NATIONAL RESEARCH UNIVERSITY HIGHER SCHOOL OF ECONOMICS

as a manuscript

Nikita A. Kazeev

MACHINE LEARNING FOR PARTICLE IDENTIFICATION IN THE LHCB DETECTOR

PhD Dissertation

for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Academic Supervisors:
Candidate of Sciences Andrey Ustyuzhanin
Dr. Barbara Sciascia
Prof. Davide Pinci

Moscow – 2020

Федеральное государственное автономное образовательное учреждение высшего образования«Национальный исследовательский университет«Высшая школа экономики»

На правах рукописи

Казеев Никита Алексадрович

ПРИМЕНЕНИЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ К ИДЕНТИФИКАЦИИ ЧАСТИЦ В ДЕТЕКТОРЕ LHCB

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата компьютерных наук

Научные руководители:
кандидат физико-математических наук
Устюжанин Андрей Евгеньевич,
доктор философии Барбара Шаша,
профессор, доктор философии Давиде Пинчи

Москва – 2020

SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning for particle identification in the LHCb detector

Il Dottorato in Fisica

Dottorato di Ricerca in Fisica – XXXII Ciclo

Candidate

Nikita Kazeev
ID number 1834931

Thesis Advisors

Dr. Barbara Sciascia
Dr. Andrey Ustyuzhanin
Dr. Davide Pinci

2019/2020

Machine Learning for particle identification in the LHCb detector
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: kazeevn@gmail.com

To my mother Inna

Abstract

LHCb experiment is a specialised b-physics experiment at the Large Hadron Collider at CERN. It has a broad physics program with the primary objective being the search for CP violations that would explain the matter-antimatter asymmetry of the Universe. LHCb studies very rare phenomena, making it necessary to process millions of collision events per second to gather enough data in a reasonable time frame. Thus software and data analysis tools are essential for the success of the experiment.

Particle identification (PID) is a crucial ingredient of most of the LHCb results. The quality of the particle identification depends a lot on the data processing algorithms. This dissertation aims to leverage the recent advances in machine learning field to improve the PID at LHCb.

The thesis contribution consists of four essential parts related to LHCb internal projects. Muon identification aims to quickly separate muons from the other charged particles using only information from the Muon subsystem. The second contribution is a method that takes into account a priori information on label noise and improves the accuracy of a machine learning model for classification of this data. Such data are common in high-energy physics and, in particular, is used to develop the data-driven muon identification methods. Global PID combines information from different subdetectors into a single set of PID variables. Cherenkov detector fast simulation aims to improve the speed of the PID variables simulation in Monte-Carlo.

Acknowledgments

I want to thank my supervisor, Andrey Ustyuzhanin, for his unwavering support of my research and character development. Without his leadership and insight, the whole data science for physics collaboration would not have happened. I want to thank my supervisor, Barbara Sciascia, for her compassion and dedication; she was the relentless force behind actually writing the manuscript. A big thank to Denis Derkach, whose high-energy-physics expertise has been essential to all of my work.

The projects would not have been possible without collaborators, to whom I am perpetually grateful.

Patrizia De Simone developed the TMVA model for muon identification and provided me with the 2012 dataset. Marco Santimaria assisted greatly with getting the 2016 dataset from CERN GRID. Barbara Sciascia secured the approval for the data release for IDAO. All of the above, with the addition of Matteo Palutan and Ricardo Vazquez Gomez, provided invaluable consultations on the muon detector physics and code. I am also thankful to Marco Cattaneo, Ben Couturier, Gerhard Raven, Roel Aaij for their advice on LHCb software.

Machine learning with background subtraction was a collaboration with Maxim Borisyak. He developed the Likelihood method and did the neural network experiments. I am also thankful to Stanislav Kirillov for a consultation on CatBoost code.

Mikhail Hushchyn developed the deep neural network-based model for global particle identification.

The models for the LHCb RICH fast simulation were trained by Artem Maevskiy – using my FastDIRC pilot code. The LHCb RICH plots were produced by Artem Maevskiy and Maxim Artemev, the high-level Lamarr plots by Lucio Anderlini.

Bruno Bauwens and Tatiana Makhalova reviewed the no free lunch theorem part.

I am broadly thankful to JustHeuristic for his machine learning insights, never-ending optimism, and dedication for the pursuit of knowledge. Ekaterina Artemova advised me about NLP. Mikhail Danilov taught me most of what I know of particle physics.

Last but not least, I am eternally grateful to my parents, who brought me up, and supported me with love and affection on my path through life.

Contents

Chapter 1

Introduction

1.1 New Physics and the LHCb Experiment in Search of It

The theory of strong and electroweak interactions, the so-called Standard Model (SM) of particle physics, has achieved outstanding success. Its predictions have been confirmed by all the experiments conducted so far. Yet there are unexplained experimental phenomena, such as the nature of the dark matter, the origin of mass of the neutrinos, the lack of explanation for the predominance of matter over anti-matter in the universe. They suggest that the SM is only an effective theory at the energies explored so far and that a more complete theory should exist.

A way to looking for New Physics is the study of fundamental properties within the SM. One of the more appealing currently is the lepton universality which requires equality of couplings between the gauge bosons and the three families of leptons. Hints of lepton non-universal effects in $B^+ \to K^+e^+e^-$, $B^+ \to K^+\mu^+\mu^-$ [1, 2] decays have been reported. But there is no definitive observation of a deviation yet. A large class of models that extend the SM contains additional interactions involving enhanced couplings to the third-generation that would violate the lepton universality principle [3]. Semileptonic decays of b hadrons to third-generation leptons provide a sensitive probe for such effects. In particular, the presence of additional charged Higgs bosons, which are often required in these models, can have a significant effect on the rate of the semitauonic decays of b-quark hadron $b \to c\tau^+\nu_\tau$ [4].

When looking for more complete theories, the physics beyond the Standard Model, one of the best places to start is where existing theory says an event is not likely to happen: any deviations will be large compared to what we expect. For example, on the one hand, the branching fractions BR$(B_{d,s} \to \mu^+\mu^-)$ are very small in the SM and can be predicted with high accuracy. On the other hand, a large class of theories that extend the Standard Model, like supersymmetry, allows significant modifications to these branching fractions and therefore an observation of any significant deviation from the SM prediction would indicate a discovery of new effects.

These decays have been extensively studied, most recently at the LHC experiments: LHCb [5, 6], CMS [7] and ATLAS [8]. There is also the combined analysis of the two results from the joint efforts of LHCb and CMS collaborations [9]. Thus far the decay $B_s \to \mu^+\mu^-$ has been observed, but only the upper limits on the branching fraction of $B_d \to \mu^+\mu^-$ have been reported.

Finding and studying rare decays means pushing the frontiers of the experiment design and data analysis. During its lifetime, the LHC provides an unprecedented, but still finite number of collision events. And the rarer the process, the more is required from the experimental hardware and software for the measurement to be statistically significant. It is fundamental to develop selection criteria, muon identification, and background parametrisation to enable the discovery of $B_d \to \mu^+\mu^-$, and place even more stringent limits on supersymmetry and other new physics models.

The LHCb experiment [10] is designed to exploit the high $pp \to c\bar{c}$ and $pp \to b\bar{b}$ cross-sections at the LHC in order to perform precision measurements of CP violation and rare decays.

Physics analyses using data from the LHCb detector [11] rely on Particle Identification (PID) to separate charged tracks of different species: pions, kaons, protons, electrons, and muons. PID is conceptually straightforward. Charged particles emit Cherenkov light when traversing the Ring-Imaging Cherenkov detector (RICH). It allows measuring the velocity, which, together with the momentum, allows to reconstruct the particle mass. Electrons are absorbed by the electronic calorimeter, hadrons by the hadronic. Muons penetrate the detector and produce hits in the muon chambers. PID relies on sophisticated algorithms to optimise its performance. First, the raw information from each PID subdetector is processed into a handful of high-level variables. Second, it is combined to make the final decision on the particle type. Such setup allows us to take advantage of the fast analysis of the data from muon and calorimeter subsystems to use them in the low-level trigger [12].

As of the moment, the LHCb experiment is undergoing an upgrade and is scheduled to start taking data in 2021. After the upgrade, the average number of visible interactions will increase by more than a factor 5, complicating event reconstruction [13]. The role of the PID in achieving the physics goals of the upgraded experiment will remain critical.

But processing experimental data is only half of the story. Simulation is of major importance for the design and construction of an experiment, as well as the development of the algorithms to analyse its data [14]. It comes with a price. Monte-Carlo generation took around 75% of CPU time in the LHCb GRID in Run 2 [15]. With the planned luminosity increase, this cost will become unsustainable and must be addressed [16].

An alluring alternative to using simulation for evaluation and development of PID algorithms are data-driven methods. The PID responses are known to be reproduced with an accuracy not sufficient for most of the analyses. The data-driven methods are based on calibration samples: samples of charged tracks of different species that have been selected without the use of PID response to the track in question. Being a product of the real world, these calibration samples come with a set of complications – the possible bias introduced by selection and presence of background.

1.2 Machine Learning

The idea of creating "intelligent" machines has been pursued since the inception of modern computing in the 1950-s. The field has seen its ups and downs – cycles of hope and disappointment. Now it is yet again hope. The currently most successful paradigm of artificial intelligence is machine learning. In very broad terms, machine learning allows a program to learn from provided examples, instead of having its behaviour explicitly programmed by a human. Classic least squares curve fitting can be viewed as the most primitive example. But the beauty and power of the machine learning methods lie in their ability to handle data with a high number of dimensions with little a priori knowledge about the problem. For example, classifying images with a modest size of 256×256 pixels already presents a problem with 6.5×10^4 dimensions.

How to recognise whether there is a cat on an image? If you were in the 2000-s, you would think of an algorithm. Find the legs, ears, check their shape, check for whiskers' presence. Things changed in 2012 when a machine-learning approach won the ImageNet Large Scale Visual Recognition Challenge and cut the classification error from 25% to 16% [17], and surpassed human-level performance in 2015 [18].

High-energy physics is a natural beneficiary from these advances. Its experiments produce data at a rate of millions of events per second, necessitating the development of algorithmic methods of data analysis and techniques for their validation. These include reasonably accurate simulations that can serve to provide training data for machine learning algorithms. This thesis is devoted to using state-of-the-art machine learning methods to advance a key part of the LHCb experiment – particle identification.

1.3 My Contribution

My work consists of four projects. Two of them are directly concerned with improving PID quality: Global PID and Muon ID. One improves the speed of Monte-Carlo simulation of a Cherenkov detector. And one is a machine-learning technique that allows dealing with background-subtracted samples (a case of noisy labels common in high-energy physics); the need for it appeared during the work on data-driven training of the Muon ID model.

Muon ID The objective of muon identification is to distinguish muons from the rest of the particles using only information from the muon subdetector. Since the algorithm is to be used early in the data selection pipeline, there are stringent requirements on CPU time. Muon identification is essential for the LHCb physics program, as muons are present in the final states of many decays sensitive to new physics that are studied by the LHCb experiment [19, 5, 20]. My goal has been the improvement of the muon identification quality. The project is described in chapter 5.

Machine learning on data with sPlot background subtraction Experimental data obtained in high energy physics experiments usually consists of contributions from different event sources. In LHCb most analyses and data-driven PID development has to deal with a mixture of signal and background. A common way of subtracting background, sPlot [21], introduces negative event weights. Training a machine learning algorithm on a dataset with negative weights means dealing with a loss that potentially has no lower bound and does not always converge. One of the goals of the thesis has been developing a robust way to apply machine learning to such data. In machine learning terms, this is a particular model of label noise. For each example, we know the probability that its label has been flipped. The distribution of flipping probabilities is independent of features' distribution for each class. The project is described in chapter 6.

Global PID PID subdetectors provide a wealth of information which must be processed into the final decision on the particle type. The goal in this thesis has been to use the state-of-the-art machine learning algorithms at the last step of PID to improve the PID quality. The corresponding chapter is 7.

Fast simulation The RICH simulation takes around 30% of the CPU time [15]. At the same, some PID variables are not well enough described by the simulation [22]. The objective of the project has been to develop fast data-driven simulation of RICH detector. The project is described in chapter 8.

The thesis is organised in the following way. An overview of machine learning is given in Chapter 2, while Chapter 3 is dedicated to the description of the use of machine learning in high-energy physics. The following Chapter 4 introduces the LHCb experiment, while the next four chapters describe my specific contributions as detailed above.

Chapter 2

Machine Learning

2.1 A Very Brief History of Artificial Intelligence

Quoting Encyclopædia Britannica [23], Artificial intelligence (AI) is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalise, or learn from experience. The idea of machines as intelligent as humans has been present at least from the antiquity and, probably, prehistory. The Iliad, written around 850 B. C., codified what were most likely already ancient traditions – Hephaestus, the god of fire and the divine smith, after being cast out of Olympus, forges intelligent automata assistants [24, 25]. This dream became an ambition with the advent of modern computing. Quoting the assertion of Dartmouth Conference of 1956: "every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it" [26]. Very roughly, the most widely explored paradigms that attempt to make a general, i. e. capable of all tasks a human can do AI, can be categorised as "reasoning as search" and "reasoning as speech".

Reasoning as search Define a problem as a game, where there are a goal and actions (steps) that the program can take to reach it. The problem can be an actual game, like chess, a formal logic reasoning task or an applied one, like controlling a robot. The program would explore different paths through the actions graph to reach a solution. The difficulty with this approach is the "combinatorial explosion" – the number of possible paths to take increases extremely rapidly with the problem complexity. Practical applications used heuristics to select exploration paths that are likely to lead to the solution [24, 27].

While impractical for early computers, this method achieved landmark success, when IBM Deep Blue in 1997 defeated the reigning world chess champion. This was made possible by a number of factors, among them a massively parallel system with special-purpose hardware and a complex path evaluation function [28].

Modern paradigms like reinforcement learning can be seen as a logical development of the reasoning as search approach. With one crucial difference – heuristics that make exploring the search space tractable are not programmed explicitly but are learned via trial-and-error by the program itself. In recent years, reinforcement learning research resulted in several high-profile achievements where an AI was able to achieve super-human performance: classic Atari games [29], Go [30], StarCraft II [31].

Reasoning as speech One of the most widely-known definitions of a “thinking machine” was proposed by computer science pioneer Alan Turing in 1950 [32]: “If a machine could carry on a conversation (over a teleprinter) that was indistinguishable from a conversation with a human being, then it was reasonable to say that the machine was thinking”. Until the 1980-s, processing of natural languages relied on increasingly complex manually developed systems of grammar rules and conceptual ontologies, which structured real-world information into computer-understandable data. This approach ran into limitations. Hand-crafted rules and knowledge graphs are expensive to create and maintain – and they still fail to capture the full complexity of natural languages and the world.

Starting from the 1980-s, the availability of computing power made it possible to use statistical approaches to learn some parts of the system from textual corpora. This culminated in very recent deep learning approaches, where general end-to-end sequence learning approaches were able to achieve the state-of-the-art performance of such tasks as machine translation [33] and speech recognition [34].

Looking back into overall history, AI research has gone through three cycles of optimism (1956–1974, 1980–1987, 2011–) and two of disappointment (1974–1980, 1987–1993). To quote a few of the first-generation optimists:

- 1958, H. A. Simon and A. Newell: ”within ten years a digital computer will be the world’s chess champion” and ”within ten years a digital computer will discover and prove an important new mathematical theorem.” [35]
- 1965, H. A. Simon: ”machines will be capable, within twenty years, of doing any work a man can do.” [36]
- 1967, M. Minsky: ”Within a generation ... the problem of creating ’artificial intelligence’ will substantially be solved.” [36]

Evidently, they failed to deliver on those promises. In retrospective, it seems that the driving force was the lack of computing power – the world is simply too complex to fit into 8.39 Megabytes and be processed by a single 80 MHz CPU (parameters of Cray-1, a 1975 state-of-the-art supercomputer [37]) [27]. The funding agencies noticed this failure. The most prominent example, the Lighthill report [38] result ”formed the basis for the decision by the British government to end support for AI research in all but two universities” [27].

Despite the current failure to deliver a general AI, the methods that were developed in the pursuit for it, foremost for computational statistics and solving difficult multidimensional optimisation problems, have proven themselves quite capable when facing narrowly defined challenges. The resulting “applied AI” discipline is known as machine learning, and it is covered in the next sections. Its success varies by field from non-existent to Nature [30] and Science [39] front covers (with results, not promises).

2.2 Machine Learning Formalism

Machine Learning is a product of a marriage between statistics and algorithms. Roughly speaking, it makes computers ”learn from example” and ”learn by experience”.

Let us define two domains, the features domain $\mathbb{X}$ and the target $\mathbb{Y}$ (also called label). Let the features domain be a real vector space, so that we may call an object belonging to that domain a vector of features[1]. Target domain can be either a real vector space or a finite set. Call a 2-tuple with the first element from the features domain and the second element from the target domain an example. Let us also have a probability distribution over examples with the probability density $p(\mathbf{x}, \mathbf{y})$, which is unknown to us. Let us have a sample from this unknown distribution – a series of points in $(\mathbf{x}, \mathbf{y})$ (example) space, which we will call the training dataset[2]. Sometimes, the dataset might be weighted. A common use case for this is correcting a biased sampling procedure. If the dataset is sampled from distribution $\tilde{p}(\mathbf{x}, \mathbf{y})$, than it can be weighted with $w(\mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y})}{\tilde{p}(\mathbf{x},\mathbf{y})}$ to match the desired probability density $p(\mathbf{x}, \mathbf{y})$. In most cases, the weights are non-negative, $w \geq 0$. Negative weights appearing in high-energy physics from the application of the sPlot [21] method are in-depth discussed in chapter 6.

The objective of machine learning is to use the training dataset to build models for different scenarios:

Classification Consider the case, where the target domain is a finite set, for example, species of cats or particles in the LHCb detector. Let us enumerate the target domain, and define γ_i as the i-th possible value. We say that an example belongs to class i if its target value $\mathbf{y}$ equals to γ_i. A classification model is a function that takes as its input a vector of features and outputs a vector of numbers, that are estimates of the probabilities that an example with given features belongs to the i-th class, $P(\mathbf{y} = \gamma_i|\mathbf{x}) = \frac{p(\gamma_i,\mathbf{x})}{\sum_j p(\gamma_j,\mathbf{x})}$, or some monotonic function of those probabilities.

Regression Consider the case, where the target domain is a real-valued vector space. For example, the energy deposited in a calorimeter by a particle. A regression model is a function from the features domain into the target domain. Its output is an estimate of the expected value of the target $\mathbf{y}$ conditioned features value $\mathbf{x}$ [40]:

$$E[\mathbf{y}|\mathbf{x}] = \int \mathbf{y} \frac{p(\mathbf{x}, \mathbf{y})}{\int p(\mathbf{x}, \mathbf{y}')d\mathbf{y}'} d\mathbf{y}. \tag{2.1}$$

[1]There can also be features that have only a finite set of non-numeric possible values, such as particle species. Such features are called categorical. They do not violate the proposed formalism, as a finite set can always be mapped to a real space.

[2]With this definition, we omit sequence learning, reinforcement learning, and unsupervised learning. They are fantastic but are not used in the dissertation.

A Generative model describes how a dataset is generated, in terms of a probabilistic model. By sampling from this model, we are able to generate new examples from the underlying distribution [41]. For example, when a particle reaches a calorimeter, a highly complicated shower process occurs inside it, that results in energy deposits in the calorimeter pixels. A generative model would allow sampling from the distribution of those responses.

Mathematically, an unconditional generative model, can be defined as consisting of two components. The first one is a probability distribution over real vectors, usually taken to be a simple one, like uniform or normal. The second component is a function from the space of real vectors to the target domain. If a function's input is a random variable whose distribution is the first component of the model, the distribution of the function value which is also a random variable is an approximation of $p(\mathbf{y})$[3]. The definition is dictated by the computational side of machine learning. A practical generative model usually consists of a (pseudo)random number generator whose output serves as input to a deterministic computer program.

The generative model definition can naturally be extended to the conditional case to sample from $p(\mathbf{y}|\mathbf{x})$. In this case, the model would still consist of two components. The first one is a probability distribution, just like in an unconditional model. The second component is a function of two arguments. The first argument is from the space of real vectors. The second argument is from the features domain. The function value is in the target domain. If a function's first argument is a random variable whose distribution is the first component of the model, and the second argument equals to $\mathbf{x}$, the distribution of the function value is an approximation of $p(\mathbf{y}|\mathbf{x})$.

Density estimation model is a function from the domain of examples into real numbers, which value is an estimate of the value of the probability density corresponding to that example, $p(\mathbf{y})$[4]. The most basic example of such a model is a histogram. In a sense, density estimation is similar to generative modelling, but there is a considerable practical difference. A generative model does not provide the explicit values of the probability density function, just a stream of examples. Density estimation does not allow to produce a sample, only to evaluate the density value on a sample given to it. Sometimes, density estimation and sampling are deemed not separate classes of models, but subclasses of generative models.

2.2.1 Model and Training

The problem searching for the best model in the abstract function space, while mathematically elegant, is not a convenient one for a computer. Therefore, a machine learning model is commonly defined as a parametric function $f(\mathbf{x}, \boldsymbol{\theta})$, where $\mathbf{x}$ is an element of the features domain and $\boldsymbol{\theta}$ are the parameters of the model, and f is fixed. Mathematically, $\boldsymbol{\theta}$ can be different objects, but usually, it can be represented as a real vector.

[3]There is no distinction between $\mathbf{x}$ and $\mathbf{y}$ for an unconditional model. We use $\mathbf{y}$ for the sake of consistent terminology.

[4]See footnote 3

The degree of correspondence between a model and data is defined in terms of some error (loss) metric. The metric usually takes the form of a function of two arguments from the target domain, $L(\mathbf{y}_{\text{predicted}}, \mathbf{y}_{\text{true}})$, where $\mathbf{y}_{\text{predicted}}$ is the value predicted by the model and $\mathbf{y}_{\text{true}}$ is the true label. It is desirable to match the loss function and the utility of the model for the purpose it is being created, such as uncertainty of a branching fraction measurement. However, it is often difficult to derive such direct relationship, and a proxy loss function is used. A common example for regression is the l^2 norm (or mean squared error) $L(\mathbf{y}_{\text{predicted}}, \mathbf{y}_{\text{true}}) = |\mathbf{y}_{\text{true}} - \mathbf{y}_{\text{predicted}}|^2$. The most used loses are discussed in subsection 2.3.

The grand objective of training is to find the parameters' values that would minimise the expected loss over all possible examples:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \left[E_{(\mathbf{x},\mathbf{y})\sim p} L(f(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y}) \right]. \tag{2.2}$$

The expectation is estimated using the training dataset; this is called empirical risk minimisation:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{N} \sum_{i=1}^{N} L(f(\mathbf{x}_i, \boldsymbol{\theta}), \mathbf{y}_i) \right], \tag{2.3}$$

where N is the number of examples in the training dataset and $\mathbf{x}_i$ and $\mathbf{y}_i$ are examples' from the training dataset features and labels, respectively. In the case of weighted dataset:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \left[\frac{\sum_{i=1}^{N} w_i L(f(\mathbf{x}_i, \boldsymbol{\theta}), \mathbf{y}_i)}{\sum_{i=1}^{N} w_i} \right], \tag{2.4}$$

where w_i is the example weight.

Given a fixed dataset, for sufficiently flexible model classes, this can result in $\mathbf{y}_i \approx f(\mathbf{x}_i, \boldsymbol{\theta})$ for the training points – for example for a polynomial fit, where the degree is equal to the number of data points. We are interested in the generalisation power of the model – its performance on the examples it has never seen. The predictions for examples the model has never seen are almost always less accurate than for examples used in training – and this gap tends to grow with model complexity. See the example in figure 2.1. A model, that is too complex for the problem and whose performance on the training dataset it much higher than on examples it has never seen, is commonly called overfitted. A model that is is not flexible enough for the problem performs similarly poorly on training, and new data is known as underfitted.

A common way of dealing with the problem of overfitting is regularisation – penalising model complexity:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{N} \sum_{i=1}^{N} L(f(\mathbf{x}_i, \boldsymbol{\theta}), \mathbf{y}_i) + R(\boldsymbol{\theta}) \right], \tag{2.5}$$

where $R(\boldsymbol{\theta})$ is a function penalising complex models, its form depending on the model in question. A common example is l^2 regularisation: $R(\boldsymbol{\theta}) = \rho|\boldsymbol{\theta}|^2$, where $\rho \in [0, +\infty)$ is a parameter defining the strength of the regularisation. It is used in some experiments in chapter 6.

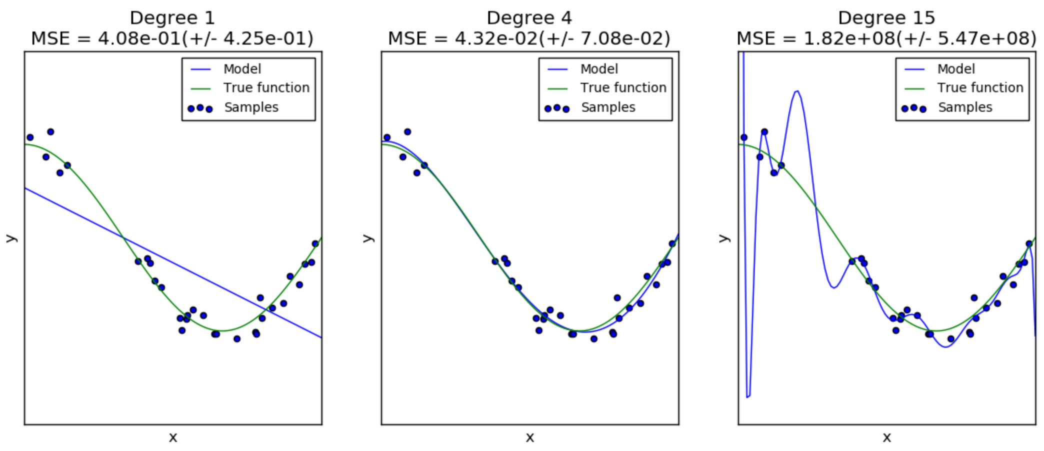


Figure 2.1. An example of underfitting and overfitting. The problem, both features x and target y are real scalars. The polynomial of degree 1 is a case of underfitting, of degree 15 – overfitting. MSE in the title refers to the mean squared error, MSE $= \sum_{i=1}^{N} \left(y_{\text{predicted}} - y_{\text{true}}\right)^2 / N$, where N is the number of test examples. Reproduced from [42].

2.2.2 Hyperparameters

Usually, a machine learning model is a member of a specific parametric function family. The parameters that define this particular model from its family, e. g. the degree of a polynomial for polynomial regression, along with the parameters of the optimisation procedure are commonly called hyperparameters. Hyperparameters are not changed during model training. If there is an automated procedure for finding the optimal value of hyperparameters, than from the point of view of that procedure, they become just model parameters.

2.3 Measuring Model Quality

A machine learning model offers an imperfect approximation of the target distribution. Thus, for practical applications, this imperfection must be measured and accounted for.

A common way to do this is by splitting the dataset into a train and test subsets. The model is trained on the training dataset, and the performance is evaluated on the test dataset. The advantage of this approach is simplicity and small computational cost. The disadvantage is data use inefficiency: both train and test dataset sizes are reduced compared to the full dataset. Since model training relies on empirical risk minimisation, the model quality usually increases with the dataset size, and this method would result in underestimating performance. The loss function value on the testing dataset is an estimation of the expected value with a finite sample. Thus, a small testing sample leads to higher statistical uncertainty.

A commonly used method that helps to deal with the data inefficiency is cross-validation illustrated in figure 2.2. The dataset is split into k subsets of equal size, commonly called folds. Then the model is independently trained k times, using all subsets, but the i-th, which is used for evaluation. Then the metric value over all

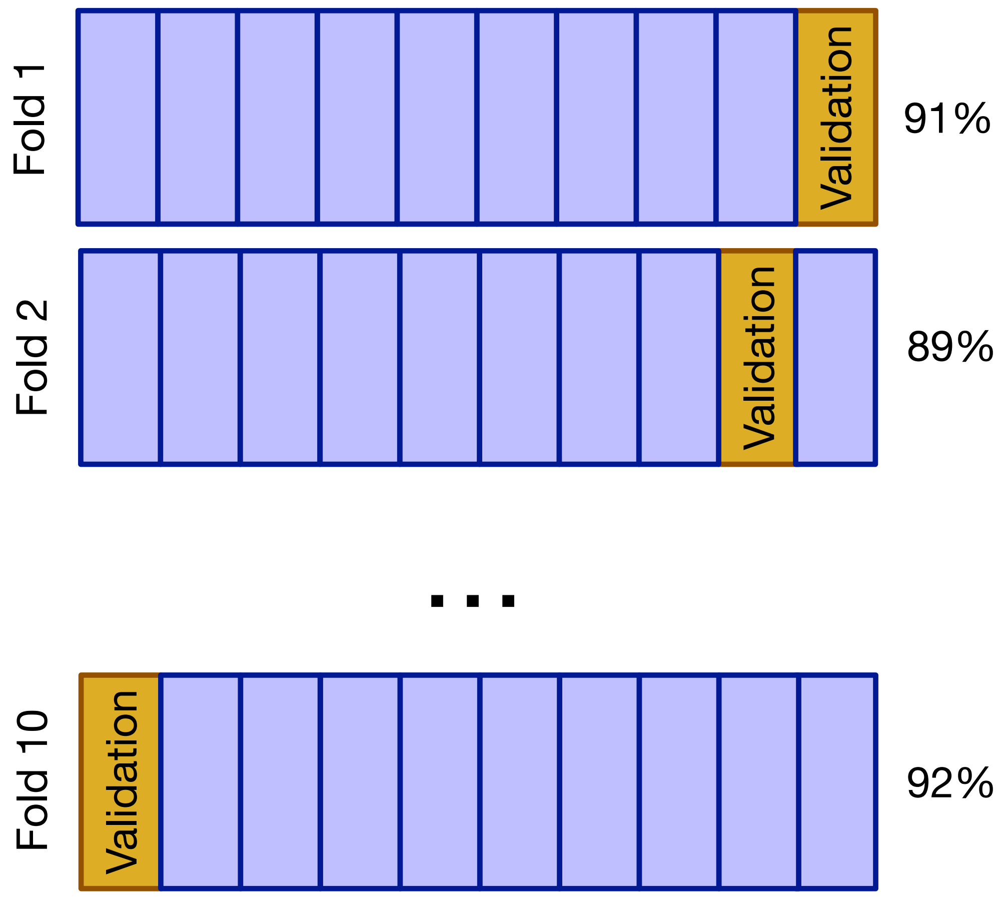


Figure 2.2. Illustration of 10-fold cross-validation. The dataset is split into 10 parts, represented as the vertical rectangles. The train datasets are combined from 9 those parts – 10 variants in total, 3 of which are represented as the horizontal stripes. Reproduced from [42].

the subsets is averaged. Common values for k range from 3 to 10. The advantage of cross-validation is that the train size is closer to the full size, and the full dataset is used for testing. The disadvantage is the computational cost, as we have to train the model k times.

The choice of the quality metric is important and non-trivial. Say, the true target values are $[1, 0, 0]$. Which model is better, the one that predicts $[0, 0, 0]$, or $[0.6, 0.4, 0.4]$? Ideally, the quality metric is defined to correspond to the performance of the whole data analysis system. For example, the Higgs Machine Learning Challenge [43] used the approximate median significance – an estimate of the significance of the result of an analysis that uses the model. Yet, for a sufficiently complex data analysis system, establishing a direct relationship between the model output and the overall system performance and distilling it into a single number is borderline impossible.

There are several general-purpose metrics that are commonly used by machine learning practitioners, the most common of which we describe below.

2.3.1 Accuracy

For a classification problem, transform the model output into the definite class labels. Compute the proportion of the correct predictions.

$$y_i^{\text{pred}} = \arg\max_k f_i^k \tag{2.6}$$

$$\text{Accuracy} = \frac{\sum_{i=1}^N [y_i^{\text{pred}} = y_i]}{N}, \tag{2.7}$$

where f_i^k is the value of the model prediction of the k-th class for the i-th example, y_i is the true class label for the i-th example, N is the number of examples, $[y_i^{\text{pred}} = y_i]$ is an Iverson bracket that equals 1 if $y_i^{\text{pred}} = y_i$ and 0 otherwise. In the case of a weighted dataset:

$$\text{Accuracy} = \frac{\sum_{i=1}^N w_i [y_i^{\text{pred}} = y_i]}{\sum_{i=1}^N w_i}. \tag{2.8}$$

While natural from a common-sense point of view, as a practical measure, accuracy has three major drawbacks. First, it discards information, accuracy score of a model that predicts 0.7 and 0.999 in a binary classification problem are the same. Second, it is misleading in the case of one class being much more common than the others. The accuracy score of a trivial model that always predicts the most frequent class will be the fraction of said class in dataset. Third, its gradient with the respect to prediction is zero almost everywhere, and thus accuracy can not be used for training models, that rely on the differentiability of the loss, such as gradient boosting decision trees (section 2.6) and neural networks (section 2.5).

Accuracy is sometimes used [44] for abstract comparison of machine learning algorithms, due to its interpretability. This is especially relevant in the case of more than two classes.

2.3.2 Mean Squared Error (MSE)

$$\text{MSE} = \frac{\sum_{i=1}^N (f_i - y_i)^2}{N}, \tag{2.9}$$

where f_i is the model prediction for the i-th example. In the case of a weighted dataset:

$$\text{MSE} = \frac{\sum_{i=1}^N w_i (f_i - y_i)^2}{\sum_{i=1}^N w_i}. \tag{2.10}$$

MSE loss is one of the commonly used losses for training regression models, owning to its mathematical convenience. For example, least-squares fitting is used in track fitting at LHCb [45, 46], and, according to reference [47] in an overwhelming majority of experimental implementations. A square root from MSE was the most used regression metric at the data science competition platform Kaggle in 2015[5] [48]. From the probabilistic point of view, a least square estimate is the same as a maximum likelihood estimate for a Gaussian model.

[5]Surprisingly, I could not find current statistics.

MSE weights already large errors more heavily than the small ones and is heavily affected by outliers [49]. In case a dataset is contaminated by examples with large mistakes in their target values, they might dominate the error.

2.3.3 LogLoss

Consider a classification problem. Let $f_i^{(k)}$ be the predicted probability that example i belongs to the class k. Note, that (k) here is not a power, but an index. Let y_i be the true class to which the i-th example belongs. Since the examples are independently sampled, the probability of all predictions being correct is:

$$P(\text{all predictions are correct}) = \prod_{i=1}^{N} f_i^{(y_i)}. \tag{2.11}$$

To simplify the computation, take the negative logarithm:

$$\text{LogLoss} = -\sum_{i=1}^{N} \ln\left(f_i^{(y_i)}\right). \tag{2.12}$$

In the case of a weighted dataset:

$$\text{LogLoss} = -\frac{\sum_{i=1}^{N} w_i \ln\left(f_i^{(y_i)}\right)}{\sum_{i=1}^{N} w_i}. \tag{2.13}$$

Logloss is closely related to cross entropy from information theory [50], these terms are sometimes used interchangeably in the context of machine learning. Logloss is a common loss function used for classification. The optimal values for predictions of a model trained with logloss are the supremely interpretable class probabilities. It is differentiable. Its drawback is the lack of the upper bound, that might result in unstable computation. A common method of addressing this is adding a small constant to the logarithm argument. Also, the values obtained by the real-world models are not always good probability estimates.

2.3.4 Area Under the Receiver Operating Characteristic (ROC AUC)

Consider a binary classification problem: $\mathbb{Y} = \{0, 1\}$. Let $f_i \in \mathbb{R}$ be the model output for i-th example, the greater it is, the more likely the example is class 1. If we need to make a binary decision using this prediction, for example whether to keep an event for further analysis, we define a threshold t and transform the continuous values f_i into the definite class labels $y_i^{\text{pred}} = [f_i > t]$. From a statistician's point of view, when classifying, we make a decision about the null hypothesis that the example is class 1. The decisions that we reach will suffer from two kinds of errors: false positive (or type 1), where we accept a background example, and false negative (or type 2), where we reject a signal one. By varying the threshold, the trade-off between the error types can be adjusted. The choice of the threshold (also called a working point) depends on the problem for which the model is designed and the costs of making the errors of different types. For example, in high-energy physics event selection, the overall output rate is usually limited by computing constraints.

For each threshold value, we can compute the true positive rate (TPR) or, in high-energy physics terms, the signal efficiency, and also known as sensitivity:

$$\mathrm{TPR} = \frac{\sum_i [y_i^{\mathrm{pred}} = 1]}{\sum_i [y_i = 1]}. \tag{2.14}$$

We can also compute the false positive rate (FPR), in high-energy physics terms, $1 -$ background rejection:

$$\mathrm{FPR} = \frac{\sum_i [y_i^{\mathrm{pred}} = 1]}{\sum_i [y_i = 0]}. \tag{2.15}$$

TPR is the fraction of the signal examples, that pass the selection threshold. FPR is a fraction of background examples that erroneously pass the selection threshold. Note, that TPR and FPR are measured using different sets of examples.

In the case, where there is no a priori information that would allow choosing the decision threshold, a common measure is to plot the TPR as a function of FPR. While allowing for maximum flexibility of evaluation, a curve is not a convenient scalar performance score. The commonly used summary statistic is the area under the curve (AUC). AUC is the average value of TPR for all possible values of FPR. An illustration of the relationship between TPR, FPR and AUC is presented in figure 2.3. If some information about the desired threshold region is available, it can be incorporated in the form of taking the area under a part of the ROC curve, or just a point on the curve.

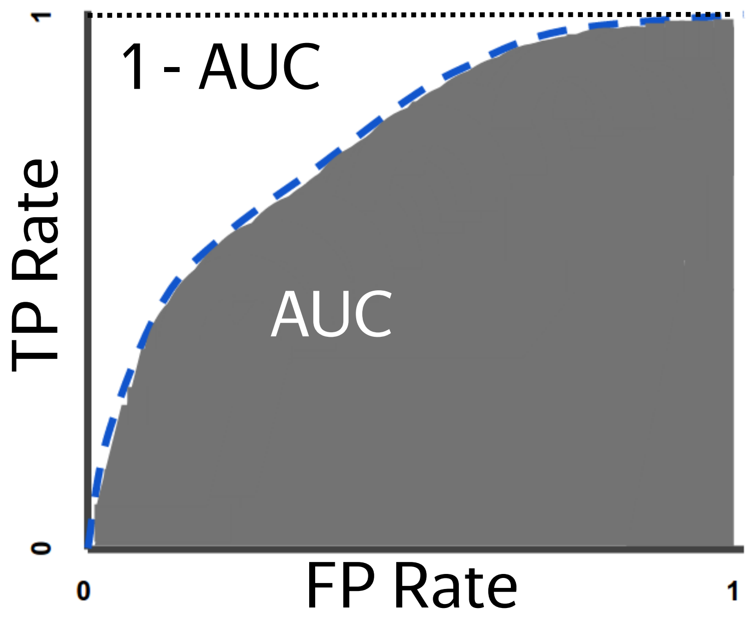


Figure 2.3. An illustration of a receiver operating characteristic (ROC) and the area under the curve (AUC). Source: Google Inc.

Another interpretation of ROC AUC is the fraction of correctly ordered pairs or the Mann-Whitney U-statistic for comparing distributions of values from the two samples [51]. Let S be the number of signal examples, B the number of background examples, s_i – model prediction for the i-th signal example, b_j – model prediction for the j-th background example.

$$\psi(s,b) = \begin{cases} 1 & b < s \\ \frac{1}{2} & b = s \\ 0 & b > s \end{cases} \tag{2.16}$$

$$\text{ROC AUC} = \vartheta = \frac{1}{SB}\sum_{i=1}^{S}\sum_{j=1}^{B}\psi(s_i,b_j). \tag{2.17}$$

ROC AUC has mathematical properties that make it attractive for a general-propose metric for comparing classifiers. ROC AUC is finite and lies in a well-defined interval. For a perfect classifier, all predicted values for signal examples are greater than for all background examples, therefore ROC AUC $= 1$. For a totally random classifier, the predictions are distributed equally for signal and background examples, and the ROC curve is a straight line from $(0,0)$ to $(1,1)$ and ROC AUC $=$ 0.5. The value of ROC AUC stays the same if any strictly monotonic transformation is applied to the predictions, unlike the values of accuracy and logloss. Compared to logloss, it is less sensitive to outliers. ROC AUC usage is ubiquitous in the machine learning community.

ROC AUC is defined for binary classification. If we have several classes to distinguish, we can replace a single ROC AUC metric with a set of numbers. There are two common ways to transform a n-class classification into sets of binary classification problems: one-vs-rest and one-vs-one. One-vs-rest reduces a multiclass classification problem to n binary classification problems. For each class it considers the value of the metric, computed with the other classes collapsed into a single virtual class. One-vs-one reduces a multiclass problem with n classes into a $\frac{n(n-1)}{2}$ binary classification problems where each class is pitted against every other class.

Note that the gradient of ROC AUC with respect to model prediction is zero almost everywhere. Because of this, ROC AUC is rarely optimised directly by machine learning models, despite being, probably, the most popular binary classification metric.

Computation of the statistical uncertainty for the ROC AUC value is not trivial. For a metric, that can be expressed as $\mathbb{E}(L(y_{\text{predicted}}, y_{\text{true}}))$, the mean is estimated from a set of *independent* observations, which opens it to straightforward statistical analysis. ROC AUC is computed over a whole dataset and can not be expressed as a sum of some loss values $L(y_{\text{predicted}}, y_{\text{true}})$ computed independently for each example. The variables from which ROC AUC is estimated in equation 2.17, $\psi(s_i, b_j)$ are not independent. References [51, 52] do the analysis of the problem, that leads to the following expression for the variance of a ROC AUC estimate:

$$\xi_{10} = \mathbb{E}[\psi(s_i,b_j)\psi(s_i,b_k)] - \vartheta^2, j \neq k \tag{2.18}$$

$$\xi_{01} = \mathbb{E}[\psi(s_i,b_j)\psi(s_k,b_j)] - \vartheta^2, i \neq k \tag{2.19}$$

$$\xi_{11} = \mathbb{E}[\psi(s_i,b_j)\psi(s_i,b_j)] - \vartheta^2, \tag{2.20}$$

then

$$\mathrm{var}(\hat{\vartheta}) = \frac{(S-1)\xi_{10} + (B-1)\xi_{01} + \xi_{11}}{SB}. \tag{2.21}$$

When the sample size is relatively small, e. g. less than 1000, variance might be a relatively bad estimate for the confidence interval, as the distribution of the AUC values is not Gaussian. For those cases, many different methods to estimate the confidence interval have been proposed, they are summarised in reference [53]. I have never encountered any of them in high-energy physics or machine learning literature.

2.4 No Free Lunch Theorem

For a given finite sample, there is an infinite amount of probability distributions that might have generated it. For machine learning, this means that in the absence of any assumptions on the generating distribution p, all machine learning algorithms, no matter how ridiculous, perform the same, when averaged over all possible datasets. This conjecture is called the No Free Lunch theorem [54].

Before diving into the proof, let us outline the idea behind it on a simple example. Consider the case of discrete $\mathbf{x}$ and binary classification ($\mathbf{y} \in \{0, 1\}$). Let us have a model and an example that was not used for training this model. The model makes some prediction. And, since we don't have any assumptions on p, with equal probability we live in the world where the target value is either 0 or 1 – thus the model is correct with 50% probability.

This section primary follows my rewrite [55] of the original paper [54].

2.4.1 Formalism

Let us consider the case of the feature and target domains being finite sets $\mathbf{X}$ and $\mathbf{Y}$. This does not rob the theorem of its value, as any practical problem deals with finite numbers, that can be discretized – and indeed are, when processed as floating-point numbers on a computer. A rigorous examination of the continuous case is available in reference [56].

Call $\mathbf{X}$ the input set, $\mathbf{Y}$ the output set. Define a metric (loss function): $L(y_1, y_2) \in \mathbb{R}$, $y_1, y_2 \in \mathbf{Y}$. Introduce the target function $f(x, y), x \in \mathbf{X}, y \in \mathbf{Y}$. $f(x, y)$ is an $\mathbf{X}$-conditioned distribution over $\mathbf{Y}$, i. .e $\forall x \in \mathbf{X}, \sum_y f(x, y) = 1$. Select a training set d of m $\mathbf{X} - \mathbf{Y}$ pairs, according to likelihood $P(d|f)$. Let d_X be its $\mathbf{X}$ component and d_Y be the $\mathbf{Y}$ component. Select a test point $q \in \mathbf{X}, q \notin d_X$ – we are interested in the generalisation power. Such selection is called off-training set (OTS). Take a classifier, train it on d, use it to predict on q. Let y_H be the prediction. Any classifier is completely described by its behaviour, $P(y_H|q, d)$. Also sample the target distribution f at point q, let y_F be the result. Define loss $c = L(y_H, y_F)$.

In this section, we determine several averages of conditional probabilities for loss values over all valid f. f is uniquely specified by an $|\mathbf{X}| \times |\mathbf{Y}|$ matrix of real numbers, so we can write a multidimensional integral $\int A(f)df$ and average $E_f A(f) = \int A(f)df / \int 1df$. All integrals over targets f in this paper are implicitly restricted to the valid $\mathbf{X}$-conditioned distributions over $\mathbf{Y}$. We do not evaluate the integrals explicitly, but for the sake of clarity, it is worth to discuss them.

f is a probability distribution, therefore $\forall x \in \mathbf{X}, \sum_y f(x,y) = 1$. This also means, that f is a mapping from $\mathbf{X}$ to an $|\mathbf{Y}|$-dimensional unit simplex, by definition of a unit simplex. The integration volume F is a Cartesian product of unit simplices, which can be expressed using a combination of Dirac delta functions and Heaviside step functions.

In this section we consider *homogeneous loss*, meaning that

$$\exists \Lambda[\mathbb{R} \to \mathbb{R}] : \forall c \in \mathbb{R}, \forall y_H \in \mathbf{Y} : \sum_{y_F \in \mathbf{Y}} \delta\left[c, L(y_H, y_F)\right] = \Lambda(c), \tag{2.22}$$

where δ is the Kronecker delta function. Intuitively, such L have no a priori preference for one $\mathbf{Y}$ value over another. For example, zero-one loss ($L(a,b) = 1$ if $a \neq b, 0$ otherwise) is homogeneous, and quadratic ($L(a,b) = (a-b)^2$; $a, b, \in \mathbb{R}$) is not. A weaker version of No Free Lunch Theorem still holds for non-homogeneous loss, it is discussed in [57].

Likelihood $P(d|f)$ determines how d was generated from f. It is *vertical* if $P(d|f)$ is independent of the values $f(x, y_F)$ for $x \notin d_X$. We do not require that the $\mathbf{Y}$ components for the test and train sets are generated from the same distribution, but they, of course, may. Verticality is needed to prevent leakage of the test set labels into the train set. For example, the conventional procedure, where d is created by repeatedly and independently choosing its $\mathbf{X}$ component d_X by sampling some distribution $\pi(x)$, and then choosing the associated d_Y value by sampling $f\left(d_x(i), y\right)$, results in a vertical independent and identically distributed (IID) likelihood:

$$P(d|f) = \prod_{i=1}^{m} \pi(d_X(i)) f(d_X(i), d_Y(i)). \tag{2.23}$$

The No Free Lunch Theorem states, that the performance of an algorithm, when averaged by all possible datasets, is equal to constant that is a function of the loss:

$$E_f\left[P\left(c|f,m\right)\right] = \Lambda(c)/r \sum_{d:|d|=m} \left[\frac{\int P\left(d|f\right) df_{x \in d_X}}{\int 1 df_{x \in d_X}}\right] = \Lambda(c)/r. \tag{2.24}$$

The proof is available in appendix A.

2.4.2 Example

Let us illustrate the counter-intuitive idea of No Free Lunch with an example. Take $\mathbf{X} = \{0,1,2,3,4\}$, $\mathbf{Y} = \{0,1\}$, a uniform sampling distribution $\pi(x)$, zero-one loss L. For clarity we will consider only deterministic f, i. e. $f : \mathbf{X} \times \mathbf{Y} \to \{0,1\}$. Set the number of distinct elements in the training set $m' = 4$. Let algorithm A always predict the label most popular in the training set, algorithm B the least popular. In case the counts of labels of different classes in the training set are equal, the algorithms choose randomly.

Let $x_i \in \mathbf{X}$ be the feature vector and $y_i \in \mathbf{Y}$ the label for i-th object. Let c_A be the loss on the test element for the algorithm A, c_B for B. We show that $E_f(c|f,m')$ is the same for A and B.

1. There is only one f for which for all $\mathbf{X}$ values, $\mathbf{Y} = 0$. In this case algorithm A works perfectly, $c_A = 0$, algorithm B always misses, $c_B = 1$.

2. There are $C_5^1 = 5$ fs with only one $y = 1$, the rest being 0. For each such f, the probability that the training set has all zeros is 0.2. For these training sets, the true test label is 1, A predicts 0, B predicts one, $c_A = 1$, $c_B = 0$. For the other 4 training sets, $c_A = 0$, $c_B = 1$. Therefore, the expected value of $Ec_A = 0.2 \times 1 + 0.8 \times 0 = 0.2$ and $Ec_B = 0.2 \times 0 + 0.8 \times 1 = 0.8$

3. There are $C_5^2 = 10$ fs with two $y_i = 1$. There is a 0.4 probability, that the training set has one 1. Therefore, the other 1 is in the test set, and $c_A = 1$, $c_B = 0$. There is a 0.6 probability that the train set has two 1s. In that case both algorithms guess randomly and $Ec_A = Ec_B = 0.5$. So for each f, $Ec_A = 0.4 \times 1 + 0.6 \times 0.5 = 0.7$, $Ec_B = 0.4 \times 0 + 0.6 \times 0.5 = 0.3$. Note that B outperforms A.

4. The cases with three, four and five 1s are equivalent to the already described.

5. Averaging over f, we have

$$E_f c_A = \frac{1 \times 0 + 5 \times 0.2 + 10 \times 0.7}{1 + 5 + 10} = 0.5$$

$$E_f c_B = \frac{1 \times 1 + 5 \times 0.8 + 10 \times 0.3}{1 + 5 + 10} = 0.5$$

2.4.3 Implications

For learning theory, the No Free Lunch (NFL) theorems invalidate any formal performance guarantees, that do not make a restriction on the problem class.

The NFL theorems also prove that performance on a "test" or "validation" set T, commonly used in practice, is not an ultimate tool to compare algorithms. That is if we are interested in the error for examples outside the train and test datasets $x \notin \{d \cup T\}$, in absence of prior assumptions, the error on T is meaningless, no matter how many elements there are in T.

The original paper [54] begins with a quote from David Hume: "Even after the observation of the frequent conjunction of objects, we have no reason to draw any inference concerning any object beyond those of which we have had experience." In some sense, the paper is a reformulation of this thesis in mathematical terms. All our experiences, the training set, belong to the past. This includes any "prior knowledge"—that targets tend to be smooth, the Occams's razor, etc. The NFL theorems state, that even if some knowledge and algorithms allowed you to generalise well in the past (your current training set), there are no formal guarantees about its behaviour in the future. So, if we are to subscribe to strict empiricism and do not make any assumptions about the world, we must accept that the world is unknowable. On the other hand, if we are to claim the ability to predict the future, we must also admit that this ability is based on some assumptions. And there are infinitely many possible assumptions and the choice can not be based on anything empirical as otherwise, it would fall under the prior knowledge.

For machine learning methods to work, some insight into the target problem is required. The power of machine learning is that these assumptions can be very general. There is no need to suppose that the data distribution is, for example, a mixture of Gaussian and exponential distributions. Most methods work under the assumption that it is “smooth”[6] – which seems to be the case for many real-world problems. This is not universal, with the prime example being cryptography, where a smallest change in input is designed to cause a complete change in the output.

2.5 Deep Learning

Deep learning is a particularly successful family of machine learning algorithms that gained prominence in the recent 15 years. It is a combination of several powerful ideas made feasible by the combination of huge amounts of data and computing power to process it. This section serves as an introduction to deep learning and is based on the classical textbooks [59, 60] and the Coursera ”Introduction to Deep Learning” course that I coauthored [61].

2.5.1 Logistic Regression

Consider the problem of binary classification. Let $\mathbf{x}_i \in \mathbb{R}^M$ be feature values for the i-th training example, M the dimensionality of the feature space, $y_i \in \{0, 1\}$ be the class label. Let us construct the simplest model to separate the classes – with a hyperplane.

$$y(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + w_0, \tag{2.25}$$

where $y(\mathbf{x})$ is the model decision function ($y > 0$ means the prediction is class 1, $y < 0$ class 0); slope (or weight vector) $\mathbf{w} \in \mathbb{R}^M$ and intercept (or bias) $w_0 \in \mathbb{R}$ are model parameters. A geometric interpretation of such model is presented in the figure 2.4.

This leaves the question of finding the optimal values of $\mathbf{w}$ and w_0. Theoretically, it is possible to use the formulation we have so far, and, for example, maximise the number of correctly classified examples:

$$\mathbf{w}, w_0 = \underset{\mathbf{w}, w_0}{\arg\max} \left[\sum_{i:y_i=1} H(y(\mathbf{x}_i)) + \sum_{i:y_i=0} H(-y(\mathbf{x}_i)) \right], \tag{2.26}$$

where H is the Heaviside step function.

$$H(n) = \begin{cases} 0 & n < 0 \\ \frac{1}{2} & n = 0 \\ 1 & n > 0. \end{cases} \tag{2.27}$$

[6] Unfortunately, for many modern machine learning algorithms that work extremely well in practice, theoretical analysis is lacking, so we can’t put a precise definition here. However, there are theorems that prove that deep neural networks (covered in section 2.5) can, in principle, approximate any Lebesgue integrable function [58].

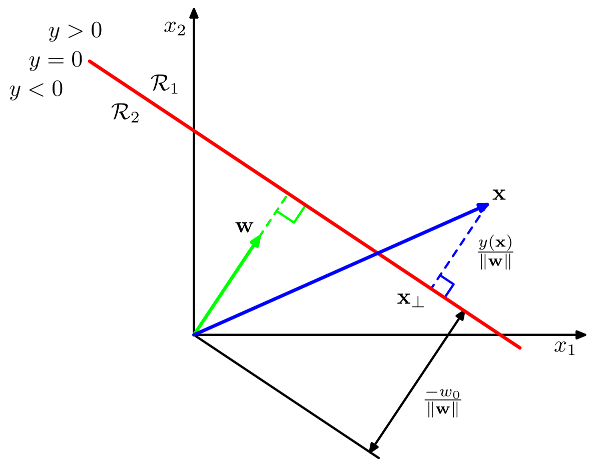


Figure 2.4. Geometry of a linear classifier. The number of dimensions of the features space is 2, in red is the decision surface, in green the vector of weights $\mathbf{w}$, in blue an example which is being classified. Its distance from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$. Reproduced from [59].

The formulation in equation 2.26, however, is inconvenient to optimise, as no closed-form solution is available and the gradient of expression inside arg max is zero almost everywhere.

If a sigmoid function σ is applied to the model output, the output can be interpreted as the posterior probability of the example belong to the class 1:

$$f_i = p(i\text{-th example is class } 1|x_i, \mathbf{w}, w_0) = \sigma(y(x_i)) = \left(1 + e^{-\mathbf{w}^T \cdot \mathbf{x}_i - w_0}\right)^{-1} \in (0, 1). \tag{2.28}$$

This model is called logistic regression, despite it being a classification model. To find the parameters of the model we can use the maximum likelihood method. The posterior probability that i-th example has been correctly classified can be formulated as

$$p(\text{example is correctly labelled}|y_i, f_i) = f_i^{y_i}(1 - f_i)^{1-y_i} \tag{2.29}$$

Since examples are sampled independently, the posterior probability is a product of per-example probabilities:

$$\mathbb{L}(\mathbf{w}, w_0) = p\left(\text{all train examples are correctly labelled}|\mathbf{w}, w_0\right) = \prod_i f_i^{y_i}\left(1 - f_i\right)^{1-y_i}. \tag{2.30}$$

To simplify optimisation, define loss as the negative logarithm of the likelihood, which leads to the cross-entropy error function:

$$L(\mathbf{w}, w_0) = -\ln \mathbb{L}\left(\mathbf{w}, w_0\right) = -\sum_i \left[y_i \ln f_i + (1 - y_i) \ln(1 - f_i)\right]. \tag{2.31}$$

This loss function has continuous gradients and thus can be optimised with gradient descent algorithms that are covered in the section 2.5.3. N. B. Despite appearing to be a simple problem, no general closed-form solution to find the optimal values of the logistic regression parameters has been found [62].

2.5.2 Deep Neural Networks

For many practical problems, linear models are not sufficient. A logical thing to do is to transform the problem to an easier formulation – use feature engineering. An example of such transformation simplifying classification is presented in figure 2.5. By transforming a dataset from Cartesian coordinates to polar, we turn the classification task from impossible to trivial for a linear model. Until the Renaissance of

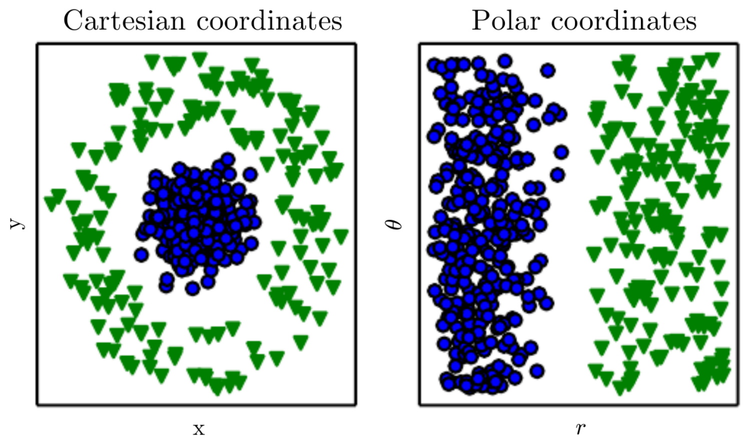


Figure 2.5. Example of a dataset in two representations: Cartesian and polar coordinates. For a linear model, classification is impossible in the Cartesian coordinates but is trivial in polar. Reproduced from [60].

deep learning, feature engineering was the way to go for computer vision tasks. It is highly applicable for all cases where expert knowledge is available that can simplify the problem. In a sense, event reconstruction in high-energy physics can be viewed as feature engineering, as it transforms raw detector readout into a handful of highly useful variables. But manual feature engineering relies on expensive expert work, and the quality of the result does not benefit much from data availability. The idea of deep learning is to automatically learn a "deep" sequence of relatively simple transformations. For some tasks, mostly related to images, the intermediate features learned by modern deep learning algorithms are human-interpretable. An example of interpretable features produced by an image-classification model is presented in figure 2.6.

Let us have two models. A feature extractor, that would take examples from the dataset and apply some function $\mathbf{h}(\mathbf{x}, \boldsymbol{\theta}_2)$ to them, $\boldsymbol{\theta}_2$ being the model parameters. And a classification model, that would make predictions based on the output of the feature extractor. The prediction of our model will be $f(\mathbf{h}(\mathbf{x}, \boldsymbol{\theta}_2), \boldsymbol{\theta}_1)$. This composite model scheme is graphically depicted in figure 2.7. The choice of the models depends on the task in question. For tabular data (where there is no known structure in the features), the most common choice is linear models with elementwise nonlinear transformations between them, as a combination of just linear models will itself be linear. For image data, the best results have been achieved by convolutional transformations, which we will not cover here, as, in this thesis, we will deal only with tabular data.

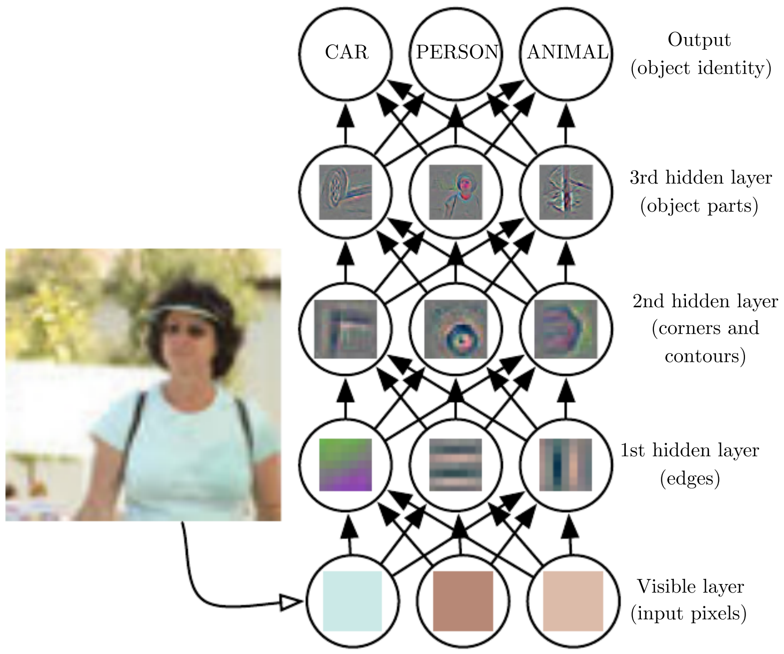


Figure 2.6. Illustration of feature engineering happening inside a deep learning model. The model does a sequence of transformations (hidden layers), developing features of increasing complexity. Using the pixels, the first transformation identifies edges by comparing the brightness of neighbouring pixels. Using the description of the edges, the second transformation searches for corners and extended contours, which are recognisable as collections of edges. Using the description of the image in terms of corners and contours, the third transformation detects entire parts of specific objects, by finding specific collections of contours and corners. Finally, this description of the image in terms of the object parts it contains is used to recognise the objects present in the image. Reproduced from [60].

The simplest "deep neural network" model is presented in figure 2.8. Its prediction is

$$p(\text{example is class } 1|x) = \sigma\left[\mathbf{w}_o^T \cdot \sigma\left(\mathbf{W}_h \cdot \mathbf{x} + \mathbf{b}_h\right) + b_o\right], \tag{2.32}$$

where $\mathbf{W}_h$ and $\mathbf{b}_h$ are the parameters of the first linear model (feature transformer); $\mathbf{w}_o$ and b_o are the parameters of the second model (logistic regression). $\mathbf{W}$ is a $N \times M$ matrix, where M is the dimensionality of $\mathbf{x}$ and N is the dimensionality of the linear transformation output. It is trivial to construct a more expressive model, by stacking more intermediate transformations.

Deep learning has its own terminology.

- It is called deep because it uses a "deep" sequence of transformation.
- A model consisting of a sequence of linear transformations and elementwise nonlinear transformation (e. g. the one in figure 2.8) is called a multilayer perceptron (MLP) or a feedforward fully-connected neural network with a single hidden layer.

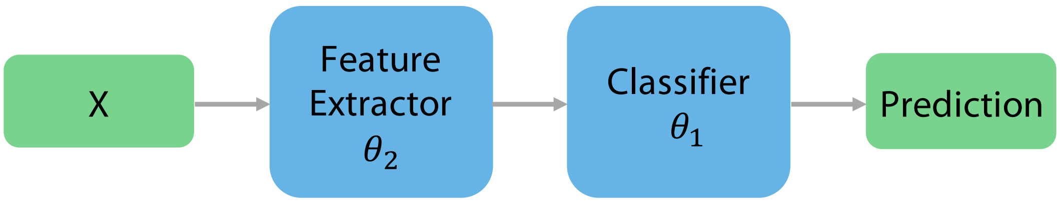


Figure 2.7. A scheme of a two-stage model with a feature extractor. Reproduced from [61].

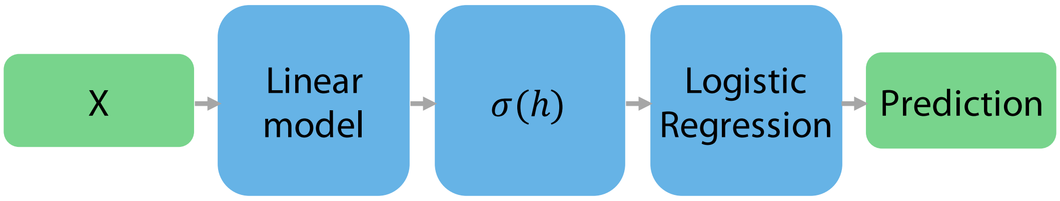


Figure 2.8. The simplest "deep neural network" model. Reproduced from [61].

- The transformations inside a neural network (e. g. the blocks in figure 2.8) are called layers. A linear transformation is a dense layer, as each output element depends on each input element. The dimensionality of the linear transformation output is called the number of neurons[7].
- Transformations (e. g. σ in the figure 2.8) without learnable parameters are called activation functions
- The first layer is called the input layer; the last layer is called the output layer
- All layers between the input and output, are called "hidden"
- And, finally, activation is the value of a layer output, for a particular input, e.g. vector $\sigma(\mathbf{h})$ in figure 2.8.

We used the sigmoid nonlinearity (σ, introduced in 2.28). It is by no means the single possibility. The most common are presented in figure 2.9:

- $\mathrm{ReLU}(x) = \max(0, x)$. It is zero for negative values of the argument and increases linearly for the positive values. Quoting [60], this activation function is the default activation function recommended for use with most feedforward neural networks. Applying this function to the output of a linear transformation yields a nonlinear transformation. However, the function remains very close to linear, in the sense that is a piecewise linear function with two linear pieces. Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimise with gradient-based methods. They also preserve many of the properties that make linear models

[7] Historically, artificial neural networks were, in part, inspired by the biological neural networks. It is possible to build the same formalism using a graph of neurons with connections between them and signals travelling from one to another. The term "number of neurons" comes from this interpretation. I believe that the presented approach with a sequence of transformations is easier to digest for a mathematically-literate reader.

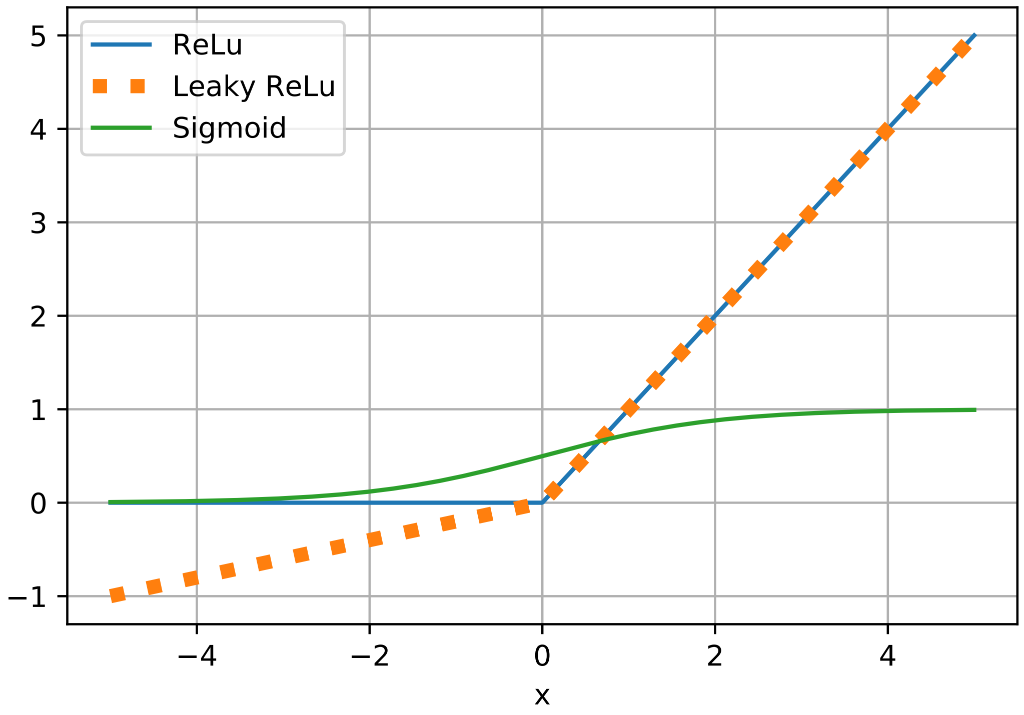


Figure 2.9. Common activation functions. $\text{sigmoid}(x) = (1+e^x)^{-1}$, $\text{ReLU}(x) = \max(0, x)$, Leaky $\text{ReLU}(x) = \max(\alpha x, x), \alpha \in (0, 1)$.

generalise well. One drawback to rectified linear units is that they cannot learn via gradient-based methods on examples for which their activation is zero.

- $\text{LeakyReLU}(x) = \max(0, x) + \alpha \min(0, x)$. The α value is usually around 0.01. It is a piecewise linear function with a small slope for the negative argument values and slope equal to 1 for positive. LeakyReLU was proposed in reference [63]. It alleviates the problems caused by the zero activation of ReLU. While attractive on paper, LeakyReLU and its variants did not manage to replace ReLU due to inconsistent gains [64].

- $\text{sigmoid}(x) = (1 + e^x)^{-1}$ nonlinearity is usually used as the last layer of a network for binary classification. Its value lies in interval $(0, 1)$. The value tends to 0 as the argument approaches negative infinity and to 1 as the argument approaches positive infinity. Its logarithm is always defined and finite, which helps to alleviate the computational instability of the cross-entropy loss. Sigmoid is a poor choice for activations of the hidden layers, as it suffers from the dying gradients problem. The gradient of a sigmoid is always less than one and tends to zero for high values of the argument.

- Other nonlinearities. The research is ongoing, and many other types of nonlinearities using different intuitions have been proposed. Yet no clear winner emerged that can demonstrate a consistent performance advantage. The most convincing studies use automated very computationally expensive trial-and-error search [64].

In case of a multiclass classification problem, a common activation function to apply to the last layer is softmax. Unlike the other discussed activation functions, it is not an elementwise transformation. Softmax ensures that all the outputs conform to basic properties of probability: each of them lies in $[0, 1]$ and the sum of all outputs equals to 1.

$$o_k = \frac{e^{-i_k}}{\sum_m e^{-i_m}}, \tag{2.33}$$

where o_k is the k-th component of the softmax output; i_k is k-th component of the softmax input.

2.5.3 Optimisation

In subsection 2.2.1 we define the problem of training a machine learning model as an optimisation problem. Function optimisation is a well-developed area with many methods proposed for different kinds of target functions [65, 66]. In general, the more is known about the target, the faster and more robust the optimisation is. The solution for the least-squares linear regression is available in the closed form. Logistic regression can be approached by a plethora of methods, such as the classic Newton's and iterative least squares. On the other end of the spectrum, if nothing is known about the function being optimised, than any algorithm is as good as random search [67]. In this subsection, we describe a family of methods, that are particularly suited for deep neural networks optimisation, as they strike a balance between the convergence speed and the burden placed on the class of the optimised function.

The idea of optimisation by gradient descent can be traced to a paper by Cauchy in 1847 [68]. Its idea is simple. Given a function $L(\boldsymbol{\theta})$, its minimum will be at a point where its gradient is zero. To arrive at such a point, iteratively follow the inverse gradient:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \nabla L\left(\boldsymbol{\theta}^{(\tau)}\right), \tag{2.34}$$

where τ is the iteration number and $\eta > 0$ is a parameter (learning rate).

In a machine learning setting, to compute the loss $L(\boldsymbol{\theta})$ and its gradient as a function of the model parameters, we need to calculate the model prediction for every example in the dataset. That can be prohibitively expensive. To address the computational complexity, stochastic gradient descent (SGD) [69] is commonly used. We describe it below.

For each optimisation iteration, SGD uses only a subset of the whole training dataset. Usually, it is organised as follows. The dataset is shuffled and arranged into a queue. For each iteration, a batch of examples is extracted from the queue and used to compute the gradients. A pass over the whole dataset is called an epoch. After an epoch, the dataset is reshuffled and processed again. Let us describe an iteration of the algorithm. Let a batch have m examples. Let $\{\mathbf{x}_1, ..., \mathbf{x}_m\}$ be their features and y_i the corresponding targets. Let $f(x_i, \boldsymbol{\theta})$ be the model prediction for i-th example. Let $\boldsymbol{\theta}$ be the vector of all model parameters. Let $L\left(f(x_i, \boldsymbol{\theta}), y_i\right)$ be the value of the loss function on the i-th example. Then, on each iteration, compute the gradient of the loss over the model parameters using only the examples from the batch; use the gradient to update the estimate of the model parameters:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(f(x_i, \boldsymbol{\theta}^{(\tau)}), y_i\right), \tag{2.35}$$

For complex models, the optimisation problem is non-convex, and it is almost certain that gradient descent would not lead to a global minimum. Whether this is a problem, depends on the difference in the value at that local minimum and the global one. It is possible to explicitly construct examples [70, 71, 72], where the difference will be significant. Whether local minima are a problem in the practical application of neural networks remains an open research question [60]. Many experts, however, suspect, that for sufficiently large neural networks, local minima have a low loss value and it is not essential to find the true global minimum [73, 74, 75].

Stochastic gradient descent follows noisy estimates of the true gradient. This slows down convergence. The problem is illustrated in figure 2.10. Another complexity is the selection of the learning rate. If it is too small, then the convergence will be slow. If it is too large, the optimiser will overshoot and might diverge. This problem of the optimiser taking a longer path is illustrated in figure 2.11.

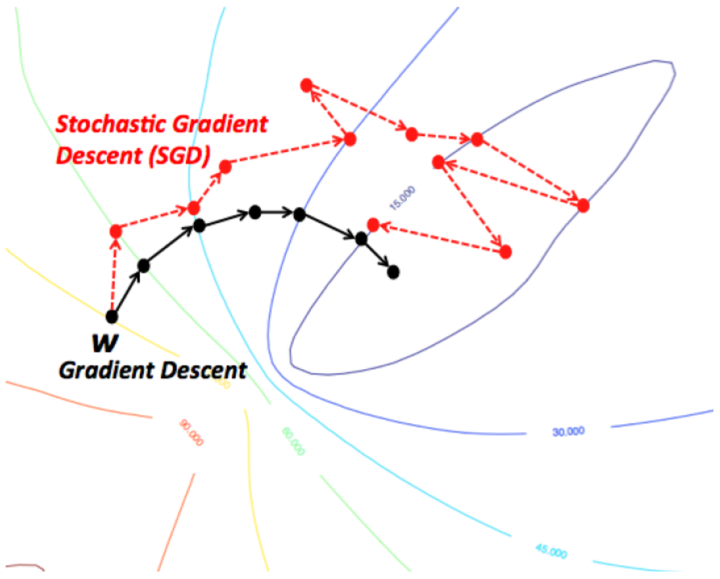


Figure 2.10. Optimising a scalar function of two parameters with gradient descent and stochastic gradient descent. The coloured contour lines represent function values. Reproduced from [76].

There is a physics-inspired solution to both problems: adding momentum [78]. Let us modify the optimisation iteration 2.34 by adding ”velocity” $\mathbf{v}$:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} + \mathbf{v} \tag{2.36}$$

$$\mathbf{v}^{(\tau+1)} = \alpha \mathbf{v}^{(\tau)} - \epsilon \frac{1}{m} \sum_{i} \nabla_{\boldsymbol{\theta}} L\left(f(x_i, \boldsymbol{\theta}^{(\tau)}), y_i\right), \tag{2.37}$$

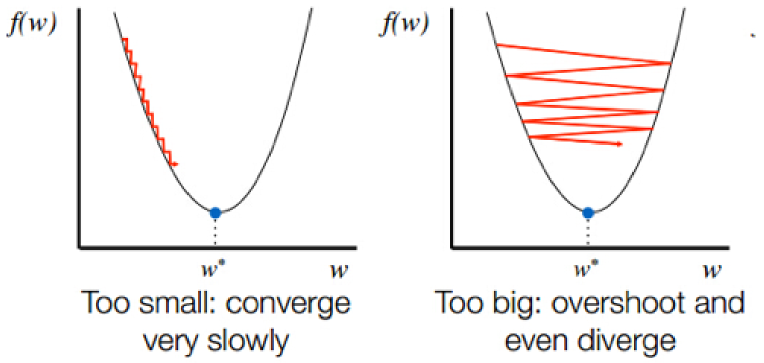

f(w)
w*
w
Too small: converge very slowly
f(w)
w*
w
Too big: overshoot and even diverge

Figure 2.11. Influence of the learning rate on the optimiser behaviour. Reproduced from [77]

where $\alpha \in [0, 1)$ and ϵ are hyperparameters. The larger is $\frac{\alpha}{\epsilon}$, the more "inertia" there is, the more previous gradients affect the current direction.

There is another observation that leads to practically better optimisation. In many real-world problems, the optimal learning rate varies with direction. To account for this, we can keep track of the running average of the gradient magnitude. Then, when applying the parameters update, we normalise the gradient by this running average. This brings us to the algorithm called RMSProp (Root Mean Square Propagation) [79], which is presented in Algorithm 1 below. It has been used for training neural networks in chapter 8.

Algorithm 1 The RMSProp algorithm

```
Require: Global learning rate $\eta$
Require: Initial guess for parameters values $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers
  Initialize the gradients accumulation vector $\mathbf{r} = 0$
  while stopping criterion not met do
      Sample a batch of $m$ training examples $\{\mathbf{x}_1, ..., \mathbf{x}_m\}$ along with the corresponding targets $y_i$
      Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_i \nabla_{\boldsymbol{\theta}} L(f(x_i, \boldsymbol{\theta}), y_i)$
      Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$    ▷ By $\odot$ we mean elementwise multiplication
      Apply parameters update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\delta + \mathbf{r}}} \odot \mathbf{g}$
  end while
```

Momentum can be combined with the adaptive learning rate idea. A popular way to do that is Adam ("adaptive moments") [80] presented in Algorithm 2. Like RMSProp it maintains a running average, this time of gradient and its square. Like the momentum methods, it uses for updates an accumulated values, not the gradient value at the latest point. It has been used for training neural networks in chapter 6.

In this subsection, we discussed the idea of using gradients for optimising model

Algorithm 2 The Adam algorithm

```
Require: Global learning rate $\epsilon$ (suggested default: $10^{-3}$)
Require: Hyperparameters $\rho_1, \rho_2 \in [0, 1)$ – exponential decay rates for momentum
    estimates (suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$, suggested $10^{-8}$, used to stabilise division by small
    numbers
Require: Initial guess for parameters values $\boldsymbol{\theta}$
    Initialise the 1-st and 2-nd moment variables $\mathbf{s} = 0$, $\mathbf{r} = 0$
    Initialise time step $t = 0$
    while stopping criterion not met do
        Sample a batch of $m$ training examples $\{\mathbf{x}_1, ..., \mathbf{x}_m\}$ along with the corre-
    sponding targets $y_i$
        Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \sum_i \nabla_{\boldsymbol{\theta}} L\left(f(x_i, \boldsymbol{\theta}), y_i\right)$
        $t \leftarrow t + 1$
        Update biased first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\mathbf{g}$
        Update biased second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$
        Correct bias in the first moment: $\widehat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}$
        Correct bias in the second moment: $\widehat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$
        Apply parameters update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \frac{\widehat{\mathbf{s}}}{\sqrt{\widehat{\mathbf{r}}}+\delta}$
    end while
```

parameters and introduced some of the most commonly used algorithms. In practical optimisation of neural networks, the presented algorithms perform fairly robustly, but choosing the best algorithms and values of its hyperparameters for a particular model and dataset remains an open question [60]. The developers of the Adam algorithm [80] show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimisation as gradients become sparser.

2.5.4 Training Deep Neural Networks

Training a deep neural network is an optimisation problem. In principle, this optimisation can be approached by various methods, such as genetic algorithms [81] and Alternating Direction Method of Multipliers [82]. In practice, most state-of-the-art architectures use variants of stochastic gradient descent [60], as utilisation of the gradient information speeds up convergence dramatically.

To use the gradient-based optimisation methods to train a model, we need to know the gradient of the loss with respect to all parameters. A deep neural network is just a composite function. Let $\mathbf{f}(\mathbf{x}, \theta) = \mathbf{f_1}(\mathbf{f_2}(...\mathbf{f_k}(\mathbf{x}, \boldsymbol{\theta}^{(k)}), ..., \boldsymbol{\theta}^{(2)}), \boldsymbol{\theta}^{(1)})$ be the prediction of the model, $\boldsymbol{\theta}^{(i)}$ the vector of parameters for the i-th layer, $\boldsymbol{\theta}$ the vector of all model parameters (a concatenation of $\boldsymbol{\theta}^{(i)}$ for all layers), $L(\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y})$ be the loss function of the model with parameters $\boldsymbol{\theta}$ on example with features $\mathbf{x}$ and target value $\mathbf{y}$.

$$L(\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y}) = L\left[\mathbf{f_1}(\mathbf{f_2}(...\mathbf{f_k}(\mathbf{x}, \boldsymbol{\theta}^{(k)}), ..., \boldsymbol{\theta}^{(2)}), \boldsymbol{\theta}^{(1)}), \mathbf{y}\right]. \tag{2.38}$$

We are interested in the gradient $\frac{dL(\mathbf{f}(\mathbf{x},\boldsymbol{\theta}),\mathbf{y})}{d\boldsymbol{\theta}}$. To compute it for each component of

$\boldsymbol{\theta}$, use the chain rule. $\frac{dL}{d\mathbf{f_1}}$ can be computed directly. Then,

$$\frac{dL}{d\mathbf{f}_i} = \frac{dL}{d\mathbf{f}_{i-1}} \cdot \frac{d\mathbf{f}_{i-1}}{d\mathbf{f}_i} \tag{2.39}$$

$$\frac{dL}{d\boldsymbol{\theta}^{(i)}} = \frac{dL}{d\mathbf{f}_i} \cdot \frac{d\mathbf{f}_i}{d\boldsymbol{\theta}^{(i)}}. \tag{2.40}$$

Note that in the equation 2.40 we take a derivative of a vector by a vector. If $\mathbf{a} = \mathbf{f}(\mathbf{b})$ is an m-element vector, and $\mathbf{b}$ is an n-element vector, the derivative $\frac{d\mathbf{a}}{d\mathbf{b}}$ is the $m \times n$ Jacobian matrix [83]:

$$\frac{d\mathbf{a}}{d\mathbf{b}} = \begin{bmatrix} \frac{da_1}{db_1} & \frac{da_1}{db_2} & \frac{da1}{db_3} & \cdots & \frac{da_1}{db_n} \\ \frac{da_2}{db_1} & \frac{da_2}{db_2} & \frac{da_2}{db_3} & \cdots & \frac{da_2}{db_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{da_m}{db_1} & \frac{da_m}{db_2} & \frac{da_m}{db_3} & \cdots & \frac{da_m}{db_n} \end{bmatrix} \tag{2.41}$$

The beauty of the chain rule is that it can be evaluated numerically. The chain rule can be followed in stages, starting from $\mathbf{f_1}$ and $\boldsymbol{\theta}^{(1)}$. When applied to neural networks, this is commonly called error backpropagation [84]. When training a neural network, first, we do a so-called forward pass – starting from $\mathbf{f}_k$ going to $\mathbf{f_1}$ compute the model output. Then we do a so-called backward pass – starting from $\mathbf{f_1}$ (the output layer) we compute the chain of derivatives.

Backpropagation allows us to easily use arbitrary differentiable transformation $\boldsymbol{\phi}(\mathbf{x}, \boldsymbol{\theta})$ for layers. We need only to supply the algothm with three functions: $\boldsymbol{\phi}(\mathbf{x}, \boldsymbol{\theta})$ itself, $\frac{d\boldsymbol{\phi}}{d\mathbf{x}}(\mathbf{x}, \boldsymbol{\theta})$, and $\frac{d\boldsymbol{\phi}}{d\boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\theta})$.

2.5.5 Designing Neural Networks

With all the flexibility comes the natural question of designing the best neural network for a given problem. The more data we have, the more parameters we can afford without encountering catastrophic overfitting. If there is an insight into the problem domain, and there is for images [17], text [85] and voice [86], architectures that take it into account have superior performance. Otherwise, like many things in machine learning, this is an open research question [60, 87, 88]. In practice, model design is done by trial and error and iterating over previous works that proposed models for a particular problem.

State-of-the-art approaches use machine learning to automate this trial and error search [89, 90, 91]. Designing a neural network can be viewed as an optimisation problem: finding the architecture, that would have the best performance on the validation dataset. This main issue with this approach is the computational cost, as training a single variant of neural network architecture can take days of GPU time. Another aspect is the structure of the search space: neural networks are complex objects with a varying number of parameters; they are not differentiable with respect to parameters, such as the number of layers. To make the optimal use of the computation time, in reference [89] they use the so-called reinforcement learning approach. They build a machine learning model that tries to predict the performance of architecture, given its description. It is used to choose the next architecture to try. Architectures are represented as strings, and to work with them a special type of neural network is used, the so-called recurrent neural network (RNN) [92, 93]. In reference [90] they propose to relax the search space to be continuous, instead of searching over a discrete set of architectures. This allows using gradient descent methods. They also propose not training each architecture variant to convergence to speed up the search. In reference [91], the authors use a discrete search space but propose an optimisation. By representing the search space as a graph, and the network architectures as subgraphs, they allow different models to share the weights. Although eschewing training from scratch reduces the accuracy of estimating the performance of architecture, they demonstrate that it remains acceptable, while allowing for a 1000x speed-up compared to the method from reference [89].

2.5.6 Implementing Neural Networks

Deep neural networks benefit a lot from graphics processing units (GPUs) and even more from the specialised hardware [94]. Compared to CPU, a GPU has two defining features [95, 96]:

- Up to thousands of cores [97], allowing for more parallelism;
- A high memory bandwidth. As of the moment of writing, top-of-the-shelf GPU allows up to 900 GB/s [98], CPU up to 141 Gb/s [99].

Matrix multiplication, the most expensive operation in MLP training, can take advantage of the parallelism [100]. The memory bandwidth allows to efficiently load the training data to be processed.

There are several competing modern frameworks with roughly similar functionality that make deep learning methods very easy to use. Prime examples are TensorFlow [101] (used in chapters 8 and 6) and PyTorch [102]. The frameworks allow for automatic computation of derivatives: the user only needs to specify the model as a combination of operations, and the frameworks will compute the gradients needed for optimisation. The frameworks also handle computation execution on GPUs and multiple CPU cores transparently to the user.

2.5.7 Conclusion

The distinguishing feature of deep learning is its ability to profit from large amounts of data. A combination of factors enables this. First, stochastic optimisation allows it to utilise an infinite amount of data at a constant cost per optimisation iteration. Second, the operations used by deep learning benefit tremendously from modern hardware developments.

2.6 Gradient Boosting Decision Tree (GBDT)

In this section, we cover another family of machine learning algorithms. Unlike the deep learning methods, they did not experience “revolutionary” bursts of advancement – but, for many problems with tabular features, they achieve the same or better quality as the deep learning models. The section is based on lectures from the Machine Learning for High-Energy Physics summer school [103, 42] and textbook [104].

2.6.1 Decision Tree

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (usually a constant) in each one [104]. From the perspective of high-energy physics, they can be viewed as automated multilevel cut-based analysis. Decision trees have been used in more-or-less current form as decision-support tools since at least mid-XX century [105, 106]. An illustration of such a tree to asses credit-worthiness is presented in figure 2.12. The idea of such hierarchical representation of information can be traced to 3rd century CE Greek Neoplatonist philosopher and logician Porphyry [107] – see also the discussion in reference [108].

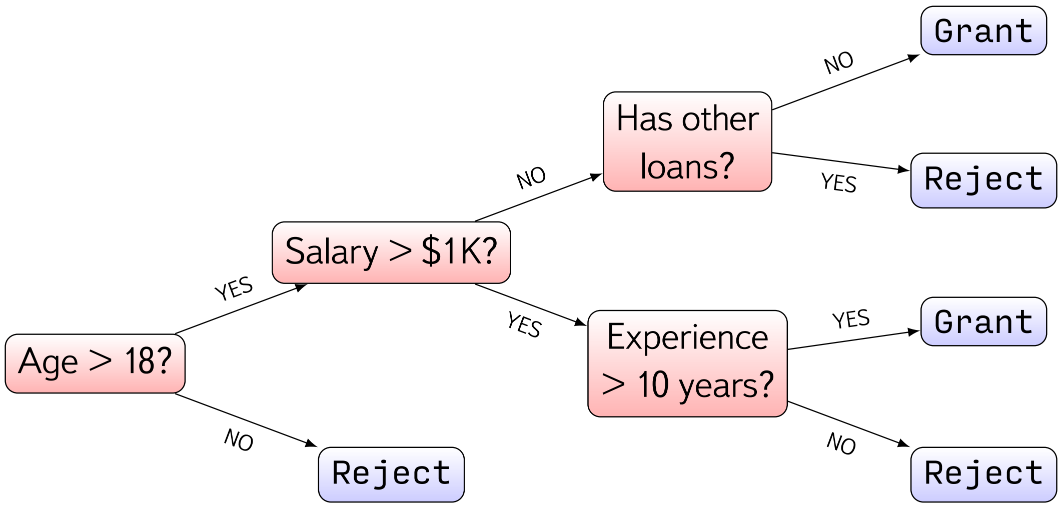


Figure 2.12. An example of decision tree for deciding whether to grant a loan. Reproduced from [103].

In this section, we consider binary trees, where at each point there are only two alternatives. Having splits with more alternatives is, of course, possible, but would complicate training and is not used in practice. A tree V is a tuple consisting of internal nodes u, leafs v, predicates β_u and leaf values $\mathbf{c}_u \in \mathcal{Y}$. $\mathcal{Y}$ depends on the problem. For a regression problem, it equals to the target domain $\mathbb{Y}$. For a classification problem, $\mathcal{Y}$ can be equal $\mathbb{Y}$. In this case, the model will output just the predicted class label, without any score that would indicate the uncertainty of a prediction. It is more useful to make a tree output a vector of scores corresponding to different classes: $\mathcal{Y} = \mathbb{R}^{|\mathbb{Y}|}$ or $\mathcal{Y} = \mathbb{R}^{|\mathbb{Y}|-1}$ (to remove redundancy). An internal node has two children, left and right, which can be either leaves or other internal nodes. u_0 is the root node. $\beta_u : \mathbb{X} \to \{0, 1\}$ is a function from the features domain in into $\{0, 1\}$. If its value for an example with features $\mathbf{x}$ is 0, the example is assumed to fall into the left subtree, otherwise into the right. A leaf is an end node, where a prediction is stored. A graph illustrating the relationship between the internal nodes and leaves is presented in figure 2.13. A formal description of the procedure

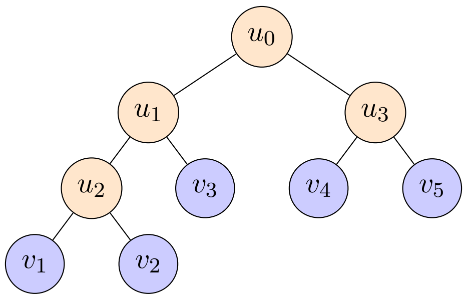


Figure 2.13. Illustration of a generic decision tree. Reproduced from [103].

of making a prediction is presented in Algorithm 3.

Algorithm 3 Decision tree prediction algorithm

```
Require: A binary tree V
  $n \leftarrow u_0$                          ▷ Begin traversal from the root node
  while $n$ is not a leaf do
      $b \leftarrow \beta_n(\mathbf{x})$
      if $b = 0$ then
          $n \leftarrow$ LeftChild($n$)
      else
          $n \leftarrow$ RightChild($n$)
      end if
  end while
  return $\mathbf{c}_n$
```

In principle, predicates β_u can be any function. In practice, in state-of-the-art machine learning algorithms, the predicates compare the value of some feature to some threshold.

$$\beta_u(\mathbf{x}) = [x_{j_u} < t_u], \tag{2.42}$$

where $[x_j < t_u]$ is the Iverson bracket, j_u is the index of the feature that is used for the split, and t_u is the threshold.

Constructing an optimal binary decision tree is an NP-complete [109] problem, thus finding the exact solution requires superpolynomial[8] time. A practical way, used by the state-of-the-art machine learning algorithms [110, 44] to build a tree is with a greedy algorithm. When training decision a decision tree in a greedy fashion, an algorithm selects splits step-by-step, on each iteration choosing the split that offers the most obvious immediate performance improvement. It doesn't think ahead about possible future splits. This approach requires three things: a loss function to quantify the performance of s split, a stopping criterion and the leaf value assignment procedure. The loss function $Q(X, j, t) \to \mathbb{R}$ estimates how good is a split of the dataset X by feature number j with threshold t. Leaf assignment usually minimises some loss. A stopping criterion looks at the state of the algorithm

[8]Let n be the input size, $t(n)$ be the number of operations needed to find the optimal tree, then $\forall c \in \mathbb{R} \lim_{n\to\infty} \left|\frac{t(n)}{n^c}\right| = \infty$.

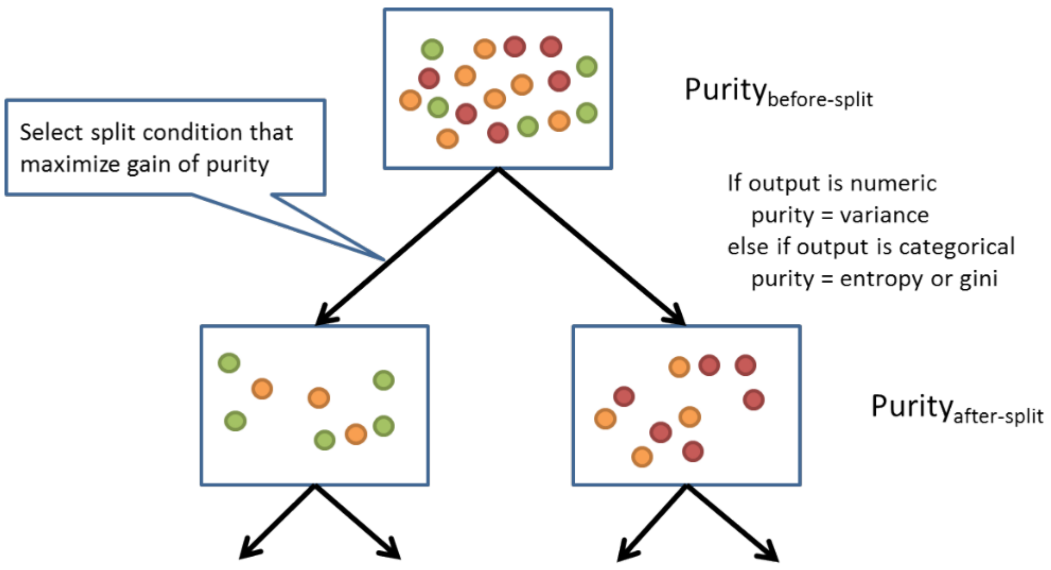

Select split condition that maximize gain of purity
Purity$_{\text{before-split}}$
If output is numeric
purity = variance
else if output is categorical
purity = entropy or gini
Purity$_{\text{after-split}}$

Figure 2.14. Illustration of the purity maximisation idea. Reproduced form [111].

and decides whether the current node should be a leaf, or additional splits must be made. We will cover the leaf value assignment and stopping criterion shortly. The greedy training algorithm is presented in Algorithm 4.

Algorithm 4 Greedy tree learning for binary classification

```
Require: Training set $X = \{(\mathbf{x}_i, \mathbf{y}_i)\}$
  procedure MakeSplit($X^\ell$: dataset to be split)
      $j, t = \arg\min_{(j,t)} Q(X^\ell, j, t)$
      Create tree node $u$ corresponding to predicate $[x_j < t]$
      if Stopping criterion is satisfied for $u$ then
          Declare $u$ a leaf, set some value $\mathbf{c}_u \in \mathcal{Y}$ as prediction
      else
          $R_1(j,t) = \{\mathbf{x} \in X^\ell | x_j < t\}$
          $R_2(j,t) = \{\mathbf{x} \in X^\ell | x_j > t\}$
          MakeSplit($R_1$)
          MakeSplit($R_2$)
      end if
  end procedure
  MakeSplit($X$)
```

There are multiple viable choices for loss function and the stopping criterion (as for many things in machine learning). The idea of the loss function is to promote splits that result in leaves with best purity – least variance of target value in the set of examples that fall into the leaf after the split. The idea of a purity maximising split for a 3-class classification problem is illustrated in figure 2.14. Let us define the impurity criterion for the case, where $\mathcal{Y} = \mathbb{Y}$:

$$H(R) = \min_{\mathbf{c}\in\mathcal{Y}} \frac{1}{|R|} \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in R} L(\mathbf{c}, \mathbf{y}_i). \tag{2.43}$$

It takes the loss function $L(y_{\text{predicted}}, y_{\text{true}})$ (as introduced in 2.2) and finds the leaf value $\mathbf{c}$ that would minimise the average loss for this set. The value of c in a leaf is selected to minimise the impurity. If the problem is a regression, the usual loss to use is the mean squared error and $\mathbf{c} = \mathbb{E}\left(\mathbf{y}_i|\mathbf{y}_i \in R\right)$. For classification, the logical choice for $\mathbf{c}$ is to be a vector of class probabilities, estimated as their frequencies in the leaf:

$$c_k = \frac{1}{|R|} \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in R} [\mathbf{y}_i = \gamma_k], \tag{2.44}$$

where $[\mathbf{y}_i = \gamma_k]$ is the Iverson bracket that equals to 1, when i-th example belongs to the k-th class and 0 otherwise. There are two primary choices for the loss function to use: Gini index and cross-entropy.

$$L_{\text{cross-entropy}} = -\sum_k c_k \ln c_k. \tag{2.45}$$

Cross-entropy loss here is directly derived from the logloss introduced in subsection 2.3.3: c_k is both the fraction of examples belonging to the k-th class, and the predicted probability of an example to belong to the k-th class.

$$L_{\text{Gini}} = \sum_k c_k(1 - c_k). \tag{2.46}$$

Gini impurity[9] [112] is defined as follows. Consider a leaf that has a set of examples. Randomly choose an example and randomly assign it a label according to the distribution of labels in the leaf. Gini impurity is the probability that the example is mislabeled. This way, if all the labels in the leaf are the same, the impurity would equal to zero. If each label is unique, the impurity would equal to $1 - 1/n$, where n is the number of examples in the leaf. Having a loss function, we can use it to define the loss function for a split:

$$Q(X^\ell, j, t) = H(X^\ell) - \frac{|R_1|}{|X^\ell|}H(R_1) - \frac{|R_2|}{|X^\ell|}H(R_2), \tag{2.47}$$

where X^ℓ is the dataset that is being split, $R_1(j,t) = \{\mathbf{x} \in X^\ell | \mathbf{x}_j < t\}$ and $R_2(j,t) = \{\mathbf{x} \in X^\ell | \mathbf{x}_j > t\}$ is the proposed split.

The stopping criterion has a large impact on model quality. Stop too early, and the model is underfitted, stop too late, and the model is overfitted. Naturally, there are quite a few choices:

- Maximum tree depth. Results in a computationally convenient uniform tree, but disregards the variation among the leaves.
- Minimum number of objects in leaf. Much like for bin of a histogram, the smaller is the leaf, the noisier would an estimate that is constructed from it.

[9]From the point of view of information theory, it corresponds to Tsallis Entropy with deformation coefficient $q = 2$, which is a generalisation the standard Boltzmann–Gibbs entropy.

- Maximum number of leaves in the tree.
- Stop when improvement gains drop below a certain threshold. The performance on the test dataset is almost always worse than performance on the training dataset. If an improvement from the proposed split is small on the training dataset, it is even smaller on the testing dataset, or might even be negative in case of overfitting.

In practice, trees usually have, in addition to the basic algorithm we outlined, various heuristics for preventing overfitting. A modern implementation can be found, for example, in the scikit-learn library [113]. Decision trees are sometimes used raw, mostly for the reason that they are human-interpretable. However, their performance is almost always inferior to the ensemble methods, the highest-performing of which we will cover in the next subsection.

2.6.2 Boosting

The idea of boosting is to train many weak models sequentially so that each next model corrects the error of the previously constructed. In principle, any models can be used, in practice, decision trees usually are. There are, of course, works that utilise different weak learners [114, 115, 116], but no method was able to demonstrate consistently better performance compared to boosting decision trees.

Boosting relies on building a sequence of weak learners. Denote them formally as a some set of predictors $\mathbb{H} = \{h_j : \mathbb{X} \to \mathbb{Y}\}$. Consider a regression problem with a single-dimensional target domain, mean squared error loss, and a training dataset $(\mathbf{x}_i, y_i)$ consisting of N examples. The naive boosting algorithm for this case is presented in Algorithm 5. On each iteration we compute the differences between the current model predictions and the true label values, fit a learner to correct these differences, and add the learner to the model.

Algorithm 5 Naive boosting for regression

```
Require: Training dataset $(\mathbf{x}_i, y_i)$ consisting of $N$ examples
Require: Mean squared error loss $L(y_{\text{predicted}}, y_{\text{true}}) = (y_{\text{true}} - y_{\text{predicted}})^2$
Require: A family of weak learners $\mathbb{H}$
  Start with a trivial learner $h_0(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} y_i$
  Start with model equal to the trivial learner $a_o(\mathbf{x}) = h_0(\mathbf{x})$
  $t \leftarrow 0$
  while stopping criterion is not met do
    Compute residuals $s_i \leftarrow y_i - a_t(\mathbf{x}_i)$
    $t \leftarrow t + 1$
    Train the next learner to predict the residuals: $h_t = \arg\min_{\mathbb{H}} \sum_{i=1}^{N}(s_i - h(\mathbf{x}_i))^2$
    $a_t \leftarrow a_{t-1}(\mathbf{x}) + h_t(\mathbf{x})$
  end while
  return trained model $a(\mathbf{x}) = \sum_{n=0}^{t} h_n(\mathbf{x})$
```

Let us then generalise this idea for problems other than regression with the mean squared error. For clarity consider the case where the model prediction is a scalar.

Let a_t be the state of the model on iteration t. Let $h \in \mathbb{H}$ be the weak learner we and add to the model on the iteration. The objective of the t-th boosting iteration is to minimise the loss:

$$\arg\min_{\gamma \in \mathbb{R}, h \in \mathbb{H}} \sum_{i=1}^{t} L(a_{t-1}(\mathbf{x}_i) + \gamma h(\mathbf{x}_i), y_i). \tag{2.48}$$

We can use gradient descent introduced in section 2.5.3 to do that. Instead of residuals, compute

$$s_i = -\frac{\delta L(y_{\text{predicted}}, y_i)}{\delta y_{\text{predicted}}}\bigg|_{y_{\text{predicted}} = a_{t-1}(\mathbf{x}_i)}, \tag{2.49}$$

train a weak learner to estimate s_i. This is called gradient boosting, first proposed in [117]. The algorithm of gradient boosting is presented in Algorithm 6. Modern implementations of gradient boosting [110, 118, 44] use many additional improvements, some of which are listed below.

Algorithm 6 Gradient Boosting

```
Require: Train dataset $(\mathbf{x}_i, y_i)$ consisting of $N$ examples
Require: Loss function $L(y_{\text{predicted}}, y_{\text{true}})$ and its gradient $\frac{\delta L}{\delta y_{\text{predicted}}}$
Require: A family of weak learners $\mathbb{H}$
  Start with a trivial learner $h_0(x) = 0$
  Start with model equal to the trivial learner $a_o(\mathbf{x}) = h_0(\mathbf{x})$
  $t \leftarrow 0$
  while stopping criterion is not met do
      Compute the loss gradient $s_i \leftarrow -\frac{\delta L(y_i, y_{\text{predicted}})}{\delta y_{\text{predicted}}}\Big|_{y_{\text{predicted}} = a_t(\mathbf{x}_i)}$
      $t \leftarrow t + 1$
      Train the next learner to predict residuals: $h_t = \arg\min_{h \in \mathbb{H}} \sum_{i=1}^{N} (s_i - h(\mathbf{x}_i))^2$
      $\gamma_t \leftarrow \arg\min_{\gamma_t \in \mathbb{R}} \sum_{i=1}^{N} L(a_{t-1}(\mathbf{x}_i) + \gamma_t h_t(\mathbf{x}_i), y_i)$    ▷ Gradient descent
      $a_t \leftarrow a_{t-1}(\mathbf{x}) + \gamma_t h_t(\mathbf{x})$
  end while
  return trained model $a(\mathbf{x}) = \sum_{n=0}^{t} \gamma_n h_n(\mathbf{x})$
```

Shrinkage. Gradient boosting is in its core a gradient descent algorithm and benefits from using a learning rate to limit the steps it takes. Shrinkage is a technique, when every next learner h_t is added to the ensemble with coefficient $\eta \in (0, 1]$:

$$a_t(\mathbf{x}_i) \leftarrow a_{t-1}(\mathbf{x}) + \eta \gamma_t h_t(\mathbf{x}) \tag{2.50}$$

Use second-order derivative in gradient descent [110]:

$$s_i = -\left.\frac{\delta L(y_{\text{predicted}}, y_i)}{\delta y_{\text{predicted}}}\right|_{y_{\text{predicted}}=a_t(\mathbf{x}_i)} \tag{2.51}$$

$$t_i = -\left.\frac{\delta^2 L(y_{\text{predicted}}, y_i)}{\delta y^2_{\text{predicted}}}\right|_{y_{\text{predicted}}=a_t(\mathbf{x}_i)} \tag{2.52}$$

$$h \leftarrow \underset{h\in\mathbb{H}}{\arg\min} \sum_{i=0}^{N} \left(-s_i h(\mathbf{x}_i) + \frac{1}{2} t_i h^2(\mathbf{x}_i)\right) \tag{2.53}$$

The second order approximation better describes the loss function behaviour and allows for faster convergence[10].

Oblivious trees as weak learners [119, 120, 44]. A regular decision tree selects each split to minimise the loss in the subtree. An oblivious decision tree adds a constraint: the split criterion is the same on each level. The difference is illustrated in figure 2.15. An oblivious decision tree is less expressive, but is a lot faster to evaluate. In order to find the leaf that corresponds to a given example in an ordinary tree, it is is necessary to traverse it node-by-node doing multiple conditional jumps. If an oblivious tree is stored as a table, and it is possible to directly compute the pointer to the correct leaf [121].

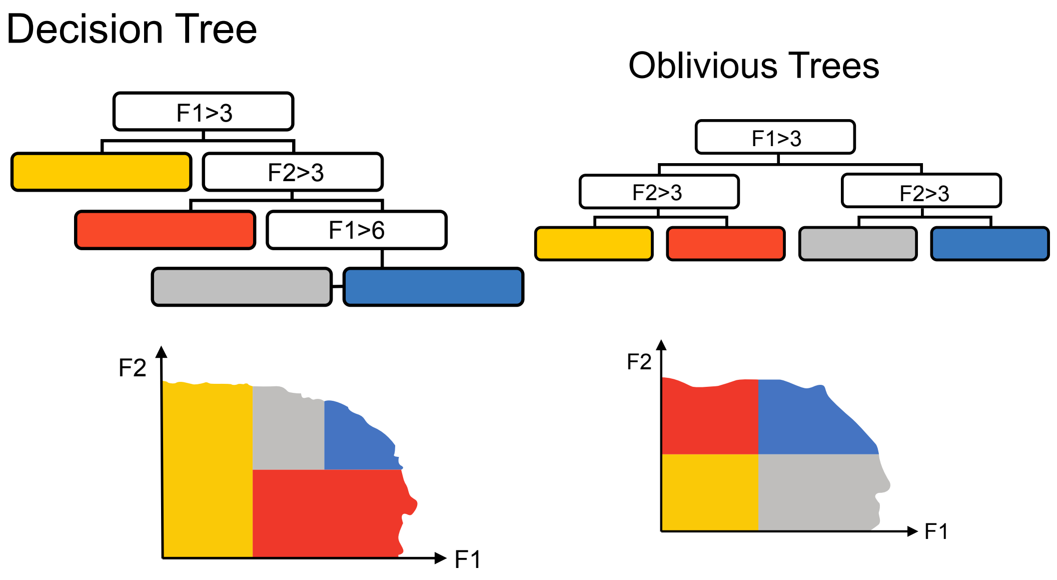


Figure 2.15. Oblivious vs classic decision trees. Reproduced from [122].

[10]If second-order approximation is better then why isn't it used in deep learning? In gradient boosting the second derivative computation means just calling a single function, which almost always has negligible cost compared to the rest of the boosting algorithm. In deep learning we would have to then backpropagate the Hessian through the layers, and this could easily become the most expensive operation.

2.6.3 Implementing GBDT

There are several competing state-of-the-art open-source packages implementing GBDT: XGBoost [110], LightGBM [118], CatBoost [44] (I worked on it a little bit during my tenure at Yandex [123]). In each, there are clever engineering, smart algorithmic enhancements, and ingenious heuristic tricks. For most common tasks, the difference in their performances is small from a practical point of view. All three libraries support multithreading and training on GPU. Oblivious trees in CatBoost make it several times faster to evaluate, compared to the other variants [124]. CatBoost is used for global particle identification in chapter 7, muon identification in chapter 5, and experiments in chapter 6.

2.6.4 Conclusion

Gradient Boosting Decision Tree (GBDT) is a powerful machine learning method, widely used in the industry. It has quality competitive with deep learning on tabular data, where there is no known structure in the features. In general[11], there is a trend that the smaller the dataset, the worse will a fully-connected neural network perform compared to gradient boosting.

Depending on hardware and implementation, GBDT tends to have faster evaluation times than deep neural networks. To predict with a neural network, we need to use all of its parameters to compute the sequence of transformations in its layers. For a tree-based model, we use only a small part of the model: the thresholds needed to find the correct leaf and the value in it.

2.7 Generative models

Generative models, introduced in section 2.2, solve a much harder task than the regression and classification models we covered in the previous sections. This is not only the case for artificial intelligence but also natural intelligence: recognising a cat on a painting is much easier than creating the painting.

Early generative models constructed an unnormalised approximation of the probability density function and then used Markov chain Monte Carlo to sample from it. One of the most successful examples of this approach is the restricted Boltzman machines [127]. They face two significant difficulties. First, the likelihood function is intractable, forcing the usage of computationally expensive inexact approximation of the likelihood for training. Second, the Markov chain presents the problem of autocorrelation. The nearby samples are not independent of each other. To acquire an approximation of a sample from the target distribution, only one in n examples can be used.

[11] The opinion expressed in the text is primarily based on my conversations with machine learning practitioners. Unfortunately, peer-reviewed literature comparing deep learning and GBDT performance in a systematic way is scarce, most likely, due to the high computational and labour cost of making a fair comparison with parameter tuning for each algorithm. Among the recent papers, [125] mostly experimented with small datasets with fewer than 10^4 examples and lacks the crucial analysis of performance as a function of dataset size, [126] limited consideration of neural networks to ones with a single hidden layer.

Those problems are resolved by the Generative Adversarial Networks (GAN), the most prominent family of state-of-the-art generative models, that are introduced in the next section.

2.7.1 Generative Adversarial Network (GAN)

The idea of Generative Adversarial Network (GAN) is to simultaneously train two neural networks – one to generate examples and another to measure the performance of the first. The architecture was first explicitly proposed in reference [128], although the broader origins of the idea are disputed [129, 130]. The GAN idea is broadly illustrated in figure 2.16. The generator produces a batch of examples, the discriminator (detective on the picture) evaluates whether they are real or fake (generated). The discriminator is trained as a classifier by using labelled real and generated (fake) examples. The generator is trained to fool the discriminator. The discriminator and generator are trained in alternating steps. By playing against one another, they improve, hopefully to the level so that the objects forged by the generator will be indistinguishable from the real ones.

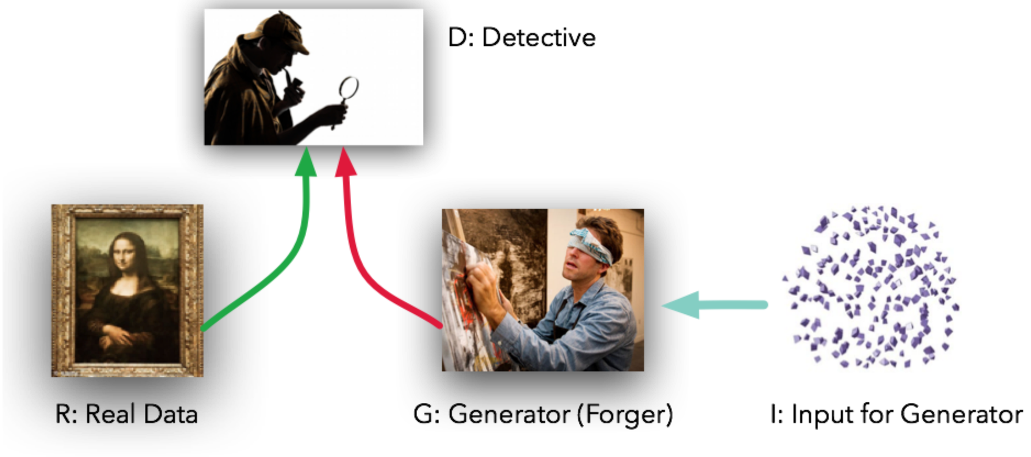
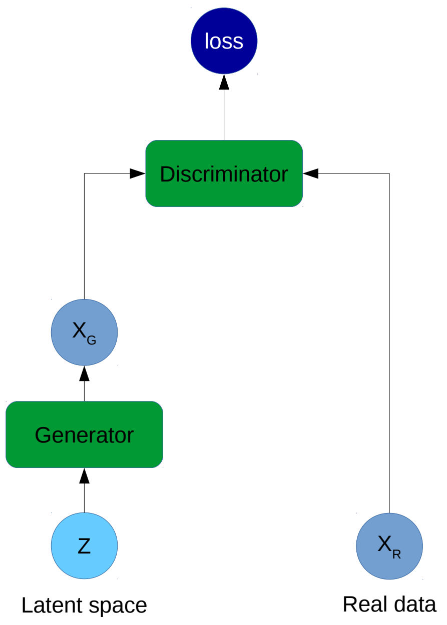
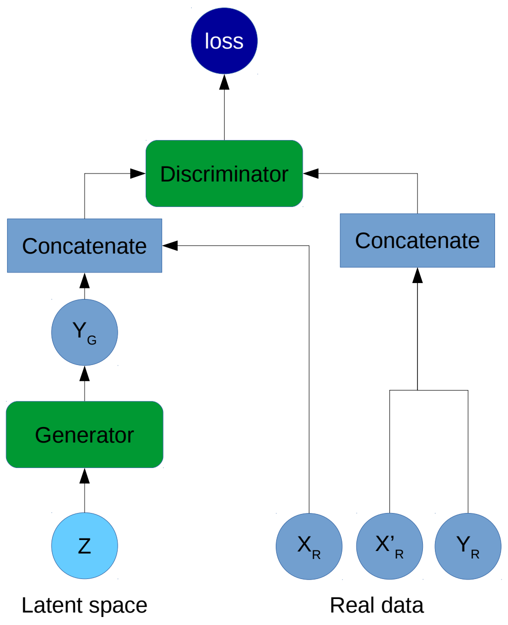


Figure 2.16. A metaphor illustrating training of a GAN. Reproduced from [131].

The generator G is a feed-forward neural network, that maps from some latent space $\mathbb{Z}$ to the example space $\mathbb{X}$. During both training and inference, the network is fed with random values $z \in \mathbb{Z}$, that come from a pre-defined distribution, usually a multivariate normal distribution. The discriminator D is a feed-forward neural network, that maps from the example space $\mathbb{X}$ to $[0, 1]$. Training a GAN proceeds in iterations. On each iteration, first, the discriminator is trained for several steps, usually $1 - 20$. It is fed batches of equal size of generated and real examples, that are labelled as such. The gradient of the loss with respect to the discriminator parameters is used to make the gradient descent steps. Second, the generator is trained. A batch of examples is sampled from it, and the gradient of the negated discriminator loss is backpropagated. A gradient descent step is made to optimise the generator weights. Mathematically, this procedure searches for the minimum with the respect to the generator parameters of a maximum with the respect to the



(a) Unconditional GAN



(b) Conditional GAN. Note the convention collision, here the generated variable is Y, and X is the input to the model.

Figure 2.17. Unconditional and conditional GANs

discriminator of a score function. It is also called a minimax optimisation problem:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\ln D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\ln(1 - D(G(\mathbf{z})))\right]. \qquad (2.54)$$

Reference [128] proves, that if the generator and discriminator have enough capacity, and on each step the discriminator reaches its optimum, then the generated distribution converges to the real one. While elegant, this theoretical proof has limited practical utility, as the preconditions are almost never satisfied for real-world problems. In the case of conditional generation, the problem becomes [132]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\ln D(\mathbf{y}|\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\ln(1 - D(G(\mathbf{z}|\mathbf{x})))\right]. \qquad (2.55)$$

Note, that we use a naming convention different from [132]: to stay consistent with the previously covered classification and regression we call the conditions $\mathbf{x}$ and the generated variables $\mathbf{y}$. Conditional and unconditional GANs architectures are illustrated in figure 2.17.

The first GAN faced practical challenges. In the case the discriminator is significantly more powerful and better-trained, than the generator, it will be able to separate the training dataset from the generated data, and its predictions will be around 0 and 1, leading to near-zero gradients. If the system happens to reach this state, it can not train further, as the discriminator is already as perfect as it can

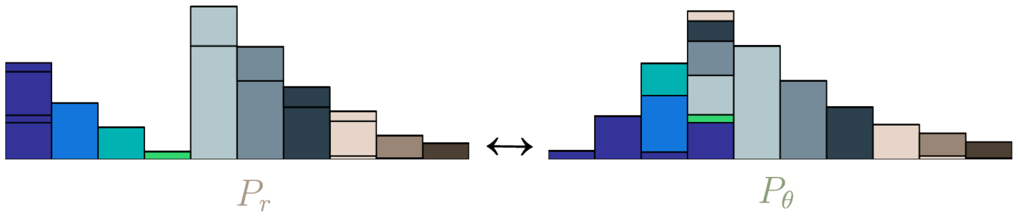


Figure 2.18. An illustration of the optimal transport for a single-dimensional case. The distribution on the left is the source distribution, distribution to the right is the target distribution. The source distribution is broken into chunks which are rearranged to match the target distribution. Reproduced from [136].

get for the given generator, and the generator is stuck with zero gradients. In practice, GANs require delicate tuning of the generator, discriminator and the training procedure for it to converge [133].

2.7.2 Wasserstein GAN

Since the publication of reference [128] in 2014, more than 500 papers proposing improvements have been published. Many of them deal with specific imaging and textual applications. One of the most significant fundamental developments was the introduction of the Wasserstein GAN [134, 135]. It replaced the optimisation problem 2.54 with a one better suited for stochastic numeric optimisation.

Training a GAN consists of measuring the discrepancy between the data and generated distributions – and minimising it. For an optimal discriminator, the formula 2.54 corresponds, up to a constant, to the so-called Jensen-Shannon divergence:

$$D_{\text{JS}}(P\|Q) = \frac{1}{2}\int p(\mathbf{x}) \ln\left(\frac{2p(\mathbf{x})}{p(\mathbf{x})+q(\mathbf{x})}\right) d\mathbf{x} + \frac{1}{2}\int q(\mathbf{x}) \ln\left(\frac{2q(\mathbf{x})}{p(\mathbf{x})+q(\mathbf{x})}\right) d\mathbf{x}, \tag{2.56}$$

where P and Q are probability distributions over $\mathbb{X}$, $p(\mathbf{x})$ and $q(\mathbf{x})$ are the corresponding density functions. In a case of density functions with disjoint supports ($\forall \mathbf{x} \in \mathbb{X} : p(\mathbf{x}) \cdot q(\mathbf{x}) = 0$), the expressions inside the logarithms are constant, and the distance is equal to $\ln 2$, a constant. The gradient is zero. In reference [134] they propose training a GAN by optimising a different distance measure — the Earth-Mover (EM) distance or Wasserstein-1:

$$D_{\text{W}}(P\|Q) = \inf_{\gamma\in\prod(P,Q)} \mathbb{E}_{(\mathbf{x},\mathbf{x}')\sim\gamma}\left[||\mathbf{x}-\mathbf{x}'||\right], \tag{2.57}$$

where $\prod(P,Q)$ denotes the all joint distributions $\gamma(\mathbf{x},\mathbf{x}')$, whose marginals are respectively P and Q. Intuitively, $\gamma(\mathbf{x},\mathbf{x}')$ indicates how much "mass" must be moved from $\mathbf{x}$ to $\mathbf{x}'$ to make the distributions match. The Earth-Mover (EM) distance equals to the minimal amount of "work" that is needed to make this happen, where "work" equals to "mass" times distance modulo. This optimal transport for a single-dimensional case is illustrated in figure 2.18. Directly computing the optimal value of $\gamma \in \prod(P,Q)$ is a hard problem. The Kantorovich-Rubenshtein duality [137] can be used to transform it into an easier one:

$$D_{\mathrm{W}}(P\|Q) = \sup_{||f||_L \le 1} \left[\mathbb{E}_{\mathbf{x}\sim P}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim Q}[f(\mathbf{x})]\right], \tag{2.58}$$

where $||f||_L \le 1$ denotes that function $f : \mathbb{X} \to \mathbb{R}$ is 1-Lipschitz. A function $f : \mathbb{R}^n \to \mathbb{R}$ is K-Lipschitz if

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n : |f(\mathbf{x}_1) - f(\mathbf{x}_2)| \le K|\mathbf{x}_1 - \mathbf{x}_2|. \tag{2.59}$$

This is illustrated in figure 2.19. Another important property of Lipschitz continuity

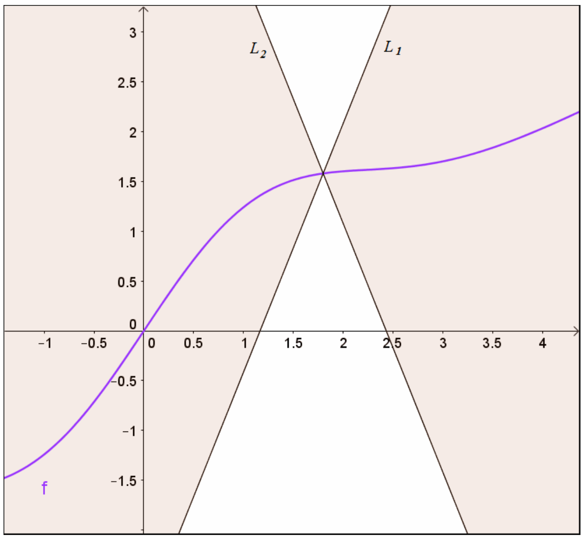


Figure 2.19. Illustration of the Lipschitz condition. The cone's origin can be placed anywhere on the function and the function graph is outside the code. Reproduced from [138].

is that if a function is everywhere differentiable and its gradient is bound by some constant than it is Lipschitz-continuous.

Wasserstein GAN uses a neural network to approximate the function f in formula 2.58. To do this, we need to enforce the Lipschitz property on the network. Reference [135] proposes bounding the gradient norm of f. To circumvent the tractability issue, they propose a soft version of the constraint, with a penalty on the gradient norm added to the loss. This results in the following expression for the discriminator loss:

$$L = \underbrace{\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[D(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[D(G(\mathbf{z}))\right]}_{\text{EMD estimate}} + \underbrace{\lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}})}\left[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2\right]}_{\text{Gradient penalty}}, \tag{2.60}$$

where λ is a hyperparameter responsible for the penalty strength, typically around 10.

Enforcing the gradient penalty everywhere is intractable. The authors suggest it is enough to define $p(\hat{\mathbf{x}})$ as uniform sampling along the lines connecting points from the data and generated distributions, as the optimal critic contains straight lines with gradient norm 1 connecting coupled points from the data and generated distributions.

Altogether, training of a Wassershtein GAN with gradient penalty is summarised in the Algorithm 7. The crucial advantage of the Wassershtein GAN over the

Algorithm 7 Wasserstein GAN with gradient penalty (WGAN-GP). Suggested in [135] default hyperparameters values: $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 10^{-4}, \beta_1 = 0, \beta_2 = 0.9$

```
Require: The gradient penalty coefficient $\lambda$
Require: The number of critic iterations per generator iteration, $n_{\text{critic}}$
Require: Batch size $m$
Require: Adam hyperparameters $\alpha, \beta_1, \beta_2$
Require: Initial discriminator parameters $w_0$
Require: Initial generator parameters $\theta_0$
  while $\theta$ has not converged do
      for $t = 1, ..., n_{\text{critic}}$ do
          for $i = 1, ..., m$ do
              Sample real data $\mathbf{x} \sim p(\mathbf{x})$
              Sample the latent variable $\mathbf{z} \sim p(\mathbf{x})$
              Sample a random number $\epsilon \sim U[0, 1]$
              $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z})$
              $\hat{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon)\tilde{\mathbf{x}}$
              $L_D^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda(\|\nabla_{\widehat{\mathbf{x}}} D(\widehat{\mathbf{x}})\|_2 - 1)^2$
          end for
          $w \leftarrow \text{Adam}\left(\nabla_\theta \frac{1}{m} \sum_{i=1}^m L_D^{(i)}, \theta, \alpha, \beta_1, \beta_2\right)$
      end for
      Sample a batch of latent variables $\left\{\mathbf{z}^{(i)}\right\}_{i=1}^m \sim p(\mathbf{z})$
      $L_G \leftarrow -D_w(G_\theta(\mathbf{z}))$
      $\theta \leftarrow \text{Adam}\left(\nabla_\theta \frac{1}{m} \sum_{i=1}^m L_G^{(i)}, \theta, \alpha, \beta_1, \beta_2\right)$
  end while
```

original design is that its gradients do not vanish, even when the discriminator is much better trained than the generator. This is illustrated in the figure 2.20. In practice this means, that training is much more stable, as there is no need to delicately balance the generator and discriminator strength.

2.7.3 Cramer (Energy) GAN

Most of the GAN advances discussed in literature concern images and use insights specific to them to improve the performance. If we are to apply GANs to physics, as we do in chapter 8, we are interested in more universal methods, that would improve the quality of the distributions reconstruction. One of them is suggested in reference [139]. The core insight is that an estimate of the gradient of the Wasserstein distance on a finite sample is biased, and therefore using those estimates as inputs to a gradient descent algorithm will not result in finding a true minimum. They propose using the energy distance [140] as a replacement. It retains the desirable property of Wasserstein distance – the non-vanishing gradients that lead to stable training, while

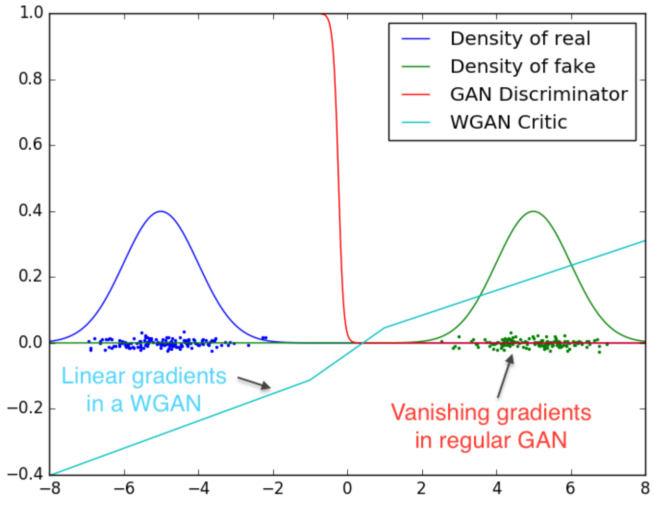


Figure 2.20. Optimal GAN discriminator and WGAN critic (discriminator) when learning to differentiate two Gaussians. The discriminator of a JS-GAN saturates and results in vanishing gradients, while Wassershtein critic (discriminator) provides clear gradients everywhere. Reproduced from [134]. Note: in this illustration the Lipsitz property is enforced strictly, via clipping the neural network weights.

also providing unbiased sample gradients. Like before, let us have two probability distributions P and Q over $\mathbb{R}^d$ and four independent random variables $X, X' \sim P$ and $Y, Y' \sim Q$. The energy distance will be:

$$D_E(P,Q) = 2\mathbb{E}\|X-Y\|_2 - \mathbb{E}\|X-X'\|_2 - \mathbb{E}\|Y-Y'\|_2, \tag{2.61}$$

where $\|X\|_2 = \sqrt{x_1^2 + \ldots + x_d^2}$ denotes the Euclidean norm. Energy distance can be reformulated as a difference of expectations. Define

$$f^\star(\mathbf{x}) = \mathbb{E}\|\mathbf{x} - Y'\|_2 - \mathbb{E}\|\mathbf{x} - X'\|_2, \tag{2.62}$$

then

$$D_E(P,Q) = \mathbb{E}f^\star(X) - \mathbb{E}f^\star(Y). \tag{2.63}$$

Training a generator to directly optimise the energy distance in the example space is problematic due to the curse of dimensionality. The authors propose using a GAN setup, where the discriminator neural network D maps from the example space to some latent space in which the energy distance is computed. The network is regularised with the gradient penalty discussed in section 2.7.2. The generator G is trained to minimise the distance, the discriminator – to maximise. Let us write the expressions for the losses. Define a helper function:

$$f(\mathbf{x}) = \mathbb{E}_{Y' \sim Q}\|D(\mathbf{x}) - D(Y')\|_2 - \mathbb{E}_{X' \sim P}\|D(\mathbf{x} - D(X')\|_2. \tag{2.64}$$

Then the generator loss will be

$$L_G(P,Q) = \mathbb{E}_{\mathbf{x} \sim P}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}' \sim Q}[f(\mathbf{x}')], \tag{2.65}$$

and discriminator loss:

$$L_D(P,Q) = -L_G(P,Q) + \lambda \mathbb{E}_{\widehat{\mathbf{x}}\sim p(\widehat{\mathbf{x}})}\left[(\|\nabla_{\widehat{\mathbf{x}}} D(\widehat{\mathbf{x}})\|_2 - 1)^2\right], \tag{2.66}$$

To estimate the gradient of the discriminator loss 2.66 with the respect to the discriminator parameters, four independent samples are required: two from the generator and two from the target distribution. In a situation, where we don't have access to two independent target samples, for example when learning conditional densities, an approximation can be used. Assume $D(\mathbf{x}) \approx 0$ for $\mathbf{x} \sim P$, then:

$$f_{\text{surrogate}}(\mathbf{x}) = \mathbb{E}_{Y\sim Q}\|D(\mathbf{x}) - D(Y)\|_2 - \|D(\mathbf{x})\|_2. \tag{2.67}$$

Altogether, the training algorithm is the same as WAGN-GP algorithm (7) with the exception of the loss computation, which is presented in Algorithm 8.

Algorithm 8 Cramer (energy) GAN losses

```
Require: The gradient penalty coefficient $\lambda$, 10 by default
  Sample a real data sample $\mathbf{x}_r \sim P$
  Sample two independent generated samples $\mathbf{x}_g, \mathbf{x}'_g \sim G(Z)$
  Sample $\epsilon \sim U(0,1)$
  $\widehat{\mathbf{x}} \leftarrow \epsilon\mathbf{x} + (1-\epsilon)\tilde{\mathbf{x}}$
  $f(\mathbf{x}) = \|D(\mathbf{x} - D(\mathbf{x}'_g)\|_2 - \|D(\mathbf{x})\|_2$          ▷ Define the helper function
  $L_G \leftarrow f(\mathbf{x}_r) - f(\mathbf{x}_g)$          ▷ Compute the surrogate generator loss
  $L_D \leftarrow -L_G + \lambda\left(|\nabla_{\widehat{\mathbf{x}}} f(\widehat{\mathbf{x}})\|_2 - 1\right)^2$
```

2.8 Conclusion

Machine learning lies in the intersection of artificial intelligence, statistics, and optimisation research. It enables a computer to automatically infer patterns from the given training dataset and use them to make predictions for cases yet unseen. Machine learning is not a silver bullet, and can not perform in the absolute absence of a priory knowledge. Its strength, however, is that said knowledge can be very general, but still, lead to superb results.

To successfully apply machine learning methods to a problem, several considerations must be addressed. The problem must be self-contained; everything needed for its solution must be present in the training data. The quality metric to evaluate the proposed solution must be formalised. Black-box solutions must be acceptable. Best-performing machine learning methods result in non-interpretable solutions.

There is no single best machine learning algorithm for all problems, and no easy way to predict which algorithm will be the best one for a particular problem. For most practical classification and regression problems on tabular data, the best results are achieved with either deep neural networks or gradient boosting decision trees. For the generative problems, methods based on deep neural networks are the clear favourites with many competing approaches tailoring them for particular use cases. Most of the state-of-the-art approaches for tabular and image data use the adversarial training, where one network is generating examples, and the other tries to distinguish between the real and generated examples.

Chapter 3

Machine Learning in High-Energy Physics

Standard Model has had a resounding success, almost all physics phenomena encountered on Earth can, in principle, be derived from it. The nuance is that almost all problems of interest are intractable if approached strictly ab initio. Even a seemingly simple problem, like predicting the result of a proton-proton collision, to be solved, has to be accompanied by a host of assumptions and simplifications [141, 142]. And for LHC experiments this complexity is multiplied by the complexity of a building-sized particle detector. And then, physical considerations only allow solving the "direct" problem: how will process X look in detector Y? In order to draw conclusions from an experiment, an inverse problem must be solved: given detector readout, what did occur inside it? Most of the utility of machine learning for particle physics comes from being able to solve this inverse problem.

Compared to traditional, expert-designed algorithms, machine learning has two primary advantages. First, empirically, for many problems, e. g. the ones listed in section 3.3, it was able to deliver better quality. Second, machine learning saves effort, by replacing manual algorithm design with an application of a general method – piggybacking on AI research, instead of developing a HEP-specific solution for every problem. The quality and convenience come at a cost. Most of the machine learning algorithms are black boxes. The main problem is not simply the lack human comprehension. Most machine learning algorithms work in quite specific mathematical setting, most importantly that the training data is distributed the same as the data to which the algorithm is to be applied. This is almost never the case in practice. Machine learning methods provide little formal guarantees of behaviour in a case of such data shift. For many non-scientific applications, this does not matter much. But science asks for rigour. There are methods, that are used to verify the data analysis techniques, both ML and not. Of course, they are not able to provide anything close to a strict guarantee, requiring expert judgement on a case-by-case basis.

In section 3.1 we discuss the typical approaches to training and validating machine learning algorithms; in section 3.2 some algorithms developed specifically for high-energy physics application; in section 3.3 some of the most established areas of application.

3.1 Training and Validation

Almost all demonstrated successful applications of machine learning in high-energy physics use the supervised learning approach. Training a supervised machine learning algorithm requires labelled data. Using simulated data is the most straightforward way for high-energy physics. They are by construction labelled and can be produced to cover the whole phase space. The biggest problem of the approach is the difference between real and simulated data, which limits the performance of the resulting algorithms and requires careful evaluation of the produced systematic uncertainty. Another concern is the computational power required to simulate a sufficient amount of events.

Training machine learning algorithms on real data mitigates the problem of the difference between simulation and data. But it faces a fundamental obstacle. To train an algorithm, labelled data is required. If there is a way to label the data, then what benefit can machine learning bring? For many practical applications, an algorithm is created merely to emulate a human, and manual labelling presents an obvious solution. But this is not the case for high-energy physics experiments.

For some problems, it is possible to label a part of the data with a method, that is specific to that part, and does not work in general. Such data are sometimes called a calibration sample. For example, in LHCb, $J/\psi \to \mu^+\mu^-$ decay can be reconstructed without using the PID response to one of the muons – see subsection 4.3.7. Using calibration samples does not suffer from the problems caused by imprecise simulation. But it is not without cons. Calibration samples do not always uniformly cover the whole phase space needed for the physics under study, the calibration sample selection procedure may introduce bias and calibration samples are often contaminated by background. These issues can be mitigated: phase space coverage can somewhat be addressed by reweighting. Background contribution can be accounted for using sideband subtraction and the methods discussed in reference [143] or chapter 6.

A combined approach is sometimes used, where the signal is simulated and the background comes from data. Background can either be selected in some expert-driven way, e. g. as an invariant mass fit sidebands, or by using evens collected under different experimental conditions. When machine learning is used in this way, extra care is needed, as the algorithm will by design confound the difference between signal and background and the difference between real and simulated data.

The available ways to train machine learning algorithms introduce bias due to the difference between the data on which an algorithm is trained, and those on which the same algorithm is applied. Conceptually, this is not unique to machine learning. Since the reconstruction and selection problems are intractable ab initio, some assumptions or heuristics are needed to solve them, with or without machine learning. The same concerns apply to both traditional cut-based selections and hand-designed reconstruction algorithms and machine learning-based ones. The concern with machine learning is that, if it is applied blindly and thoroughly, its potential to obtain a mathematically good, but physically meaningless solution is infinite. A good illustration is the "Flavours of Physics: Finding $\tau \to \mu\mu\mu$" competition [144]. The objective for participants was to build an operator, that would select $\tau \to \mu\mu\mu$ decays, one of the rarest process allowed in the SM with a branching fraction of the order of 10^{-40}. In the dataset the background was selected from real data side-bands obtained by cutting on the reconstructed invariant mass, and the signal was simulated. The winning solution, that has almost perfect validation score, reconstructs the invariant mass of the mother particle [145]. There was a check to prevent this sort of behaviour, but it was not sufficient. The resulting solution is not useful for physics. First, it provides little advantage over the mass cut that was used to construct the training dataset. Second, even if it offers some additional discriminating power, such selection makes it impossible to estimate the amount of background passing the selection with the usual method of fitting the signal and background invariant masses.

In general, HEP community approaches validation of machine-learning methods in the same way, as validating all other data analysis methods. Case-by-case physical considerations are used at every stage of the problem solving to arrive at an acceptable result:

1. Training data selection. The data must cover the whole desired phase space and all the physical processes that we want to learn.

2. Features selection. Only the features, that correspond to the physics involved, but not the bias in data selection/simulation are used. For example, when using the $J/\psi \to \mu^+\mu^-$ calibration sample for training muon identification, using muon detector occupancy as a feature would result in bias, as a muon obtained from $J/\psi \to \mu^+\mu^-$ is always accompanied by another one, which might not be the case for muons produced in other decays.

3. Validating on different data samples. For an algorithm trained on simulated data, its performance on a calibration sample is checked, e. g. PID validation is subsection 7.5.2. For an algorithm trained on a calibration sample, its performance on a different calibration sample is checked, e. g. fast simulation in 8.5.2.

4. Manual inspection of physically-meaningful distributions. E. g. it is expected, that for a correctly selected and reconstructed J/ψ sample the invariant mass distribution is a Gaussian with mean 3.1 GeV/c^2, and this is monitored during the LHCb data collection.

3.2 HEP-specific Machine Learning

As discussed in the previous section, incorporating machine learning algorithms into a data analysis pipeline is a delicate affair. There are a few machine learning algorithms developed specifically to take advantage of the common assumptions used during high-energy physics data analysis – as opposed to the more common modus operandi, where the rest of the analysis is twisted to accommodate an out-of-the-box machine learning solution. It this section we cover two most established such methods.

3.2.1 Learning to Pivot with Adversarial Networks

The method is proposed by reference [146]. For some problems in high-energy physics, the difference between the data used for training a model and the data to which it is applied can be narrowed to a small number of nuisance parameters. We assume that the data generation process is conditional on those parameters, and the distribution of those for train and test data are different. For example, reference [147] builds a jet substructure tagger, that is largely uncorrelated[1] with the jet mass.

The idea of the method builds on the adversarial training described in subsection 2.7.1 and the idea of a pivot – a quantity whose distribution is invariant with the nuisance parameters. Formally, let $p(\mathbf{x}, \mathbf{y}, \mathbf{z})$ be the probability density function, where $\mathbf{x}$ are features, $\mathbf{y}$ – label, and $\mathbf{z}$ – the nuisance parameters. Consider a classification or regression problem, where we learn a function $f : (\mathbb{X}, \boldsymbol{\theta}_f) \to \mathbb{S}$, where $\boldsymbol{\theta}_f$ is the vector of the model parameters. $\boldsymbol{\theta}_f$ are the model parameters, e. g. neural network weights; $\mathbb{S}$ is the model output space. For regression, $\mathbb{S} = \mathbb{Y}$; for classification $\mathbb{S}$ are the class scores, $\mathbb{S} = \mathbb{R}^{|\mathbb{Y}|}$.

To require robustness with respect to the nuisance parameter, the authors introduce an additional objective:

$$\forall \mathbf{z}, \mathbf{z\prime} \in \mathbb{Z}; \forall s \in \mathbb{S} : \mathbf{x} \sim p(\mathbb{X}, \mathbf{z}), \mathbf{x\prime} \sim p(\mathbb{X}, \mathbf{z\prime}), P(f(\mathbf{x}, \boldsymbol{\theta}_f) = s) = P(f(\mathbf{x\prime}, \boldsymbol{\theta}_f) = s). \tag{3.1}$$

In other words, they require that the output of the learned operator $f(\mathbf{x}, \boldsymbol{\theta}_f)$ and the nuisance parameters are independent random variables.

This is achieved by an adversarial training setup illustrated in figure 3.1. Model r is trained to predict the posterior probability $p_{\theta_r}(\mathbf{z}|f(\mathbb{X}, \boldsymbol{\theta}_f) = s)$, where $\boldsymbol{\theta}_r$ are its parameters. If it is able to recover information about z, then model f is not robust with the respect to nuisance parameters. Mathematically, the training procedure is

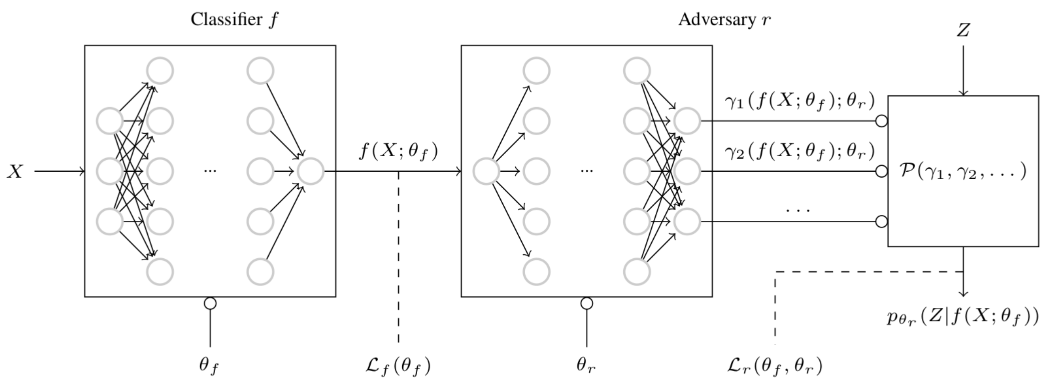

Classifier f
Adversary r
Z
X
...
$f(X;\theta_f)$
...
$\gamma_1(f(X;\theta_f);\theta_r)$
$\gamma_2(f(X;\theta_f);\theta_r)$
...
$\mathcal{P}(\gamma_1, \gamma_2, \ldots)$
$p_{\theta_r}(Z|f(X;\theta_f))$
θ_f
$\mathcal{L}_f(\theta_f)$
θ_r
$\mathcal{L}_r(\theta_f, \theta_r)$

Figure 3.1. Sketch of the adversarial training mechanism. f is a binary classifier, Z are the nuisance parameters. The adversary is trained to infer information about the nuisance parameters from the classifier output. The classifier is trained to both classify and fool the adversary. Reproduced from [146].

[1]Strictly speaking, "uncorrelated" means lack of a very specific linear relationship between random variables. In the high-energy physics community, it is sometimes used as a soft synonym to "independent".

a minimax problem:

$$E(\boldsymbol{\theta}_f, \boldsymbol{\theta}_r) = L_f(\boldsymbol{\theta}_f) - \lambda L_r(\boldsymbol{\theta}_f, \boldsymbol{\theta}_r), \tag{3.2}$$

where $L_f(\boldsymbol{\theta}_f)$ is the loss function for the original problem and $L_r(\boldsymbol{\theta}_f, \boldsymbol{\theta}_r)$ is the loss function for the adversary and λ is a coefficient responsible for the trade-off between the pivotal quantity and classification/regression performance. For a binary classification problem, where f models $P(\mathbf{y} = 1|\mathbf{x})$, $L_f(\boldsymbol{\theta}_f)$ and $L_r(\boldsymbol{\theta}_f, \boldsymbol{\theta}_r)$ are set, respectively, to the expected value of the negative log-likelihood of $\mathbf{y}|\mathbf{x}$ under f and of $\mathbf{z}|f(\mathbb{X}, \boldsymbol{\theta}_f)$ under r:

$$L_f(\theta_f) = \mathbb{E}_{\mathbf{x},\mathbf{y}}[-\log p_{\boldsymbol{\theta}_f}(y|x)] \tag{3.3}$$

$$L_r(\theta_f, \theta_r) = \mathbb{E}_{s\sim f(\mathbb{X},\boldsymbol{\theta}_f), z\sim \mathbb{Z}|s}[-\log p_{\boldsymbol{\theta}_r}(z|s)]. \tag{3.4}$$

The training objective then is:

$$\boldsymbol{\theta_f}, \boldsymbol{\theta}_r = \arg\min_{\boldsymbol{\theta}_f} \max_{\boldsymbol{\theta}_r} E(\boldsymbol{\theta}_f, \boldsymbol{\theta}_r). \tag{3.5}$$

This is a very elegant result. Its practical utility for physics is limited by two factors. First, as the authors write, the optimal classifier often does not have the pivotal quantity, and instead of a well-defined solution, it is often desirable to find some heuristic trade-off by varying the λ parameter. Second, consider this conundrum. If there is a dataset to evaluate an algorithm for the desired range of the nuisance parameters, the dataset can also be used to train an optimal algorithm, without a need to pivot. If there is no such dataset, the performance of the algorithm can not be evaluated and it is difficult to use it in a scientific context.

An example where this method is useful is presented in reference [147]. The authors create a model for jet classification. Since they lack the knowledge of the true background model, they use a common experimentalist method for for estimating the background count in the signal region. First, do a mass fit, second, interpolate the background distribution from the sidebands into the signal region. For this method to work for data selected by a classifier, the output of the classifier must be flat with the respect to reconstructed mass. Such classifier is built with the pivoting method.

3.2.2 Boosting to Uniformity

Consider a somewhat easier problem than in the previous section. First, instead of requiring independence between the nuisance parameters and the model output, require only that the classifier efficiency is uniform with the respect to them[2]. Second, consider the situation, where the nuisance parameters are low-dimensional. It is the case in many high-energy physics data analysis cases, where the nuisance parameter is just an invariant mass, or two, in a case of a Dalitz plot. This allows for avoiding the complexity of an adversarial setup. The idea was first proposed in reference [148] and then later refined in [149].

[2]Maybe this simplification is not necessary and the algorithms described in this section also theoretically provide independence. The references where they are proposed do not make any such claim though.

The authors propose to add to the loss function a component that would penalise non-uniform efficiency. To do that, for each event find the group of events, that are near it in the space of the nuisance parameters. This can be done either by binning or by explicitly finding the closest events. Then, check whether the distribution of the classifier response in each group is the same as the global response. There are many possible measures to use, they are discussed in the appendix of reference [149]. The one that has the best performance in their test is based on the Kolmogorov–Smirnov test. Consider two single-dimensional probability distributions defined by their cumulative distribution functions $F_1(s)$ and $F_2(s)$. The Kolmogorov–Smirnov statistic describing the difference between them would be:

$$D_{\text{KS}} = \sup_s |F_1(s) - F_2(s)|\,. \tag{3.6}$$

It is illustrated in figure 3.2.

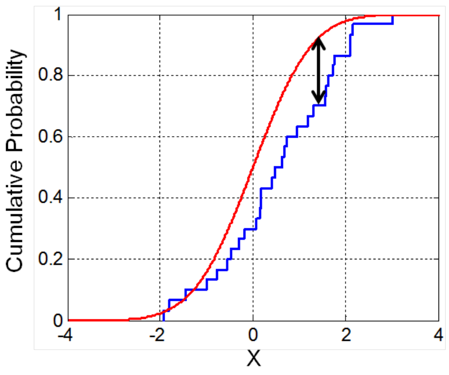


Figure 3.2. Illustration of the Kolmogorov–Smirnov statistic. Red and blue lines are the cumulative distribution functions. The black arrow is the KS statistic. Reproduced from [150].

Optimising the Kolmogorov–Smirnov statistic as a flatness measure leads to instability, as its gradient is zero for examples with responses $s' > \arg\max_s |F_1(s) - F_2(s)|$. The authors propose the following measure which does not suffer from this problem:

$$D_{\text{flat}} = \int |F_1(s) - F_2(s)|^2\, ds. \tag{3.7}$$

The overall flatness loss would be

$$L_{\text{flat}} = \sum_b \int |F_b(s) - F(s)|^2\, ds, \tag{3.8}$$

where b is the bin index, $F_b(s)$ is the CDF of the classifier response in the b-th bin, $F(s)$ is the overall classifier response CDF. The total loss would be a sum of the classification and flatness losses

$$L = L_{\text{classification}} + \lambda L_{\text{flat}}, \tag{3.9}$$

where λ is a heuristic parameter that governs the trade-off between flatness and classification accuracy.

The method addresses the same problem as the pivotal neural networks, namely incompatibility of a straightforward application of machine learning with a heuristic invariant mass fit. Compared to it, flatness loss is computationally easier – an optimisation problem, instead of a minimax problem. Flatness loss also does not impose the requirement, that the models are continuously differentiable, allowing the usage of boosted decision trees. As a price, it does not scale to the case of high-dimensional nuisance parameters – but this is not the case in the majority of applications anyway. In reference [151] their performance is compared for a jet tagging problem, with pivoting showing slightly better results.

Uniformity boosting has been used in several LHCb analyses, namely references [152, 153].

3.3 Primary Applications

3.3.1 Event Selection: Separating Signal and Background

Selecting events that contain interesting processes is a fundamental requirement of high-energy physics experiments and probably is the most established area of application of machine learning in high-energy physics. Most analyses consist of measuring the fraction of events that contain a specific decay channel. The usual way of doing that consists of building an event selection algorithm, estimating its efficiency and background rejection and measuring the count of events passing it. The traditional way of selection is the so-called cut-based selection, basically building a decision tree described in subsection 2.6.1 manually, using either physical considerations, Monte-Carlo simulation, or both. So it was only natural to automate this procedure, the earliest reference goes at least to 1991 [154].

To train an event selection algorithm, one needs datasets with signal and background. The usual source is the simulation, e. g. ATLAS [155] and LHCb [156]. Sometimes the approach of combining the data and simulation is used. For example, a Higgs search in ATLAS [157] uses training on simulated signal and real-data background selected as a mass fit sidebands. Another possibility is using an invariant mass fit to obtain the probabilities that given examples are signal and use them to train a machine learning algorithm. This approach has been applied to background suppression in the measurement of the Υ polarization at CDF II [158]. Using the invariant mass fits for the purposes of training classifiers is discussed in-depth in chapter 6.

3.3.2 Event Reconstruction

Reconstruction is a process by which raw detector readout is transformed into physically-meaningful objects, namely particle tracks, particle types, and vertices. For ATLAS, CMS and LHCb experiments, this is separated into three distinct operations. First, the charged tracks are reconstructed using the input from the tracking subsystem. Second, the information from the particle identification subsystems is used to assign the particle types to the charged tracks. Third, the calorimeters are used to find some of the neutral particles. Conceptually, reconstruction is the inverse problem for simulation. For the later we compute the detector response given the particles, for the former, we find the particles that have caused given detector response.

Machine learning is a natural way to approach the problem, especially considering its purely algorithmic nature. The most straightforward way is to simulate something, use the detector response as features for a machine learning algorithm, and the Monte-Carlo truth as the labels. The most challenging aspects are algorithmic. A high-energy physics event is a complex, structured object, detector response even more so. Out-of-the-box machine learning algorithms are not good at dealing with this kind of data. The usual approach is to use domain expertise to program most of the reconstruction, augmented by machine learning, where some subproblem can be formulated as a classic classification/regression problem.

Tracking. LHCb uses neural networks used to reject fake tracks and track seeds at various stages of the reconstruction algorithm [159]. They are trained on simulation. In ATLAS tracking neural networks are used to identify the merged clusters, that appear when two tracks pass through the tracking station close together [160]. They are also trained on simulation.

Jet tagging. Jets are very complex objects to simulate with high fidelity, and even more so to classify without resorting to heuristic algorithms. And machine learning is the pinnacle of heuristic algorithms. LHCb uses boosted decision trees on top of expert-designed feature extraction for jet flavor classification [161, 162]. There is also a pilot study on using deep learning for jet flavour classification with deep learning for a generic simplified detector [163].

Particle identification. It is a classification problem, so it yields naturally to machine learning methods. Given a track and detector readout near it, what is the likelihood that the particle was of the given type? In addition to my work on LHCb described in chapters 7 and 5, neural networks have been used for LHCb charged particle identification [10]. It uses neural networks on top of high-level expert-generated features [164]. There is a work that proposes using boosted decision trees applied to raw calorimeter readout [165] for neutral particle identification.

Identifying the flavour of B^0 mesons at LHCb by using particles associated with its production, the so-called same-side (SS) tagging [166]. The algorithms exploit the charge correlation of pions and protons with B^0 mesons. For the training dataset, a real data sample of B^0 mesons decaying into the flavour-specific final states $D^-\pi^+$ and $K^+\pi^-$ is selected by the final state. As features, they use high-level kinematic observables along with particle type likelihoods.

3.3.3 Monitoring and Data Quality

High-energy physics detectors are very complex and sensitive machines. If even a small part of a detector malfunctions, this has a potential to invalidate the collected data. Therefore, all the detectors, and the LHC machine itself, are equipped with sophisticated monitoring systems. They check that parameters on all levels of the data collection and processing lie within acceptable range: from voltages on individual electronic components to reconstructed masses of particles from known decays. A big challenge for the monitoring system is that it must be able to recognise legitimate changes of data taking conditions from equipment malfunctions.

The LHCb data quality monitoring system operates by comparing different variables to references [167]. The variables that must be monitored are chosen by experts and range from low-level, e. g. muon pads occupancies, to high-level, e. g. the reconstructed J/ψ mass. The references are also provided by the relevant experts and are updated as needed. The primary operation protocol of the monitoring system is humans looking at histograms. If an observed variable distribution are different from the corresponding reference, the operator investigates the discrepancy; if it has not been flagged as addressed before, the incident is recorded and the relevant expert is contacted.

The operator is assisted by a system of automatic alarms, that call his attention to variables, where large differences between the reference and observed values is being observed. Such automation system is susceptible both to false alarms and missing anomalous data. False alarms occur when the reference is not updated in time and the discrepancy is due to a legitimate change in data taking conditions. Missing anomalous data occurs when an anomaly manifests itself in a shift in many variables, that is small enough to pass below the alarm threshold for each variable individually. In principle, machine learning can do well in addressing the second issue, but it is not trivial to train such an algorithm. Since the anomalies in data are, thankfully, rare, it is hard to build a training dataset. Also, since the references change with time, training on old data must take that into account. And finally, even if an algorithm successfully determines that data is anomalous, this information per se is hard to act upon without knowing the part of the detector where the malfunction is. Altogether, this means that the problem of data quality monitoring is not a straightforward one to address with machine learning, and expert-written algorithms and human operators are here to stay.

Reference [168] describes a machine learning solution for assisting the human operator used at LHCb, called Roboshifter. As features, it uses Kolmogorov-Smirnov distances between the histograms and the corresponding references. This allows the Roboshifter to take into account the evolving nature of detector configuration. The algorithm is trained on a database of historic LHCb data collection intervals (runs), marked by experts as good or bad. Roboshifter uses boosted decision trees to predict the probability that the current run is bad. It uses shallow trees of depth 1, so that each tree makes a decision based on a single histogram. This way the contributions of individual histograms to the final decision can be estimated and the suspicious histograms reported to the user. I could not find a published peer-reviewed evaluation of Roboshifter quality, but it is deployed in the LHCb monitoring system. Roboshifter makes the job of the operator easier – and makes him less likely to miss an anomaly. The principal limitation is that it relies on expert-selected variables, expert-provided references, and expert-labelled training data, so roboshifter is unlikely to detect a problem, that would have gone undetected by the manual system.

3.4 Conclusion and Outlook

For natural scientists, the utility of machine learning is tackling complex relationships in the data, that are difficult to establish explicitly. Many analyses in high-energy physics essentially count the density of events in a particular region of the phase space. And most of the effort goes into defining said region and determining which events fall into it. Machine learning allows to use simulated and, in some cases, real data to automate the process, often leading to better quality results. For example, for a Higgs search, the increase in sensitivity due to the use of machine learning was equivalent to collecting 50% more data [155].

Machine learning has been used in high-energy physics since at least 1988 [169], more or less keeping with the advancement in the field. There is a continuous flow of works applying newly-developed machine learning methods to high-energy physics. In addition to the established applications discussed in this chapter, there is a host of potential areas being explored. Generative models application to simulation is discussed in depth in chapter 8. As machine learning methods for handling structured data advance, so are studies to applying them to high-energy physics. References [170, 171] propose jet classification algorithms based on the similarity between images and calorimeter responses. One of the most-funded and rapidly developing areas of machine learning is natural language processing. And natural languages are sequences. References [172, 173, 174] consider jet as a sequence of particles and apply state-of-the-art sequence processing algorithms to them. Tracking is another area, where data can be interpreted as sequences. Reference [175] explores machine-learning based approach to sequences for tracking. Graph neural networks [176, 177, 178, 179, 180] allow using machine learning for the cases, where an example can be represented as a graph. Tracking again provides a perspective case, where hits are considered edges and close hits are connected with vertices [181].

The trend is that machine learning applications for high energy physics grow both in number and diversity of problems solved. ML can deliver a meaningful improvement in terms of physics performance over the previously used methods. Considering the AI winters, I will stay clear of making predictions. But the future looks quite exciting for sure.

Chapter 4

The LHCb experiment

This chapter is a description of the LHCb experiment with a focus on the particle identification-related areas, where this dissertation made contributions. As of the moment (April 2020), the Large Hadron Collider (LHC) and the detectors on it are being upgraded in preparation to Run 3 starting in 2021 [182, 13]. Unless otherwise noted, the state for the LHC Run 2 (2015-2018) is described.

This chapter is structured as follows. In section 4.1, we introduce the Large Hadron Collider. In section 4.2, we describe the hardware of the LHCb detector. In section 4.3, we describe the data analysis software and algorithms at the LHCb experiment.

4.1 The Large Hadron Collider (LHC)

Quoting the CERN website, the Large Hadron Collider (LHC) is the world's largest and most powerful particle accelerator. It consists of a 27-kilometre ring of superconducting magnets with several accelerating structures to boost the energy of the particles along the way [183]. The purpose of the machine is to answer some of the open questions in fundamental physics, such as the origin of mass, existence of dark matter, baryonic asymmetry.

So far (April 2020), all the results obtained are consistent with the predictions of the Standard Model of particle physics. They provide valuable limits for possible theories extending it. At the same time, some observations testing lepton universality by LHCb and Belle experiments deviate from the Standard Model [4]. The measurement uncertainty is too high to claim a discovery – but demands further study.

4.1.1 The LHC Accelerator System

In order to accelerate protons to the energy of 6.5 TeV, a multiple-stage system is used. It is depicted on figure 4.1. The first stage is a linear accelerator, LINAC2. The second stage is the Proton Synchrotron Booster. The third stage is the Proton Synchrotron (PS), which accelerates protons up to an energy of 25 GeV. The next step is the Super Proton Synchrotron (SPS). It accelerates protons up to an energy of 450 GeV. SPS has been used as collider on its own in the 1980-s and provided

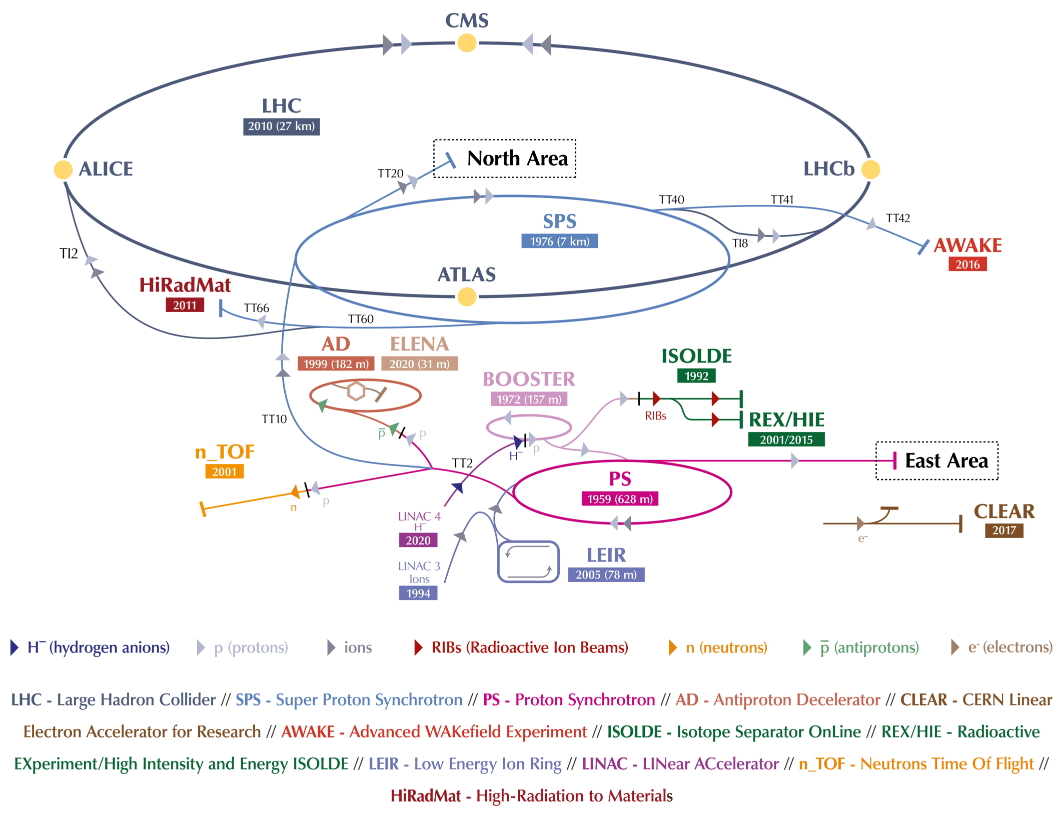

CMS
LHC
2010 (27 km)
ALICE
North Area
LHCb
TT20
TT40
TT41
TT42
SPS
1976 (7 km)
TI8
AWAKE
2016
TI2
HiRadMat
2011
TT66
ATLAS
TT60
AD
1999 (182 m)
ELENA
2020 (31 m)
BOOSTER
1972 (157 m)
ISOLDE
1992
RIBs
REX/HIE
2001/2015
TT10
n_TOF
2001
TT2
East Area
PS
1959 (628 m)
CLEAR
2017
LINAC 4
H⁻
2020
LINAC 3
Ions
1994
LEIR
2005 (78 m)
H⁻ (hydrogen anions)
p (protons)
ions
RIBs (Radioactive Ion Beams)
n (neutrons)
p̄ (antiprotons)
e⁻ (electrons)
LHC - Large Hadron Collider // SPS - Super Proton Synchrotron // PS - Proton Synchrotron // AD - Antiproton Decelerator // CLEAR - CERN Linear Electron Accelerator for Research // AWAKE - Advanced WAKefield Experiment // ISOLDE - Isotope Separator OnLine // REX/HIE - Radioactive EXperiment/High Intensity and Energy ISOLDE // LEIR - Low Energy Ion Ring // LINAC - LINear ACcelerator // n_TOF - Neutrons Time Of Flight // HiRadMat - High-Radiation to Materials

Figure 4.1. The CERN accelerator complex. Reproduced from [184].

the data for the discovery of W and Z bosons. From SPS the protons are injected into the LHC.

4.1.2 The Large Experiments at the LHC

Four large experiments study particle collisions at the LHC. Two of them use general-purpose detectors; CMS (Compact Muon Solenoid) and ATLAS (A Toroidal LHC ApparatuS). They have the same conceptual composition. Both detectors consist of four concentric subsystems: an inner tracker, composed of semiconductor chips, contained within a magnetic field; an electromagnetic calorimeter (ECAL); a Hadronic calorimeter (HCAL), and a muon detector. True to being general-purpose detectors, they serve an extensive physics program, searching for New Physics in a broad range of phenomena. ATLAS [185] and CMS [186] are responsible for, probably the most famous CERN discovery, that of Higgs boson in 2012.

ALICE (A Large Ion Collider Experiment) aims to study heavy-ion (Pb-Pb nuclei) collisions. Its objectives are exploring the physics of strongly interacting matter at extreme energy densities and investigating quark-gluon plasma.

The LHCb (Large Hadron Collider beauty) experiment studies b-physics. The LHCb detector was designed with priority given to momentum resolution and particle identification precision. It is single-arm, not hermetic, and covers the pseudo-rapidity range $2 < \eta < 5$ (15 to 250 mrad of solid angle). The LHCb detector is described in the next section.

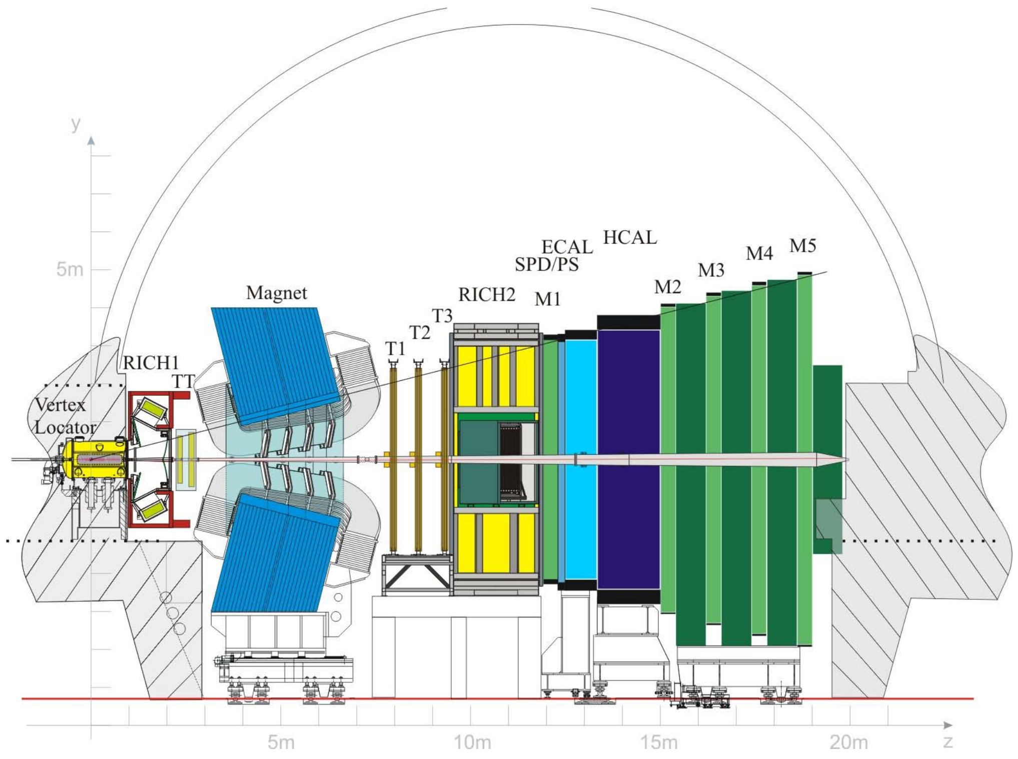


Figure 4.2. The layout of the LHCb detector, non-bending vertical plane. Illustration source: LHCb collaboration.

4.2 The LHCb Detector

The detector consists of the vertex detector (VELO), the first Cerenkov detector (RICH1), tracing stations (TT, Trigger Tracker, Tracker Turicensis), the magnet, tracking stations (T1-T3), the second Cerenkov detector (RICH2), the electromagnetic calorimeter (ECAL), the hadron calorimeter (HCAL), and the muon system (M1-M5), the first station (M1) is located before the calorimeters and the rest – after. M1 has already been removed during Long Shutdown 2 in view of the new trigger approach adopted for Run 3. A scheme of the LHCb detector is presented in figure 4.2.

4.2.1 Tracking

The LHCb uses several subsystems for track reconstruction [187]. A high-precision silicon-strip vertex detector, VELO, surrounding the *pp* interaction region; a large-area silicon-strip detector, TT, located upstream of the magnet; three stations of silicon-strip detectors (Inner Tracker, IT) and straw drift tubes (Outer Tracker, OT) placed downstream of the magnet, together called the T stations. The LHCb collaboration classifies the tracks into several types, based on the subdetectors that have hits that correspond to the track. An overview of the classification is presented in figure 4.3. A long track has hits in all the subsystems; a downstream track only in the TT and the T stations; an upstream track in the VELO and the TT; a VELO

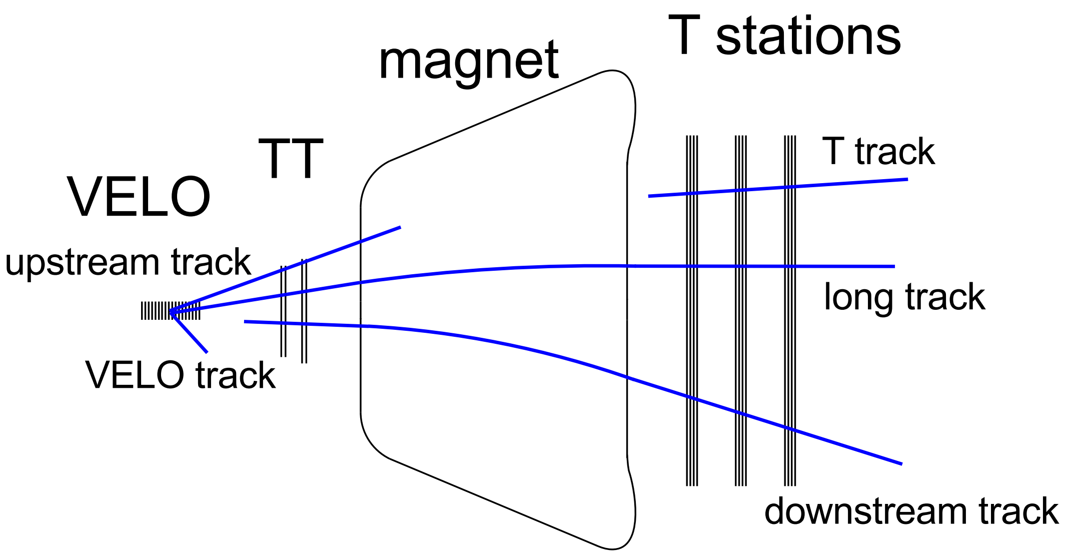


Figure 4.3. Track types in LHCb. Reproduced from [187].

track only in the VELO; and, finally, a T track only in the T stations.

4.2.1.1 Vertex locator (VELO)

The objective of the vertex locator is high-precision track reconstruction close to the interaction region. Its sensors are placed only 7 mm from the LHC beams, and the geometry is designed so that all tracks inside the nominal LHCb acceptance cross at least three VELO stations[188]. This allows achieving high resolution of the secondary vertices position, which is crucial for measuring the decay time of weakly decaying particles. An overview of the VELO is presented in figure 4.4. The VELO consists of silicon microstrips arranged along the beam direction. To allow measurements close to the beamline, the subdetector consists of two retractable halves that are closed at the beginning of each fill. There are 21 standard modules in each VELO half.

4.2.1.2 Tracking stations

The main tracking system comprises four tracking stations: TT located between RICH-1 and the magnet, and three stations (T1-T3) located 3 meters downstream between after the magnet and before the RICH-2 (see figure 4.2).

A sketch of the layout of TT layers is shown in figure 4.5. The TT and IT detectors use silicon microstrip sensors with a strip pitch of 183 μm and 198 μm, respectively. The TT is about 150 cm wide and 130 cm high. The IT covers a 120 cm wide and 40 cm high cross-shaped region in the centre of the three tracking stations T1 – T3. Despite covering around 2% of the total area, it processes around 20% of particle flux [190]. Each of the tracking stations has four detection layers in an $x-u-v-x$ arrangement with vertical strips in each of the two x layers, and strips rotated by a stereo angle of $-5°$ and $+5°$ in the u and v layers, respectively [10].

OT uses cheaper straw-tube drift chambers. It consists of approximately 200

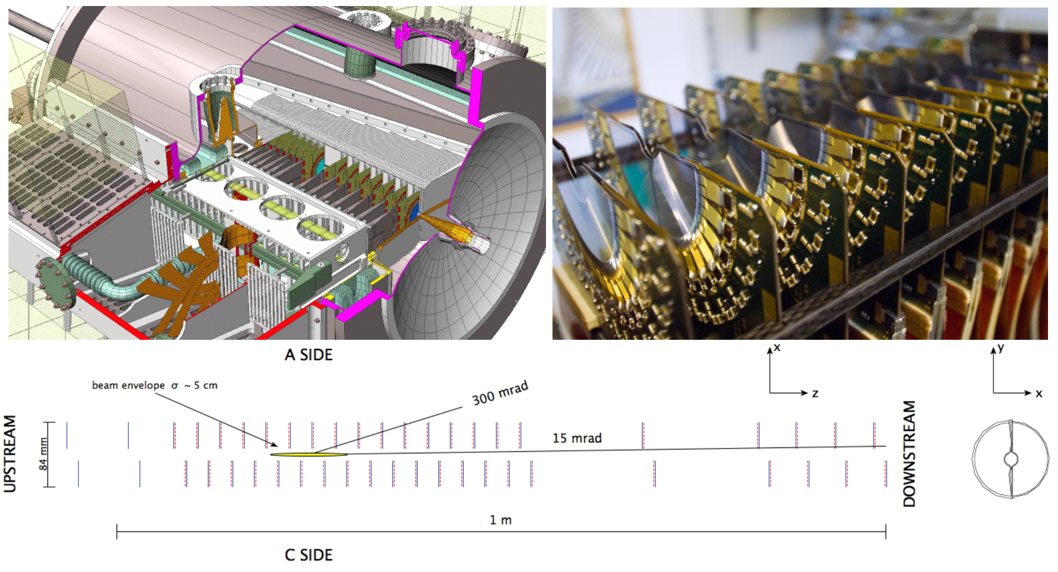


Figure 4.4. (top left) The LHCb VELO vacuum tank. (top right) A photograph of one side of the VELO during assembly showing the silicon sensors and the readout hybrids. (bottom) Cross-section of the detector in the closed position in the xz plane at $y = 0$ of the sensors; and a view of the sensors in the xy plane. Reproduced from [189].

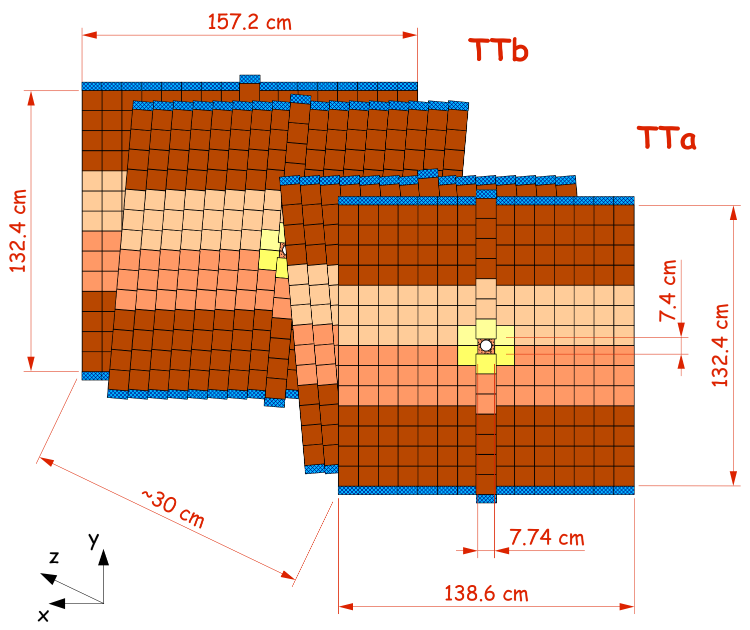


Figure 4.5. Layout of TT layers. Reproduced from [191]

modules. Each module contains two staggered layers of drift-tubes with an inner diameter of 4.9 mm. A mixture of Argon (70%), CO_2 (28.5%) and O_2 (1.5%) is chosen to guarantee a drift time below 50 ns and a spatial resolution of 200 μm. As the T1-T3 stations, the OT has four layers arranged in an $x-u-v-x$ geometry [10].

4.2.2 Particle Identification

Particle identification at LHCb relies on the Cherenkov detectors, calorimeters, and the muon system. Overall they provide better particle identification, than in ATLAS and CMS detectors, in particular because of the presence of the two RICHs.

This subsection describes some of the PID subsystems and the particle identification algorithms specific to them. The Muon detector is described in the dedicated chapter, section 5.1. In chapter 7 we describe how information from the subsystems is combined to make the final decision about the particle type.

4.2.2.1 Ring-Imaging Cherenkov detector (RICH)

The Cherenkov detectors rely on the Cherenkov radiation – electromagnetic radiation emitted when a charged particle passes through a dielectric medium at speed greater than the phase velocity of light in that medium [192]. The emission angle θ is defined by the equation:

$$\cos\theta = \frac{c}{nv}, \tag{4.1}$$

where c is the speed of light in vacuum, and n is the medium refractive index, v is the particle velocity. Therefore, it is possible to find the particle velocity from the emission angle, and, combining it with the known momentum, find the particle mass.

The LHCb detector includes two RICH detectors. They are similar in construction and principle of operation; a sophisticated system of spherical and plane mirrors which reflect Cherenkov light from the radiator to the photomultipliers (PMTs).

RICH1 (depicted on figure 4.6) is located upstream of the magnet and covers the low momentum range (around $1 - 60$ GeV/c) [11]. It covers the full LHCb angular momentum acceptance range and uses a C_4F_{10} radiator. The refractive index depends on the wavelength, at $\lambda = 400$ nm, $n = 1.0014$.

RICH2 is located downstream of the magnet and is optimized for high-momentum particles. It is depicted in figure 4.7. RICH2 uses a CF_4 radiator with refractive index $n = 1.0005$ at wavelength $\lambda = 400$. It covers the high momentum range, from about 15 GeV/c up to and beyond 100 GeV/c [11].

A typical LHCb event in RICH1 is shown on figure 4.8. To identify particles, the following procedure is used [194]:

1. Compute the emission angles for pairs of tracks and pixels.

2. Construct the likelihood for all photons, all tracks and all radiators at once:

$$\mathbb{L} = -\sum_{\text{track } j} \mu_j + \sum_{\text{pixel } i} n_i \ln\left(\sum_{\text{track } j} a_{ij} + b_i\right), \tag{4.2}$$

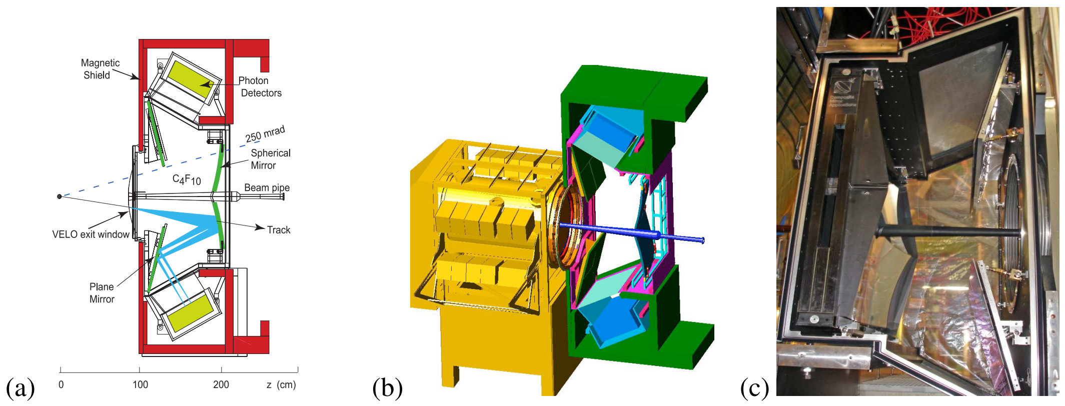


Figure 4.6. RICH1 (a) Side view scheme, Run 2 configuration. (b) Cut-away 3D model, to the left is the the VELO tank, Run 1 configuration. (c) Photo of the gas enclosure containing the flat and spherical mirrors, Run 1 configuration. In (a) and (b) the interaction point is on the left, in (c) it is on the right. Reproduced from [11, 193].

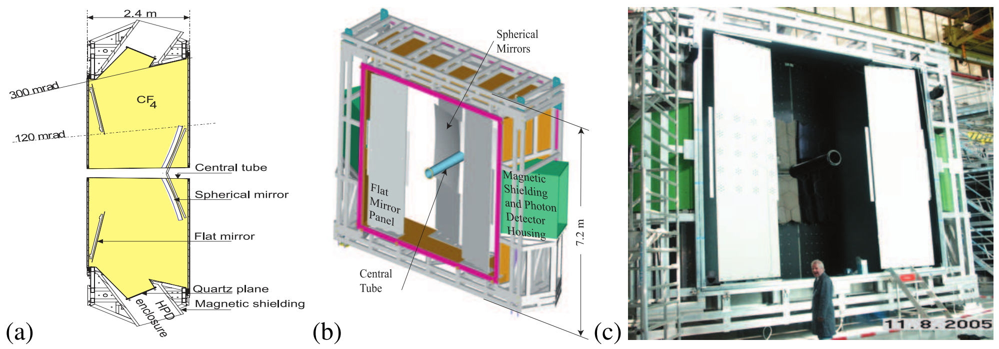


Figure 4.7. RICH2 in Run 1 configuration. (a) A top view schematic. (b) A 3D rendering schematic. (c) A photograph with the entrance window removed. Reproduced from [11].

where a_{ij} is the number of expected photons from track j in pixel i (for a given PID hyposesis); b_i is background in pixel i (without any associated track), n_i is the number of photons in pixel i, $\mu_j = \sum a_{ij}$ is expected total number of detected photons; $n_i \in \{0, 1\}$ is the digital readout.

3. Assume all PID hypotheses to be pions (or seed from previous reconstruction), estimate the background parameters b_i.
4. Calculate the likelihood of the observed pixel distribution.
5. Greedely maximize the likelihood. Iterate:
    - Change PID hypothesis for one track at a time
    - Recalculate likelihood
    - Select the change with the biggest likelihood improvement
    - Assign new PID to that track

The computed likelihoods for hypothesis h are expressed as delta-log-likelihoods (DLL) relative to the pion hypothesis (the most abundant species):

$$\mathrm{DLL}(\mathrm{track}, h) = \log \mathbb{L}(\mathrm{track}, h) - \log \mathbb{L}(\mathrm{track}, \pi). \tag{4.3}$$

The effective momentum range is primarily limited by the overlapping angle distributions for the mass hypotheses at high momentum and photon emission threshold at low momentum. The distribution of the reconstructed Cherenkov angle as a function of track momentum is presented on figure 4.9. The performance of the RICH PID are presented in figures 4.10, 4.11, 4.12. In most analyses, the RICH DLLs are not used directly, but as a component in a single particle identification decision combining the information from all the subdetectcors. The algorithms for this are described in chapter 7.

4.2.2.2 Calorimeters

The calorimeters at LHCb serve several objectives [190]. First, they are the only subsystem that measures the energy and position of neutral particles. They separate single photons and π_0 decays, which is essential for flavour tagging and the study of B-meson decays [11]. Second, the measured energies are used in selecting interesting events in the trigger (see section 4.3). Third, the calorimeters help with particle identification between electrons, photons and hadrons.

LHCb uses a classical structure with an electromagnetic calorimeter (ECAL) followed by a hadron calorimeter (HCAL). Before the ECAL there is the Scintillator Pad Detector (SPD), a 2.5 radiation lengths lead wall, and the Preshower (PS) [195]. Together, they allow for high-quality separation of neutral particles, electrons and hadrons, as they deposit energy in different layers, as shown on figure 4.13.

All calorimeters follow the same basic principle: scintillation light is transmitted to a Photo-Multiplier (PMT) by wavelength-shifting fibres. The hit density falls drastically with the distance from the beam pipe. The calorimeters are therefore

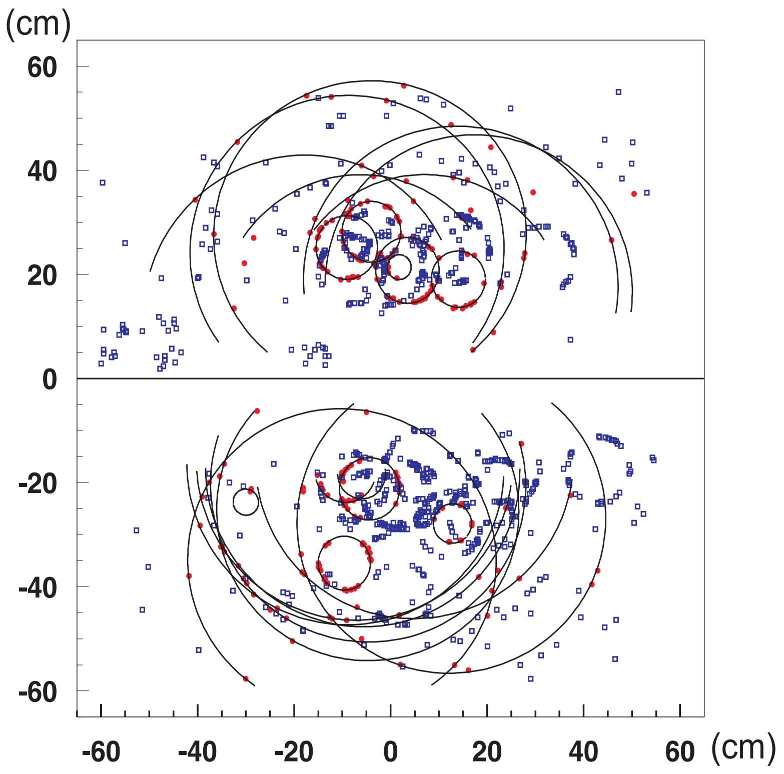


Figure 4.8. A typical LHCb event in RICH1. The circles correspond to the assigned particles types, red dots to hits associated to tracks, and blue dots to hits not associated to tracks (and thus considered noise). Reproduced from [11].

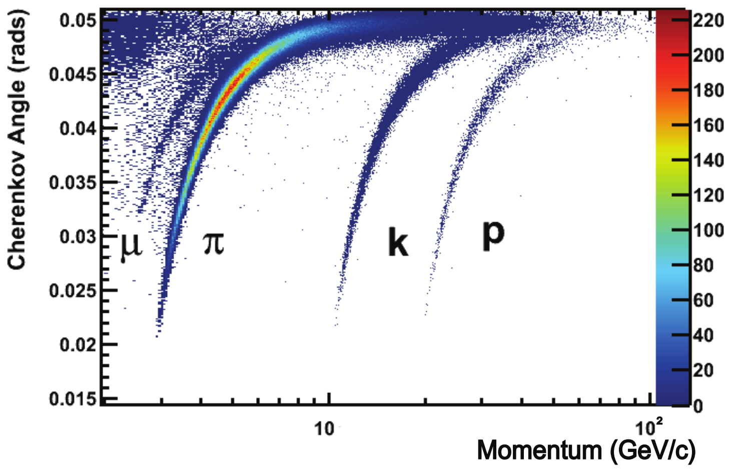


Figure 4.9. Reconstructed Cherenkov angle as a function of track momentum in RICH1 for isolated tracks selected in data (2% of all tracks). Reproduced from [168].

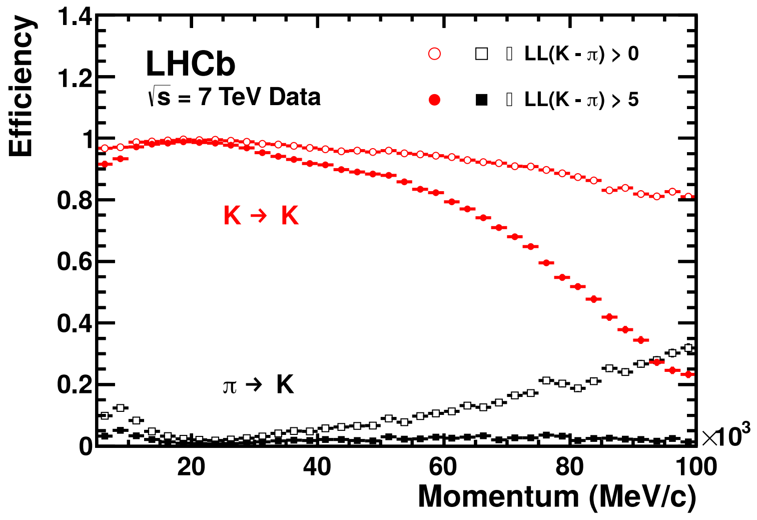


Figure 4.10. Kaon identification efficiency and pion misidentification rate measured on data as a function of track momentum. The results obtained by using two different DLL requirements are shown with open and filled markers. Reproduced from [168].

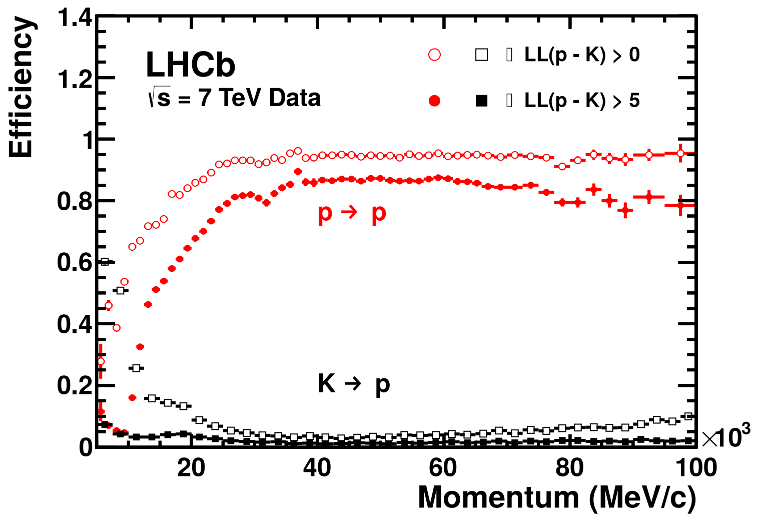


Figure 4.11. Proton identification efficiency and kaon misidentification rate measured on data as a function of track momentum. The results obtained by using two different DLL requirements are shown with open and filled markers. Reproduced from [168].

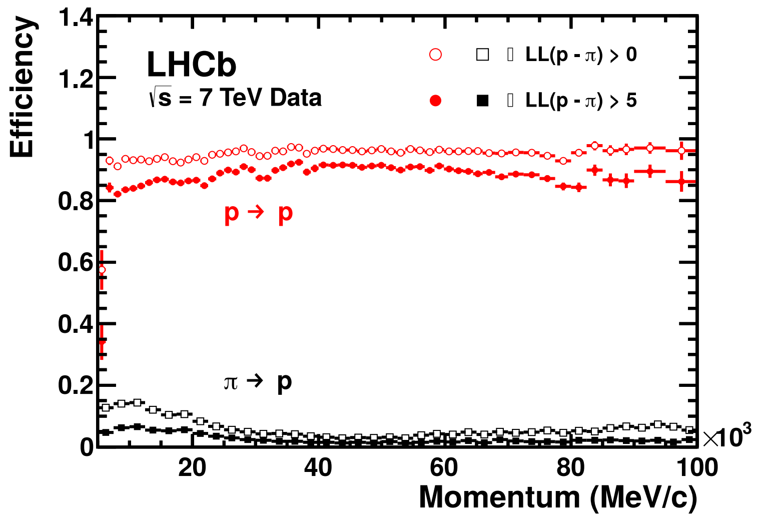


Figure 4.12. Proton identification efficiency and pion misidentification rate measured on data as a function of track momentum. The results obtained by using two different DLL requirements are shown with open and filled markers. Reproduced from [168].

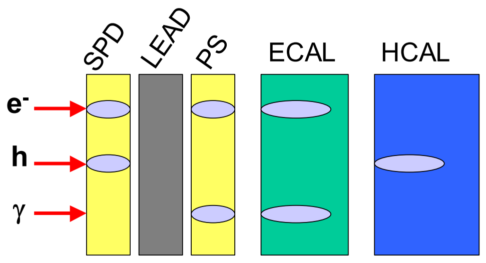


Figure 4.13. Signal deposited on the different parts of the calorimeter by an electron, a hadron, and a photon. Reproduced from [195].

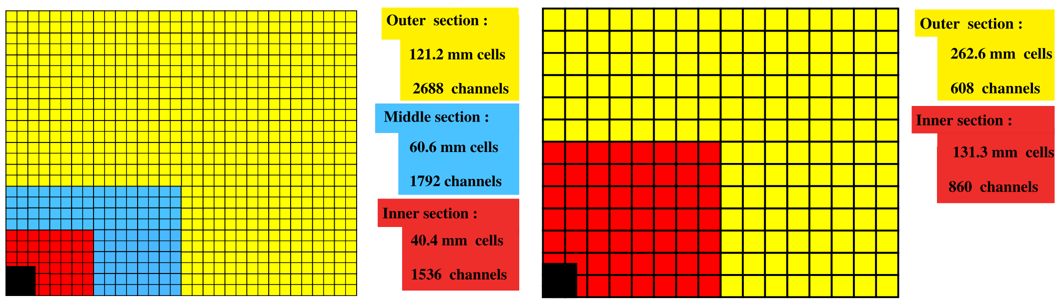


Figure 4.14. Lateral segmentation of the SPD/PS and ECAL (left) and the HCAL (right). One quadrant of the detector front face is shown. In the left figure, the cell dimensions are given for the ECAL. Reproduced from [11].

subdivided into inner, middle and outer sections with appropriate cell size [11]; this is depicted in figure 4.14.

The SPD and PS are two almost identical planes of scintillator pads separated by a lead 12 mm thick sheet [195]. The sensitive area of the detector is 7.6 m wide and 6.2 m high. Due to the projectivity requirements, all dimensions of the SPD plane are smaller than those of the PS by 0.45% [11]. The e/π separation performance of the PS prototype was measured in the X7 test beam. Pion rejection factors of 99.6%, 99.6% and 99.7% with electron retention factors of 91%, 92% and 97% are achieved for 10, 20 and 50 GeV/c particle momentum, respectively [11].

The ECAL uses shashlik (Russian: шашлык) scheme. It consists of layers of scintillator and lead with readout by plastic WLS fibres. The design has the advantages of fast time response, acceptable radiation resistance and reliability, at the cost of modest energy resolution [11]. The energy resolution is

$$\frac{\sigma_E}{E} = \frac{10\%}{\sqrt{E[\text{GeV}]}} \bigoplus 1\%. \tag{4.4}$$

It allows for B mass resolution of 65 MeV/c^2 for the $B \to K^\star\gamma$ penguin decay with a high-E_T photon. For the $B \to \rho\pi$ decay the π_0 mass resolution around 8 MeV/c^2 which translates into 75 MeV/c^2 B mass resolution [11].

The HCAL is a sampling device made from an iron absorber and scintillating tiles. Unlike ECAL, the HCAL scintillating tiles run parallel to the beam axis. This can be used to obtain a better angular resolution. The trigger requirements on the HCAL resolution do not impose a stringent hadronic shower containment condition. Its thickness is therefore set to 5.6 interaction lengths due to space limitations [11].

The observables provided by the calorimeters are listed in B.1.

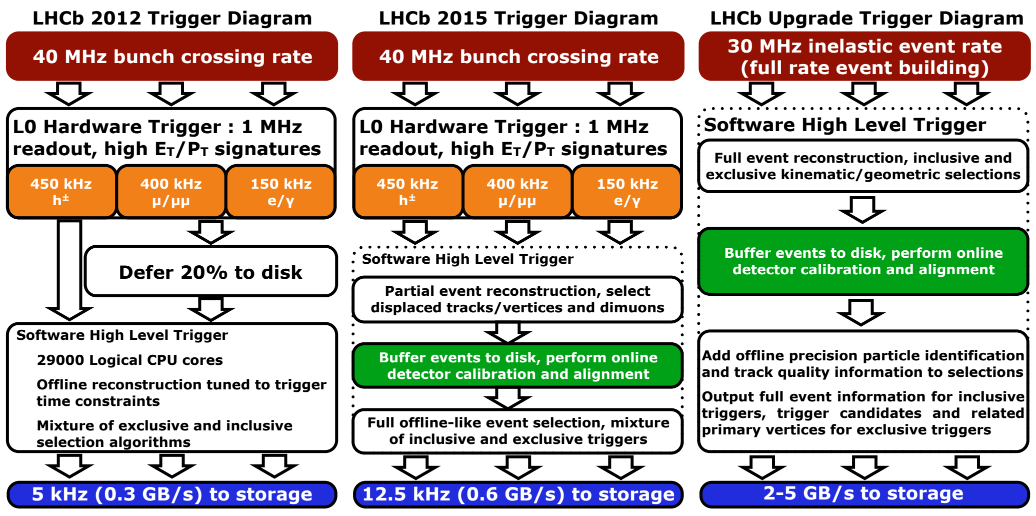
4.3 LHCb Data Processing

This section describes the process by which the raw LHCb data becomes published articles. The section is primarily based on references [196, 11]. Unless otherwise noted, the state for Run 2 is described.

LHCb detector observes hadronic collisions at high energies, thus events in it have a high level of background present. For LHC Run 1 and Run 2 the luminosity was levelled so that, on average, each event contained a single proton-proton collision. For Run 3, the luminosity will be increased fivefold, making event reconstruction even more challenging. To deal with the data flow under budgetary constraints, a sophisticated trigger system is used. A broad overview of it is presented in figure 4.15, and a more detailed in the subsections that follow.

4.3.1 Hardware Trigger (L0)

During normal operations, the LHCb bunch crossing frequency is 40 MHz. The maximum rate, at which all the LHCb subdetectors can be read out is 1 MHz. It is imposed by the bandwidth and frequency of the front-end electronics. The first level of trigger, L0, is designed to bring the event rate down to 1MHz. The trigger


LHCb 2012 Trigger Diagram
40 MHz bunch crossing rate
L0 Hardware Trigger : 1 MHz readout, high ET/PT signatures
450 kHz h±
400 kHz μ/μμ
150 kHz e/γ
Defer 20% to disk
Software High Level Trigger
29000 Logical CPU cores
Offline reconstruction tuned to trigger time constraints
Mixture of exclusive and inclusive selection algorithms
5 kHz (0.3 GB/s) to storage
LHCb 2015 Trigger Diagram
40 MHz bunch crossing rate
L0 Hardware Trigger : 1 MHz readout, high ET/PT signatures
450 kHz h±
400 kHz μ/μμ
150 kHz e/γ
Software High Level Trigger
Partial event reconstruction, select displaced tracks/vertices and dimuons
Buffer events to disk, perform online detector calibration and alignment
Full offline-like event selection, mixture of inclusive and exclusive triggers
12.5 kHz (0.6 GB/s) to storage
LHCb Upgrade Trigger Diagram
30 MHz inelastic event rate (full rate event building)
Software High Level Trigger
Full event reconstruction, inclusive and exclusive kinematic/geometric selections
Buffer events to disk, perform online detector calibration and alignment
Add offline precision particle identification and track quality information to selections
Output full event information for inclusive triggers, trigger candidates and related primary vertices for exclusive triggers
2-5 GB/s to storage

(a) Run 1 trigger architecture **(b)** Run 2 trigger architecture **(c)** Planned Run 3 trigger architecture – a single fully-software layer of processing.

Figure 4.15. Schematic representations of the LHCb trigger. Run 2 setup is described through the section unless otherwise noted, Run 1 in subsection 4.3.4, Run 3 in subsection 4.3.5. Reproduced from [197].

is implemented using field-programmable gate arrays (FPGA) and operates in real time, the latency does not depend on the detector occupancy.

The trigger overview is presented in figure 4.16. The trigger combines the information from the pile-up system in VELO, the calorimeters and the muon system. The main idea is to look for muons with large transverse momentum (p_T) and particles with high transverse energy (E_T), as those are commonly produced in B-meson decays. The L0 trigger performance is shown on figures 4.17, 4.18.

The pile-up system aids in the determination of the luminosity. It consists of two silicon planes placed downstream of the VELO detector. By tracing tracks through the hits in the planes, it can estimate the number of primary vertices in an event.

The Calorimeter trigger computes the transverse energy deposited in clusters of 2×2 cells [199]. Transverse energy is defined as follows:

$$E_T = \sum_{i=1}^{4} E_i \sin\theta_i, \tag{4.5}$$

where E_i is the energy deposited in the i-th cell, and θ_i is the angle between the z-axis and a virtual track connecting the cell centre to the centre of the interaction envelope. The trigger computes the transverse energy for three types of candidates [199]:

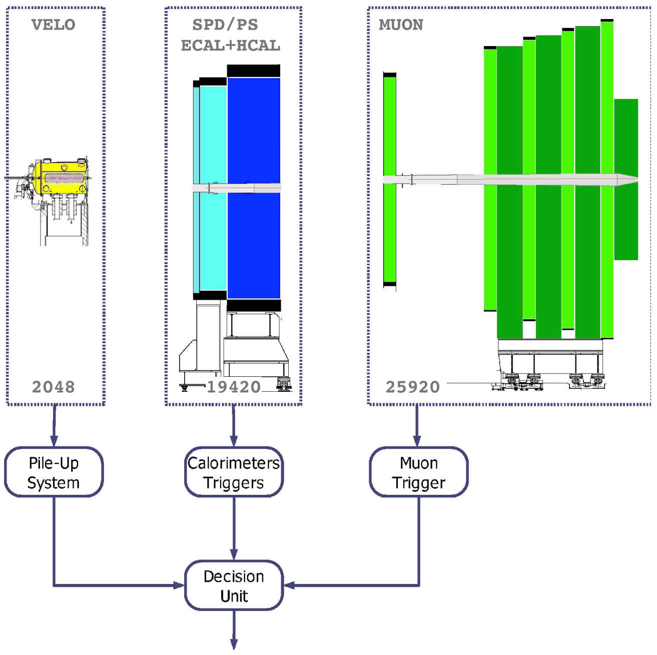


Figure 4.16. A scheme of the L0 hardware trigger. The numbers on the picture are the numbers of the readout channels. Reproduced from [11].

- `L0Hadron`: the HCAL cluster with the highest E_T. If there is a highest E_T ECAL cluster in front of the HCAL cluster, the output is the sum of their energies.
- `L0Photon`: the ECAL cluster with the highest E_T with 1 or 2 PS hits in front of it and no hits in the SPD cells corresponding to the PS cells. In the inner zone of the ECAL, an ECAL cluster with 3 or 4 PS cells hit in front of it is also accepted as a photon.
- `L0Electron`: the same requirements as for a photon candidate, with also at least one SPD cell hit in front of the PS cells.

The computed transverse energies are compared to a threshold. L0 retains events containing at least one candidate above the threshold.

The Muon trigger tries to identify the muon tracks with the largest and the second-largest transverse (z) momentum in each x-y quadrant. The trigger sets a single threshold either on the largest p_T among the eight candidates (`L0Muon` line), or a threshold on the largest and the second largest (`L0DiMuon` line) [199]. Each quadrant of the muon detector is connected to a separate L0 processor. There is no exchange of information between them. Muons traversing quadrant boundaries between the stations are not reconstructed in the trigger. An overview of the muon L0 trigger is presented in figure 4.19.

Removal of L0 for Run 3. According to reference [197], the L0 part is the most limiting part of the trigger; by removing this bottleneck and reading the full detector

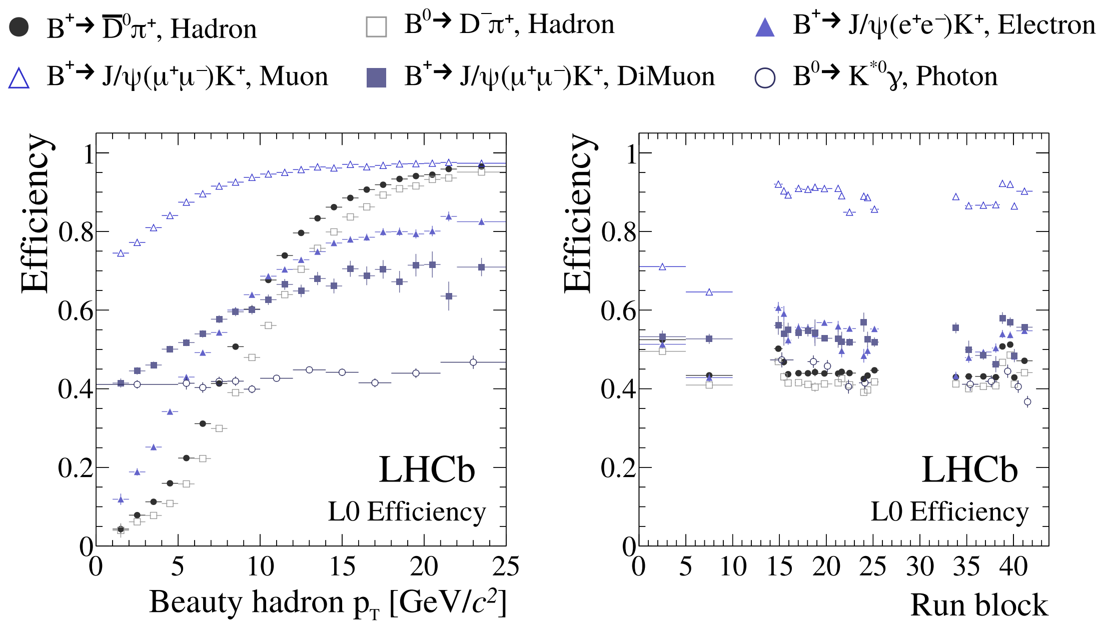


Figure 4.17. Efficiencies of the L0 trigger in Run 2 data for b-hadron decays. The left plot shows the efficiency as a function of the hadron transverse momentum (p_T). The right plot shows the evolution of the efficiency as a function of the different trigger configurations used during data taking. Reproduced from [198]

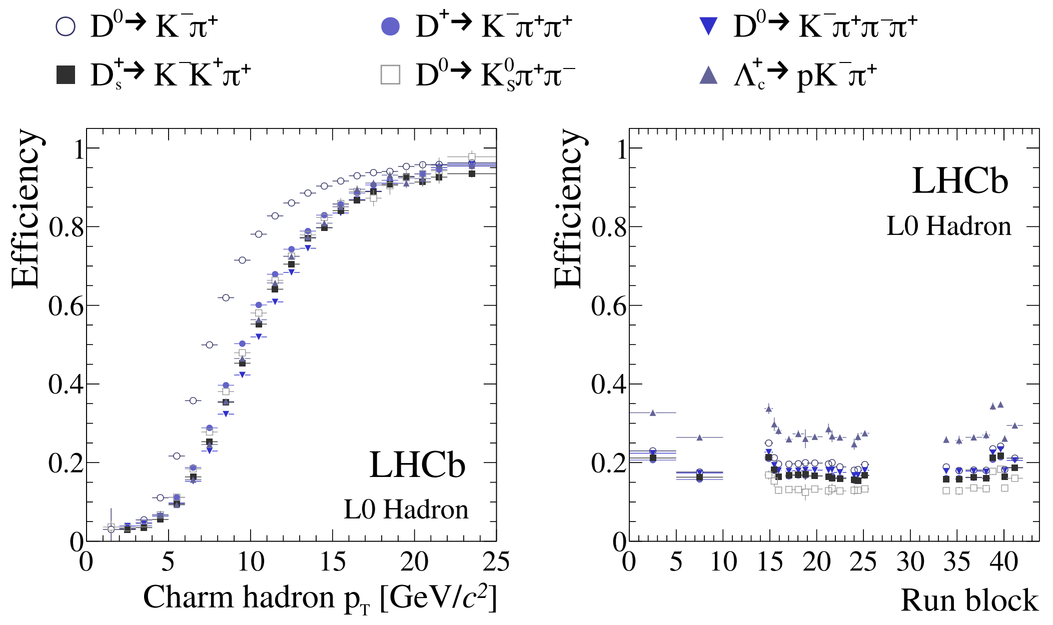


Figure 4.18. Efficiencies of the L0 trigger in Run 2 data for c-hadron decays. The left plot shows the efficiency as a function of the hadron transverse momentum (p_T). The right plot shows the evolution of the efficiency as a function of the different trigger configurations used during data taking. Reproduced from [198].

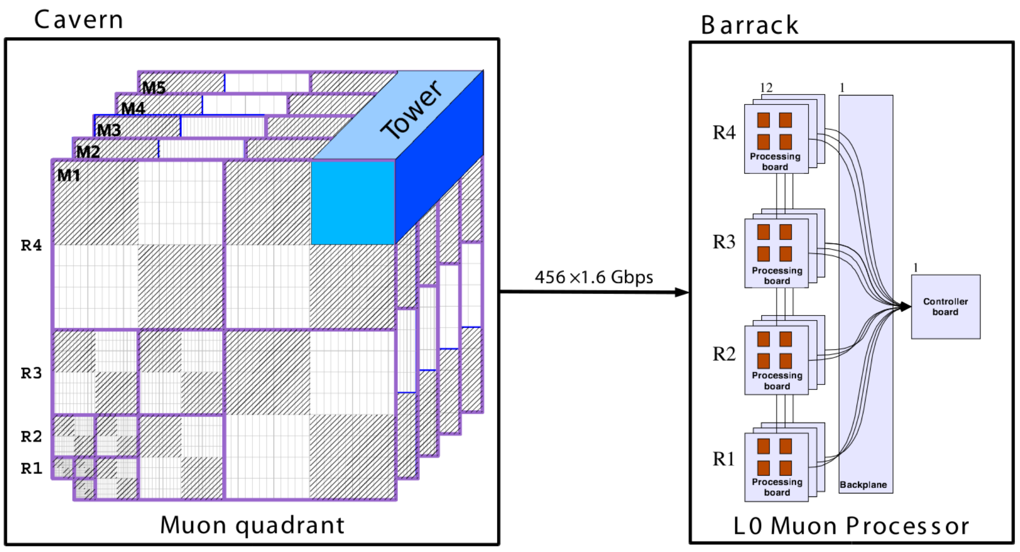


Figure 4.19. A schematic overview of the L0 muon trigger. Reproduced from [11].

at 30 MHz, physics performance will be significantly improved. Reference [200] estimated that the removal of the hardware trigger would increase by a factor two the trigger efficiency for most of the physics programme.

In the context of the thesis, the move to the full software trigger provides flexibility, that would allow benefiting even more from the developed advanced algorithms for muon identification.

4.3.2 Software Trigger (HLT)

Software trigger is designed to reduce the event rate to a rate that can be sent to permanent offline storage. It increased with the computing resources available, from 3.5 kHz at the beginning of Run 1 to 12.5 kHz at the end of Run 2.

The software trigger runs on the Event Filter Farm (EFF). The EFF consists of approximately 1700 nodes, 800 of which were added for Run 2, with 27000 physical cores. The EFF can accommodate around 50000 single-threaded processes using the hyper-threading technology [198].

The trigger consists of two stages, that are run sequentially: HLT1 and HLT2. The first stage performs a fast partial event reconstruction and rough selections based on it. The second stage uses all available information and does a full reconstruction.

4.3.2.1 HLT1

HLT1 reconstructs long tracks (ones that have hits in all the tracking subsystems) and the precise position of the primary vertex (PV). And it runs a fast version of the muon identification algorithm.

The tracking sequence is outlined in figure 4.21. First, the hits in the VELO subdetector are searched for combinations that match straight lines loosely pointing towards the beamline. Then, tracks are extrapolated into the TT. Tracks falling outside of the TT acceptance are passed. For tracks inside the TT acceptance, at least three matching TT hits are required. TT is located in in the fringe field of the magnet. This allows making a rough estimate of momentum with the resolution of around 20%. The tracks with low transverse momentum (p_T) are then rejected. Then the tracks are matched with hits in the IT and OT. The tracks are fitted with a Kalman filter.

In 2016, an artificial neural network was implemented in HLT1 to filter out fake tracks [159, 201]. During Run I it was available only offline, then a speed-up by a factor of ~ 90 allowed for its inclusion in HLT2 and, subsequently HLT1. The main improvements were

- Switching from $\tanh x$ activation to $x/\sqrt{1+x^2}$,
- Using input variables that are already available and do not require additional computations
- Manual code optimisation.

The neural network has a fully-connected architecture with 21 input variables and a single hidden layer with 26 neurons. Its input variables are the overall χ^2 of the Kalman filter, the χ^2 values of the fits for the different track segments, the numbers of hits in the different tracking detectors, and the p_T of the track. The model was trained using the Caffe library [202]. It was trained on simulated data. The performance is shown in figure 4.20.

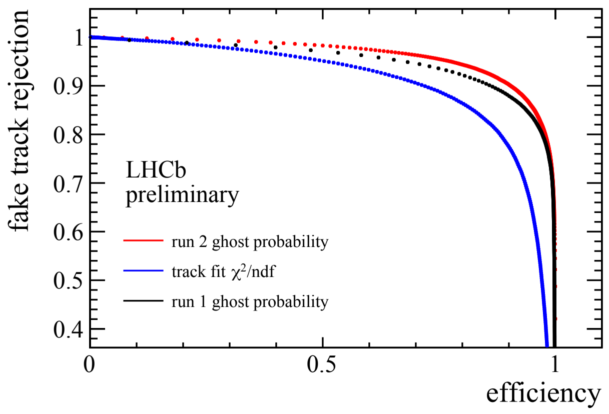


Figure 4.20. ROC curves for fake track rejection using for Run II. “Ghost probability” refers to the output of the neural network. Reproduced from [159].

The tracks are used to select b and $c-$ hadron decays, the performance is presented in figure 4.22.

For muon identification, HLT1 implements an algorithm called `IsMuon`. It extrapolates the tracks into the muon subsystem and searches for the hits around the track extrapolation inside a momentum-dependent Field of Interest (FOI). The FOI is determined from the analytical approximation of multiple scattering [203]. If and only if there are hits in enough stations, `IsMuon` considers the particle a muon [19].

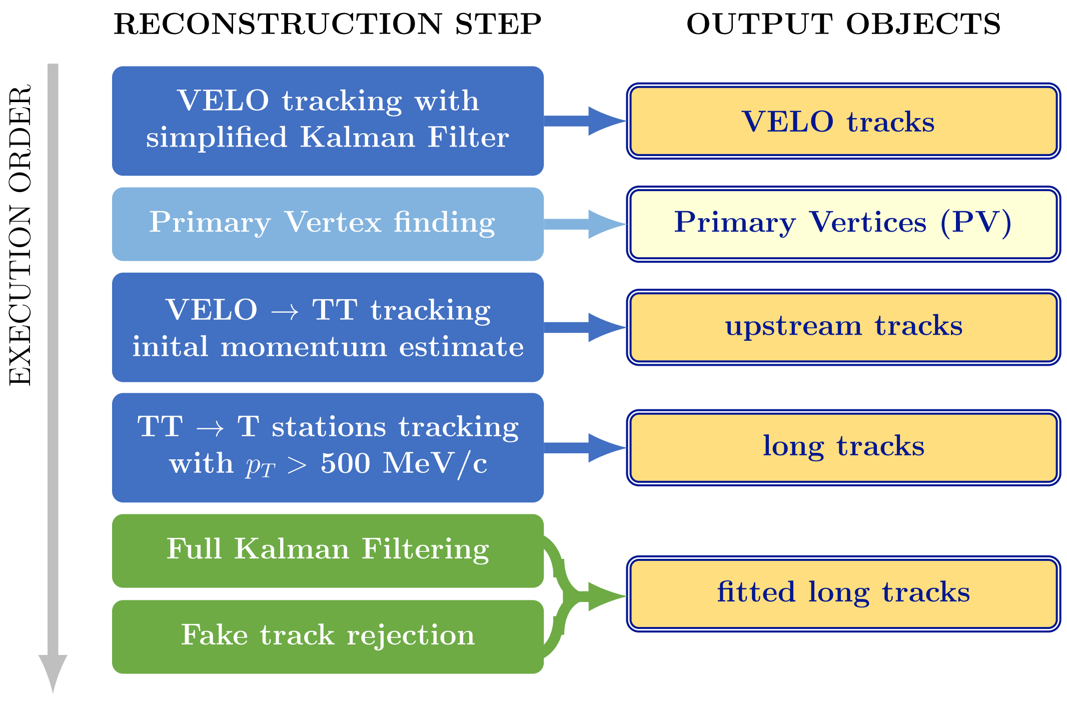


Figure 4.21. A sketch of the HLT1 track and vertex reconstruction. Reproduced from [198].

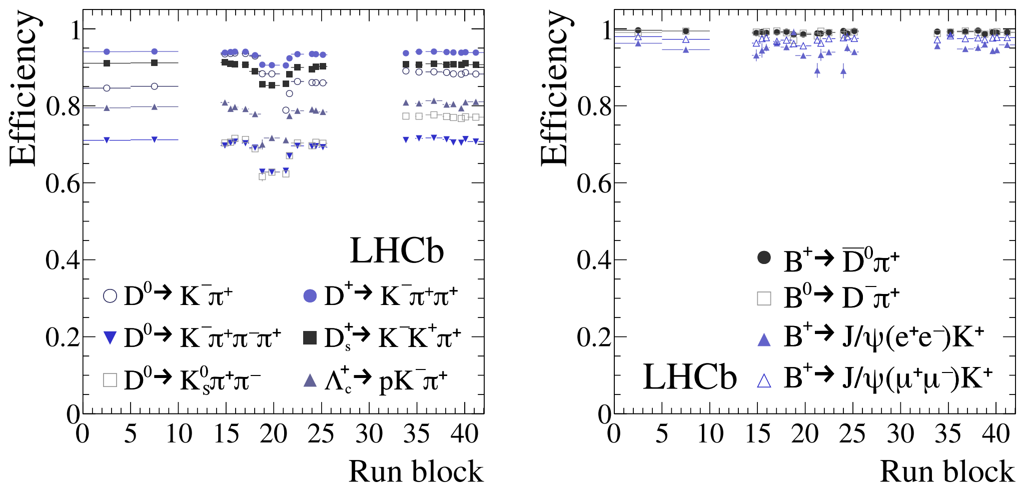


Figure 4.22. The trigger efficiency as a function of the trigger configurations used during the data taking for *c*-hadrons (left) and *b*-hadrons (right). The three clusters separated by the vertical gaps correspond to the three years of data taking (2015 – 2017). Reproduced from [198].

The stations requirement is a function of track momentum and is presented in table 4.1.

Momentum range	Muon stations
3 GeV/c $< p <$ 6 GeV/c	M2 and M3
6 GeV/c $< p <$ 10 GeV/c	M2 and M3 and (M4 or M5)
$p >$ 10 GeV/c	M2 and M3 and M4 and M5

Table 4.1. Muon stations requirements of `IsMuon`

There is a variant of `IsMuon`, called IsMuonLoose. It imposes looser stations requirement that is presented in table 4.2.

Momentum range	Muon stations
3 GeV/c $< p <$ 6 GeV/c	at least one hit in at least two stations among M2, M3, M4
$p >$ 6 GeV/c	at least one hit in at least three stations among M2, M3, M4, M5

Table 4.2. Muon stations requirements of `IsMuonLoose`

The procedure that is used in HLT1 for muon identification is presented in figure 4.23. For tracks with $p_T >$ 500 MeV/c, the reconstruction described in the previous paragraph is used. For tracks with $p_T <$ 500 MeV/c, the full track reconstruction is not run due to timing constraints. The TT track segments are extrapolated directly to the MUON stations. The algorithm searches for hits around the extrapolations. The fields of interest are larger than those used for otherwise-reconstructed long tracks. If hits are found in the muon system, the VELO-TT segment is extrapolated through the magnetic field using the momentum estimate and matched to hits not already used in the HLT1 long-track reconstruction. This allows to identify muon tracks with p_T starting from 80 MeV/c. Performance of HLT1 muon identification is presented in figures 4.24 and 4.25.

4.3.2.2 HLT2

The second stage of the software trigger runs full event reconstruction. It consists of three main steps: reconstruction of charged particle tracks, reconstruction of neutral particles and particle identification (PID).

An overview of the HLT2 tracking procedure is presented in figure 4.26. The first step is a repeat of the HLT1 track reconstruction. The second step is the reconstruction of the lower-momentum tracks, that were previously discarded due to the kinematic thresholds. Then, the tracks that do not originate from inside the VELO are reconstructed from the T-station segments. A machine learning model is used to identify and reject the ghost T-tracks [206]. It was trained on a simulated sample containing signal $B \to J/\psi K_s^0$ events. The model was trained using the XGBoost [110] library. It uses the following features:

- χ^2/ndf of the T track,
- Momentum,

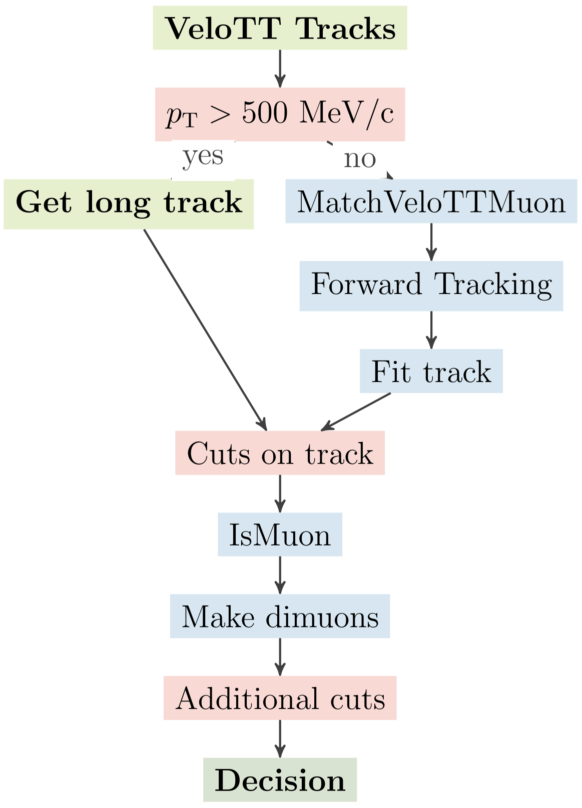


Figure 4.23. HLT1 muon identification algorithm. Reproduced from [204].

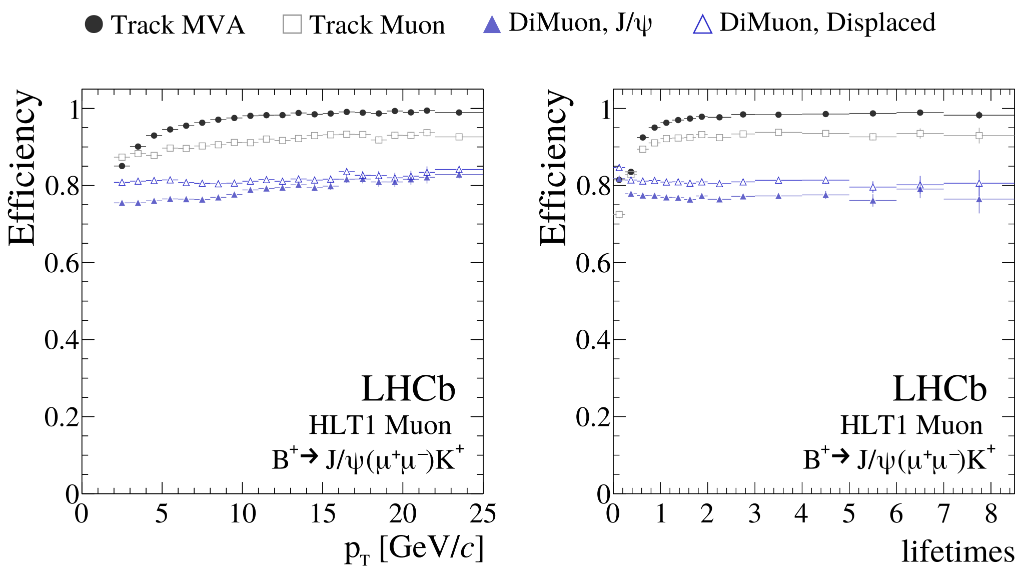


Figure 4.24. The efficiency of the HLT1 muon trigger lines as a function of the b-hadron p_T (left) and the average B^+ decay time (right). The times are plotted in the units of the B^+ lifetime in its rest frame. The efficiency of the inclusive single-track HLT1 trigger is plotted for reference. Reproduced from [198].

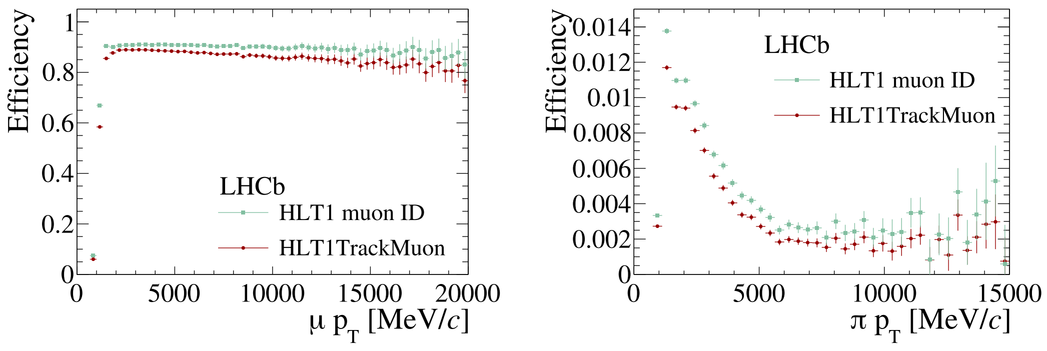


Figure 4.25. HLT1 muon identification efficiency for muons from $J/\psi \to \mu^+\mu^-$ decays (left) and pions from $D^0 \to K^-\pi^+$ decays (right). Green circles show only the identification efficiency (HLT1 Muon ID). The red squares show the efficiency of the additional trigger line (named HLT1TrackMuon), that has reduced (transverse) momentum requirements, and relaxed track quality cuts [205]. Reproduced from [198].

- Transverse momentum,
- The number of hits contributing to a given T track,
- The $|x|$ and $|y|$ coordinates absolute values of the T track's first state,
- The distance from the beamline to the track's first state,
- The absolute values of the slope of the track in x–z and y–z planes,
- Pseudorapidity.

The T-stations segments are extrapolated backwards through the magnetic field and combined with hits in the TT. The track candidates are again filtered with the help of machine learning [206]. A fully connected neural network is utilised. It has a single hidden layer with 15 nodes and is implemented with the TMVA framework [207]. The network uses the ReLu activation function for the hidden layer and the sigmoid activation for the output layer. The features are the following:

- $\ln(\chi^2/\mathrm{ndf})$ of the track candidate,
- Momentum and transverse momentum logarithms,
- The difference between the momentum estimate of the T track and the downstream track candidate,
- Logarithm of the x displacement with respect to the point in the magnet after the fit,
- Logarithm of the distance of the hits with respect to the initial track estimate,
- Logarithm of the absolute x and y positions at $z = 0$,
- The number of layers in the TT with hits associated with the track.

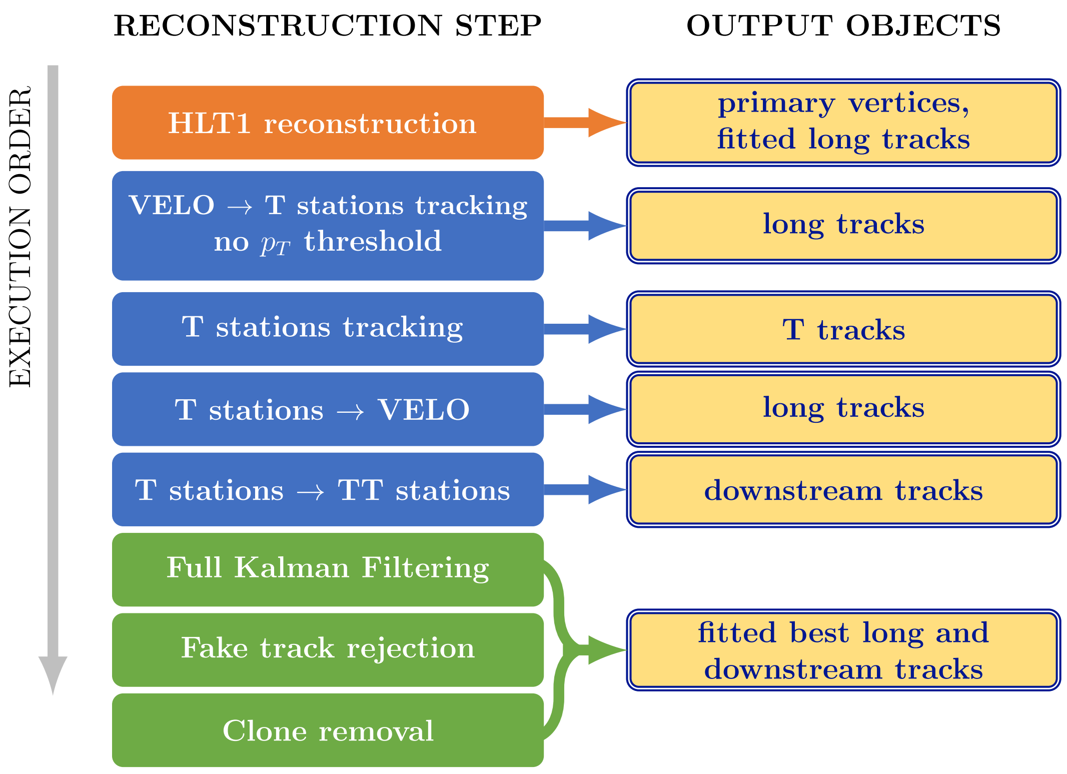


Figure 4.26. HLT2 track reconstruction procedure. Reproduced from [198].

The network was train on simulated $D^{\star +} \to D^0 \left(\to K^0_S K^+ K^-\right) \pi^+$ decays.

The next stage is the filtering of fake tracks. First, all tracks are fitted using the same Kalman filter. Second, the same neural network as in HLT1, is applied.

The final operation of the track reconstruction is the removal of the so-called clones. Tracks are defined as clones if they share enough hits in each subdetector. The track with more hits in total is kept, and the other is discarded.

For muon identification, the same algorithm as in HLT1 is used, `IsMuon`. It is described in the subsection 4.3.2.1. The algorithm is applied to all the tracks available after the HLT2 reconstruction.

The RICH detectors are the primary tool for discrimination between deuterons, kaons, pions, and protons. For each mass hypotheses, the photon yields expected Cherenkov angles, and estimates of the per-track Cherenkov angle resolution are computed. The reconstruction algorithm considers all tracks and all photons in both RICH1 and RICH2 simultaneously. For each mass hypothesis, the likelihood is calculated. A more detailed description of the RICH PID algorithm is in the subsection 4.2.2.1.

The calorimeters perform the reconstruction of electromagnetic particles (photons, electrons, and π^0 mesons). A cellular automaton algorithm builds clusters from the energy deposits in the different calorimeter subsystems [208]. It works as following. First, it finds the local maximums – the cells with larger energy deposit, than their neighbours. Those cells are tagged. The second step is to process the rest of the cells according to the following rules:

- A tagged cell does not change;
- A non-tagged cell is processed:
  - If none of its neighbours is tagged, no action;
  - If all the neighbours that are tagged are tagged with one unique tag, assign the same tag;
  - If neighbours are tagged with different tags, the cell is identified as being shared by several clusters, and the tag numbers are stored.

The second step is repeated for all cells until no change occurs. The clusters are then associated with the reconstructed tracks. Neutral particles are identified as clusters that do not correspond to any reconstructed charged track. Electrons are identified by combining the information from the isolation of clusters in the ECAL, the presence of clusters in the PS, the energy deposited in the HCAL, and the position of possible Bremsstrahlung photons. High-E_T π^0 mesons and photons are indistinguishable at the trigger level.

Events that pass the HLT2 selection are written to disk and subjected to offline processing.

4.3.3 Offline Data Processing

Offline, the events are organised into the so-called streams. Each stream has a number of selections associated with it and retains the events that pass them. Depending on the stream, a different portion of event information is retained. For the so-called Turbo stream (described in subsection 4.3.4), only the reconstructed signal candidates are retained. The events in the Turbo stream are ready to be analysed. For the rest of the events, the full raw event is stored, and there is an additional processing step, called stripping. The events are reconstructed once again offline profiting of more relaxed time constraints, and tight selections, called stripping lines, are applied. The events are then grouped into streams and made available for user analysis.

LHCb software allows only sequential access to the events. A typical user job only requires 1–2% of the whole data, but has to read the entire stream – and thus spends time on the unnecessary disk reads [209]. Streams are a way to reduce this overhead so that a job only has to read the events in a single stream, not all of them. This approach introduces information duplication – if multiple streams select an event, it is stored multiple times. In reference [209] I propose a method to find the optimal stream composition for different speed/space trade-off values.

4.3.4 Historical Perspective: Run 1

All the results in the thesis have been obtained using the Run 2 data and simulation. This subsection describes the difference between Run 1 (2010–2012) and Run 2 data collection.

Run 1 had the collisions at centre-of-mass energies of 7 and 8 TeV, which was increased to 13 TeV for Run 2. Higher energy means higher production cross-sections. This means more interesting processes per bunch crossing, but also higher detector occupancy and more difficult reconstruction.

L0 trigger functioned similarly in Run 1 and Run 2, reducing the event rate to 1 MHz at which the LHCb detector can be read out. The software trigger output rate increased 3.5 kHz at the beginning of Run 1 to 12.5 kHz at the end of Run 2.

During Run 1, 20% of the data, that passed the hardware trigger, was buffered to disk, and the rest passed to the software trigger for near-real-time processing. The buffered data was processed during periods when there were no collisions. For Run 2, HLT1 and HLT2 were explicitly separated. All events passing L0 were immediately processed by HLT1 and then buffered to disk to be processed by HLT2 as soon as all the calibration and alignment constants were ready. The difference in the trigger schemes is illustrated in figure 4.15.

The critical difference between Run 1 and Run 2 is the online detector calibration and alignment. It allowed the online reconstruction in HLT2 to reach identical quality to offline reconstruction. This paved the way for the Turbo stream [210]. For Run 1, the events, that had passed the trigger, were fully written to disk and subjected to offline reconstruction. Turbo stream is an approach introduced during Run 2, where only the information about the reconstructed signal candidates is written to disk, not the full event data. This allows reducing the event size by an order of magnitude. Turbo stream is expected to become the dominant data processing model for Run 3.

4.3.5 Upgrade Towards Run 3

Run 3 will see a five-fold increase in luminosity, with a corresponding increase of the signal rate (good) and overall occupancy and reconstruction difficulty (bad). There will be a number of hardware and software changes to accommodate [211].

4.3.5.1 Detector Hardware

The major change is the readout electronics, which is upgraded for all the subdetectors to provide detector readout at 40 MHz to manage the 30 MHz bunch crossing rate expected in LHCb.

Tracking. VELO will move from silicon microstrips technology to pixel technology which will allow to avoid an explosion of ghost crossings with the increase in luminosity. The detector will be closer to the beam axis, 5.1 mm compared to 8.4 mm for Run 2. Overall, the impact parameter resolution will be improved by a factor of around 40%. The tracking detector before the magnet (the Upstream Tracker, UT) will be composed of new, high-granularity silicon micro-strip planes with improved coverage of the LHCb acceptance. Behind the magnet, a Scintillating Fibre Tracker will be built, which is composed of 2.5 m long fibres read out by silicon photomultipliers at the edge of the acceptance [212].

RICH1 will be modified to handle the much higher particle occupancy of the upgrade conditions. In particular, the focal length of the mirrors will be increased by a factor $\sqrt{2}$ thus halving the occupancy [211].

Muon subsystem will see M1 removed.

Calorimeters will see little change, apart from the electronics.

4.3.5.2 Software

The evolution of the trigger is presented in figure 4.15. Run 1 featured a monolithic software trigger. Full information from the events passing the trigger was stored. The stored events were additionally reconstructed and filtered offline. This approach provided maximal flexibility and allowed to take advantage of the evolution of the reconstruction software by reprocessing older data with newer versions of the reconstruction hardware.

Run 2 saw an explicit separation of the software trigger into HLT1 and HLT2 and the alignment of offline and online reconstruction. To take advantage of this, Turbo stream was introduced, which persisted only the signal parts of events and made them available for user analysis without a need for further processing.

In Run 3, the Turbo approach is further refined into the so-called real-time analysis[1]. The format will allow selective persistence – saving a portion of an event, striking a balance between disk usage and analysis flexibility. It is illustrated in figure 4.27.

It is also planned to forbid individual users from sending jobs in favour of centrally-managed productions [200] to reduce the load on computing resources.

4.3.6 HLT1 on GPU (Allen)

Graphics processing units (GPUs) are designed for massively parallel data processing, with up to 5000 cores on a 2019 chip [98]. Events are processed in the HLT1 independently from each other. With the average event size of 100 kB, several thousands of events can fit into a GPU memory. There is also a possibility for intra-event parallelism. The LHCb collaboration has developed a prototype application for GPU-based HLT1, called Allen [214, 215, 216].

Allen processes several thousands of events in parallel. Typically, the raw input is segmented by the readout unit, for example, a module of the vertex detector, and decoding is parallelised. During the tracking, many combinations of hits are tested in parallel; track fit is applied to every track in parallel; extrapolating tracks from one subdetector to the next is executed in parallel. Finally, combinations of tracks are built when finding vertices, and those are treated in parallel.

As a part of HLT1, Allen does muon identification. It features the `IsMuon` algorithm described in subsection 4.3.2.1 and CatBoost trained for Run 2 configuration described in section 5.4. Sergei Popov, a student I supervised, wrote the code for predicting using CatBoost trees on CUDA kernels during his MSc studies.

The evaluation shows that Allen covers the majority of the LHCb physics programme with satisfactory performance in terms of track and vertex reconstruction, momentum resolution, and muon identification. It would require around 500 of GPUs to run the full HLT1 sequence, for RTX 2080 Ti, V100 or Quadro RTX 6000

[1]"real time" refers the interval between a collision occurring and the point at which the corresponding event must be either discarded forever or sent offline for permanent storage [213]

Nvidia cards. In May 2020 the LHCb Collaboration decided to use Allen for Run 3 trigger.

4.3.7 Calibration Samples

Given the computational expense [217] and the imperfect accuracy [22] of simulation, it is desirable to use data-driven methods for design and evaluation of data analysis algorithms. This section describes the approach developed by the LHCb collaboration for development and evaluation of particle identification. It primarily follows [218, 219].

The calibration samples are datasets, where the decay candidates have a kinematic structure that allows unambiguous identification of one of the daughters, without the use of any information about the PID response to the daughters track in question. There are such datasets for the five most common charged particle species that interact with the LHCb detector: protons, kaons, pions, muons and electrons [219]. The list of the decays is presented in table 4.3.

Species	Low momentum	High momentum
$e^\pm$	$B^+ \to (J/\psi \to \underline{e^+e^-})K^+$	
$\mu^\pm$	$B^+ \to (J/\psi \to \underline{\mu^+\mu^-})K^+$	$J/\psi \to \mu^+\mu^-$
$\pi^\pm$	$K^0_s \to \pi^+\pi^-$	$D^{\star+} \to (D^0 \to K^-\underline{\pi^+})\pi^+$
$K^\pm$	$D^+_s \to (\phi \to \underline{K^+K^-})\pi^+$	$D^{\star+} \to (D^0 \to \underline{K^-}\pi^+)\pi^+$
$p\bar{p}$	$\Lambda^0 \to p\pi^-$	$\Lambda^0 \to p\pi^-$; $\Lambda_c^+ \to pK^-\pi^+$

Table 4.3. Decay modes that are used to select calibration samples. In case of ambiguity, the used daughters are underlined.

The calibration sample selection is complicated by the fact that the hardware trigger (described in the subsection 4.3.1) sometimes uses the PID systems (namely, MUON and CALO) to make its decision which events to keep. To avoid the bias, the selection strategy of the calibration samples requires that either the trigger did not use the PID information, or used it only for particles that are not being considered for the calibration sample.

Some of the selections are implemented using the so-called tag-and-probe method [220]. For example, take the $J/\psi \to \mu^+\mu^-$ decay. First, a list of well-identified *tag* muons is compiled. Second, a list of *probe* particles with the opposite charges is made without using the PID information. Then, the *tag* and *probe* tracks are combined to form muon pairs with invariant-mass consistent with the J/ψ mass. Finally, the pairs are filtered based on the quality of the fit of the decay vertex.

The residual background that remains is subtracted with the sPlot method [21]. A fit to the invariant mass of the decaying particle is performed, resulting in probabilities of each track to be signal. In several cases, two-dimensional fits are performed to better account for the complex background. sPlot uses the fit-derived signal probabilities to assign each event a signed weight (sWeight). If a sample is weighted with the sWeights, the distribution of the variables that are independent of mass will be an unbiased estimate of the variables' distribution in a purely signal sample. Some of the sWeights are negative. This complicates using machine learning methods on such data – in more details, the problem and my solution are covered in chapter 6.

The distributions of the invariant mass and fit results for some of the calibration decays are shown in the figure 4.28.

To use the calibration samples to compute the PID efficiency, it is usually assumed that the PID variables can be parametrised by a small set of variables, such as the track momentum, pseudorapidity, and the total number of tracks in the event (track multiplicity). For computing the PID efficiency on a sample other than the calibration sample (reference sample), the calibration sample is weighted, so that the distribution of the parameterising variables matches the distribution in the reference sample. An estimate of the probability density ratios is required for the weighting:

$$w(\mathbf{x}) = \frac{p_{\text{reference}}(\mathbf{x})}{p_{\text{calibration}}(\mathbf{x})}, \tag{4.6}$$

where $\mathbf{x}$ is a vector of the parametrising variables. Typically, in LHCb, a low-dimensional (2 to 3) parametrisation is used, and the probability density ratio is estimated by histograms or kernel density estimation[2] [221]. For the work on muon identification described in chapter 5, I used a calibrated classifier, as it allows for a more fine-grained probability estimate. The reweighted sample can be used to compute the efficiency of a PID selection:

$$\begin{aligned}\bar{\epsilon} &= \int d\mathbf{x}\epsilon(\mathbf{x})p_{\text{reference}}(\mathbf{x}) \\ &= \int d\mathbf{x}\epsilon(\mathbf{x})w(\mathbf{x})p_{\text{calibration}}(\mathbf{x}) \\ &\approx \frac{\sum_i w_i I_i}{\sum_i w_i},\end{aligned} \tag{4.7}$$

where $\epsilon(\mathbf{x})$ is the selection efficiency for the tracks with parameters $\mathbf{x}$; I_i is the indicator whether the i-th track passes the calibration requirements.

The calibration samples play an important role in the LHCb experiment. They are used to measure the PID performance, to correct the simulated samples, and to monitor the detector performance during the data-taking [218]. In the context of the thesis, calibration samples are used for global PID algorithms evaluation in chapter 7; for muon identification model training and evaluation in chapter 5. The parametric fast simulation approach in chapter 8 is a logical extension of the PID calibration and uses the calibration samples for data-driven training. The technique proposed in chapter 6 was developed with the LHCb PID samples as the primary use case.

4.3.8 Machine Learning at LHCb

Machine learning is used through the LHCb data processing. Machine learning usage for tracking is described in subsection 4.3.2.1; for event selection in HLT2 in

[2]Kernel density estimation works by modelling the probability density function as a sum of simple functions centred on the data points, called kernels. Common choices for kernels are Gaussians or parabolas.

subsection 3.3.1; for charged particles identification in chapters 7 and 5; for neutral particles identification in subsection 3.3.1; for simulation in chapter 8; for detector monitoring in subsection 3.3.3.

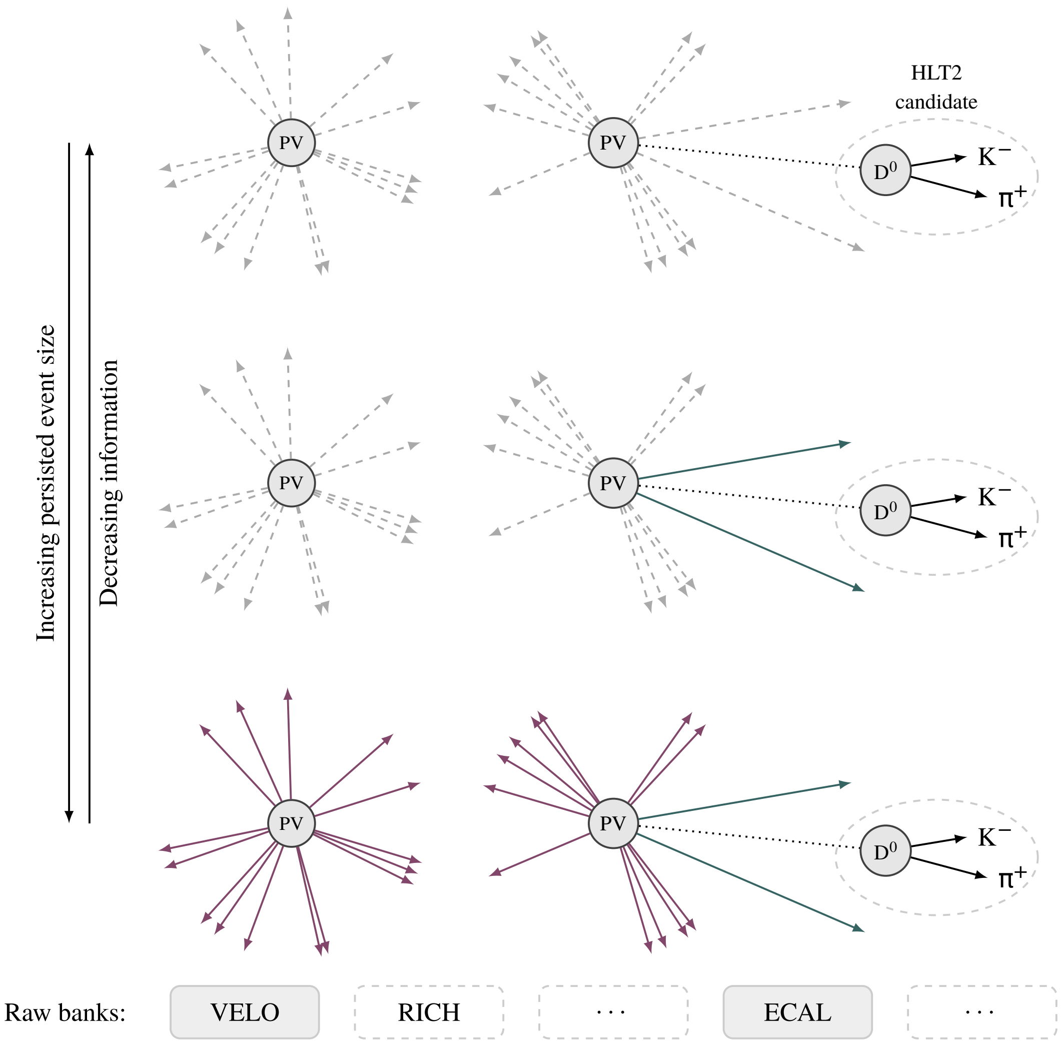


Figure 4.27. A cartoon of the same reconstructed event with varying levels of object persistence: Turbo (top); selective persistence (middle); and complete reconstruction persistence (bottom). Solid objects are those persisted in each case. A trigger selection may also ask for one or more sub-detector raw banks to also be stored, shown as solid rectangles. Reproduced from [213].

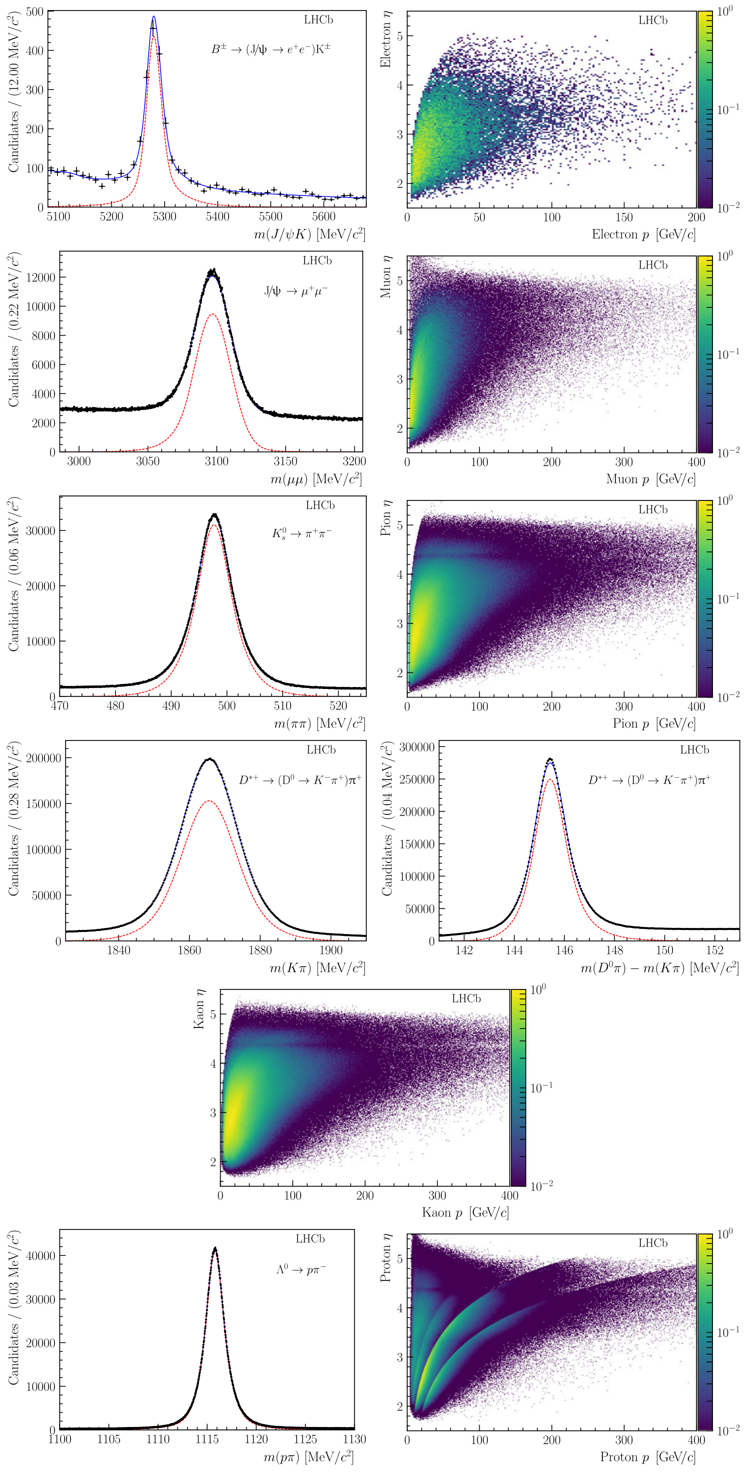


Figure 4.28. On the left, mass distributions of the decaying particles with the results of the fit superimposed; signal contributions are shown by the red dashed curves, and the total fit functions including background contributions are shown by the blue solid curves. On the right, the background-subtracted distributions of the calibration samples for electrons, muons, pions, kaons and protons as a function of the track pseudorapidity (η) and momentum (p). The colour scale units are arbitrary. Reproduced from [218].

Chapter 5

Muon Identification

Muons are present in the final states of many decays sensitive to new physics that are studied by the LHCb experiment [19, 5, 20]. Efficient and precise muon identification is of paramount importance for the LHCb. There are two main sources of misidentification. The first is combinatorial – when unrelated hits align by chance with the track. The second is decays in flight for pions and kaons.

My contribution towards it is a machine learning model based on gradient boosting. For Run II, it shows the best discrimination quality among all proposed algorithms. It is also 3 times faster than the other considered machine learning approach based on TMVA [207]. I have integrated the model into the LHCb software, and its output is available for analysis of the data taken in 2018. I've also investigated the feasibility of the machine learning approach for high-occupancy data, and show that it outperforms the other considered alternative (χ^2_{CORR}). The work has been submitted to proceedings of the ACAT 2019 conference and is included in the LHCb (as of April 2020, internal) note [222]. A journal paper is under review. I had also prepared muon identification as a problem for a data analysis competition [223].

The LHCb muon detector is described in section 5.1. The fast muon identification algorithms that were run online in the trigger during the LHC Run 2 are described in subsections 4.3.1 (L0, hardware trigger), 4.3.2.1 (`IsMuon`, runs in HLT1, the first level of the software trigger). This chapter takes on from there and continues with the HLT2 and offline selections that were used for Run 2 is section 5.2). The advancement of those methods is presented in section 5.3. My contribution to Run II is presented in section 5.4 and towards Run III in section 5.5. In section 5.7 we discuss a data science competition organised on the muon identification data.

5.1 Muon Detector

The main idea behind the muon identification is to utilise the unique penetration power of muons. A charged particle that was able to pass through the calorimeters and the iron shields is highly likely a muon. The schematic view of the LHCb Muon subdetector is presented in figure 5.1. It consists of 5 sensitive planes (M1-M5) placed perpendicular to the beam axis. The muon detector is placed downstream of the rest of the detector, except for M1, which is in front of the calorimeters. Stations M2 to M5 are placed downstream the calorimeters and are interleaved with 80-cm thick iron absorbers. The minimum momentum of a muon to cross the five stations is approximately 6 GeV/c [11].

The stations are divided into 4 regions with increasing distance from the beam axis. The regions' size is chosen, so that particle flux is roughly the same over all the regions. All stations, except the innermost region (R1) of M1, are equipped with multi-wire proportional chambers (MWPCs). M1R1 has to face the highest particle flux and thus is equipped with more resilient with gas electron multiplier (GEM) detectors.

For the detector to meet the physics requirements, the stations' efficiency must be at least 99% in a time window smaller than 25 ns [220]. The performance is achieved with an optimized charge-collection geometry and using a fast gas mixture; $Ar/CO_2/CF_4 \approx 40/55/5$ for the MWPCs and 45/15/40 for the GEM chambers. An overview of MWPC is presented in figure 5.2a and overview of GEM in figure 5.2b. In stations M2–M5 the MWPCs are composed of four gas gaps arranged in two sensitive layers with independent readout. In station M1 the chambers have only two gas gaps to minimise the material in front of the electromagnetic calorimeter. In the region M1R1, two superimposed GEM chambers connected in OR are used [11].

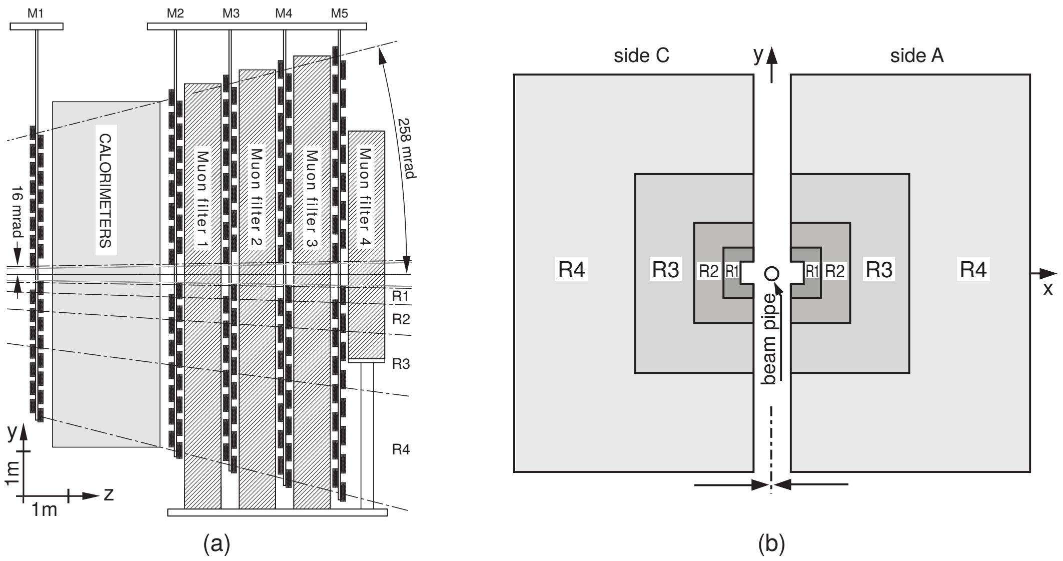
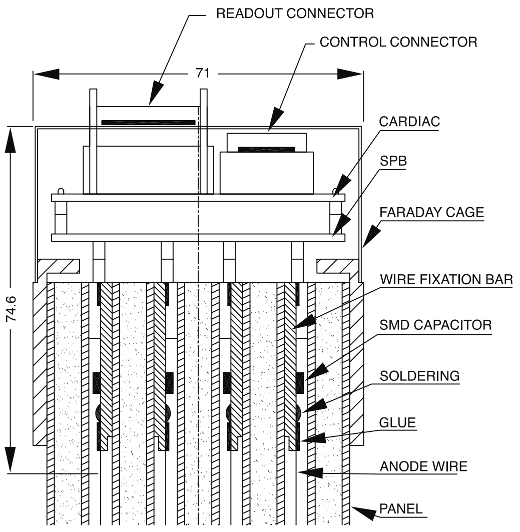


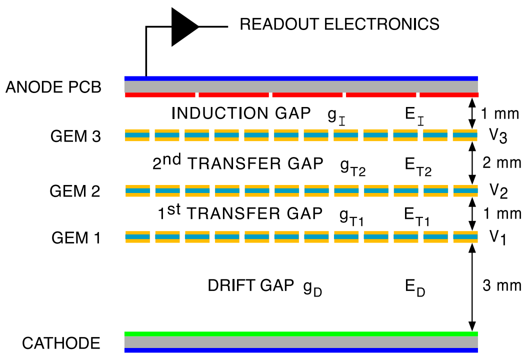
Figure 5.1. Side (a) and front (b) views of the LHCb Muon system. Reproduced from [220]

.

(Un)crossed hits. The requirements of spatial resolution and rate capability vary strongly over the muon system. To satisfy them, MWPC readout is implemented differently in different stations and regions [11]. All the chambers are segmented into physical pads: anode wire pads or cathode pads in the MWPCs and anode pads in the GEM chambers. In the outer region, R4, the spatial resolution requirements allow for directly using groups of adjacent wires connected to the same electronics channel as physical pads. The size of these pads is used as the pad size for the identification algorithms. In inner regions, however, this approach can not be used,



(a) A cross-section view of a four-gap multi-wire proportional chamber used in M2-M5. Reproduced from [11].



(b) A schematic cross section of a gas electron multiplier (GEM) chamber used in M1R1. Reproduced from [11].

as the required pads are too small to be practically built. The logical pas is defined as an intersection of the horizontal and vertical physical pads. If there is a simultaneous readout in both physical pads, a hit to the logical pad is recorded. If there is no simultaneous readout, a hit is still recorded, but the pad size of the physical pads is used, which is much larger in one of the directions. Single hits given by the physical pads are called uncrossed hits, and by the logical pads are called crossed hits. The difference is illustrated in figure 5.3.

Changes for the Run III. Following [224, 225]:

- M1 station has been removed. It was used only to provide an adequate muon momentum resolution in the L0 trigger. The luminosity increase would make a correct association of M1 hits to the muon track segments virtually impossible.
- Additional shielding (30 cm tungsten) is being installed (as of April 2020) in front of M2R1 to mitigate the higher rates of the innermost regions. The expected rate reduction expected from the simulations is around 25%.
- Additional shielding (Pb) has been installed behind M5 to reduce back-scattered particles from LHC magnets since at least November 2015.

Muon identification algorithms, that were run in the trigger during Run 2 are described in the sections 4.3.1 and 4.3.2.1. The advanced algorithms that were used for offline reconstruction and are being developed for use at Run 3 (including my contribution) are described in the next section of this chapter.

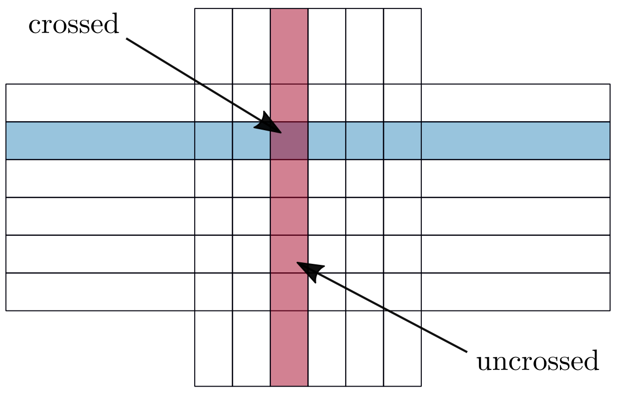


Figure 5.3. Illustration of crossed and uncrossed hits. A hit is considered a crossed hit if it is registered both by horizontal vertical strips. If a hit is only seen by one of them, it is considered uncrossed. Uncrossed hits have, by construction, much larger pad sizes in one of the dimensions. Reproduced from [204].

5.2 muDLL

muDLL is the oldest second-stage muon ID algorithm. A likelihood discriminant between muons and background is computed as the cumulative probability distribution of the average squared distance significance D^2 of the hits in the muon chambers with respect to the linear extrapolation of the tracks from the tracking system [19, 203].

$$D^2 = \frac{1}{N}\sum_{i=0}^{N}\left\{\left(\frac{x^i_{\text{closest}} - x_{\text{track}}}{d^x_i}\right)^2 + \left(\frac{y^i_{\text{closest}} - y_{\text{track}}}{d^y_i}\right)^2\right\}, \tag{5.1}$$

where N is the number of stations containing hits within the momentum-dependent Field of Interest (FOI). The same FOI is used in `IsMuon`. The FOI is determined from the analytical approximation of multiple scattering [203]. $\{x, y\}_{\text{closest}}$ are the coordinates of the hit closest to the track extrapolation; $\{x, y\}_{\text{track}}$ are the coordinates of the track extrapolation to the muon stations and $d_i^{\{x,y\}}$ are the muon pad sizes for the i-th hits that determine the hit coordinates uncertainties. The D^2 distribution for both signal and background depends on multiple scattering – and thus on track momentum and polar angle. To avoid biasing the calibration to kinematic parameters of particular calibration samples, calibration is performed separately in momentum bins and muon detector regions.

5.3 Correlated χ^2

Correlated χ^2 is a logical development of the muDLL approach that takes into account the correlations between the hit residuals in different stations. It has been developed for some time during Run 2; its current form has been finalised at the beginning of 2019.

The idea behind the correlated χ^2 is illustrated on figure 5.4. When the trajectory of a muon changes due to multiple scattering in an early part of the detector, the readings in all the later parts are affected in a correlated fashion.

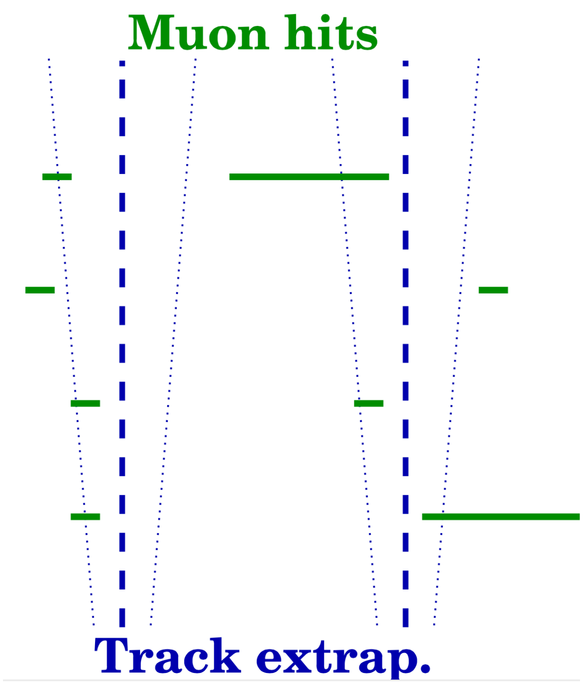


Figure 5.4. Two combinations of muon with the same MuonDLL value. A clear muon pattern (left), and a combinatorial background event (right). Reproduced from [222].

For a given collection of hits (one per station), the variable is defined as follows:

$$\chi^2_{\mathrm{CORR}} = \delta\mathbf{x}^T V_x^{-1}\delta\mathbf{x} + \delta\mathbf{y}^T V_y^{-1}\delta\mathbf{y}, \tag{5.2}$$

where $\delta\mathbf{x}$ and $\delta\mathbf{y}$ are the vectors with distances, in the x and y directions, between the track extrapolation points and the hit position in the M2-M5 stations. V_x and V_y are the covariance matrices, defined as a sum of two parts: V^{RES} and V^{MS}. V^{RES} is a diagonal matrix, computed separately for x and y direction, that takes into account the spatial resolution of the muon stations:

$$V^{\mathrm{RES}}_{jj} = \frac{d_j^2}{12} \tag{5.3}$$

$$V^{\mathrm{RES}}_{jk} = 0 \text{ for } j \neq k, \tag{5.4}$$

where d_j is the x or y pad size corresponding to the muon hit. V^{MS} accounts for the uncertainty introduced by multiple scattering (MS):

$$V^{\mathrm{MS}}_{jk} = \sum_{z_i < z_j, z_k} (z_j - z_i)(z_k - z_i)\sigma^2_{\mathrm{MS},i}. \tag{5.5}$$

$\sigma^2_{\mathrm{MS},i}$ is estimated as [226, 222]:

$$\sigma^2_{\mathrm{MS},i} = \frac{13.6\mathrm{MeV}}{\beta cp} q\sqrt{\frac{z_i}{X_0}}[1 + 0.038\ln(z_i/X_0)] \approx \frac{13.6\mathrm{MeV}}{\beta cp} q\sqrt{\frac{z_i}{X_0}}, \tag{5.6}$$

where p is the particle's momentum, βc the velocity, and q the charge; z_i/X_0 is the thickness of the scattering medium in units of radiation length. The position along the $z-$axis and the thickness of the considered scattering centres are presented in the table 5.1.

MS contribution	z position (m)	z_i/X_0
ECAL+SPD+PS	12.8	28
HCAL	14.3	53
M23 filter	15.8	47.5
M34 filter	17.1	47.5
M45 filter	18.3	47.5

Table 5.1. Position and thickness in units of radiation length for the scattering media contributing to the multiple scattering experienced by particles traversing the muon system. Reproduced from [222].

Choice of the hits In principle, the true track trajectory does not have to correspond to the closest hits to the track extrapolation, as the track might have been affected early by multiple scattering. Two approaches have been explored [222]: the one which would minimise the χ^2_{CORR}, and the one using the closest hits. Minimising the χ^2_{CORR} has been found to give negligible improvement while requiring significantly more computation time. Therefore, the closest hits are used.

Momentum dependence The probability of a muon to reach a given muon station depends on its momentum [227]. For a low-momentum particle, hits in the farthest stations are likely to be combinatorial background. Therefore, for low-momentum particles ($p < 6$ GeV/c), only M2 and M3 stations are used, while for particles with $p \geq 6$ GeV/c all stations are used. The 6 GeV/c threshold coincides with one of the used in `IsMuon`, around 90% of muons with this momentum reach M5.

5.4 Machine learning for Run II

The muon system provides valuable information besides hit coordinates. The number of views (as defined in 5.1) is useful, as noise or spillover hits typically have one single view in contrast to two views for signals hits. For hits with two views, the time difference between the hits is available. Machine learning also allows catching the fine details of the real-world detector behaviour, as it does not correspond precisely to the approximations used in muDLL and χ^2_{CORR}.

We trained machine learning algorithms using the following features, per closest hit in each station:

- space residuals in x, y.

$$\text{x residual} = \frac{x_{\text{closest}} - x_{\text{track}}}{d^x/\sqrt{12} + \text{MS}_{\text{error}}}, \tag{5.7}$$

where MS_{error} is the track extrapolation error estimated from multiple scattering; d^x is the pad x size. For y dimension the formula is the same.

- hit time
- whether the hit is crossed (as defined in 5.1)
- dT: difference of the hit time read by the vertical and horizontal strips

To train the models, we used calibration samples [228] taken from *pp*-collision data collected in 2012, where the particle species is determined from the two decays:

- for muons: $J/\psi \to \mu^+\mu^-$, $4 \cdot 10^5$ tracks
- for pions: $D^{*+} \to D^0(\to K^-\pi^+)\pi^+$, $2 \cdot 10^5$ tracks

The `IsMuon` selection is already applied to these samples. In the training data, the background sample were weighted to match the signal (p, p_T) spectrum.

We used the same data set to train two algorithms. AdaBoost [229] from TMVA [207] is the most widely used in the HEP community implementation of the classic decision tree boosting algorithm. CatBoost [44] is a modern gradient boosting toolkit that uses ordered boosting, a permutation-driven alternative to the classic algorithm, along with several heuristics to achieve state-of-the-art performance. It is based on oblivious decision trees (described in subsection 2.6.2) that allow for better evaluation speed compared to regular trees [124], which is important to meet the time budget of the trigger.

5.5 Machine Learning Towards Run III

The main challenge for Run III is the five-fold occupancy increase. The hardware trigger will be removed in favour of a more flexible, fully software approach. The muon identification procedure has not yet been finalised. Still, it most likely will consist of two stages: fast rough selection with `IsMuon` combined with a second more accurate and computationally expensive step [222]. The second step will provide a continuous decision variable, instead of binary selection – so it may be tuned for conditions of individual analyses, and provide more information for the global PID algorithm.

During Run II, a Global Event Cut was used in the L0 trigger to remove highly occupied events, that are expensive to reconstruct in terms of computing power. Together with the lower Run II luminosity, this means, that there is not enough data in the calibration samples with a high number of primary vertices (nPVs) to accurately emulate the situation after the LHCb Upgrade. As a compromise, to tilt the algorithm towards the high-occupancy conditions, we used a weighting that adds more emphasis on high-nPVs events while retaining reasonable statistics. The distributions are shown in figure 5.5.

For training the classifier, we used the following data calibration samples collected in 2016 *pp* collisions to which the IsMuon selection is applied:

- muons: $J/\psi \to \mu^+\mu^-$, $8 \cdot 10^6$ tracks
- pions: $D^* \to D^{0+}(\to K^-\pi^+)\pi^-$, $4 \cdot 10^5$ tracks
- protons: $\Lambda_0 \to p\pi^-$, $3 \cdot 10^5$ tracks

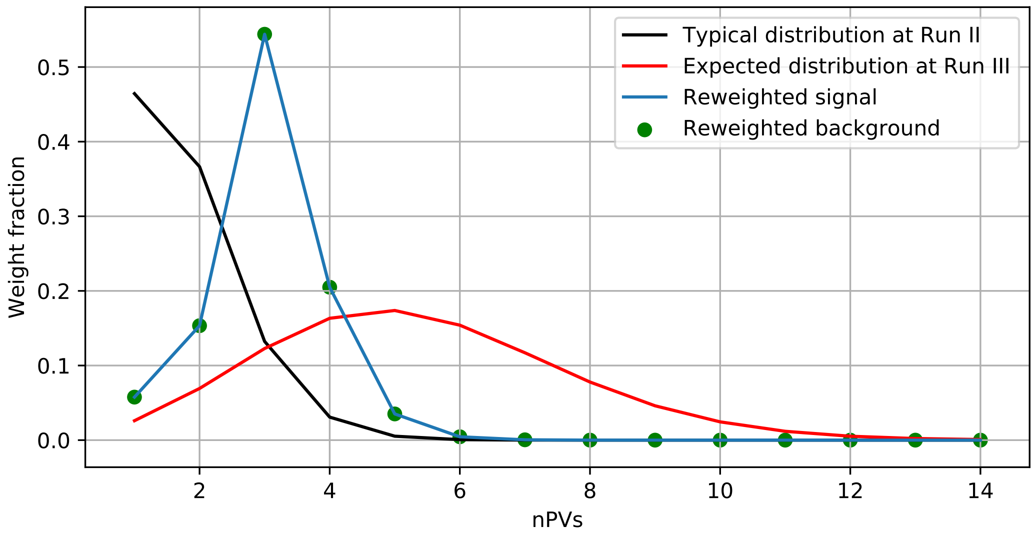


Figure 5.5. The Run II data was weighted to put more emphasis on events with multiple primary vertices.

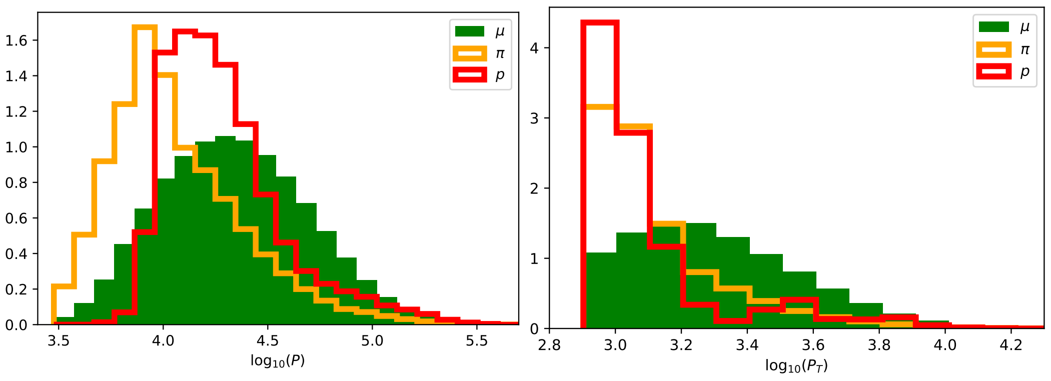
The background sample was weighted to match the signal (p, p_T) spectrum. Both signal and background were weighted by nPVs. The choice of the background samples was determined by two considerations. First, by including both protons and pions, we train the algorithm to distinguish both the combinatorial background, and, to some extent, decays in flight. Second, the background samples momentum coverage matches the muon signal momentum range, as illustrated in figure 5.6. Since we are interested in a classifier that is not biased towards the kinematic distribution of the calibrations samples, it is important to ensure that the momentum distributions of signal and background training samples match. Reweighting can correct the distributions' shape, but it can not make the tracks appear in the phase space regions where there are none. Therefore it is important that the supports of the signal and background kinematic distributions match.

The calibration samples use the sPlot [21] method to subtract the background contribution. sPlot works by assigning weights to examples, and some of the weights are by design negative. Negative weights are not expected by machine learning algorithms and might cause issues from performance drop up to divergent training, depending on the algorithm and dataset in question. We have developed a method to include the results of sPlot into the training; it is described in chapter 6.

The set of training features was expanded compared to Run II to include the output of the χ^2_{CORR} algorithm. The features list is:

- Information about the closest hits in each station: space residuals, hit time, hit delta time, whether the hit is crossed
- χ^2_{CORR}
- Track momentum and transverse momentum

We also trained the algorithm without including χ^2_{CORR} as a feature to understand their relative performance better.



(a) Momentum **(b)** Transverse momentum

Figure 5.6. Kinematic variables distributions for the samples used for classifier training. The plot is made without the momentum and nPVs reweighing, but uses the sWeights.

5.6 Algorithms Evaluation

We evaluated the algorithms on the 2016 calibration data [219]. For the models trained on these data, we used the cross-validation method (described in the section 2.3). We split the data into 5 folds. Then, to obtain the predictions for events in each fold, we trained on the rest four.

The results obtained on the Run II calibration samples without additional weighting are presented in figures 5.7 and 5.8; with weighting by nPVs, momentum and transverse momentum in figures 5.9 and 5.10. CatBoost with χ^2_{CORR} as a feature has the best performance, closely followed by CatBoost without χ^2_{CORR}, the χ^2_{CORR} itself, and the legacy algorithms. The most significant improvement is observed in the low-momentum region. Low-momentum particles are more susceptible to multiple scattering and do not usually penetrate the whole detector, thus leaving fewer hits to work with. By using the advanced algorithms and including the hit timing information, those difficulties can be addressed. A study of CatBoost timing is ongoing. Preliminary testing shows that CatBoost evaluation takes less than 1% of HLT2 time in Run III conditions, and is an order of magnitude slower than χ^2_{CORR}.

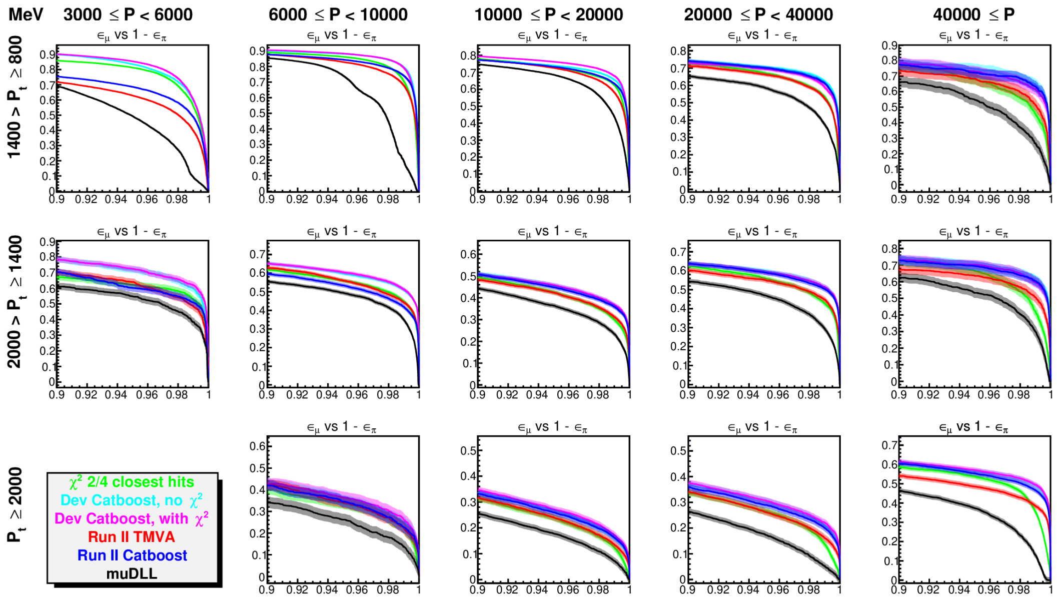


Figure 5.7. Muon efficiency versus pion rejection after applying IsMuon selection, no kinematic weighting using 2016 calibration data. “χ^2 2/4 closest hits” is described in section 5.3. “Run II Catboost” and “Run II TMVA” are described in section 5.4. “Dev Catboost” are the models trained on data weighted to emphasize the high nPVs events as described in section 5.5. “Dev Catboost, with χ^2” uses χ^2_{CORR} as a feature, “Dev Catboost, no χ^2” does not.

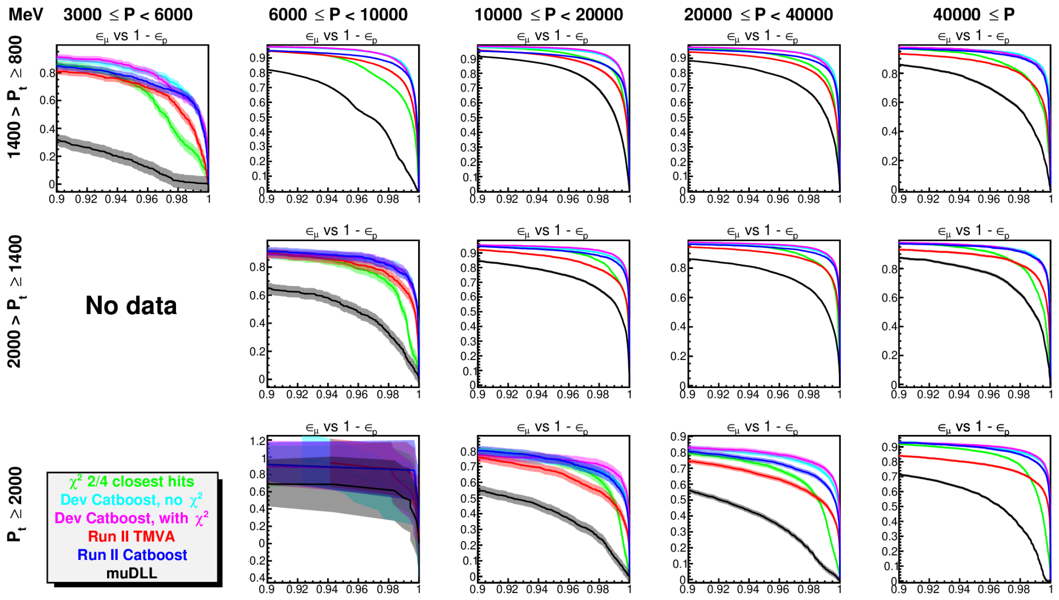


Figure 5.8. Muon efficiency versus proton rejection after applying IsMuon selection, no kinematic weighting using 2016 calibration data. “χ^2 2/4 closest hits” is described in section 5.3. “Run II Catboost” and “Run II TMVA” are described in section 5.4. “Dev Catboost” are the models trained on data weighted to emphasize the high nPVs events as described in section 5.5. “Dev Catboost, with χ^2” uses χ^2_{CORR} as a feature, “Dev Catboost, no χ^2” does not.

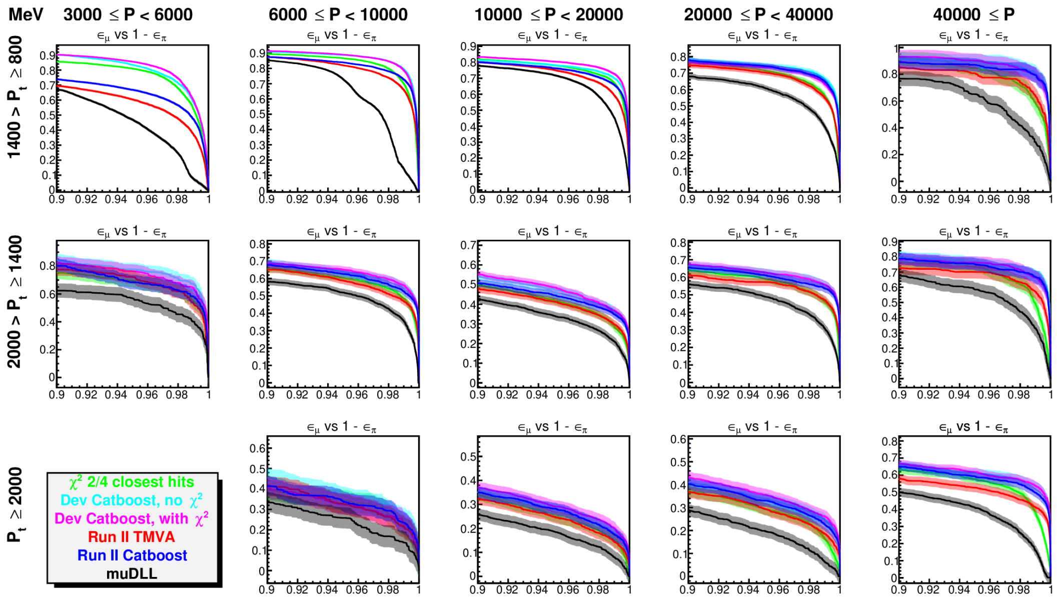


Figure 5.9. Muon efficiency versus pion rejection after applying `IsMuon` selection using 2016 calibration data that is weighted in momentum and nPVs. “χ^2 2/4 closest hits” is described in section 5.3. “Run II Catboost” and “Run II TMVA” are described in section 5.4. “Dev Catboost” are the models trained on data weighted to emphasize the high nPVs events as described in section 5.5. “Dev Catboost, with χ^2” uses χ^2_{CORR} as a feature, “Dev Catboost, no χ^2” does not.

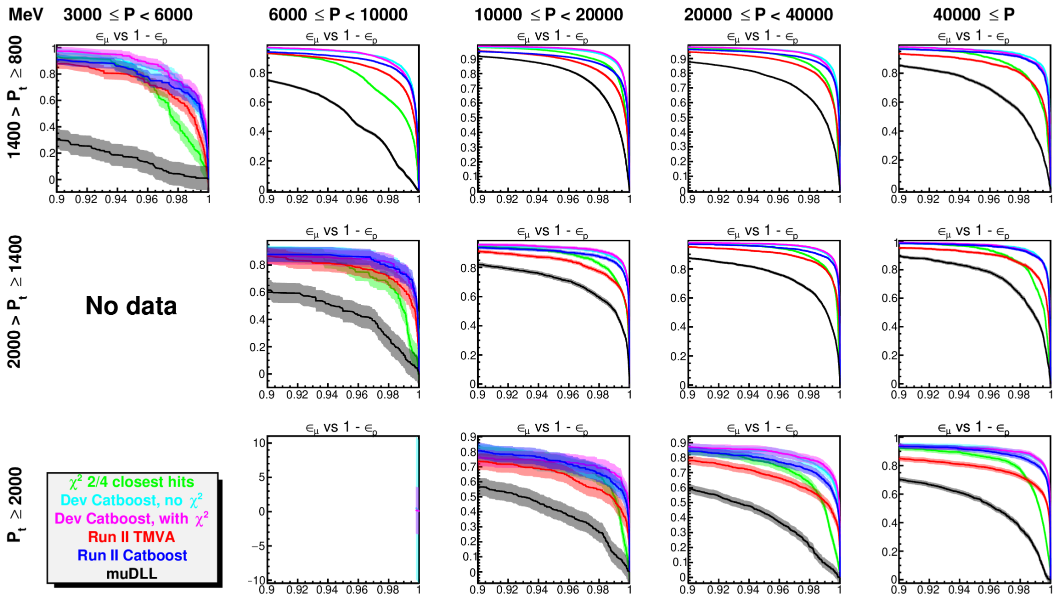


Figure 5.10. Muon efficiency versus proton rejection after applying IsMuon selection using 2016 calibration data that is weighted in momentum and nPVs. “χ^2 2/4 closest hits” is described in section 5.3. “Run II Catboost” and “Run II TMVA” are described in section 5.4. “Dev Catboost” are the models trained on data weighted to emphasize the high nPVs events as described in section 5.5. “Dev Catboost, with χ^2” uses χ^2_{CORR} as a feature, “Dev Catboost, no χ^2” does not. 2

5.7 Data Analysis Olympiad (IDAO)

5.7.1 Introduction

The muon identification problem, as presented in section 5.5, was given as a problem for the International Data Analysis Olympiad [223] organised by the Higher School of Economics and Yandex LLC in winter-spring 2019.

The idea of outsourcing a challenging scientific problem to the general public is old. For example, a series of awards for a practical method of determining longitude has been offered by Spain (1567, 1598) and England (1714) [230]. Modern data science competition took its form in 2006 when Netflix, an online DVD-rental and video streaming service, offered US$1M in a competition to create an algorithm for predicting user ratings for films [231]. Since then, machine learning competitions have become somewhat ubiquitous.

A "classic" competition consists of a supervised learning problem. The participants are provided with a training dataset, both features and labels; the test dataset consisting only of features. The competitors train a model, make a prediction on the test dataset, submit this prediction to the organisers. The organisers evaluate those predictions on the part of the test dataset, called public, and publish the score. This way, there is a public ranking, that allows the participants to compare their solution to the other solutions. Its purpose is to maintain the ardour among the competitors. The rest of the test dataset, the so-called private test, is used at the end of the competition to make a final evaluation of the solutions for prize awards. To claim the prize, the winner typically has to provide the organiser with source code of the solution.

It is hard to estimate the impact of such competitions. Many organisations that run the competitions are commercial companies that are understandably hushed about their operations. When the organisers do publish something, the people doing that are usually the ones who have organised the competition and are incentivised to inflate its achievements. I am not aware of a systematic study of data science competitions.

The winning solutions are usually over-engineered, which means that implementing them requires substantial effort and computation cost, which are not always warranted by the increased prediction quality compared to out-of-the-box machine learning [232, 233]. In many cases, the most difficult part is the formulation of the problem in terms of machine learning, e. g. the discussion for high-energy physics in chapter 3. This is not to say that data science competitions can never produce a valuable result. For example, reference [233] describes a competition, where the algorithm produced by a winning team increased precision from 70% to 90% without decreasing recall for the problem of recognising ships on satellite images. But they didn't claim the baseline solution to be state-of-the-art.

While the effect of an individual competition on the machine learning community might be small, they serve an important role of an arena, where new ideas and techniques are tried, refined, and shared with the world. For example, the XGBoost rise to ubiquity is credited to a data science competition, the Higgs Boson Challenge [234].

One undoubted success of some competitions is the outreach effect. The prime example is the aforementioned Higgs Boson Challenge by the ATLAS collaboration. It attracted 1785 teams (1942 people) who uploaded 35772 solutions, making it the largest ever competition at the time on the most popular competition platform Kaggle [235].

Data science competitions are uniquely useful to organisations that lack in-house expertise in machine learning [236]. A competition allows the organiser to get a reasonable estimate of the capabilities of state-of-the-art machine learning methods to solve their problem – at a comparatively low cost, as there is no need to hire a team of experts. An additional benefit is the establishment of contacts with the participants, who might be contracted to implement the solution, should it be deemed satisfying.

Due to confrontational nature of a competition, it imposes a burden on the organiser to ensure that it is not possible to "hack" a competition – produce a solution that performs well according to the validation metric but is fundamentally flawed. The most basic example is when the competition data, including the test part, turns out to be available elsewhere. Or when the features contain the information, that will not be available when making real-world predictions. For example, there was a dataset used to predict whether a patient had prostate cancer, which had feature telling whether the patient had received prostate surgery [237]. A more sophisticated example involving particle physics is described in section 3.1.

A sensitive issue is opening the data for the competitions. The data can represent a competitive advantage for a commercial company. If the data is about humans, publishing it might violate their privacy, e. g. Netflix was hit with a lawsuit for its competition, even though the data they published had been anonymised [238]. Big CERN experiments have committed themselves to release their data – but only after an embargo period, that would allow extracting most of the scientific value [239, 240, 241, 242].

5.7.2 Muon ID Competition

The problem offered to competitors was the one presented in section 5.5. The training dataset consisted of $2 \cdot 10^7$; the public test of $2.7 \cdot 10^6$ examples; the private test of $1 \cdot 10^7$ examples. The dataset contained muon, pion and proton calibration samples, weighted by nPVs, momentum and transverse momentum as described in section 5.5. The features were track momentum, transverse momentum, extrapolation coordinates, hits in the FOI around the track, and the set of hits, that provided the minimal χ^2_{CORR} value. muDLL and χ^2_{CORR} values were not provided. Since the number of FOI hits is different from event to event, without additional preprocessing, the competition could not be addressed by the majority out-of-the-box machine learning algorithms. Another interesting aspect was the negative example weights used to subtract the background; they are covered in chapter 6.

Deriving a single metric to compare solutions is complicated. If one solution performs better than the other across the momentum spectrum for all decision thresholds, like χ^2_{CORR} compared to muDLL in figures 5.8 and 5.7, we can be confident that it is better. In all other cases, there is a potential trade-off: there might be usage cases, for which each of the algorithms beats the other, e.g. in some momentum regions, or some efficiency thresholds. But a competition requires a single number to rank the submissions. Thus, the competition metric was the background rejection at 90% signal efficiency.

The competition offered two lanes[1]. On the first one, solutions were ranked only by the quality of signal/background classification. This type of competition is the most straightforward from the participants' point of view, as they only need to submit the predictions. The drawback is that the complexity of the solutions is not restricted – while in real life the classification quality must be balanced by the computational cost. Hence, the participants had to send the algorithm code for the second lane to comply with the execution time limit, which was set to twice the Run 2 software trigger requirements and only then ranked them by classification quality. This imposes an additional burden on the competitors who must ensure that their code runs in the competition server environment.

Unfortunately, we struggled to use the competition to improve the algorithms used at LHCb. The winning solutions for both lanes mostly replicated the current approach, where the closest hits coordinates, residuals and physics-derived variables (muDLL) are used as features for off-the-shelf machine learning algorithms. The winning solutions also use as features a large number of various coordinates' differences, angles, and vector norms. The winner of the first lane used an average of XGBoost and CatBoost [243], the winner of the second lane used CatBoost [244]. To handle the negative sWeights, one of the solutions discarded the events with negative weights, and the other used the absolute value of the weights. It can be shown that both these approaches are mathematically unsound and lead to biasing the resulting classifier as it is not trained to separate muons vs non-muons, but some variant of muons plus the background vs non-muons plus the background. One of the leading teams described their solution in a blog post [245] (in Russian). The overall improvement over straightforward CatBoost training was marginal and very limited due to high generalisation error.

The primary benefit was outreach. 1287 teams from 78 countries registered for participation, and there have been around 10 publications in Russian mass media about the Olympiad [223]. We hope that exposing machine learning practitioners to a (subjectively interesting) problem from the field of high-energy physics might result in more collaboration (like this thesis).

5.8 Conclusion

Machine learning allows gaining significant improvement of muon identification with the muon subsystem at LHCb. On Run II calibration samples, background passthrough at 90% signal efficiency is reduced by a third in the crucial low momentum region compared to the next best algorithm (χ^2_{CORR}). This gain stays true when the data are reweighted to emphasise events with multiple primary vertices.

We have integrated the Run 2 model into the LHCb software, and its output is available for analysis of the data taken in 2016 – 2018 [246].

For Run 2, using CatBoost improves the evaluation speed by a factor of 3 compared to TMVA BDT while giving higher discriminating power. Thus, it led to

[1]On the website the lanes are called tracks. I use a different term in the thesis to avoid possible confusion with the particle tracks.

discontinuation the development of the TMVA-based algorithm.

The design of the Run 3 muon identification procedure will depend on two things, which are not available at the moment of writing this thesis. First, the trigger time budget distribution – whether the performance gains are worth the computational cost. Second, the performance of the algorithms on the Monte-Carlo simulation of Run 3 conditions.

Chapter 6

Machine Learning on Data With sPlot Background Subtraction

Experimental data obtained in high energy physics experiments usually consists of contributions from different processes. A large part of a typical data analysis consists of selecting the target decay from all collected data. A common technique used as a part of this process is sideband subtraction. It requires a signal-enriched and a signal-poor phase-space regions be identified (usually by an invariant mass fit). A commonly used method is sPlot [21]. It assigns weights (sWeights) to events, some of them negative. This does not present a problem if the next analysis steps use simple single-dimensional tools, like histograms, but is an obstacle for some multivariate machine learning methods. In this chapter, we prose a mathematically rigorous way of training machine learning algorithms on such data.

In machine learning terms, we are dealing with a particular model of label noise coupled with prior knowledge. For each example, we know the probability that its label has been flipped. The flipping probabilities are not constant but are sampled from some distribution independently of the features' distribution. As an example, consider the case where you have a dataset of with stars labelled from telescope observations. The observations were taken in different atmospheric conditions. For each observation the atmospheric conditions are known; the probability of misclassifying a star is a known function of atmospheric conditions.

This chapter is structured as follows. In section 6.1 we introduce the sPlot technique. In section 6.2 we discuss the implications of negative event weights on training of machine learning algorithms. In section 6.3 we review the existing approaches to machine learning on data weighted with the sPlot. In section 6.4 we propose methods to robustly training of machine learning models. In section 6.5 we present experimental results that demonstrate practical viability of the proposed method for synthetic problem constructed from the Higgs dataset and the LHCb muon identification problem.

The method presented in this chapter has been used to train the high-occupancy muon identification models described in section 5.5.

6.1 sPlot

sPlot is a method for background subtraction widely used in high-energy physics, particularly in B-physics [247]; reference [21] has over 1460 citations. In addition to analyses, sPlot is used at LHCb for subtracting background in the LHCb calibration samples described in subsection 4.3.7.

Take a dataset populated by events from N_s sources. Assume that distribution of some variables is known for each source, call these variables discriminative. Usually, the discriminative variable is the reconstructed invariant mass, and the probability densities are estimated by a maximum likelihood fit. sPlot is a statistical technique that reconstructs the distributions of the rest of the variables (call them control), provided they are independent of the discriminative variables for each event source.

Let $p_k(m)$ be probability density function of the discriminative variable m of the k-th species, N_k be the number of events expected on the average for the k-th species, N be the total number of events, m_e be the value of m for the e-th event. Compute the covariance matrix of the signal and background probability density functions $\mathbf{V}$:

$$\left(\mathbf{V}^{-1}\right)_{nj} = \sum_{e=1}^{N} \frac{p_n(m_e)p_j(m_e)}{\left(\sum_{k=1}^{N_s} N_k p_k(m_e)\right)^2} \tag{6.1}$$

The sWeight for the e-th event corresponding to the n-the species is obtained using the following transformation:

$$\text{sWeight}_n(m_e) = \frac{\sum_{j=1}^{N_s} \mathbf{V}_{nj} p_j(m_e)}{\sum_{k=1}^{N_s} N_k p_k(m_e)} \tag{6.2}$$

If the dataset is weighted with sWeights, the distribution of the control variables will be an unbiased estimate of the corresponding pure species. In the rest of the chapter we deal with the two species scenario, named signal and background. An example of sPlot application is presented on figure 6.1. The proof is available in reference [21].

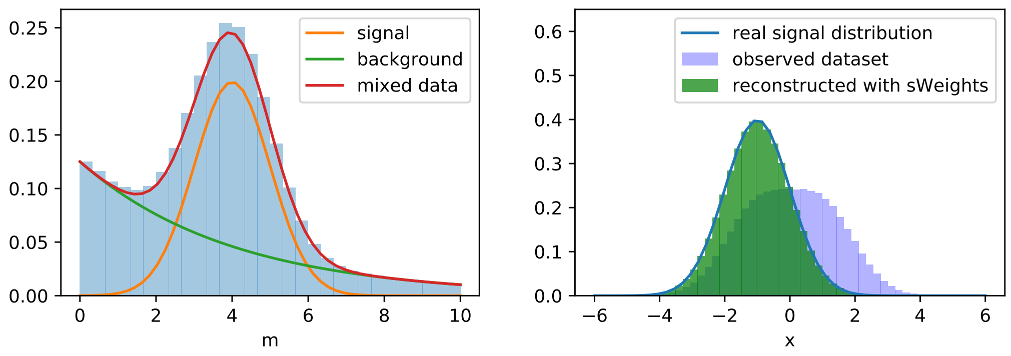


Figure 6.1. An example of sPlot application. To the left: the known distributions of the discriminative variable m. To the right: the observed mixture and the reconstructed with sPlot signal distribution of the control variable x.

6.2 The Problem of Negative Weights

Let us consider an event with positive signal weight $w_s > 0$ and negative background weight $w_b < 0$ and the classic cross-entropy loss function (equation 2.31):

$$L = -w_s \log(p_s) - w_b \log(1 - p_s), \tag{6.3}$$

where p_s is the model output – the predicted probability of this event being signal.

$$\lim_{p_s \to 1} L = -\left(-|w_b|\right) \lim_{p_s \to 1} \log(1 - p_s) = -\infty. \tag{6.4}$$

Thus, directly incorporating sWeights into the cross-entropy loss causes it to lose the lower bound. The same holds for the mean squared error and other losses without an upper bound in the unweighted case. Training most machine learning algorithms is an optimization problem (see sections 2.5, 2.6), and, for some algorithms, such as a large-capacity fully-connected neural network, negative event weights make this optimization problem ill-defined, as the underlying optimization target loses the lower bound as well. An example illustrating diverging training is presented in figure 6.2. The model training using the sWeights as event weights quickly diverges in contrast to the same model training using the true labels or our losses. The model trained on true labels does not even start to overfit – the test score keeps climbing, while the test score of the model trained with sWeights as event weights drops dramatically. As expected, the test ROC AUC for the model that was trained using the true labels is the best among all methods.

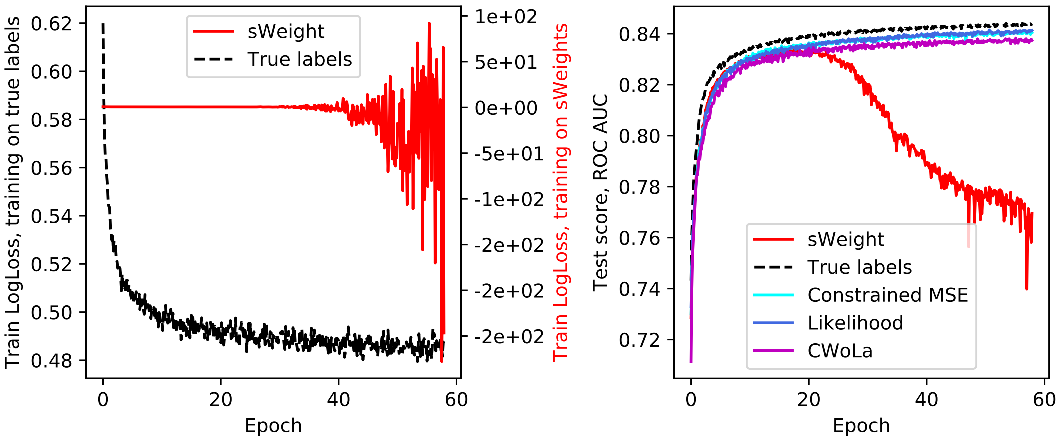


Figure 6.2. Learning curves of a neural network trained on the Higgs dataset using the true labels and the artificially introduced sWeights. Likelihood and Constrained MSE methods are described in section 6.4, CWoLa in [143], the dataset in section 6.5, and the network in subsection 6.5.1.

However, most machine learning algorithms do not blindly minimize the loss value on the training dataset, as this is highly likely to lead to overfitting. They add various regularization terms to the optimized functional that, in general, penalize model complexity or overconfidence (equation 2.5). We are not aware of peer-reviewed literature exploring the impact of regularization on learning with negative weights. One such technique appears in discussions within the High Energy Physics community [248]. They propose avoiding the unbounded loss by requiring leaves of decision tree to have positive total weight. It is also possible to regularize neural networks into having bounded loss. For example, use the L2 regularization on weights and take the root of the degree equal to the number of layers plus one from the cross-entropy loss.

6.3 Related Work

The practical viability of using machine learning methods on data weighted with sPlot is demonstrated in references [249, 250, 251, 207, 158]. They propose a training procedure for the case where a classifier is desired to separate the same signal and background that are defined by the sPlot. Take each event twice, once as signal, once as background with the corresponding sWeights, then train the classifier as usual, in case of splitting the dataset (e. g. for cross-validation), take care to include both event and its mirror match in the same fold. The papers, however, do not attempt to analyze the core issue of negative weights impact on machine learning algorithms, limiting themselves to requiring that the classifier supports negative weights.

Another possible approach is suggested by [143] – classification without labels (CWoLa). Take a binary classification problem, where the true labels are not known, but there are two datasets for which is known they contain the classes in different proportion. They also require that events with the same label are independently sampled from the same distribution. The authors prove that the optimal classifier for those two datasets is also optimal for separating the classes. It is possible to use the discriminative variable to define regions with different signal/background proportions. This method ignores the per-event probability information. In our experiments (presented in section 6.5), CWoLa has shown worse quality on a real finite dataset than the methods that use this information.

6.4 Proposed Approaches

6.4.1 sWeights Averaging (Constrained MSE)

Let m be the variable that was used to compute the sWeights, x is the rest of variables. Let $p_{\text{signal}}(x)$ be the probability density function of signal, $p_{\text{mix}}(x)$ be the probability function of the signal and background mixture from which the data were sampled. It is possible to weight the dataset, so distribution of x will be that of the signal component with positive weights equal to class probabilities $W(x) = \frac{p_{\text{signal}}(x)}{p_{\text{mix}}(x)}$. The data weighted with the sWeights achieve the same distribution. Therefore the signal probability for an example with features x will be the average of the sWeights over examples with features x:

$$E_m\left[w(m) \mid x\right] = \frac{p_{\text{signal}}(x)}{p_{\text{mix}}(x)}. \tag{6.5}$$

To prove formally. Let m be the variable that was used to compute the sWeigths, x be the rest of variables and $f(x)$ is any smooth function of x.

$$E_{x\sim p_{\text{sig}}}(f(x)) = \int dx f(x) p_{\text{sig}}(x). \tag{6.6}$$

Define

$$W(x) = \frac{p_{\text{sig}}(x)}{p_{\text{mix}}(x)}. \tag{6.7}$$

Then

$$E_{x\sim p_{\text{sig}}}(f(x)) = \int dx f(x) W(x) p_{\text{mix}}(x) \tag{6.8}$$

By definition of sWeights:

$$E_{x\sim p_{\text{sig}}}(f(x)) = \int dx dm \cdot w(m) f(x) p_{\text{mix}}(x, m), \tag{6.9}$$

where $w(m)$ is the sWeight.

$$E_{x\sim p_{\text{sig}}}(f(x)) = \int dx dm \cdot w(m) f(x) p_{\text{mix}}(x) p_{\text{mix}}(m|x) \tag{6.10}$$

Rearrange the multipliers in the double integral:

$$E_{x\sim p_{\text{sig}}}(f(x)) = \int dx f(x) p_{\text{mix}}(x) \int dm w(m) p_{\text{mix}}(m|x) \tag{6.11}$$

From equation 6.8,

$$\int dx f(x) W(x) p_{\text{mix}}(x) = \int dx f(x) p_{\text{mix}}(x) \int dm w(m) p_{\text{mix}}(m|x). \tag{6.12}$$

An obvious solution to this equation is $W(x)$ equal to the second integral:

$$W(x) = \int dm w(m) p_{\text{mix}}(m|x) = \mathbb{E}_m(\text{sWeight}(x, m)) \tag{6.13}$$

Notice, that the left-hand side of (6.13) is the optimal output of a classifier, while the right-hand side can be estimated by a regression model. Our first proposed approach is to perform mean-square regression directly on sWeights. Since the optimal output lies in $[0, 1]$, one can easily avoid a priori incorrect solutions by, for example, applying the sigmoid function to the model output. The resulting loss function is the following:

$$L = \sum_i \left(w_i - \frac{e^{f_\theta(x_i)}}{1 + e^{f_\theta(x_i)}} \right)^2, \tag{6.14}$$

where w_i is the sWeight and $f_\theta(x_i)$ is the model output. We have implemented this loss for the CatBoost machine learning library [44] and the source code is available on GitHub.[1]

[1] https://github.com/kazeevn/catboost/tree/constrained_regression

6.4.2 Exact Maximum Likelihood

Alternatively, one can invoke Maximum Likelihood principle and avoid the sPlot technique altogether. Let us denote signal and background classes as S and B, model parameters as θ. By definition of the log-likelihood $l(\theta)$:

$$l(\theta) = \sum_i \log\left[p(x_i, m_i, \mathrm{S} \mid \theta) + p(x_i, m_i, B \mid \theta)\right] = \sum_i \log\left[p(x_i, m_i \mid \theta, \mathrm{S}) P(\mathrm{S} \mid \theta) + p(x_i, m_i \mid \theta, \mathrm{B}) P(\mathrm{B} \mid \theta)\right]. \quad (6.15)$$

Since x_i and m_i are assumed to be independent within individual classes, expression (6.15) can be simplified further:

$$l(\theta) = \sum_i \log \sum_{\mathrm{C} \in \{\mathrm{S,B}\}} p(x_i \mid \theta, \mathrm{C}) p(m_i \mid \mathrm{C}, \theta) P(\mathrm{C} \mid \theta) \quad (6.16)$$

Using the Bayes theorem, we expand the conditional probabilities:

$$l(\theta) = \sum_i \log \sum_{\mathrm{C} \in \{\mathrm{S,B}\}} \frac{P(\mathrm{C} \mid x_i, \theta) p(x_i \mid \theta)}{P(\mathrm{C} \mid \theta)} p(m_i \mid \mathrm{C}, \theta) P(\mathrm{C} \mid \theta) \quad (6.17)$$

After simplification we get:

$$\begin{aligned} l(\theta) &= \sum_i \log \sum_{\mathrm{C} \in \{\mathrm{S,B}\}} P(\mathrm{C} \mid x_i, \theta) p(x_i \mid \theta) p(m_i \mid \mathrm{C}, \theta) \\ &= \sum_i \log p(x_i \mid \theta) \sum_{\mathrm{C} \in \{\mathrm{S,B}\}} P(\mathrm{C} \mid x_i, \theta) p(m_i \mid \mathrm{C}, \theta) \\ &= \sum_i \log p(x_i \mid \theta) + \sum_i \log \sum_{\mathrm{C} \in \{\mathrm{S,B}\}} P(\mathrm{C} \mid x_i, \theta) p(m_i \mid \mathrm{C}, \theta) \end{aligned} \quad (6.18)$$

Since both summands are independent, they can be maximised independently. Since we are not interested in modelling $p(x \mid \theta)$, we can omit it from further analysis:

$$\max_\theta l(\theta) = \max_\theta \sum_i \log\left[P(\mathrm{S} \mid x_i, \theta) p(m_i \mid \mathrm{S}, \theta) + P(\mathrm{B} \mid x_i, \theta) p(m_i \mid \mathrm{B}, \theta)\right] + \text{const}. \quad (6.19)$$

We know the true values of $p(m_i \mid S)$ from the problem setup (they are used to compute the sWeights), and choose $p(m_i \mid \mathrm{S}, \theta) = p(m_i \mid \mathrm{S})$. The final loss function is the following:

$$L(\theta) = -\sum_i \log\left[f_\theta(x_i) p_{\text{signal}}(m_i) + (1 - f_\theta(x_i)) p_{\text{background}}(m_i)\right], \quad (6.20)$$

where $f_\theta(x_i)$ is the output of the model, $p_{\text{signal}}(m_i)$ and $p_{\text{background}}(m_i)$ are the probability densities of the signal and background m distributions.

Note, that by substituting $p_{\text{signal}}(m_i)$ and $p_{\text{background}}(m_i)$ by the class indicator (0 or 1) in loss (6.20), it becomes conventional cross-entropy loss (2.31).

6.4.3 Classes with Separate Background

Our approach trivially extends to the case where there are several classes, each with its own background defined by a separate set of sWeights. Labels are available that say to which class signal/background mixture each example belongs. This problem naturally arises in the context of particle identification as selections of calibration samples for each particles species are different.

For simplicity, consider the case of two classes. We want to train a model that would be optimal for separating the signal parts of the dataset, ignoring both backgrounds. A loss function of a machine learning model is the expected value of the per-example loss function. In order to ignore the backgrounds, we can use the sWeights when calculating the expectations:

$$L = E_{x,y}\left(E_m\left[\mathrm{w}_1(m) \mid x\right]\right) y \log f(x) + \left(E_m\left[\mathrm{w}_1(m) \mid x\right]\right)(1-y)\log(1-f(x)) \approx \sum_{i=1}^{N} y_i p_1(x_i)\log(f(x_i)) + (1-y_i)p_2(x_i)\log(1-f(x_i)), \quad (6.21)$$

where $w_1(m)$ is the sWeight value corresponding to class 1 (signal) and $p_1(x)$, $p_2(x)$ are the probabilities that given example is signal estimated by applying our methods from sections 6.4.1 and 6.4.2 to portion of the dataset selected by label y.

6.5 Experimental Evaluation

We tested the methods on two datasets. The UCI Higgs dataset [252, 253] and the LHCb muon identification dataset (which was used for the IDAO competition described in section 5.7).

6.5.1 UCI Higgs

This is the largest open dataset from the field of High-Energy Physics. It has 28 tabular features. We split it into a train and a test parts containing $8.8 \cdot 10^6$ (80% of the total) and $2.2 \cdot 10^6$ (20%) events respectively. The dataset is labelled and it does not feature sWeights, so we introduced them artificially. For both signal and background events, we added a virtual "mass" distributed as shown in figure 6.1 and used it to compute sWeights. We also compared with the CWoLa method [143] by splitting the data into a signal-majority and a background-majority regions. The signal region was chosen with centre at $m = 4$ (centre of the signal m distribution) and width so that it would include half of the events. We used neural network and gradient boosting models described in the following subsections.

6.5.1.1 Neural Networks

For the experiments with fully-connected neural networks we use networks with 3 hidden layers: 128, 64, 32 neurons for the experiments with $8.8 \cdot 10^6$ and $8.8 \cdot 10^5$ samples in the training sets, and 64, 32, 16 neurons in cases of $8.8 \cdot 10^4$ and $8.8 \cdot 10^3$ training samples. Model capacity varies to adjust for the low sample sizes.

All networks use leaky ReLu (0.05) activation function. Networks are optimised by Adam [80] algorithm with learning rate $2 \cdot 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ for $2.2 \cdot 10^6$ steps (32 full "passes" over the original training sample) with batch size 128. Learning rate is set to the value lower than commonly used ones due to high variance of the gradients for CWoLa.

Each experiment is repeated five times with varying training dataset (if subsampled) and initial weights. However, within each experiment networks for different methods have identical conditions: they share initial weights, are trained in the same subsample and use an identical sequence of batches.

Due to the presence of logarithm, loss function (6.20) might become computationally unstable for networks with large weights. For the experiments with $8.8 \cdot 10^4$ training samples, l_2 regularisation is introduced to the exact maximum likelihood loss function. This regularisation does not affect results in any significant way, besides limiting network weights to computationally stable values.

6.5.1.2 CatBoost

We use CatBoost with the following parameters: 1000 trees, leaf_estimation_method="Gradient", version 0.10.2 with our losses added and check for negative weights removed, source code is available [2]. Each experiment is repeated 10 times with varying training dataset (if subsampled).

6.5.1.3 Results

The results are presented in figures 6.2 and 6.3.

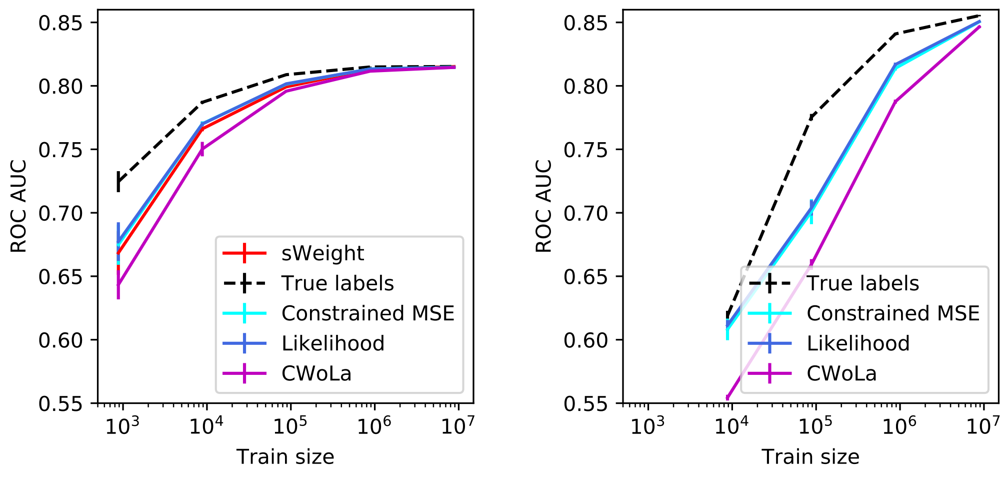
For neural networks, using sWeights as event weights for neural network leads to divergent training. As illustrated in figure 6.2, after the 20-th epoch, training loss oscillates with a high magnitude and the test score decreases. Training on true labels gives the best performance, which is expected. All methods, besides directly using the sWeights, converge to the same quality as the size of the training dataset increases. Constrained MSE and Likelihood loss functions show the same performance, both methods outperform CWoLa.

For CatBoost, directly using sWeights as event weights results in stable training and the same performance as for our methods. Training on true labels again gives the best performance, CWoLa the worst. The code of the experiments is available on GitHub[3].

6.5.2 LHCb Muon Identification

We evaluate the proposed methods on the problem of LHCb muon identification. To our knowledge, this is the only open dataset [223] that has sWeights. The dataset contains different particle species obtained from different calibration decays [218] and is labelled. Each species has its own background that is subtracted with a

[2] `https://github.com/kazeevn/catboost/tree/constrained_regression`
[3] `https://github.com/yandexdataschool/ML-sWeights-experiments`



(a) CatBoost **(b)** Neural network

Figure 6.3. Experimental evaluation of performance of different loss functions on the Higgs dataset as a function of train dataset size. sWeight – cross-entropy weighted with sWeights, it is not reported for the neural network due to divergence of optimization and, hence, highly stochastic nature of the results; True labels – logloss using the true labels; Constrained MSE – our loss function defined by (6.14); Likelihood – our loss function defined by (6.20); CWoLa – method from [143]

separate application of the sPlot method. For simplicity, for study in this subsection, we use only pion and muon species and ignore the kinematic weights. We split the dataset into train and test parts containing $2 \cdot 10^6$ and $6 \cdot 10^5$ examples respectively. For each train dataset size, the classifier was fitted 10 times to estimate the standard deviation of the scores.

The results are in figure 6.4. Direct application of sWeights statistically significantly loses to our approach. The gap decreases with the training set size increase. However, ignoring the sWeights during training also yields good results, so it seems that the impact of sWeights on the problem is limited. While this is a good thing for LHCb muon identification, this limits the utility of the data set with respect to evaluating the methods for training machine learning algorithms on data with background subtraction.

So we did an additional test, where we discarded the original sWeights and constructed our own in such a way as to maximize their impact on the classification problem. The idea of the procedure is to artificially introduce a very muon-like component into the pionic background. Such background will greatly affect a "naive" classifier, that disregards the information provided by the sPlot. The procedure is the following:

1. Train a CatBoost classifier to separate signal and background using the method from subsection 6.4.1.

2. Use its predictions as event weights to train a classifier to separate muons and pions as proposed in section 6.4.3

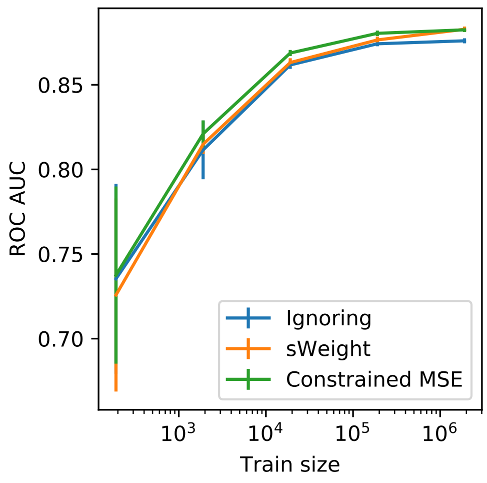


Figure 6.4. Experimental evaluation of CatBoost with on the MuID dataset. sWeights – using sWeights directly as weights for logloss; Constrained MSE – our regression (6.14) used to transform the sWeights, the result used as weights for classification; Ignoring weight – training ignoring the sWeights.

3. Select 30% of the examples with muon ground truth label with the highest muon classification scores and mark them as pionic background
4. Take the signal probabilities obtained from the first step and assign deterministic signal and background labels by cutting on its output. At this stage, the dataset is fully labelled – for each example there is a label whether it is considered ”muon signal”, ”muon background”, ”pion signal” or ”pion background”.
5. Generate “pseudomass” for both muon and pion samples (distributions are on figure 6.5), used it to compute the new sWeights

After generating the artificial sWeights, we evaluated different classification methods on the dataset. The results are presented in figure 6.6. As expected from the experiment design, the performance of a classifier that ignores the sWeights is significantly lower compared to the rest. Using the sWeights as example weights has the worst performance among the methods that account for the sWeights, with a surprising exception of CatBoost with train size equal to $2 \cdot 10^3$. For CatBoost both our methods show the same performance. For neural networks, Constrained MSE beats the Likelihood.

The models' parameters are:

- Fully-connected neural networks (NN): 4 layers, 512, 256, 128 neurons in the first three layers and either 1 or 2 neurons in the last one, leaky ReLu (0.05), optimised by Adam [80] algorithm with learning rate $2 \cdot 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, trained for 32 epochs. Regressors and classifiers use the same architecture;

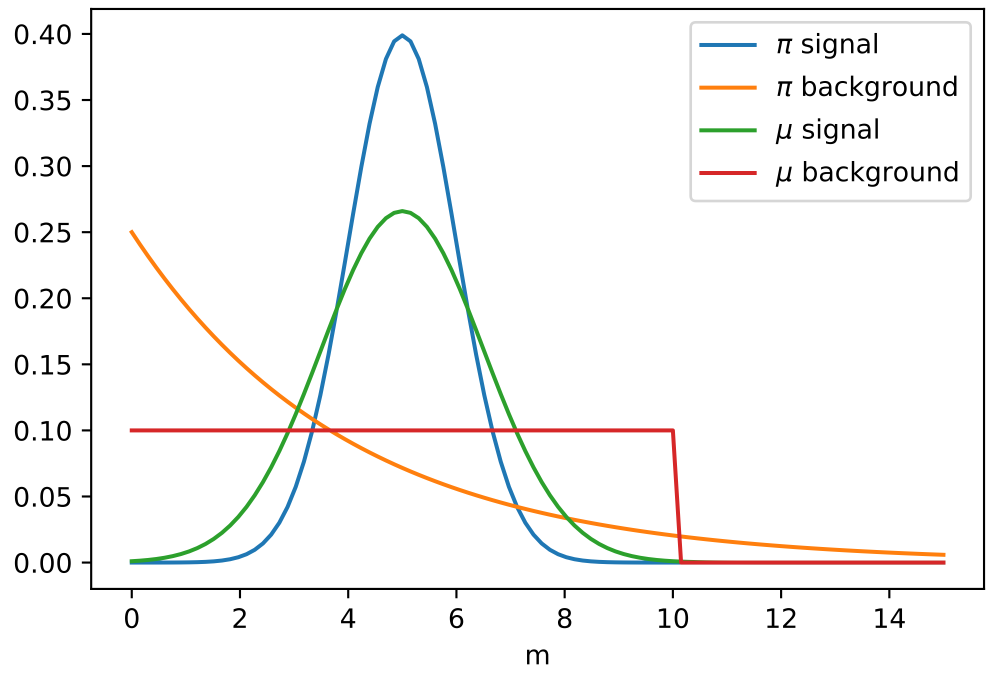


Figure 6.5. Discriminative variable distributions used to regenerate the sWeights for the additional test.

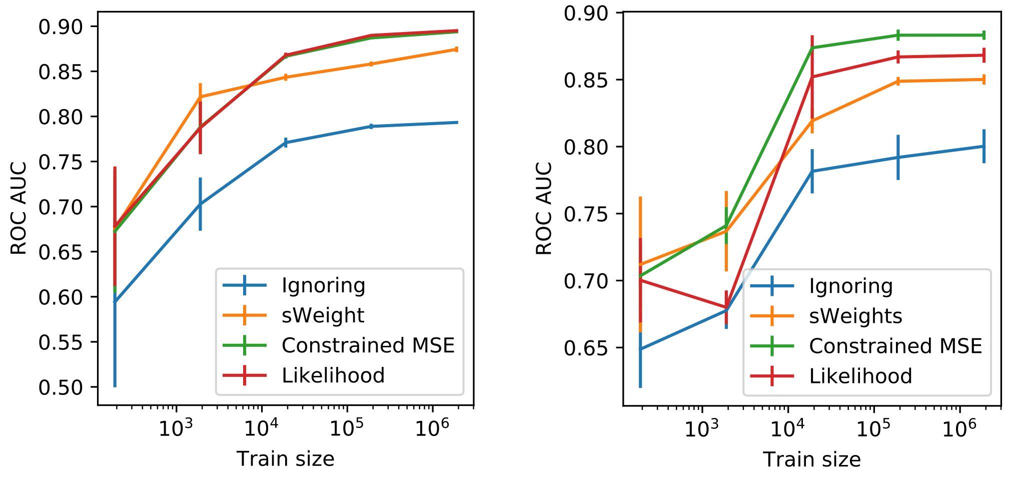
- Catboost: 500 trees, leaf_estimation_method="Gradient", version 0.10.2 with our losses added and check for negative weights removed.

6.6 Conclusion

Training machine learning models on real experimental data (as opposed to Monte-Carlo) is very desirable, as it allows the model avoid bias caused by imperfection of the simulation. One of the obstacles is that real data often come contaminated with some sort of background. A common way of dealing with it, the sPlot method, introduces negative event weights. Training a machine learning algorithm on a dataset with negative weights means coping with a loss that potentially has no lower bound. The implications depend on the algorithm in question. In our experiments, neural network training diverges, while gradient boosting oblivious decision tree does not.

Our contribution is two loss functions that allow a machine-learning algorithm to obtain class probabilities from background-subtracted data without encountering negative event weight at all. The probability of an event with given control variables values to be signal is the expected sWeight, that can be estimated by a regression with the corresponding loss function (6.4.1). We invoke the Maximum Likelihood principle to construct a loss function that avoids the sPlot technique and therefore doesn't face the issue of negative event weights (6.4.2).

Our work paves a rigorous way to use any machine learning methods on data with sPlot-based background subtraction. The Constrained MSE method has been



(a) Catboost **(b)** Deep Neural Network

Figure 6.6. Experimental evaluation of our loss functions on the MuID dataset with artificial sWeights. sWeights – using sWeights directly as weights for logloss; Constrained MSE – our regression (6.14) used to transform the sWeights, the result used as weights for classification; Likelihood – our likelihood (6.20) for the transformation; Ignoring weight – training ignoring the sWeights.

used to train the high-occupancy models described in section 5.5.

Chapter 7

Global Charged Particle Identification

The LHCb detector consists of a number of different subdetectors that do the particle identification (PID): Ring-Imaging Cherenkov detectors (described in subsection 4.2.2.1), calorimeters (subsection 4.2.2.2), and the muon subsystem (section 5.1). The basic physics behind the particle identification in LHCb is illustrated in figure 7.1. The tracking system reconstructs charged particles tracks. Momentum is measured from the track bent in the magnet. Particles, that fall into the RICH momentum acceptance range, emit Cherenkov photons. Electrons are absorbed by the electromagnetic calorimeter (ECAL), hadrons by the hadronic calorimeter (HCAL). The calorimeters measure the deposited energy. Muons pass through the detector and leave hits in the muon chambers. For each track, there is a variety of variables that are associated with it and may aid in determining the particle type. They are described in detail in appendix B.1. There are dedicated algorithms

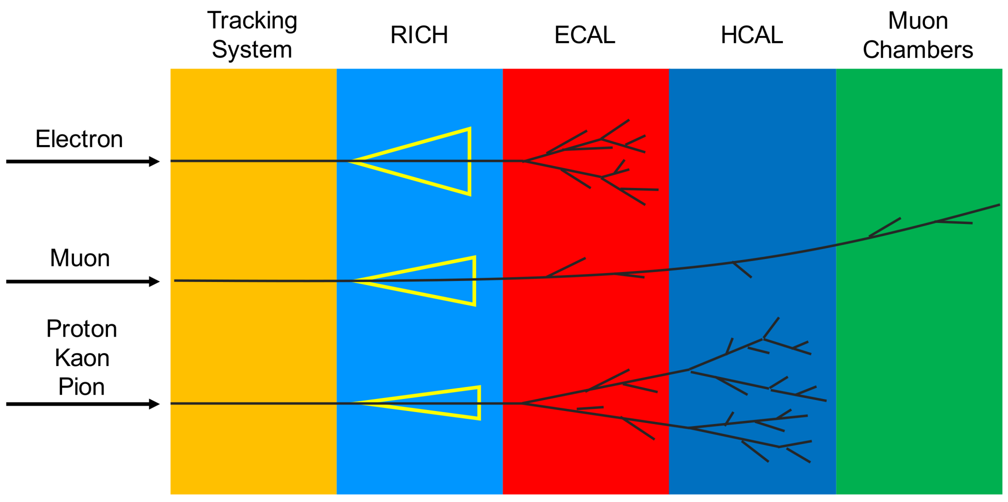


Figure 7.1. A sketch of the responses of the different LHCb particle identification subdetectors to different charged particles. Reproduced from [254].

for each subdetector that process the subdetectors' response and estimate different variables for each charged tracks. For physics analysis, we need to aggregate this information into a single decision about particle type. In the context of this chapter, we call this aggregation global particle identification, in contrast to "local" PID, that is based on a single subdetector.

In this chapter, we describe how global particle identification works at the LHCb. In section 7.1 we discuss the mathematical formalism of the PID objective. In section 7.2 we describe the baseline approach, where for each subdetector explicit particle type hypotheses likelihoods are computed. In section 7.3 we provide the holistic approach based on machine learning that leads to a better result. In section 7.4 we describe the machine learning algorithms that we evaluate – state-of-the-art gradient Gradient Boosting Decision Trees (GBDT) and deep neural networks. In section 7.5 we evaluate the performance of those approaches.

My contribution to global particle identification is design and implementation of machine learning algorithm based on gradient boosting. Its error, as measured by 1-vs-all $1 - \text{ROC AUC}$ (area *over* the ROC curve) score, is 18%-54% lower compared to the baseline solution on simulated data. The work has been published in [255], and the algorithms have been implemented in the LHCb software [256].

7.1 Objective and Formalisation of the Global PID

PID algorithm assigns each charged track in the detector the scores of it corresponding to a particular particle type: electron, muon, pion, kaon, proton and ghost (reconstruction error). It can be viewed as a summary statistic, that collapses the raw detector readout into a small set of variables. PID is an intermediate step in the LHCb data processing. In principle, it is not necessary to designate it as a separate step, as signal selection can be made to use the raw detector response directly. But having PID as a single distinct step offers important benefits:

- Labour and expertise division. Separate teams produce the PID variables that are then reused in multiple analyses.
- Computational efficiency. Processing the information from the PID detectors can be quite expensive, with a single set of variables, this has to be done only once.

One would want to determine the types of particles in the detector precisely. In the real world, absolute certainty is impossible – and the goal of the PID becomes fuzzy. A good way to see the difficulty is figure 4.9. If a particle has momentum and Cherenkov angle in a region, where the bands for muons and pions intersect, what PID hypothesis should be assigned to it? There are several reasonable ways to answer this question:

1. Posterior probabilities, that are the best possible estimate of the particle type for each track, taking into account the full event information. The estimate incorporates the knowledge of both the distribution of the parameters of the $p - p$ collision products and the detector response.

2. Optimal from the point of view of important analyses. Not all tracks are equally important for the physics goals of LHCb. If we specify the target decays, PID can be tuned to optimally separate those important decays, without regard to performance in the other cases.

3. "Agnostic" PID. Compute posterior probabilities, but instead of the real distribution of the $p-p$ collision products kinematics and types, use some arbitrary distributions.

Computing the true posterior probabilities of particles types conditioned on the full detector readout is appealing from the mathematical point of view. However, this imposes a fundamental challenge. The PID variables computed in such a fashion would depend on the quantities that we try to measure. If some decay never occurs in the SM, the "true posterior" PID variables will make sure it is never observed in the real detector.

Producing PID variables, that are optimised for the needs of the analyses is the most sensible choice from the point of view of the physics goals. PID variables have no utility on its own; they only matter as an intermediate step. There are two primary complications with this approach. First, analyses are complex, not fully formalised and automated. In order to produce the optimal PID, they must be formalised and automated. There is ongoing research into designing optimal summary statistics [257], but it is yet to demonstrate feasibility for a large high-energy physics experiment. Second, it is desirable to have a single set of PID variables, that can be utilised in centralised selections. There are many analyses in LHCb, and an analysis-aware PID will have to take into account all of them.

"Agnostic" PID provides the absolute certainty of the independence of its output from the physical parameters of interest. The main drawback for this approach is an inherently inferior performance when applied in analyses. There are two main reasons for this. First, there are many particles in an event – and the responses of the PID detectors depend on all of them together. If particles are generated from simple artificial distributions, these occupancy effects will be modelled incorrectly. Second, the distributions of the kinematic variables for different particle types, that is assumed during the model training is different from the one observed in the real detector.

An important conclusion to draw here is that a single set of PID variables can't be optimal for every use case and at the same time be independent of the physics parameters of interest. An algorithm that is optimal for a given distribution of the kinematic variables will be suboptimal for another decay with a different distribution.

The LHCb collaboration uses a variation of the second option. It is not formalised, relies on proxy metrics for quality and physics considerations for ensuring independence from the physical quantities of interest. It is used to produce an overwhelming majority of the published LHCb results. The training procedure and assumptions behind it have not been published in a peer-reviewed publication; its performance studies have been published [10].

The training dataset consists of a series of MC samples, that include some of the most important decays for LHCb. The training uses all the tracks in the events. The independence of the PID from the physical parameters of interest is ensured through the selection of features. Each track is considered independently, and the variables are supposed to correspond to the detector response to this particular track in isolation.

Mathematically speaking, define a track as tuple containing the kinematic parameters k (momentum, transverse momentum, origin vertex) and species s. A physical event E is a set of tracks. The raw response of the LHCb detector to the event is R. This response is a random variable, which is defined by some probability distribution $P(R, E)$. These and other notations are summarised in table 7.1. The goal of the LHCb experiment as a whole is to estimate

$$P(E) = \int dRP(R, E), \tag{7.1}$$

with E restricted to a region of the phase space, that corresponds to the LHCb physics program.

Variable	Definition
k	kinematic parameters of a track
s	particle species of a track
E	physical event, a set of tracks
R	raw response of the LHCb detector to an event
v	PID variables associated to a track, reconstructed from R and k
V	set of the PID variables of all tracks in an event

Table 7.1. Notation used in section 7.1

Monte-Carlo simulation allows to approximate $P(R, E)$ – given a model of physics, compute the distribution of long-lived tracks inside the detector, and the detector response to them. This presents a conceptual problem, introduced in the "true posterior" paragraph – without further assumptions, we can not estimate the true $P(E)$. The precise assumptions are many, formulating and justifying them is the most laborious part of a physics analysis.

In the case of the global PID, the core assumption we make is that the unknown physics parameters are contained only in the distribution of the tracks $P(E)$ — and their interaction with the detector is governed by the physics assumed known. While this seems obvious, it is easy to construct an example, where this does not hold. Consider the measurement of the muon lifetime. The detector response – whether the particles in question reach the muon chambers – depends wholly on the lifetime. In LHCb, we assume, that the properties of the relatively stable particles that reach the PID subdetectors are known. In principle, this already allows us to resolve the core problem:

$$P(E) = \frac{P(R)P(E|R)}{P(R|E)}, \tag{7.2}$$

$P(R)$ can be estimated from the experimental detector response, by assumption, the conditional probability $P(R|E)$ can be estimated from MC and its inverse $P(E|R)$ using machine learning. However, computationally, this is not an easily tractable problem. An event is a very complex object and training a machine learning algorithm to estimate $P(E|R)$ would require an impractically large amount of data and computing power.

Therefore, for PID, we enter the second and third assumptions. The tracks are independent random variables. Let v be the PID variables that correspond to a track and V be the set of all PID variables in an event. V is reconstructed from R. We assume that the PID variables for different tracks are also independent random variables:

$$P(R, E) \approx P(V, E) = \prod_{e \in E} P(v|e)P(e). \tag{7.3}$$

This allows to finally formulate the PID problem:

$$P(s|v,k) = \frac{P(s,v,k)}{P(v,k)} = \frac{P(v|s,k)P(s,k)}{P(v|k)P(k)} = \frac{P(v|s,k)P(s|k)P(k)}{P(v|k)P(k)} = \frac{P(v|s,k)P(s|k)}{P(v|k)} \tag{7.4}$$

If we specify the $P(s,k)$, an MC simulation allows to sample from $P(v,s,k)$, and a machine learning algorithm can learn $P(s|v,k)$. The learned distribution depends on the $P(s,k)$ distribution – so the result, strictly speaking, will be wrong, if the model is applied to a sample with a different $P(s,k)$. To illustrate, imagine a decay, that produces muons always with momentum in the range $[1.1, 1.2]$ GeV and pions in the range $[2.1, 2.2]$. In this setting, muons and the pions can be fully separated, $P(s|v,k) \in \{0, 1\}$. However, a PID model trained over a more broad dataset, will not be able to reproduce this certainty, as, in general, muons and pions are not separated by momentum – just like a model trained using this decay will fail in the general case.

This is an unavoidable limitation of having a single set of PID variables. But they are still useful. Finding physics beyond SM means finding events, that occur with different frequency in the real world and SM. Ideally, we just want to take a process of interest, count the fraction of events, where it occurs and compare the result to the SM prediction. The problem is that we can't observe the process of interest directly, only its imprint on the detector. Therefore, we design a selection based on the observed data, R. We want it to select all the events, that contain the process of interest, and only them. But even if it is not perfect, a discovery can still be made, it would just require more data. If it is built upon suboptimal PID variables than it would be suboptimal, but still perfectly useful. On the other hand, the more precise those variables are, the easier it is to use them to construct a more sensitive selection operator.

On the other hand, equation 7.4 opens a way to adjust the PID to a different sample. Let this sample be drawn from a different distribution $Q(v,s,k)$:

$$Q(s|v,k) = \frac{Q(v|s,k)Q(s,k)}{Q(v,k)} = P(s|v,k)\frac{Q(v|s,k)}{P(v|s,k)}\frac{Q(s,k)}{P(s,k)}\frac{P(v,k)}{Q(v,k)}. \tag{7.5}$$

$$\frac{Q(v|s,k)}{P(v|s,k)} = 1, \tag{7.6}$$

as it is determined by the detector, which stays the same. $\frac{Q(s,k)}{P(s,k)}$ can be estimated by a classifier trained to classify the two MC samples, one from each distribution, using (s,k) as features. $\frac{P(v,k)}{Q(v,k)}$ can be estimated a classifier trained to classify the two data samples, one from each distribution, using (v,k) as features. For this to work, the kinematic coverage of the two samples must overlap, otherwise the ratios can not be estimated by ML with any degree of certainty.

The alternative to this procedure is training an optimal PID operator directly on MC for the second sample. The advantage of this procedure is that it does not require simulation of the PID subsystems response – only a physics simulation to obtain the (s, k) sample and a data sample of (v, k). Since MC does not fully correspond to the data, it remains an open question that must be decided on a case-by-case basis, whether they can be freely mixed like this.

The result of this procedure relies on the $P(s|v, k)$, $\frac{Q(s,k)}{P(s,k)}$, $\frac{P(v,k)}{Q(v,k)}$ estimates. The quality of the estimates relies on the quality of machine learning – which is my contribution.

7.2 Adding Likelihoods

Historically, the first way to combine the information from different subsystems was adding the log-likelihoods. If we assume that the responses of different subsystems are independent random variables, and the likelihoods are indeed the true likelihoods derived from the same priors, the particle hypotheses' likelihoods are the sum of the likelihoods computed from the individual subsystems' responses:

$$\mathrm{DLL}^{\mathrm{species}} = \mathrm{DLL}^{\mathrm{species}}_{\mathrm{RICH}} + \mathrm{DLL}^{\mathrm{species}}_{\mathrm{CALO}} + \mathrm{DLL}^{\mathrm{species}}_{\mathrm{MUON}}. \tag{7.7}$$

The performance improvement compared to using just the response of the "corresponding" detector is illustrated in figure 7.2. The improvement can be clearly seen, for example at 90% electron identification efficiency the pion misidentification rate drops from $\sim 6\%$ to $\sim 0.6\%$ [10].

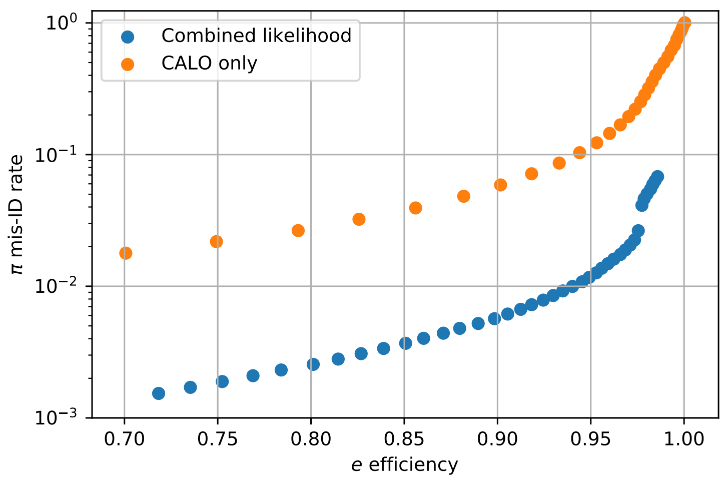


Figure 7.2. Electron identification efficiency versus pion misidentification rate. Data from [10].

Another example of the advantage of the combined likelihood over the per-subsystem is also presented in reference [10]. It refers to one of the most prominent LHCb results, a search for the rare decays $B^0_s \to \mu^+\mu_-$ and $B^0 \to \mu^+\mu^-$, which deeply relies on muon identification [258]. Decays modes $B^0_{(s)} \to h^+h^-$, where $h \in \{K, \pi\}$, fake signal, if one or both hadrons are misidentified as muons. This misidentification probability has been evaluated using the $D^0 \to K\pi$ and $D^{\star+} \to D^0\pi^+$ calibration samples in bins of p and p_t matched to the kinematic spectrum of the simulated $B^0_{(s)} \to h^+h^-$ decays. When the muon candidate tracks are also required to satisfy a criterion on the combined likelihoods, $\mathrm{DLL}^K < 10$ and $\mathrm{DLL}^\mu > -5$, the background rejection is improved by a factor of 6 at the price of 3% signal efficiency reduction.

The DLL addition approach does not allow to realise the full potential of the LHCb particle identification [259]. Conceptually, just adding log-likelihood disregards potential correlations between the underlying variables. Practically, adding log-likelihoods relies on them being mathematically well defined (i.e. logarithm of the product of probability values) and correspond to the same prior distribution, which is hard to achieve. An additional issue is the difficulty of building likelihoods, that would include all the information from subsystems, including the acceptance flags and such.

7.3 Combining Information with Machine Learning

Machine learning is a logical way to combine the various variables into a decision on particle type. It does not rely on the variables to have a particular form. From the point of machine learning, this is a 6-class multiclassification problem on tabular data. Within LHCb this approach has been floated since at least 2001, as per references [260, 261], although, from the subsequent performance report [262] it appears that the proposed algorithms did not leave the prototype stage. The approach commonly used by the LHCb experiment during Run 2 is called ProbNN [10, 263]. ProbNN consists of separate neural networks for each particle type (electron, muon, pion, kaon, proton and ghost) and track type (long, downstream and upstream) – $6 \times 3 = 18$ models in total. It uses different subsets of features for different particle types. Those are selected by physical considerations, and the number of features ranges from 18 to 49. The full feature list is provided in the appendix B.1. The model uses fully-connected neural networks with a single hidden layer. The number of neurons in the first layer is equal to the number of features, the number of neurons in the hidden layer is 1.4 the number of features, the output layer has a single neuron.

The improved quality of particle identification for μ-vs-π and p-vs-π is illustrated in figure 7.3. A more detailed study of ProbNN performance can be found in reference [10].

7.4 State-of-the-art Machine Learning

The quality of the approach described in the previous section is limited by the simplicity of the machine learning model. To test the hypothesis whether better PID can be obtained using the same data, we built models for long tracks using feature engineering and several state-of-the-art machine learning algorithms. We evaluated two top approaches, introduced in chapter 2: gradient boosting decision

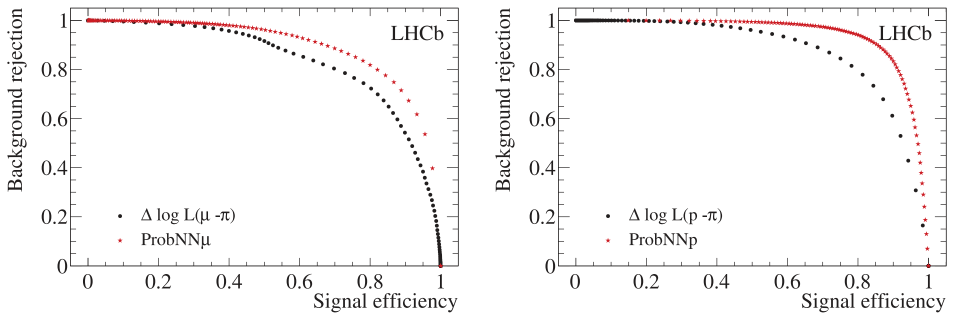


Figure 7.3. Background misidentification rates versus muon (left) and proton (right) identification efficiency, as measured in the $\Sigma^+ \to p\mu^+\mu^-$ decay study. The momentum range is 5–10 GeV/c for muons and 5–50 GeV/c for protons. Data sidebands are used for backgrounds and Monte Carlo simulation for the signal. Reproduced from [10].

trees (as implemented by CatBoost [44]) and a dense deep neural network (Deep NN). The new ML approaches reuse the ProbNN training datasets and variables, but exchange the machine learning algorithms for better ones.

For CatBoost, we train a model for each track type in one-vs-rest fashion as is in ProbNN as this approach has shown slightly superior performance compared to a single multiclassification model. The parameters of the CatBoost model were optimised. To do this, we used the random search method. For the parameters, that affect the result in the most, reasonable distributions were chosen, based on the recommendations from CatBoost developers [264]. Those distributions were sampled, and each sampled combination of the parameters was evaluated via cross-validation. The choice of the optimisation method was dictated by the availability of the proprietary Nirvana [265] computing system at Yandex, that allows for easy execution of a massive number of embarrassingly-parallel tasks.

The deep neural network has 5 layers with 60, 300, 300, 400 and 6 neurons correspondingly, and with ReLU activation function. The last layer uses the softmax activation, to ensure that the network outputs satisfy basic properties of probability: each of them lies in $[0, 1]$ and their sum equals 1. Each of the outputs of the neural network corresponds to the probability of the track to be of a type. Like in ProbNN, to improve convergence and ensure equal treatment of all features, the input features are scaled, so that they lie in the interval $[-1, 1]$.

The models were trained on a simulated dataset with 10^6 examples per particle type of the same nature, that was used for ProbNN training.

7.5 Performance

As discussed in section 7.1, the performance of a PID algorithm varies with the distribution of the kinematics distribution of the particles it is applied to. We did the following tests:

- On the simulated data, generated in the same way as the training dataset. This provides a way to assess the performance over a broad kinematic spectrum, the one the LHCb collaboration settled on as having sufficient coverage.

- On the calibration samples. They contain real data, but their kinematic spectrum is determined by what is available, and they are not representative of the overall needs of the LHCb analyses.

7.5.1 Simulation

For evaluating the models we used a simulated test dataset with 10^6 examples per particle type, that was not used in model training, but produced using the same procedure. The following figures display the performance of different models measured in different ways:

1. One-vs-rest ROC curves in figure 7.4
2. One-vs-rest ROC AUC scores in table 7.2
3. Relative improvement of the one-vs-rest $1 - \text{AUC}$ scores compared to the baseline (ProbNN) in table 7.3
4. One-vs-one ROC AUC scores for 6 particularly important for analysis pairs in table 7.4
5. One-vs-one ROC curves in figure 7.5.

Model	ghost	electron	muon	pion	kaon	proton
ProbNN	0.9484	0.9855	0.9844	0.9346	0.9148	0.9178
Deep NN	0.9632	0.9915	0.9925	**0.9587**	0.9320	0.9319
CatBoost	**0.9664**	**0.9917**	**0.9929**	0.9586	**0.9322**	**0.9323**

Table 7.2. One-vs-rest ROC AUC scores for different models on simulated data. All the differences are statistically significant with p-value less than 10^{-6} as estimated by the method from reference [52].

Model	ghost	electron	muon	pion	kaon	proton
Deep NN	−29%	−41%	−52%	−37%	−20%	−17%
CatBoost	−30%	−43%	−54%	−32%	−20%	−18%

Table 7.3. Relative increase of the one-vs-rest $1 - \text{AUC}$ scores compared to the baseline (ProbNN) for different particles species and ghosts (lower is better) on simulated data. The statistical uncertainty is lower than 1%

Model	μ-vs-π	K-vs-π	p-vs-π	p-vs-K	e-vs-π	e-vs-K
ProbNN	0.9763	0.9298	0.9295	0.7450	0.9888	0.9935
Deep NN	0.9869	0.9399	**0.9456**	0.7633	0.9941	0.9965
CatBoost	**0.9878**	**0.9406**	0.9418	**0.7716**	**0.9945**	**0.9969**

Table 7.4. One-vs-one ROC AUC scores for different models on simulated data. The uncertainty is less than 0.0003.

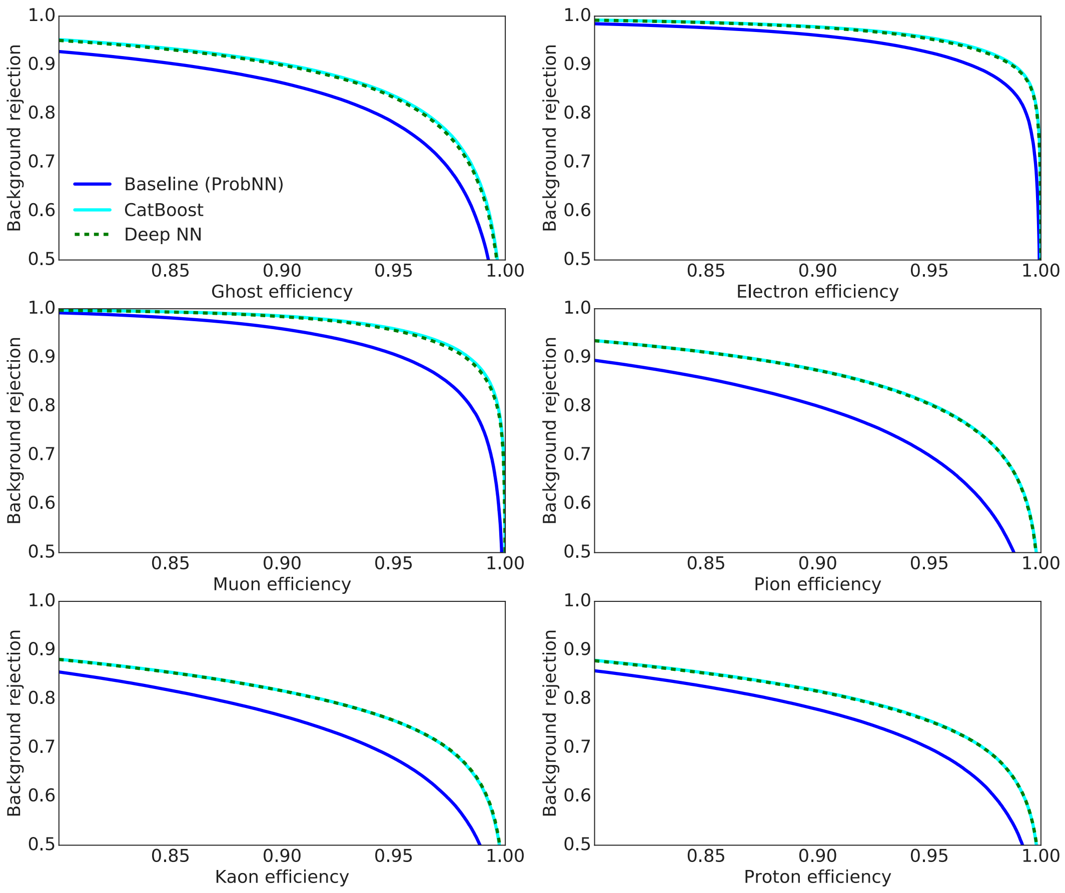


Figure 7.4. Performance of the models in terms of one-vs-rest ROC curves. Reproduced from [266].

Both of our models show noticeable improvement compared to ProbNN. Almost everywhere, CatBoost has a slight, but statistically significant lead over the deep neural network. Nevertheless, the performance of both methods, despite their very different nature, is extremely close. This allows us to speculate that this is the limit achievable with the current state of machine learning research for these data and problem formulation.

7.5.2 Real Data: Calibration Samples

We studied the operators' performance using the high-momentum calibration samples described in subsection 4.3.7. The performance of the different models in terms of one-vs-one ROC curves for select pairs is presented in figure 7.6, in terms of one-vs-one ROC AUC scores in table 7.5. For all pairs, except for K-vs-π, CatBoost shows the best performance.

To study the lower performance in the K-vs-π separation, we additionally measured the performance using samples coming from different decays: $K_S \to \pi\pi$ and $D_S \to \phi(KK)\pi$ [267]. The momentum and transverse momentum distributions of the particles in these calibration decays are closer to the ones in the MC sample used for training. This provides for a way to understand whether the performance

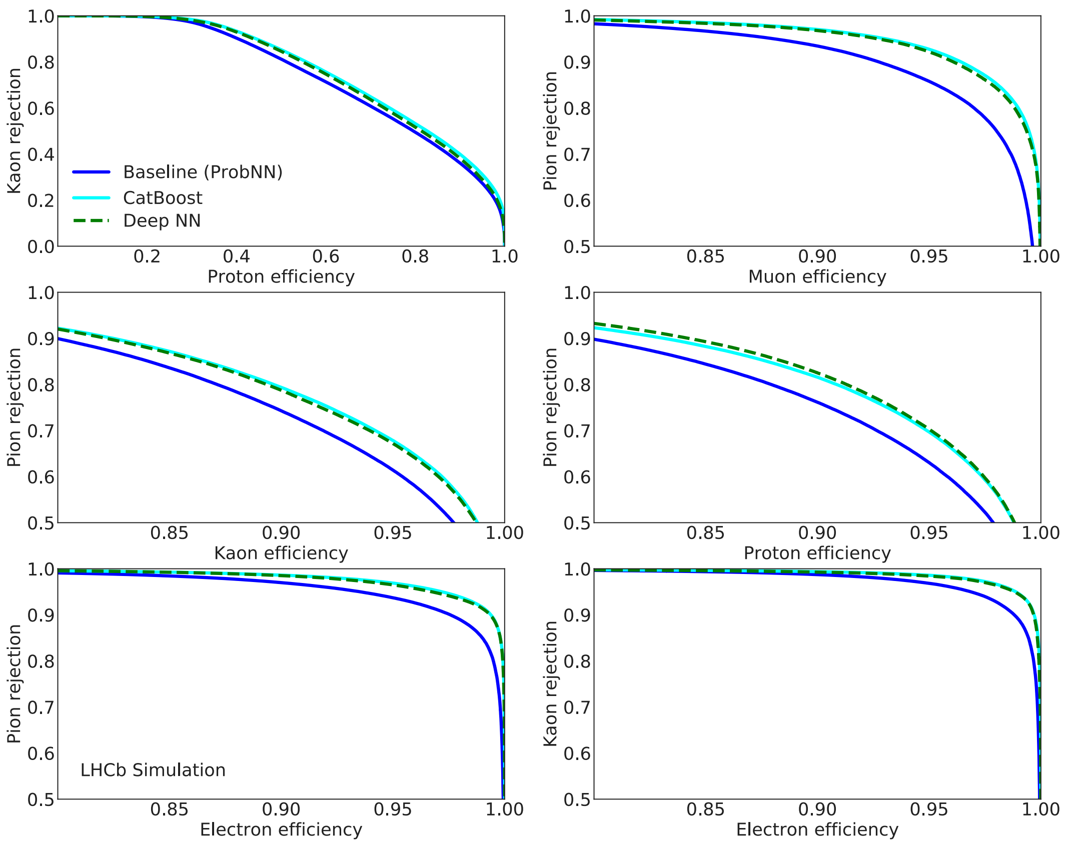


Figure 7.5. One-vs-one ROC curves for select particle pairs obtained on simulated samples. Private run. Reproduced from [266].

gap is due to more powerful models overfitting to the peculiarities of MC, or due to unavoidable limitation of a single set of momentum-dependent PID variables. The datasets used in the study are:

- The MC samples, generated similarly to the one used for training ProbNN and our models
- Calibration data samples, $D^\star \to D^0(K\pi)\pi$
- Calibration MC samples, $D^\star \to D^0(K\pi)\pi$
- Calibration data samples in the low momentum region, $K_S \to \pi\pi$ and $D_S \to \phi(KK)\pi$

The momentum and transverse momentum distributions are shown in figure 7.7. We did a series of tests comparing performances of different algorithms:

- On the high-momentum calibration sample and the simulation of the same decays. The result is presented in figure 7.8. The ordering of the algorithms in terms of performance is the same. The performance on MC is slightly better than the performance on data.

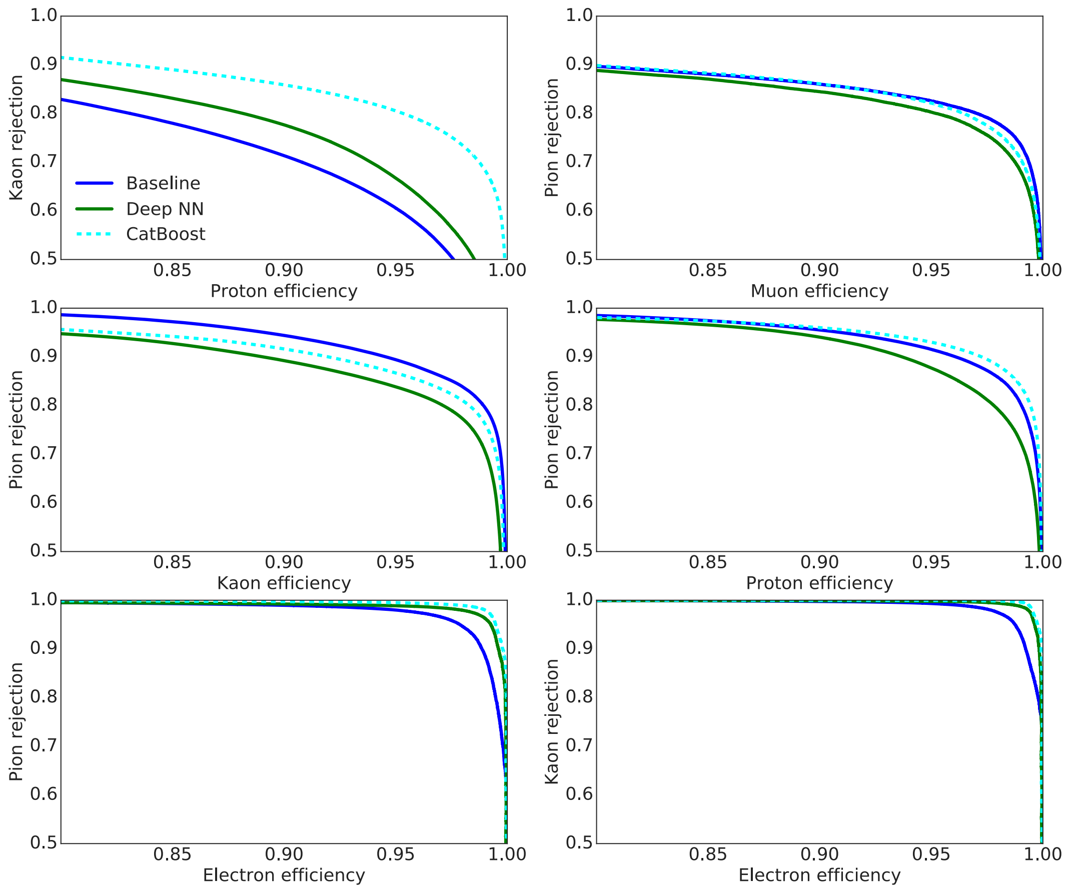


Figure 7.6. One-vs-one ROC curves for select particle pairs obtained on 2016 calibration samples. Private run. Reproduced from [266].

- On the low-momentum calibration sample. The result is presented in figure 7.9. Like when using the MC data, CatBoost has the best performance.

As a final test, we measured the algorithms' performance as a function of the momentum and transverse momentum. The result is presented in figure 7.10.

7.6 Conclusion

Using state-of-the-art machine learning methods allows for significant improvement of Global PID performance across the board. The new methods have been evaluated on both simulated and real data. Different algorithms show the best performance on the low- and high-momentum calibration datasets, with comparable performance gap between the algorithms. This suggests that the evaluated methods push to its limit the current approach with having a single set of PID variables trained on MC. There are two possible venues for future improvement. The first is closing the gap between the simulated and real data – by improving the simulation quality or incorporating the calibration samples. The second is meticulously building the training dataset as a composition of datasets with the numbers of events proportional to their importance to the LHCb physics program.

Model	μ-vs-π	K-vs-π	p-vs-π	p-vs-K	e-vs-π	e-vs-K
ProbNN	0.9353	**0.9830**	0.9846	0.9067	0.9944	0.9980
Deep NN	0.9308	0.9649	0.9772	0.9303	0.9957	0.9986
CatBoost	**0.9372**	0.9734	**0.9847**	**0.9571**	**0.9980**	**0.9992**

Table 7.5. One-vs-one ROC AUC scores for different models on 2016 calibration data. The uncertainty is less than 0.0003.

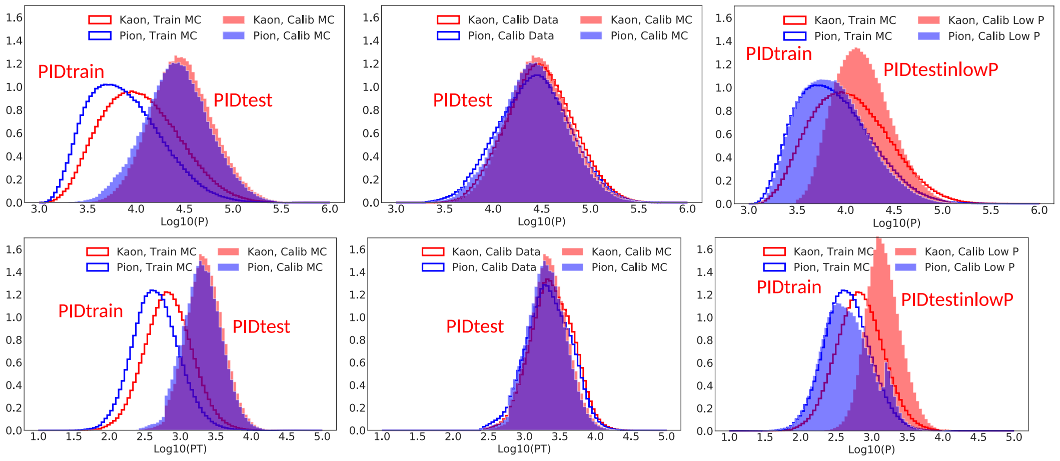
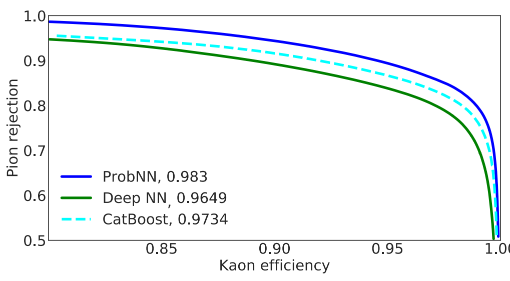
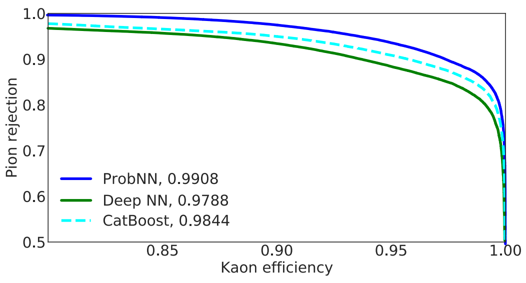


Figure 7.7. Momentum (upper panels) and transverse momentum (lower panels) distributions. Train MC is the simulated dataset used for PID training; Calib Data dataset is the high-momentum calibration dataset; Calib MC is the simulated high-momentum calibration dataset; Calib Low P is the low-momentum calibration dataset



(a) High-momentum calibration dataset



(b) High-momentum calibration MC dataset

Figure 7.8. ROC curves for different algorithms. Reproduced from [267]. Private run.

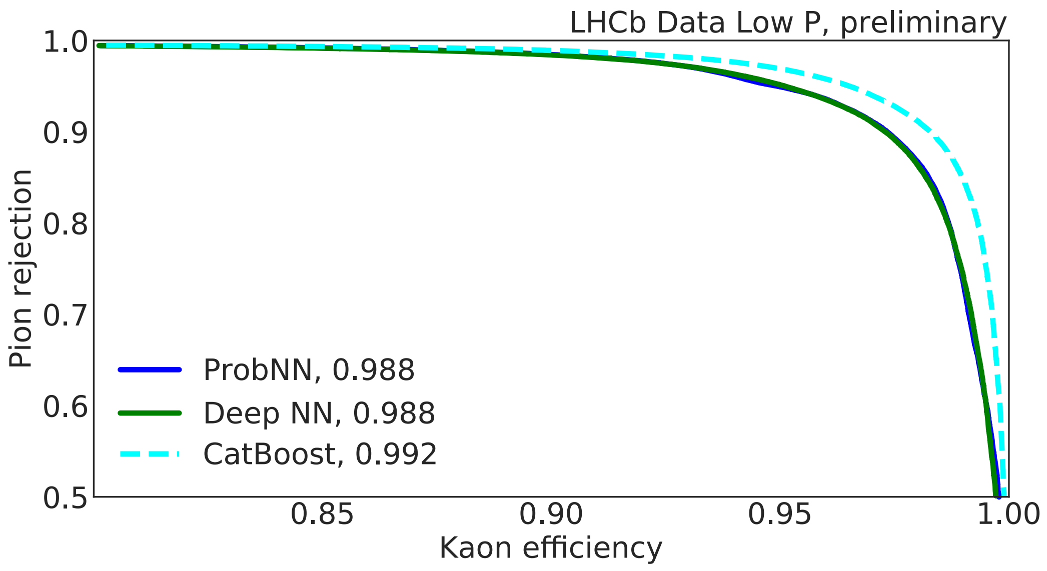
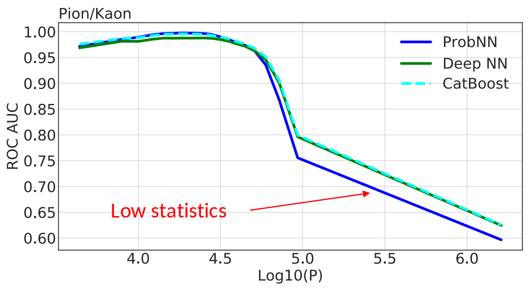
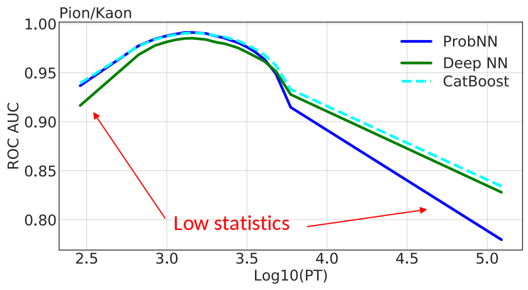


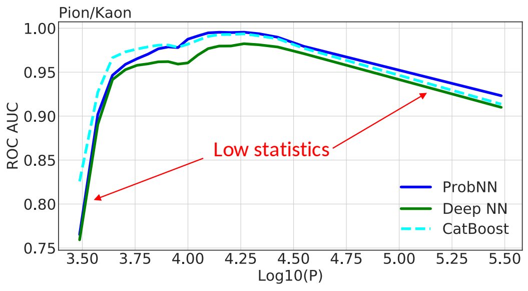
Figure 7.9. ROC curves for different algorithms for the low-momentum calibration sample. Reproduced from [267]. Private run.



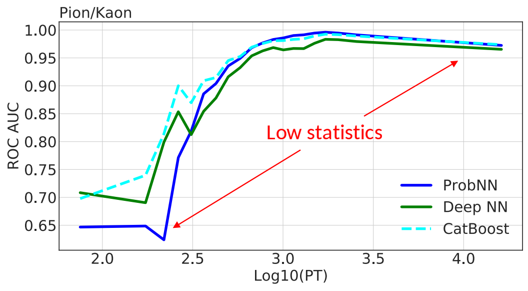
(a) ROC AUC values as a function of momentum for different PID algorithms on the high-momentum calibration sample. Reproduced from [267]. Private run.



(b) ROC AUC values as a function of transverse momentum for different PID algorithms on the high-momentum calibration sample. Reproduced from [267]. Private run.



(c) ROC AUC values as a function of momentum for different PID algorithms on the low-momentum calibration sample. Reproduced from [267]. Private run.



(d) ROC AUC values as a function of transverse momentum for different PID algorithms on the low-momentum calibration sample. Reproduced from [267]. Private run.

Figure 7.10. ROC AUC values as a function of momentum and transverse momentum.

Chapter 8

Fast Simulation of the Cherenkov Detector

New runs of the Large Hadron Collider and the next generation of colliding experiments with increased luminosity will require an unprecedented amount of simulated events to be produced. This would bring an extreme challenge to the computing resources. Thus new approaches to events generation and simulation of detector responses are needed. Cherenkov detectors, being relatively slow to simulate, are well suited for applying recent approaches to fast simulation.

In section 8.1 we discuss the applications of simulation of physics experiments. In section 8.2 on overview of the LHCb simulation methods is presented. In section 8.3 we review existing techniques of improving the simulation speed. In section 8.4 we present my pilot study on using machine learning to build a fast simulation of the BaBar Cherenkov detector. In section 8.5 we present the framework that the LHCb experiment is developing for fast simulation.

8.1 The Role of Simulated Data in High-Energy Physics Experiments

Conducting a scientific experiment usually means testing a hypothesis. To do so, we must construct an experiment and formulate what would its results look like if the hypothesis is true and if it is not true. For simple enough experiments, that can be done analytically. For example, the groundbreaking Davisson–Germer experiment proving electron diffraction, required checking whether the resulting pattern confirms to the formula derived from the Bragg's law.

Measuring more fine effects requires more sophisticated experiments, and, at some point, it becomes impossible to define how the results should look like, using only hand-derived mathematical formulas. The prime examples of this trend are the high-energy physics experiments at CERN. In their case, computed simulation must be used to establish a link between the physical processes of interest happening inside the detector and its response. The situation is complicated further by the objectively random nature of quantum mechanics. The applications of the simulation methods can be broadly categorised into two groups: detector design and data analysis.

8.1.1 Detector Design

Not all experiments are born equally sensitive to new physics. In order to create a sensitive installation, a good understanding and validation of its properties are required. For a modern complex instrument, the only way to acquire that understanding is simulating it. Practically every part of the LHCb experiment has been simulated prior to commissioning [11].

8.1.2 Data Analysis

A typical analysis in high-energy physics is the measure of some signal yields. The most effort goes into designing a procedure to select the relevant signal, applying it to the data gathered by the detector, and estimating the precision of the selection.

The selection procedure can be separated into two stages: the common reconstruction and the signal selection. Reconstruction transforms the raw detector readout into the tracks and vertices and labels the particle types. Simulated data is used to train some of the algorithms at this stage, for example, see LHCb tracking in subsection 4.2.1 and LHCb particle identification in chapter 7. Simulated data are also used for measuring the performance of the various aspects of reconstruction [11, 268].

Signal selection algorithms use the reconstructed events and decide whether they contain the signal of interest. Simulated data is foremost used to study the signal efficiency and the background rejection of the selection. Simulation is essential for modelling the background for the cases where data-driven selections are not available (for example, combinatorial background in [190]). If machine learning is used for building the selection operator, it is typically trained using the simulated signal, for example, see the cases in chapter 3.

A large amount of simulated data is needed due to two factors. First, the simulated data needed for signal selection is generally unique per analysis. Second, the limited number of simulated events translates into systematic uncertainty of any result obtained from them. For example, the statistical uncertainty of signal efficiency estimation will directly become systematic uncertainty of the measured branching fraction. Therefore, in most cases, it is desirable to have at least the same number of simulated signal events as there are real data.

8.2 Simulation in LHCb

Simulation at LHCb is implemented in the Gauss package [14]. The structure of the Gauss software is presented in figure 8.1. The simulation consists of two major steps [269]:

1. Simulation of the processes occurring in the proton-proton collision, until all products are stable/quasi-stable particles. It is typically split into phases: matrix elements calculation, parton shower, and hadronisation. LHCb uses Pythia [142] for this purpose. Being a b-physics experiment, LHCb needs a very detailed simulation of B decays. For this purpose, LHCb uses EvtGen [141] for simulating time evolution of the produced particles. EvtGen was

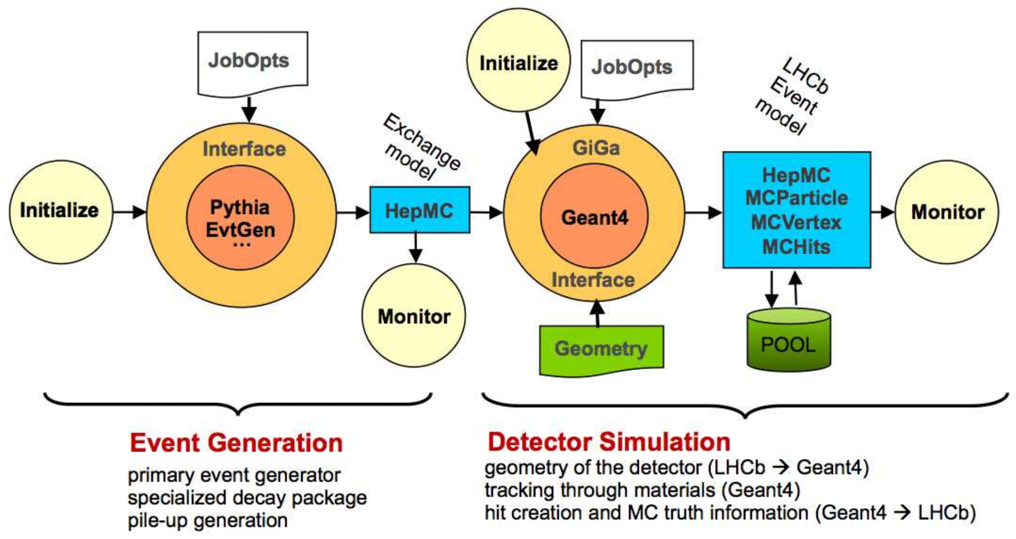


Figure 8.1. The scheme of the Gauss software. Reproduced from [270].

originally developed for the BaBar experiment and then further customised to handle incoherent B^0 and B_s^0 production. At this stage, a sample generation tool is employed. It checks whether the produced event adheres to the specified requirements. There are several categories of events, depending on the simulation objective [270]:

- Minimum bias events do not impose any additional requirements.
- Inclusive: events containing at least one particle from the specified list. The most common case is to check for beauty and charm hadrons.
- Signal: events containing at least one signal particle. At least one of the signal particle is forced to decay to a predefined decay mode. For rare signal hadrons, for example, B_s^0 compared to B^+, the following technique is used: when an event containing a b-quark is obtained, it is re-hadronised until the desired B-meson is produced
- Special: events defined with special generator settings, usually containing processes with very low cross-sections, such as Z_0 production.

2. Simulation of the interaction of the produced particles with the detector is handled by Geant4 [271]. It propagates the particles and outputs hits in the sensitive parts on the detector along with the Monte-Carlo truth information about the particles.

The simulated data is then processed through the same reconstruction and selection pipeline as the real data.

8.2.1 Technical Improvements to Full Simulation

The Gauss framework was written 15 years ago and processes the events in a single thread. The parallelism is achieved by launching multiple copies of the application as different processes. Nowadays, this is not sustainable, as the number of CPU cores is growing more rapidly than the available memory. For example, when the LHCb trigger farm is used for Monte-Carlo production, it is not possible to utilise all the available CPU cores [272]. To resolve this, the LHCb collaboration is moving to multithreaded event processing. It allows to reduce the memory footprint by not duplicating in RAM the data, that is common to multiple events, such as the detector geometry.

8.3 Fast Simulation

During the LHC Run II approximately 75% [273] to 90% [274] of the offline computing resources of the LHCb collaboration were used for Monte Carlo simulations. This is also the case for many other high-energy physics experiments [275]. Despite the dedicated resources, an insufficient amount of simulated data is a major source of systematic uncertainty [276]. The planned trigger purity increase and a five-fold luminosity increase for the LHCb Run III are not accompanied by a corresponding increase in the computing budget. Therefore, to continue to have the number of simulated events comparable to the number of real data events, faster simulation methods are needed. The projected CPU requirements and the pledged resources are presented in figure 8.2.

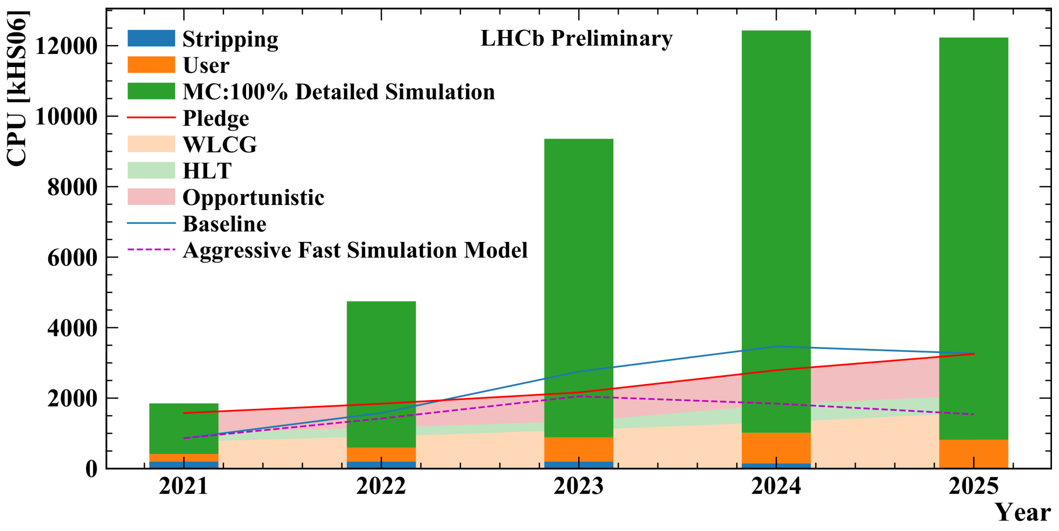


Figure 8.2. Projected LHCb computing needs breakdown by category. If the full simulation is utilised, the MC requirements (green bars) are several times the pledged resources (red line). Fast simulation, if aggressively adopted, will allow fitting into the budget (dashed magenta line). Reproduced from [277].

Simulating the interaction of the particles with the detector takes 95% – 99% of the total computation time [272] and scales linearly with the event multiplicity. In some cases, the simulation precision requirements can be relaxed: evaluating systematic uncertainties, detector prototyping, phase space scans. In some analysis, it can be acceptable to use a rough model to model some samples, e. g. [278, 279]. For these applications, the simulation speed can be increased at the price of reduced fidelity.

8.3.1 ReDecay

When a signal MC event is generated the most, and sometimes the only, interesting particles are the ones belonging to the signal decay. This opens a way towards optimisation. Naively, we might simply generate only the signal decay, without bothering with the rest of the event. While this provides a dramatic speed-up, it disregards the occupancy effects and overestimates the efficiencies. A development of this intuition is the ReDecay method [272].

1. A full Monte Carlo event, including the signal decay, is generated
2. After generating the particles (Pyhtia+EvtGen stage), but before simulating the detector interaction (Geant4 stage), the signal particle and its decay products are removed from the event. The kinematic parameters of the signal particle are stored.
3. The rest of the event (ROE) is simulated, the information about the true particles and the detector energy deposits is stored.
4. The following is repeated several times:
   (a) Some decay of the signal particle with the stored kinematics is simulated
   (b) The detector energy deposits for the ROE and signal decays are merged

This approach provides resolutions and efficiencies equal to a normal simulation. The main difficulty is that it violates the commonly used property, that the events are independent of each other. The effect of these correlations depends on the studied observables. In the case of a significant impact, analysis is greatly complicated, and the analysts must either use some sort of parametrisation of the correlation effect or have to resort to full simulation.

The ReDecay approach improves the simulation speed by a factor of 20-50 while providing high-fidelity signal simulation. It has been widely adopted in the LHCb collaboration, see the figure 8.3.

8.3.2 Parametrisation and Simplification

Since the most expensive part of the simulation is the computation of the particles' interaction with the detector, a natural thing is to try simplifying the model and replacing ab initio calculations with parametric estimates. This can be done at different levels, depending on the accuracy and timing requirements. To achieve the optimal result, extensive knowledge of the detector and the desired use cases is required.

8.3.2.1 CMS Fast Simulation

A prime example of this approach is CMS fast simulation [280, 281, 282]. It provides speed up by two orders of magnitude compared to the full simulation. Its main trade-offs are:

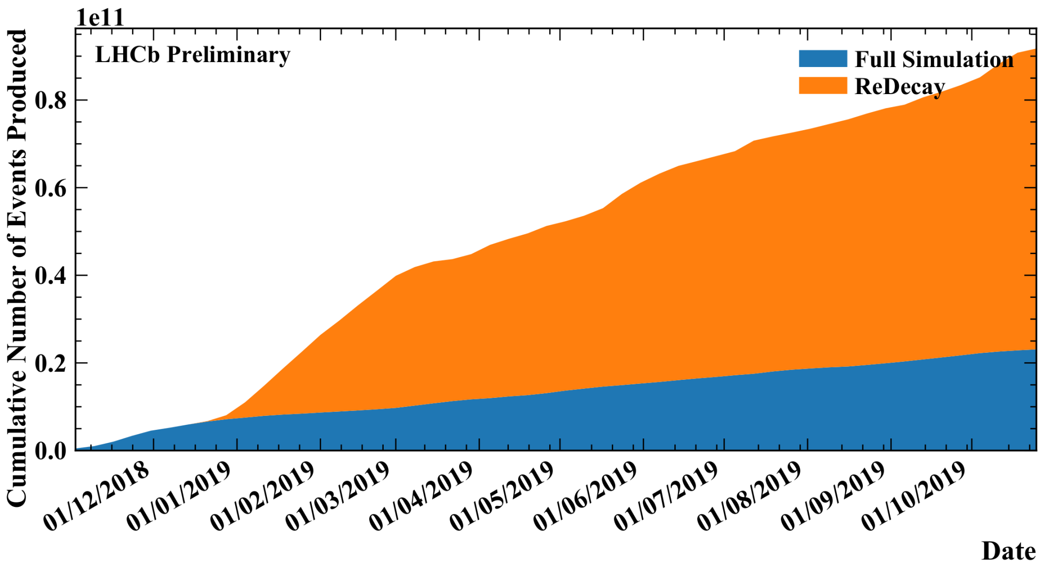


Figure 8.3. ReDecay adoption an LHCb. Reproduced from [277].

- simplified geometry;
- material layers thickness is neglected, the difference of the simulated material thickness is presented in figure 8.4 for the tracker;
- material interaction is parametrised by analytical formulas;
- simplified tracker emulation.

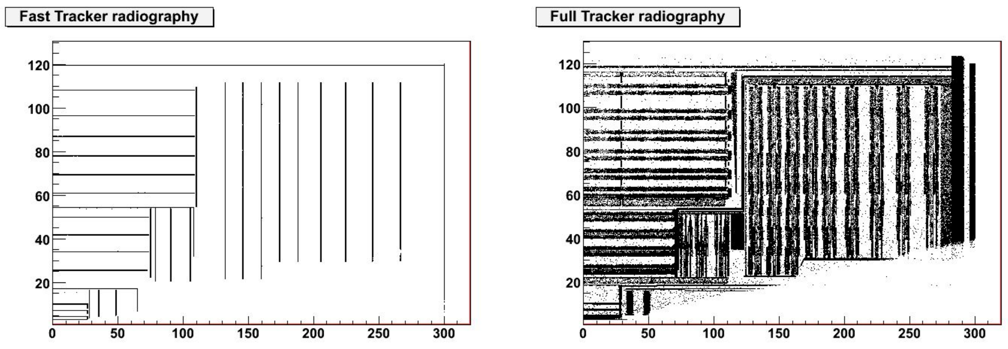


Figure 8.4. A radiography of a quarter of the simulated CMS tracker geometry in the fast (left) and full (right) simulation. Reproduced from [283].

The fast simulation is used for a “simplified model scan” [284] signal in most of the CMS supersymmetry (SUSY) searches. It is also used for evaluating the systematic uncertainties by producing samples with varying parameters of $t\bar{t}$, single top, W^+ jets, Z^+ jets [283].

8.3.3 CaloGAN

One of the first applications of machine learning for simulating a high-energy physics detector was the calorimeter simulation proposed in references [285, 286]. High-fidelity simulation of the interaction of a high-energy particle with a calorimeter requires simulating the evolution of particle showers. This can be computationally expensive [287, 283].

The objective of CaloGAN is to simulate the component readouts in an electromagnetic calorimeter conditioned on the incident particle type and energy. The authors consider a simplified 3-layer model of a calorimeter, that is presented in figure 8.5. The authors use a complicated architecture to provide internally-consistent

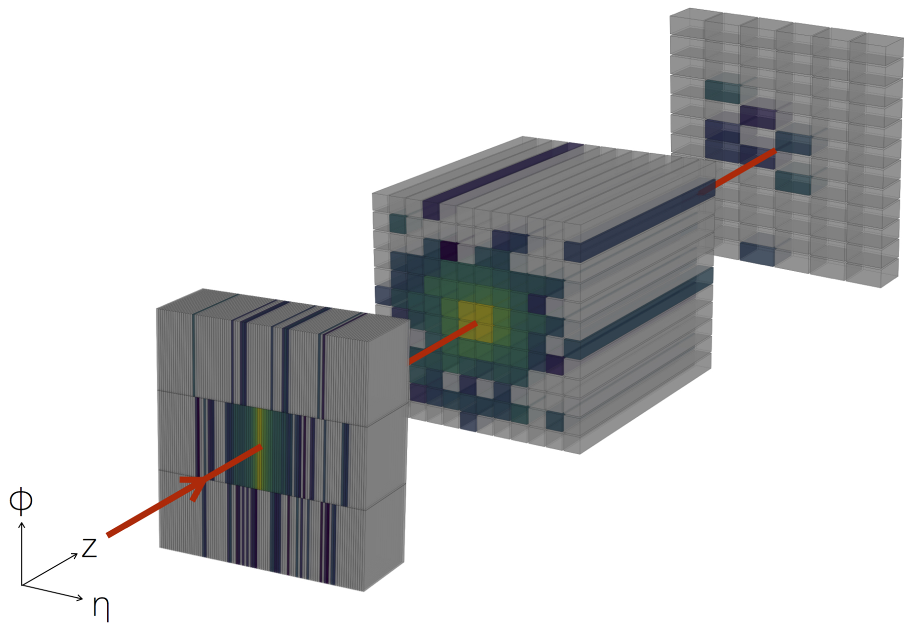


Figure 8.5. Tree-dimensional representation of a 10 GeV e^+ incident perpendicular to the centre of the detector. Not-to-scale separation among the longitudinal layers is added for visualisation purposes. Reproduced from [285].

three-dimensional output. While individual simulated events look realistic, the overall model fidelity is lacking, when considering the reproduction of distributions of high-level physically-meaningful quantities [285]. As of the moment of writing (April 2020), I am not aware of GAN-based calorimeter simulation being used for physics analysis. There is work in this direction in the ATLAS [288], CMS [289], and LHCb [290] experiments.

8.3.3.1 Fully Parametric Simulation

CMS fast simulation reduces the cost of detector simulation from 8x the cost of reconstruction to 0.3x [283]. Therefore, if further speed-up is required, the reconstruction must also be included. This leads to a fully parametrised simulation. Its main idea is mapping from the particle-level simulation directly to high-level observables, bypassing the energy deposits in the detector and the reconstruction algorithms. A prime example is Delphes [291], which we briefly describe next.

Delphes simulates track propagation in a magnetic field, electromagnetic (ECAL) and hadron (HCAL) calorimeters, and a muon identification subsystem. It outputs the tracks and calorimeters energy deposits, along with the high-level observables, such as isolated electrons, jets, taus, and missing energy. It can handle the pile-up.

The particles, whose lifetime allows them to reach the detector, are propagated through the magnetic field, which is assumed to be parallel to the beam direction and localised in the inner tracker volume. The charged tracks are reconstructed with the user-specified efficiency. Smearing on the norm of the transverse momentum is applied. The resolution is user-specified.

Then the particles reach the calorimeters. The calorimeters have a finite segmentation in pseudorapidity and azimuthal angle. The cell size is again user-specified. The particles deposit a fixed fraction of energy in the ECAL and HCAL. The calorimeters' resolutions are parametrised by a simple analytical function of the energy and pseudorapidity. The energy deposits are smeared by a log-normal distribution.

Particle identification is likewise parametrised:

- **Muons** are reconstructed with some efficiency. The efficiency tends to zero outside the tracker acceptance and for momentum below some threshold. The final muon 4-momentum is smeared by a Gaussian distribution with a user-supplied resolution.

- **Electrons** are recognised with efficiency, that is a function of the energy and pseudorapidity. As with muons, the efficiency vanishes outside the tracker acceptance and below some energy.

- **Photons** reconstruction relies on ECAL. True photons and electrons with no reconstructed track that reach the ECAL are reconstructed as photons in Delphes.

The LHCb collaboration began adapting Delphes to its needs, but in 2019 two things became evident. First, only a small fraction of Delphes package was really useful for the LHCb purposes. Second, the person-power needed to interface and maintain the joint LHCb-Delphes codebase was high. For these reasons, an in-house parametrisation tool has been developed, called Lamarr [277]. It is described in section 8.5.

A parametric simulation is only as good as the parametrisations inside it. In principle, full simulation allows creating arbitrarily detailed parametrisation, as it gives access to all the variables. The limiting factor is the multivariate analysis. Simple analytical formulas and histograms do not scale well beyond 2 dimensions. They are not applicable, if, for example, we want to model the particle identification variables, one for each particle type. Machine learning-based models described in section 2.7 offer a promising way to address these concerns. In the next sections, I describe a pilot work I did on machine learning-based parametric simulation of a high-energy physics detector and the proposed model to be integrated into the LHCb data processing.

8.4 Pilot study: BaBar DIRC

8.4.1 DIRC detector

Cherenkov detectors present a ripe target for parametric simulation. Propagating photons through media requires expensive computation. Simulating the Cherenkov detectors takes estimated 30% of the total event simulation computing time at LHCb [15]. For a pilot study, we used Generative Adversarial Networks to simulate the BaBar DIRC (Detection of Internally Reflected Cherenkov light) [292]. The choice was dictated by the availability of the open-source FastDIRC software [293], which allowed generating private Monte-Carlo in a resource-efficient way.

The overview of the BaBar detector is presented in figure 8.6. It consists of a magnet, trackers, a calorimeter, Instrumented Flux Return (IFR), and the DIRC. The DIRC detector and its operating principles are described in figure 8.7.

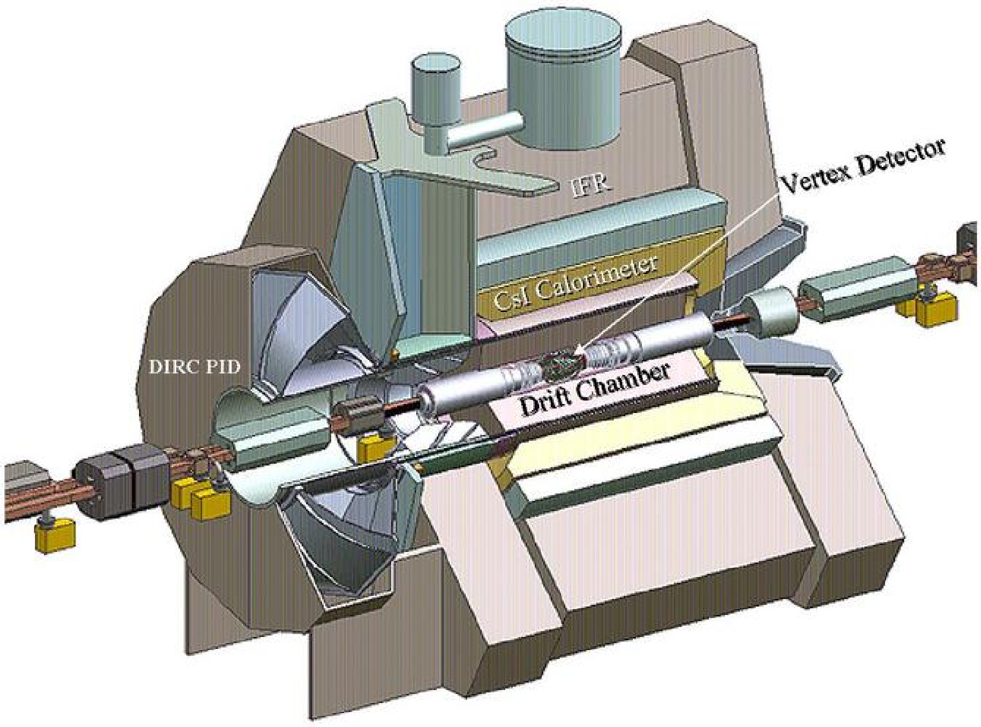


Figure 8.6. An overview of the BaBar detector. Reproduced from [294].

To generate the event samples for model training and validation, we used the open-source FastDIRC [293] software. The generation has two stages. First, for

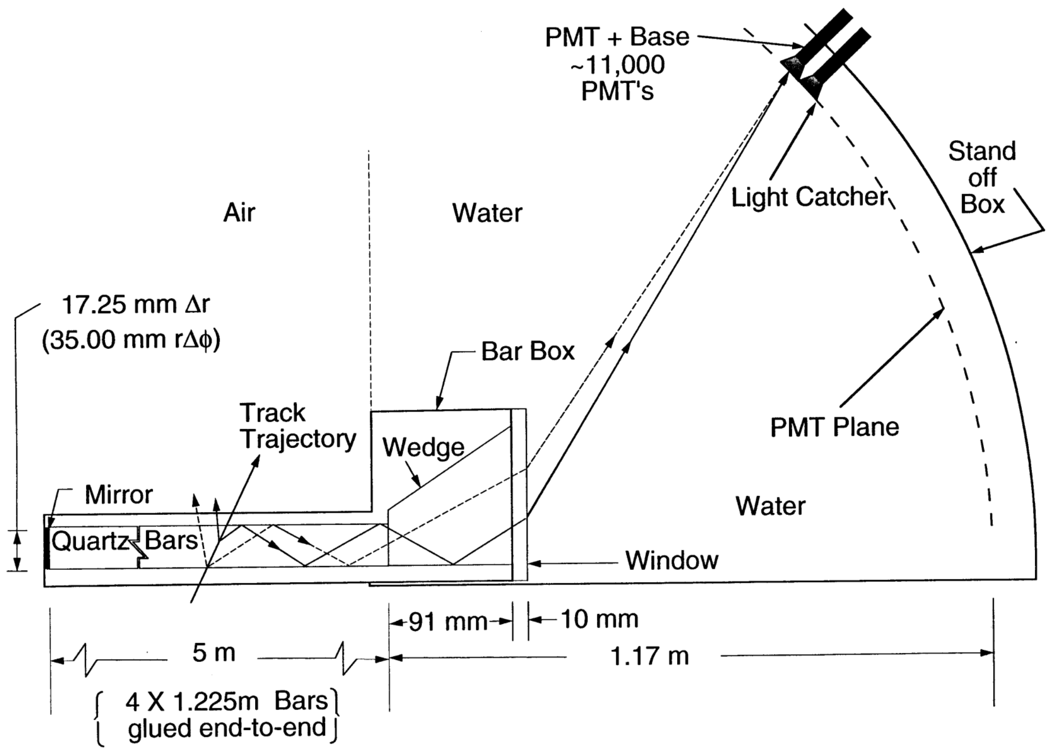


Figure 8.7. The DIRC detector consists of fused silica bars arranged in a 12-sided polygon barrel-like formation around the interaction region. This figure presents a transverse view of one side of the barrel. Cherenkov light is emitted by the charged particle traversing it and is transferred to the water tank via total internal reflection. The light is then observed by an array of photomultiplier tubes (PMT). Reproduced from [295].

the given particle parameters, a large amount of the photons for each particle type hypothesis is generated. The detector responses are fitted with kernel density estimation to produce the probability density functions corresponding to different particle types. Second, a realistic number of photons with parameters corresponding to the true particle type is generated, and the previously stored probability density functions are used to compute the likelihoods of different particle type hypotheses.

The tracks have a flat distribution in pseudorapidity between -1.5 and 1.5, and a Gaussian distribution in energy (mean 6 GeV, width 2 GeV). There is an additional cut that excludes the tracks with energy less than 2.5 GeV. We simulate two-particle events. For one of the particles, which we call signal, we run the PID computation procedure described in the previous paragraph. The second particle, which we call background, serves to add the noise photos that spoil the accuracy of PID for the signal particle. In its base version FastDirc only simulates a single track, we modified it to support multiple-track events.

8.4.2 Our model

The objective of the parametric simulation is a model, that would simulate the response of the DIRC detector to a given particle. Its inputs are the signal particle type and the energies, pseudorapidities and the transverse coordinates of both particles in the event — 7 variables in total. Its outputs are the delta log-likelihoods with pion hypothesis for the electron, kaon, muon, proton and "beyond threshold" particle type hypotheses — 5 variables.

The number of dimensions (7 input and 5 output) makes the typically used parametrisation methods, such as the histograms and kernel density, unfeasible. To parametrise the distribution, we used the Cramer (energy) Generative Adversarial Networks introduces in the section 2.7. The number of input observables and the architecture of the neural network was optimised to obtain a sub per cent quality of the prediction. The final architecture design uses fully connected neural networks with 10 layers, each containing 128 neurons with ReLu activation for both generator and discriminator. We trained a separate model for each signal particle type. Each model was trained using 1 million generated events. We transformed each observable distribution into a Gaussian using quantile transformation before passing them to the neural network. We started with traditional Jensen-Shannon GAN, then switched to Wasserstein GAN with gradient penalty, as it provides stable training largely irrespective of the hyperparameters. Finally, we switched to the Cramer (energy) GAN; this allowed for a small increase in precision, likely due to the unbiased gradients estimates advertised in the original paper [139].

8.4.3 Evaluation Results

We first check that the predictions are consistent with our expectations for one particle tests: we check one- and multidimensional distributions of the output likelihoods in order to understand whether they are consistent with the output of FastDIRC. We find the histograms to be in good agreement. The plots are shown in figure 8.8.

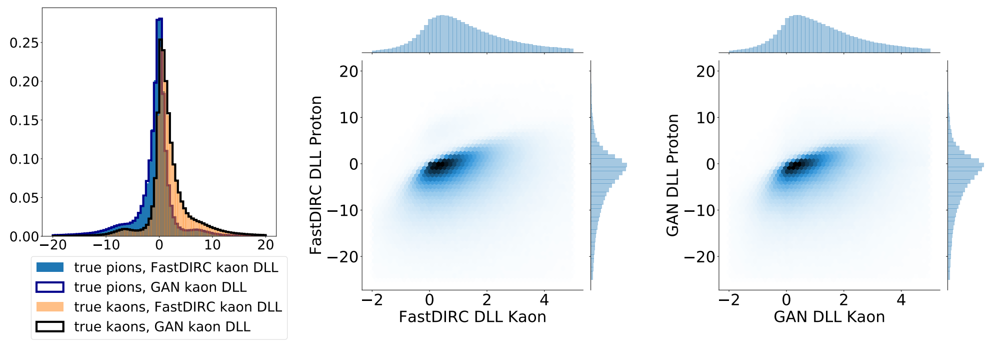


Figure 8.8. Left: An example of 1D projection to kaon delta log-likelihood observables for FastDIRC and GAN simulation for samples consisting true pions (blue) and true kaons (brown). Center and right: An example of 2D projection to kaon and proton delta log-likelihood observables for FastDIRC (left) and GAN (right) simulation of a pion sample. First published in [296].

One of the issues of high-level observable generation is the influence of other particles in an event. The issue is not especially severe, as general occupancy effects are implicitly factored into the probability distributions. But it might come up in the case of a signal signature containing particles with close tracks which will interfere with the detector readings of each other. We investigate the ability of a parametric simulation to model these effects. To do so, we add the information about the kinematics of the background particle into input observables list. A parametric model that successfully models the interference effects should preserve the dependency of the PID variables on the background particle parameters. In figure 8.9 we demonstrate this for the dependency of the ROC AUC between pions and kaons on the pseudorapiditis.

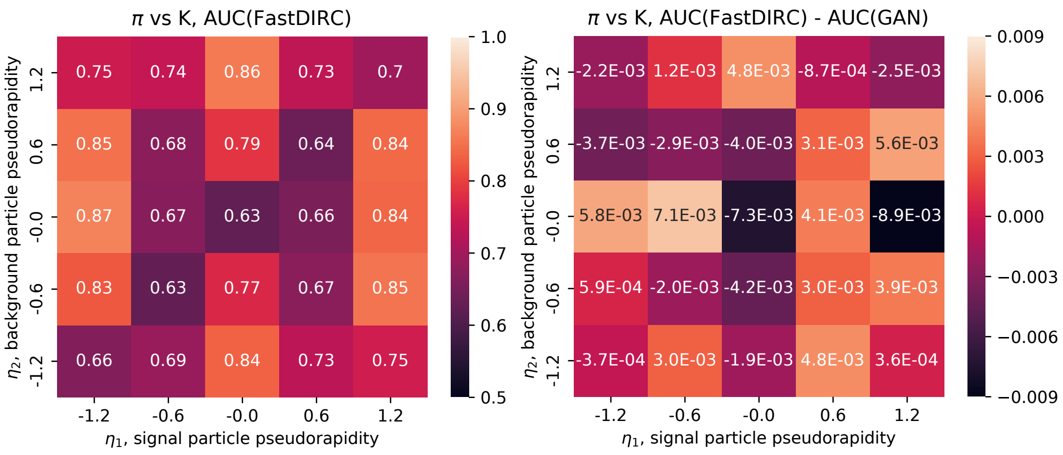


Figure 8.9. Separation power between kaons and pions measured as the area under the receiver operating characteristic curve (ROC AUC). Left is the FastDIRC simulation; right is the difference between GAN and FastDIRC AUC scores. The statistical uncertainty is around 0.005. First polished in [296].

The obtained generation model is lightweight. The speed improvement with respect to the full simulation in GEANT 4 [271] is $8 \cdot 10^4$ times on a single CPU core. The speed is also improved with respect to the FastDIRC generation, where a factor up to 80 can be achieved. The batch generation on GPU produces up to 1 million track predictions per second.

The main conclusion of the pilot is that the GANs provide a fast and accurate way modelling probability distributions and thus make an ideal model to be used for a fully-parametric simulation. The hardware of the LHCb RICH and BaBar DIRC detectors is quite different. Still, this difference is abstracted away as we deal with the high-level PID variables of similar dimensionality. The main distinction is the complexity of an LHCb event which makes it impossible to include all the tracks as model inputs.

8.5 Fast Parametric Simulation at LHCb (Lamarr)

The LHCb collaboration is developing an application for parametrised simulation of the detector response. The main characteristics of Lamarr are described in the following. As of the moment of writing (April 2020), the work is in progress and so details are changing fast.

The structure of the PID part of the Lamarr application is presented in figure 8.10. As its input, it takes the particles generated by the Gauss package, introduced in section 8.2. The propagation is made with a simple transport into a dipole field

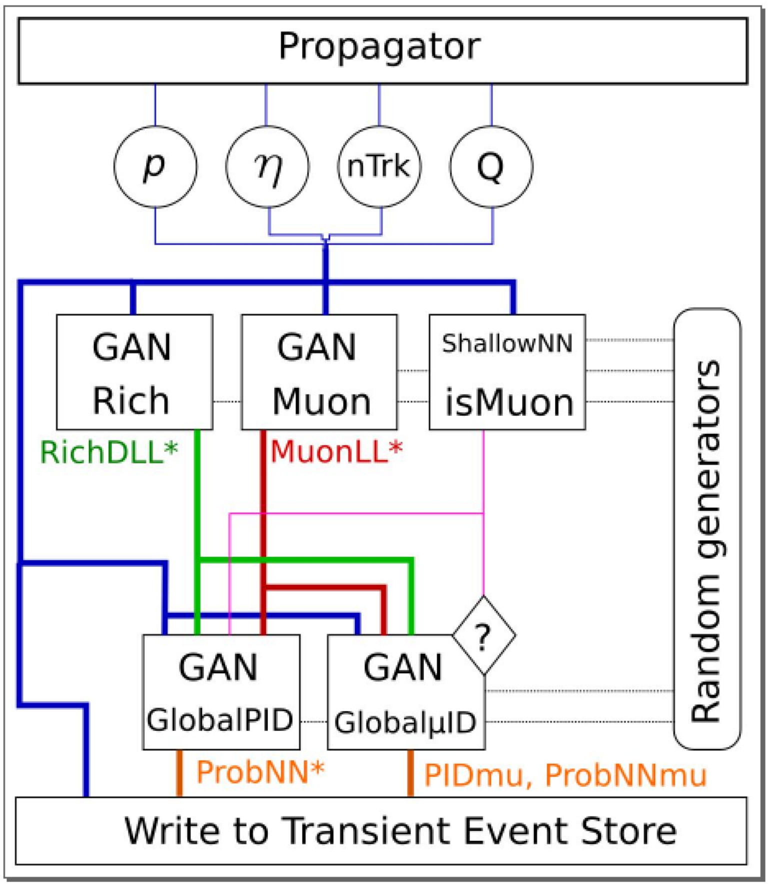


Figure 8.10. The architecture of the Lamarr application. Reproduced from [297].

assuming a mean value for the shift of the transverse momentum component of the particles [298]. The point where the bending is applied is parametrised as a function of the inverse of the momentum. The propagator performs acceptance checks before and after the magnet. For tracks that are in acceptance, tracking resolution and efficiency are emulated according to the measurements on the full LHCb simulation. Track reconstruction efficiency is modelled by discarding the track with the probability dependent on its parameters. Resolutions effects are modelled by applying Gaussian smearing to the track parameters.

For Lamarr, the goal for the simulation of the charged particles PID is learning the distribution of the analysis-level PID variables conditioned on the tracks kinematics and the particle types. There are two important questions when designing a parametric model:

- Which parameters to use?
- Which data to use for training the model?

And there are two possibilities. The first is to train on the full simulation; the second is to use the calibration samples described in subsection 4.3.7. In the full simulation, the Monte-Carlo truth is available, allowing to condition the PID response on the (simulated) physical tracks, not the reconstructed ones. Simulation can be made without the selection bias; this allows the free choice of the parameters, up to conditioning all the PID variables on the whole event. But, of course, simulation produced in this way can only be as accurate as the full simulation. And the full simulation of the PID detectors has insufficient precision for many analyses. This necessitated the development of a data-driven way of measuring the PID requirements efficiencies based on the calibration samples – the PIDCalib package [22].

Training on real data presents two challenges. First, the set of parameters must be carefully selected to avoid the bias caused by selection, while still being rich enough to capture the essential dependencies. Second, the real data is contaminated by background. Both of the issues have been addressed by the LHCb collaboration developing the PIDCalb project. A reliable set of parameters consists of the track momentum, pseudorapidity and the number of reconstructed tracks in the event. The background is accounted for using the sPlot technique, described in section 6.1. Overall, this provides the same setup as was used in the pilot study described in section 8.4.

The PID is simulated with 4 GANs, for RICH, muon likelihood (with and without `IsMuon` requirement enforced), and global PID. The global PID GAN is conditioned on the output of the subsystem-level GANs. The 2016 calibration samples are used to train the models:

- $D^{*+} \to D^0\pi^+$ with $D^0 \to K^-\pi^+$ for pions and kaons;
- $\Lambda^0 \to pK^-$ for protons;
- $K^0_S \to \pi^+\pi^-$ for pions;
- $D^+_s \to \phi(1020)\pi^+$ with $\phi(1020) \to K^+K^-$ for kaons, different momentum distribution with respect to the ones from D^0.

The following section describes in details the GANs developed for simulating the LHCb RICH.

8.5.1 RICH Fast Simulation

The FastDIRC pilot was designed with the simulating the LHCb RICH in mind, so the problems are very similar. The model consists of 5 GANs: one for each particle type. The GAN architecture is presented in figure 8.11. The model input parameters are the ones used in PIDCalib: momentum, pseudorapidity and the number of reconstructed tracks in the event. The model outputs are `RichDLL*` – the delta log-likelihoods between a particle type hypothesis and the pion hypothesis. The model is trained on the 2016 calibration samples. The calibrations samples use the sPlot method to remove the influence of the background. The sWeights are applied to the loss function. This did not cause the problems like the ones we describe in chapter 6 for the classification models. A possible explanation is that during a GAN training the amount of generated data is effectively infinite, which prevents catastrophic overfitting from occurring, despite the finite real sample.

Both generator and discriminator consist of 10 fully connected hidden layers with 128 units in each, with rectified linear unit (ReLU) activation functions. The latent space dimensionality for the generator is 64, and the distribution is the standard normal random vector $\{N(0,1), ..., N(0,1)\}$. The GANs are conditional: the latent variable concatenated with the input parameters.

The output dimensionality of the discriminator network is 256. The output layers of both generator and discriminator do not use activation functions.

We use the quantile transformation to transform both features and target variables distributions into standard normal. For our dataset, this results in faster convergence and higher output fidelity, than the commonly used linear scaling. We use exponential learning rate decay.

On a single CPU core the proposed model is at least 2 orders of magnitude faster that the full simulation with Geant4.

8.5.2 Preliminary Evaluations

The most basic requirement for the model is a faithful reproduction of the underlying distributions. To verify this, we check the 1D distributions of the `RichDLL*` variables, they are presented in figure 8.12. The marginal momentum bins contain smaller statistics and therefore show slightly worse generation quality, while the central bins show quite good correspondence to the real data.

The objective of the PID variables is distinguishing the particle types. This leads to a natural quality metric: checking whether the discriminative performances of the real and generated variables are the same. We do this by computing the ROC AUC scores for the different particle pairs using the real and simulated `RichDLL*`. If the simulation was perfect, the difference in performance would only be caused by statistical uncertainty. A difference that can not be explained a statistical fluctuation means an imperfection in simulation. The ratio of the differences in ROC AUCs between the data and simulation to the statistical uncertainties is presented in figure 8.13. The uncertainty of the differences between AUC values was estimated using bootstrap [301]. Most of the differences are not greater than a few standard deviations, with no obviously biased regions. Overall, the proposed model shows a good approximation of the real data distributions with some imperfections in the regions with low statistics.

Since GANs are good in reproducing the distributions, it is likely, that the main source of systematic uncertainty in Lamarr will be the selection of parametrisation and the training data. To estimate them, we validate the model using a calibration decay, that was not used for the model training: $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$, $\Lambda_c^+ \to pK^-\pi^+$. In figure 8.14 the invariant mass of the Λ_c^+ baryons produced in the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ is presented. The invariant mass is mean is reproduced accurately, while the peak width is 25% narrower than the reference. In figure 8.15 the decay vertex χ^2 for the decay $\Lambda_c^+ \to pK^-\pi^+$ is presented. The overall shape of the distribution is reproduced, but the it is somewhat shifted towards higher values. Note, that in this preliminary evaluation tracking parameterization is based on 2012 data (Run I), while the data is from 2016 (Run II). The relevant calibration sample is not available for Run I. In figure 8.16 we present the comparison between data and Lamarr for the response of the neural network trained to identify ghost tracks, i.e.

random combinations of hits not associated to any charged particle. In figure 8.17 the efficiency of proton identification is presented.

8.5.3 Future outlook

Overall, in the preliminary tests, Lamarr shows reasonable agreement with the data. A few issues must be addressed before Lamarr can be reliably used for physics:

- As demonstrated in figures 8.15 and 8.14 there is a room for improvement in terms of tracking quality. A tracking parametrisation based on Run II data is highly likely to improve it;
- Further validation of global PID, that would include all the particle species.

Lamarr will also benefit greatly, should the LHCb collaboration decide to use GPU for trigger (subsection 4.3.6). Neural networks can run very efficiently on GPUs and thus the trigger farm would be able to produce a large number of simulated events during the time when there are no collisions.

Another question is whether the setup with multiple nested GANs is preferable to a setup with a single GAN that would output just the combined probability variables. First, having multiple GANs is slower during inference. The needed computational and memory complexity of a joint model almost certainly will be lower than for three separate models. Second, having multiple independent subsystem-level models ignores the correlation between the RICH and Muon variables. When we model the responses, we model the conditional probability densities: $P(\text{RICH DLL}|\text{track kinematics})$ and $P(\text{Muon DLL}|\text{track kinematics})$. In reality, the subdetector responses also depend on the variables, that we do not input into the parametric models, such as the distances to the neighbouring tracks. For example, two tracks might be very close together, making their PID uncertain. Even if we do not input this information into the parametric model, it can model the integral effect of the track closeness. To do so, it “tosses a coin” to decide whether the track in question has a neighbour that would smear its PID. And if the RICH GAN and Muon GAN “coins” land differently, the result would be nonphysical, with one model simulating an isolated track and the other a track whose identification is smeared by a close neighbour. The magnitude of this effect is yet to be measured, though. The advantage of having multiple GANs is that by training the GANs separately, we reduce the problem dimensionality, so each GAN individually is easier to train. Also, RICH and Muon likelihood variables are sometimes directly used in physics analysis, although rarely. Those are the cases, where some systematic effects can be studied, possibly understood and eventually corrected, only at the level of the single sub-detector.

8.6 Conclusion and outlook

Improving the Monte-Carlo simulation speed is crucial for the LHC Run 3, which will see a drastic luminosity increase. To keep under control the uncertainty caused by the simulated sample size, the simulated sample size must increase correspondingly to the collected real data sample size. The full simulation methods based on Geant 4, that were used during the Run 2 can not deliver them under the available computing budget.

There are technical improvements to simulation software, that take advantage of the latest hardware developments. The main venues are vector CPU instructions and switch from multiprocessing to multithreading, that takes advantage of the increased CPU core count per RAM trend. Yet, those improvements are likely to be insufficient.

Some methods sacrifice fidelity for speed. Straightforward ab-initio simulation in high-energy physics fundamentally contains redundant computations, as they simulate similar processes in the same detector over and over again. Capitalising on this allows making simulations 1–2 orders of magnitude faster. Some methods, that build upon this insight, sacrifice events independence by reusing large parts of the event – e. g. the ReDecay described in subsection 8.3.1. Others replace the full computation of some interactions in the detector by an empirical approximation[1].

For LHCb, we are developing a fully-parametric fast simulation framework, that covers both the interactions of the particles with the detector and the event reconstruction. For Run 2, it is trained on the calibration data.

In my part of it, I pioneered using GANs to create a fully-parameterised simulation of high-energy physics detectors, that directly outputs the high-level reconstruction variables, as opposed to the previous CALO GANs [286, 303], that output raw calorimeter response. The unique advantage of machine learning approach is that it can be easily retrained, given a model and a dataset. This is not the case for most of the other fast simulation methods.

Most fast simulation frameworks are distinct from the corresponding full simulation. A promising way forward is proposed by the GeantV [304]: creating a single modular software, that would allow the users to tune the various aspects of the trade-off for their needs. For example, for an LHCb analysis, that relies heavily on muons in the final state, it would be possible to use the rough parametric simulation of the RICH and CALO, while retaining the best possible accuracy for the Muon subdetector. An added benefit of this approach is the streamlined training of parametric models with a possibility for automation, as both model inputs and outputs will be available without the need for additional selection and preprocessing.

Looking into the future, the simulation will remain crucial for achieving the physics goals of high-energy physics experiments [275]. Various experiment-specific methods for fast simulation have been developed, but their use has not been universal. Their application poses an additional burden on the analysts who must spend the effort to evaluate the impact of the reduced fidelity. Yet, the pressure to sacrifice the simulation accuracy for speed is mounting, so their application will likely be more widespread in the years to come.

[1]One may argue that the difference is of a degree, not a kind. Geant4 also does not solve the field theory equations, but uses an assortment of higher-level approximations.

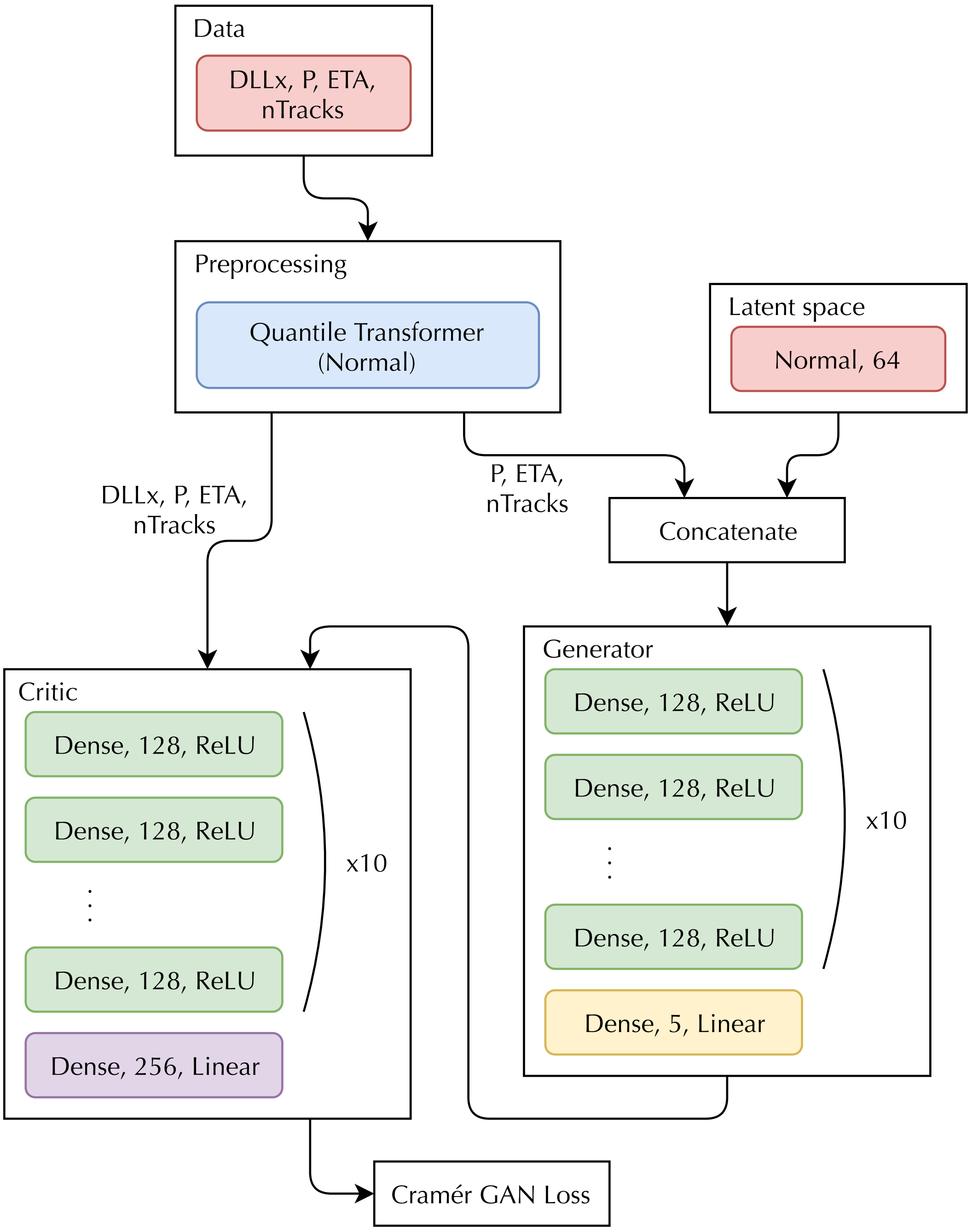


Figure 8.11. Architecture of the RICH GAN. First presented at [299].

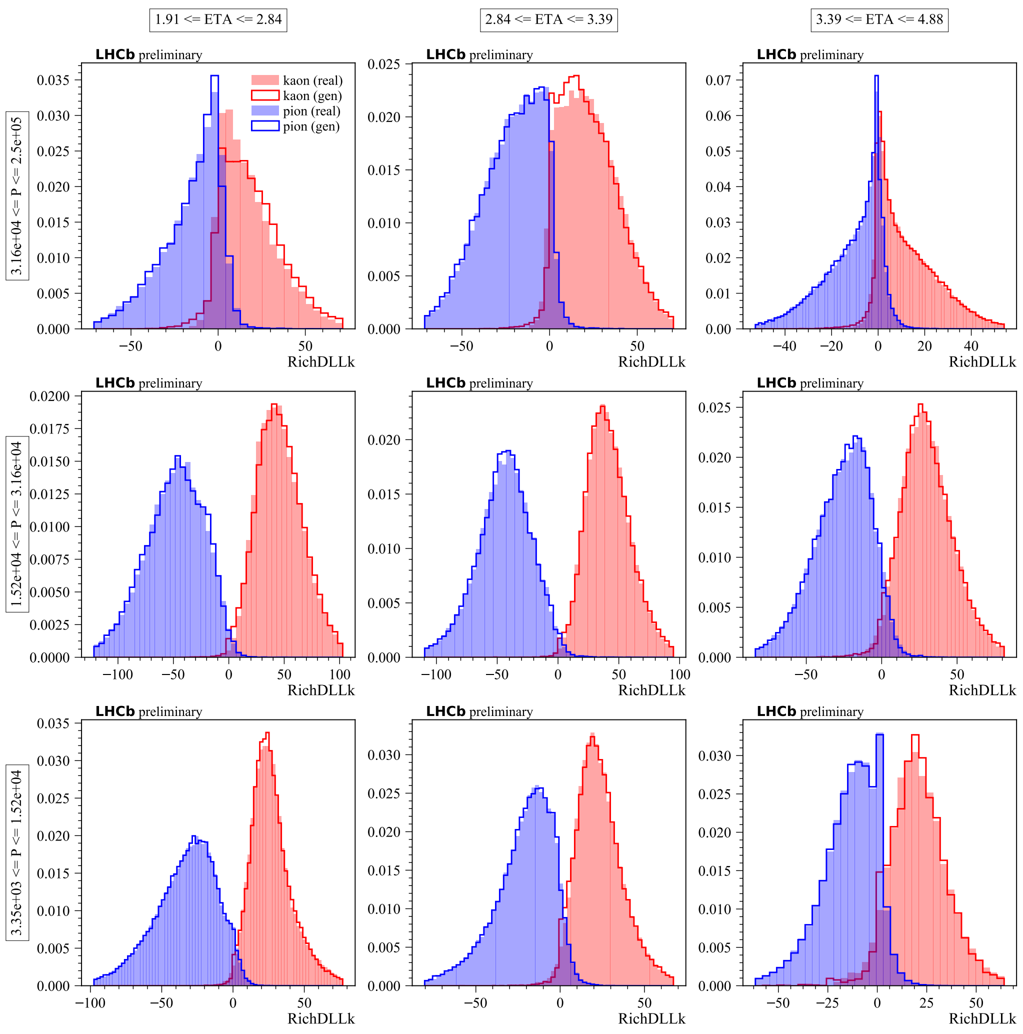


Figure 8.12. Weighted real data and generated distributions of `RichDLLk` for kaon and pion track candidates in bins of pseudorapidity (ETA) and momentum (P, MeV) over the full phase-space. First published in [300].

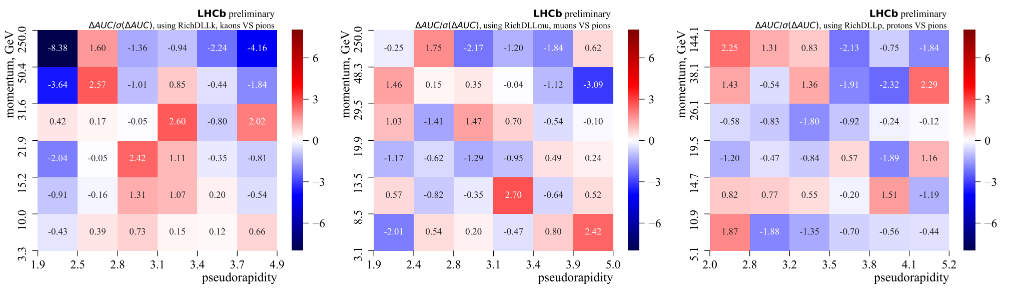


Figure 8.13. The ratio of the differences in ROC AUCs between the data and simulation to the statistical uncertainties is for discriminating respectively kaons, muons and protons from pions, classifying with the `RichDLLk`, `RichDLLmu` and `RichDLLp` variables, respectively, in bins of momentum and pseudorapidity. First published in [300].

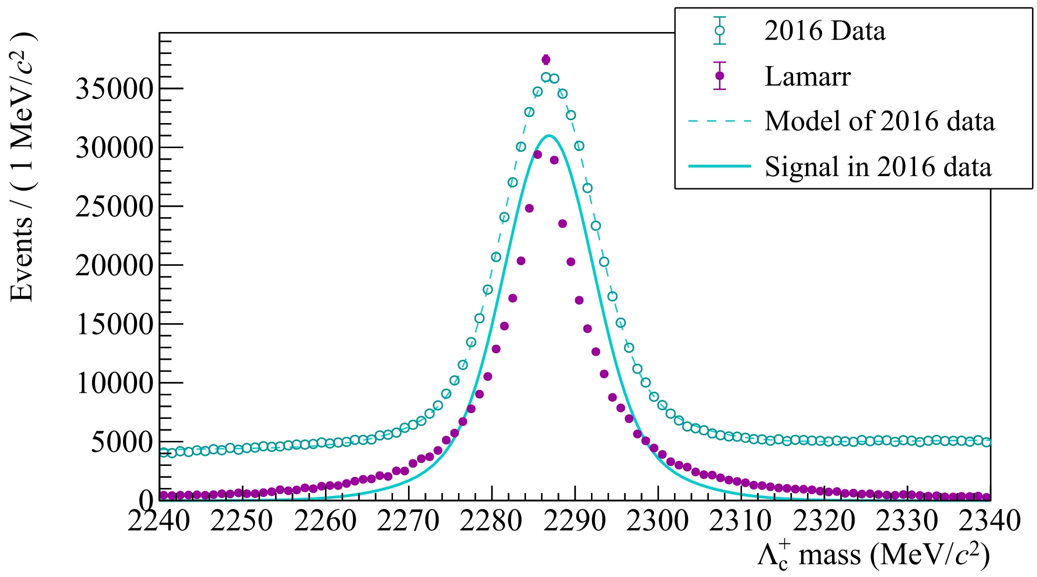


Figure 8.14. Invariant mass distribution of reconstructed and simulated Λ_c^+ baryons produced in the decay $\Lambda_b^0 \to \Lambda_c^+\mu^-\bar{\nu}_\mu$ and decays as $\Lambda_c^+ \to pK^-\pi^+$. The data is fitted with a model composed of a double Gaussian function for the signal and a second-order polynomial for the combinatorial background. The shape of the signal peak in data (solid blue line) is compared to the distribution obtained from simulation (magenta points). First published in [302].

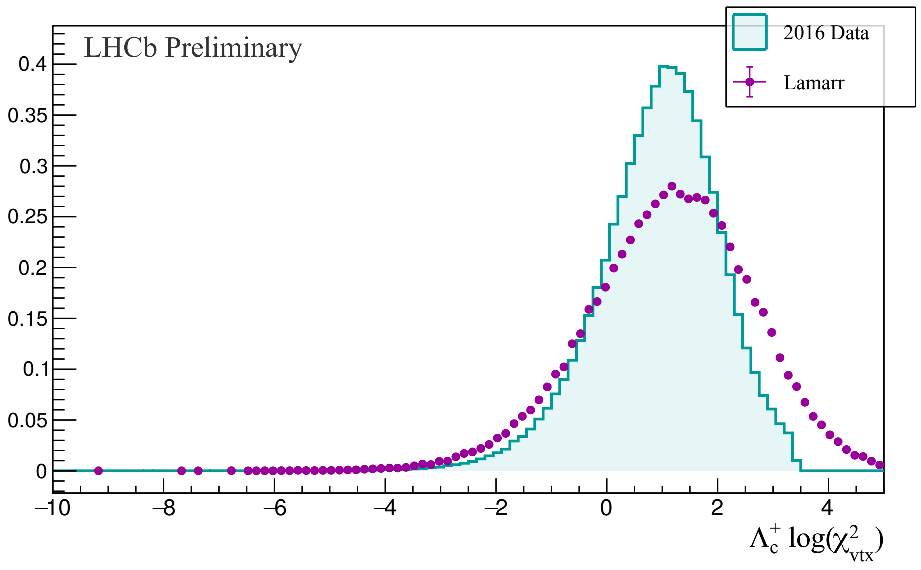


Figure 8.15. Decay vertex χ^2 for the decay $\Lambda_c^+ \to pK^-\pi^+$ of Λ_c baryons produced in the semileptonic decay $\Lambda_b^0 \to \Lambda_c^+\mu^-\bar{\nu}_\mu$. The overall shape of the distribution is reproduced, but the it is somewhat shifted towards higher values. First published in [302].

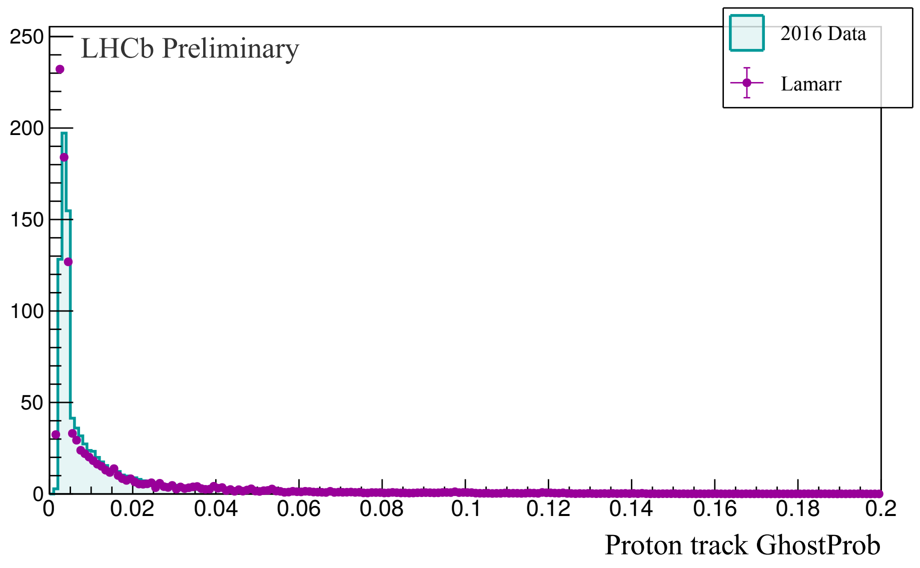
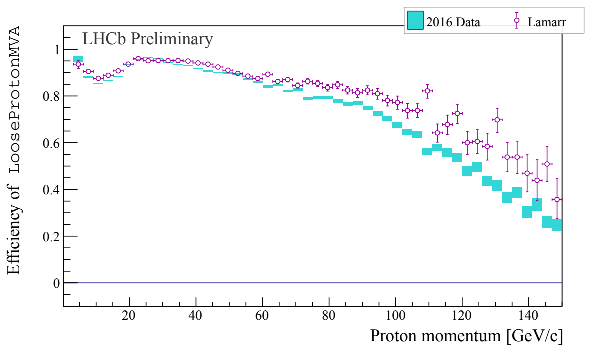
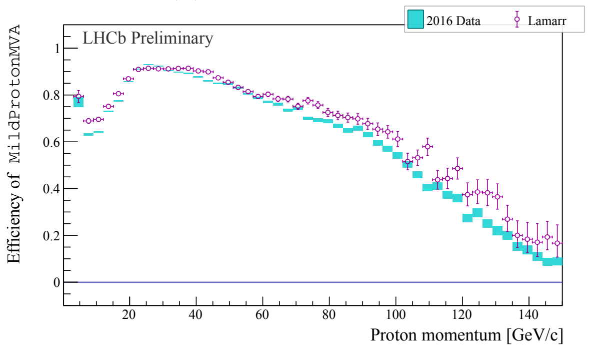


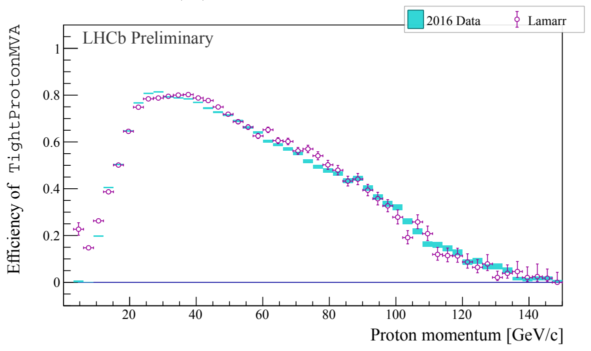
Figure 8.16. Response of a neural network trained to identify ghost tracks, i.e. random combinations of hits not associated to any charged particle. On data, the neural network is evaluated on a calibration sample of protons tagged through the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to p K^- \pi^+$. In the simulated sample the ghost probability is the response of the neural network inside Lamarr. The overall distribution shape is well-reproduce. The difference in the peak height is around 15% and is compounded by the binning effects. First published in [302].



(a) Loose selection



(b) Mild selection



(c) Tight selection

Figure 8.17. Efficiency of proton identification as a function of momentum evaluated on the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. Cyan bars are the 2016 calibration data, magenta points are the simulation with Lamarr. The three plots correspond to different decision thresholds applied to the PID response variable. For the majority of the bins the agreement is within 10%. First published in [302].

Chapter 9

Conclusion

The thesis advances particle identification at LHCb on several essential frontiers:

For muon identification, which is discussed in chapter 5, I developed a machine learning model. When evaluated on data after preliminary filtration (called `IsMuon`, described in subsection 4.3.2.1) and setting the decision threshold to maintain 90% signal efficiency (true positive rate), it reduces background passthrough (false positive rate) from 14% to 10%, compared to the baseline in the crucial low-momentum region. The model is implemented in the LHCb trigger. I've also investigated the feasibility of this approach for the data with a high number of primary vertices. I show that it outperforms the other alternative considered of the upgrade, χ^2_{CORR} (described in section 5.3). The work has been submitted to proceedings of the ACAT 2019 conference and is included in the LHCb (as of the time of writing, internal) note [222]. A paper is in preparation to be submitted to JINST. I have also prepared muon identification as a problem for a data science competition. The competition, called the International Data Analysis Olympiad [223], is organised by Yandex and the Higher School of Economics. It attracted 1287 teams from 78 countries in 2019. Most of the participants are undergraduate Computer Science students. It aims to bridge the gap between the all-increasing complexity of machine learning models and the real-world requirements of the industry.

Machine learning on data with sPlot background subtraction My contribution is a rigorous way to use any machine learning methods to obtain class probabilities from data with sPlot-based background subtraction. Its primary use case is data-driven training of machine learning models. Aside from that, it can also be used to check whether a particular event selection routine violates the assumptions behind the sPlot method, as proposed by reference [305]. The method presented in the thesis has been published in [306] and its extension for the case of multiple classes with separate background sources has been submitted to ACAT 2019 conference proceedings. The project is described in chapter 6.

For global particle identification, I developed a solution based on gradient boosting decision trees. It allowed reducing error, as measured by 1-vs-all $1-\text{AUC}$ (area *over* the ROC curve) score, by 18%-54% compared to the baseline solution on simulated data. The method is implemented in the LHCb software stack. It has been published in [255]. It is described in chapter 7.

For fast simulation, combining the calibration samples with the Generative Adversarial Networks [128] allows for a very attractive possibility – a model that is both faster and more accurate than the full simulation with GEANT4. I performed a pilot study. I generated Monte-Carlo for BaBar DIRC and developed a model to simulate it. The simulation is $\sim 8 \cdot 10^4$ times faster than GEANT4 while retaining reasonable quality. The method has been published in [296]. The work to include this method into the LHCb software stack is ongoing and has been submitted to proceedings of the ACAT 2019 conference. The project is described in chapter 8.

On the whole, the developed methods and approaches constitute an interdisciplinary study across computer science and physics. Those methods improve the state-of-the-art in the aforementioned high energy physics projects. Thus, it allows for increasing the efficiency of current experiments searching for new physics phenomena and lays the ground for even more sensitive and sophisticated data processing in future experiments.

Appendix A

No Free Lunch Theorem Proof

This proof is my rewrite of the original proof [54] and was first published in [55]. The general idea behind the No Free Lunch theorems is calculating the uniform average ($P(f) = \text{const}$) over f of the distribution of classifier performance (loss c) conditioned on various variables.

Lemma 1.

$$P(c|d,f) = \sum_{y_H,y_F,q} \delta\left[c, L\left(y_H, y_F\right)\right] P\left(y_H|q,d\right) P\left(y_F|q,f\right) P\left(q|d\right) \tag{A.1}$$

Proof.

$$c = L\left(y_H, y_F\right) \tag{A.2}$$

This value is a function of two random variables: the model prediction y_H and sampled target function value y_F. By writing this down and expanding the conditional probabilities, we arrive to the expression A.1:

$$\begin{aligned} P\left(c|q,d,f\right) &= \sum_{y_H,y_F} \delta\left[c, L\left(y_H, y_F\right)\right] P\left(y_H, y_F|q,d,f\right) \\ P\left(c|d,f\right) &= \sum_{y_H,y_F,q} \delta\left[c, L\left(y_H, y_F\right)\right] P\left(y_H, y_F|q,d,f\right) P\left(q|d\right) \\ &= \sum_{y_H,y_F,q} \delta\left[c, L\left(y_H, y_F\right)\right] P\left(y_H|q,d\right) P\left(y_F|q,f\right) P\left(q|d\right) \end{aligned} \tag{A.3}$$

□

Lemma 2. *For homogeneous loss L, the uniform average over all f of $P\left(c|d,f\right)$ equals $\Lambda\left(c\right)/r$.*

For any training set, any $|\mathbf{Y}|$, any homogeneous loss L, any OTS method $P(q|d)$ of selecting the test point, including sampling the same $\pi(x)$, that was used to select d_X, for any learning algorithm, the average performance over all possible targets is a constant.

This result ignores the relationship between d and f. In other words, the $\mathbf{Y}$ values for train and test sets are generated from different distributions. Thus it is not particularly interesting in itself but will rather serve as a base for further inquiries.

Proof. Using Lemma 1, the uniform average over all targets f of $P(c|d,f)$ can be written as

$$E_f\left[P(c|d,f)\right] = \sum_{y_H,y_F,q} \delta\left[c, L(y_H,y_F)\right] P(y_H|q,d)\, E_f\left[P(y_F|q,f)\right] P(q|d) \tag{A.4}$$

$$E_f\left[P(y_F|q,f)\right] = E_f f(q,y_F) \tag{A.5}$$

The integration is over all possible values of f. The space of all possible values is the same for all components, thus the average is a constant that does not depend on q and y_F. Also,

$$\sum_{y_F} E_f\left[f(q,y_F)\right] = E_f\left[\sum_{y_F} f(q,y_F)\right] = 1, \tag{A.6}$$

therefore

$$E_f\left[f(q,y_F)\right] = 1/r. \tag{A.7}$$

Using the homogeneity property of L:

$$E_f P(c|d,f) = \sum_{y_H,q} \Lambda(c) P(y_H|q,d)\, P(q|d)\,/r = \Lambda(c)/r \tag{A.8}$$

$\square$

Theorem A.0.1. *For vertical $P(d|f)$, and a homogeneous loss L, the uniform average over all targets f of $P(c|f,m) = \Lambda(c)/r$*

For any $|\mathbf{Y}|$, any homogeneous loss L, any fixed training set size m, any vertical method of training set generation, including the conventional IID-generated, for any OTS method $P(q|d)$ of selecting the test point, including sampling the same $\pi(x)$, that was used to select d_X, for any learning algorithm, the average performance over all possible targets is a constant.

This is the result advertised in the beginning. If an algorithm “beats” some other, including the random guess, on some f’s, it will necessarily lose on the rest, so that the average losses will be the same.

Proof.

$$P(c|f,m) = \sum_{d:|d|=m} P(c|d,f)\, P(d|f) \tag{A.9}$$

From Lemma 1:

$$P(c|d,f) = \sum_{y_H,y_F,q} \delta\left[c, L(y_H,y_F)\right] P(y_H|q,d)\, P(y_F|q,f)\, P(q|d)\,. \tag{A.10}$$

Because we consider OTS error, $P(q|d)$ will be non-zero only for $q \notin d_X$, so $P(y_F|q,f)$ only depends on components of $f(x,y)$ that correspond to $x \notin d_X$.

We also know that $P(d|f)$ is vertical, so it is independent of the values $f(x,y_F)$ for $x \notin d_X$.

Therefore the integral can be split into two parts, over dimensions corresponding to d_X and $\mathbf{X} \setminus d_X$:

$$E_f\left[P\left(c|f,m\right)\right] = \sum_{d:|d|=m}\left[\frac{\int P\left(c|d,f\right) df_{x\notin d_X} \int P\left(d|f\right) df_{x\in d_X}}{\int 1 df_{x\notin d_X} df_{x\in d_X}}\right] \tag{A.11}$$

Again using $P\left(c|d,f\right)$ independence from $x \in d_X$ and Lemma 2:

$$E_{f_{x\notin d_X}}\left[P\left(c|d,f\right)\right] = E_f\left[P\left(c|d,f\right)\right] = \Lambda(c)/r \tag{A.12}$$

$$E_f\left[P\left(c|f,m\right)\right] = \Lambda(c)/r \sum_{d:|d|=m}\left[\frac{\int P\left(d|f\right) df_{x\in d_X}}{\int 1 df_{x\in d_X}}\right] = \Lambda(c)/r \tag{A.13}$$

□

No Free Lunch theorem can also be formulated in Bayesian analysis terms:

Theorem A.0.2. *For a vertical $P(d|f)$, uniform $P(f)$, and a homogeneous loss L, $P(c|d) = \Lambda(c)/r$.*

Proof.

$$E_f\left[P(c|d)\right] = \frac{\int P(c|d,f)P(f|d)df}{\int df} \tag{A.14}$$

Using the Bayes theorem,

$$P(f) = P(f|d)P(d)/P(d|f), \tag{A.15}$$

and uniformity of $P(f)$:

$$E_f\left[P(c|d)\right] = \frac{\int P(c|d,f)P(f)P(d|f)/P(d)df}{\int 1df} = \alpha(d)\frac{\int P(c|d,f)P(d|f)df}{\int 1df}, \tag{A.16}$$

where $\alpha(d)$ is some function. Like in Theorem A.0.1, the integral can be split into parts that depend on $f\left(x \in d_X\right)$ and $f\left(x \notin d_X\right)$:

$$E_f\left[P(c|d)\right] = \alpha(d)\frac{\int P\left(c|d,f\right) df_{x\notin d_X} \int P\left(d|f\right) df_{x\in d_X}}{\int 1 df_{x\notin d_X} df_{x\in d_X}}. \tag{A.17}$$

The integral $\int P\left(d|f\right) df_{x\in d_X}$ can again be absorbed into the d-dependent constant:

$$E_f\left[P(c|d)\right] = \beta(d)\frac{\int P\left(c|d,f\right) df_{x\notin d_X}}{\int 1 df_{x\notin d_X} df_{x\in d_X}} = \frac{\Lambda(c)}{r}\frac{\beta(d)}{\int 1 df_{x\in d_X}}. \tag{A.18}$$

To obtain the value of the constant, we integrate both sides over c:

$$\int E_f\left[P(c|d)\right]dc = 1 = \frac{\beta(d)}{\int 1 df_{x\in d_X}}\int \frac{\Lambda(c)}{r}dc. \tag{A.19}$$

From Theorem A.0.1 we know that $\frac{\Lambda(c)}{r}$ is, in fact, a probability, thus $\int \frac{\Lambda(c)}{r}dc = 1$. Therefore $\frac{\beta(d)}{\int 1 df_{x\in d_X}} = 1$ as well. Substituting it back to Formula A.18, we obtain:

$$E_f\left[P(c|d)\right] = \frac{\Lambda(c)}{r} \tag{A.20}$$

□

Appendix B

Global PID input variables

B.1 Used in ProbNN and our models

Tracking hardware and algorithms are described in the subsection 4.2.1. Here we list the variables it provides and that are later used in determining the particle type. Quoting the LHCb documentation [307]:

- `TrackChi2PerDof`, the track fit χ^2 per degree of freedom (the higher is χ^2, the more the tacking algorithm is sure in the track parameters)
- `TrackNumDof`, the number of degrees of freedom in the track fit
- `TrackDOCA`, the distance between the track and the z axis
- `TrackFitTChi2` the track fit χ^2 in TT
- `TrackFitTNDoF` the number of degrees of freedom in the track fit in TT
- `TrackFitVeloChi2` the track fit χ^2 in VELO
- `TrackFitVeloNDoF` the number of degrees of freedom in the track fit in VELO
- `TrackGhostProbability` output of the NN of the particle to be a ghost
- `TrackType` the track type (Long, Downstream, Upstream)
- `TrackP` the track momentum
- `TrackPt` the track transverse momentum
- `TrackMatchChi2` the track fit χ^2 per degree of freedom
- `TrackNumDof` the number of degrees of freedom in the track fit

RICH hardware and algorithms are described in the subsection 4.2.2.1. Here we list the variables it provides and that are later used in determining the particle type. Quoting the LHCb documentation [307]:

- `RichAboveElThres` binary indicator whether the track momentum is above momentum threshold for electrons to produce Cherenkov light

- `RichAboveKaThres` binary indicator whether the track momentum is above momentum threshold for kaons to produce Cherenkov light
- `RichAboveMuThres` binary indicator whether the track momentum is above momentum threshold for muons to produce Cherenkov light
- `RichAbovePiThres` binary indicator whether the track momentum is above momentum threshold for pions to produce Cherenkov light
- `RichAbovePrThres` binary indicator whether the track momentum is above momentum threshold for protons to produce Cherenkov light
- `RichDLLbt` the RICH delta log-likelihood value for the 'Below Threshold' hypothesis.
- `RichDLLe` the RICH delta log-likelihood value for the electron hypothesis
- `RichDLLk` the RICH delta log-likelihood value for the kaon hypothesis
- `RichDLLmu` the RICH delta log-likelihood value for the muon hypothesis
- `RichDLLp` the RICH delta log-likelihood value for the proton hypothesis
- `RichDLLpi` the RICH delta log-likelihood value for the pion hypothesis
- `RichUsedR1Gas` used the RICH1 detector in this track ID
- `RichUsedR2Gas` used the RICH2 detector in this track ID

CALO hardware is described in the subsection 4.2.2.2. Here we list the variables it provides and that are later used in determining the particle type. Quoting [307, 308, 309]:

- `InAccBrem` binary indicator whether the bremsstrahlung photons fall into acceptance
- `InAccEcal` binary indicator whether the track falls into the geometric acceptance of the ECAL
- `InAccHcal` binary indicator whether the track falls into the geometric acceptance of the HCAL
- `InAccPrs` binary indicator whether the track falls into the geometric acceptance of the PRS
- `InAccSpb` binary indicator whether the track falls into the geometric acceptance of the SPD
- `BremPIDe` the difference between the log-likelihoods for the electron and non-electron hypotheses
- `PrsPIDe` the difference between the log-likelihoods for the electron and non-electron hypotheses derived from the energy deposition in the preshower (PS)

- `EcalPIDe` the difference between the log-likelihoods for the electron and non-electron hypotheses derived from the ECAL reading
- `HcalPIDe` the difference between the log-likelihoods for the electron and non-electron hypotheses derived from energy deposition in HCAL along the particle trajectory
- `EcalPIDmu` the difference between the log-likelihoods for the muon and non-muon hypotheses derived from the ECAL energy deposition along the particle trajectory
- `HcalPIDmu` the difference between the log-likelihoods for the muon and non-muon hypotheses derived from the HCAL energy deposition along the particle trajectory
- `CaloChargedEcal` ECAL cluster energy associated to the track
- `CaloChargedPrs` Preshower (PS) digits associated to the track
- `CaloChargedSpd` SPD digits associated to the track
- `CaloBremMatch` the χ^2 value of the track-cluster matching for the bremsstrahlung photon hypothesis, using the track extrapolation from its direction before the magnet
- `CaloEcalE` the ECAL energy deposition along the track line
- `CaloElectronMatch` the χ^2 value of the track-cluster matching the electron hypothesis, using the track extrapolation from its direction after the magnet
- `CaloHcalE` the HCAL energy deposition along the track line
- `CaloPrsE` the PS energy deposition along the track line
- `CaloSpdE` the SPD energy deposition along the track line
- `CaloTrMatch` 2D χ^2 for Track/CaloCluster matching
- `CaloTrajectoryL` longitudinal parameter of the ECAL shower
- `CaloNeutralEcal` cluster energy associated to neutral hypothesis
- `CaloNeutralPrs` PS digits associated to neutral hypothesis
- `CaloNeutralSpd` SPD digits associated to neutral hypothesis

MUON hardware is described in the section 5.1. Here we list the variables it provides and that are later used in determining the particle type. Quoting the LHCb documentation [307]:

- `InAccMuon` binary indicator whether the track falls into the geometric acceptance of the MUON system

- MuonIsMuon binary indicator whether the track passed the IsMuon selection (subsection 4.3.2.1)
- MuonIsLooseMuon binary indicator whether the track passed the IsMuonLoose selection (subsection 4.3.2.1)
- MuonBkgLL the log likelihood computed by the muDLL algorithm (section 5.2) for the track to be not a muon
- MuonMuLL the log likelihood computed by the muDLL algorithm (section 5.2) for the track to be a muon
- MuonNShared the number of hits in the muon chambers that are shared with the other tracks

B.2 Additional engineered features

In addition to the variables listed in B.1, we used 20 physically-meaningful combined features, which were experimentally found to increase performance [263]:

- CaloSumPIDmu = EcalPIDmu + HcalPIDmu
- CaloSumSpdPrsE = CaloSpdE + CaloPrsE
- SpdCaloChargedNeutral = CaloChargedSpd + CaloNeutralSpd
- SpdCaloChargedAcc = CaloChargedSpd + InAccSpd
- SpdCaloNeutralAcc = CaloNeutralSpd + InAccSpd
- RichUsedGas = RichUsedR1Gas + RichUsedR2Gas
- RichAboveSumKaPrTHres = RichAboveKaThres + RichAbovePrThres
- RichAboveSumPiKaElMuTHres = RichAbovePiThres + RichAboveKaThres + RichAboveElThres + RichAboveMuThres
- acc_cum_sum_3 = InAccSpd + InAccPrs + InAccBrem + InAccEcal
- acc_cum_sum_5 = InAccSpd + InAccPrs + InAccBrem + InAccEcal + InAccHcal + InAccMuon
- CombDLLmu_LL combined log likelihood (not delta) for the muon hypothesis
- CombDLLpi_LL combined log likelihood (not delta) for the pion hypothesis
- CombDLLp_LL combined log likelihood (not delta) for the proton hypothesis
- CombDLLe_LL combined log likelihood (not delta) for the electron hypothesis
- CombDLLk_LL combined log likelihood (not delta) for the kaon hypothesis
- RichDLLpi_LL RICH log likelihood (not delta) for the pion hypothesis

- `RichDLLe_LL` RICH log likelihood (not delta) for the electron hypothesis
- `RichDLLp_LL` RICH log likelihood (not delta) for the proton hypothesis
- `RichDLLmu_LL` RICH log likelihood (not delta) for the muon hypothesis
- `RichDLLk_LL` RICH log likelihood (not delta) for the kaon hypothesis

Bibliography

[1] R. Aaij et al. “Test of Lepton Universality Using $B^+ \to K^+\ell^+\ell^-$ Decays”. In: *Phys. Rev. Lett.* 113 (15 Oct. 2014), p. 151601. DOI: 10.1103/PhysRevLett.113.151601. URL: https://link.aps.org/doi/10.1103/PhysRevLett.113.151601.

[2] R. Aaij et al. “Search for Lepton-Universality Violation in $B^+ \to K^+\ell^+\ell^-$ Decays”. In: *Phys. Rev. Lett.* 122 (19 May 2019), p. 191801. DOI: 10.1103/PhysRevLett.122.191801. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.191801.

[3] R. Aaij et al. “Measurement of the Ratio of Branching Fractions $\mathcal{B}(\overline{B}^0 \to D^{*+}\tau^-\overline{\nu}_\tau)/\mathcal{B}(\overline{B}^0 \to D^{*+}\mu^-\overline{\nu}_\mu)$”. In: *Phys. Rev. Lett.* 115 (11 Sept. 2015), p. 111803. DOI: 10.1103/PhysRevLett.115.111803. URL: https://link.aps.org/doi/10.1103/PhysRevLett.115.111803.

[4] “Beauty quarks test lepton universality”. In: *CERN Courier* 58.3 (Mar. 2018). URL: https://cds.cern.ch/record/2315229.

[5] R Aaij et al. “Measurement of the ${B_s}^0 \to \mu^+\mu^-$ branching fraction and search for $B^0 \to \mu^+\mu^-$ decays at the LHCb experiment”. In: *Physical review letters* 111.10 (2013), p. 101805.

[6] R. Aaij et al. “Measurement of the $B_s^0 \to \mu^+\mu^-$ Branching Fraction and Effective Lifetime and Search for $B^0 \to \mu^+\mu^-$ Decays”. In: *Phys. Rev. Lett.* 118 (19 May 2017), p. 191801. DOI: 10.1103/PhysRevLett.118.191801. URL: https://link.aps.org/doi/10.1103/PhysRevLett.118.191801.

[7] Serguei Chatrchyan et al. “Measurement of the $B_s^0 \to \mu^+\mu^-$ branching fraction and search for $B^0 \to \mu^+\mu^-$ with the CMS experiment”. In: *Physical review letters* 111.10 (2013), p. 101804.

[8] ATLAS collaboration et al. “Study of the rare decays of B_s^0 and B^0 mesons into muon pairs using data collected during 2015 and 2016 with the ATLAS detector”. In: *Journal of High Energy Physics* 2019.4 (2019), p. 98.

[9] Vardan Khachatryan et al. “Observation of the rare $B_s^0 \to \mu^+\mu^-$ decay from the combined analysis of CMS and LHCb data”. In: *Nature* 522 (2015), pp. 68–72. DOI: 10.1038/nature14474. arXiv: 1411.4413 [hep-ex].

[10] LHCb Collaboration. “LHCb detector performance”. In: *International Journal of Modern Physics A* 30.07 (2015), p. 1530022.

[11] A. A. Alves Jr. et al. “The LHCb detector at the LHC”. In: *JINST* 3 (2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005.

[12] *LHCb Trigger and Online Upgrade Technical Design Report*. Tech. rep. CERN-LHCC-2014-016. LHCB-TDR-016. May 2014. URL: https://cds.cern.ch/record/1701361.

[13] I Bediaga et al. *Framework TDR for the LHCb Upgrade: Technical Design Report*. Tech. rep. CERN-LHCC-2012-007. LHCb-TDR-12. Apr. 2012. URL: https://cds.cern.ch/record/1443882.

[14] M Clemencic et al. "The LHCb simulation application, Gauss: design, evolution and experience". In: *Journal of Physics: Conference Series*. Vol. 331. 3. IOP Publishing. 2011, p. 032023.

[15] Gloria Corti. *LHCb Fast Simulation(s)*. 2016. URL: https://indico.cern.ch/event/523577/contributions/2178698/attachments/1279812/1900824/GCorti-LHCbFastSim-20160526.pdf.

[16] Benedetto Gianluca Siddi. "A fully parametric option in the LHCb simulation framework". In: *EPJ Web of Conferences*. Vol. 214. EDP Sciences. 2019, p. 02024.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[18] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[19] F Archilli et al. "Performance of the muon identification at LHCb". In: *Journal of Instrumentation* 8.10 (2013), P10020.

[20] R Aaij et al. "Differential branching fraction and angular analysis of the decay B 0→ K* 0 μ+ μ-". In: *Physical review letters* 108.18 (2012), p. 181806.

[21] Muriel Pivk and Francois R Le Diberder. "sPlot: A statistical tool to unfold data distributions". In: *NIMA* 555.1-2 (2005), pp. 356–369.

[22] Lucio Anderlini et al. *The PIDCalib package*. Tech. rep. LHCb-PUB-2016-021. CERN-LHCb-PUB-2016-021. Geneva: CERN, July 2016. URL: https://cds.cern.ch/record/2202412.

[23] *Artificial intelligence*. Encyclopædia Britannica, Inc. URL: https://www.britannica.com/technology/artificial-intelligence (visited on 10/05/2019).

[24] Pamela McCorduck. "Machines who think". In: (2004).

[25] ΄Ομηρο. *Ιλιάδα*. 850.

[26] John McCarthy et al. "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955". In: *AI magazine* 27.4 (2006), pp. 12–12.

[27] P Russel Norvig and S Artificial Intelligence. *A modern approach*. Prentice Hall, 2002.

[28] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep blue". In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[29] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[30] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), p. 484.

[31] Oriol Vinyals et al. “AlphaStar: Mastering the real-time strategy game StarCraft II”. In: *DeepMind Blog* (2019).

[32] AM Turing. “Computing machinery and intelligence”. In: *Mind* 59.236 (1950), p. 433.

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[34] Alex Graves and Navdeep Jaitly. “Towards end-to-end speech recognition with recurrent neural networks”. In: *International conference on machine learning*. 2014, pp. 1764–1772.

[35] Herbert A Simon and Allen Newell. “Heuristic problem solving: The next advance in operations research”. In: *Operations research* 6.1 (1958), pp. 1–10.

[36] Daniel Crevier. *AI: the tumultuous history of the search for artificial intelligence*. Basic Books, 1993.

[37] Richard M Russell. “The CRAY-1 computer system”. In: *Communications of the ACM* 21.1 (1978), pp. 63–72.

[38] John McCarthy. *Artificial intelligence: a paper symposium: Professor Sir James Lighthill, FRS. Artificial Intelligence: A General Survey. In: Science Research Council, 1973*. 1974.

[39] Jelena Stajic et al. *Rise of the Machines*. 2015.

[40] *Conditional expectation*. URL: https://en.wikipedia.org/wiki/Conditional_expectation.

[41] David Foster. *Generative deep learning: teaching machines to paint, write, compose, and play*. O’Reilly Media, 2019.

[42] Alexey Artemov. “Intro into Machine Learning”. In: *The Fourth Machine Learning summer school* (2018). URL: https://indico.cern.ch/event/687473/sessions/259653/.

[43] Claire Adam-Bourdarios et al. “The Higgs machine learning challenge”. In: *Journal of Physics: Conference Series*. Vol. 664. 7. IOP Publishing. 2015, p. 072015.

[44] Liudmila Prokhorenkova et al. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6638–6648.

[45] Olivier Callot. *FastVelo, a fast and efficient pattern recognition package for the Velo*. Tech. rep. 2011.

[46] S Viret, C Parkes, and D Petrie. *LHCb VELO software alignment, Part I: the alignment of the VELO modules in their half boxes*. Tech. rep. LHCb-2005-101. CERN-LHCb-2005-101. Geneva: CERN, Dec. 2005. URL: https://cds.cern.ch/record/914076.

[47] Are Strandlie and Rudolf Frühwirth. “Track and vertex reconstruction: From classical to adaptive methods”. In: *Reviews of Modern Physics* 82.2 (2010), p. 1419.

[48] Yassine Alouini. *Histogram of most used eval metrics*. URL: https://www.kaggle.com/yassinealouini/histogram-of-most-used-eval-metrics (visited on 10/07/2019).

[49] Sergio Bermejo and Joan Cabestany. “Oriented principal component analysis for large margin classifiers”. In: *Neural Networks* 14.10 (2001), pp. 1447–1461.

[50] Solomon Kullback. *Statistics and information theory*. J. Wiley and Sons, New York, 1959.

[51] Donald Bamber. “The area above the ordinal dominance graph and the area below the receiver operating characteristic graph”. In: *Journal of mathematical psychology* 12.4 (1975), pp. 387–415.

[52] Elizabeth R DeLong, David M DeLong, and Daniel L Clarke-Pearson. “Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach.” In: *Biometrics* 44.3 (1988), pp. 837–845.

[53] Martina Kottas, Oliver Kuss, and Antonia Zapf. “A modified Wald interval for the area under the ROC curve (AUC) in diagnostic case-control studies”. In: *BMC medical research methodology* 14.1 (2014), p. 26.

[54] David H Wolpert. “The supervised learning no-free-lunch theorems”. In: *Soft computing and industry*. Springer, 2002, pp. 25–42.

[55] Nikita Kazeev. *The Lack of A Priori Distinctions Between Learning Algorithms aka No Free Lunch Theorems for Learning*. URL: https://github.com/kazeevn/no_free_lunch.

[56] Anne Auger and Olivier Teytaud. “Continuous lunches are free plus the design of optimal optimization algorithms”. In: *Algorithmica* 57.1 (2010), pp. 121–146.

[57] David H Wolpert. “The existence of a priori distinctions between learning algorithms”. In: *Neural Computation* 8.7 (1996), pp. 1391–1420.

[58] Zhou Lu et al. “The expressive power of neural networks: A view from the width”. In: *Advances in neural information processing systems*. 2017, pp. 6231–6239.

[59] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[60] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[61] Higher School of Economics. *Advanced Machine Learning Specialization*. URL: https://www.coursera.org/specializations/aml.

[62] Stan Lipovetsky. "Analytical closed-form solution for binary logit regression by categorical predictors". In: *Journal of applied statistics* 42.1 (2015), pp. 37–49.

[63] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[64] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[65] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[66] Aimo Törn and Antanas Žilinskas. *Global optimization*. Vol. 350. Springer, 1989.

[67] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[68] Augustin Cauchy. "Méthode générale pour la résolution des systemes d'équations simultanées". In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[69] Jack Kiefer, Jacob Wolfowitz, et al. "Stochastic estimation of the maximum of a regression function". In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 462–466.

[70] Eduardo D Sontag and Héctor J Sussmann. "Backpropagation can give rise to spurious local minima even for networks without hidden layers". In: *Complex Systems* 3.1 (1989), pp. 91–106.

[71] Martin L Brady, Raghu Raghavan, and Joseph Slawny. "Back propagation fails to separate where perceptrons succeed". In: *IEEE Transactions on Circuits and Systems* 36.5 (1989), pp. 665–674.

[72] Marco Gori and Alberto Tesi. "On the problem of local minima in backpropagation". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 1 (1992), pp. 76–86.

[73] Andrew M Saxe, James L McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120* (2013).

[74] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[75] Anna Choromanska et al. "The loss surfaces of multilayer networks". In: *Artificial Intelligence and Statistics*. 2015, pp. 192–204.

[76] Joohyoung. *Deep Learning* 이론과 실습 (개정증). 2015. URL: https://wikidocs.net/book/498.

[77] Subir Varma and Sanjiv Das. *Deep Learning*. 2018. URL: https://srdas.github.io/DLBook/.

[78] Boris T Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[79] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[80] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[81] Felipe Petroski Such et al. “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning”. In: *arXiv preprint arXiv:1712.06567* (2017).

[82] Gavin Taylor et al. “Training Neural Networks Without Gradients: A Scalable ADMM Approach”. In: *International conference on machine learning.* 2016, pp. 2722–2731.

[83] Randal J Barnes. “Matrix differentiation”. In: *Springs Journal* (2006), pp. 1–9.

[84] Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. In: *Master’s Thesis (in Finnish), Univ. Helsinki* (1970), pp. 6–7.

[85] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[86] Geoffrey Hinton et al. “Deep neural networks for acoustic modeling in speech recognition”. In: *IEEE Signal processing magazine* 29 (2012).

[87] Jimmy Ba and Rich Caruana. “Do deep nets really need to be deep?” In: *Advances in neural information processing systems.* 2014, pp. 2654–2662.

[88] Gregor Urban et al. “Do deep convolutional nets really need to be deep and convolutional?” In: *arXiv preprint arXiv:1603.05691* (2016).

[89] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).

[90] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search”. In: *arXiv preprint arXiv:1806.09055* (2018).

[91] Hieu Pham et al. “Efficient neural architecture search via parameter sharing”. In: *arXiv preprint arXiv:1802.03268* (2018).

[92] Ronald J Williams and David Zipser. “A learning algorithm for continually running fully recurrent neural networks”. In: *Neural computation* 1.2 (1989), pp. 270–280.

[93] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to forget: Continual prediction with LSTM”. In: (1999).

[94] Norman Jouppi et al. “Motivation for and evaluation of the first tensor processing unit”. In: *IEEE Micro* 38.3 (2018), pp. 10–19.

[95] Shaohuai Shi et al. "Benchmarking state-of-the-art deep learning software tools". In: *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE. 2016, pp. 99–104.

[96] MissingLink. *The Complete Guide to Deep Learning with GPUs*. URL: https://missinglink.ai/guides/computer-vision/complete-guide-deep-learning-gpus/ (visited on 10/06/2019).

[97] Nvidia. *NVIDIA TESLA V100 GPU ACCELERATOR*. URL: https://images.nvidia.com/content/technologies/volta/pdf/437317-Volta-V100-DS-NV-US-WEB.pdf (visited on 10/06/2019).

[98] *NVIDIA V100 tensor core GPU*. URL: https://www.nvidia.com/en-us/data-center/v100/ (visited on 02/03/2020).

[99] *Xeon Platinum 8253 - Intel*. URL: https://en.wikichip.org/wiki/intel/xeon_platinum/8253.

[100] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. "Understanding the efficiency of GPU algorithms for matrix-matrix multiplication". In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM. 2004, pp. 133–137.

[101] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[102] Adam Paszke et al. "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*. 2017.

[103] Alexey Artemov and Andrey Ustyuzhanin. "Decision Trees and Ensembling algorithms". In: *The Fifth Machine Learning summer school* (2019). URL: https://indico.cern.ch/event/768915/.

[104] Tibshirani R Hastie and JH Friedman. *Elements of statistical learning: data mining, inference, and prediction*. 2003.

[105] *Statistical decision series*. Statistical Decision Series . 4. HBR, 1959. URL: https://books.google.ru/books?id=8WwPAQAAMAAJ.

[106] L.M.L. Cam and J. Neyman. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability: Weather modification*. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability: Held at the Statistical Laboratory, University of California, June 21-July 18, 1965 and December 27, 1965-January 7, 1966. University of California Press, 1967. URL: https://books.google.ru/books?id=QsBQCBgTx8gC.

[107] Πορφύριος. *Εἰσαγωγή*. 270.

[108] Mike Hunter (https://stats.stackexchange.com/users/78229/mike-hunter). *Who invented the decision tree?* Cross Validated. URL:https://stats.stackexchange.com/q/257585 (version: 2017-04-08). eprint: https://stats.stackexchange.com/q/257585. URL: https://stats.stackexchange.com/q/257585.

[109] Laurent Hyafil and Ronald L. Rivest. "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1 (1976), pp. 15–17. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(76)90095-8. URL: http://www.sciencedirect.com/science/article/pii/0020019076900958.

[110] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

[111] Ricky Ho. *Machine Learning: Patterns for Predictive Analytics*. 2012. URL: https://dzone.com/refcardz/machine-learning-predictive?chapter=1.

[112] *Decision tree learning*. URL: https://en.wikipedia.org/wiki/Decision_tree_learning.

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

[114] Kai Ming Ting and Lian Zhu. "Boosting support vector machines successfully". In: *International Workshop on Multiple Classifier Systems*. Springer. 2009, pp. 509–518.

[115] Dinesh Govindaraj. "Can Boosting with SVM as Week Learners Help?" In: *arXiv preprint arXiv:1604.05242* (2016).

[116] Holger Schwenk and Y. Bengio. "Boosting Neural Networks". In: *Neural computation* 12 (Sept. 2000), pp. 1869–87. DOI: 10.1162/089976600300015178.

[117] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[118] Guolin Ke et al. "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3146–3154.

[119] Michal Ferov and Marek Modrý. *Enhancing LambdaMART Using Oblivious Trees*. 2016. arXiv: 1609.05610 [cs.IR].

[120] Andrey Gulin, Igor Kuralenok, and Dimitry Pavlov. "Winning the transfer learning track of yahoo!'s learning to rank challenge with yetirank". In: *Proceedings of the Learning to Rank Challenge*. 2011, pp. 63–76.

[121] Ron Kohavi and Dan Sommerfield. "Targeting Business Users with Decision Table Classifiers." In: *KDD*. 1998, pp. 249–253.

[122] Michael Levin. *MatrixNet*. 2016. URL: https://www.slideshare.net/mlprague/michael-levin-matrixnet-applications-at-yandex.

[123] Anna Veronika Dorogush et al. "Fighting biases with dynamic boosting". In: *arXiv preprint arXiv:1706.09516v1* (2017). URL: https://arxiv.org/abs/1706.09516v1.

[124] Yandex. *Best in class inference and a ton of speedups*. URL: https://catboost.ai/news/best-in-class-inference-and-a-ton-of-speedups.

[125] Chen-Ying Hung et al. “Comparing deep neural network and other machine learning algorithms for stroke prediction in a large-scale population-based electronic medical claims database”. In: *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE. 2017, pp. 3110–3113.

[126] Jacques Wainer. “Comparison of 14 different families of classification algorithms on 115 binary datasets”. In: *arXiv preprint arXiv:1606.00930* (2016).

[127] Ruslan Salakhutdinov and Geoffrey Hinton. “Deep boltzmann machines”. In: *Artificial intelligence and statistics*. 2009, pp. 448–455.

[128] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[129] Jürgen Schmidhuber. “Learning factorial codes by predictability minimization”. In: *Neural Computation* 4.6 (1992), pp. 863–879.

[130] *Were generative adversarial networks introduced by Jürgen Schmidhuber?* URL: https://stats.stackexchange.com/questions/251460/were-generative-adversarial-networks-introduced-by-j%C3%BCrgen-schmidhuber (visited on 12/20/2019).

[131] *Generative Adversarial Networks (GANs) in 50 lines of code (PyTorch)*. URL: https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f (visited on 12/20/2019).

[132] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[133] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[134] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html.

[135] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5767–5777. URL: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf.

[136] Vincent Herrmann. *Wasserstein GAN and the Kantorovich-Rubinstein Duality*. URL: https://vincentherrmann.github.io/blog/wasserstein/ (visited on 12/22/2019).

[137] Канторович, Леонид Витальевич and Рубинштейн, Геннадий Шлемович. “Об одном функциональном пространстве и некоторых экстремальных задачах”. In: *Доклады Академии наук*. Vol. 115. 6. Академия наук СССР. 1957, pp. 1058–1061.

[138] *Lipschitz continuity*. URL: https://en.wikipedia.org/wiki/Lipschitz_continuity (visited on 12/23/2019).

[139] Marc G Bellemare et al. “The Cramer distance as a solution to biased Wasserstein gradients”. In: *arXiv preprint arXiv:1705.10743* (2017).

[140] Gábor J Székely. “E-Statistics: The energy of statistical samples”. In: *Bowling Green State University, Department of Mathematics and Statistics Technical Report* 3.05 (2003), pp. 1–18.

[141] David J Lange. “The EvtGen particle decay simulation package”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 462.1-2 (2001), pp. 152–155.

[142] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. “PYTHIA 6.4 physics and manual”. In: *Journal of High Energy Physics* 2006.05 (2006), p. 026.

[143] Eric M Metodiev, Benjamin Nachman, and Jesse Thaler. “Classification without labels: Learning from mixed samples in high energy physics”. In: *Journal of High Energy Physics* 2017.10 (2017), p. 174.

[144] *Flavours of Physics: Finding $\tau \to \mu\mu\mu$*. 2015. URL: https://www.kaggle.com/c/flavours-of-physics/ (visited on 01/27/2020).

[145] Vladislav Mironov (littus) and Alexander Guschin. *Flavours of Physics Finding $\tau \to \mu\mu\mu$. First place solution.* 2015. URL: https://github.com/aguschin/flavours-of-physics/blob/master/Flavours_of_Physics_Finding_tau-3mu_First_place_solution.pdf (visited on 01/27/2020).

[146] Gilles Louppe, Michael Kagan, and Kyle Cranmer. “Learning to pivot with adversarial networks”. In: *Advances in neural information processing systems*. 2017, pp. 981–990.

[147] Chase Shimmin et al. “Decorrelated jet substructure tagging using adversarial neural networks”. In: *Physical Review D* 96.7 (2017), p. 074034.

[148] Justin Stevens and Mike Williams. “uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers”. In: *Journal of Instrumentation* 8.12 (2013), P12013.

[149] Alex Rogozhnikov et al. “New approaches for boosting to uniformity”. In: *Journal of Instrumentation* 10.03 (2015), T03002.

[150] *Kolmogorov–Smirnov test*. URL: https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test (visited on 01/29/2020).

[151] Layne Bradshaw et al. “Mass Agnostic Jet Taggers”. In: *SciPost Phys.* 8 (1 2020), p. 11. DOI: 10.21468/SciPostPhys.8.1.011. URL: https://scipost.org/10.21468/SciPostPhys.8.1.011.

[152] LHCb Collaboration. “Measurement of forward $t\bar{t}$, $W + b\bar{b}$ and $W + c\bar{c}$ production in pp collisions at $\sqrt{s} = 8$ TeV”. In: *Physics Letters B* 767 (2017), pp. 110–120.

[153] Roel Aaij et al. “First observation of forward $Z \to b\bar{b}$ production in pp collisions at $\sqrt{s} = 8$ TeV”. In: *Physics Letters B* 776 (2018), pp. 430–439.

[154] S.H. Clearwater and E.G. Stern. "A rule-learning program in high energy physics event classification". In: *Computer Physics Communications* 67.2 (1991), pp. 159–182. ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(91)90014-C. URL: http://www.sciencedirect.com/science/article/pii/001046559190014C.

[155] Alexander Radovic et al. "Machine learning at the energy and intensity frontiers of particle physics". In: *Nature* 560.7716 (2018), p. 41.

[156] Tatiana Likhomanenko et al. "LHCb topological trigger reoptimization". In: *Journal of Physics: Conference Series*. Vol. 664. 8. IOP Publishing. 2015, p. 082025.

[157] M. Aaboud et al. "Observation of Higgs boson production in association with a top quark pair at the LHC with the ATLAS detector". In: *Physics Letters B* 784 (2018), pp. 173–191. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2018.07.035. URL: http://www.sciencedirect.com/science/article/pii/S0370269318305732.

[158] Claudia Marino. *X_b Search and Measurement of the $Y(1)$, $Y(2S)$ and $Y(3S)$ Polarization*. Tech. rep. Fermi National Accelerator Lab.(FNAL), Batavia, IL (United States), 2009.

[159] Michel De Cian. "Machine Learning and Parallelism in the Reconstruction of LHCb and its Upgrade". In: *EPJ Web of Conferences*. Vol. 127. EDP Sciences. 2016, p. 00006.

[160] M Aaboud et al. "Performance of the ATLAS track reconstruction algorithms in dense environments in LHC Run 2". In: *The European Physical Journal C* 77.10 (2017), p. 673.

[161] The LHCb collaboration. "Identification of beauty and charm quark jets at LHCb". In: *Journal of Instrumentation* 10.06 (July 2015), P06013–P06013. DOI: 10.1088/1748-0221/10/06/p06013. URL: https://doi.org/10.1088%2F1748-0221%2F10%2F06%2Fp06013.

[162] Roel Aaij et al. "A new algorithm for identifying the flavour of B0s mesons at LHCb". In: *Journal of Instrumentation* 11.05 (2016), P05010.

[163] Pierre Baldi et al. "Jet substructure classification in high-energy physics with deep neural networks". In: *Physical Review D* 93.9 (2016), p. 094034.

[164] Miriam Calvo Gomez et al. *A tool for γ/π^0 separation at high energies*. Tech. rep. LHCb-PUB-2015-016. CERN-LHCb-PUB-2015-016. Geneva: CERN, Aug. 2015. URL: https://cds.cern.ch/record/2042173.

[165] Donald Hill, Viktoriia Chekalina, and Martino Borsato. "Boosting Neutral Particles Identification by Boosting Trees: LHCb case". CHEP 2018 conference. 2018. URL: https://indico.cern.ch/event/587955/contributions/2937533/.

[166] R Aaij et al. "New algorithms for identifying the flavour of B^0 mesons using pions and protons". In: *The European Physical Journal C* 77.4 (2017), p. 238.

[167] M Adinolfi et al. “LHCb data quality monitoring”. In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 092027. DOI: 10.1088/1742-6596/898/9/092027. URL: https://doi.org/10.1088%2F1742-6596%2F898%2F9%2F092027.

[168] M Adinolfi et al. “Performance of the LHCb RICH detector at the LHC”. In: *Eur. Phys. J. C* 73.arXiv:1211.6759. CERN-LHCb-DP-2012-003. LHCb-DP-2012-003 (Oct. 2012), 2431. 25 p. DOI: 10.1140/epjc/s10052-013-2431-9. URL: https://cds.cern.ch/record/1495721.

[169] B Denby. “Neural networks and cellular automata in experimental high energy physics”. In: *Computer Physics Communications* 49.3 (1988), pp. 429–448.

[170] Luke de Oliveira et al. “Jet-images—deep learning edition”. In: *Journal of High Energy Physics* 2016.7 (2016), p. 69.

[171] Pierre Baldi et al. “Jet substructure classification in high-energy physics with deep neural networks”. In: *Phys. Rev. D* 93 (9 May 2016), p. 094034. DOI: 10.1103/PhysRevD.93.094034. URL: https://link.aps.org/doi/10.1103/PhysRevD.93.094034.

[172] Daniel Guest et al. “Jet flavor classification in high-energy physics with deep neural networks”. In: *Physical Review D* 94.11 (2016), p. 112002.

[173] Gilles Louppe et al. “QCD-aware recursive neural networks for jet physics”. In: *Journal of High Energy Physics* 2019.1 (2019), p. 57.

[174] Taoli Cheng. “Recursive neural networks in quark/gluon tagging”. In: *Computing and Software for Big Science* 2.1 (2018), p. 3.

[175] Steven Farrell et al. “Novel deep learning methods for track reconstruction”. In: *arXiv preprint arXiv:1810.06111* (2018).

[176] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[177] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2015. arXiv: 1511.05493 [cs.LG].

[178] F. Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2005605.

[179] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* 2018. arXiv: 1810.00826 [cs.LG].

[180] Jie Zhou et al. *Graph Neural Networks: A Review of Methods and Applications*. 2018. arXiv: 1812.08434 [cs.LG].

[181] Lindsey Gray et al. “Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors”. In: *Machine Learning and the Physical Sciences Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS)* (2019). URL: https://ml4physicalsciences.github.io/files/NeurIPS_ML4PS_2019_83.pdf.

[182] M. Bernardini and K. Foraz. “Long Shutdown 2 @ LHC”. In: *CERN Yellow Reports* 2.00 (2016), p. 290. URL: https://e-publishing.cern.ch/index.php/CYR/article/view/159.

[183] CERN. *The Large Hadron Collider*. URL: https://home.cern/science/accelerators/large-hadron-collider.

[184] Esma Mobs. “The CERN accelerator complex - 2019. Complexe des accélérateurs du CERN - 2019”. In: (June 2019). General Photo. URL: https://cds.cern.ch/record/2684277.

[185] Georges Aad et al. “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC”. In: *Physics Letters B* 716.1 (2012), pp. 1–29.

[186] Serguei Chatrchyan et al. “Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC”. In: *Physics Letters B* 716.1 (2012), pp. 30–61.

[187] LHCb Collaboration et al. “Measurement of the track reconstruction efficiency at LHCb”. In: *Journal of Instrumentation* 10.02 (2015), P02007.

[188] R Aaij et al. “Performance of the LHCb Vertex Locator”. In: *Journal of Instrumentation* 9.09 (Sept. 2014), P09007–P09007. DOI: 10.1088/1748-0221/9/09/p09007. URL: https://doi.org/10.1088%2F1748-0221%2F9%2F09%2Fp09007.

[189] Roel Aaij et al. “Performance of the LHCb vertex locator”. In: *Journal of Instrumentation* 9.09 (2014), P09007.

[190] Lucio Anderlini. “Measurement of the B_c^+ meson lifetime using $B_c^+ \to J/\psi\mu^+\nu_\mu X$ decays with the LHCb detector at CERN”. Presented 03 Feb 2015. Jan. 2014. URL: http://cds.cern.ch/record/1980460.

[191] LHCb Collaboration. *LHCb Silicon Tracker - Material for Publications*. URL: https://lhcb.physik.uzh.ch/ST/public/material/.

[192] *Cherenkov radiation*. URL: https://en.wikipedia.org/wiki/Cherenkov_radiation (visited on 09/11/2019).

[193] LHCb Collaboration. *RICHPicturesAndFigures*. URL: https://twiki.cern.ch/twiki/bin/view/LHCb/RICHPicturesAndFigures (visited on 09/12/2019).

[194] Claus Peter Buszello. “LHCb RICH Reconstruction”. In: (Oct. 2007). URL: https://cds.cern.ch/record/1432173.

[195] Eduardo Picatoste Olloqui, LHCb Collaboration, et al. “Lhcb preshower (ps) and scintillating pad detector (spd): Commissioning, calibration, and monitoring”. In: *Journal of Physics: Conference Series*. Vol. 160. 1. IOP Publishing. 2009, p. 012046.

[196] Oliver Lupton. “Studies of $\mathrm{D}^0 \to \mathrm{K}^0_\mathrm{S}\mathrm{h}^+\mathrm{h}'^-$ decays at the LHCb experiment”. Presented 14 Sep 2016. July 2016. URL: https://cds.cern.ch/record/2230910.

[197] Tomasz Szumlak. “IOP: Real time analysis with the upgraded LHCb trigger in Run III”. In: *J. Phys.: Conf. Ser.* Vol. 898. 2017, p. 032051.

[198] R Aaij et al. "Design and performance of the LHCb trigger and full real-time reconstruction in Run 2 of the LHC". In: *Journal of Instrumentation* 14.04 (2019), P04013.

[199] R Aaij et al. "The LHCb trigger and its performance in 2011". In: *Journal of Instrumentation* 8.04 (2013), P04022.

[200] CERN (Meyrin) LHCb Collaboration. *Computing Model of the Upgrade LHCb experiment*. Tech. rep. CERN-LHCC-2018-014. LHCB-TDR-018. Geneva: CERN, May 2018. URL: https://cds.cern.ch/record/2319756.

[201] Michel De Cian et al. *Fast neural-net based fake track rejection in the LHCb reconstruction*. Tech. rep. LHCb-PUB-2017-011. CERN-LHCb-PUB-2017-011. Geneva: CERN, Mar. 2017. URL: https://cds.cern.ch/record/2255039.

[202] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[203] G Lanfranchi et al. Tech. rep. LHCb-PUB-2009-013. CERN-LHCb-PUB-2009-013. Geneva: CERN, Aug. 2009.

[204] Roel Aaij et al. *Optimization of the muon reconstruction algorithms for LHCb Run 2*. Tech. rep. LHCb-PUB-2017-007. CERN-LHCb-PUB-2017-007. Geneva: CERN, Feb. 2017. URL: https://cds.cern.ch/record/2253050.

[205] Vladimir V Gligorov, Christopher Thomas, and Michael Williams. *The HLT inclusive B triggers*. Tech. rep. LHCb-PUB-2011-016. CERN-LHCb-PUB-2011-016. LHCb-INT-2011-030. LHCb-INT-2011-030. Geneva: CERN, Sept. 2011. URL: https://cds.cern.ch/record/1384380.

[206] Adam Davis et al. *PatLongLivedTracking: a tracking algorithm for the reconstruction of the daughters of long-lived particles in LHCb*. Tech. rep. LHCb-PUB-2017-001. CERN-LHCb-PUB-2017-001. Geneva: CERN, Jan. 2017. URL: https://cds.cern.ch/record/2240723.

[207] Andreas Hoecker et al. "TMVA-Toolkit for multivariate data analysis". In: *arXiv preprint physics/0703039* (2007).

[208] V Breton, N Brun, and P Perret. *A clustering algorithm for the LHCb electromagnetic calorimeter using a cellular automaton*. Tech. rep. LHCb-2001-123. Geneva: CERN, Sept. 2001. URL: https://cds.cern.ch/record/681262.

[209] D Derkach et al. "LHCb trigger streams optimization". In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 062026. DOI: 10.1088/1742-6596/898/6/062026. URL: https://doi.org/10.1088%2F1742-6596%2F898%2F6%2F062026.

[210] T Head. "The LHCb trigger system". In: *Journal of Instrumentation* 9.09 (Sept. 2014), pp. C09015–C09015. DOI: 10.1088/1748-0221/9/09/c09015. URL: https://doi.org/10.1088%2F1748-0221%2F9%2F09%2Fc09015.

[211] Alessio Piucci. "The LHCb Upgrade". In: *Journal of Physics: Conference Series* 878 (July 2017), p. 012012. DOI: 10.1088/1742-6596/878/1/012012. URL: https://doi.org/10.1088%2F1742-6596%2F878%2F1%2F012012.

[212] LHCb Collaboration. *LHCb Tracker Upgrade Technical Design Report*. Tech. rep. CERN-LHCC-2014-001. LHCB-TDR-015. Feb. 2014. URL: https://cds.cern.ch/record/1647400.

[213] R. Aaij et al. "A comprehensive real-time analysis model at the LHCb experiment". In: *Journal of Instrumentation* 14.04 (Apr. 2019), P04006–P04006. DOI: 10.1088/1748-0221/14/04/p04006. URL: https://doi.org/10.1088%2F1748-0221%2F14%2F04%2Fp04006.

[214] R. Aaij et al. *Allen: A high level trigger on GPUs for LHCb*. 2019. arXiv: 1912.09161 [physics.ins-det].

[215] CERN (Meyrin) LHCb Collaboration. *LHCb Upgrade GPU High Level Trigger Technical Design Report*. Tech. rep. CERN-LHCC-2020-006. LHCB-TDR-021. Geneva: CERN, May 2020. URL: https://cds.cern.ch/record/2717938.

[216] R. Aaij et al. "Allen: A High-Level Trigger on GPUs for LHCb". In: *Computing and Software for Big Science* 4.1 (Apr. 2020). ISSN: 2510-2044. DOI: 10.1007/s41781-020-00039-7. URL: http://dx.doi.org/10.1007/s41781-020-00039-7.

[217] Concezio Bozzi. *LHCb Computing Resources: 2020 requests and preview of the subsequent years*. Tech. rep. LHCb-PUB-2019-003. CERN-LHCb-PUB-2019-003. Geneva: CERN, Feb. 2019. URL: https://cds.cern.ch/record/2657832.

[218] Roel Aaij et al. "Selection and processing of calibration samples to measure the particle identification performance of the LHCb experiment in Run 2". In: *EPJ Techniques and Instrumentation* 6.1 (2019), p. 1.

[219] Olli Lupton et al. *Calibration samples for particle identification at LHCb in Run 2*. Tech. rep. LHCb-PUB-2016-005. CERN-LHCb-PUB-2016-005. Geneva: CERN, Apr. 2016. URL: https://cds.cern.ch/record/2134057.

[220] A Augusto Alves Jr et al. "Performance of the LHCb muon system". In: *Journal of Instrumentation* 8.02 (2013), P02022.

[221] Bernard W Silverman. *Density estimation for statistics and data analysis*. Vol. 26. CRC press, 1986.

[222] Lucio Anderlini et al. *New muon identification operators*. Tech. rep. LHCb-INT-2019-020. CERN-LHCb-INT-2019-020. Geneva: CERN, Aug. 2019. URL: https://cds.cern.ch/record/2687369.

[223] HSE. *International Data Analysis Olympiad*. 2016. URL: https://idao.world/.

[224] P. de Simone on behalf of the LNF LHCb group. Tech. rep. URL: https://agenda.infn.it/event/10287/contributions/.

[225] LHCb Collaboration. *LHCb PID Upgrade Technical Design Report*. Tech. rep. CERN-LHCC-2013-022. LHCB-TDR-014. Nov. 2013. URL: https://cds.cern.ch/record/1624074.

[226] C. Patrignani et al. "Review of Particle Physics". In: *Chin. Phys.* C40.10 (2016), p. 100001. DOI: 10.1088/1674-1137/40/10/100001.

[227] G Lanfranchi et al. *The muon identification procedure of the LHCb experiment for the first data*. Tech. rep. 2009.

[228] Sarti et al. Tech. rep. LHCb-PUB-2010-002, CERN-LHCb-PUB-2010-002. 2010.

[229] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[230] Karen Davies and Robert Steiner. *A list of Acts of Parliament and awards, held by Royal Greenwich Observatory Archives*. Royal Greenwich Observatory Archives, 2013. URL: http://cudl.lib.cam.ac.uk/view/MS-RGO-00014-00001/19.

[231] Xavier Amatriain and Justin Basilico. *Netflix Recommendations: Beyond the 5 stars (Part 1)*. 2012. URL: https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429 (visited on 02/03/2020).

[232] Arindam Bhattacharya. *Lessons Learned from Data Science Competitions*. June 19, 2019. URL: https://medium.com/opex-analytics/lessons-learned-from-data-science-competitions-9f9d17bd6ec0.

[233] Jeff Faudi. *Important Things you should know before organizing a Kaggle Competition*. URL: https://blog.usejournal.com/important-things-you-should-know-before-organizing-a-kaggle-competition-3911b71701fb?gi=b0cce3372488.

[234] Tianqi Chen. *Story and lessons behind the evolution of xgboost*. URL: https://homes.cs.washington.edu/~tqchen/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html (visited on 02/03/2020).

[235] C Adam-Bourdarios et al. "The Higgs Machine Learning Challenge". In: *Journal of Physics: Conference Series* 664.7 (Dec. 2015), p. 072015. DOI: 10.1088/1742-6596/664/7/072015. URL: https://doi.org/10.1088%2F1742-6596%2F664%2F7%2F072015.

[236] Konrad Budek and Patryk Miziuła. *Four ways to use a Kaggle competition to test artificial intelligence in business*. Aug. 24, 2018. URL: https://deepsense.ai/four-ways-to-use-a-kaggle-competition-to-test-artificial-intelligence-in-business/.

[237] Inc. Kaggle. *How to Use Kaggle*. URL: https://www.kaggle.com/docs/competitions (visited on 05/04/2020).

[238] Ryan Singel. *Netflix Spilled Your Brokeback Mountain Secret, Lawsuit Claims*. 2009. URL: https://www.wired.com/2009/12/netflix-privacy-lawsuit/.

[239] CMS Collaboration et al. "2018 CMS data preservation, re-use and open access policy". In: *CERN Open Data Portal* (2018).

[240] *ATLAS Data Access Policy*. Tech. rep. ATL-CB-PUB-2015-001. Geneva: CERN, Mar. 2015. URL: https://cds.cern.ch/record/2002139.

[241] Peter Clarke. *LHCb External Data Access Policy*. Tech. rep. LHCb-PUB-2013-003. CERN-LHCb-PUB-2013-003. Geneva: CERN, Feb. 2020. URL: https://cds.cern.ch/record/1543410.

[242] ALICE collaboration et al. "ALICE data preservation strategy". In: *CERN Open Data Portal* (2013). URL: http://doi.org/10.7483/OPENDATA.ALICE.54NE.X2EA.

[243] 2019. URL: https://github.com/stsopov/IDAO_2019_MuID_first_track/.

[244] 2019. URL: https://github.com/danchern97/IDAO-2019.

[245] O Shapovalov. "Большой адронный коллайдер и Одноклассники". In: *Хабр, Блог компании Singularis* (2019). URL: https://habr.com/ru/company/singularis/blog/449430/.

[246] *Official productions of LHCb real data*. URL: https://twiki.cern.ch/twiki/bin/view/Main/ProcessingPasses.

[247] A. J. Bevan et al. "The Physics of the B Factories". In: *The European Physical Journal C* 74.11 (Nov. 2014), p. 3026. ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-014-3026-9. URL: https://doi.org/10.1140/epjc/s10052-014-3026-9.

[248] *GitHub issue: Ensemble models (and maybe others?) don't check for negative sample_weight, last accesed: 2019-06-01*. https://github.com/scikit-learn/scikit-learn/issues/3774. (Visited on 06/01/2019).

[249] Thomas Keck. "Machine learning algorithms for the Belle II experiment and their validation on Belle data". PhD thesis. KIT, Karlsruhe, 2017. DOI: 10.1007/978-3-319-98249-6. URL: https://publikationen.bibliothek.kit.edu/1000078149.

[250] Benjamin Lipp. "sPlot-based training of multivariate classifiers in the Belle II analysis software framework". In: *Bachelor thesis, KIT* (2015).

[251] D Martschei et al. "Advanced event reweighting using multivariate analysis". In: *Journal of Physics: Conference Series*. Vol. 368. 1. IOP Publishing. 2012, p. 012028.

[252] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for exotic particles in high-energy physics with deep learning". In: *Nature communications* 5 (2014), p. 4308.

[253] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[254] Mikhail Hushchyn. "Machine Learning based Global Particle Identification Algorithms at the LHCb Experiment". July 2018. URL: https://cds.cern.ch/record/2631902.

[255] Denis Derkach et al. "Machine-Learning-based global particle-identification algorithms at the LHCb experiment". In: *Journal of Physics: Conference Series* 1085 (Sept. 2018), p. 042038. DOI: 10.1088/1742-6596/1085/4/042038. URL: https://doi.org/10.1088%2F1742-6596%2F1085%2F4%2F042038.

[256] *Rec project v30r4*. 2019. URL: https://gitlab.cern.ch/lhcb/Rec/tree/v30r4/Rec/ChargedProtoANNPID/src.

[257] Pablo de Castro and Tommaso Dorigo. "INFERNO: Inference-Aware Neural Optimisation". In: *Computer Physics Communications* 244 (2019), pp. 170–179. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2019.06.007. URL: http://www.sciencedirect.com/science/article/pii/S0010465519301948.

[258] R. Aaij et al. "Measurement of the $B_s^0 \to \mu^+\mu^-$ Branching Fraction and Search for $B^0 \to \mu^+\mu^-$ Decays at the LHCb Experiment". In: *Phys. Rev. Lett.* 111 (10 Sept. 2013), p. 101805. DOI: 10.1103/PhysRevLett.111.101805. URL: https://link.aps.org/doi/10.1103/PhysRevLett.111.101805.

[259] C. Jones. *ANN PID Retuning*. URL: https://indico.cern.ch/event/508832/.

[260] E Polycarpo and J R T De Mello-Neto. *Muon identification in LHCb*. Tech. rep. LHCb-2001-009. revised version number 1 submitted on 2001-08-03 10:41:14. Geneva: CERN, Mar. 2001. URL: https://cds.cern.ch/record/691581.

[261] A C S Assis-Jesus et al. *Multivariate Methods for Muon Identification at LHCb*. Tech. rep. LHCb-2001-084. revised version number 1 submitted on 2001-07-27 11:31:43. Geneva: CERN, July 2001. URL: https://cds.cern.ch/record/684673.

[262] M Gandelman and E Polycarpo. *The Performance of the LHCb Muon Identification Procedure*. Tech. rep. LHCb-2007-145. CERN-LHCb-2007-145. Geneva: CERN, Mar. 2008. URL: https://cds.cern.ch/record/1093941.

[263] Гущин Михаил Иванович . "Применение методов машинного обучения в задачах обработки и хранения данных в экспериментах физики высоких энергий ". PhD thesis. Московский физико-технический институт (государственный университет) , 2019.

[264] *Parameter tuning*. URL: https://catboost.ai/docs/concepts/parameter-tuning.html.

[265] Elena Truskova. "Познаём Нирвану – универсальную вычислительную платформу Яндекса". In: *Хабр, Блог компании Яндекс* (2018). URL: https://habr.com/ru/company/yandex/blog/351016/.

[266] Mikhail Hushchyn and Denis Derkach. *Plots for yPID*. URL: https://indico.cern.ch/event/668115/.

[267] Mikhail Hushchyn and Denis Derkach. *YandexPID: MC/Data Studies*. 2018. URL: https://indico.cern.ch/event/704687/.

[268] S Tolk et al. *Data driven trigger efficiency determination at LHCb*. Tech. rep. LHCb-PUB-2014-039. CERN-LHCb-PUB-2014-039. Geneva: CERN, May 2014. URL: http://cds.cern.ch/record/1701134.

[269] Norraphat Srimanobhas. *Introduction to Monte Carlo for Particle Physics Study*. 2010. URL: https://indico.cern.ch/event/92209/contributions/2114409/.

[270] I Belyaev et al. “Handling of the generation of primary events in Gauss, the LHCb simulation framework”. In: *Journal of Physics: Conference Series*. Vol. 331. 3. IOP Publishing. 2011, p. 032047.

[271] John Allison et al. “Geant4 developments and applications”. In: *IEEE Transactions on nuclear science* 53.1 (2006), pp. 270–278.

[272] D Müller et al. “ReDecay: A novel approach to speed up the simulation at LHCb”. In: *The European Physical Journal C* 78.12 (2018), p. 1009.

[273] C Bozzi and S Roiser. “The LHCb software and computing upgrade for Run 3: opportunities and challenges”. In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 112002. DOI: 10.1088/1742-6596/898/11/112002.

[274] Dominik Müller. “Adopting new technologies in the LHCb Gauss simulation framework”. In: *EPJ Web Conf.* 214 (2019), 02004. 6 p. DOI: 10.1051/epjconf/201921402004. URL: https://cds.cern.ch/record/2701777.

[275] The HEP Software Foundation et al. “A Roadmap for HEP Software and Computing R&D for the 2020s”. In: *Computing and Software for Big Science* 3.1 (Mar. 2019), p. 7. ISSN: 2510-2044. DOI: 10.1007/s41781-018-0018-8. URL: https://doi.org/10.1007/s41781-018-0018-8.

[276] R. Aaij et al. “Measurement of the Ratio of the $B^0 \to D^{*-}\tau^+\nu_\tau$ and $B^0 \to D^{*-}\mu^+\nu_\mu$ Branching Fractions Using Three-Prong τ-Lepton Decays”. In: *Phys. Rev. Lett.* 120 (17 Apr. 2018), p. 171802. DOI: 10.1103/PhysRevLett.120.171802. URL: https://link.aps.org/doi/10.1103/PhysRevLett.120.171802.

[277] Adam Davis. *Fast Simulations at LHCb*. Nov. 2019. URL: https://indico.cern.ch/event/773049/contributions/3474742/.

[278] V. Khachatryan et al. “Searches for electroweak neutralino and chargino production in channels with Higgs, Z, and W bosons in pp collisions at 8 TeV”. In: *Phys. Rev. D* 90 (9 Nov. 2014), p. 092007. DOI: 10.1103/PhysRevD.90.092007. URL: https://link.aps.org/doi/10.1103/PhysRevD.90.092007.

[279] The CMS Collaboration et al. “Search for top-squark pair production in the single-lepton final state in pp collisions at $\sqrt{s} = 12$TeV”. In: *The European Physical Journal C* 73.12 (Dec. 2013), p. 2677. ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-013-2677-2. URL: https://doi.org/10.1140/epjc/s10052-013-2677-2.

[280] Salavat Abdullin et al. “The fast simulation of the CMS detector at LHC”. In: *Journal of Physics: Conference Series*. Vol. 331. 3. IOP Publishing. 2011, p. 032049.

[281] Sezen Sekmen. *Fast simulation in CMS*. July 2017. URL: https://indico.cern.ch/event/614935/contributions/2625507/.

[282] Sezen Sekmen et al. “Recent developments in CMS fast simulation”. In: *arXiv preprint arXiv:1701.03850* (2017).

[283] Andrea Giammanco. “The Fast Simulation of the CMS Experiment”. In: *Journal of Physics: Conference Series* 513.2 (June 2014), p. 022012. DOI: 10.1088/1742-6596/513/2/022012. URL: https://doi.org/10.1088%2F1742-6596%2F513%2F2%2F022012.

[284] Johan Alwall, Philip C Schuster, and Natalia Toro. “Simplified models for a first characterization of new physics at the LHC”. In: *Physical Review D* 79.7 (2009), p. 075020.

[285] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. “CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks”. In: *Physical Review D* 97.1 (2018), p. 014021.

[286] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. “Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters”. In: *Phys. Rev. Lett.* 120 (4 Jan. 2018), p. 042003. DOI: 10.1103/PhysRevLett.120.042003. URL: https://link.aps.org/doi/10.1103/PhysRevLett.120.042003.

[287] Georges Aad et al. “The ATLAS simulation infrastructure”. In: *The European Physical Journal C* 70.3 (2010), pp. 823–874.

[288] D. Salamani et al. “Deep Generative Models for Fast Shower Simulation in ATLAS”. In: *2018 IEEE 14th International Conference on e-Science (e-Science)*. Oct. 2018, pp. 348–348. DOI: 10.1109/eScience.2018.00091.

[289] Martin Erdmann, Jonas Glombitza, and Thorben Quast. “Precise simulation of electromagnetic calorimeter showers using a Wasserstein Generative Adversarial Network”. In: *Computing and Software for Big Science* 3.1 (2019), p. 4.

[290] Chekalina, Viktoria et al. “Generative Models for Fast Calorimeter Simulation: the LHCb case>”. In: *EPJ Web Conf.* 214 (2019), p. 02034. DOI: 10.1051/epjconf/201921402034. URL: https://doi.org/10.1051/epjconf/201921402034.

[291] The DELPHES 3 collaboration et al. “DELPHES 3: a modular framework for fast simulation of a generic collider experiment”. In: *Journal of High Energy Physics* 2014.2 (Feb. 2014), p. 57. ISSN: 1029-8479. DOI: 10.1007/JHEP02(2014)057. URL: https://doi.org/10.1007/JHEP02(2014)057.

[292] I Adam et al. “The DIRC particle identification system for the BaBar experiment”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 538.1-3 (2005), pp. 281–357.

[293] John Hardin and Mike Williams. “FastDIRC: a fast Monte Carlo and reconstruction algorithm for DIRC detectors”. In: *JINST* 11.10 (2016), P10007. DOI: 10.1088/1748-0221/11/10/P10007. arXiv: 1608.01180 [physics.data-an].

[294] *The BaBar Detector*. URL: https://www.slac.stanford.edu/BFROOT/www/doc/workbook/detector/detector.html (visited on 12/27/2019).

[295] I. Adam et al. “The DIRC detector at BaBar”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 433.1 (1999), pp. 121–127. ISSN: 0168-9002. DOI: https://doi.org/10.1016/S0168-9002(99)00352-6. URL: http://www.sciencedirect.com/science/article/pii/S0168900299003526.

[296] Denis Derkach et al. “Cherenkov detectors fast simulation using neural networks”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 952 (2020), p. 161804. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2019.01.031.

[297] B. Siddi L. Anderlini A. Davis. *Update on Lamarr.* Oct. 1, 2019. URL: https://indico.cern.ch/event/850731/contributions/3584359/ (visited on 12/27/2019).

[298] Adam Davis and Benedetto Gianluca Siddi. *Development and Deployment of a Delphes Based Simulation in Gauss.* Tech. rep. LHCb-INT-2019-013. CERN-LHCb-INT-2019-013. Geneva: CERN, May 2019. URL: https://cds.cern.ch/record/2674629.

[299] Denis Derkach et al. “Data Driven Simulation of Cherenkov Detectors using Generative Adversarial Network”. In: *Machine Learning and the Physical Sciences Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS)* (Dec. 2019). URL: https://ml4physicalsciences.github.io/files/NeurIPS_ML4PS_2019_40.pdf.

[300] A Maevskiy et al. “Fast Data-Driven Simulation of Cherenkov Detectors Using Generative Adversarial Networks”. In: *Journal of Physics: Conference Series* 1525 (Apr. 2020), p. 012097. DOI: 10.1088/1742-6596/1525/1/012097. URL: https://doi.org/10.1088%2F1742-6596%2F1525%2F1%2F012097.

[301] Patrice Bertail, Stéphan J Clémençcon, and Nicolas Vayatis. “On bootstrapping the ROC curve”. In: *Advances in Neural Information Processing Systems.* 2009, pp. 137–144.

[302] “Performance of the Lamarr Prototype: the ultra-fast simulation option integrated in the LHCb simulation framework”. In: (Oct. 2019). URL: https://cds.cern.ch/record/2696310.

[303] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. “Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis”. In: *Computing and Software for Big Science* 1.1 (Sept. 2017), p. 4. ISSN: 2510-2044. DOI: 10.1007/s41781-017-0004-6. URL: https://doi.org/10.1007/s41781-017-0004-6.

[304] S. Vallecorsa. “Generative models for fast simulation”. In: *Journal of Physics: Conference Series* 1085 (Sept. 2018), p. 022005. DOI: 10.1088/1742-6596/1085/2/022005. URL: https://doi.org/10.1088%2F1742-6596%2F1085%2F2%2F022005.

[305] Benjamin Bentner. "Study of the performance of a kinematic constraint reconstruction of the missing momentum in partially reconstructed events in". Bachelor's Thesis. Department of Physics and Astronomy Heidelberg University, 2019. URL: https://www.physi.uni-heidelberg.de/Publications/ThesisBentner.pdf.

[306] M. Borisyak and N. Kazeev. "Machine Learning on data with sPlot background subtraction". In: *Journal of Instrumentation* 14.08 (Aug. 2019), P08020–P08020. DOI: 10.1088/1748-0221/14/08/p08020. URL: https://doi.org/10.1088%2F1748-0221%2F14%2F08%2Fp08020.

[307] *PIDCalib Packages*. URL: https://twiki.cern.ch/twiki/bin/view/LHCb/PIDCalibPackage.

[308] Vanya Belyaev et al. *Calorimeter PIDs*. Mar. 2003. URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwiNm-vvme_kAhWCo4sKHcAEAFUQFjAAegQIABAB&url=https%3A%2F%2Fwww.slideserve.com%2Fbeata%2Fcalorimeter-pids-vanya-belyaev-ioury-gouz-vladimir-romanovsky-grygory-rybkin&usg=AOvVaw0_yJmuLOIthAedfATB382j.

[309] Johannes Albrecht et al. *Upgrade trigger &; reconstruction strategy: 2017 milestone*. Tech. rep. LHCb-PUB-2018-005. CERN-LHCb-PUB-2018-005. Geneva: CERN, Mar. 2018. URL: http://cds.cern.ch/record/2310579.