour and work of different detectors and the influence of the construction elements. Finally, the simulated MDV, STS, RICH, TDR, TOF, and PSD hits are reconstructed into tracks and clusters and processed into a special data format, called *Analysis Tree* [19].

Machine learning

Machine learning (ML) is a branch of **artificial intelligence (AI)** and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. The learning part is called training and is performed on training data, the trained algorithm is then tested on test data. Through the use of statistical methods, algorithms are trained to make classifications or predictions [20]. The term "Machine Learning" was first introduced in a paper from 1959, where the computer was trained to play the game of checkers [21]. ML can be divided into three primary categories based on how they learn [20]:

- **supervised learning** - using labeled datasets to train algorithms that classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately
- **unsupervised learning** - using machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention
- **reinforcement machine learning** - behavioral machine learning model that is similar to supervised learning, but the algorithm is not trained using sample data. This model learns using the trial and error method

The most known uses of ML are i.e. speech recognition, computer vision, recommendation engines. Furthermore, it has been also successfully used already in the high-energy physics (more specifically the subcategory of ML, the *deep learning*, which automates much of the feature extraction piece of the process, requiring less human intervention to learn), for tasks as simulating the calorimeter response in the LHCb experiment [1], and inspection of the silicon micro-strips of the STS detector in the CBM experiment [22] (shown on Figure 4.1).

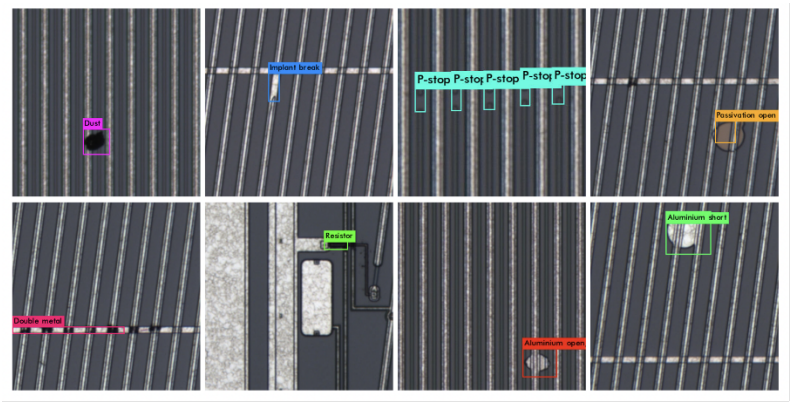


Figure 4.1: Various surface defects on the silicon sensor identified using deep neural network [22]

4.1 XGBoost

XGBoost is a **decision-tree-based, supervised** ML algorithm that uses gradient boosting, available as an open-source package. It has unique tree structure and it offers parallel processing and regularization parameters. [23]. It is highly efficient when used with e.g., tabular data. [2] It combines three elements (also shown on Figure 4.2):

- **Decision trees** - Graphical representation of possible decisions based on certain conditions
- **Random Forest** - Selecting a random subset of features from multiple decision trees and *bagging* - making a decision based on the majority of their predictions
- **Gradient Boosting** - Applying gradient descent algorithm to minimize the errors from random forests to train the algorithm [24]

Contrary to normal decision trees, it returns the **probability** of a certain decision (not a 0-1 decision) and trains itself given the error of its predictions (using mentioned before *gradient boosting*).

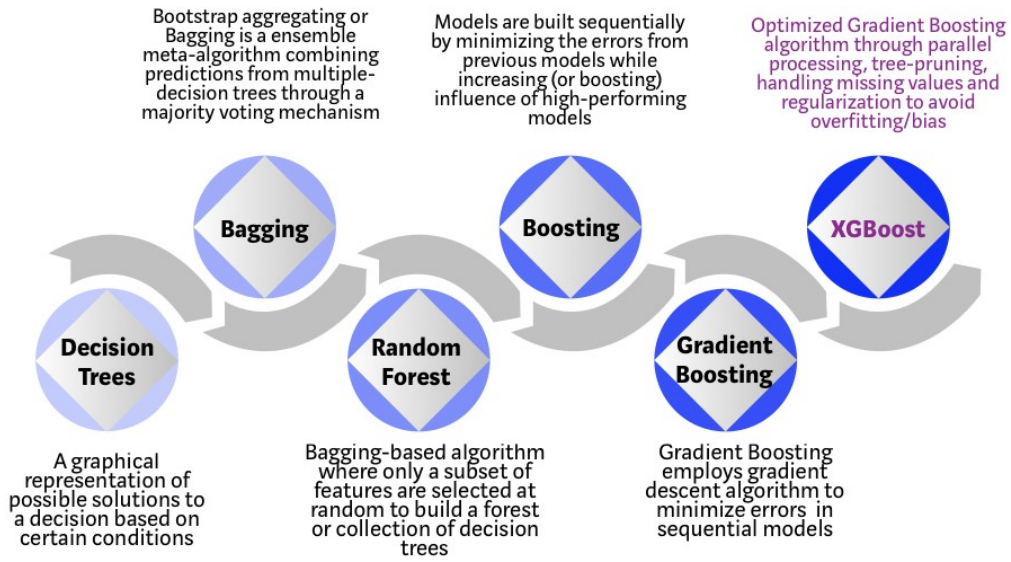


Figure 4.2: Evolution of XGBoost Algorithm from Decision Trees [24]

Each XGBoost model (and in general almost all ML models) has **hyperparameters** - parameters that describe the model itself (and have a big impact on its results). The most important ones, which will be changed in this work are (in xgboost version: 1.3.3) [25]:

- **max_depth** (default value = 6) - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit
- **gamma** (default = 0) - Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be
- **alpha** (default = 0) - A L1 regularization term on weights. Increasing this value will make the model more conservative
- **learning_rate** (default = 0.3) - Step size shrinkage used in update to prevent overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative
- **n_estimators** (default = 6) - Number of gradient boosted trees. Equivalent to number of boosting rounds

There are multiple ways to choose the optimal configuration (values) of hyperparameters, notably Grid Search, Random Search, and Bayesian Optimisation [26]. The comparison between them is shown in Table 4.1 (the +/- sign means that time needed is shorter than in Grid Search, but longer than in Bayesian Optimisation). Following the comparison and the method chosen earlier by the CBM-ML group [2], **Bayesian Optimisation** will be used in this work.

Table 4.1: Comparison of hyperparameters optimization methods

	Grid Search	**Random Search**	**Bayesian Optimisation**
Checks every configuration	+	-	-
Not much time needed to find the optimal configuration	-	+/-	+
Learns with each step	-	-	+

Traditional methods of particles identification

In this chapter, the traditional methods of particles reconstruction and identification (without use of ML) are described.

5.1 Particle identification using the time-of-flight method

The time-of-flight method (TOF) allows to differentiate between the groups of particles such as pions, kaons, and protons (positrons). Using the reconstructed:

- momentum (p) and charge (q) (from the **STS** and **MVD** detectors)
- time of flight (t) and length of the track in the detector (L) (from the **TOF** detector)

the mass-squared of a reconstructed particle can be calculated following the formula:

$$m^2 = \frac{p^2}{c^2} \cdot \left(\frac{c^2 t^2}{L^2} - 1 \right) \tag{5.1}$$

where c - velocity of light in vacuum. Making a 2D plot, showing m^2 on the *y*-axis, and $p \cdot q$ on the *x*-axis, we receive a plot such as in Figure 5.1. To further differentiate between the groups of the particles, Gaussian functions can be fitted to the distributions of the mass-squared. However, the last step is a non-trivial task, causing the majority of the inaccuracy, especially for particles whose mass-squared is far from the mean m^2 value of their group. While several solutions to this problem, such as multidimensional fitting (using more reconstructed variables) exist, the application of ML algorithms will be discussed in this work.

Other groups of particles such as e.g., muons and electrons, can also be reconstructed using the data from other detectors (MUCH and RICH respectively), but it goes beyond the scope of this thesis.

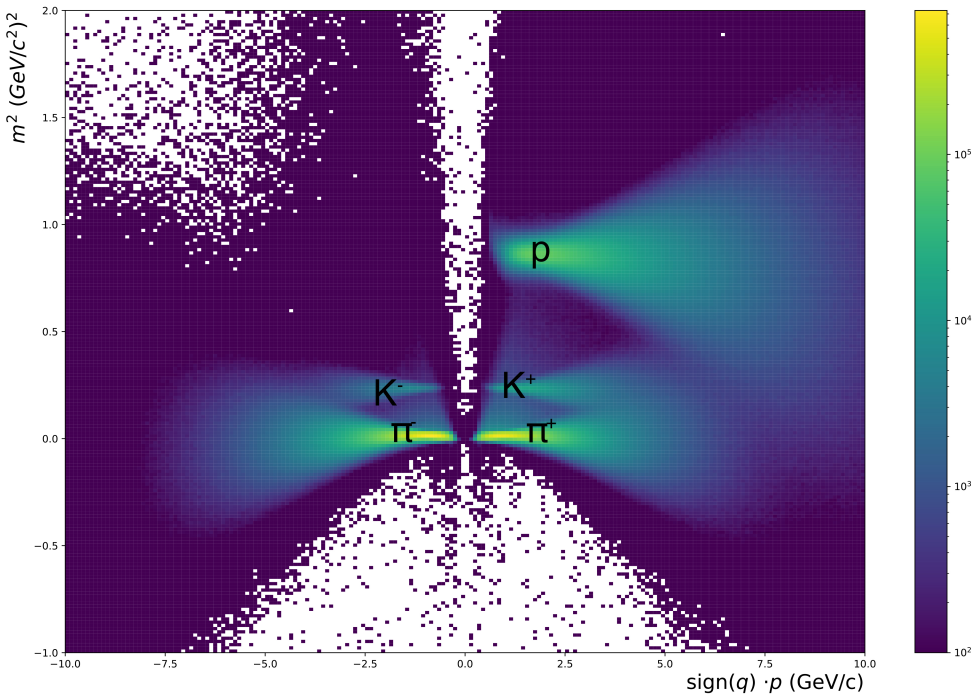


Figure 5.1: 2D TOF plot with pions, kaons, and protons shown

5.2 Short-lived particles reconstruction

Short-lived particles like Λ^0, and K_S^0 have a neutral charge, therefore they cannot be detected directly using e.g., the TOF method. However, we can reconstruct these particles by investigating their *daughter particles* - particles coming from their decays. For example, in the main decay mode (66.7%) a K-short particle decays into a positive and negative pion (Figure 5.2). Using i.e. the STS, we can reconstruct the tracks of pions (identified earlier using e.g., TOF method) coming from this decay, and thus reconstruct the K-short particle.

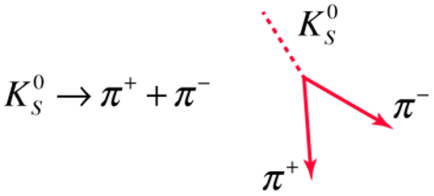


Figure 5.2: K_S^0 main decay mode [28]

In order to do so, the **KFParticleFinder** was developed [29]. It is an on-line optimized particle reconstruction package based on Kalman Filter mathematics. It finds pairs of positive and negative pions (in the K_S^0 case) which could be a result of the *mother* particle's decay.

5.2.1 PFSimple

For offline selection optimization and analysis, **PFSimple** package was created [30]. Using it, one can optimize selection criteria to differentiate between:

- **signal** - pion pairs created in K-short decay
- **background** - pions pairs returned by KFParticleFinder which are not the result of K-short decay

Also, PFSimple returns these attributes of each reconstructed particle (visualised on Figure 5.3):

- **L/ΔL** - distance between primary and secondary vertex divided over its error
- **DCA** - distance of closest approach between pion tracks
- χ^2 **geo** - dimensionless distance of closest approach between pion tracks
- χ^2 **topo** - dimensionless distance of closest approach between extrapolated kaon trajectory and primary vertex
- χ^2 **prim** - dimensionless distance between extrapolated secondary track and primary vertex
- $\cos(\alpha)$ - cosine of angle between pion and K-short momenta

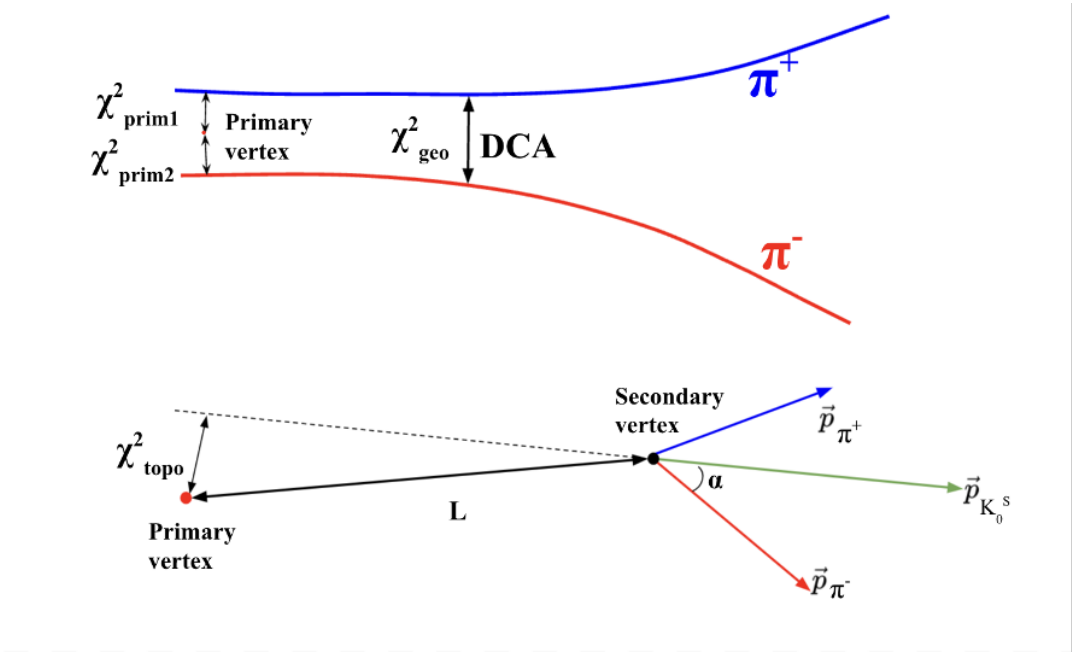


Figure 5.3: Decay scheme and topological variables of a K^0_S decay in the PFSimple

Optimization of selection criteria

As the KFParticle Finder (and PFSimple) return all the possible pion pairs, which could be a result of K-short decay, some selection criteria need to be applied to differentiate between the signal and background.

The manual selection criteria can also be visualised with a decision tree (Figure 5.4):

- For each variable of a reconstructed particle, a conditional statement is being set

- If all conditions are fulfilled, a K-short candidate is treated as a signal candidate

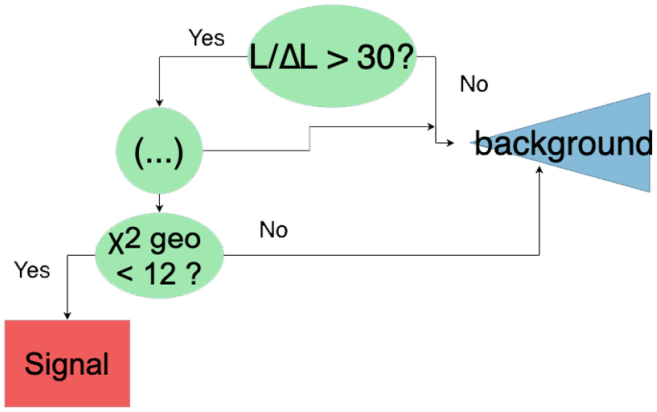


Figure 5.4: Example of a decision tree using manual selection criteria

The manual procedure for selection criteria optimization [30] is as follows:

1. The goal is to suppress as much background and to preserve as much signal as possible
2. Plot distribution of signal and background for some variable (e.g., on Figure 5.5), and select a point above which all entries are considered as signal or background
3. Repeat it for all the variables

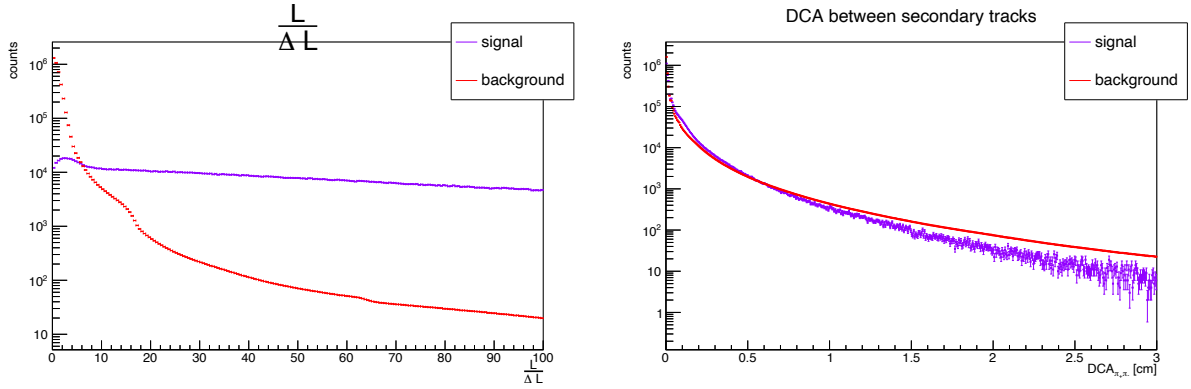


Figure 5.5: Example of distributions to be checked with the manual selection criteria optimization

However, this task is non-automatized (must be repeated for each particle), collision simulating model dependent, and only linear (as one variable at a time is used). In this work, the application of ML algorithm for the optimization of selection criteria will be discussed.

Optimizing K_S^0 reconstruction using ML

In this chapter, the application of the ML algorithms for the optimization of selection criteria for K_S^0 reconstruction (and thus proper identification) will be discussed. As opposed to the traditional method of optimization of selection criteria, mentioned in the chapter before, the predicted advantages and disadvantages are presented in Table 6.1.

Table 6.1: Advantages and disadvantages of ML selection criteria optimization

Advantages	Disadvantages
Non-linear in multi-dimensional space	Not easily interpretable
Automatization of the selection process	Computationally expensive
Partially collision simulating model-independent	
With *probability* a value from which we consider a particle as a K-short particle can be chosen; it can aim for better reconstruction efficiency or a better background reduction	

The accuracy of these predictions will be checked in this chapter. The results of the K-short reconstruction optimization have been presented before in [31] (some fragments of this report will be presented in the following sections).

6.1 Model preparation

The CBM collaboration stores its analysis data in a storage efficient format called an AnalysisTree. For this analysis, it is later passed through the PFSimple (C++ with ROOT framework), which combines positive and negative pions tracks (possible K-short's daughter particles pairs) and returns variables associated with these tracks, from which we could reconstruct a K-short particle. All the possible pairs along with the variables (described in the chapter before) are available for training in the *PlainTree* format, which can be loaded into Pandas Dataframe (Python), using a function prepared by the CBM-ML group [32][33]. Version 3.7 of Python was used, with the following libraries:

- pandas (version 1.3.4)
- xgboost (version 1.3.3)
- scikit-learn (version 1.0.1)

- bayesian-optimization (version 1.1.0)

A part of the code used for the ML model training is shown in Appendix A.

6.1.1 Data enriching

There are two problems to tackle before providing datasets to our ML model: underrepresentation and dependence of collision simulation model.

Dependence of collision simulation model

To make the ML model partially independent of the collision simulation model, select:

- primary Ks candidates only in 5σ region: $0.43485 - 0.56135$ (GeV/c^2) from DCM-QGSM-SMM model, (simulation data)
- background outside 5σ region from UrQMD model, which mimics experimental data (will be replaced by real experimental data while CBM experiment starts)

The selection process is shown on Figure 6.1. Also, the UrQMD model is being used as a test dataset (as the model does not "know" the signal candidates from this MC model).

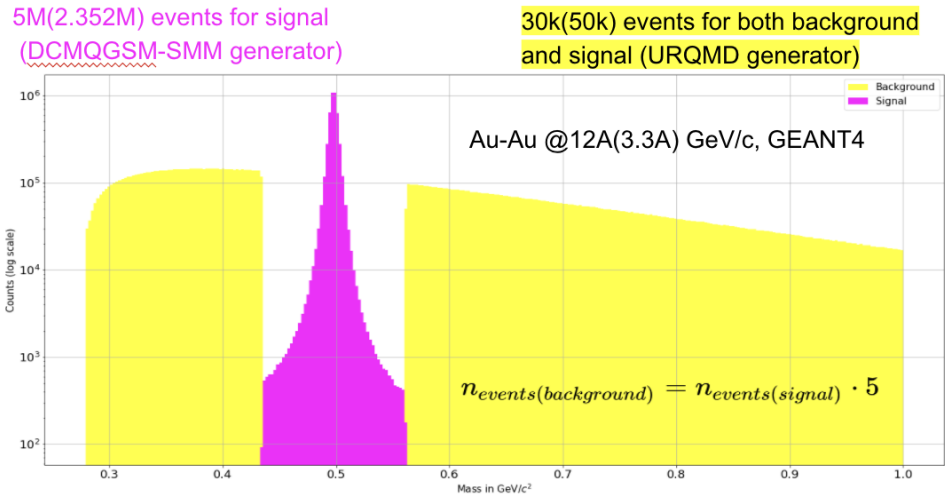


Figure 6.1: Data enriching

In total, the following datasets will be used:

- 5M (2.352M) events for signal generated in DCM-QGSM-SMM
- 30k(50k) events for both background and test dataset generated in UrQMD

Au-Au @12A(3.3A) GeV/c passed through CBM setup in GEANT4, and PFSimple.

Underrepresentation

As in the simulated data, less than 0.1% of K-short candidates are signal candidates, the ratio in the training dataset has to be changed so that the model learns how to identify signal candidates well. Using the trail and error method, it is decided that the number of background

entries is rescaled by this formula:

$$n_{\text{events(background)}} = 5 \cdot n_{\text{events(signal)}} \tag{6.1}$$

6.1.2 Data cleaning

To reject the numeric values of parameters that do not have physical sense, but are present in the data set, some selection criteria are applied before the beginning of the model training. Similarly, some values which might be possible but their uncertainties are big are rejected, to reduce the amount of data.

Invariant mass

As the K-short particle decays into two pions, its invariant mass cannot be smaller than the mass of the two pions, so:

$$m_{\text{inv}} > 0.279 \text{ GeV}/c^2$$

Also, to reduce the amount of data:

$$m_{\text{inv}} < 1 \text{ GeV}/c^2$$

Distances and *x, y, z* coordinates

Distance between the primary vertex (the point where the collision of the nuclei happens), and the secondary vertex (the extrapolated point where the two daughter particles should have crossed each other) - l and the distance of closest approach between the two pions - DCA - cannot be smaller than zero:

$$DCA,\, l,\, \frac{l}{\Delta l} > 0$$

Also, due to the sizes of the tracking system (the largest station has an area of 1m^2):

$$DCA < 100 \text{ cm}$$

For the same reason:

$$|x|, |y| < 50 \text{ cm}$$

As the particle has to hit 3 stations of the tracking system, and the last two are placed above 80 cm:

$$l < 80 \text{cm}$$

For the same reason, and because of the fixed target geometry of the detector:

$$-1 \text{ cm} < z < 80 \text{ cm}$$

To reduce the data, we set:

$$\frac{l}{\Delta l} < 15000$$

However, in the KFParticle package l is assumed to be signed by design, and one can notice that actually, some data entries have a negative value of distance, for both signal and background. As the *quality cuts* should be rather conservative, the following ranges are set:

$l >$ -5 (cm)
$\frac{l}{\Delta l} >$ -25

Momentums

The fixed target geometry of the detector requires that:

$$p_Z > 0 \text{ GeV/c}$$

To reduce the data:

$$p < 20 \text{ GeV/c}; p_T < 3 \text{ GeV/c}$$

Chi square

Since χ^2 is a squared distance, all the values must be larger than zero:

$$\chi^2 > 0$$

To reduce the data, following the maximal values are selected:

- χ^2 first and second $< 3 \cdot 10^7$
- $\chi^2_{geo} < 10000$
- $\chi^2_{topo} < 100000$

Pseudorapidity

Pseudorapidity in terms of the polar angle can be defined as:

$$\eta = -\ln \tan(\frac{\theta}{2}) \tag{6.2}$$

where θ - polar angle, and the STS covers the polar angles between 2.5° and 25°, for which the pseudorapidity values would equal:

$$1.5 < \eta < 3.82$$

However, due to the magnetic field, the pseudorapidity is constrained to the following values:

$$1.0 < \eta < 6.5$$

with which 0.06% of data for signal is lost (instead of 5.66%) and 0.08% of data for background (instead of 6.75%)

6.1.3 Variables selection

The last step before training our model is the selection of variables that will be used in the training of the discriminator. To make the model simpler, there is no need to use multiple variables if they are strongly correlated with each other. Also, to avoid signal/background classification directly by invariant mass, variables strongly correlated with the invariant mass of background should be omitted.

Correlation matrix

The Pearson correlation coefficient (which shows linear correlation) of each variable is being calculated following this formula:

$$\rho = \frac{\mathrm{COV}(X, Y)}{\sigma_X \times \sigma_Y} \tag{6.3}$$

where:

$$\mathrm{COV(X, Y)} = \mathrm{E}\left[(X - \mathrm{E}\left[X\right])\left(Y - \mathrm{E}\left[Y\right]\right)\right] \tag{6.4}$$

$$\sigma_X = \sqrt{\mathrm{E}\left[(X - \mathrm{E}\left[X\right])^2\right]} \tag{6.5}$$

They are then plotted in a matrix shown on Figure 6.2 (where chi2... means χ^2..., distance - DCA, loverdl - $\frac{l}{\Delta l}$). For example e, it shows that there is no need to use *rapidity*, p, p_T and p_z for training, as those four variables are highly correlated ($\rho > 0.5$).

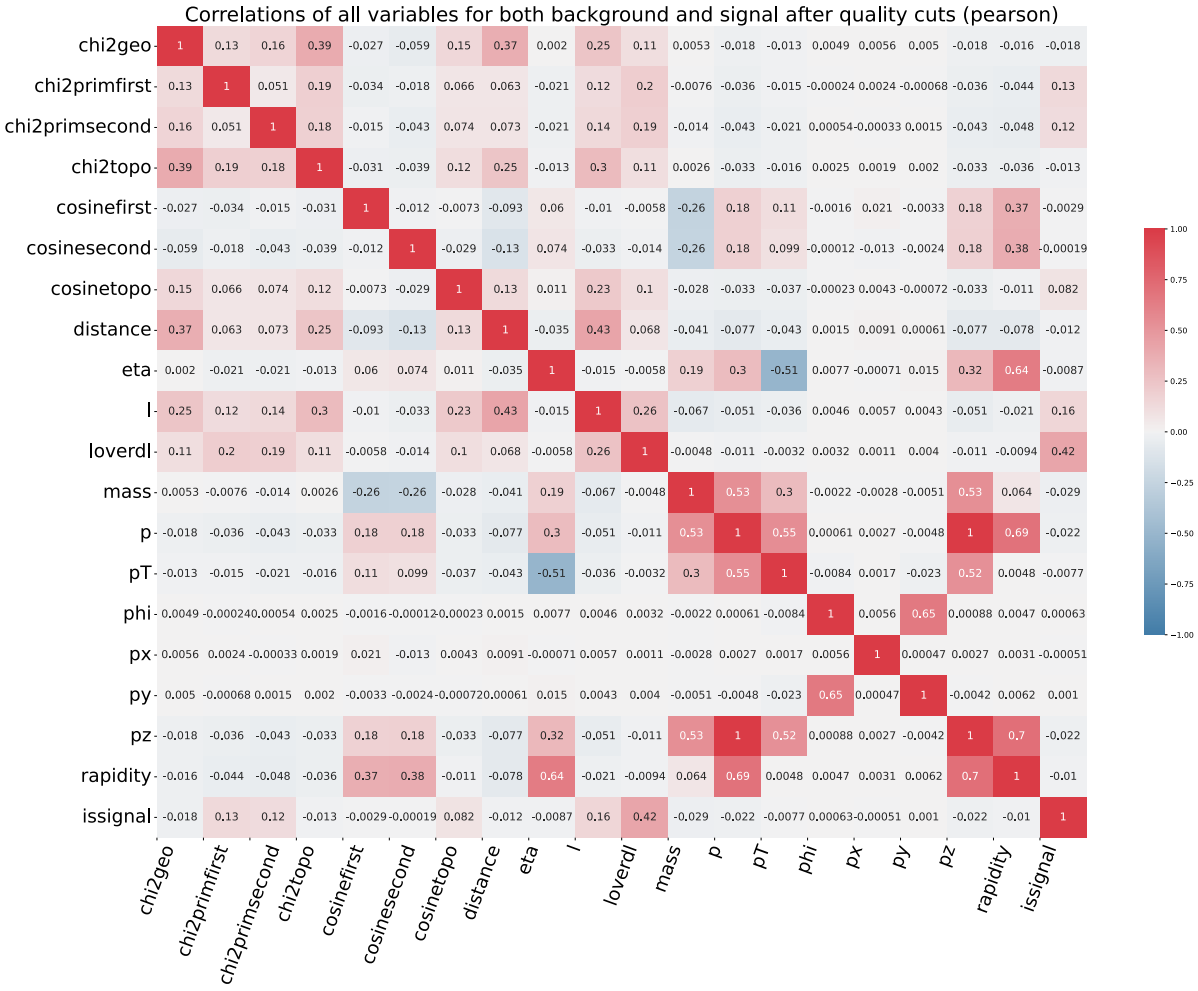


Figure 6.2: Correlation matrix

Correlation with invariant mass

The correlation (Pearson coefficient) of each variable with the invariant mass of (separately) signal and background is being checked (plotted on Figure 6.3).

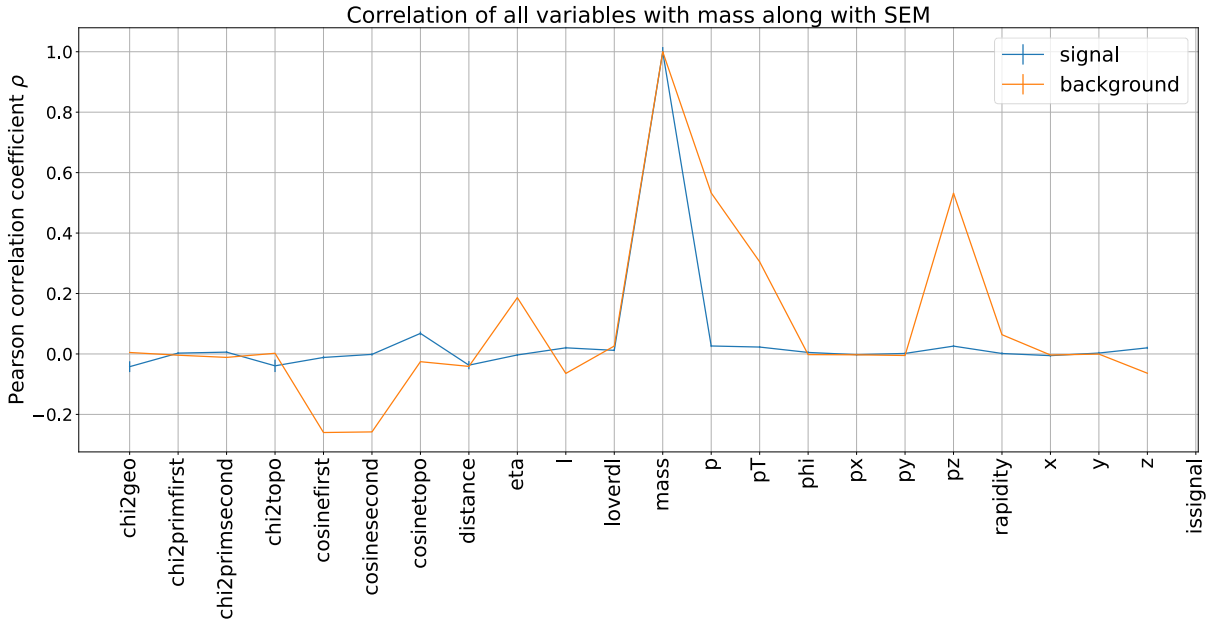


Figure 6.3: Correlation of all variables with mass

Selected variables

Based on that, the following 6 variables are selected for the training: $L/\Delta L$, DCA; χ^2_{geo}, χ^2_{topo}, $\chi^2_{primfirst}$, $\chi^2_{primsecond}$

6.2 Model training

The model will be trained and tested differently on two collision energies of the CBM experiment: p_{beam} = 12A GeV/c (described in the first subsection) and p_{beam} = 3.3A GeV/c (described in the second subsection).

6.2.1 Results for p_{beam} = 12A GeV/c

Having prepared the data, the training of the ML algorithm may be initiated. To choose the hyper-parameters values of the XGBoost model, the Bayesian Optimisation package was used. Along with splitting the train dataset into two equal parts: one for actual training, and the second for validation, the aim is to omit overfitting of the model on the training data.

Validation on test dataset

To check if the model works as well on the test data, as on the training dataset, Receiver Operating Characteristic (Figure 6.4) and probability plots (Figure 6.5) are drawn.

Receiver Operating Characteristic illustrates the diagnostic ability of a binary classifier. Threshold on the ROC (Receiver Operating Characteristic) curve which maximizes Approximate Median Significance

$$\text{AMS} = \sqrt{2}[(tpr + fpr)\log(1 + tpr/fpr) - tpr] \tag{6.6}$$

(where $t(f)pr$ is true (false) positive rate) on the test sample is the best threshold.

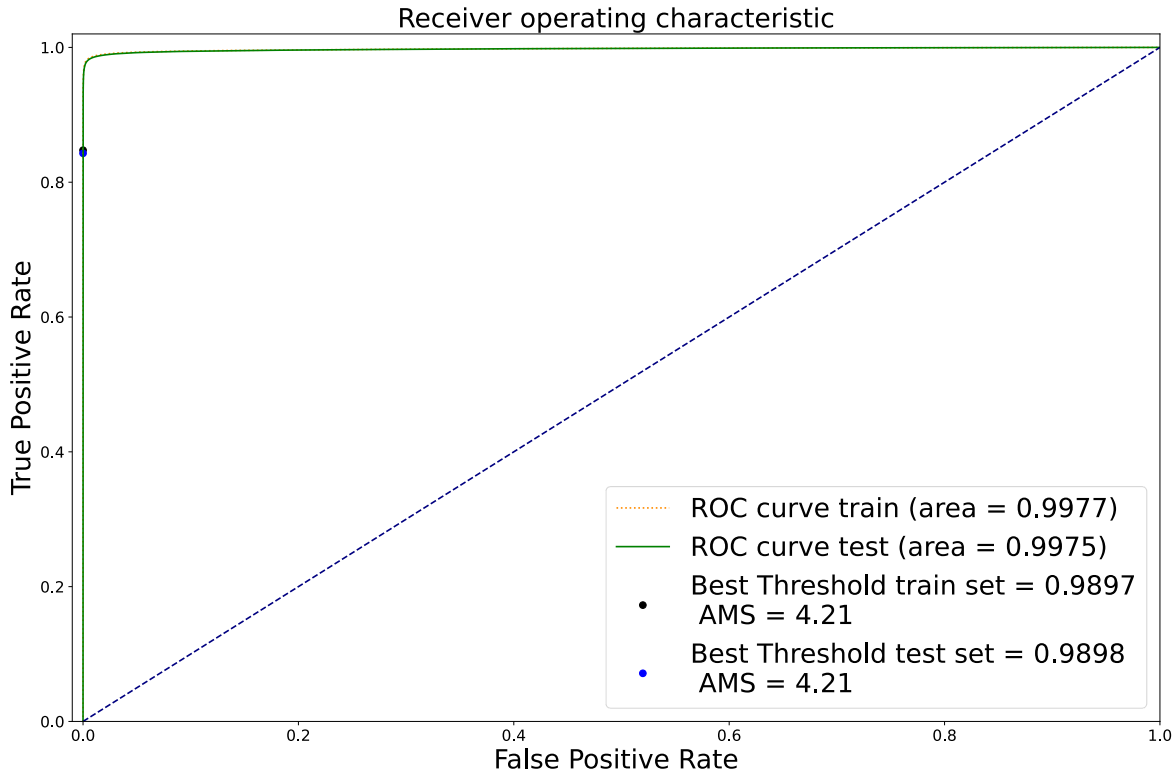


Figure 6.4: Receiver Operating Characteristic

We see that the optimal point on ROC is similar for both train and test datasets. Based on signal/background share in both train and test datasets from the probability plot (Figure 6.5), it is observed that the results are similar for the two datasets. Hence, we can say that our model is not overtrained (works well on new data).

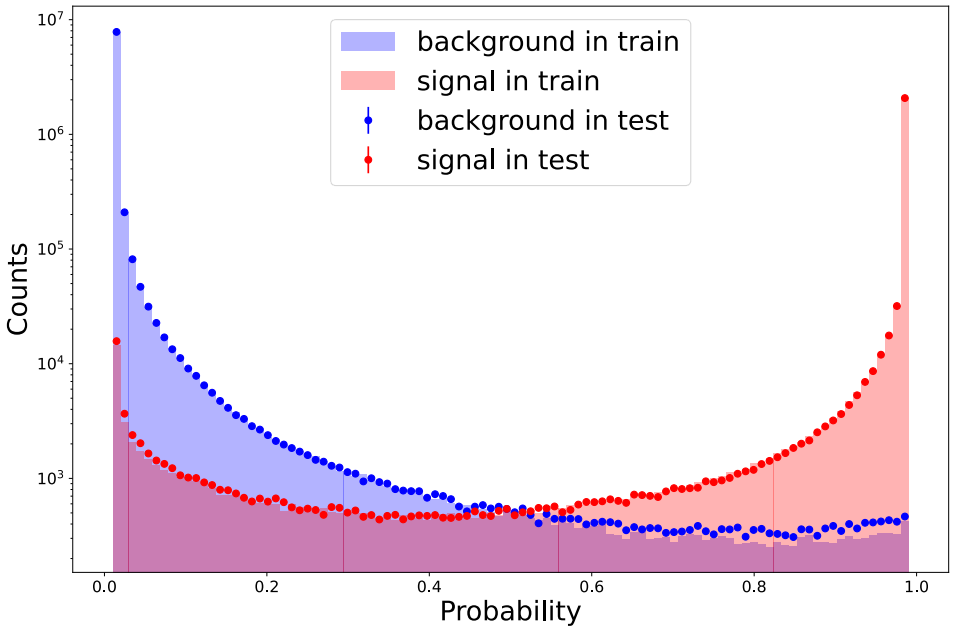


Figure 6.5: Probability plot

The invariant mass distribution before and after XGB selection is presented on Figure 6.6:

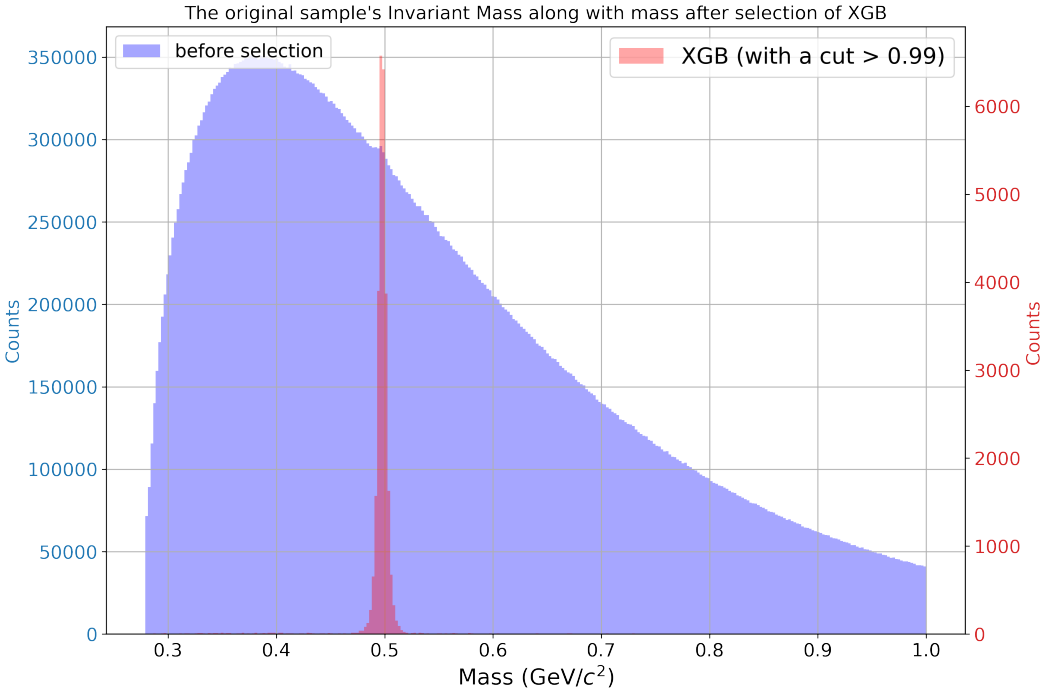


Figure 6.6: Invariant mass distribution

Comparison with default KFPF Cuts

KFParticleFinder has default selection criteria, which can be compared with ML selection:

- $L/\Delta L > 5$
- $DCA < 1$ cm

- $\chi^2_{geo} < 3$
- $\chi^2_{prim} > 18.4$
- $\cos(\alpha) > 0$

Depending on the cut value, better background reduction or better efficiency can be obtained.

With a cut value set to 0.99, we get $50\times$ less background (Fig. 6.7):

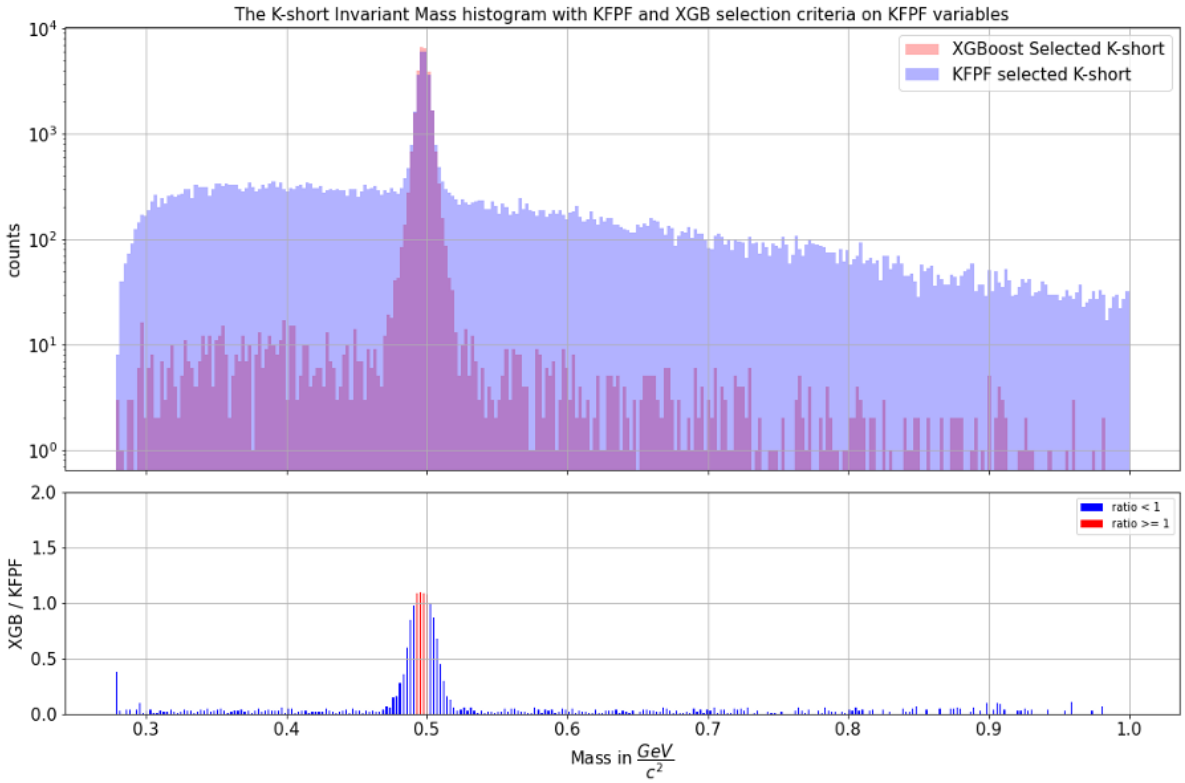
- **Reconstructed K^0_S/ reconstructible K^0_S = 80.67%** vs. 76.93% with default KFPF cuts
- **false / true positive rate = 0.04** vs. 2.01 with default KFPF cuts

With a cut value set to 0.86, we get 20% better efficiency (Fig. 6.8):

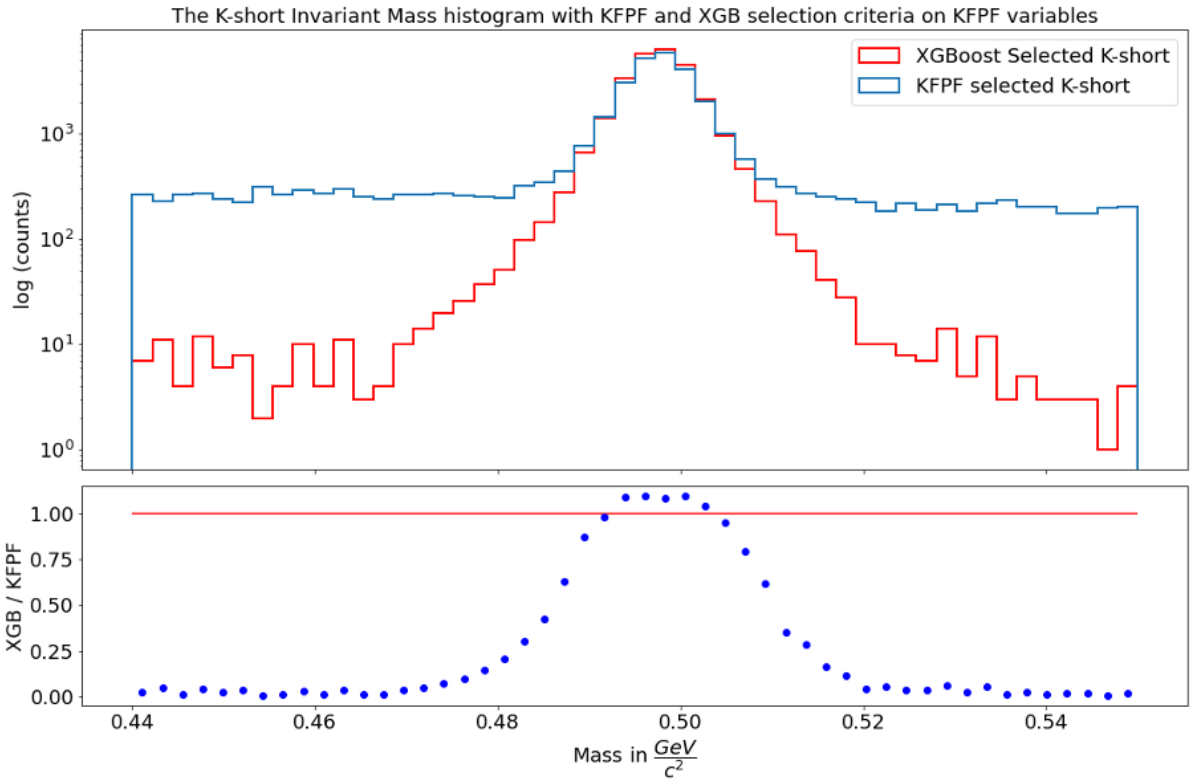
- **Reconstructed K^0_S/ reconstructible K^0_S = 95.91%** vs. 76.93% with default KFPF cuts
- **false / true positive rate = 1.27** vs. 2.01 with default KFPF cuts

Investigation of potential bias

To check if the XGB selection criteria cut tails of the distribution, or are biased in some regions, the invariant mass distribution of true and false positives is plotted (Figure. 6.9). We see that our model does not seem to be biased towards any invariant mass region.

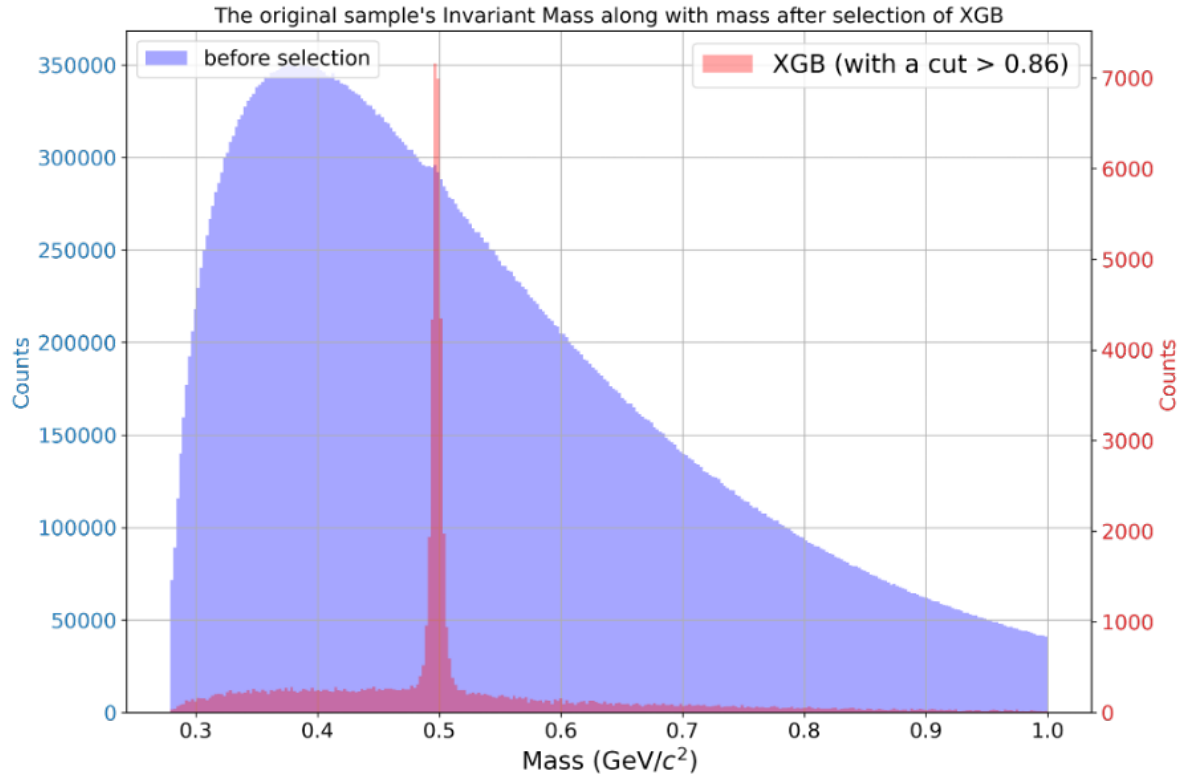


(a) Invariant mass distribution (y log scale)

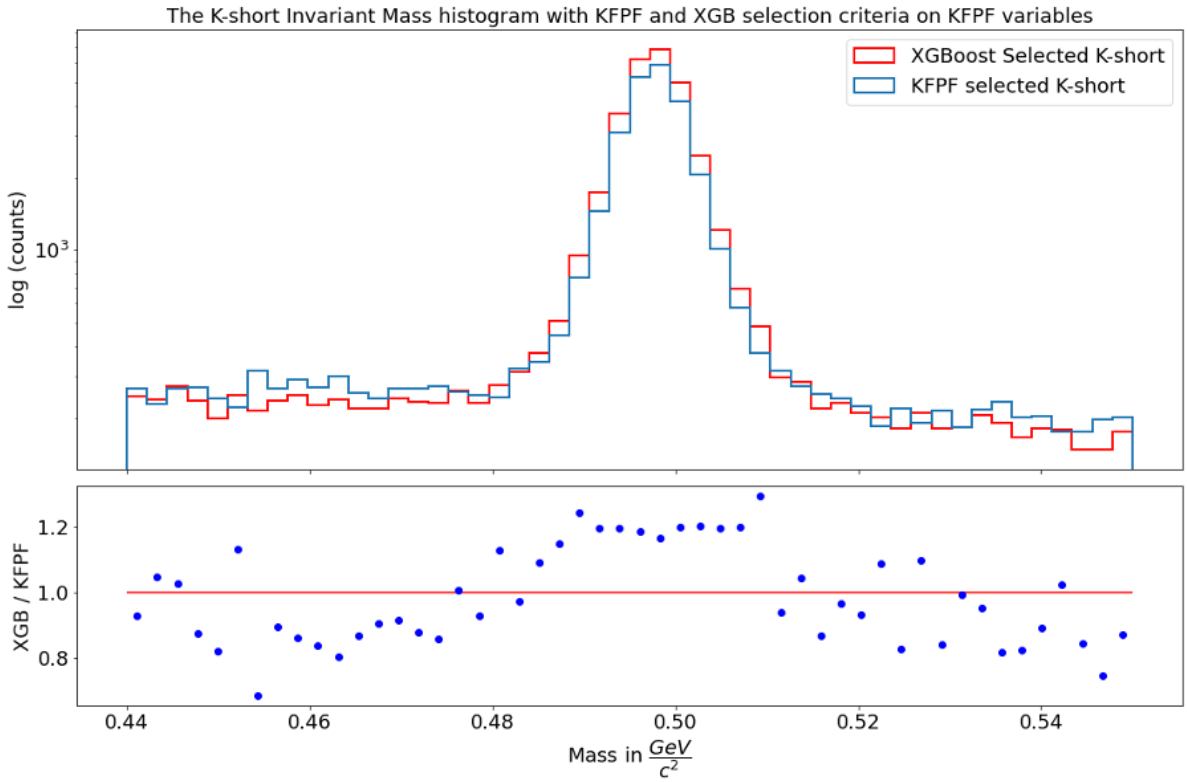


(b) Invariant mass distribution (close-up)

Figure 6.7: Comparison with KFPF cuts

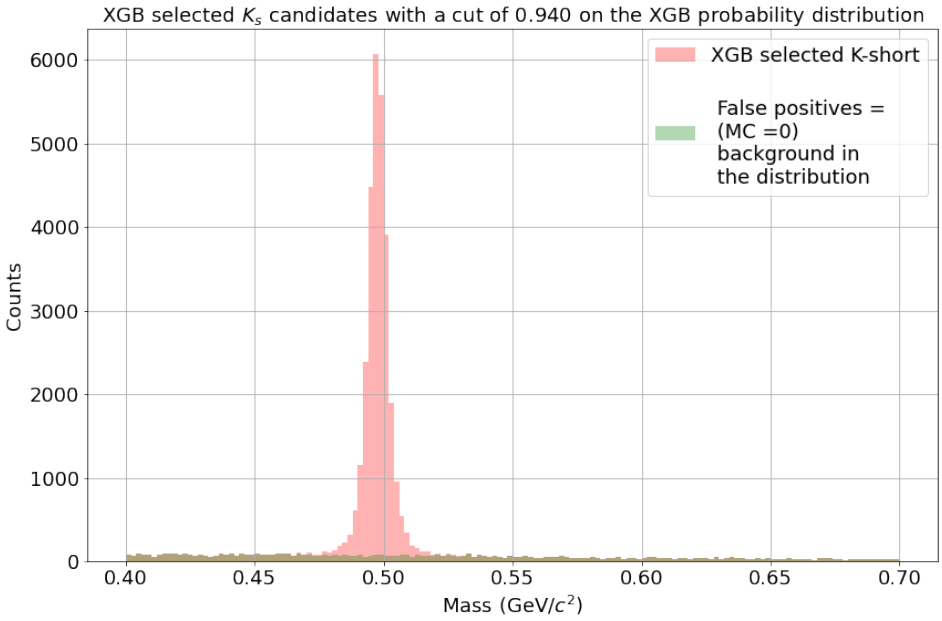


(a) Invariant mass distribution (y log scale)

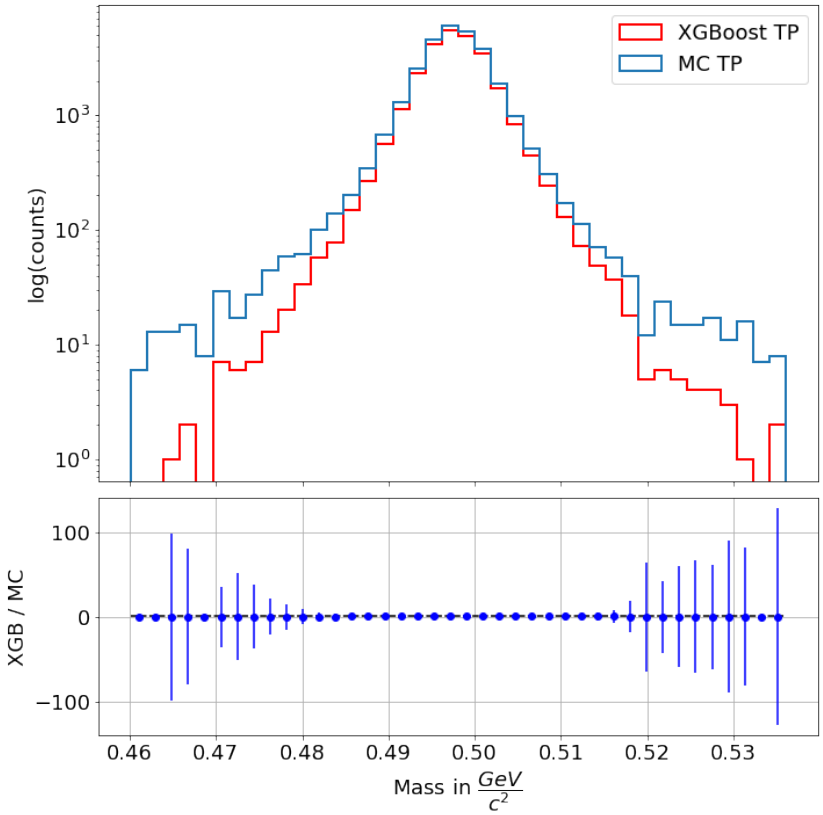


(b) Invariant mass distribution (close-up) - comparison with KFPF cuts

Figure 6.8: Invariant mass distribution for probability $>$ 0.86



(a) Selected K_S^0 and false positives distribution



(b) Reconstructed and MC true positives (y log scale)

Figure 6.9: True-false positives investigation

6.2.2 Results for p_{beam} = 3.3A GeV/c

Comparison with KFPF cuts

The same code can be used to obtain similar results for another CBM energy level. Comparing to default KFPF cuts, with probability cut 0.9635 (Fig. 6.10):

- **Reconstructed K_S^0/ reconstructible K_S^0 = 93.45%** vs. 78.94% with default KFPF cuts
- **false / true positive rate = 0.19** vs. 1.38 with default KFPF cuts

Due to the smaller statistics for this energy level, the ML model training should be redone for a bigger dataset.

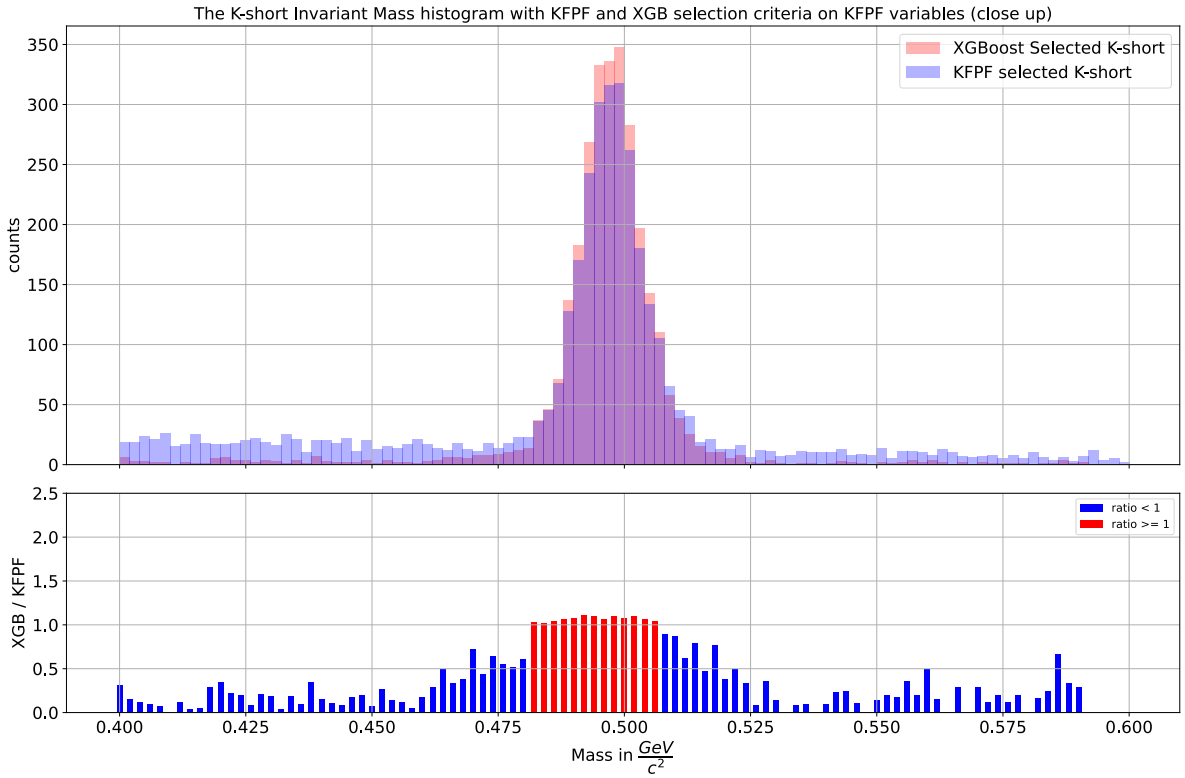
6.2.3 Influence of the magnetic field scaling

Obtained ML model can be adapted for the investigation of the influence of the magnetic field scaling. We compare: 100% MF strength (for p_{beam} = 12A GeV/c), 56% and 27.5% MF strength (for p_{beam} = 3.3A GeV/c, both with only DCM generated data for 0.4M events for training and 0.1M for validation). We see (Fig. 6.11) that the stronger the magnetic field is, the broader K_S^0 invariant mass distribution peak is. Also, we observe much more signal entries for p_{beam} = 12A GeV/c (for the same number of events).

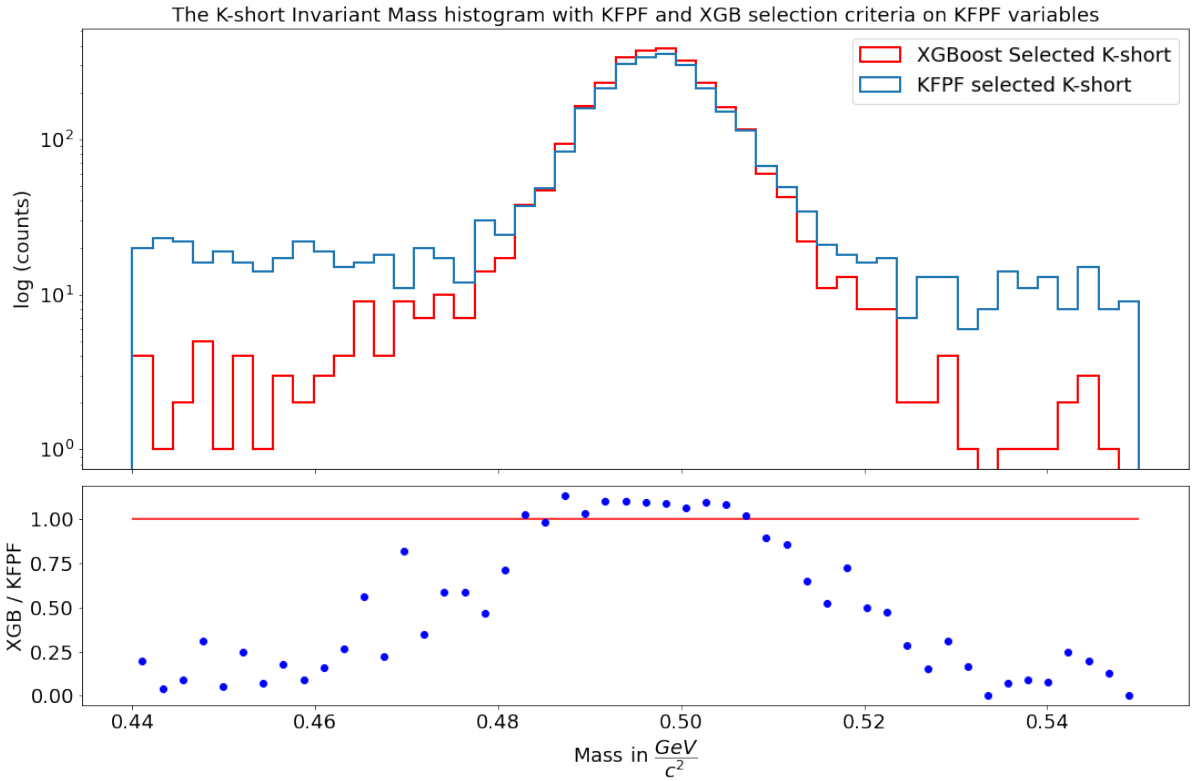
For p_{beam} = 3.3A GeV/c the number of signal entries is almost the same; we select the probability cut so that the false/true positive ratio is the same for both % of MF and compare the efficiency of the reconstruction (Tab. 6.2). We see that weaker MF does not necessarily worsen efficiency (only 2% difference). However, this comparison should also be redone for a bigger dataset.

Table 6.2: Comparison of MF strength vs. efficiency

ratio	**12 A GeV/c MF=100%**	**3.3 A GeV/c MF=56%**	**3.3 A GeV/c MF=27.5%**
reconstructed/ reconstructible	89.98%	90.36%	88.49%
false / true positive	0.2	0.2	0.2



(a) Invariant mass distribution (y log scale)



(b) Invariant mass distribution (close-up)

Figure 6.10: Comparison with KFPF cuts for 3.3A GeV/c

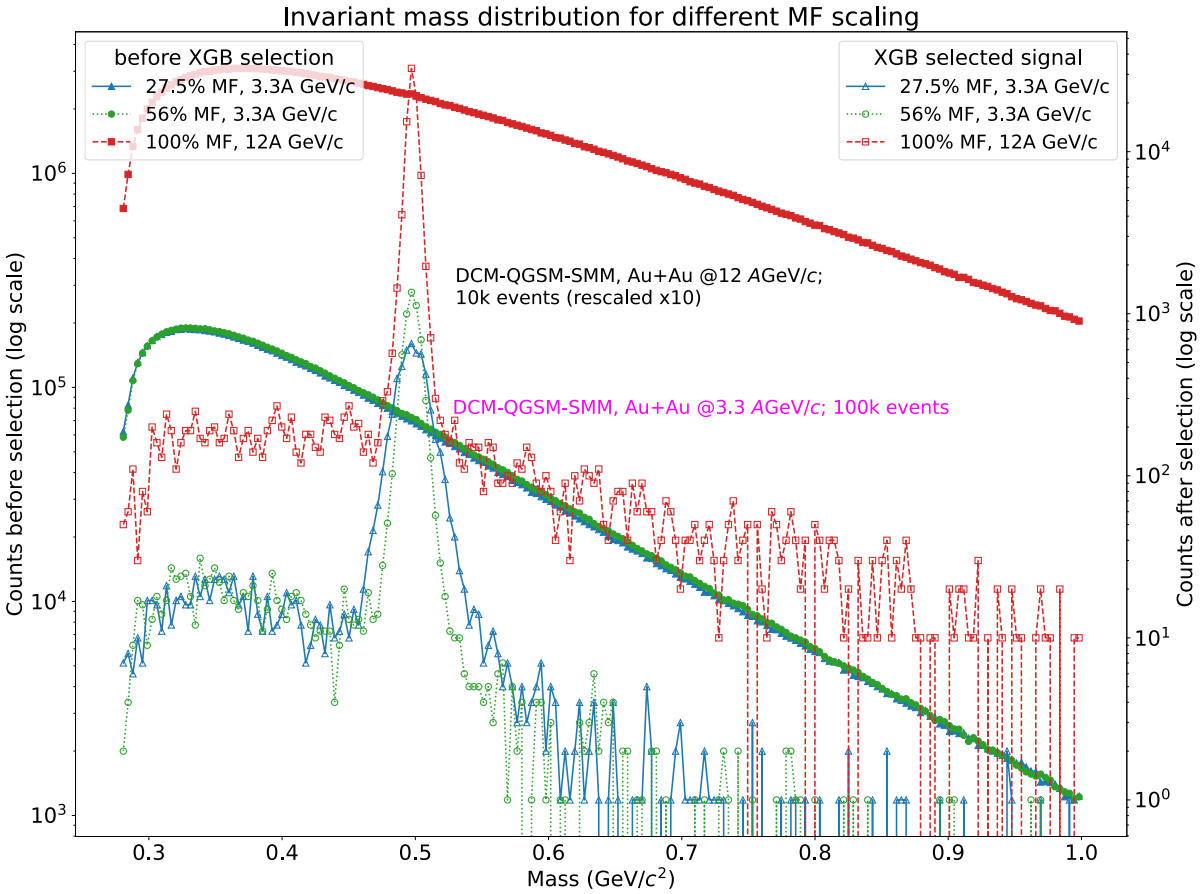


Figure 6.11: Invariant mass distribution for different MF scaling

”TOF” particles identification using ML

In this chapter, the application of the ML algorithms for particles identification (as the equivalent of the TOF method) will be discussed. As opposed to the traditional TOF method, instead of plotting mass-squared and $p \cdot q$, and then fitting Gaussians to the distributions to differentiate between the particles group, the data from MVD+STS and TOF detectors will be provided directly to the ML model. The aim is to differentiate between the three groups of particles:

- protons
- kaons
- pions (in the third group, the muons and electrons are included as well, as their mass-squared is almost indistinguishable without the data from other detectors)

The mass-squared of all the particles to be distinguished is shown on Figure 7.1.

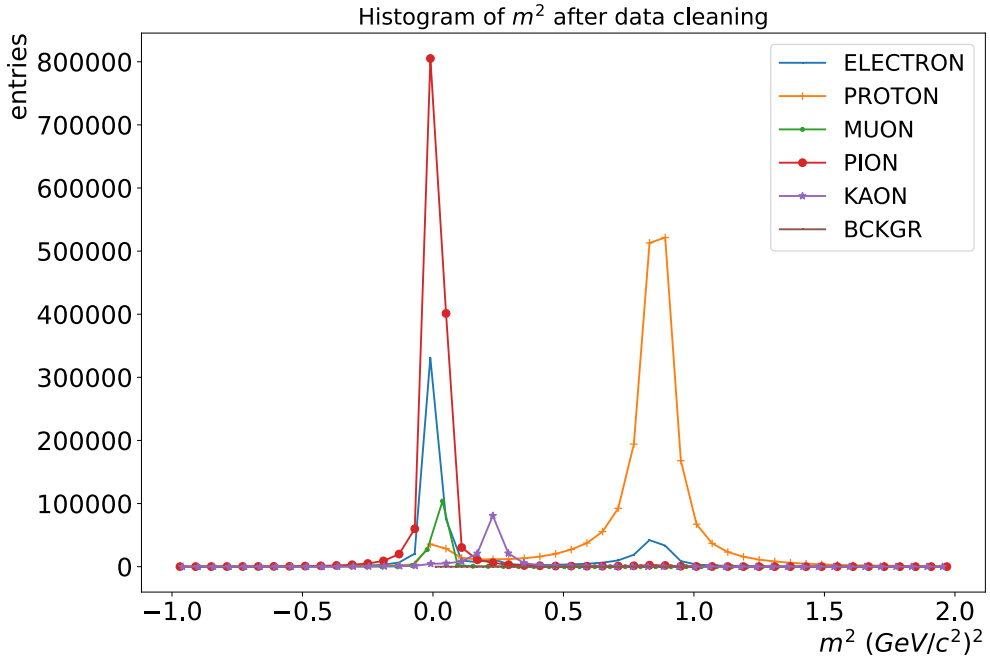


Figure 7.1: Histograms of mass-squared of each particle class (differences of quantity of each particle type can also be observed)

7.1 Model preparation

In this chapter, the data from the detectors saved in the AnalysisTree format is converted into a PlainTree format using a C++ program with ROOT libraries. Its fragments are presented in the appendix B. The following variables can be used:

- From MVD+STS: charge sign (q); momentum components: p, p_T, p_x, p_y, p_z; rapidity and pseudorapidity (η); azimuthal angle (ϕ)
- From TOF: mass-squared (m^2), calculated by formula 5.1 (using p value from MVD+STS)
- From MC model: invariant mass; PID code, which returns the information about the type of particle in the PDG convention, e.g., -2212 = (-)anti—(2212)proton[34]

The same libraries were used as in chapter 6. For training and validation, the following datasets are used:

- 2M events generated in DCM-QGSM-SMM (training dataset)
- 1M events generated in UrQMD (test dataset)

Au-Au @12A GeV/c passed through CBM setup in GEANT4, and PFSimple. The different models used for training and validation allow us to check if the ML model is not biased towards a specific MC model.

7.1.1 Data enriching

Remapping

The four different classes will be loaded into the ML model:

- 0: protons
- 1: kaons
- 2: pions (with the muons and electrons)
- 3: *background* - all the other particles

The classes 0-2 are loaded into the model during the training. Later, if the probability returned by the XGB is smaller than a specific threshold, the particle is identified as 3 - background.

Underrepresentation

As different classes have a different number of particles, they are rescaled so that each class has the same number of elements (in this case, it is specified by the number of Kaons, as they are the most underrepresented class).

7.1.2 Data cleaning

To reject the numeric values of parameters that do not have physical sense, but are present in the data set, some selection criteria are applied before the beginning of the model training. As in chapter 6, some values which might be possible but are rare enough are rejected, to reduce the amount of data.

Mass-squared

To constraint the mass-squared region to the one of the selected classes:

$$\text{-1} < m^2 < \text{2}\ (GeV/c^2)^2$$

Later during the training (not for the validation dataset), the mass-squared of each class will be constrained to a specific value of σ - standard deviation (Equation 6.5) around its mean value. After some tests, the following values were chosen: 1σ region for ID=0 and ID=2; 1.5σ for kaons (ID=1). The TOF histogram after data cleaning is shown on Figure 7.2; the histograms after σ-selection are shown on Figure 7.3, Figure 7.4, and Figure 7.5.

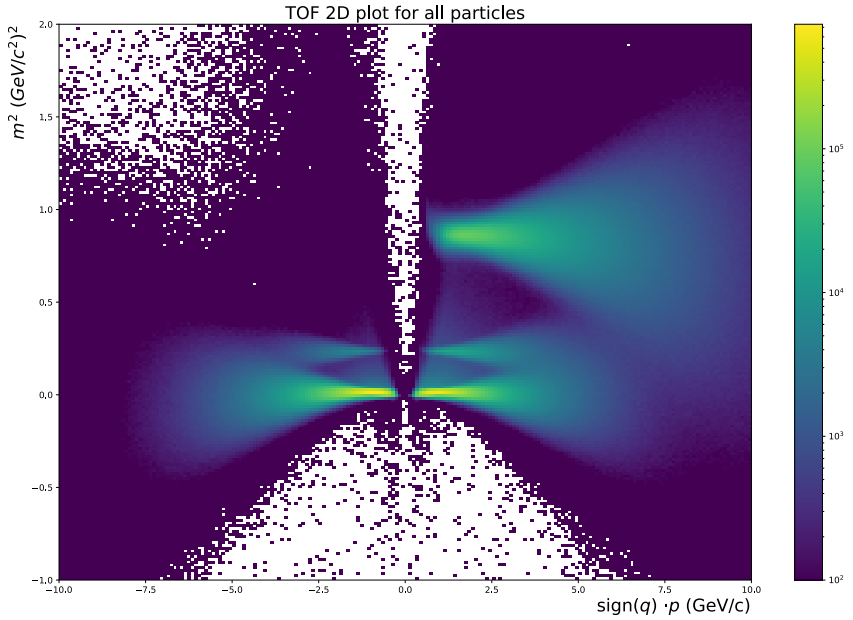


Figure 7.2: 2D TOF histogram of all particles after data cleaning

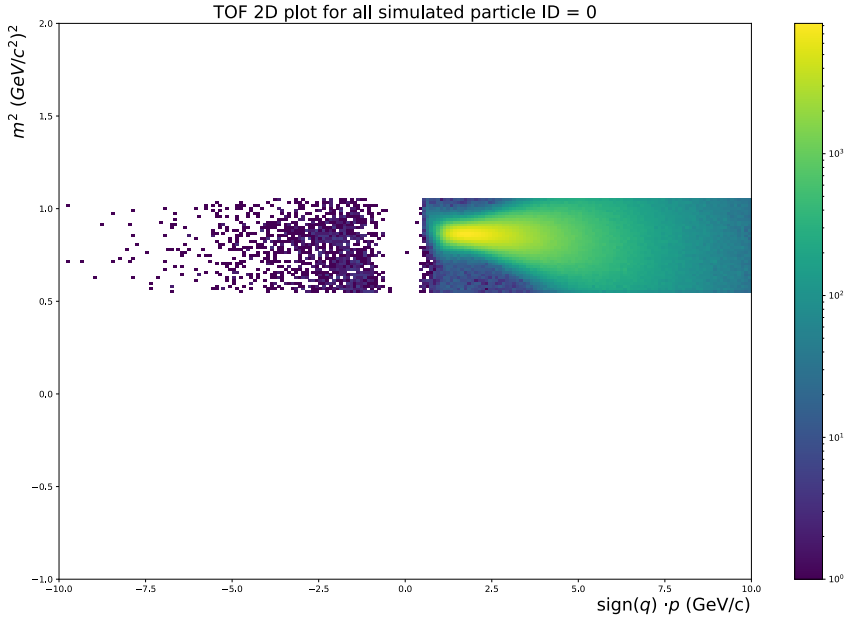


Figure 7.3: 2D TOF histogram of protons (ID=0) after 1σ selection

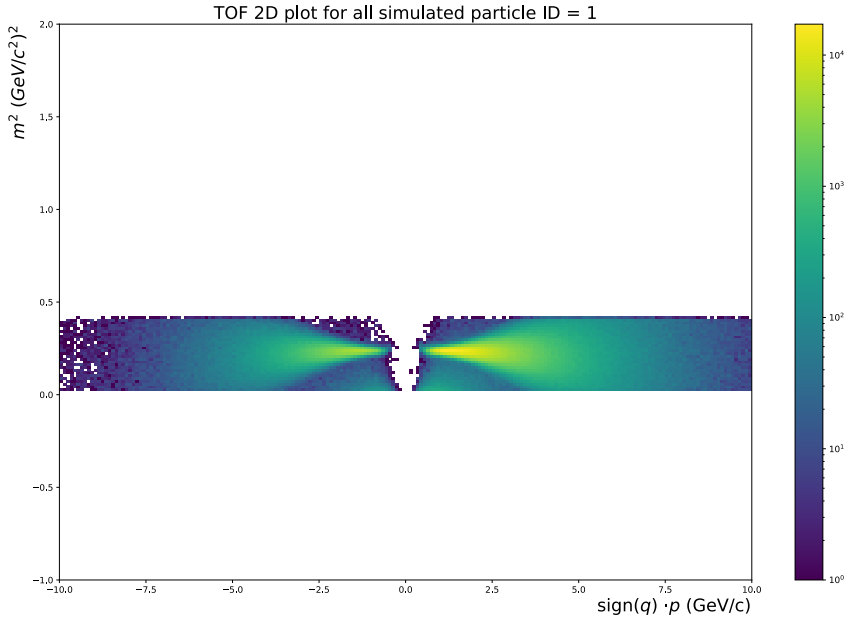


Figure 7.4: 2D TOF histogram of kaons (ID=1) after 1.5σ selection

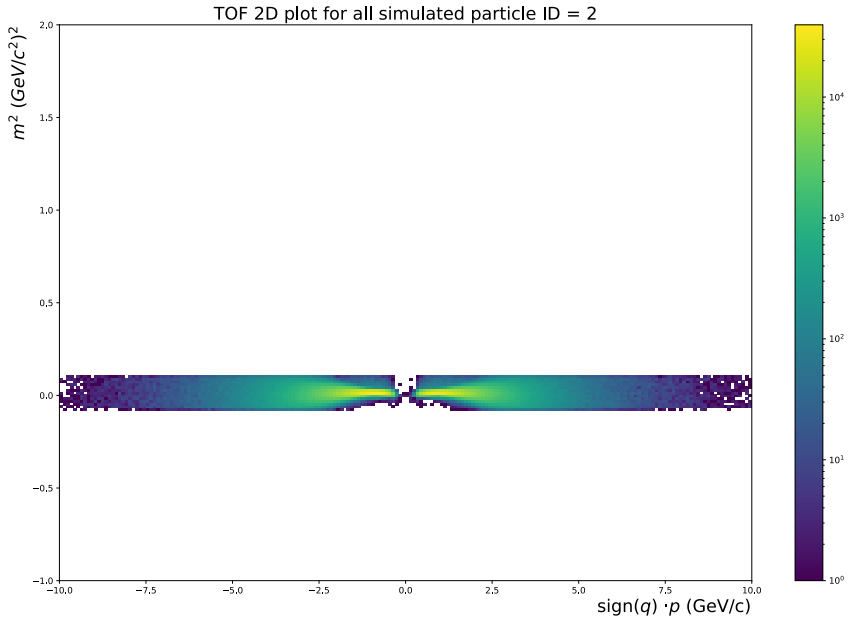


Figure 7.5: 2D TOF histogram of pions, muons and electrons (ID=2) after 1σ selection

Momentums

The fixed target geometry of the detector requires that:

$$p_Z > 0 \text{ GeV/c}$$

To reduce the data:

$$p < 12 \text{ GeV/c}; p_T < 2 \text{ GeV/c}$$

7.1.3 Variables selection

In this chapter, the TOF identification method is recreated, so the m^2, p and q values are being used. However, the results coming from only those 2 values were not perfect (usually, it is better to train an XGBoost model using multiple variables). After plotting the correlation matrix (with Pearson correlation efficient 6.3) (Figure 7.6), and following the other works [27], the p_T and η values are being selected also. The correlation matrix shows that these two variables are not highly correlated ($\rho < 0.4$).

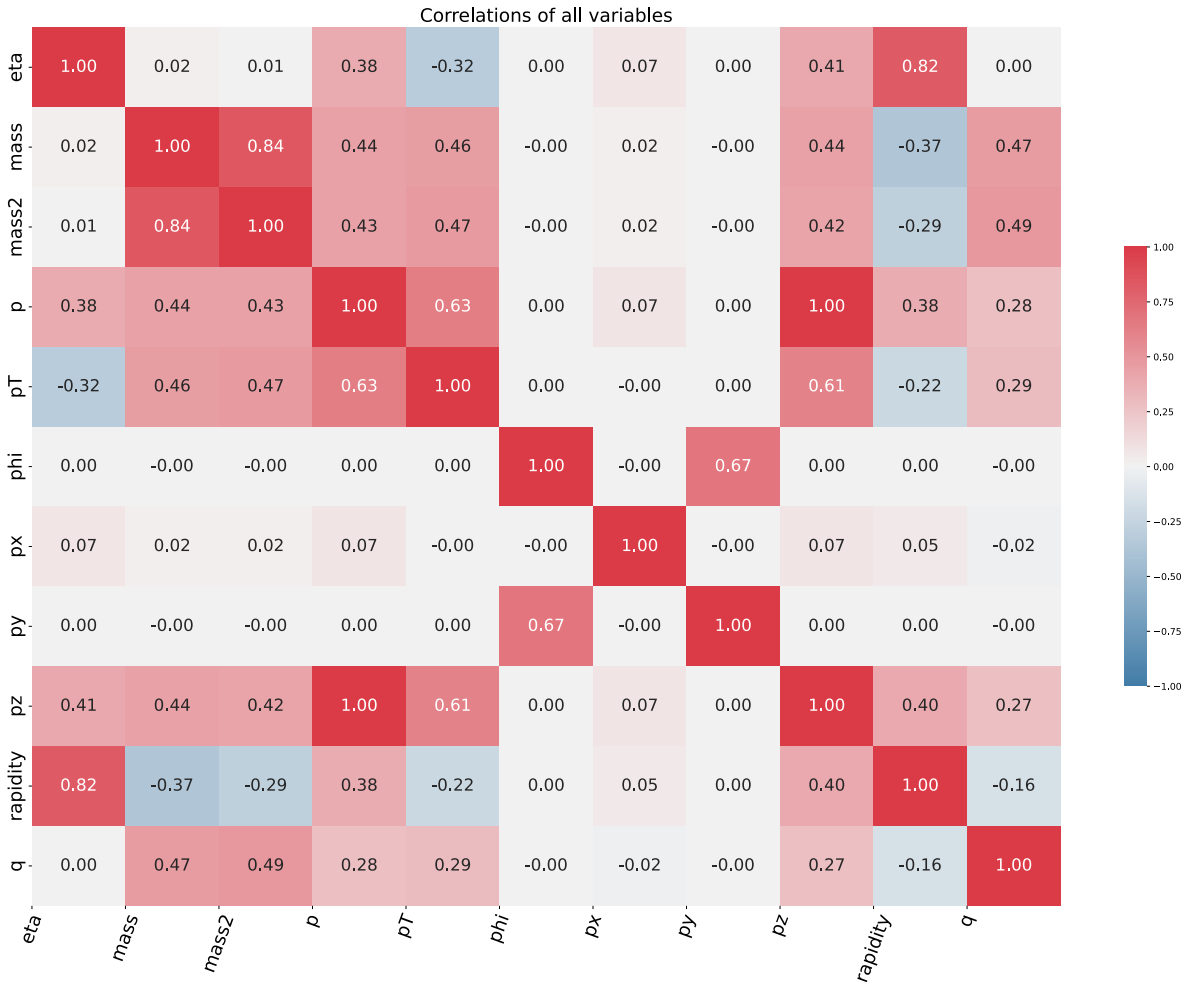


Figure 7.6: Correlation matrix

7.2 Model training

With the prepared data, the ML model training can be started. Once again, the hyperparameters are being selected using Bayesian Optimisation.

7.2.1 Probability plots

This time (contrary to K-short reconstruction optimisation), our model is a multi-class classifier (non-binary). For each particle, it returns the probability of belonging to each (0-2) class. The probability plot for all the classes can be plotted (Figure 7.7).

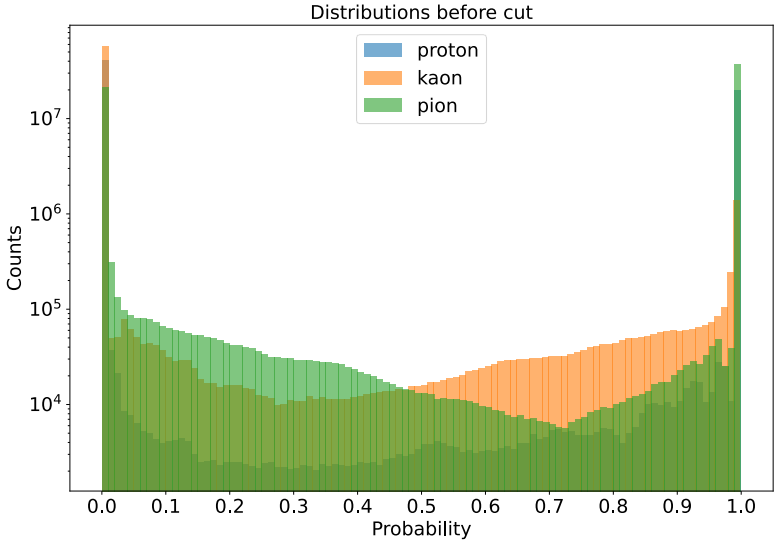


Figure 7.7: Probabilities distribution for all the classes

Later, the probability plot for each class can also be made, showing probability for all the particle vs. true-positives (particles belonging to the class). They are shown on Figure 7.8, Figure 7.9, and Figure 7.10. It allows to set a certain threshold, below which it is better to classify a particle as background (ID=3). With this approach, the number of false-positives is smaller; it also includes particles which don't belong to the 0-2 classes (e.g., $\Sigma^{\pm}$):

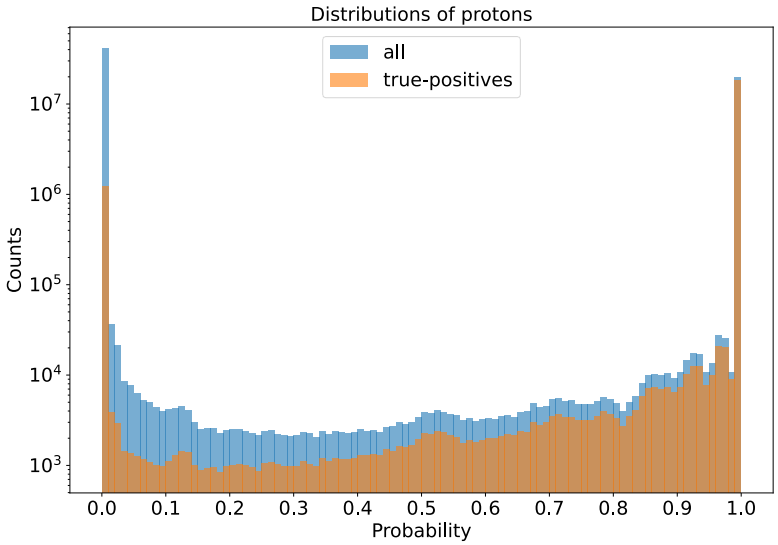


Figure 7.8: Probability distribution for ID=0

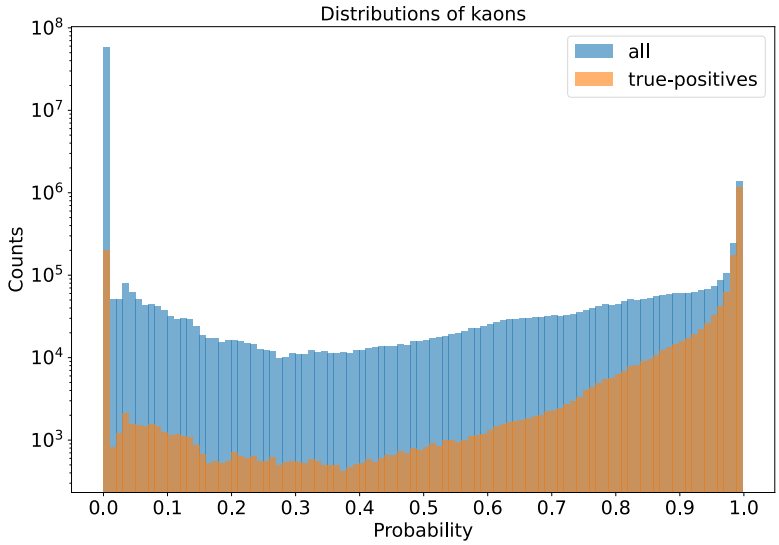


Figure 7.9: Probability distribution for ID=1

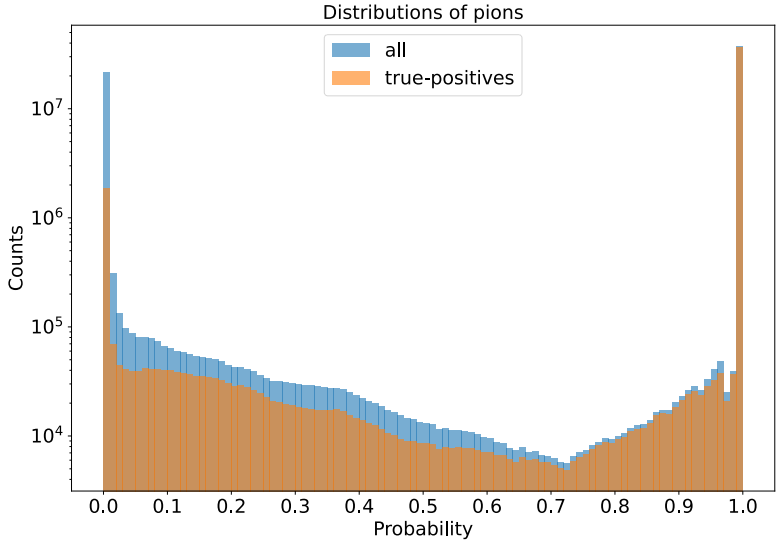


Figure 7.10: Probability distribution for ID=2

The following thresholds (which aim for as big efficiency as possible, with correct shape of mass-squared distribution (plotted later)): were decided:

- for ID = 0 : 0.88
- for ID = 1 : 0.96
- for ID = 2 : 0.82

7.2.2 Confusion matrix

The confusion matrix is useful in checking the accuracy of a classifier. By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j. Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$[35]. The normalised confusion matrix shows not the number of observations for each class, but the percentage.

For our model, the confusion matrices after applying the probability threshold are plotted on Figures 7.11 (without normalisation - the number of entries in each category is shown) and 7.12 (with normalisation - the percentage shown shows the efficiency of true positives in each category). It is observed, that the efficiency is higher for more represented classes (pions and protons).

Using the information from the confusion matrix, the efficiency of the classifier can be put in the Table 7.1.

Table 7.1: Efficiency of the ML model for TOF identification

ID	Efficiency	Efficiency of true positives	False/True ratio
0	108%	92.88%	0.17
1	119%	72.48%	0.65
2	102%	91.60	0.12

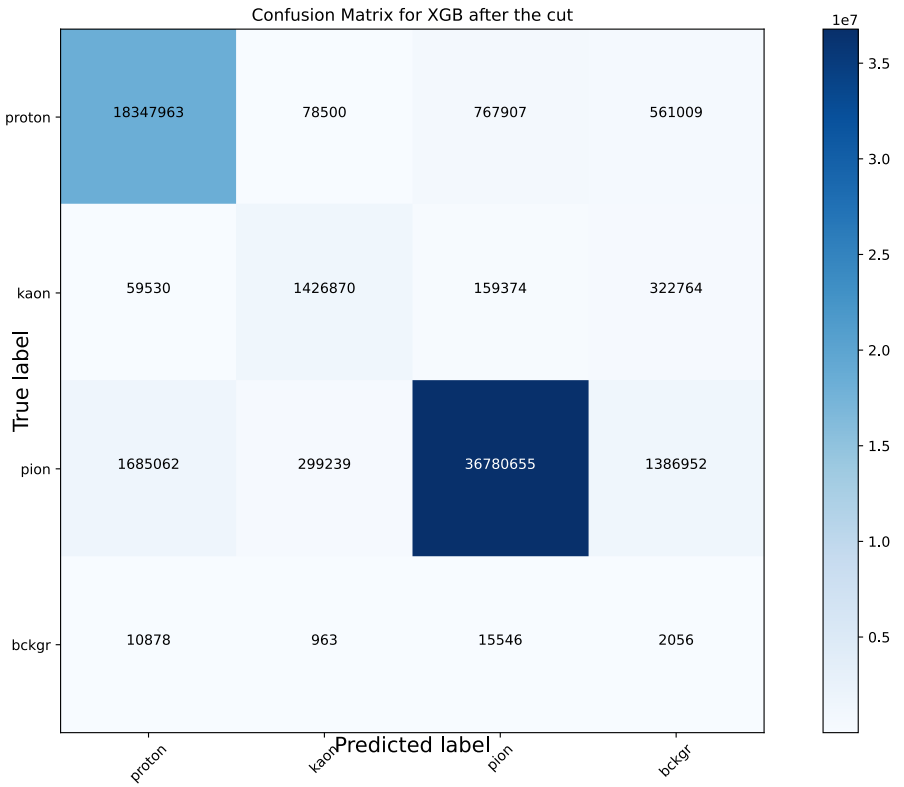


Figure 7.11: Confusion matrix

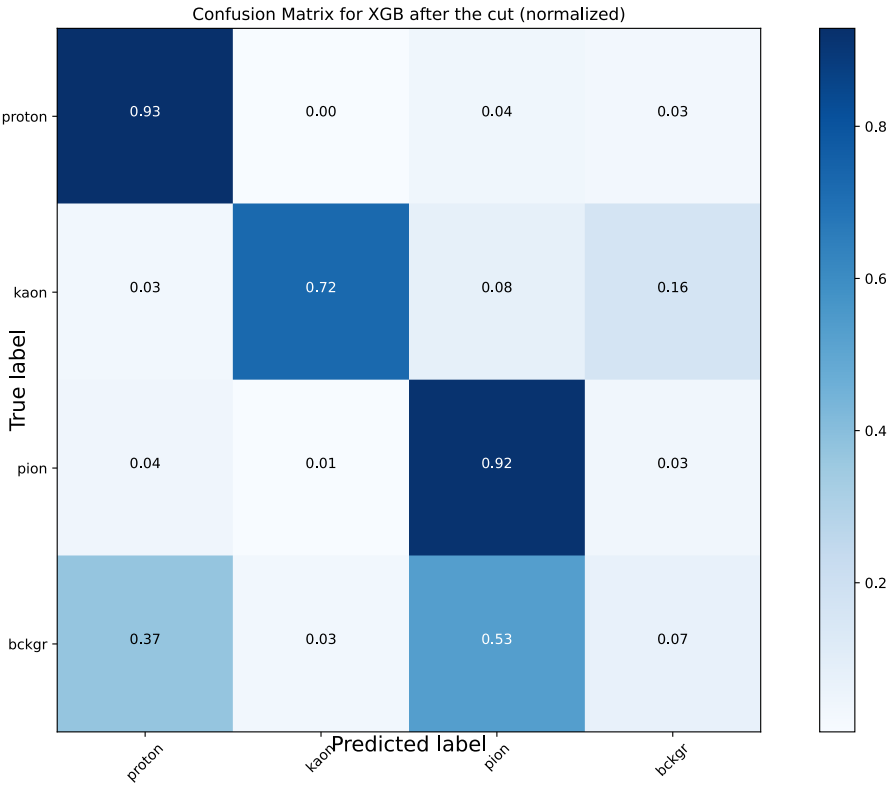


Figure 7.12: Confusion matrix (normalised)

7.2.3 Mass-squared distributions

After the ML model training and running it on the test dataset, the invariant mass distributions are returned, presented of Figure 7.13, Figure 7.14, and Figure 7.15.

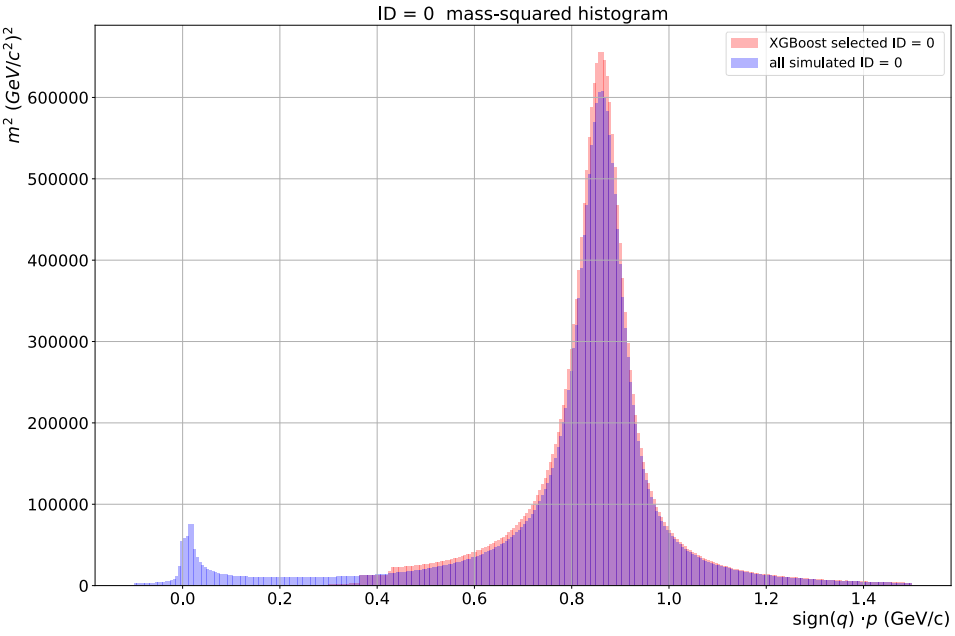


Figure 7.13: Mass-squared distributions for particles ID = 0

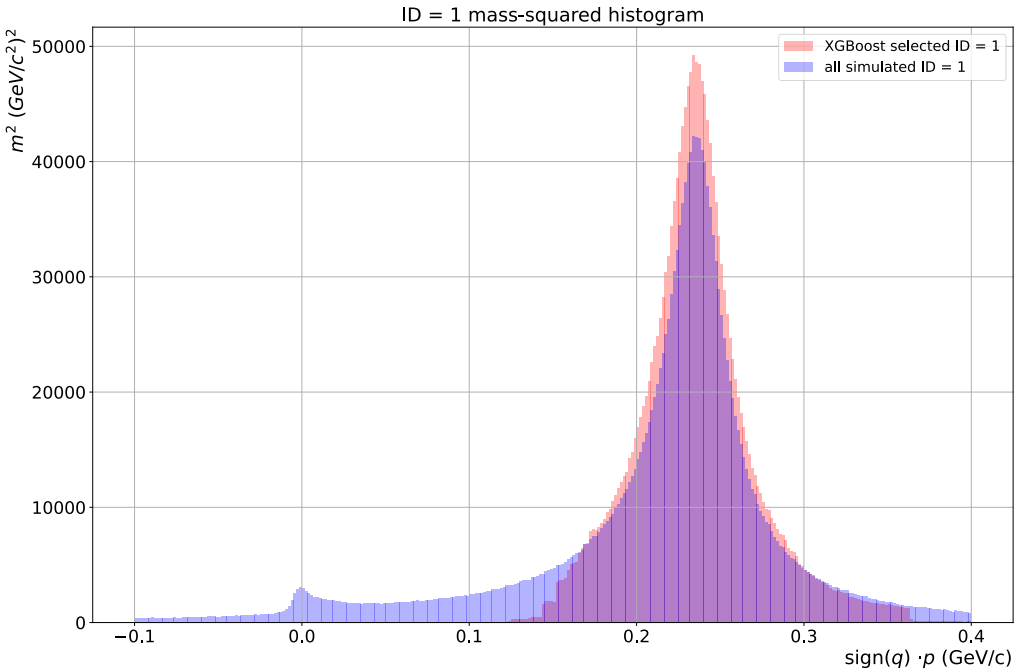


Figure 7.14: Mass-squared distributions for particles ID = 1

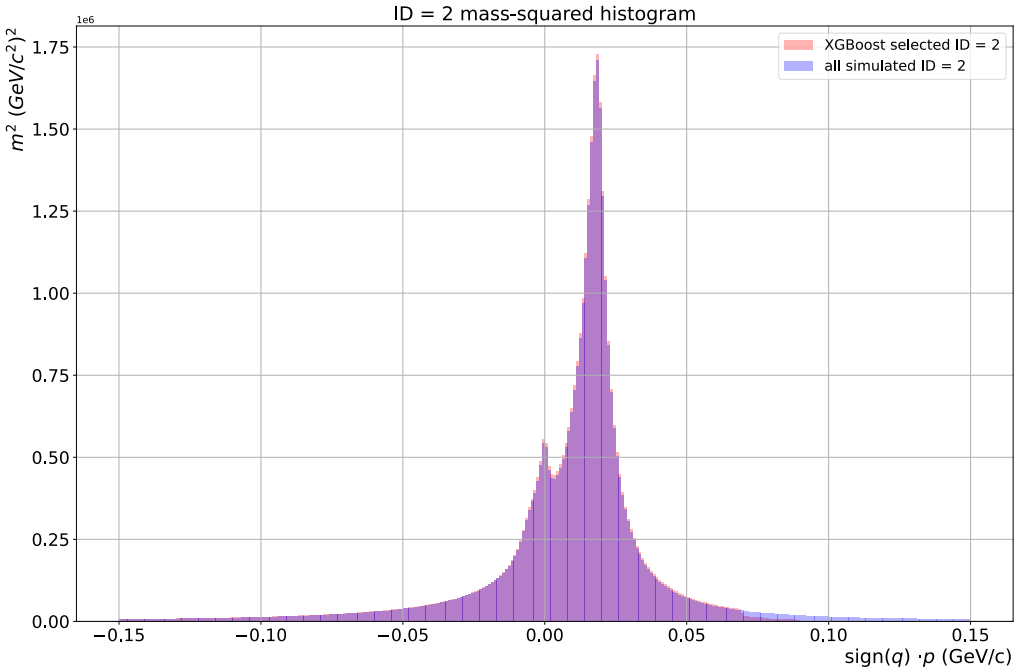


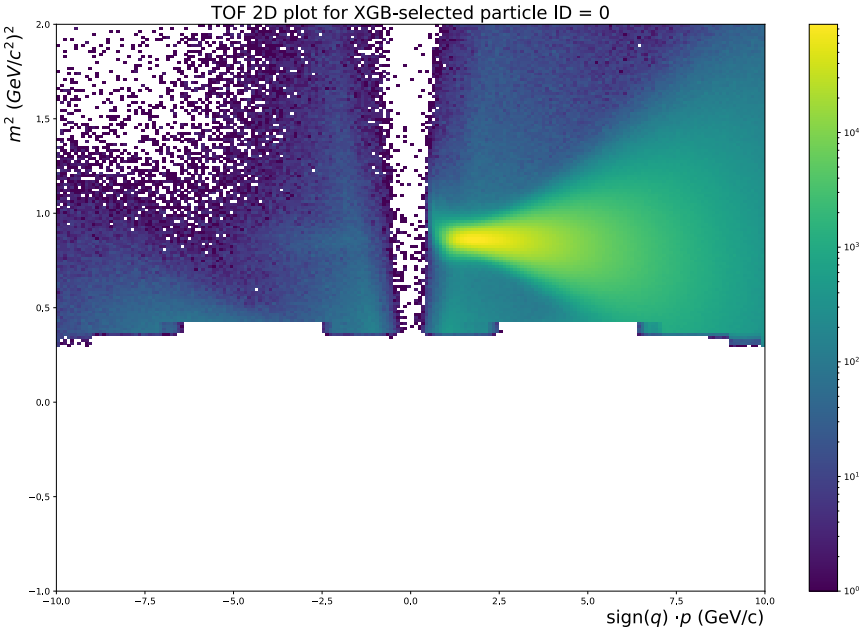
Figure 7.15: Mass-squared distributions for particles ID = 2

7.3 Analysis of the results

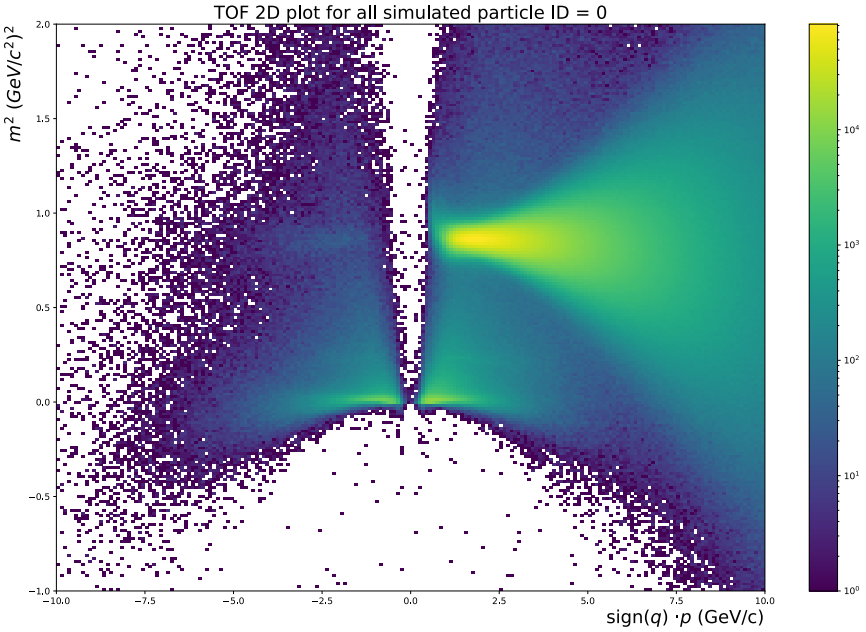
While the mass-squared distributions are useful, simple visualisation of the classifier results, the TOF plots, are handier in analyzing the way the ML model works. The XGBoost-selected particles can be plotted with the simulated particles (all the particles that should be reconstructed) for each class.

ID = 0 (protons)

For particles ID = 0 (protons) it is to observe that the protons of mass-squared close to 0 (mostly protons coming directly from the bullet particle) are not identified correctly. However, they should be treated as a different class, as the sigma-selection cuts them out during the training. For the same reason, protons with high p, but small mass-squared are not identified at all. The 2D TOF plot of both simulated, and XGBoost-selected particles, are shown on Figure 7.16.



(a) XGBoost-selected particles

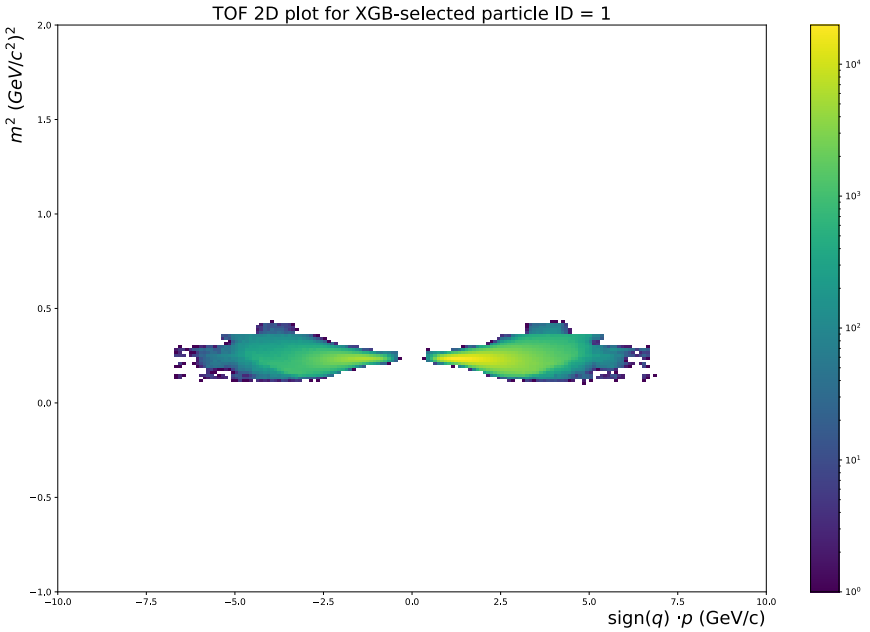


(b) Simuated particles

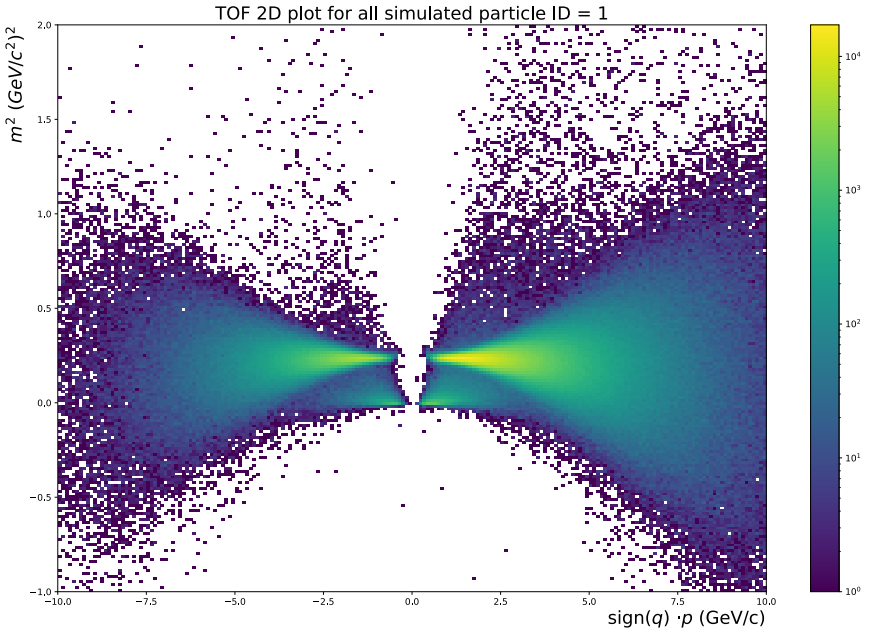
Figure 7.16: 2D TOF plot for ID = 0

ID = 1 (kaons)

For particles ID = 1 (kaons) the efficiency, and the false-positive ratio, are most deviated from correctness. The ML model with a high value of the chosen threshold picks only kaons with small p value. Also, one region of kaons close to $m^2 = 2$ is not used in training - however, it may be a result of a mismatch in the simulation, as the kaons are not expected in this region. The 2D TOF plot of both simulated, and XGBoost-selected particles, are shown on Figure 7.17.



(a) XGBoost selected particles

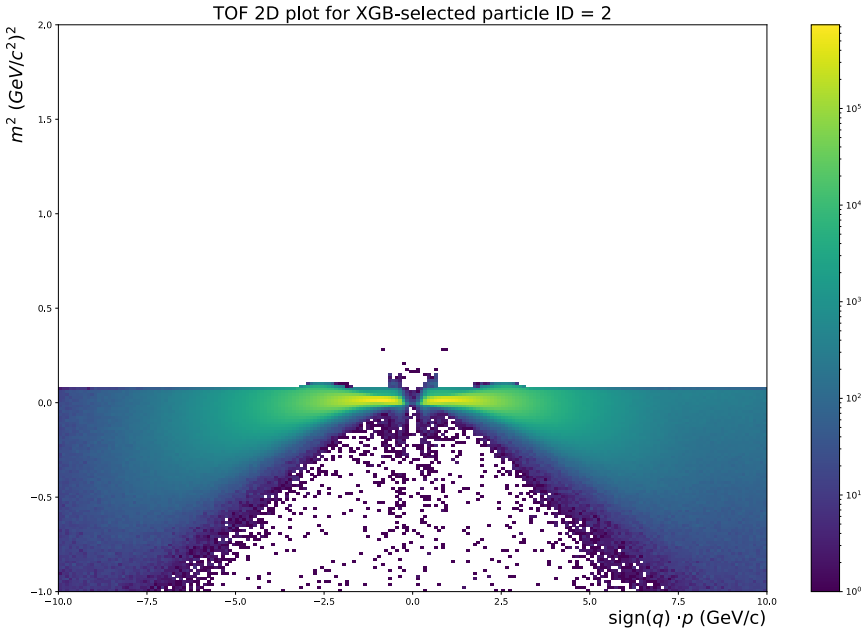


(b) Simulated particles

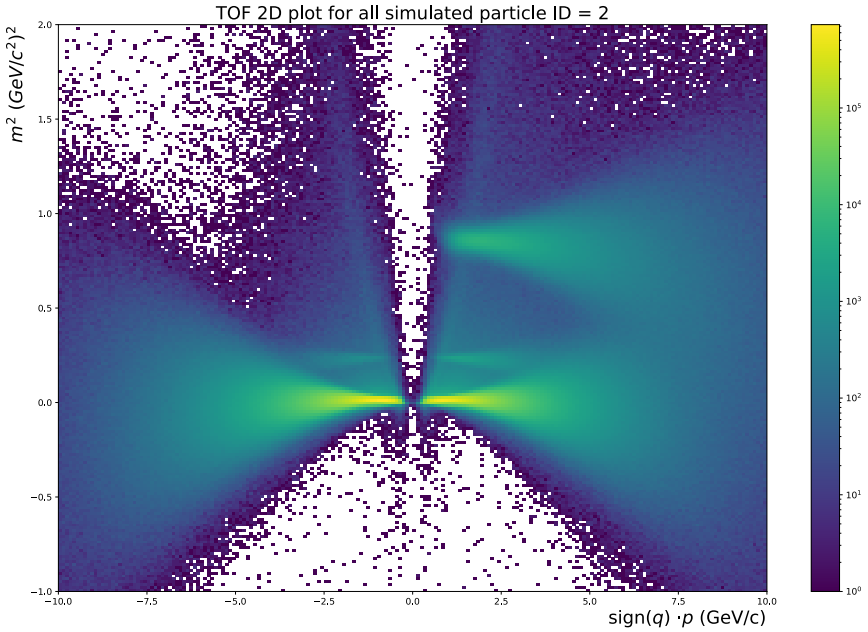
Figure 7.17: 2D TOF plot for ID = 1

ID = 2 (pions, muons, and electrons)

For particles ID = 2 (pions, muons, and electrons), the efficiency of the reconstruction is the highest, as most of the particles of the lower mass-squared values belong to one of these classes. Once again, there is a region that is not expected - electrons of $m^2 \approx 2$ which are most probably a result of the mismatch in the simulation of the CBM setup. The 2D TOF plot of both simulated, and XGBoost-selected particles, are shown on Figure 7.18.



(a) XGBoost selected particles

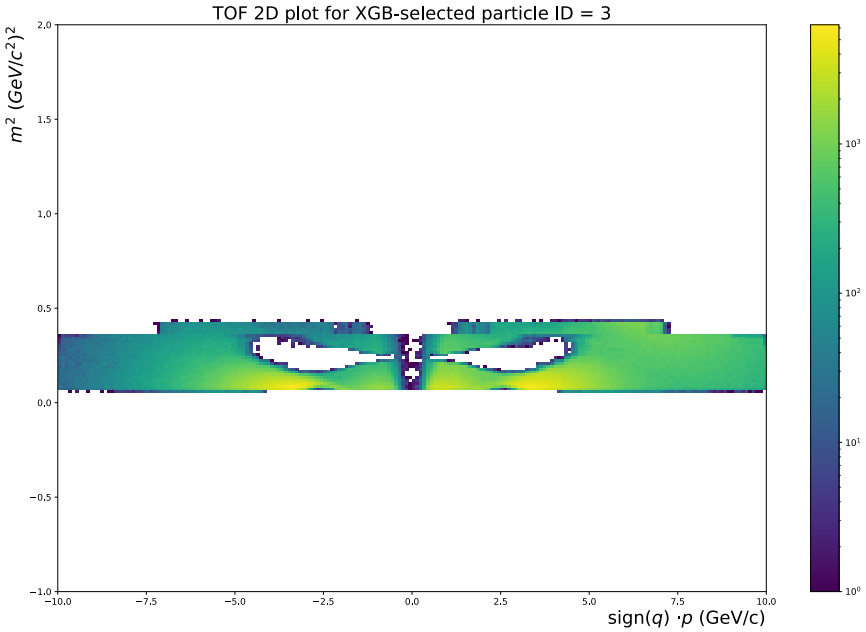


(b) Simulated particles

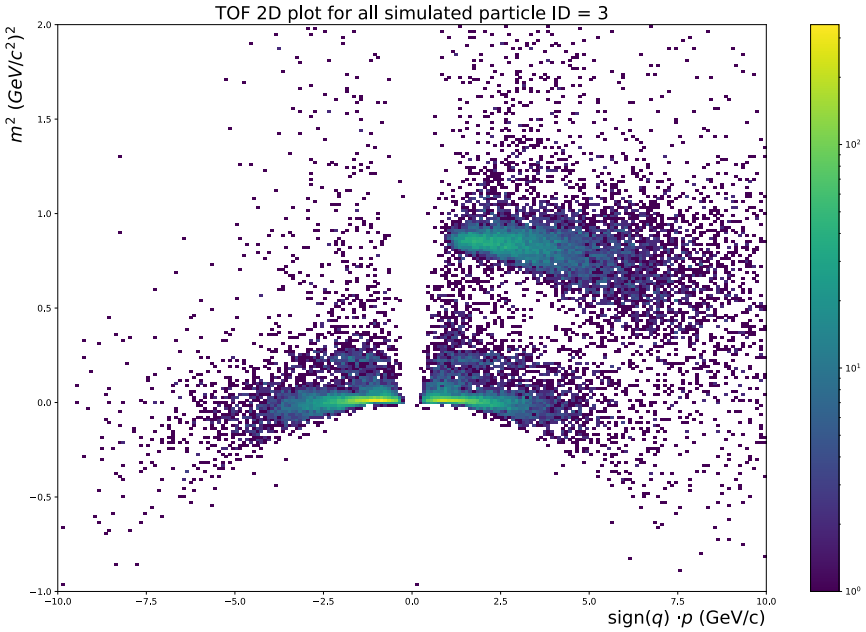
Figure 7.18: 2D TOF plot for ID = 2

ID = 3 (background)

The particles of mass-squared close to the value of the mean mass-squared of kaons, but with bigger p values, were mostly categorized as ID = 3 (background). It can also be seen in the confusion matrix - 0.14% of kaons are misidentified as background, as well as kaons and ID = 2 particles which were found in this m^2 region. The 2D TOF plot of both simulated, and XGBoost-selected particles, are shown on Figure 7.19.



(a) XGBoost selected particles



(b) Simulated particles

Figure 7.19: 2D TOF plot for ID = 3

Discussion and summary

In the first part, the application of machine learning for K-short reconstruction results in better background reduction (Figure 6.7) and better efficiency (Figure 6.8), compared to the traditional, manual method. A complicated and extensive search of selection criteria using the manual method can be omitted this way; the same code works for different collision energies and magnetic field scaling settings as well. It allows e.g., the investigation of the influence of the magnetic field scaling on the efficiency of reconstruction (Figure 6.11). The usage of XGBoost allowed for a relatively short training time. Exporting the trained classifier into the CBM experiment simulation and analysis software (CbmRoot) was not part of this thesis. However, the XGBoost model can be easily deployed into ROOT using e.g., Treelite package [36]. Also, the classifier should be retrained on more data before applying it into the CbmRoot and tested on real data once the experiment starts.

In the second part, the XGBoost model was trained for the identification of three groups of particles, following the traditional TOF method. It allowed receiving efficiencies of identification for protons and pions (along with muons and electrons) around 90%. However, the least represented class, kaons, achieves an identification efficiency of about 70% (Table 7.1). Upon comparison of 2D TOF graphs of simulated and identified particles (Figures 7.16 - 7.19), it is observed that the model works the best for particles of mass-squared close to mean value for each particles group and smaller momentum values. The efficiency drops for the ”tails” of the distributions, which is an important challenge using Bayesian-fitting (in the traditional TOF method) as well.

Two possible solutions are discussed in the CBM-ML group. The first is dividing training datasets into a few p-value bins, e.g., from 0-2 GeV/c, then 2-4GeV/c, etc. It could minimize the mismatch of the identified particles in the ”tails” of the distribution. The second solution would be using data from more detectors that could allow identifying pions, muons, and electrons separately, as well as other *background* particles, resulting in less mismatch. In this case, however, the *deep learning* network could be needed, as the XGBoost gives the best results when it is used as a binary classifier, and using multiple classes (more than three in this case), could alter the results.

To sum up, the application of machine learning algorithms for particle identification in heavy-ion collisions is a very promising solution. It could be used for the reconstruction of

short-lived particles such as K-short, after deploying the model into the CbmRoot. Applying ML for the identification of particles that can be detected directly is also promising, although is not ready for deployment in the CBM experiment software yet. This topic should be further investigated, as due to the amount of data that should be produced in the CBM experiment, the application of ML for particle identification could speed up this task significantly. It should offer physicists, who analyze the results of the experiment, faster access to data of better quality.

Bibliography

[1] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin and E. Zakharov, “Generative Models for Fast Calorimeter Simulation: the LHCb case” EPJ Web Conf. **214** (2019), 02034 doi:10.1051/epjconf/201921402034

[2] S. Khan, V. Klochkov, O. Lavoryk, O. Lubynets, A. I. Khan, A. Dubla and I. Selyuzhenkov, “Machine Learning Application for Λ Hyperon Reconstruction in CBM at FAIR,” [arXiv:2109.02435 [physics.ins-det]].

[3] Hanna Paulina Zbroszczyk. ”Eksperymentalne aspekty badania korelacji femtoskopowych w zderzeniach relatywistycznych ciezkich jonów”. Oficyna Wydawnicza Politechniki Warszawskiej, 2019, pp. 12, 24, 29.

[4] D. Castelvecchi, “What’s next for physics’ standard model? Muon results throw theories into confusion,” Nature **593** (2021) no.7857, 18-19 doi:10.1038/d41586-021-01033-8

[5] wikipedia.org

[6] Grebieszkow K.:"Fizyka zderzeń ciężkich jonów", Lectures at Faculty of Physics of WUT, http://www.if.pw.edu.pl/ kperl/HIP/hip.html

[7] "Compressed Baryonic Matter Experiment at FAIR, Progress Report 2020" Senger, Peter et. al (CBM Collaboration) doi:10.15120/GSI-2021-00421.

[8] Angela Bracco (2017) ”The NuPECC Long Range Plan 2017: Perspectives in Nuclear Physics”, Nuclear Physics News, 27:3, 3-4, DOI: 10.1080/10619127.2017.1352311

[9] CBM Collaboration, EPJA 53 3 (2017) 60 T.Galatyuk, NPA982 (2019), update (2021)

[10] K. Aamodt *et al.* [ALICE], “The ALICE experiment at the CERN LHC,” JINST **3** (2008), S08002 doi:10.1088/1748-0221/3/08/S08002

[11] T. Ablyazimov *et al.* [CBM], “Challenges in QCD matter physics –The scientific programme of the Compressed Baryonic Matter experiment at FAIR,” Eur. Phys. J. A **53** (2017) no.3, 60 doi:10.1140/epja/i2017-12248-y [arXiv:1607.01487 [nucl-ex]].

[12] M. Orsaria, H. Rodrigues, F. Weber and G. A. Contrera, “Quark deconfinement in high-mass neutron stars,” Phys. Rev. C **89** (2014) no.1, 015806 doi:10.1103/PhysRevC.89.015806 [arXiv:1308.1657 [nucl-th]].

[13] fair-center.eu, accessed 9.12.2021

[14] fair-center.eu/for-users/experiments/cbm.html, accessed 9.12.2021

[15] S. A. Bass, M. Belkacem, M. Bleicher, M. Brandstetter, L. Bravina, C. Ernst, L. Gerland, M. Hofmann, S. Hofmann and J. Konopka, *et al.* "Microscopic models for ultrarelativistic heavy ion collisions," Prog. Part. Nucl. Phys. **41** (1998), 255-369 doi:10.1016/S0146-6410(98)00058-1 [arXiv:nucl-th/9803035 [nucl-th]].].

[16] M. Baznat, A. Botvina, G. Musulmanbekov, V. Toneev and V. Zhezher, "Monte-Carlo Generator of Heavy Ion Collisions DCM-SMM," Phys. Part. Nucl. Lett. **17** (2020) no.3, 303-324 doi:10.1134/S1547477120030024 [arXiv:1912.09277 [nucl-th]].

[17] Słodkowski M.:"Modelowanie Procesów Jądrowych", Lectures at Faculty of Physics of WUT, http://efizyka.if.pw.edu.pl/MPJ

[18] S. Agostinelli *et al.* [GEANT4], "GEANT4–a simulation toolkit," Nucl. Instrum. Meth. A **506** (2003), 250-303 doi:10.1016/S0168-9002(03)01368-8

[19] D. Blau, O. Golosov, E. Kashirin, V. Klochkov and I. Selyuzhenkov, "Performance studies for collective flow measurements with CBM at FAIR," J. Phys. Conf. Ser. **1390** (2019) no.1, 012027 doi:10.1088/1742-6596/1390/1/012027

[20] https://www.ibm.com/cloud/learn/machine-learning, accessed 17.12.2021

[21] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," in IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-229, July 1959, doi: 10.1147/rd.33.0210.

[22] E. Lavrik, et. al., "Optical Inspection of the Silicon Micro-strip Sensors for the CBM Experiment employing Artificial Intelligence" arXiv:2107.07714

[23] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016. [arXiv:1603.02754]

[24] https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d, accessed 18.12.2021

[25] https://xgboost.readthedocs.io/en/latest/parameter.html, accessed 18.12.2021

[26] http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html, accessed 18.12.2021

[27] Ostrowski J., "Particle Identification Framework extension for multidimensional fitting and mismatch studies", presentation on the 38th CBM Collaboration Meeting

[28] http://hyperphysics.phy-astr.gsu.edu/hbase/Particles/kaon.html, accessed 19.12.2021

[29] Zyzak, PhD thesis, 165 (2016)

[30] Internal CBM documents: communication with O. Lubynets

[31] Nowak J. ”Optimizing K0S reconstruction at different collision energies using Machine Learning algorithms”, Internship and Training Project Report from the Get_Inolved programme

[32] https://github.com/shahidzk1/CBM_ML_Lambda_Library.git

[33] https://github.com/scikit-hep/uproot4

[34] https://pdg.lbl.gov/2019/reviews/rpp2019-rev-monte-carlo-numbering.pdf, accessed 27.12.2021

[35] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html, accessed 27.12.2021

[36] https://github.com/dmlc/treelite

List of Figures

List of Tables

Appendix A

Fragments of code (most important functions) used for K-short reconstruction ML model preparation are presented here.

```
#Installing package for importing simulated data in AnalysisTree format
#into Pandas Dataframe

!git clone https://github.com/shahidzk1/CBM_ML_Lambda_Library.git
%cd CBM_ML_Lambda_Library
!git pull origin main
!pip install -r requirements.txt
!python setup.py install
from CBM_ML import tree_importer

#Importing simulated data in AnalysisTree data into Pandas Dataframe format
sign = tree_importer.tree_importer(signalFileName,'PlainTree',7)

# We only select lambda candidates in the 5 sigma region
#around the kaon mass peak

sign = sign[(sign['Candidates_generation']==1) &
    (sign['Candidates_mass']>lower5SigmaCutSign) &
    (sign['Candidates_mass']<upper5SigmaCutSign)]

# Similarly for the background, we select background candidates
#which are not in the 5 sigma region of the kaon peak

bckgr = tree_importer.tree_importer(allURQMD,'PlainTree',7)
bckgr = bckgr[(bckgr['Candidates_generation'] < 1)
    & ((bckgr['Candidates_mass'] > lower5SigmaCutBckgr)
    & (bckgr['Candidates_mass'] < lower5SigmaCutSign) |
    (bckgr['Candidates_mass']>upper5SigmaCutSign)
    & (bckgr['Candidates_mass'] < upper5SigmaCutBckgr))
```

```
    ].sample(n=4*(sign.shape[0]))
    #we select bckgr so that we have 4*more
    #entries that for signal (before cleaning)

#Cleaning dataset
def clean_df(df):

    # Dropping NaNs and Infinities
    with pd.option_context('mode.use_inf_as_na', True):
        df = df.dropna()

    #Experimental constraints
    is_good_mom = (df['pz'] > pzLowerCut) &
        (df['p']<pUpperCut) &
        (df['pT']<ptUpperCut)

    is_good_coord = (abs(df['x']) < absXCut) &
        (abs(df['y']) < absYCut) & (df['z']>lowerZCut) & (df['z']<upperZCut)

    is_good_params = (df['distance'] > lowerDcaCut) &
    (df['distance'] < upperDcaCut) &
        (df['chi2geo']>lowerChi2GeoCut) &
        (df['chi2geo'] < upperChi2GeoCut) &
        (df['chi2topo'] > lowerChi2TopoCut) &
        (df['chi2topo'] < upperChi2TopoCut) &
        (df['eta']>lowerEtaCut) &
        (df['eta']<upperEtaCut)& (df['l']>lowerLCut) & (df['l']<upperLCut) &
        (df['loverdl']>lowerLdlCut) &
        (df['loverdl']<upperLdlCut)

    is_good_daughters = (df['chi2primfirst']>lowerChi2PrimFirstCut) &
        (df['chi2primfirst'] < upperChi2PrimSecondCut) &
    (df['chi2primsecond']>lowerChi2PrimSecondCut) &
        (df['chi2primsecond']<upperChi2PrimFirstCut)

    is_good_mass = (df['mass']>lowerMassCut) &
        (df['mass']<upperMassCut)

    is_good_df = (is_good_mom) & (is_good_coord) &
        (is_good_params) & (is_good_daughters) & (is_good_mass)

    return df[is_good_df]
```

```
#Concatenate background and signal
df_scaled = pd.concat([signal, background])
del signal, background

#List of variables used for training
cuts = [ 'loverdl', 'distance', 'chi2topo', 'chi2primfirst',
    'chi2primsecond', 'chi2geo']

#Dividing training dataset into Dataframe with variables and
    'issignal' information

x = df_scaled[cuts].copy()
y =pd.DataFrame(df_scaled['issignal'], dtype='int')

#Splitting dataset into training and
    validation sets
x_train, x_test, y_train, y_test =
    train_test_split(x, y, test_size=0.5, random_state=324)
del df_scaled, x, y

#DMatrix is a internal data structure that used by XGBoost
#which is optimized for both memory efficiency and training speed

#Splitting data for training and validati into DMatrices
dtrain = xgb.DMatrix(x_train, label = y_train)
dtest1=xgb.DMatrix(x_test, label = y_test)
del x_test

#Bayesian Optimization function for xgboost

def bo_tune_xgb(max_depth, gamma, alpha, n_estimators,learning_rate):
    params = {'max_depth': int(max_depth),
              'gamma': gamma,
              'alpha':alpha,
              'n_estimators':n_estimators,
              'learning_rate':learning_rate,
              'subsample': 0.8,
              'eta': 0.1,
              'eval_metric': 'auc', 'nthread' : 8}
    cv_result = xgb.cv(params, dtrain, num_boost_round=10, nfold=5,
        verbose_eval=1)
```

```
    return  cv_result['test-auc-mean'].iloc[-1]

#Invoking the Bayesian Optimizer with the specified parameters to tune
xgb_bo = BayesianOptimization(bo_tune_xgb, {'max_depth': (6, 16),
                                             'gamma': (0, 1),
                                            'alpha': (2,12),
                                             'learning_rate':(0,.5),
                                             'n_estimators':(300,1000)
                                            },
                              bounds_transformer=bounds_transformer)
xgb_bo.maximize(n_iter=5, init_points=5)

#Setting hyperparemetres for training
max_param = max_params['params']
param= {'alpha': max_param['alpha'], 'gamma': max_param['gamma'],
    'learning_rate': max_param['learning_rate'],
    'max_depth': int(round(max_param['max_depth'],0)),
    'n_estimators': int(round(max_param['n_estimators'],0)),
    'objective': 'binary:logistic','nthread' : 8}


#Training of the model
bst = xgb.train(param, dtrain)

#We apply our model on the training data that was trained
#on the training data, this helps us to control overfitting

bst1= bst.predict(dtrain)
#we store training results in the bst_train Dataframe
bst_train = pd.DataFrame(data=bst.predict(dtrain),  columns=["xgb_preds"])
y_train=y_train.set_index(np.arange(0,bst_train.shape[0]))
bst_train['issignal']=y_train['issignal']
```

Appendix B

Framgents of code (*ATreePlainer.cpp* class) used for transforming data in AnalysisTree format into PlainTrees format resented here.

```
#include "ATreePlainer.hpp"
#include "AnalysisTree/TaskManager.hpp"

//Initializes variables associated with each AnalysisTree Branch
void ATreePlainer::Init()
{
  //Initating AnalysisTree Task Manager
  auto* man = AnalysisTree::TaskManager::GetInstance();
  auto* chain = man->GetChain();

  //Variables for each AnalysisTree branch
  tofhits_ = ANALYSISTREE_UTILS_GET<AnalysisTree::
    Detector<AnalysisTree::Hit>*>(chain->GetPointerToBranch("TofHits"));
  simulated_= ANALYSISTREE_UTILS_GET<AnalysisTree::
    Particles*>(chain->GetPointerToBranch("SimParticles"));
  tof2sim_match_ = chain->GetMatchPointers().find(config_->GetMatchName(
    "TofHits", "SimParticles"))->second;
  vtxtracks_ = ANALYSISTREE_UTILS_GET<AnalysisTree::Detector<AnalysisTree::
    Track>*>(chain->GetPointerToBranch("VtxTracks"));
  vtx2tof_match_ = chain->GetMatchPointers().find(config_->GetMatchName(
    "VtxTracks", "TofHits"))->second;

  //Initating output PlainTree
  auto out_config = AnalysisTree::TaskManager::GetInstance()->GetConfig();
  AnalysisTree::BranchConfig out_particles("Complex", AnalysisTree::
    DetType::kParticle);
  out_particles.AddField<float>("mass2");
  out_particles.AddField<int>("q");
  man->AddBranch("Complex", plain_branch_, out_particles);
  InitIndices();
```

```
}

//Transformation of AnalysisTree into PlainTree
void ATreePlainer::Exec()
{
  plain_branch_ -> ClearChannels();
  auto out_config = AnalysisTree::TaskManager::GetInstance()->GetConfig();

    //Loop over all reconstructed particles from the TOF detector
   for(auto& input_particle : *tofhits_)
   {
       const auto matched_particle_sim_id = tof2sim_match_->GetMatch(
        input_particle.GetId());
       //Matching with input from MC information about particles
       if (matched_particle_sim_id > 0){
         const auto matched_particle_vtx_id = vtx2tof_match_->GetMatchInverted(
            input_particle.GetId());
         //Matching with data from STS+MVD (VtxTracks branch)
         if (matched_particle_vtx_id > 0){
           //from tof
            auto& output_particle = plain_branch_->AddChannel(
                out_config->GetBranchConfig(plain_branch_->GetId()));
            output_particle.SetField(input_particle.GetField<float>(
                mass2_id_tof_), mass2_id_w1_);
            //from simulated
            auto& matched_particle_sim = simulated_->GetChannel(
                matched_particle_sim_id);
            output_particle.SetMass(matched_particle_sim.GetMass());
            output_particle.SetPid(matched_particle_sim.GetPid());
            //from VtxTracks
            auto& matched_particle_vtx = vtxtracks_->GetChannel(
                matched_particle_vtx_id);
            output_particle.SetMomentum3(matched_particle_vtx.GetMomentum3());
            output_particle.SetField(matched_particle_vtx.GetField<int>(
                q_id_vtx_), q_id_w1_);
            }
          }
      }
}

//Indicies for the output PlainTree
void ATreePlainer::InitIndices()
```

```
{
  //Variables from the input file, which are not defined by default
   auto in_branch_tof = config_->GetBranchConfig("TofHits");
   mass2_id_tof_      = in_branch_tof.GetFieldId("mass2");
   auto in_branch_vtx = config_->GetBranchConfig("VtxTracks");
   q_id_vtx_          = in_branch_vtx.GetFieldId("q");
  //Variables in the output file
   auto out_config = AnalysisTree::TaskManager::GetInstance()->GetConfig();
   const auto& out_branch = out_config->GetBranchConfig(
    plain_branch_->GetId());
   mass2_id_w1_           = out_branch.GetFieldId("mass2");
   q_id_w1_               = out_branch.GetFieldId("q");
}
```

Appendix C

Fragments of code (most important functions) used for TOF identification ML model preparation are presented here (those that do not differ from the K-short recontruction code are presented in Appendix A).

```
#Enumarate class for particles ID
class Pid(Enum):
    ELECTRON = 11
    PROTON = 2212
    MUON = 13
    PION = 211
    KAON = 321
    BCKGR = 999
    @classmethod
    def is_known_particle(cls, value):
        return value in cls._value2member_map_

#Downsampling the data
def downsample(df:pd.DataFrame, label_col_name:str) -> pd.DataFrame:
    # find the number of observations in the smallest group
    nmin = df[label_col_name].value_counts().min()
    return (df
            # split the dataframe per group
            .groupby(label_col_name)
            # sample nmin observations from each group
            .apply(lambda x: x.sample(nmin))
            # recombine the dataframes
            .reset_index(drop=True)
            )

#Sigma region selection
def sigma(df:pd.DataFrame, pid, nsigma=1, info=False):
```

```
    mean = df[df['pid']==pid]['mass2'].mean()
    std = df[df['pid']==pid]['mass2'].std()
    out_sigma = (df['pid']==pid) & ((df['mass2'] < (mean-nsigma*std)) |
        (df['mass2'] > (mean+nsigma*std)))
    df1 = df[~ out_sigma]
    return df1

#Setting probability threshold
def xgb_preds(df, probaProton, probaKaon, probaPion):
    #getting max field
    df['xgb_preds']=df[[0, 1, 2]].idxmax(axis = 1)
    #setting to bckgr if smaller than probability threshold
    proton = (df['xgb_preds'] == 0) & (df[0] < probaProton )
    pion   = (df['xgb_preds'] == 1) & (df[1] < probaKaon )
    kaon   = (df['xgb_preds'] == 2) & (df[2] < probaPion )
    df.loc[( proton | pion | kaon ), 'xgb_preds'] = 3
    return df

#Remapping PIDs for training and
from collections import defaultdict
def remap_names(dataframe):
    return dataframe.pid.abs().map(defaultdict(lambda: 3,
        {Pid.PROTON.value : 0, Pid.KAON.value : 1,  Pid.PION.value : 2,
        Pid.ELECTRON.value : 2, Pid.MUON.value : 2}),
        na_action='ignore')

#Tunning hyperparameters
def bo_tune_xgb(max_depth, gamma, alpha, eta, subsample, n_estimators):
    params = {'max_depth': int(max_depth),
              'gamma': gamma,
              'alpha':alpha,
              'eta':eta,
              'subsample': subsample,
              'n_estimators': n_estimators,
              # 'eta': 0.3, - it is now same as the learning rate
              'num_class':np.unique(dtrain.get_label()).shape[0],
              'objective':'multi:softprob',
              'eval_metric': 'mlogloss',
              # 'tree_method':'hist', 'nthread' : 7}
    cv_result = xgb.cv(params=params, dtrain=dtrain, num_boost_round=10,
        nfold=5)
    return (1 -  cv_result['test-mlogloss-mean'].iloc[-1])
```